Chapter 5

Motivation for wavelets and some simple examples

In the first part of the book our focus was to approximate functions or vectors with trigonometric functions. We saw that the Discrete Fourier transform could be used to obtain a 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 most sound data, where the 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 sound may not be simple. 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 for representing functions. This alternative is called 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 some very simple wavelets. The first wavelet we look at can be interpreted as an approximation scheme based on piecewise constant functions. The next wavelet we look at is similar, but with piecewise linear functions used instead. Following these examples we will establish a more general framework, based on experiences from the simple wavelets. In the following chapters we will interpret this framework in terms of filters, and use this connection to construct even more interesting wavelets.

Core functions in this chapter are collected in a module called `dwt`.

5.1 Why wavelets?

The left image in Figure 5.1 shows a view of the entire Earth.

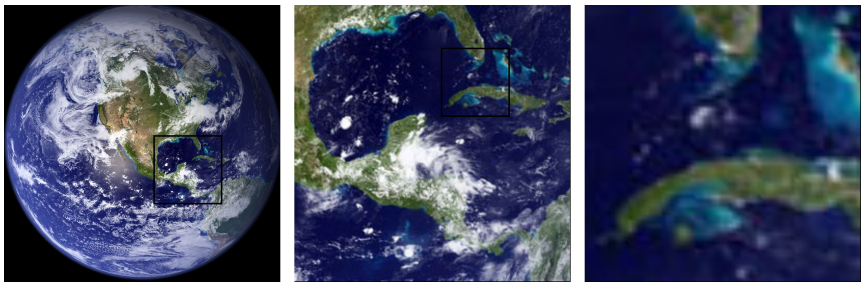

Figure 5.1: A view of Earth from space, together with versions of the image where we have zoomed in.

The startup image in Google Earth^{TM}, a program for viewing satellite images, maps and other geographic information, is very similar to this. In the middle image we have zoomed in on the Mexican Gulff, as marked with a rectangle in the left image. Similarly, in the right image we have further zoomed in on Cuba and a small portion of Florida, as marked with a rectangle in the middle image. There is clearly an amazing amount of information available behind a program like Google Earth^{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^{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 color 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 color 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. *Images model.*

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 approximating functions, just as Taylor series and Fourier series. Our new approximation 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$^{\text{TM}}$, 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 how wavelets are defined more formally, and construct the simplest wavelet we have. Its construction goes in the following steps: First we introduce what we call resolution spaces, and the corresponding scaling function. Then we introduce the detail spaces, and the corresponding mother wavelet. These two functions will give rise to certain bases for these spaces, and we will define the Discrete Wavelet Transform as a change of coordinates between these bases.

5.2 A wavelet based on piecewise constant functions

Our starting point will be the space of piecewise constant functions on an interval $[0, N)$. This will be called a resolution space.

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. An example of a function in V_0 for $N = 10$ is shown in Figure 5.2. It is easy to check that V_0 is a linear space, and for computations it is useful to know the dimension of the space and have a basis.

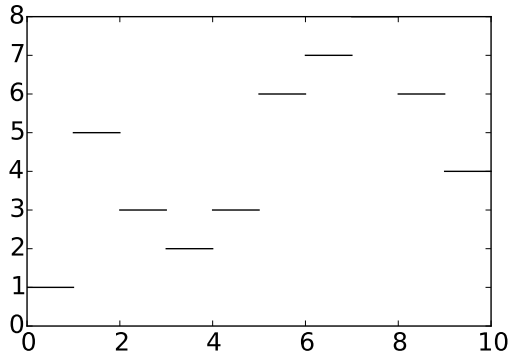

Figure 5.2: A piecewise constant function.

Lemma 5.3. *The function ϕ.*
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 5.3 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 $\|\phi_n\| = 1$ 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. □

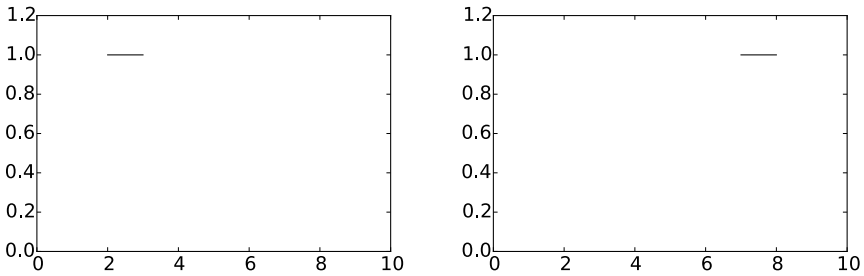

Figure 5.3: The basis functions ϕ_2 and ϕ_7 from $\boldsymbol{\phi}_0$.

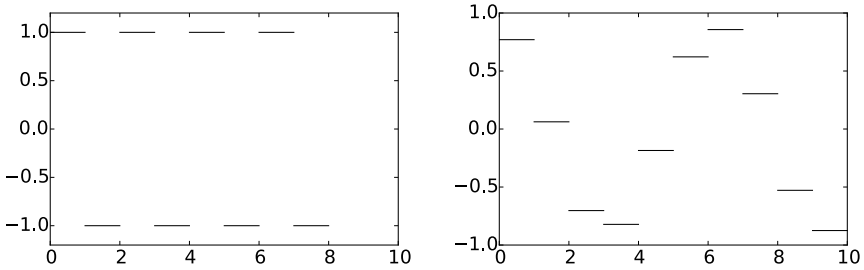

Figure 5.4: Examples of functions from V_0. The square wave in V_0 (left), and an approximation to $\cos t$ from V_0 (right).

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 the left plot in Figure 5.4. The difference is that we have more flexibility since we have a whole space at our disposal instead of just one function — the right plot in Figure 5.4 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 constant functions defined on $[0, N)$ that are constant on each subinterval $[n/2^m, (n+1)/2^m)$ for $n = 0, 1, \ldots, 2^m N - 1$.

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 the rightmost is a piecewise constant function that oscillates like $\sin(2\pi 2^2 t)$ on the interval $[0, 10]$.

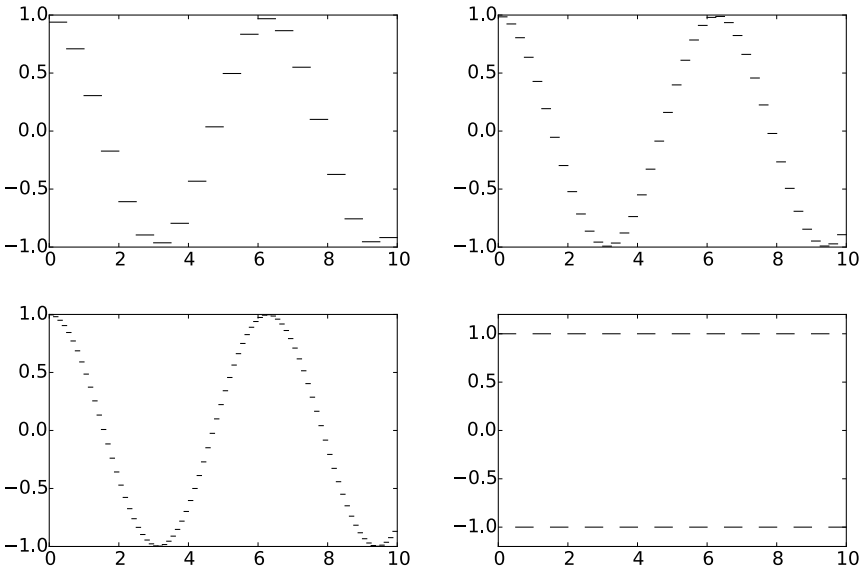

Figure 5.5: Piecewise constant approximations to $\cos t$ on the interval $[0, 10]$ in the spaces V_1, V_2, and V_3. The lower right plot shows the square wave in V_2.

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

Lemma 5.5. *Basis for V_m.*

Let $[0, N)$ be a given interval with N some positive integer. Then the dimension of V_m is $2^m N$. The functions

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

form an orthonormal basis for V_m, which we will denote by $\boldsymbol{\phi}_m$. Any function $f \in V_m$ can thus be represented uniquely as

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

Proof. The functions given by Equation (5.4) are nonzero on the subintervals $[n/2^m, (n+1)/2^m)$ which we referred to in Definition 5.4, so that $\phi_{m,n_1}\phi_{m,n_2} = 0$ when $n_1 \neq n_2$, since these intervals are disjoint. The only 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 thus ensures that $\|\phi_{m,n}\| = 1$ for all m. □

In the following we will always denote the coordinates in the basis $\boldsymbol{\phi}_m$ by $c_{m,n}$. Note that our definition restricts the dimensions of the spaces we study to be on the form $N2^m$. In Chapter 6 we will explain how this restriction can be dropped, but until then the dimensions will be assumed to be on this form. 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. *Resolution spaces and approximation.*

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

$$\big|f(t) - g(t)\big| \leq \epsilon$$

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

Proof. Since f is (uniformly) continuous on $[0, N]$, we can find an integer m so that $\big|f(t_1) - f(t_2)\big| \leq \epsilon$ 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

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

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. *Resolution spaces and approximation.*
Let f be a given continuous function on the interval $[0, N]$. Then

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

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.

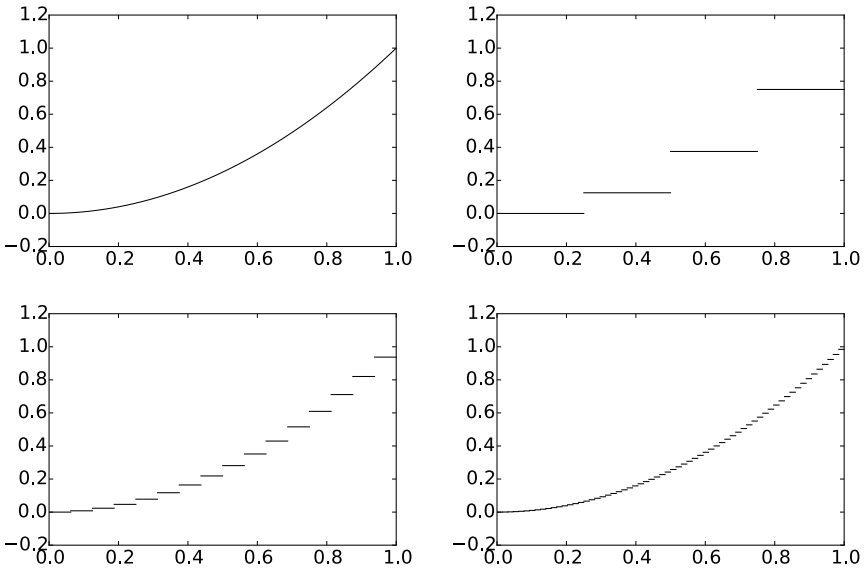

Figure 5.6: Comparison of the function defined by $f(t) = t^2$ on $[0, 1]$ with the projection onto V_2, V_4, and V_6, respectively.

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. We start by observing that since

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

we have

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

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

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

and the right-hand side clearly lies in V_1. Since also

$$\begin{aligned}\phi_{m-1,n}(t) &= 2^{(m-1)/2}\phi(2^{m-1}t - n) = 2^{(m-1)/2}\phi_{0,n}(2^{m-1}t)\\ &= 2^{(m-1)/2}\frac{1}{\sqrt{2}}(\phi_{1,2n}(2^{m-1}t) + \phi_{1,2n+1}(2^{m-1}t))\\ &= 2^{(m-1)/2}(\phi(2^m t - 2n) + \phi(2^m t - (2n+1))) = \frac{1}{\sqrt{2}}(\phi_{m,2n}(t) + \phi_{m,2n+1}(t)),\end{aligned}$$

we also have that

$$\phi_{m-1,n} = \frac{1}{\sqrt{2}}\phi_{m,2n} + \frac{1}{\sqrt{2}}\phi_{m,2n+1}, \tag{5.5}$$

so that also $V_k \subset V_{k+1}$ for any integer $k \geq 0$.

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

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

This means that it is meaningful to project V_{k+1} onto V_k. The next step is to characterize the projection from V_1 onto V_0, and onto the orthogonal complement of V_0 in V_1. Before we do this, let us make the following definitions.

Definition 5.9. *Detail spaces.*
The orthogonal complement of V_{m-1} in V_m is denoted W_{m-1}. All the spaces W_k are also called detail spaces, or error spaces.

The name detail space is used since the projection from V_m onto V_{m-1} in considered as a (low-resolution) approximation, and the error, which lies in W_{m-1}, is the detail which is left out when we replace with this approximation. We will also write $g_m = g_{m-1} + e_{m-1}$ when we split $g_m \in V_m$ into a sum of a low-resolution approximation and a detail component. In the context of our Google Earth$^{\text{TM}}$example, in Figure 5.1 you should interpret g_0 as the left image, the middle image as an excerpt of g_1, and e_0 as the additional details which are needed to reproduce the middle image from the left image.

Since V_0 and W_0 are mutually orthogonal spaces they are also linearly independent spaces. When U and V are two such linearly independent spaces, we will write $U \oplus V$ for the vector space consisting of all vectors of the form $\boldsymbol{u} + \boldsymbol{v}$, with $\boldsymbol{u} \in U$, $\boldsymbol{v} \in V$. $U \oplus V$ is also called the *direct sum* of U and V. This also makes sense if we have more than two vector spaces (such as $U \oplus V \oplus W$),

and the direct sum clearly obeys the associate law $U \oplus (V \oplus W) = (U \oplus V) \oplus W$. Using the direct sum notation, we can first write

$$V_m = V_{m-1} \oplus W_{m-1}. \tag{5.6}$$

Since V_m has dimension $2^m N$, it follows that also W_m has dimension $2^m N$. We can continue the direct sum decomposition by also writing V_{m-1} as a direct sum, then V_{m-2} as a direct sum, and so on, and end up with

$$V_m = V_0 \oplus W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}, \tag{5.7}$$

where the spaces on the right hand side have dimension $N, N, 2N, \ldots, 2^{m-1}N$. This decomposition wil be important for our purposes. It says that the resolution space V_m acan be written as the sum of a lower order resolution space V_0, and m detail spaces $W_0, \ldots, W_{m-1}$. We will later interpret this splitting into a low-resolution component and m detail components.

It turns out that the following function will play the same role for the detail space W_k as the function ϕ plays for the resolution space V_k.

Definition 5.10. *The function ψ.*

We define

$$\psi(t) = \big(\phi_{1,0}(t) - \phi_{1,1}(t)\big)/\sqrt{2} = \phi(2t) - \phi(2t-1), \tag{5.8}$$

and

$$\psi_{m,n}(t) = 2^{m/2}\psi(2^m t - n), \quad \text{for } n = 0,\, 1,\, \ldots,\, 2^m N - 1. \tag{5.9}$$

The functions ϕ and ψ are shown in Figure 5.7.

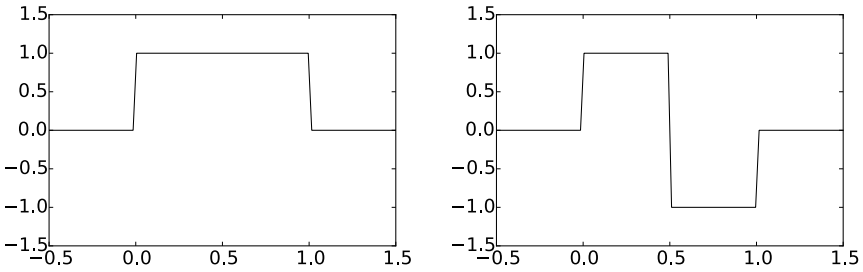

Figure 5.7: The functions ϕ and ψ we used to analyse the space of piecewise constant functions.

As in the proof for Equation (5.5), it follows that

$$\psi_{m-1,n} = \frac{1}{\sqrt{2}}\phi_{m,2n} - \frac{1}{\sqrt{2}}\phi_{m,2n+1}, \tag{5.10}$$

Clearly ψ is supported on $[0, 1)$, and $\|\psi\| = 1$. From this it follows as for $\boldsymbol{\phi}_0$ that the $\{\psi_{0,n}\}_{n=0}^{N-1}$ are orthonormal. In the same way as for $\boldsymbol{\phi}_m$, it follows

also that the $\{\psi_{m,n}\}_{n=0}^{2^m N-1}$ is orthonormal for any m. We will write $\boldsymbol{\psi}_m$ for the orthonormal basis $\{\psi_{m,n}\}_{n=0}^{2^m N-1}$, and we will always denote the coordinates in the basis $\boldsymbol{\psi}_m$ by $w_{m,n}$. The next result motivates the definition of ψ, and states how we can project from V_1 onto V_0 and W_0, i.e. find the low-resolution approximation and the detail component of $g_1 \in V_1$.

Lemma 5.11. *Orthonormal bases.*
For $0 \le n < N$ we have that

$$\mathrm{proj}_{V_0}(\phi_{1,n}) = \begin{cases} \phi_{0,n/2}/\sqrt{2}, & \text{if } n \text{ is even;} \\ \phi_{0,(n-1)/2}/\sqrt{2}, & \text{if } n \text{ is odd.} \end{cases} \tag{5.11}$$

$$\mathrm{proj}_{W_0}(\phi_{1,n}) = \begin{cases} \psi_{0,n/2}/\sqrt{2}, & \text{if } n \text{ is even;} \\ -\psi_{0,(n-1)/2}/\sqrt{2}, & \text{if } n \text{ is odd.} \end{cases} \tag{5.12}$$

In particular, $\boldsymbol{\psi}_0$ is an orthonormal basis for W_0. More generally, if $g_1 = \sum_{n=0}^{2N-1} c_{1,n}\phi_{1,n} \in V_1$, then

$$\mathrm{proj}_{V_0}(g_1) = \sum_{n=0}^{N-1} c_{0,n}\phi_{0,n}, \text{ where } c_{0,n} = \frac{c_{1,2n} + c_{1,2n+1}}{\sqrt{2}} \tag{5.13}$$

$$\mathrm{proj}_{W_0}(g_1) = \sum_{n=0}^{N-1} w_{0,n}\psi_{0,n}, \text{ where } w_{0,n} = \frac{c_{1,2n} - c_{1,2n+1}}{\sqrt{2}}. \tag{5.14}$$

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

$$\left[\frac{n}{2}, \frac{n+1}{2}\right) \cap [n_1, n_1 + 1) \tag{5.15}$$

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

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

Using this we also get

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

This proves the expressions for both projections when n is even. When n is odd, the intersection (5.15) is nonempty only if $n_1 = (n-1)/2$, which gives the expressions for both projections when n is odd in the same way. In particular we get

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

$\boldsymbol{\psi}_0$ must be an orthonormal basis for W_0 since $\boldsymbol{\psi}_0$ is contained in W_0, and both have dimension N.

We project the function g_1 in V_1 using the formulas in (5.11). We first split the sum into even and odd values of n,

$$g_1 = \sum_{n=0}^{2N-1} c_{1,n}\phi_{1,n} = \sum_{n=0}^{N-1} c_{1,2n}\phi_{1,2n} + \sum_{n=0}^{N-1} c_{1,2n+1}\phi_{1,2n+1}. \tag{5.16}$$

We can now apply the two formulas in (5.11),

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

which proves Equation (5.13). Equation (5.14) is proved similarly. □

In Figure 5.8 we have used Lemma 5.11 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 Lemma 5.11:

Proposition 5.12. *Projections.*

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 $\mathrm{proj}_{V_0}(f)$ is the function in V_0 which equals $(f_{n,1} + f_{n,2})/2$ on the interval $[n, n+1)$. Moreover, $\mathrm{proj}_{W_0}(f)$ 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 $[n+1/2, n+1)$.

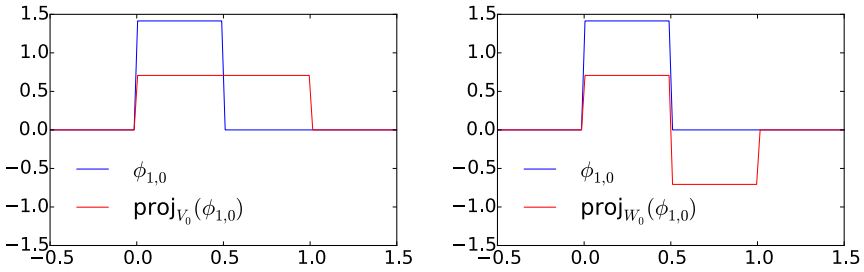

Figure 5.8: The projection of $\phi_{1,0} \in V_1$ onto V_0 and W_0.

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

In the same way as in Lemma 5.11, it is possible to show that

$$\text{proj}_{W_{m-1}}(\phi_{m,n}) = \begin{cases} \psi_{m-1,n/2}/\sqrt{2}, & \text{if } n \text{ is even;} \\ -\psi_{m-1,(n-1)/2}/\sqrt{2}, & \text{if } n \text{ is odd.} \end{cases} \tag{5.17}$$

From this it follows as before that $\boldsymbol{\psi}_m$ is an orthonormal basis for W_m. If $\{\mathcal{B}_i\}_{i=1}^n$ are mutually independent bases, we will in the following write $(\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n)$ for the basis where the basis vectors from $\mathcal{B}_i$ are included before $\mathcal{B}_j$ when $i < j$. With this notation, the decomposition in Equation (5.7) can be restated as follows

Theorem 5.13. *Bases for V_m.*
$\boldsymbol{\phi}_m$ and $(\boldsymbol{\phi}_0, \boldsymbol{\psi}_0, \boldsymbol{\psi}_1, \cdots, \boldsymbol{\psi}_{m-1})$ are both bases for V_m.

The function ψ thus has the property that its dilations and translations together span the detail components. Later we will encounter other functions, which also will be denoted by ψ, and have similar properties. In the theory of wavelets, such ψ are called *mother wavelets*. There is one important property of ψ, which we will return to:

Observation 5.14. *Vanishing moment.*
We have that $\int_0^N \psi(t)dt = 0$.

This can be seen directly from the plot in Figure 5.7, since the parts of the graph above and below the x-axis cancel. In general we say that ψ has k vanishing moments if the integrals $\int t^l \psi(t)dt = 0$ for all $0 \leq l \leq k-1$. Due to Observation 5.14, ψ has one vanishing moment. In Chapter 7 we will show that mother wavelets with many vanishing moments are very desirable when it comes to approximation of functions.

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

Definition 5.15. *Discrete Wavelet Transform.*

The DWT (Discrete Wavelet Transform) is defined as the change of coordinates from $\boldsymbol{\phi}_1$ to $(\boldsymbol{\phi}_0, \boldsymbol{\psi}_0)$. More generally, the m-level DWT is defined as the change of coordinates from $\boldsymbol{\phi}_m$ to $(\boldsymbol{\phi}_0, \boldsymbol{\psi}_0, \boldsymbol{\psi}_1, \cdots, \boldsymbol{\psi}_{m-1})$. In an m-level DWT, the change of coordinates from

$$(\boldsymbol{\phi}_{m-k+1}, \boldsymbol{\psi}_{m-k+1}, \boldsymbol{\psi}_{m-k+2}, \cdots, \boldsymbol{\psi}_{m-1}) \text{ to } (\boldsymbol{\phi}_{m-k}, \boldsymbol{\psi}_{m-k}, \boldsymbol{\psi}_{m-k+1}, \cdots, \boldsymbol{\psi}_{m-1}) \tag{5.18}$$

is also called the k'th stage. The (m-level) IDWT (Inverse Discrete Wavelet Transform) is defined as the change of coordinates the opposite way.

The DWT 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. We now can state the following result.

Theorem 5.16. *Expression for the DWT.*

If $g_m = g_{m-1} + e_{m-1}$ with

$$g_m = \sum_{n=0}^{2^m N-1} c_{m,n}\phi_{m,n} \in V_m,$$

$$g_{m-1} = \sum_{n=0}^{2^{m-1}N-1} c_{m-1,n}\phi_{m-1,n} \in V_{m-1} \quad e_{m-1} = \sum_{n=0}^{2^{m-1}N-1} w_{m-1,n}\psi_{m-1,n} \in W_{m-1},$$

then the change of coordinates from $\boldsymbol{\phi}_m$ to $(\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1})$ (i.e. first stage in a DWT) is given by

$$\begin{pmatrix} c_{m-1,n} \\ w_{m-1,n} \end{pmatrix} = \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{pmatrix} \begin{pmatrix} c_{m,2n} \\ c_{m,2n+1} \end{pmatrix} \tag{5.19}$$

Conversely, the change of coordinates from $(\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1})$ to $\boldsymbol{\phi}_m$ (i.e. the last stage in an IDWT) is given by

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

Proof. Equations (5.5) and (5.10) say that

$$\phi_{m-1,n} = \phi_{m,2n}/\sqrt{2} + \phi_{m,2n+1}/\sqrt{2} \quad \psi_{m-1,n} = \phi_{m,2n}/\sqrt{2} - \phi_{m,2n+1}/\sqrt{2}.$$

The change of coordinate matrix from the basis $\{\phi_{m-1,n}, \psi_{m-1,n}\}$ to $\{\phi_{m,2n}, \phi_{m,2n+1}\}$ is thus $\begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{pmatrix}$. This proves Equation (5.20). Equation (5.19) follows immediately since this matrix equals its inverse. □

Above we assumed that N is even. In Exercise 5.8 we will see how we can handle the case when N is odd.

From Theorem 5.16, we see that, if we had defined

$$\mathcal{C}_m = \{\phi_{m-1,0}, \psi_{m-1,0}, \phi_{m-1,1}, \psi_{m-1,1}, \cdots, \phi_{m-1,2^{m-1}N-1}, \psi_{m-1,2^{m-1}N-1}\}. \tag{5.21}$$

i.e. we have reordered the basis vectors in $(\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1})$ (the subscript m is used since $\mathcal{C}_m$ is a basis for V_m), it is apparent from Equation (5.20) that $G = P_{\boldsymbol{\phi}_m \leftarrow \mathcal{C}_m}$ 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 $2^{m-1}N$ times. Also, from Equation (5.19) it is apparent that $H = P_{\mathcal{C}_m \leftarrow \boldsymbol{\phi}_m}$ is the same matrix. Such matrices are called *block diagonal matrices*. This particular block diagonal matrix is clearly orthogonal. Let us make the following definition.

Definition 5.17. *DWT and IDWT kernel transformations.*

The matrices $H = P_{\mathcal{C}_m \leftarrow \boldsymbol{\phi}_m}$ and $G = P_{\boldsymbol{\phi}_m \leftarrow \mathcal{C}_m}$ are called the *DWT and IDWT kernel transformations*. The DWT and the IDWT can be expressed in terms of these kernel transformations by

$$\text{DWT} = P_{(\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1}) \leftarrow \mathcal{C}_m} H \text{ and IDWT} = G P_{\mathcal{C}_m \leftarrow (\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1})},$$

respectively, where

- $P_{(\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1}) \leftarrow \mathcal{C}_m}$ is a permutation matrix which groups the even elements first, then the odd elements,
- $P_{\mathcal{C}_m \leftarrow (\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1})}$ is a permutation matrix which places the first half at the even indices, the last half at the odd indices.

Clearly, the kernel transformations H and G also invert each other. The point of using the kernel transformation is that they compute the output sequentially, similarly to how a filter does. Clearly also the kernel transformations are very similar to a filter, and we will return to this in the next chapter.

At each level in a DWT, V_k is split into one low-resolution component from V_{k-1}, and one detail component from W_{k-1}. We have illustrated this in figure 5.9, where the arrows represent changes of coordinates.

The detail component from W_{k-1} is not subject to further transformation. This is seen in the figure since $\boldsymbol{\psi}_{k-1}$ is a leaf node, i.e. there are no arrows going out from $\boldsymbol{\psi}_{m-1}$. 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,

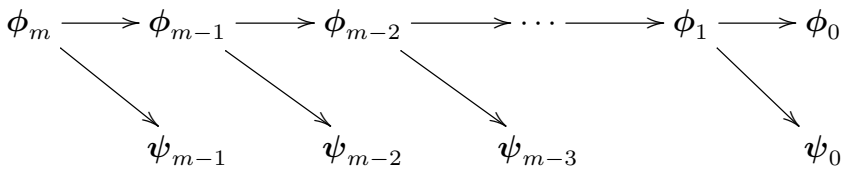

Figure 5.9: Illustration of a wavelet transform.

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.

Exercise 5.1: The vector of samples is the coordinate vector

Show that the coordinate vector for $f \in V_0$ in the basis $\{\phi_{0,0}, \phi_{0,1}, \ldots, \phi_{0,N-1}\}$ is $(f(0), f(1), \ldots . f(N-1))$. This shows that, for $f \in V_m$, there is no loss of information in working with the samples of f rather than f itself.

Exercise 5.2: Proposition 5.12

Prove Proposition 5.12.

Exercise 5.3: Computing projections 1

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 5.11 to write down the matrix for proj_{V_0} relative to the bases $\boldsymbol{\phi}_1$ and $\boldsymbol{\phi}_0$.

b) Similarly, use Lemma 5.11 to write down the matrix for $\text{proj}_{W_0} : V_1 \to W_0$ relative to the bases $\boldsymbol{\phi}_1$ and $\boldsymbol{\psi}_0$.

Exercise 5.4: Computing projections 2

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

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

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.

Exercise 5.5: Computing projections 3

Show that

$$\text{proj}_{V_0}(f) = \sum_{n=0}^{N-1} \left(\int_n^{n+1} f(t)dt \right) \phi_{0,n}(t) \tag{5.22}$$

for any f. Show also that the first part of Proposition 5.12 follows from this.

Exercise 5.6: Finding the least squares error

Show that

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

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). 2DO: Generalize to m

Exercise 5.7: Projecting on W_0

Show that

$$\text{proj}_{W_0}(f) = \sum_{n=0}^{N-1} \left(\int_n^{n+1/2} f(t)dt - \int_{n+1/2}^{n+1} f(t)dt \right) \psi_{0,n}(t) \tag{5.23}$$

for any f. Show also that the second part of Proposition 5.12 follows from this.

Exercise 5.8: When N is odd

When N is odd, the (first stage in a) DWT is defined as the change of coordinates from $(\phi_{1,0}, \phi_{1,1}, \ldots, \phi_{1,N-1})$ to

$$(\phi_{0,0}, \psi_{0,0}, \phi_{0,1}, \psi_{0,1}, \ldots, \phi_{0,(N-1)/2}, \psi_{(N-1)/2}, \phi_{0,(N+1)/2}).$$

Since all functions are assumed to have period N, we have that

$$\phi_{0,(N+1)/2} = \frac{1}{\sqrt{2}}(\phi_{1,N-1} + \phi_{1,N}) = \frac{1}{\sqrt{2}}(\phi_{1,0} + \phi_{1,N-1}).$$

From this relation one can find the last column in the change of coordinate matrix from $\boldsymbol{\phi}_0$ to $(\boldsymbol{\phi}_1, \boldsymbol{\psi}_1)$, i.e. the IDWT matrix. In particular, when N is odd, we see that the last column in the IDWT matrix circulates to the upper right corner. In terms of coordinates, we thus have that

$$c_{1,0} = \frac{1}{\sqrt{2}}(c_{0,0} + w_{0,0} + c_{0,(N+1)/2}) \qquad c_{1,N-1} = \frac{1}{\sqrt{2}} c_{0,(N+1)/2}. \tag{5.24}$$

a) If $N = 3$, the DWT matrix equals

$$\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1 & 1\\ 1 & -1 & 0\\ 0 & 0 & 1\end{pmatrix},$$

and the inverse of this is

$$\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1 & -1\\ 1 & -1 & -1\\ 0 & 0 & 2\end{pmatrix}.$$

Explain from this that, when N is odd, the DWT matrix can be constructed by adding a column on the form $\frac{1}{\sqrt{2}}(-1,-1,0,\ldots,0,2)$ to the DWT matrices we had for N even (in the last row zeros are also added). In terms of the coordinates, we thus have the additional formulas

$$\begin{aligned} c_{0,0} &= \frac{1}{\sqrt{2}}(c_{1,0}+c_{1,1}-c_{1,N-1})\\ w_{0,0} &= \frac{1}{\sqrt{2}}(c_{1,0}-c_{1,1}-c_{1,N-1})\\ c_{0,(N+1)/2} &= \frac{1}{\sqrt{2}}2c_{1,N-1}. \end{aligned} \tag{5.25}$$

b) Explain that the DWT matrix is orthogonal if and only if N is even. Also explain that it is only the last column which spoils the orthogonality.

5.3 Implementation of the DWT and examples

The DWT is straightforward to implement: Simply iterate Equation (5.19) for $m, m-1, \ldots, 1$. For each iteration we will use a *kernel function* which takes as input the coordinates $(c_{m,0}, c_{m,1}, \ldots)$, and returns the coordinates $(c_{m-1,0}, w_{m-1,0}, c_{m-1,1}, w_{m-1,1}, \ldots)$, i.e. computes one stage of the DWT, but with a different order of the coordinates than in the basis $(\boldsymbol{\phi}_m, \boldsymbol{\psi}_m)$. This turns out to be the natural ordering for computing the DWT in-place. As an example, the kernel function for the Haar wavelet can be implemented as follows (for simplicity this first version of the code assumes that N is even):

```
def dwt_kernel_haar(x, bd_mode):
    x /= sqrt(2)
    for k in range(2,len(x) - 1,2):
        a, b = x[k] + x[k+1], x[k] - x[k+1]
        x[k], x[k+1] = a, b
```

The code above accepts two-dimensional data, just as our function `FFTImpl`. Thus, the function may be applied simultaneously to all channels in a sound. The reason for using a general kernel function will be apparent later, when we

change to different types of wavelets. It is not meant that you call this kernel function directly. Instead every time you apply the DWT call the function

```
DWTImpl(x, m, wave_name, bd_mode='symm', dual=False, transpose=False)
```

`x` is the input to the DWT, and `m` is the number of levels. The three last parameters will be addressed later in the book (the `bd_mode`-parameter addresses how the boundary should be handled). The function also sets meaningful default values for the three last parameters, so that you mostly only need to provide the three first parameters.

We will later construct other wavelets, and we will distinguish them by using different names. This is the purposes of the `wave_name` parameter. This parameter is sent to a function called `find_kernel` which looks up a kernel function by name (`find_kernel` also uses the `dual` and `transpose` parameters to take a decision on which kernel to choose). The Haar wavelet is identified with the name `"Haar"`. When this is input to `DWTImpl`, `find_kernel` returns the `dwt_kernel_haar` kernel. The kernel is then used as input to the following function:

```
def DWTImpl_internal(x, m, f, bd_mode):
    for res in range(m):
        f(x[0::2**res], bd_mode)
    reorganize_coeffs_forward(x, m)
```

The code is applied to all columns if the data is two-dimensional, and we see that the kernel function is invoked one time for each resolution. To reorder coordinates in the same order as $(\boldsymbol{\phi}_m, \boldsymbol{\psi}_m)$, note that the coordinates from $\boldsymbol{\phi}_m$ above end up at indices $k2^m$, where m represents the current stage, and k runs through the indices. The function `reorganize_coeffs_forward` uses this to reorder the coordinates (you will be spared the details in this implementation). Although the DWT requires this reorganization, this may not be required in practice. In Exercise 5.27 we go through some aspects of this implementation. The implementation is not recursive, as the for-loop runs through the different stages.

In this implementation, note that the first levels require the most operations, 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.

There is a similar setup for the IDWT:

```
IDWTImpl(x, m, wave_name, bd_mode='symm', dual=False, transpose=False)
```

If the `wave_name`-parameter is set to `"Haar"`, also this function will use the `find_kernel` function to look up another kernel function, `idwt_kernel_haar`

(when N is even, this uses the exact same code as `dwt_kernel_haar`. For N is odd, see Exercises 5.8 and 5.26). This is then sent as input to

```
def IDWTImpl_internal(x, m, f, bd_mode):
    reorganize_coeffs_reverse(x, m)
    for res in range(m - 1, -1, -1):
        f(x[0::2**res], bd_mode)
```

Here the steps are simply performed in the reverse order, and by iterating Equation (5.20).

In the next sections we will consider other cases where the underlying function ϕ may be something else, and 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*. Let us round off this section with some important examples.

Example 5.9: Computing the DWT by hand

In some cases, the DWT can be computed by hand, keeping in mind its definition as a change of coordinates. As an example, consider the simple vector $\boldsymbol{x}$ of length $2^{10} = 1024$ defined by

$$x_n = \begin{cases} 1 & \text{for } n < 512 \\ 0 & \text{for } n \geq 512, \end{cases}$$

and let us compute the 10-level DWT of this vector by first visualizing the function with these coordinates. Since $m = 10$ here, we should view $\boldsymbol{x}$ as coordinates in the basis $\boldsymbol{\phi}_{10}$ of a function $f(t) \in V_{10}$. This is $f(t) = \sum_{n=0}^{511} \phi_{10,n}$, and since $\phi_{10,n}$ is supported on $[2^{-10}n, 2^{-10}(n+1))$, the support of f has width $512 \times 2^{-10} = 1/2$ (512 translates, each with width 2^{-10}). Moreover, since $\phi_{10,n}$ is $2^{10/2} = 2^5 = 32$ on $[2^{-10}n, 2^{-10}(n+1))$ and 0 elsewhere, it is clear that

$$f(t) = \begin{cases} 32 & \text{for } 0 \leq t < 1/2 \\ 0 & \text{for } 0t \geq 1/2. \end{cases}$$

This is by definition a function in V_1: f must in fact be a multiplum of $\phi_{1,0}$, since this also is supported on $[0, 1/2)$. We can thus write $f(t) = c\phi_{1,0}(t)$ for some c. We can find c by setting $t = 0$. This gives that $32 = 2^{1/2}c$ (since $f(0) = 32$, $\phi_{1,0}(0) = 2^{1/2}$), so that $c = 32/\sqrt{2}$. This means that $f(t) = \frac{32}{\sqrt{2}}\phi_{1,0}(t)$, f is in V_1, and with coordinates $(32/\sqrt{2}, 0, \ldots, 0)$ in $\boldsymbol{\phi}_1$.

When we run a 10-level DWT we make a change of coordinates from $\boldsymbol{\phi}_{10}$ to $(\boldsymbol{\phi}_0, \boldsymbol{\psi}_0, \cdots, \boldsymbol{\psi}_9)$. The first 9 levels give us the coordinates in $(\boldsymbol{\phi}_1, \boldsymbol{\psi}_1, \boldsymbol{\psi}_2, \ldots, \boldsymbol{\psi}_9)$, and these are $(32/\sqrt{2}, 0, \ldots, 0)$ from what we showed. It remains thus only to perform the last level in the DWT, i.e. perform the change of coordinates from $\boldsymbol{\phi}_1$ to $(\boldsymbol{\phi}_0, \boldsymbol{\psi}_0)$. Since $\phi_{1,0} = \frac{1}{\sqrt{2}}(\phi_{0,0} + \psi_{0,0})$, so that we get

$$f(t) = \frac{32}{\sqrt{2}}\phi_{1,0}(t) = \frac{32}{\sqrt{2}}\frac{1}{\sqrt{2}}(\phi_{0,0} + \psi_{0,0}) = 16\phi_{0,0} + 16\psi_{0,0}.$$

From this we see that the coordinate vector of f in $(\boldsymbol{\phi}_0, \boldsymbol{\psi}_0, \cdots, \boldsymbol{\psi}_9)$, i.e. the 10-level DWT of $\boldsymbol{x}$, is $(16, 16, 0, 0, \ldots, 0)$. Note that here V_0 and W_0 are both 1-dimensional, since V_{10} was assumed to be of dimension 2^{10} (in particular, $N = 1$).

It is straightforward to verify what we found using the algorithm above:

```
x = hstack([ones(512), zeros(512)])
DWTImpl(x, 10, 'Haar')
print x
```

The reason why the method from this example worked was that the vector we started with had a simple representation in the wavelet basis, actually it equaled the coordinates of a basis function in $\boldsymbol{\phi}_1$. Usually this is not the case, and our only possibility then is to run the DWT on a computer.

Example 5.10: DWT on sound

Let us plot the samples of our audio sample file, and compare them with the first order DWT. Both are shown in Figure 5.10.

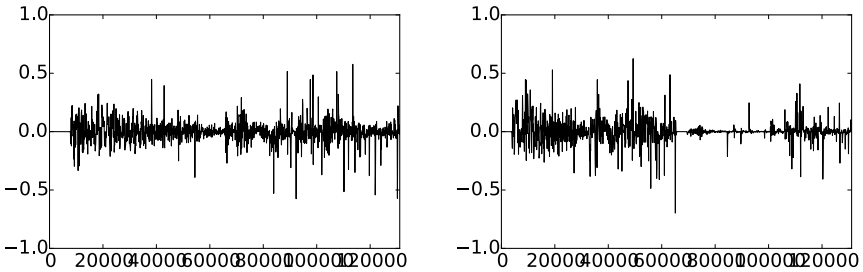

Figure 5.10: The 2^{17} first sound samples (left) and the DWT coefficients (right) of the sound `castanets.wav`.

The first part of the DWT plot represents the low resolution part, the second the detail.

Since $\phi(2^m t - n) \in V_m$ oscillates more quickly than $\phi(t - n) \in V_0$, one is lead to believe that coefficients from lower order resolution spaces correspond to lower frequencies. The functions $\phi_{m,n}$ do not correspond to pure tones in the setting of wavelets, however, but let us nevertheless listen to sound from the different resolution spaces. The code base includes a function `forw_comp_rev_DWT` which runs an m-level DWT on the first samples of the audio sample file, extracts the detail or the low-resolution approximation, and runs an IDWT to reconstruct the sound. Since the returned values may lie outside the legal range $[-1, 1]$, the values are normalized at the end.

To listen to the low-resolution approximation, write

```
x, fs = forw_comp_rev_DWT(m, 'Haar')
play(x, fs)
```

It is instructive to run this code for different values of m. For $m = 2$ we clearly hear a degradation in the sound. For $m = 4$ and above most of the sound is unrecognizable, since too much of the detail is omitted. To be more precise, when listening to the sound by throwing away detail from W_0, W_1,...,W_{m-1}, we are left with a 2^{-m} share of the data.

Let us also consider the detail. For $m = 1$ this can be played as follows

```
x, fs = forw_comp_rev_DWT(1, 'Haar', 0)
play(x, fs)
```

We see that the detail is quite significant, so that the first order wavelet approximation does not give a very good approximation. For $m = 2$ the detail can be played as follows

```
x, fs = forw_comp_rev_DWT(2, 'Haar', 0)
play(x, fs)
```

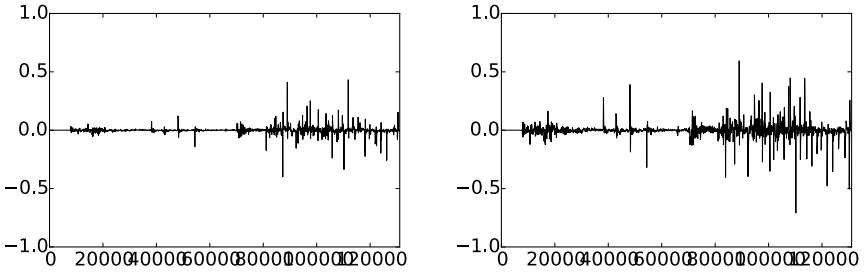

Figure 5.11: The detail in our audio sample file, for $m = 1$ (left) and $m = 2$ (right).

The errors are shown in Figure 5.11. 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. Since more and more information is contained in the detail components as we increase m, we here see the opposite effect: The sound gradually improves in quality as we increase m.

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 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 5.11: DWT on the samples of a mathematical function

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 5.12 plots the error for different functions, with $N = 1024$.

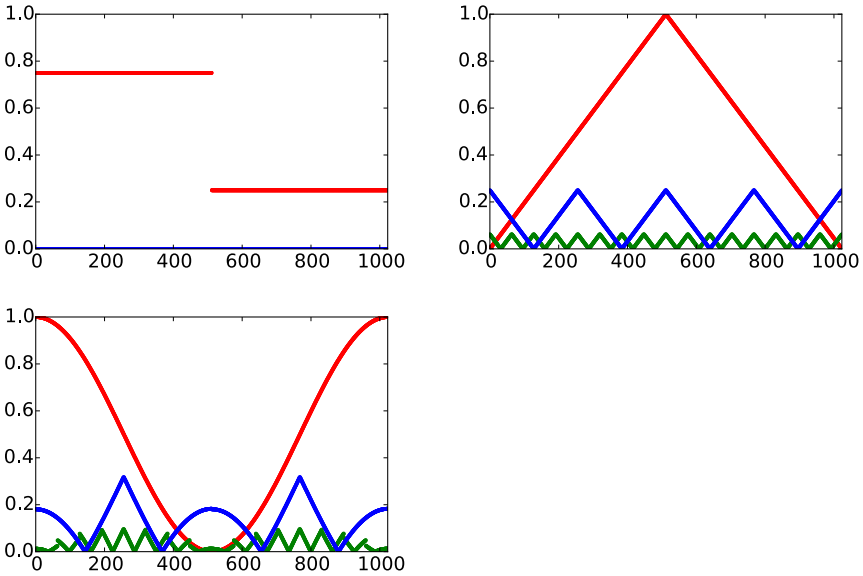

Figure 5.12: The error (i.e. the contribution from $W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}$) for $N = 1024$ when f is a square wave, the linear function $f(t) = 1 - 2|1/2 - t/N|$, and the trigonometric function $f(t) = 1/2 + \cos(2\pi t/N)/2$, respectively. The detail is indicated for $m = 6$ and $m = 8$.

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.

Example 5.12: Computing the wavelet coefficients

For the functions we plotted in the previous example we used the functions `DWTImpl`, `IDWTImpl` to plot the error, but it is also possible to compute the wavelet coefficients $w_{m,n}$ exactly. You will be asked to do this in exercises 5.23 and 5.24. To exemplify the general procedure for this, consider the function $f(t) = 1 - t/N$. This decreases linearly from 1 to 0 on $[0, N]$, so that it is not piecewise constant, and does not lie in any of the spaces V_m. We can instead consider $\text{proj}_{V_m} f \in V_m$, and apply the DWT to this. Let us compute the $\boldsymbol{\psi}_m$-coordinates $w_{m,n}$ of $\text{proj}_{V_m} f$ in the orthonormal basis $(\boldsymbol{\phi}_0, \boldsymbol{\psi}_0, \boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_{m-1})$. The orthogonal decomposition theorem says that

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

Using the definition of $\psi_{m,n}$ we see that this can also be written as

$$2^{m/2} \int_0^N (1 - t/N)\psi(2^m t - n)dt = 2^{m/2} \left(\int_0^N \psi(2^m t - n)dt - \int_0^N \frac{t}{N}\psi(2^m t - n)dt \right).$$

Using Observation 5.14 we get that $\int_0^N \psi(2^m t - n)dt = 0$, so that the first term above vanishes. Moreover, $\psi_{m,n}$ is nonzero only on $[2^{-m}n, 2^{-m}(n+1))$, and is 1 on $[2^{-m}n, 2^{-m}(n+1/2))$, and -1 on $[2^{-m}(n+1/2), 2^{-m}(n+1))$. We therefore get

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

We see in particular that $w_{m,n} \to 0$ when $m \to \infty$. Also, all coordinates were equal, i.e. $w_{m,0} = w_{m,1} = w_{m,2} = \cdots$. It is not too hard to convince oneself that this equality has to do with the fact that f is linear. 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 $w_{m,n}$ symbolically, but instead run implementations like `DWTImpl`, `IDWTImpl` on a computer.

Exercise 5.13: Implement IDWT for The Haar wavelet

Write a function `idwt_kernel_haar` which uses the formulas (5.24) to implement the IDWT, similarly to how the function `dwt_kernel_haar` implemented the DWT using the formulas (5.25).

Exercise 5.14: Computing projections 4

Generalize Exercise 5.4 to the projections from V_{m+1} onto V_m and W_m.

Exercise 5.15: Scaling a function

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

Exercise 5.16: Direct sums

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

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

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

Exercise 5.17: Eigenvectors of direct sums

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?

Exercise 5.18: Invertibility of direct sums

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

Exercise 5.19: Multiplying direct sums

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

Exercise 5.20: Finding N

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 $\boldsymbol{\phi}_m$ to $(\boldsymbol{\phi}_0, \boldsymbol{\psi}_0, \boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_{m-2}, \boldsymbol{\psi}_{m-1})$.

Exercise 5.21: Different DWTs for similar vectors

In Figure 5.13 we have plotted the DWT's of two vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. In both vectors we have 16 ones followed by 16 zeros, and this pattern repeats cyclically so that the length of both vectors is 256. The only difference is that the second vector is obtained by delaying the first vector with one element.

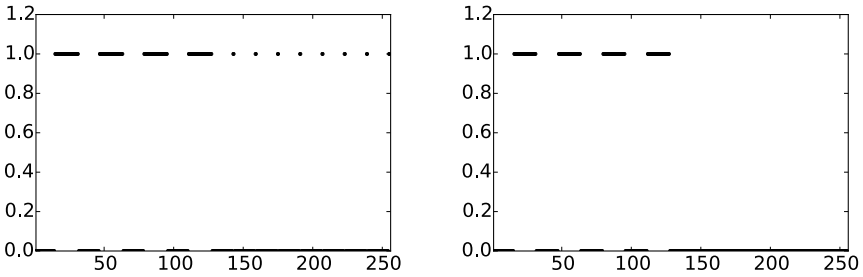

Figure 5.13: 2 vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ which seem equal, but where the DWT's are very different.

You see that the two DWT's are very different: For the first vector we see that there is much detail present (the second part of the plot), while for the second vector there is no detail present. Attempt to explain why this is the case. Based on your answer, also attempt to explain what can happen if you change the point of discontinuity for the piecewise constant function in the left part of Figure 5.11 to something else.

Exercise 5.22: Construct a sound

Attempt to construct a (nonzero) sound where the low resolution approximations equal the sound itself for $m = 1$, $m = 2$.

Exercise 5.23: Exact computation of wavelet coefficients 1

Compute the wavelet detail coefficients analytically for the functions in Example 5.11, 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 5.12.

Exercise 5.24: Exact compution of wavelet coefficients 2

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 5.12. How do these compare with the coefficients from the Exercise 5.23?

Exercise 5.25: Computing the DWT of a simple vector

Suppose that we have the vector $\boldsymbol{x}$ with length $2^{10} = 1024$, defined by $x_n = 1$ for n even, $x_n = -1$ for n odd. What will be the result if you run a 10-level DWT on $\boldsymbol{x}$? Use the function `DWTImpl` to verify what you have found.

Hint. We defined ψ by $\psi(t) = (\phi_{1,0}(t) - \phi_{1,1}(t))/\sqrt{2}$. From this connection it follows that $\psi_{9,n} = (\phi_{10,2n} - \phi_{10,2n+1})/\sqrt{2}$, and thus $\phi_{10,2n} - \phi_{10,2n+1} = \sqrt{2}\psi_{9,n}$. Try to couple this identity with the alternating sign you see in $\boldsymbol{x}$.

Exercise 5.26: The Haar wavelet when N is odd

Use the results from Exercise 5.8 to rewrite the implementations `dwt_kernel_haar` and `idwt_kernel_haar` so that they also work in the case when N is odd.

Exercise 5.27: in-place DWT

Show that the coordinates in $\boldsymbol{\phi}_m$ after an in-place m-level DWT end up at indices $k2^m$, $k = 0, 1, 2, \ldots$. Show similarly that the coordinates in $\boldsymbol{\psi}_m$ after an in-place m-level DWT end up at indices $2^{m-1} + k2^m$, $k = 0, 1, 2, \ldots$. Find these indices in the code for the function `reorganize_coefficients`.

5.4 A wavelet based on piecewise linear functions

Unfortutately, piecewise constant functions are too simple to provide good approximations. In this section 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. We will later see that this implies that the corresponding DWT and IDWT have simple implementations with higher precision. Our experience from deriving Haar wavelets will guide us in the construction of piecewise linear wavelets. The first task is to define the new resolution spaces.

Definition 5.18. *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 $[n2^{-m}, (n+1)2^{-m})$.

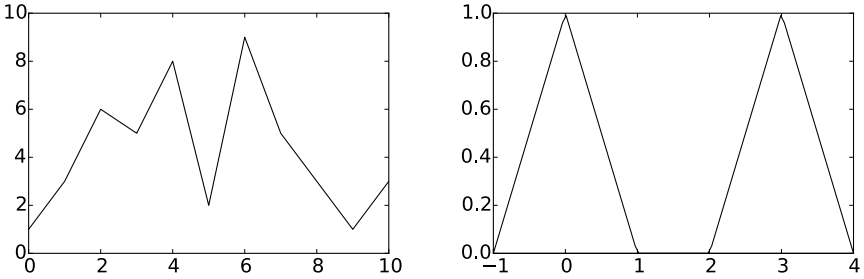

Figure 5.14: A piecewise linear function and the two functions $\phi(t)$ and $\phi(t-3)$.

Any $f \in V_m$ is uniquely determined by its values in the points $\{2^{-m}n\}_{n=0}^{2^m N-1}$. The linear mapping which sends f to these samples is thus an isomorphism from V_m onto R^{N2^m}, so that the dimension of V_m is $\mathbb{N}2^m$. The lft plot in Figure 5.14 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 the right plot in Figure 5.14. 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.

Lemma 5.19. *The function ϕ.*

Let the function ϕ be defined by

$$\phi(t) = \begin{cases} 1 - |t|, & \text{if } -1 \le t \le 1; \\ 0, & \text{otherwise;} \end{cases} \tag{5.26}$$

and for any $m \ge 0$ set

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

and $\boldsymbol{\phi}_m = \{\phi_{m,n}\}_{n=0}^{2^m N-1}$. $\boldsymbol{\phi}_m$ is a basis for V_m, and $\phi_{0,n}(t)$ is the function in V_0 with smallest support that is nonzero at $t = n$.

Proof. It is clear that $\phi_{m,n} \in V_m$, and

$$\phi_{m,n'}(n2^{-m}) = 2^{m/2}\phi(2^m(2^{-m}n) - n') = 2^{m/2}\phi(n - n').$$

Since ϕ is zero at all nonzero integers, and $\phi(0) = 1$, we see that $\phi_{m,n'}(n2^{-m}) = 2^{m/2}$ when $n' = n$, and 0 if $n' \neq n$. Let $L_m : V_m \to R^{N2^m}$ be the isomorphism mentioned above which sends $f \in V_m$ to the samples in the points $\{2^{-m}n\}_{n=0}^{2^m N-1}$. Our calculation shows that $L_m(\phi_{m,n}) = 2^{m/2}e_n$. Since L_m is an isomorphism it follows that $\boldsymbol{\phi}_m = \{\phi_{m,n}\}_{n=0}^{2^m N-1}$ is a basis for V_m.

Suppose that the function $g \in V_0$ has smaller support than $\phi_{0,n}$, but is nonzero at $t = n$. We must have that $L_0(g) = c\boldsymbol{e}_n$ for some c, since g is zero on the integers different from n. But then g is a multiple of $\phi_{0,n}$, so that it is the function in V_0 with smallest support that is nonzero at $t = n$. □

The function ϕ and its translates and dilates are often referred to as hat functions for obvious reasons. Note that the new function ϕ is nonzero for small negative x-values, contrary to the ϕ we defined in Chapter 5. If we plotted the function on $[0, N)$, we would see the nonzero parts at the beginning and end of this interval, due to the period N, but we will mostly plot on an interval around zero, since such an interval captures the entire support of the function. Also for the piecewise linear wavelet the coordinates of a basis function is given by the samples:

Lemma 5.20. *Writing in terms of the samples.*
A function $f \in V_m$ may be written as

$$f(t) = \sum_{n=0}^{2^m N-1} f(n/2^m)2^{-m/2}\phi_{m,n}(t). \tag{5.27}$$

An essential property also here is that the spaces are nested.

Lemma 5.21. *Resolution spaces are nested.*
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 5.3 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. 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 5.22. *The two-scale equation.*
The functions $\phi_{0,n}$ satisfy the relation

$$\phi_{0,n} = \frac{1}{\sqrt{2}}\left(\frac{1}{2}\phi_{1,2n-1} + \phi_{1,2n} + \frac{1}{2}\phi_{1,2n+1}\right). \tag{5.28}$$

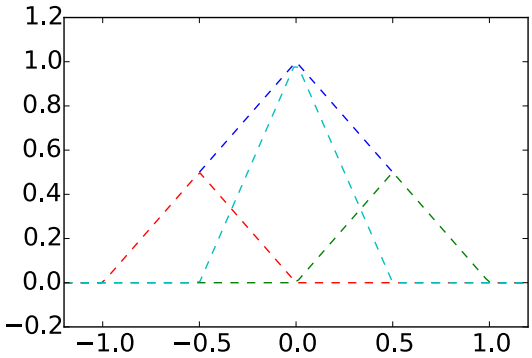

Figure 5.15: How $\phi(t)$ can be decomposed as a linear combination of $\phi_{1,-1}$, $\phi_{1,0}$, and $\phi_{1,1}$.

Proof. Since $\phi_{0,n}$ is in V_0 it may be expressed in the basis $\boldsymbol{\phi}_1$ with formula (5.27),

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

The relation (5.28) 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. □

The relationship given by Equation (5.28) is shown in Figure 5.15.

5.4.1 Detail spaces and wavelets

The next step in our derivation of wavelets for piecewise linear functions is the definition of the detail spaces. We need to determine a space W_0 that is linearly independent from V_0, and so that $V_1 = V_0 \oplus W_0$. In the case of piecewise constant functions we started with a function g_1 in V_1, computed the least squares approximation g_0 in V_0, and then defined the error function $e_0 = g_1 - g_0$, with $e_0 \in W_0$ and W_0 as the orthogonal complement of V_0 in V_1.

It turns out that this strategy is less appealing in the case of piecewise linear functions. The reason is that the functions $\phi_{0,n}$ are not orthogonal anymore (see Exercise 5.32). Due to this we have no simple, orthogonal basis for the set of piecewise linear functions, so that the orthogonal decomposition theorem fails to give us the projection onto V_0 in a simple way. It is therefore no reason to use the orthogonal complement of V_0 in V_1 as our error space, since it is hard to write a piecewise linear function as a sum of two other piecewise linear functions which are orthogonal. Instead of using projections to find low-resolution approximations, and orthogonal complements to find error functions, we will attempt the following simple approximation method:

Definition 5.23. *Alternative projection.*
Let g_1 be a function in V_1 given by

$$g_1 = \sum_{n=0}^{2N-1} c_{1,n}\phi_{1,n}. \tag{5.29}$$

The approximation $g_0 = P(g_1)$ in V_0 is defined as the unique function in V_0 which has the same values as g_1 at the integers, i.e.

$$g_0(n) = g_1(n), \quad n = 0, 1, \ldots, N-1. \tag{5.30}$$

It is easy to show that $P(g_1)$ actually is different from the projection of g_1 onto V_0: If $g_1 = \phi_{1,1}$, then g_1 is zero at the integers, and then clearly $P(g_1) = 0$. But in Exercise 5.31 you will be asked to compute the projection onto V_0 using different means than the orthogonal decomposition theorem, and the result will be seen to be nonzero. It is also very easy to see that the coordinates of g_0 in $\boldsymbol{\phi}_0$ can be obtained by dropping every second coordinate of g_0 in $\boldsymbol{\phi}_1$. To be more precise, the following holds:

Lemma 5.24. *Expression for the alternative projection.*
We have that

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

Once this approximation method is determined, it is straightforward to determine the detail space as the space of error functions.

Lemma 5.25. *Resolution spaces.*
Define

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

and

$$\psi(t) = \frac{1}{\sqrt{2}}\phi_{1,1}(t) \qquad \psi_{m,n}(t) = 2^{m/2}\psi(2^m t - n). \tag{5.31}$$

Suppose that $g_1 \in V_1$ and that $g_0 = P(g_1)$. Then

- the error $e_0 = g_1 - g_0$ lies in W_0,
- $\boldsymbol{\psi}_0 = \{\psi_{0,n}\}_{n=0}^{N-1}$ is a basis for W_0.
- V_0 and W_0 are linearly independent, and $V_1 = V_0 \oplus W_0$.

Proof. Since $g_0(n) = g_1(n)$ for all integers n, $e_0(n) = (g_1 - g_0)(n) = 0$, so that $e_0 \in W_0$. This proves the first statement.

For the second statement, note first that

$$\begin{aligned}\psi_{0,n}(t) = \psi(t-n) &= \frac{1}{\sqrt{2}}\phi_{1,1}(t-n) = \phi(2(t-n)-1) \\ &= \phi(2t-(2n+1)) = \frac{1}{\sqrt{2}}\phi_{1,2n+1}(t). \end{aligned} \tag{5.32}$$

$\boldsymbol{\psi}_0$ is thus a linearly independent set of dimension N, since it corresponds to a subset of $\boldsymbol{\phi}_1$. Since $\phi_{1,2n+1}$ is nonzero only on $(n, n+1)$, it follows that all of $\boldsymbol{\psi}_0$ lies in W_0. Clearly then $\boldsymbol{\psi}_0$ is also a basis for W_0, since W_0 also has dimension N (its image under L_1 consists of points where every second component is zero).

Consider finally a linear combination from $\boldsymbol{\phi}_0$ and $\boldsymbol{\psi}_0$ which gives zero:

$$\sum_{n=0}^{N-1} a_n \phi_{0,n} + \sum_{n=0}^{N-1} b_n \psi_{0,n} = 0.$$

If we evaluate this at $t = k$, we see that $\psi_{0,n}(k) = 0$, $\phi_{0,n}(k) = 0$ when $n \neq k$, and $\phi_{0,k}(k) = 1$. When we evaluate at k we thus get a_k, which must be zero. If we then evaluate at $t = k + 1/2$ we get in a similar way that all $b_n = 0$, and it follows that V_0 and W_0 are linearly independent. That $V_1 = V_0 \oplus W_0$ follows from the fact that V_1 has dimension $2N$, and V_0 and W_0 both have dimension N. □

We can define W_m in a similar way for $m > 0$, and generalize the lemma to W_m. We can thus state the following analog to Theorem 5.16 for writing $g_m \in V_m$ as a sum of a low-resolution approximation $g_{m-1} \in V_{m-1}$, and a detail/error component $e_{m-1} \in W_{m-1}$.

Theorem 5.26. *Decomposing V_m.*

The space V_m can be decomposed as the direct sum $V_m = V_{m-1} \oplus W_{m-1}$ where

$$W_{m-1} = \{f \in V_m \mid f(n/2^{m-1}) = 0, \quad \text{for } n = 0, 1, \ldots, 2^{m-1}N - 1\}.$$

W_m has the base $\boldsymbol{\psi}_m = \{\psi_{m,n}\}_{n=0}^{2^m N-1}$, and V_m has the two bases

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

With this result we can define the DWT and the IDWT with their stages as before, but the matrices themmselves are now different. For the IDWT (i.e. $P_{\boldsymbol{\phi}_1 \leftarrow (\boldsymbol{\phi}_0, \boldsymbol{\psi}_0)}$), the columns in the matrix can be found from equations (5.28) and (5.32), i.e.

$$\begin{aligned} \phi_{0,n} &= \frac{1}{\sqrt{2}} \left(\frac{1}{2}\phi_{1,2n-1} + \phi_{1,2n} + \frac{1}{2}\phi_{1,2n+1} \right) \\ \psi_{0,n} &= \frac{1}{\sqrt{2}}\phi_{1,2n+1}. \end{aligned} \tag{5.33}$$

This states that

$$G = P_{\boldsymbol{\phi}_m \leftarrow \mathcal{C}_m} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 1/2 & 1 & 1/2 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ 1/2 & 0 & 0 & 0 & \cdots & 0 & 1/2 & 1 \end{pmatrix} \tag{5.34}$$

In general we will call a matrix on the form

$$\begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \lambda & 1 & \lambda & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ \lambda & 0 & 0 & 0 & \cdots & 0 & \lambda & 1 \end{pmatrix} \tag{5.35}$$

an *elementary lifting matrix of odd type*, and denote it by B_λ. This results from the identity matrix by adding λ times the preceding and succeeding rows to the odd-indexed rows. Since the even-indexed rows are left untouched, the inverse is clearly obtained by subtracting λ times the preceding and succeeding rows to the odd-indexed rows, i.e. $(B_\lambda)^{-1} = B_{-\lambda}$. This means that the matrix for the DWT is also easily found, and

$$H = P_{\mathcal{C}_m \leftarrow \boldsymbol{\phi}_m} = \sqrt{2} B_{-1/2} \tag{5.36}$$

$$G = P_{\boldsymbol{\phi}_m \leftarrow \mathcal{C}_m} = \frac{1}{\sqrt{2}} B_{1/2}. \tag{5.37}$$

In the exercises you will be asked to implement a function `lifting_odd_symm` which computes B_λ. Using this the DWT kernel transformation for the piecewise linear wavelet can be applied to a vector `x` as follows.

```
x *= sqrt(2)
lifting_odd_symm(-0.5, x, symm)
```

The IDWT kernel transformation is computed similarly. Functions `dwt_kernel_pwl0`, `idwt_kernel_pwl0` which perform these steps are included in the code base. The 0 stands for 0 vanishing moments. We defined vanishing moments after Observation 5.14, and we will have more to say about vanishing moments later.

Example 5.28: DWT on sound

Using the new kernels, let us plot the listen to the new low resolution approximations, as well as plot and listen to the detail, as we did in Example 5.10. First we listen to the low-resolution approximation.

```
x, fs = forw_comp_rev_DWT(m, 'pwl0')
play(x, fs)
```

There is a new and undesired effect when we increase m here: The castanet sound seems to grow strange. The sounds from the castanets are perhaps the sound with the highest frequencies.

Now for the detail. For $m = 1$ this can be played as follows

```
x, fs = forw_comp_rev_DWT(1, 'pwl0', 0)
play(x, fs)
```

For $m = 2$ the detail can be played as follows

```
x, fs = forw_comp_rev_DWT(2, 'pwl0', 0)
play(x, fs)
```

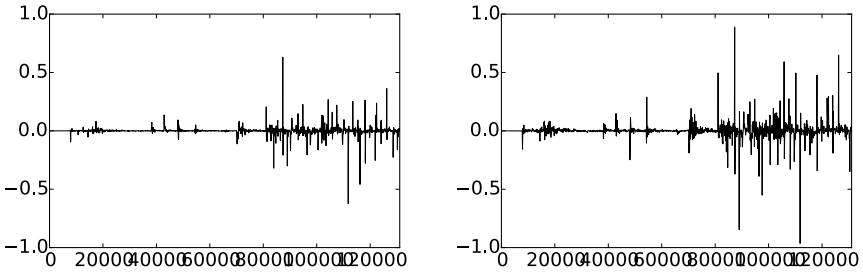

Figure 5.16: The detail in our audio sample file for the piecewise linear wavelet, for $m = 1$ (left) and $m = 2$ (right).

The errors are shown in Figure 5.16. When comparing with Example 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 5.29: DWT on the samples of a mathematical function

Let us also repeat Example 5.11, where we plotted the detail/error at different resolutions for the samples of a mathematical function.

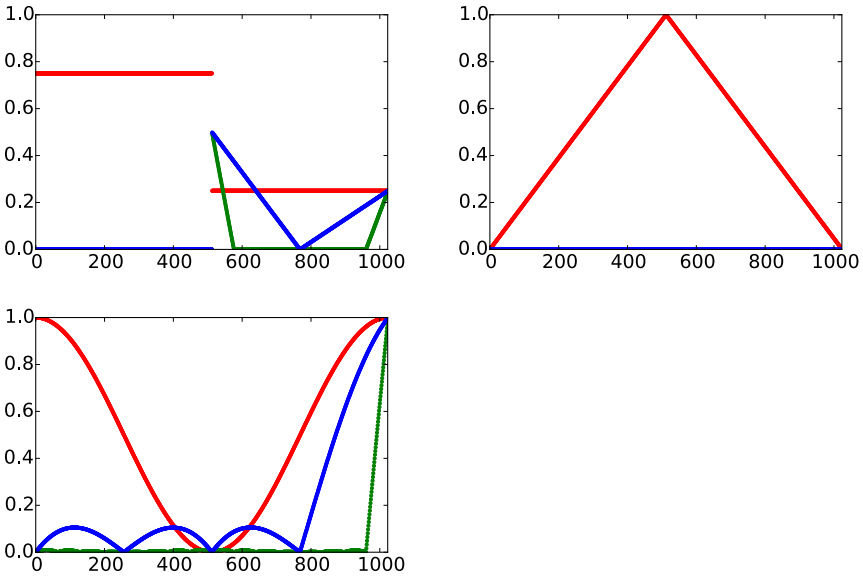

Figure 5.17: The error (i.e. the contribution from $W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}$) for $N = 1025$ when f is a square wave, the linear function $f(t) = 1 - 2|1/2 - t/N|$, and the trigonometric function $f(t) = 1/2 + \cos(2\pi t/N)/2$, respectivey. The detail is indicated for $m = 6$ and $m = 8$.

Figure 5.17 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 linear, so there is no error when we represent the function from the space V_0. With the third function, however, we see an error.

Exercise 5.30: The vector of samples is the coordinate vector 2

Show that, for $f \in V_0$ we have that $[f]_{\phi_0} = (f(0), f(1), \ldots, f(N-1))$. This shows that, also for the piecewise linear wavelet, there is no loss of information in working with the samples of f rather than f itself.

Exercise 5.31: Computing projections

In this exercise we will show how the projection of $\phi_{1,1}$ onto V_0 can be computed. We will see from this that it is nonzero, and that its support is the entire $[0, N]$. Let $f = \text{proj}_{V_0}\phi_{1,1}$, and let $x_n = f(n)$ for $0 \leq n < N$. This means that, on $(n, n+1)$, $f(t) = x_n + (x_{n+1} - x_n)(t - n)$.

a) Show that $\int_n^{n+1} f(t)^2 dt = (x_n^2 + x_n x_{n+1} + x_{n+1}^2)/3$.

b) Show that

$$\int_0^{1/2} (x_0 + (x_1 - x_0)t)\phi_{1,1}(t)dt = 2\sqrt{2}\left(\frac{1}{12}x_0 + \frac{1}{24}x_1\right)$$
$$\int_{1/2}^{1} (x_0 + (x_1 - x_0)t)\phi_{1,1}(t)dt = 2\sqrt{2}\left(\frac{1}{24}x_0 + \frac{1}{12}x_1\right).$$

c) Use the fact that

$$\int_0^N (\phi_{1,1}(t) - \sum_{n=0}^{N-1} x_n \phi_{0,n}(t))^2 dt$$
$$= \int_0^1 \phi_{1,1}(t)^2 dt - 2\int_0^{1/2} (x_0 + (x_1 - x_0)t)\phi_{1,1}(t)dt - 2\int_{1/2}^1 (x_0 + (x_1 - x_0)t)\phi_{1,1}(t)dt$$
$$+ \sum_{n=0}^{N-1} \int_n^{n+1} (x_n + (x_{n-1} - x_n)t)^2 dt$$

and a) and b) to find an expression for $\|\phi_{1,1}(t) - \sum_{n=0}^{N-1} x_n \phi_{0,n}(t)\|^2$.

d) To find the minimum least squares error, we can set the gradient of the expression in c) to zero, and thus find the expression for the projection of $\phi_{1,1}$ onto V_0. Show that the values $\{x_n\}_{n=0}^{N-1}$ can be found by solving the equation $S\boldsymbol{x} = \boldsymbol{b}$, where $S = \frac{1}{3}\{1, \underline{4}, 1\}$ is an $N \times N$ symmetric filter, and $\boldsymbol{b}$ is the vector with components $b_0 = b_1 = \sqrt{2}/2$, and $b_k = 0$ for $k \geq 2$.

e) Solve the system in d. for some values of N to verify that the projection of $\phi_{1,1}$ onto V_0 is nonzero, and that its support covers the entire $[0, N]$.

Exercise 5.32: Non-orthogonality for the piecewise linear wavelet

Show that

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

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

Exercise 5.33: Implement elementary lifting steps of odd type

Write a function

```
lifting_odd_symm(lambda, x, bd_mode)
```

which applies an elementary lifting matrix of odd type (Equation (5.35)) to $\boldsymbol{x}$. Assume that N is even. The parameter `bd_mode` should do nothing, as we will return to this parameter later. The function should not perform matrix multiplication, and apply as few multiplications as possible.

Exercise 5.34: Wavelets based on polynomials

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.

5.5 Alternative wavelet based on piecewise linear functions

For the scaling function used for piecewise linear functions, $\{\phi(t-n)\}_{0 \leq n < N}$ were not orthogonal anymore, contrary to the case for piecewise constant functions. We were still able to construct what we could call resolution spaces and detail spaces. We also mentioned that having many vanishing moments is desirable for a mother wavelet, and that the scaling function used for piecewise constant functions had one vanishing moment. It is easily checked, however, that the mother wavelet we now introduced for piecewise linear functions (i.e. $\psi(t) = \frac{1}{\sqrt{2}}\phi_{1,1}(t)$) has no vanishing moments. Therefore, this is not a very good choice of mother wavelet. We will attempt the following adjustment strategy to construct an alternative mother wavelet $\hat{\psi}$ which has two vanishing moments, i.e. one more than the Haar wavelet.

Idea 5.27. *Adjusting the wavelet construction.*
Adjust the wavelet construction in Theorem 5.26 to

$$\hat{\psi} = \psi - \alpha\phi_{0,0} - \beta\phi_{0,1} \tag{5.38}$$

and choose α, β so that

$$\int_0^N \hat{\psi}(t)\,dt = \int_0^N t\hat{\psi}(t)\,dt = 0, \tag{5.39}$$

and define $\boldsymbol{\psi}_m = \{\hat{\psi}_{m,n}\}_{n=0}^{N2^m-1}$, and W_m as the space spanned by $\boldsymbol{\psi}_m$.

We thus have two free variables α, β in Equation (5.38), to enforce the two conditions in Equation (5.39). In Exercise 5.38 you are taken through the details of solving this as two linear equations in the two unknowns α and β, and this gives the following result:

Lemma 5.28. *The new function ψ.*
The function

$$\hat{\psi}(t) = \psi(t) - \frac{1}{4}\big(\phi_{0,0}(t) + \phi_{0,1}(t)\big) \tag{5.40}$$

satisfies the conditions (5.39).

Using Equation (5.28), which stated that

$$\phi_{0,n} = \frac{1}{\sqrt{2}}\left(\frac{1}{2}\phi_{1,2n-1} + \phi_{1,2n} + \frac{1}{2}\phi_{1,2n+1}\right), \tag{5.41}$$

we get

$$\begin{aligned}
\hat{\psi}_{0,n} &= \psi_{0,n} - \frac{1}{4}\big(\phi_{0,n} + \phi_{0,n+1}\big) \\
&= \frac{1}{\sqrt{2}}\phi_{1,2n+1} - \frac{1}{4}\frac{1}{\sqrt{2}}\left(\frac{1}{2}\phi_{1,2n-1} + \phi_{1,2n} + \frac{1}{2}\phi_{1,2n+1}\right) \\
&- \frac{1}{4}\frac{1}{\sqrt{2}}\left(\frac{1}{2}\phi_{1,2n+1} + \phi_{1,2n+2} + \frac{1}{2}\phi_{1,2n+3}\right) \\
&= \frac{1}{\sqrt{2}}\left(-\frac{1}{8}\phi_{1,2n-1} - \frac{1}{4}\phi_{1,2n} + \frac{3}{4}\phi_{1,2n+1} - \frac{1}{4}\phi_{1,2n+2} - \frac{1}{8}\phi_{1,2n+3}\right)
\end{aligned} \tag{5.42}$$

In summary we have

$$\begin{aligned}
\phi_{0,n} &= \frac{1}{\sqrt{2}}(\frac{1}{2}\phi_{1,2n-1} + \phi_{1,2n} + \frac{1}{2}\phi_{1,2n+1}) \\
\hat{\psi}_{0,n} &= \frac{1}{\sqrt{2}}\left(-\frac{1}{8}\phi_{1,2n-1} - \frac{1}{4}\phi_{1,2n} + \frac{3}{4}\phi_{1,2n+1} - \frac{1}{4}\phi_{1,2n+2} - \frac{1}{8}\phi_{1,2n+3}\right),
\end{aligned} \tag{5.43}$$

The new function $\hat{\psi}$ is plotted in Figure 5.18.

We see that $\hat{\psi}$ has support $(-1, 2)$, and consist of four linear segments glued together. This is in contrast with the old ψ, which was simpler in that it had the shorther support $(0, 1)$, and consisted of only two linear segments glued together.

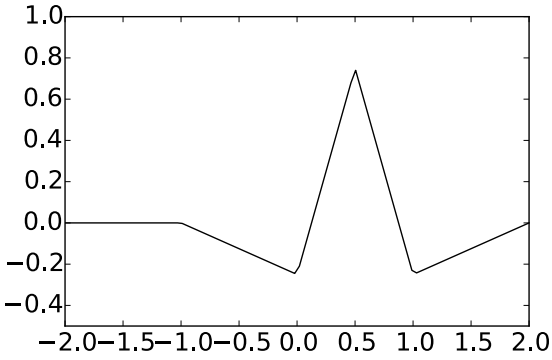

Figure 5.18: The function $\hat{\psi}$ we constructed as an alternative wavelet for piecewise linear functions.

It may therefore seem surprising that $\hat{\psi}$ is better suited for approximating functions than ψ. This is indeed a more complex fact, which may not be deduced by simply looking at plots of the functions.

The DWT in this new setting is the change of coordinates from $\boldsymbol{\phi}_m$ to

$$\hat{\mathcal{C}_m} = \{\phi_{m-1,0}, \hat{\psi}_{m-1,0}, \phi_{m-1,1}, \hat{\psi}_{m-1,1}, \cdots, \phi_{m-1,2^{m-1}N-1}, \hat{\psi}_{m-1,2^{m-1}N-1}\}.$$

Equation (5.40) states that

$$P_{\mathcal{C}_m \leftarrow \hat{\mathcal{C}_m}} = \begin{pmatrix} 1 & -1/4 & 0 & 0 & \cdots & 0 & 0 & -1/4 \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & -1/4 & 1 & -1/4 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & -1/4 & 1 & -1/4 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix}$$

(Column j for j even equals $\boldsymbol{e}_j$, since the basis functions $\phi_{0,n}$ are not altered). In general we will call a matrix on the form

$$\begin{pmatrix} 1 & \lambda & 0 & 0 & \cdots & 0 & 0 & \lambda \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \lambda & 1 & \lambda & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \lambda & 1 & \lambda \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix} \tag{5.44}$$

an *elementary lifting matrix of even type*, and denote it by A_λ. Using Equation (5.34) we can write

$$G = P_{\phi_m \leftarrow \hat{\mathcal{C}_m}} = P_{\phi_m \leftarrow \mathcal{C}_m} P_{\mathcal{C}_m \leftarrow \hat{\mathcal{C}_m}}, = \frac{1}{\sqrt{2}} B_{1/2} A_{-1/4}$$

This gives us a factorization of the IDWT in terms of lifting matrices. The inverse of elementary lifting matrices of even type can be found similarly to how we found the inverse of elementary lifting matrices of odd type, i.e. $(A_\lambda)^{-1} = A_{-\lambda}$.

This means that the matrix for the DWT is easily found also in this case, and

$$H = P_{\hat{\mathcal{C}_m} \leftarrow \phi_m} = \sqrt{2} A_{1/4} B_{-1/2} \tag{5.45}$$

$$G = P_{\phi_m \leftarrow \hat{\mathcal{C}_m}} = \frac{1}{\sqrt{2}} B_{1/2} A_{-1/4}. \tag{5.46}$$

Note that equations (5.43) also computes the matrix G, but we will rather use these factorizations, since the elementary lifting operations are already implemented in the exercises. We will also explain later why such a factorization is attractive in terms of saving computations. In the exercises you will be asked to implement a function `lifting_even_symm` which computes A_λ. Using this the DWT kernel transformation for the alternative piecewise linear wavelet can be applied to a vector `x` as follows.

```
x *= sqrt(2)
lifting_odd_symm(-0.5, x, symm)
lifting_even_symm(0.25, x, symm)
```

The IDWT kernel transformation is computed similarly. Functions `dwt_kernel_pwl2`, `idwt_kernel_pwl2` which perform these steps are included in the code base (2 stands for 2 vanishing moments).

Example 5.35: DWT on sound

Using the new kernels, let us also here listen to the low resolution approximations and the detail. First the low-resolution approximation:

```
x, fs = forw_comp_rev_DWT(m, 'pwl2')
play(x, fs)
```

The new, undesired effect in the castanets from Example 5.28 now seem to be gone. The detail for $m = 1$ this can be played as follows

```
x, fs = forw_comp_rev_DWT(1, 'pwl2', 0)
play(x, fs)
```

For $m = 2$ the detail can be played as follows

```
x, fs = forw_comp_rev_DWT(2, 'pwl2', 0)
play(x, fs)
```

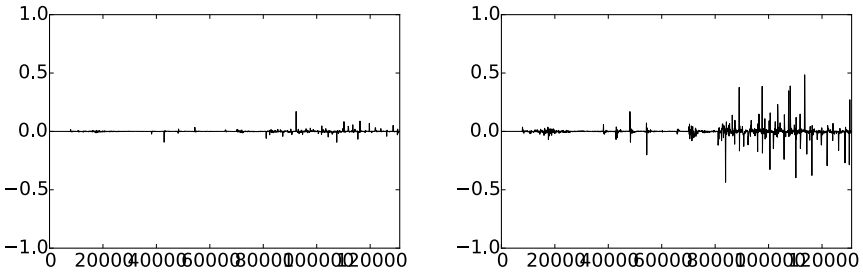

Figure 5.19: The detail in our audio sample file for the alternative piecewise linear wavelet, for $m = 1$ (left) and $m = 2$ (right).

The errors are shown in Figure 5.19. Again, when comparing with Example 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 piecewise linear wavelet. A partial explanation is that the wavelet we now have constructed has two vanishing moments, while the other had not.

Example 5.36: DWT on the samples of a mathematical function

Let us also repeat Exercise 5.11 for our alternative wavelet, where we plotted the detail/error at different resolutions, for the samples of a mathematical function.

Figure 5.20 shows the new plot. Again for the square wave there is an error, which seems to be slightly lower than for the previous 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 linear. 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 fact that this new wavelet has a bigger number of vanishing moments.

Exercise 5.37: Implement elementary lifting steps of even type

Write a function

```
lifting_even_symm(lambda, x, bd_mode)
```

which applies an elementary lifting matrix of even type (Equation (5.44)) to $\boldsymbol{x}$. As before, assume that N is even, and that the parameter `bd_mode` does nothing.

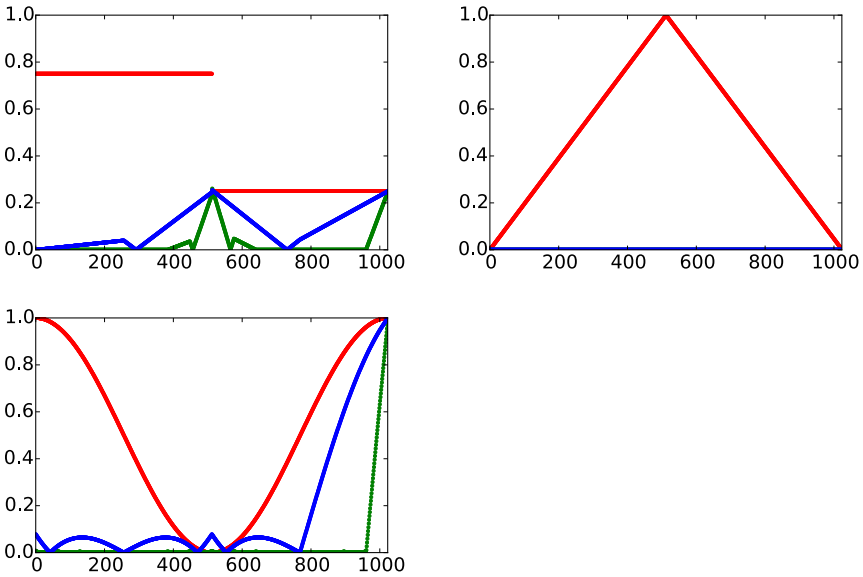

Figure 5.20: The error (i.e. the contribution from $W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}$) for $N = 1025$ when f is a square wave, the linear function $f(t) = 1 - 2|1/2 - t/N|$, and the trigonometric function $f(t) = 1/2 + \cos(2\pi t/N)/2$, respectivey. The detail is indicated for $m = 6$ and $m = 8$.

Exercise 5.38: Two vanishing moments

In this exercise we will show that there is a unique function on the form given by Equation (5.38) which has two vanishing moments.

a) Show that, when $\hat{\psi}$ is defined by Equation (5.38), we have that

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

b) Show that

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

c) Explain why there is a unique function on the form given by Equation (5.38) which has two vanishing moments, and that this function is given by Equation (5.40).

Exercise 5.39: Implement finding ψ with vanishing moments

In the previous exercise we ended up with a lot of calculations to find α, β in Equation (5.38). Let us try to make a program which does this for us, and which also makes us able to generalize the result.

a) Define

$$a_k = \int_{-1}^{1} t^k(1-|t|)dt, \quad b_k = \int_{0}^{2} t^k(1-|t-1|)dt, \quad e_k = \int_{0}^{1} t^k(1-2|t-1/2|)dt,$$

for $k \geq 0$. Explain why finding α, β so that we have two vanishing moments in Equation (5.38) 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. you can integrate functions in Python with the function `quad` in the package `scipy.integrate` . As an example, the function $\phi(t)$, which is nonzero only on $[-1, 1]$, can be integrated as follows:

```
res, err = quad(lambda 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

$$\hat{\psi} = \psi_{0,0} - \alpha\phi_{0,0} - \beta\phi_{0,1} - \gamma\phi_{0,-1} - \delta\phi_{0,2}. \tag{5.47}$$

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

$$g_k = \int_{-2}^{0} t^k(1-|t+1|)dt, \qquad d_k = \int_{1}^{3} t^k(1-|t-2|)dt$$

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 your computer.

c) Plot the function defined by (5.47), 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 $(\boldsymbol{\phi}_0, \boldsymbol{\psi}_0)$ is

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

Hint. 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.

Exercise 5.40: ψ for the Haar wavelet with two vanishing moments

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

a) Compute $\text{proj}_{V_0}(f(t))$, 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 function $\hat{\psi}$.

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 $\boldsymbol{\phi}_1$.

Exercise 5.41: More vanishing moments for the Haar wavelet

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

$$\begin{pmatrix} c_{0,0} & c_{0,1} & \cdots & c_{0,k-1} \\ c_{1,0} & c_{1,1} & \cdots & c_{1,k-1} \\ \vdots & \vdots & \vdots & \vdots \\ c_{k-1,0} & c_{k-1,1} & \cdots & c_{k-1,k-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{k-1} \end{pmatrix} = \begin{pmatrix} e_0 \\ e_1 \\ \vdots \\ e_{k-1} \end{pmatrix} \tag{5.48}$$

b) Write a function `vanishingmomshaar` which takes k as input, solves Equation (5.48), and returns the vector $\boldsymbol{a} = (a_0, a_1, \ldots, a_{k-1})$.

Exercise 5.42: Listening experiments

Run the function `forw_comp_rev_DWT` for different m for the Haar wavelet, the piecewise linear wavelet, and the alternative piecewise linear wavelet, but listen to the detail components $W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}$ instead. Describe the sounds you hear for different m, and try to explain why the sound seems to get louder when you increase m.

5.6 Multiresolution analysis: A generalization

Let us summarize the properties of the spaces V_m. In both our examples 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, 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 functions we have considered fit into the following general framework.

Definition 5.29. *Multiresolution analysis.*

A Multiresolution analysis, or MRA, is a nested sequence of function spaces

$$V_0 \subset V_1 \subset V_2 \subset \cdots \subset V_m \subset \cdots , \tag{5.49}$$

called resolution spaces, so that

- Any function can be approximated arbitrarily well from V_m, as long as m is large enough,
- $f(t) \in V_0$ if and only if $f(2^m t) \in V_m$,
- $f(t) \in V_0$ if and only if $f(t-n) \in V_0$ for all n.
- There is a function ϕ, called a scaling function, so that $\boldsymbol{\phi} = \{\phi(t-n)\}_{0 \le n < N}$ is a basis for V_0.

When $\boldsymbol{\phi}$ is an orthonormal basis we say that the MRA is *orthonormal*.

The wavelet of piecewise constant functions was an orthonormal MRA, while the wavelets for piecewise linear functions were not. Although the definition above states that any function can be approximated with MRA's, in practice one needs to restrict to certain functions: Certain pathological functions may be difficult to approximate. In the literature one typically requires that the function is in $L^2(\mathbb{R})$, and also that the scaling function and the spaces V_m are in $L^2(\mathbb{R})$. 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 examples we have considered we also chose a mother wavelet. 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. This was necessary in order to efficiently decompose the $g_m \in V_m$ into a low resolution approximation $g_{m-1} \in V_{m-1}$, and a detail/error e_{m-1} in a detail space we called W_{m-1}. We have freedom in how we define these detail spaces, as well as how we define a mother wavelet whose translates span the detail space (in general we choose a mother wavelet which simplifies the computation of 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). Once we agree on the detail spaces and the mother wavelet, we can perform a change of coordinates to find detail and low resolution approximations. We thus have the following general recipe.

Idea 5.30. *Recipe for constructing wavelets.*

In order to construct MRA's which are useful for practical purposes, we need to do the following:

- Find a function ϕ which can serve as the scaling function for an MRA,
- 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 Discrete Wavelet Transform is defined as the change of coordinates from $\boldsymbol{\phi}_1$ to $(\boldsymbol{\phi}_0, \boldsymbol{\psi}_0)$.

More generally, if

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

then the m-level DWT is defined by $\mathrm{DWT}(\boldsymbol{c}_m) = (\boldsymbol{c}_0, \boldsymbol{w}_0, \ldots, \boldsymbol{w}_{m-1})$. It is useful to interpret m as frequency, n as time, and $w_{m,n}$ as the contribution 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 frequency representations.

While there are in general many possible choices of detail spaces, in the case of an orthonormal wavelet we saw that it was natural to choose the detail space W_{m-1} as the orthogonal complement of V_{m-1} in V_m, and obtain the mother wavelet by projecting the scaling function onto the detail space. Thus, for orthonormal MRA's, the low-resolution approximation and the detail can be obtained by computing projections, and the least squares approximation of f from V_m can be computed as

$$\text{proj}_{V_m}(f) = \sum_n \langle f, \phi_{m,n} \rangle \phi_{m,n}(t).$$

Working with the samples of f rather than f itself: The first crime of wavelets. In Exercise 5.1 we saw that for the piecewise constant wavelet the coordinate vector of f in Φ_m equaled the sample vector of f. In Exercise 5.30 we saw that the same held for the piecewise linear wavelet. The general statement is false, however: The coordinate vector of f in Φ_0 may not equal the samples $(f(0), f(1), ...)$, so that $\sum_n f(n)\phi_{0,n}$ and f are two different functions.

In most applications, a function is usually only available through its samples. In many books on wavelets, one starts with these samples, and computes their DWT. This means that the underlying function is $\sum_n f(n)\phi_{0,n}$, and since this is different from f in general, we compute something completely different than we want. This shows that many books apply a wrong procedure when computing the DWT. This kind of error is also called *the first crime of wavelets*.

So, how bad is this crime? We will address this with two results. First we will see how the samples are related to the wavelet coefficients. Then we will see how the function $\sum_s f(s/2^m)\phi_{m,s}(t)$ is related to f (wavelet crime assumes equality).

Theorem 5.31. *Relation between samples and wavelet coefficients.*

Assume that $\tilde{\phi}$ has compact support and is absolutely integrable, i.e. $\int_0^N |\tilde{\phi}(t)|\,dt < \infty$. Assume also that f is continuous and has wavelet coefficients $c_{m,n}$. Then we have that

$$\lim_{m\to\infty} 2^{m/2} c_{m,n2^{m-m'}} = f(n/2^{m'}) \int_0^N \tilde{\phi}(t)\,dt.$$

Proof. Since $\tilde{\phi}$ has compact support, $\tilde{\phi}_{m,n2^{m-m'}}$ will be supported on a small interval close to $n/2^{m'}$ for large m. Since f is continuous, given $\epsilon > 0$ we can choose m so large that $f = f(n/2^{m'}) + r(t)$, where $|r(t)| < \epsilon$ on this interval. But then

$$\begin{aligned} c_{m,n2^{m-m'}} &= \int_0^N f(t)\tilde{\phi}_{m,n2^{m-m'}}(t)\,dt \\ &= f(n/2^{m'})\int_0^N \tilde{\phi}_{m,n2^{m-m'}}(t)\,dt + \int_0^N r(t)\tilde{\phi}_{m,n2^{m-m'}}(t)\,dt \\ &\leq 2^{-m/2}f(n/2^{m'})\int_0^N \tilde{\phi}(t)\,dt + \epsilon\int_0^N |\tilde{\phi}_{m,n2^{m-m'}}(t)|\,dt \\ &= 2^{-m/2}f(n/2^{m'})\int_0^N \tilde{\phi}(t)\,dt + 2^{-m/2}\epsilon\int_0^N |\tilde{\phi}(t)|\,dt. \end{aligned}$$

From this it follows that $\lim_{m\to\infty} 2^{m/2}c_{m,n2^{m-m'}} = f(n/2^{m'})\int_0^N \tilde{\phi}(t)\,dt$, since ϵ was arbitrary, and $\tilde{\phi}(t)$ was assumed to be absolutely integrable. □

This result has an important application. It turns out that there is usually no way to find analytical expressions for the scaling function and the mother wavelet. Their coordinates in $(\boldsymbol{\phi}_0, \boldsymbol{\psi}_0)$ are simple, however, since there is only one non-zero coordinate:

- The coordinates of ϕ in $(\boldsymbol{\phi}_0, \boldsymbol{\psi}_0)$ is $(1, 0, ..., 0)$, where there are $2^m N - 1$ zeros.
- The coordinates of ψ in $(\boldsymbol{\phi}_0, \boldsymbol{\psi}_0)$ is $(0, ..., 0, 1, 0, ..., 0)$, where there are $2^{m-1}N$ zeros at the beginning.

If we know that ϕ and ψ are continuous, we can apply an m stage IDWT to these coordinates and use Theorem 5.31 to find arbitrarily good estimates to the samples $\phi(n/2^{m'})$, $\psi(n/2^{m'})$. The coordinates we find have to be scaled with $2^{m/2}$ in order for the values to be of comparable size. Also, this algorithm will miss the actual samples by a factor of $\int_0^N \tilde{\phi}(t)\,dt$. Nevertheless, the graphs will be similar. The algorithm is also called the cascade algorithm.

Definition 5.32. *The cascade algorithm.*

The *cascade algorithm* applies a change of coordinates for the functions ϕ, ψ from the basis $(\boldsymbol{\phi}_0, \boldsymbol{\psi}_0, \boldsymbol{\psi}_1 \ldots)$ to the basis $\boldsymbol{\phi}_m$, and uses the new coordinates as an an approximation to the function values of these functions.

Now for the second result.

Theorem 5.33. *Using the samples.*

If f is continuous and ϕ has compact support, for all t with a finite bit expansion (i.e. $t = n/2^{m'}$ for some integers n and m') we have that

$$\lim_{m\to\infty} 2^{-m/2} \sum_{s=0}^{2^m N-1} f(s/2^m)\phi_{m,s}(t) = f(t)\sum_s \phi(s).$$

This says that, up to the constant factor $c = \sum_n \phi(n)$, the functions $f_m \in V_m$ with coordinates $2^{-m/2}(f(0/2^m), f(1/2^m), ...)$ in Φ_m converge pointwise to f as $m \to \infty$ (even though the samples of f_m may not equal those of f).

Proof. With $t = n/2^{m'}$, for $m > m'$ we have that

$$\phi_{m,s}(t) = \phi_{m,s}(n2^{m-m'}/2^m) = 2^{m/2}\phi(2^m n 2^{m-m'}/2^m - s) = 2^{m/2}\phi(n2^{m-m'} - s).$$

We thus have that

$$\sum_{s=0}^{2^m N-1} 2^{-m/2} f(s/2^m)\phi_{m,s}(t) = \sum_{s=0}^{2^m N-1} f(s/2^m)\phi(n2^{m-m'} - s).$$

In the sum finitely many s close to $n2^{m-m'}$ contribute (due to finite support), and then $s/2^m \approx t$. Due to continuity of f this sum thus converges to $f(t)\sum_s \phi(s)$, and the proof is done. □

Let us see how we can implement the cascade algorithm. As input to the algorithm we must have the number of levels m, and the kernel to use for the IDWT. Also we need to know an interval $[a, b]$ so large that it contains the supports of ϕ, ψ (we will see later how we can compute this supports). Else we can not obtain complete plots of the functions. a and b thus also need to be input to the algorithm. We now set $N = b - a$. The space $\boldsymbol{\phi}_m$ is then of dimension $(b-a)2^m$, so the cascade algorithm needs a coordinate vector of this size as starting point. the coordinates of ϕ in the $(b-a)2^m$-dimensional basis $(\boldsymbol{\phi}_0, \boldsymbol{\psi}_0, \boldsymbol{\psi}_1 \ldots)$ is $(1, 0, ..., 0)$ while the coordinates of ψ in the same basis is

$$(\underbrace{0, ..., 0}_{b-a \text{ times}}, 1, 0, ..., 0).$$

Our algorithm can take as input whether we want to plot the ϕ or the ψ function (and thereby choose among these sets of coordinates), and also the value of the dual parameter, which we will return to. The following algorithm can be used for all this

```
def cascade_alg(m, a, b, wave_name, scaling, dual):
    coords = zeros((b-a)*2**m)
    if scaling:
        coords[0] = 1
    else:
        coords[b - a] = 1

    t = linspace(a, b, (b-a)*2**m)
    IDWTImpl(coords, m, wave_name, 'per', dual)
    coords = concatenate([coords[(b*2**m):((b-a)*2**m)], \
                          coords[0:(b*2**m)]])
    plt.figure()
    plt.plot(t, 2**(m/2.)*coords, 'k-')
```

Example 5.43: Implementing the cascade algorithm

One thing should be noted in the function `cascade_alg`. As the scaling function of the piecewise linear wavelet, it may be that the function is nonzero for small, negative values. If we plot the function over $[0, N]$, we would see two disconnected segments - one to the left, and one to the right. In the code we shift the values so that the graph appears as one connected segment.

We will use $a = -2$ and $b = 6$ in what follows, since $[-2, 6]$ will turn out to contain all supports. We will also use $m = 10$ levels in the cascade algorithm. The following code then runs the cascade algorithm for the three wavelets we have considered, to reproduce all previous scaling functions and mother wavelets.

```
cascade_alg(10, -2, 6, 'Haar', True, False)
cascade_alg(10, -2, 6, 'Haar', False, False)
```

```
cascade_alg(10, -2, 6, 'pwl0', True, False)
cascade_alg(10, -2, 6, 'pwl0', False, False)
```

```
cascade_alg(10, -2, 6, 'pwl2', True, False)
cascade_alg(10, -2, 6, 'pwl2', False, False)
```

5.7 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 time-frequency representations of signals. We used an example based on Google Earth 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 another simply mounts to adding some detail information to the lower resolution version of the image. This also made wavelets useful for compression, since the images at different resolutions can serve as compressed versions of the image.

We defined the simplest wavelet, the Haar wavelet, which is a function approximation scheme based on piecewise constant functions, and deduced its properties. We defined the Discrete Wavelet Transform (DWT) as a change of coordinates corresponding to the function spaces we defined. This transform is the crucial object to study when it comes to more general wavelets also, since it is the object which makes wavelets useful for computation. In the following chapters, we will see that reordering of the source and target bases of the

DWT will aid in expressing connections between wavelets and filters, and in constructing optimized implementations of the DWT.

We then defined another wavelet, which corresponded to a function approximation scheme based on piecewise linear functions, instead of piecewise constant functions. 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 we now defined were 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 mentioned that we could define it so that it is more suitable for approximation of functions, by adding what we called vanishing moments.

From these examples of wavelets and their properties we made a generalization to what we called a multiresolution analysis (MRA). In an MRA we construct successively refined spaces of functions that may be used to approximate functions arbitrarily well. We will continue in the next chapter to construct even more general wavelets, within the MRA framework.

The book [29] goes through developments for wavelets in detail. While wavelets have been recognized for quite some time, it was with the important work of Daubechies [12, 13] that they found new arenas in the 80's. Since then they found important applications. The main application we will focus on in later chapters is image processing.

What you should have learned in this chapter.

- Definition of resolution spaces (V_m), detail spaces (W_m), scaling function (ϕ), and mother wavelet (ψ) for the wavelet based on piecewise constant functions.
- The nesting of resolution spaces, and how one can project from one resolution space onto a lower order resolution space, and onto its orthogonal complement.
- The definition of the Discrete Wavelet Transform as a change of coordinates, and how this can be written down from relations between basis functions.
- Definition of the m-level Discrete Wavelet Transform.
- Implementation of the Haar wavelet transform and its inverse.
- Experimentation with wavelets on sound.
- Definition of scaling function, mother wavelet, resolution spaces, and detail spaces for the wavelet of piecewise linear functions.
- How one alters the mother wavelet for piecewise linear functions, in order to add a vanishing moment.
- Definition of a multiresolution analysis.

Chapter 6

The filter representation of wavelets

Previously we saw 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 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 between the Fourier and wavelet settings, however:

- The DWT is not constructed by looking at the samples of a function, but rather by looking at coordinates in a given basis.
- The function spaces we work in (i.e. V_m) are different from the Fourier spaces.
- The DWT gave rise to two different types of analog filters: The filter defined by Equation (7.16) for obtaining $c_{m,n}$, and the filter defined by Equation (7.17) for obtaining $w_{m,n}$. We want both to correspond to digital filters.

Due to these differences, the way we realize the DWT in terms of filters will be a bit different. Despite the differences, this chapter will make it clear that the output of a DWT can be interpreted as the combined output of two different filters, and each filter will have an interpretation in terms of frequency representations. We will also see that the IDWT has a similar interpretation in terms of filters.

In this chapter we will also see that expressing the DWT in terms of filters will also enable us to define more general transforms, where even more filters are used. It is fruitful to think about each filter as concentrating on a particular frequency range, and that these transforms thus simply splits the input into different frequency bands. Such transforms have important applications to the

processing and compression of sound, and we will show that the much used MP3 standard for compression of sound takes use of such transforms.

6.1 The filters of a wavelet transformation

We will make the connection with digital filters by looking again at the different examples of wavelet bases we have seen: In each case we saw that every second row/column in the kernel transformations $G = P_{\boldsymbol{\phi}_m \leftarrow \mathcal{C}_m}$ and $H = P_{\mathcal{C}_m \leftarrow \boldsymbol{\phi}_m}$ repeated, as in a circulant 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 6.1. *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 6.2. *H_0 and H_1.*
We denote by H_0 the (unique) filter with the same first row as H, and by H_1 the (unique) filter with the same second row as H. H_0 and H_1 are also called the *DWT filter components*.

Using this definition it is clear that

$$(H\boldsymbol{c}_m)_k = \begin{cases} (H_0\boldsymbol{c}_m)_k & \text{when } k \text{ is even} \\ (H_1\boldsymbol{c}_m)_k & \text{when } k \text{ is odd,} \end{cases}$$

since the left hand side depends only on row k in the matrix H, 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 $H\boldsymbol{c}_m$ can be computed with the help of H_0 and H_1 as follows:

Theorem 6.3. *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:

- Compute $H_0\boldsymbol{c}_m$. The even-indexed entries in the result are the cordinates $\boldsymbol{c}_{m-1}$ in $\boldsymbol{\phi}_{m-1}$.
- 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. Keeping only every second coordinate is called *downsampling* (with a factor of two). Had we not performed downsampling, we would have ended up with twice as many coordinates as we started with. Downsampling with a factor of two means that we end up with the same number of samples as we started with. We also say that the output of the two filters is *critically sampled*. Due to the critical sampling, it is inefficient to compute the full application of the filters. We will return to the issue of making efficient implementations of critically sampled filter banks later.

We can now complement Figure 5.9 by giving names to the arrows as follows:

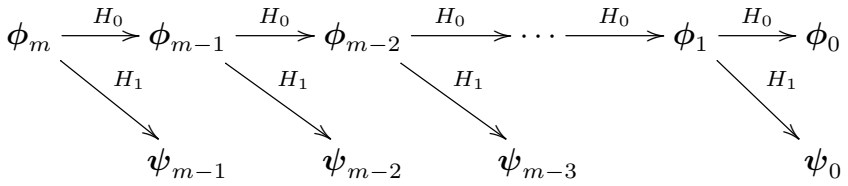

Figure 6.1: Detailed illustration of a wavelet transform.

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

Definition 6.4. *G_0 and G_1.*

We denote by G_0 the (unique) filter with the same first column as G, and by G_1 the (unique) filter with the same second column as G. 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

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

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 G 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 6.5. *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:

$$
\boldsymbol{c}_m = G_0\begin{pmatrix} c_{m-1,0} \\ 0 \\ c_{m-1,1} \\ 0 \\ \cdots \\ c_{m-1,2^{m-1}N-1} \\ 0 \end{pmatrix} + G_1\begin{pmatrix} 0 \\ w_{m-1,0} \\ 0 \\ w_{m-1,1} \\ \cdots \\ 0 \\ w_{m-1,2^{m-1}N-1} \end{pmatrix}. \tag{6.1}
$$

Making a new vector where zeroes have been inserted in this way is also called *upsampling* (with a factor of two). We can now also complement Figure 5.9 for the IDWT with named arrows. This has bee done in Figure 6.2

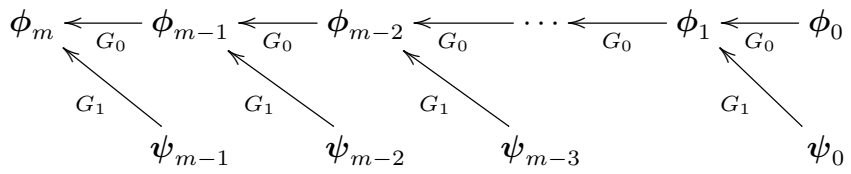

Figure 6.2: Detailed illustration of an IDWT.

Note that the filters G_0, G_1 were defined in terms of the columns of G, while the filters H_0, H_1 were defined in terms of the rows of H. 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. Let us summarize what we have found as follows.

Fact 6.6. *Computing DWT/IDWT through filters.*
The DWT can be computed with the help of two filters H_0, H_1, as explained in Theorem 6.3. Any linear transformation computed from two filters H_0, H_1 in this way is called a *forward filter bank transform*. The IDWT can be computed with the help of two filters G_0, G_1 as explained in Theorem 6.5. Any linear transformation computed from two filters G_0, G_1 in this way is called a *reverse filter bank transform*.

In Chapter 8 we will go through how any forward and reverse filter bank transform can be implemented, once we have the filters H_0, H_1, G_0, and G_1. When we are in a wavelet setting, the filter coefficients in these four filters can be found from the relations between the bases $\boldsymbol{\phi}_1$ and $(\boldsymbol{\phi}_0, \boldsymbol{\psi}_0)$. The filters H_0, H_1, G_0, G_1 can also be constructed from outside a wavelet setting, i.e. that they do not originate from change of coordinate matrices between certain function bases. The important point is that the matrices invert each other, but in a signal processing setting it may also be meaningful to allow for the reverse transform not to invert the forward transform exactly. This corresponds to some loss of information when we attempt to reconstruct the original signal using the reverse transform. A small such loss can, as we will see at the end of this chapter, be acceptable.

Note that Figure 6.1 and 6.2 do not indicate the additional downsampling and upsampling steps described in Theorem 6.3 and 6.5. If we indicate downsampling with $\downarrow_2$, and upsampling with $\uparrow_2$, the algorithms given in Theorem 6.3 and 6.5 can be summarized as in Figure 6.3.

Here $\oplus$ represents summing the elements which point inwards to the plus sign. In this figure, the left side represents the DWT, the right side the IDWT. In the literature, wavelet transforms are more often illustrated in this way using filters, since it makes alle steps involved in the process more clear. This type of figure also opens for generalization. We will shortly look into this.

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

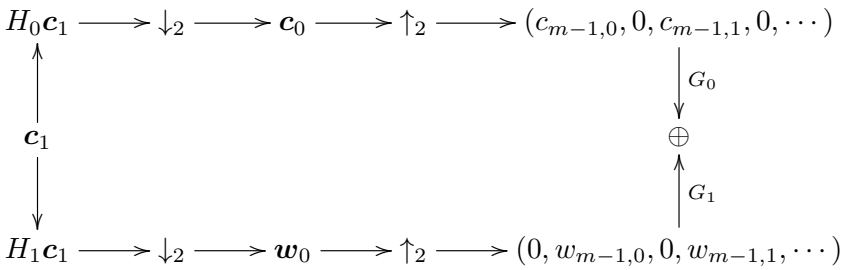

Figure 6.3: Detailed illustration of a DWT.

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

$$\phi(t) = \sum_{n=0}^{2N-1} (G_0)_{n,0}\phi_{1,n}(t) \tag{6.2}$$

$$\psi(t) = \sum_{n=0}^{2N-1} (G_1)_{n-1,1}\phi_{1,n}(t)., \tag{6.3}$$

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 Equation (6.2), the filter G_0 should have lowpass characteristics, since it extracts the information at lower frequencies. Similarly, G_1 should have highpass characteristics due to Equation (6.3).

Example 6.1: The Haar wavelet

For the Haar wavelet we saw that, in G, the matrix

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \tag{6.4}$$

repeated along the diagonal. The filters G_0 and G_1 can be found directly from these columns:

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

We have seen these filters previously: G_0 is a moving 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 high-pass 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 high-pass filter corresponding to the low-pass 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

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

By considering the filters where the rows are as in Equation (6.4), it is clear that

$$H_0 = \{1/\sqrt{2}, \underline{1/\sqrt{2}}\}$$
$$H_1 = \{\underline{-1/\sqrt{2}}, 1/\sqrt{2}\},$$

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

Example 6.2: Wavelet for piecewise linear functions

For the wavelet for piecewise linear functions we looked at in the previous section, Equation (5.34) gives that

$$G_0 = \frac{1}{\sqrt{2}}\{1/2, \underline{1}, 1/2\}$$
$$G_1 = \frac{1}{\sqrt{2}}\{\underline{1}\}. \tag{6.5}$$

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

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

$\lambda_{G_1}(\omega)$ thus has magnitude $\frac{1}{\sqrt{2}}$ at all points. Comparing with Figure 6.5 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 we have that

$$H_0 = \sqrt{2}\{\underline{1}\}$$
$$H_1 = \sqrt{2}\{-1/2, \underline{1}, -1/2\}. \tag{6.6}$$

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).

Example 6.3: The alternative piecewise linear wavelet

We previously wrote down the first two columns in $P_{\boldsymbol{\phi}_m \leftarrow \mathcal{C}_m}$ for the alternative piecewise linear wavelet. This gives us that the filters G_0 ans G_1 are

$$G_0 = \frac{1}{\sqrt{2}}\{1/2, \underline{1}, 1/2\}$$
$$G_1 = \frac{1}{\sqrt{2}}\{-1/8, -1/4, \underline{3/4}, -1/4, -1/8\}. \tag{6.7}$$

Here G_0 was as for the wavelet of piecewise linear functions since we use the same scaling function. G_1 was changed, however. 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.

Note that the role of H_1 as the high-pass 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.

6.1.1 The dual filter bank transform and the `dual` parameter

Since the reverse transform inverts the forward transform, $GH = I$. If we transpose this expression we get that $H^T G^T = I$. Clearly H^T is a reverse filter bank transform with filters $(H_0)^T, (H_1)^T$, and G^T is a forward filter bank transform with filters $(G_0)^T, (G_1)^T$. Due to their usefulness, these transforms have their own name:

Definition 6.7. *Dual filter bank transforms.*

Assume that H_0, H_1 are the filters of a forward filter bank transform, and that G_0, G_1 are the filters of a reverse filter bank transform. By the *dual transforms* we mean the forward filter bank transform with filters $(G_0)^T, (G_1)^T$, and the reverse filter bank transform with filters $(H_0)^T, (H_1)^T$.

In other words, if H and G are the kernel transformations of the DWT and the IDWT, respectively, the kernel transformations of the dual DWT and the dual IDWT are G^T and H^T, respectively. In Section 5.3 we used a parameter `dual` in our call to the DWT and IDWT kernel functions. This parameter can now be explained as follows:

Fact 6.8. *The `dual`-parameter in DWT kernel functions..*

- If the `dual` parameter is false, the DWT is computed as the forward filter bank transform with filters H_0, H_1, and the IDWT is computed as the reverse filter bank transform with filters G_0, G_1.
- If the `dual` parameter is true, the DWT is computed as the forward filter bank transform with filters $(G_0)^T, (G_1)^T$, and the IDWT is computed as the reverse filter bank transform with filters $(H_0)^T, (H_1)^T$.

This means that we can differ between the DWT, IDWT, and their duals as follows.

```
DWTImpl(x,  m, wave_name, True, False)  # DWT
IDWTImpl(x,  m, wave_name, True, False) # IDWT
DWTImpl(x,  m, wave_name, True, True)   # Dual DWT
IDWTImpl(x,  m, wave_name, True, True)  # Dual IDWT
```

Note that, even though the reverse filter bank transform G can be associated with certain function bases, it is not clear if the reverse filter bank transform H^T also can be associated with such bases. We will see in the next chapter that such bases can in many cases be found. We will also denote these bases as *dual bases*.

The construction of the dual wavelet transform was function-free - we have no reason to believe that they correspond to scaling functions and mother wavelets. In the next chapter we will show that such dual scaling functions and dual mother wavelets exist in many cases. We can set the `dual` parameter to `True` in

the implementation of the cascade algorithm in Example 5.43 to see how the functions must look. In Figure 6.4 we have plotted the result. 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 $\psi \approx \tilde{\psi}$.

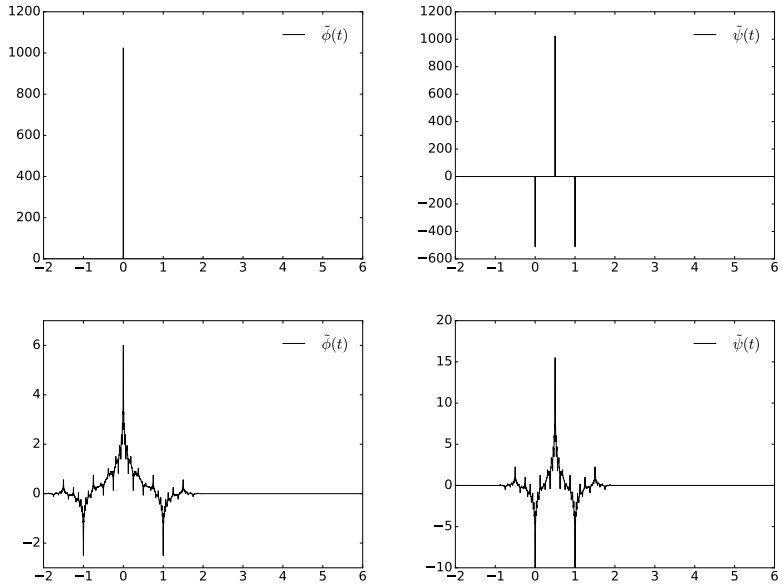

Figure 6.4: Dual functions for the two piecewise linear wavelets.

In the construction of the alternative piecewise linear wavelet we actually made a DWT/IDWT implementation before we found the filter coefficients themselves. But since the filter coefficients of G_0 and G_1 can be found in the columns of the matrix, they can be extracted by applying the IDWT kernel implementation to $\boldsymbol{e}_0$ (for the low-pass filter coefficients) and $\boldsymbol{e}_1$ (for the high-pass filter coefficients). Finally the frequency responses can by plotted by copying what we did in Section 3.3 The following algorithm does this.

```
def freqresp_alg(wave_name, lowpass, dual):
    idwt_kernel = find_kernel(wave_name, 0, dual, False);
    N = 128
    n = arange(0,N)
    omega = 2*pi*n/float(N)

    g = zeros(N)
    if lowpass:
        g[0] = 1
    else:
        g[1] = 1

    idwt_kernel(g, 'per')
```

```
    plt.figure()
    plt.plot(omega, abs(fft.fft(g)), 'k-')
```

If the parameter `dual` is set to `True`, the dual filters $(H_0)^T$ and $(H_1)^T$ are plotted instead. If the filters have real coefficients, $|\lambda_{H_i^T}(\omega)| = |\lambda_{H_i}(\omega)|$, so the correct frequency responses are shown.

Example 6.4: Plotting the frequency responses

In order to verify the low-pass/high-pass characteristics of G_0 and G_1, let us plot the frequency responses of the wavelets we have considered. To plot $\lambda_{G_0}(\omega)$ and $\lambda_{G_1}(\omega)$ for the Haar wavelet, we can write

```
freqresp_alg('Haar', True, False)
freqresp_alg('Haar', False, False)
```

To plot the same frequency response for the alternative piecewise linear wavelet, we can write

```
freqresp_alg('pwl2', True, False)
freqresp_alg('pwl2', False, False)
```

The resulting frequency responses are shown in Figure 6.5. Low-pass/high-pass characteristics are clearly seen here.

6.1.2 The support of the scaling function and the mother wavelet

The scaling functions and mother wavelets we encounter will turn out to always be functions with compact support. An interesting consequence of equations (6.2) and (6.3) is that we can find the size of these supports from the number of filter coefficients in G_0 and G_1. In the following we will say that the support of a filter is $[E, F]$ if $t_{-E}, ..., t_F$ are the only nonzero filter coefficients.

Theorem 6.9. *Support size.*
Assume that G_0 has support $[M_0, M_1]$, G_1 has support $[N_0, N_1]$. Then the support of ϕ is $[M_0, M_1]$, and the support of ψ is $[(M_0+N_0+1)/2, (M_1+N_1+1)/2]$.

Proof. Let $[m_0, m_1]$ be the support of ϕ. Then $\phi_{1,n}$ clearly has support $[(m_0 + n)/2, (m_1 + n)/2]$. In the equation

$$\phi(t) = \sum_{n=M_0}^{M_1} (G_0)_n \phi_{1,n},$$

the function with the leftmost support is ϕ_{1,M_0}, while the one with the rightmost one is ϕ_{1,M_1}. These supports are $[(m_0 + M_0)/2, (m_1 + M_1)/2]$. In order for the supports of the two sides to match we clearly must have

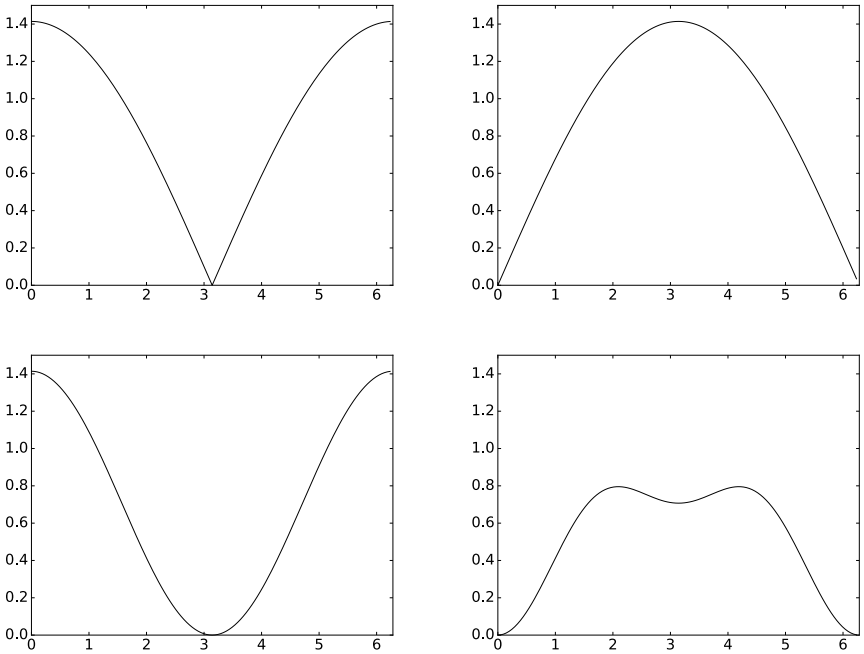

Figure 6.5: The frequency responses $\lambda_{G_0}(\omega)$ and $\lambda_{G_1}(\omega)$ for the Haar wavelet (top), and for the alternative piecewise linear wavelet (bottom).

$$m_0 = (m_0 + M_0)/2 \qquad\qquad m_1 = (m_1 + M_1)/2.$$

It follows that $m_0 = M_0$ and $m_1 = M_1$, so that the support of ϕ is $[M_0, M_1]$. Similarly, let $[n_0, n_1]$ be the support of ψ. We have that

$$\psi(t) = \sum_{n=N_0+1}^{N_1+1} (G_1)_{n-1}\phi_{1,n},$$

and we get in the same way

$$n_0 = (m_0 + N_0 + 1)/2 \qquad\qquad n_1 = (m_1 + N_1 + 1)/2.$$

It follows that $n_0 = (M_0 + N_0 + 1)/2$. $n_1 = (M_1 + N_1 + 1)/2$, so that the support of ψ is $((M_0 + N_0 + 1)/2, (M_1 + N_1 + 1)/2)$. □

There are two special cases of the above we will run into.

Wavelets with symmetric filters. The results then say the support of ϕ is $[-M_1, M_1]$ (i.e. symmetric around 0), and the support of ψ is $1/2 + [-(M_1 + N_1)/2, (M_1 + N_1)/2]$, i.e. symmetric around $1/2$. The wavelet with most such filter coefficients we will consider has 7 and 9 filter coefficients, respectively, so that the support of ϕ is $[-3, 3]$, and the support of ψ is $[-3, 4]$. This is why we have plotted these functions over $[-4, 4]$, so that the entire function can be seen. For the alternative piecewise linear wavelet the same argument gives that support of ϕ is $[-1, 1]$, and the support of ψ is $[-1, 2]$ (which we already knew from Figure 5.18). For the piecewise linear wavelet the support of ψ is deduced to be $[0, 1]$.

Orthonormal wavelets. For these wavelets it will turn that G_0 has filter coefficients evenly distributed around $1/2$, and G_1 has equally many, and evenly distributed around $-1/2$. It is straightforward to check that the filters for the Haar wavelet are of this kind, and this will turn out to be the simplest case of an orthonormal wavelet. For such supports Theorem 6.9 says that both supports are symmetric around $1/2$, and that both ϕ, ψ, G_0 and G_1 have the same support lengths. This can also be verified from the plots for the Haar wavelet. We will only consider orthonormal wavelets with at most 8 filter coefficients. This number of filter coefficients is easily seen to give the support $[-3, 4]$, which is why we have used $[-4, 4]$ as a common range when we plot functions on this form.

6.1.3 Symmetric extensions and the `bd_mode` parameter.

Continuous functions $f : [0, N] \to \mathbb{R}$ are approximated well from V_m, at least if the wavelet has one vanishing moment. The periodic extension of f is not continuous, however, if $f(0) \neq f(N)$. We can instead form the symmetric extension $\breve{f}$ of f, as given by Definition 1.22, which is defined on $[0, 2N]$, and which can be periodically extended to a continuous function. We make a smaller error if we restrict to the approximation of $\breve{f}$ from V_m, when compared to that of f from V_m.

The input to the DWT is given in terms of a vector $\boldsymbol{c}$, however, so that our f is really the function $\sum_n c_{m,n}\phi_{m,n}$. If ϕ has compact support, then clearly the function $\sum_n \breve{c}_{m,n}\phi_{m,n}$ retains the same values as the symmetric extension (except near the boundary), when $\breve{\boldsymbol{c}}$ is some kind of symmetric extension of the vector $\boldsymbol{c}$. We are free to decide how vectors are symmetrically extended near the boundary. In the theory of wavelets, the following symmetric extension strategy for vectors is used.

Definition 6.10. *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

$$\breve{\boldsymbol{x}}_k = \begin{cases} x_k & 0 \leq k < N \\ x_{2N-2-k} & N \leq k < 2N-3 \end{cases} \tag{6.8}$$

This is different from the symmetric extension given by Definition 4.1. Note that $(\breve{f}(0), \breve{f}(1), ..., \breve{f}(N-1), \breve{f}(N), \breve{f}(N+1), ..., \breve{f}(2N-1)) \in \mathbb{R}^{2N}$ is now the symmetric extension of $(f(0), f(1), ..., f(N))$, so that this way of defining symmetric extensions is perhaps the most natural when it comes to sampling which includes the boundaries.

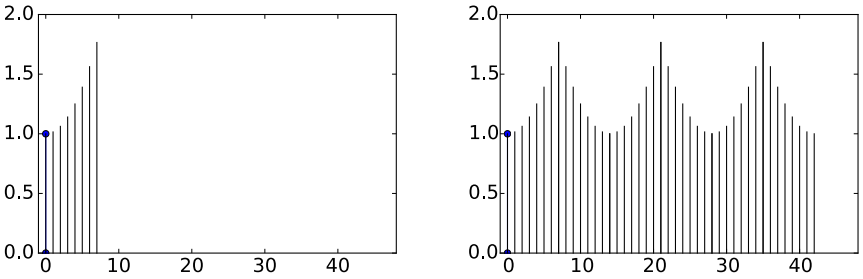

Figure 6.6: A vector and its symmetric extension. Note that the period of the vector is now $2N-2$, while it was $2N$ for the vector shown in Figure 4.1.

Consider applying the DWT to a symmetric extension $\breve{\boldsymbol{x}}$ of length $2N-2$. Assume that all filters are symmetric $(2N-2) \times (2N-2)$ filters. Accorins to Chapter 4, they preserve vectors symmetric around 0 and $N-1$, so that there exist $N \times N$-matrices $(H_0)_r$, $(H_1)_r$, $(G_0)_r$, and $(G_1)_r$, so that for all $\boldsymbol{x} \in \mathbb{R}^N$, $H_i \breve{\boldsymbol{x}} = \breve{(H_i)}_r \boldsymbol{x}$ and $G_i \breve{\boldsymbol{x}} = \breve{(G_i)}_r \boldsymbol{x}$. In particular, the first N entries in $H_i \breve{\boldsymbol{x}}$ are $(H_i)_r \boldsymbol{x}$. The first N entries of $H \breve{\boldsymbol{x}}$ are thus obtained by assembling the even-indexed entries from $(H_0)_r \boldsymbol{x}$, and the odd-indexed entries from $(H_1)_r \boldsymbol{x}$.

Since the difference between $N-1-k$ and $N-1+k$ is even, and since $H_i \breve{\boldsymbol{x}}$ is a symmetric extension, for one i we have that

$$(H\breve{\boldsymbol{x}})_{N-1-k} = (H_i\breve{\boldsymbol{x}})_{N-1-k} = (H_i\breve{\boldsymbol{x}})_{N-1+k} = (H\breve{\boldsymbol{x}})_{N-1+k}.$$

It follows that H preserves the same type of symmetric extensions, i.e. there exists an $N \times N$-matrix H_r so that $H\breve{\boldsymbol{x}} = \breve{H_r \boldsymbol{x}}$. Moreover, the entries in $H_r \boldsymbol{x}$ are assembled from the entries in $(H_i)_r \boldsymbol{x}$, in the same way as the entries in $H\boldsymbol{x}$ are assembled from the entries in $H_i \boldsymbol{x}$.

Note also that setting every second element to zero in a symmetric extension only creates a new symmetric extension, so that G also preserves symmetric extensions. It follows that there exist $N \times N$-matrices G_r, $(G_0)_r$, $(G_1)_r$ so that $G\breve{\boldsymbol{x}} = \breve{G_r \boldsymbol{x}}$, and so that the entries $0, ..., N-1$ in the output of G_r are obtained by combining $(G_0)_r$ and $(G_1)_r$ as in Theorem 6.5.

Theorem 6.11. *Symmetric filters and symmetric extensions.*

If the filters H_0, H_1, G_0, and G_1 in a wavelet transform are symmetric, then the DWT/IDWT preserve symmetric extensions (as defined in Definition 6.10). 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 $\boldsymbol{x} \in \mathbb{R}^N$ as described in theorems 6.3 and 6.5.

In our implementations, we factored H and G in terms of elementary lifting matrices. Note that the filters in these matrices are symmetric so that, after applying $A\breve{\boldsymbol{x}} = \breve{A_r\boldsymbol{x}}$ repeatedly,

$$\prod_i A_{\lambda_{2i}} B_{\lambda_{2i+1}} \breve{\boldsymbol{x}} = \prod_i \breve{(A_{\lambda_{2i}})_r (B_{\lambda_{2i+1}})_r \boldsymbol{x}}$$

It follows that $(\prod_i A_{\lambda_{2i}} B_{\lambda_{2i+1}})_r = \prod_i (A_{\lambda_{2i}})_r (B_{\lambda_{2i+1}})_r$. It is straightforward to find expressions for $(A_\lambda)_r$ and $(B_\lambda)_r$ (Exercise 6.7). DWT and IDWT implementations can thus apply symmetric extensions by replacing A_λ and B_λ with $(A_\lambda)_r$ and $(B_\lambda)_r$.

The purpose of the `bd_mode` parameter is to control how we should handle the boundary of the signal, in particular whether symmetric extensions should be performed. This parameter is passed to `lifting_even_symm` and `lifting_odd_symm`, which are the building blocks in our kernel functions. If this parameter equals 'symm' (this is the default value), the methods apply symmetric extensions by using $(A_\lambda)_r/(B_\lambda)_r$ rather than A_λ/B_λ. If the parameter equals 'per', a periodic extension at the boundaries is made.

Fact 6.12. *The `bd_mode`-parameter in DWT kernel functions.*

Assume that the filters H_0, H_1, G_0, and G_1 are symmetric. If the `bd_mode` parameter is `"symm"`, the symmetric versions $(H_0)_r$, $(H_1)_r$, $(G_0)_r$, and $(G_1)_r$ are applied in the DWT and IDWT, rather than the filters H_0, H_1, G_0, and G_1 themselves. If the 'parameter is `"per"`, the filters H_0, H_1, G_0, and G_1 are applied.

Exercise 6.5: Implement the dual filter bank transforms

a) Show that $A_\lambda^T = B_\lambda$ and $B_\lambda^T = A_\lambda$, i.e. that the transpose of an elementary lifting matrix of even/odd type is an elementary lifting matrix of odd/even type.

b) Let H be the kernel of the DWT, and assume that we have a factorization of it in terms of elementary lifting matrices. Use a) how that the dual DWT is obtained from the DWT by replacing each A_λ with $B_{-\lambda}$, and B_λ with $A_{-\lambda}$ in this factorization.

c) Previously we expressed the DWT and the IDWT of the piecewise linear wavelets in terms of elementary liftings. Use b) to write down the dual DWT and IDWT of these two wavelets in terms of lifting matrices. Verify your answer by going through the code in the functions `dwt_kernel_pwl0`, `idwt_kernel_pwl0`, `dwt_kernel_pwl2`, and `idwt_kernel_pwl2` where the `dual` parameter is set to true..

Exercise 6.6: Transpose of the DWT and IDWT

Explain why

- The transpose of the DWT can be computed with an IDWT with the kernel of the dual IDWT

- The transpose of the dual DWT can be computed with an IDWT with the kernel of the IDWT
- The transpose of the IDWT can be computed with a DWT with the kernel of the dual DWT
- The transpose of the dual IDWT can be computed with a DWT with the kernel of the DWT

Exercise 6.7: Reduced matrices for elementary lifting

Show that the reduced matrices for elementary lifting are

$$(A_\lambda)_r = \begin{pmatrix} 1 & 2\lambda & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \lambda & 1 & \lambda & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \lambda & 1 & \lambda \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix} \tag{6.9}$$

$$(B_\lambda)_r = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \lambda & 1 & \lambda & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 2\lambda & 1 \end{pmatrix}. \tag{6.10}$$

Also, change the implementations of `liftingstevensymm` and `liftingstoddsymm` so that these expressions are used (rather than A_λ, B_λ) when the parmeter `symm` is set to `True`.

Exercise 6.8: Prove expression for S_r

Show that, with

$$S = \begin{pmatrix} S_1 & S_2 \\ S_3 & S_4 \end{pmatrix} \in \mathbb{R}^{2N-2} \times \mathbb{R}^{2N-2}$$

a symmetric filter, with $S_1 \in \mathbb{R}^N \times \mathbb{R}^N$, $S_2 \in \mathbb{R}^N \times \mathbb{R}^{N-2}$, we have that

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

Use the proof of Theorem 4.9 as a guide.

Exercise 6.9: Orthonormal basis for the symmetric extensions

In this exercise we will establish an orthonormal basis for the symmetric extensions, as defined by Definition 6.10. This parallels Theorem 4.6.

a) Explain why, if $\boldsymbol{x} \in \mathbb{R}^{2N-2}$ is a symmetric extension (according to Definition 4.1), then $(\widehat{\boldsymbol{x}})_n = z_n e^{-\pi i n}$, where $\boldsymbol{z}$ is a real vectors which satisfies $z_n = z_{2N-2-n}$

b) Show that

$$\left\{ \boldsymbol{e}_0, \left\{ \frac{1}{\sqrt{2}}(\boldsymbol{e}_i + \boldsymbol{e}_{2N-2-i}) \right\}_{n=1}^{N-2}, \boldsymbol{e}_{N-1} \right\} \tag{6.11}$$

is an orthonormal basis for the vectors on the form $\widehat{\boldsymbol{x}}$ with $\boldsymbol{x} \in \mathbb{R}^{2N-2}$ a symmetric extension.

c) Show that

$$\begin{gathered} \frac{1}{\sqrt{2N-2}} \cos\left(2\pi \frac{0}{2N-2} k\right) \\ \left\{ \frac{1}{\sqrt{N-1}} \cos\left(2\pi \frac{n}{2N-2} k\right) \right\}_{n=1}^{N-2} \\ \frac{1}{\sqrt{2N-2}} \cos\left(2\pi \frac{N-1}{2N-2} k\right) \end{gathered} \tag{6.12}$$

is an orthonormal basis for the symmetric extensions in $\mathbb{R}^{2N-2}$.

d) Assume that S is symmetric. Show that the vectors listed in (6.12) are eigenvectors for S_r, when the vectors are viewed as vectors in $\mathbb{R}^N$, and that they are linearly independent. This shows that S_r is diagonalizable.

Exercise 6.10: Diagonalizing S_r

Let us explain how the matrix S_r can be diagonalized, similarly to how we previously diagonalized using the DCT. In Exercise 6.9 we showed that the vectors

$$\left\{ \cos\left(2\pi \frac{n}{2N-2} k\right) \right\}_{n=0}^{N-1} \tag{6.13}$$

in $\mathbb{R}^N$ is a basis of eigenvectors for S_r when S is symmetric. S_r itself is not symmetric, however, so that this basis can not possibly be orthogonal (S is symmetric if and only if it is orthogonally digonalizable). However, when the

vectors are viewed in $\mathbb{R}^{2N-2}$ we showed in Exercise 6.9c) an orthogonality statement which can be written as

$$\sum_{k=0}^{2N-3} \cos\left(2\pi\frac{n_1}{2N-2}k\right)\cos\left(2\pi\frac{n_2}{2N-2}k\right) = (N-1)\times\begin{cases}2 & \text{if } n_1=n_2\in\{0,N-1\}\\ 1 & \text{if } n_1=n_2\notin\{0,N-1\}\\ 0 & \text{if } n_1\neq n_2\end{cases}. \tag{6.14}$$

a) Show that

$$\begin{aligned}&(N-1)\times\begin{cases}1 & \text{if } n_1=n_2\in\{0,N-1\}\\ \frac{1}{2} & \text{if } n_1=n_2\notin\{0,N-1\}\\ 0 & \text{if } n_1\neq n_2\end{cases}\\ =&\frac{1}{\sqrt{2}}\cos\left(2\pi\frac{n_1}{2N-2}\cdot 0\right)\frac{1}{\sqrt{2}}\cos\left(2\pi\frac{n_2}{2N-2}\cdot 0\right)\\ &+\sum_{k=1}^{N-2}\cos\left(2\pi\frac{n_1}{2N-2}k\right)\cos\left(2\pi\frac{n_2}{2N-2}k\right)\\ &+\frac{1}{\sqrt{2}}\cos\left(2\pi\frac{n_1}{2N-2}(N-1)\right)\frac{1}{\sqrt{2}}\cos\left(2\pi\frac{n_2}{2N-2}(N-1)\right).\end{aligned}$$

Hint. Use that $\cos x=\cos(2\pi-x)$ to pair the summands k and $2N-2-k$.

Now, define the vector $\boldsymbol{d}_n^{(\mathrm{I})}$ as

$$d_{n,N}\left(\frac{1}{\sqrt{2}}\cos\left(2\pi\frac{n}{2N-2}\cdot 0\right),\left\{\cos\left(2\pi\frac{n}{2N-2}k\right)\right\}_{k=1}^{N-2},\frac{1}{\sqrt{2}}\cos\left(2\pi\frac{n}{2N-2}(N-1)\right)\right),$$

and define $d_{0,N}^{(\mathrm{I})}=d_{N-1,N}^{(\mathrm{I})}=1/\sqrt{N-1}$, and $d_{n,N}^{(\mathrm{I})}=\sqrt{2/(N-1)}$ when $n>1$. The orthogonal $N\times N$ matrix where the rows are $\boldsymbol{d}_n^{(\mathrm{I})}$ is called the DCT-I, and we will denote it by $D_N^{(\mathrm{I})}$. DCT-I is also much used, just as the DCT-II of Chapter 4. The main difference from the previous cosine vectors is that $2N$ has been replaced by $2N-2$.

b) Explain that the vectors $\boldsymbol{d}_n^{(\mathrm{I})}$ are orthonormal, and that the matrix

$$\sqrt{\frac{2}{N-1}}\begin{pmatrix}1/\sqrt{2} & 0 & \cdots & 0 & 0\\ 0 & 1 & \cdots & 0 & 0\\ \vdots & \vdots & \vdots & \vdots & \vdots\\ 0 & 0 & \cdots & 1 & 0\\ 0 & 0 & \cdots & 0 & 1/\sqrt{2}\end{pmatrix}\left(\cos\left(2\pi\tfrac{n}{2N-2}k\right)\right)\begin{pmatrix}1/\sqrt{2} & 0 & \cdots & 0 & 0\\ 0 & 1 & \cdots & 0 & 0\\ \vdots & \vdots & \vdots & \vdots & \vdots\\ 0 & 0 & \cdots & 1 & 0\\ 0 & 0 & \cdots & 0 & 1/\sqrt{2}\end{pmatrix}$$

is orthogonal.

c) Explain from b) that $\left(\cos\left(2\pi\frac{n}{2N-2}k\right)\right)^{-1}$ can be written as

$$\frac{2}{N-1}\begin{pmatrix}1/2 & 0 & \cdots & 0 & 0\\ 0 & 1 & \cdots & 0 & 0\\ \vdots & \vdots & \vdots & \vdots & \vdots\\ 0 & 0 & \cdots & 1 & 0\\ 0 & 0 & \cdots & 0 & 1/2\end{pmatrix}\left(\cos\left(2\pi\frac{n}{2N-2}k\right)\right)\begin{pmatrix}1/2 & 0 & \cdots & 0 & 0\\ 0 & 1 & \cdots & 0 & 0\\ \vdots & \vdots & \vdots & \vdots & \vdots\\ 0 & 0 & \cdots & 1 & 0\\ 0 & 0 & \cdots & 0 & 1/2\end{pmatrix}$$

With the expression we found in c) S_r can now be diagonalized as

$$\left(\cos\left(2\pi\frac{n}{2N-2}k\right)\right)D\left(\cos\left(2\pi\frac{n}{2N-2}k\right)\right)^{-1}.$$

Exercise 6.11: Compute filters and frequency responses 1

Write down the corresponding filters G_0 og G_1 for Exercise 5.40. Plot their frequency responses, and characterize the filters as low-pass- or high-pass filters.

Exercise 6.12: Symmetry of MRA matrices vs. symmetry of filters 1

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

Exercise 6.13: Symmetry of MRA matrices vs. symmetry of filters 2

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.

Exercise 6.14: Finding H_0, H_1 from H

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

$$H=\begin{pmatrix}1/5 & 1/5 & 1/5 & 0 & 0 & 0 & \cdots & 0 & 1/5 & 1/5\\ -1/3 & 1/3 & -1/3 & 0 & 0 & 0 & \cdots & 0 & 0 & 0\\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 & 0 & \cdots & 0 & 0 & 0\\ 0 & 0 & -1/3 & 1/3 & -1/3 & 0 & \cdots 0 & 0 & & 0\\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots\end{pmatrix}$$

Write down the compact form for the corresponding filters H_0, H_1, and compute and plot the frequency responses. Are the filters symmetric?

Exercise 6.15: Finding G_0 ,G_1 from G

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

$$G = \begin{pmatrix} 1/2 & -1/4 & 0 & 0 & \cdots \\ 1/4 & 3/8 & 1/4 & 1/16 & \cdots \\ 0 & -1/4 & 1/2 & -1/4 & \cdots \\ 0 & 1/16 & 1/4 & 3/8 & \cdots \\ 0 & 0 & 0 & -1/4 & \cdots \\ 0 & 0 & 0 & 1/16 & \cdots \\ 0 & 0 & 0 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots \\ 1/4 & 1/16 & 0 & 0 & \cdots \end{pmatrix}$$

Write down the compact form for the filters G_0, G_1, and compute and plot the frequency responses. Are the filters symmetric?

Exercise 6.16: Finding H from H_0, H_1

Assume that $H_0 = \{1/16, 1/4, \underline{3/8}, 1/4, 1/16\}$, and $H_1 = \{-1/4, \underline{1/2}, -1/4\}$. Plot the frequency responses of H_0 and H_1, 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 $P_{\mathcal{C}_1 \leftarrow \boldsymbol{\phi}_1}$ for the wavelet corresponding to these filters.

Exercise 6.17: Finding G from G_0, G_1

Assume that $G_0 = \frac{1}{3}\{1, \underline{1}, 1\}$, and $G_1 = \frac{1}{5}\{1, -1, \underline{1}, -1, 1\}$. Plot the frequency responses of G_0 and G_1, 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 $P_{\boldsymbol{\phi}_1 \leftarrow \mathcal{C}_1}$ for the wavelet corresponding to these filters.

Exercise 6.18: Computing by hand

In Exercise 5.21 we computed the DWT of two very simple vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, using the Haar wavelet.

a) Compute $H_0\boldsymbol{x}_1$, $H_1\boldsymbol{x}_1$, $H_0\boldsymbol{x}_2$, and $H_1\boldsymbol{x}_2$, where H_0 and H_1 are the filters used by the Haar wavelet.

b) Compare the odd-indexed elements in $H_1\boldsymbol{x}_1$ with the odd-indexed elements in $H_1\boldsymbol{x}_2$. From this comparison, attempt to find an explanation to why the two vectors have very different detail components.

Exercise 6.19: Comment code

Suppose that we run the following algorithm on the sound represented by the vector `x`:

```
c = (x[0::2] + x[1::2])/sqrt(2)
w = (x[0::2] - x[1::2])/sqrt(2)

newx = concatenate([c, w])
newx /= abs(newx).max()
play(newx,44100)
```

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 the inverse operation. What will you hear if you play the new sound you then get?

Exercise 6.20: Computing filters and frequency responses

Let us return to the piecewise linear wavelet from Exercise 5.39.

a) With $\hat{\psi}$ as defined as in b) in Exercise 5.39, 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

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

which is the coordinate vector you computed in d) in Exercise 5.39. For this, you should use the function `IDWTImpl`, with the kernel of the piecewise linear wavelet without symmetric extension as input. Explain that this gives you the filter coefficients of G_1.

b) Plot the frequency response of G_1.

Exercise 6.21: Computing filters and frequency responses 2

Repeat the previous exercise for the Haar wavelet as in Exercise 5.41, and plot the corresponding frequency responses for $k = 2, 4, 6$.

Exercise 6.22: Implementing with symmetric extension

In Exercise 3.9 we implemented a symmetric filter applied to a vector, i.e. when a periodic extension is assumed. The corresponding function was called `filterS(t, x)`, and used the function `numpy.convolve`.

a) Reimplement the function `filterS` so that it also takes a third parameter `symm`. If `symm` is false a periodic extension of `x` should be performed (i.e. filtering as we have defined it, and as the previous version of `filterS` performs it). If `symm` is true, symmetric extensions should be used (as given by Definition 6.10).

b) Implement functions

```
dwt_kernel_filters(H0, H1, G0, G1, x, bd_mode)
idwt_kernel_filters(H0, H1, G0, G1, x, bd_mode)
```

which return the DWT and IDWT kernels using theorems 6.3 and 6.5, respectively. This function thus bases itself on that the filters of the wavelet are known. The functions should call the function `filterS` from a). Recall also the definition of the parameter `dual` from this section.

With the functions defined in b) you can now define standard DWT and IDWT kernels in the following way, once the filters are known.

```
dwt_kernel  = lambda x, bd_mode: dwt_kernel_filters(H0, H1, G0, G1, x, bd_mode)
idwt_kernel = lambda x, bd_mode: idwt_kernel_filters(H0, H1, G0, G1, x, bd_mode)
```

6.2 Properties of the filter bank transforms of a wavelet

We have now described the DWT/IDWT as linear transformations G, H so that $GH = I$, and where two filters G_0, G_1 characterize G, two filters H_0, H_1 characterize H. G and H are not Toeplitz matrices, however, so they are not filters. Since filters produce the same output frequency from an input frequency, we must have that G and H produce other (undesired) frequencies in the output than those that are present in the input. We will call this phenomenon *aliasing*. In order for $GH = I$, the undesired frequencies must cancel each other, so that we end up with what we started with. Thus, GH must have what we will refer to as *alias cancellation*. This is the same as saying that GH is a filter. In order for $GH = I$, alias cancellation is not enough: We also need that the amount at the given frequency is unchanged, i.e. that $GH\phi_n = \phi_n$ for any Fourier basis vector ϕ_n. We then say that we have *perfect reconstruction*. Perfect reconstruction is always the case for wavelets by construction, but in signal processing many interesting examples (G_0, G_1, H_0, H_1) exist, for which we do not have perfect reconstruction. Historically, forward and reverse filter bank transforms have been around long before they appeared in a wavelet context. Operations where $GH\phi_n = c_n\phi_n$ for all n may also be useful, in particular when c_n is close to 1 for all n. If c_n is real for all n, we say that we have *no phase distortion*. If we have no phase distortion, the output from GH has the same phase, even if we do not have perfect reconstruction. Such "near-perfect reconstruction systems" have also been around long before many perfect reconstruction wavelet systems were designed. In signal processing, these transforms also exist in more general variants, and we will define these later. Let us summarize as follows.

Definition 6.13. *Alias cancellation, phase distortion, and perfect reconstruction.*

We say that we have *alias cancellation* if, for any n,

$$GH\phi_n = c_n\phi_n,$$

for some constant c_n (i.e. GH is a filter). If all c_n are real, we say that we *no phase distortion*. If $GH = I$ (i.e. $c_n = 1$ for all n) we say that we have *perfect reconstruction*. If all c_n are close to 1, we say that we have *near-perfect reconstruction*.

In signal processing, one also says that we have perfect- or near-perfect reconstruction when GH equals E_d, or is close to E_d (i.e. the overall result is a delay). The reason why a delay occurs has to do with that the transforms are used in real-time processing, for which we may not be able to compute the output at a given time instance before we know some of the following samples. Clearly the delay is unproblematic, since one can still can reconstruct the input from the output. We will encounter a useful example of near-perfect reconstruction soon in the MP3 standard.

Let us now find a criterium for alias cancellation: When do we have that $GHe^{2\pi irk/N}$ is a multiplum of $e^{2\pi irk/N}$, for any r? We first remark that

$$H(e^{2\pi irk/N}) = \begin{cases} \lambda_{H_0,r} e^{2\pi irk/N} & k \text{ even} \\ \lambda_{H_1,r} e^{2\pi irk/N} & k \text{ odd.} \end{cases}$$

The frequency response of $H(e^{2\pi irk/N})$ is

$$\begin{aligned} &\sum_{k=0}^{N/2-1} \lambda_{H_0,r} e^{2\pi ir(2k)/N} e^{-2\pi i(2k)n/N} + \sum_{k=0}^{N/2-1} \lambda_{H_1,r} e^{2\pi ir(2k+1)/N} e^{-2\pi i(2k+1)n/N} \\ &= \sum_{k=0}^{N/2-1} \lambda_{H_0,r} e^{2\pi i(r-n)(2k)/N} + \sum_{k=0}^{N/2-1} \lambda_{H_1,r} e^{2\pi i(r-n)(2k+1)/N} \\ &= (\lambda_{H_0,r} + \lambda_{H_1,r} e^{2\pi i(r-n)/N}) \sum_{k=0}^{N/2-1} e^{2\pi i(r-n)k/(N/2)}. \end{aligned}$$

Clearly, $\sum_{k=0}^{N/2-1} e^{2\pi i(r-n)k/(N/2)} = N/2$ if $n = r$ or $n = r + N/2$, and 0 else. The frequency response is thus the vector

$$\frac{N}{2}(\lambda_{H_0,r} + \lambda_{H_1,r})\boldsymbol{e}_r + \frac{N}{2}(\lambda_{H_0,r} - \lambda_{H_1,r})\boldsymbol{e}_{r+N/2},$$

so that

$$H(e^{2\pi irk/N}) = \frac{1}{2}(\lambda_{H_0,r} + \lambda_{H_1,r})e^{2\pi irk/N} + \frac{1}{2}(\lambda_{H_0,r} - \lambda_{H_1,r})e^{2\pi i(r+N/2)k/N}. \tag{6.15}$$

Let us now turn to the reverse filter bank transform. We can write

$$\begin{aligned} (e^{2\pi ir\cdot 0/N}, 0, e^{2\pi ir\cdot 2/N}, 0, \ldots, e^{2\pi ir(N-2)/N}, 0) &= \frac{1}{2}(e^{2\pi irk/N} + e^{2\pi i(r+N/2)k/N}) \\ (0, e^{2\pi ir\cdot 1/N}, 0, e^{2\pi ir\cdot 3/N}, \ldots, 0, e^{2\pi ir(N-1)/N}) &= \frac{1}{2}(e^{2\pi irk/N} - e^{2\pi i(r+N/2)k/N}). \end{aligned}$$

This means that

$$\begin{aligned}
&G(e^{2\pi irk/N}) = G_0\left(\frac{1}{2}\left(e^{2\pi irk/N} + e^{2\pi i(r+N/2)k/N}\right)\right) + G_1\left(\frac{1}{2}\left(e^{2\pi irk/N} - e^{2\pi i(r+N/2)k/N}\right)\right)\\
=&\frac{1}{2}(\lambda_{G_0,r}e^{2\pi irk/N} + \lambda_{G_0,r+N/2}e^{2\pi i(r+N/2)k/N}) + \frac{1}{2}(\lambda_{G_1,r}e^{2\pi irk/N} - \lambda_{G_1,r+N/2}e^{2\pi i(r+N/2)k/N})\\
=&\frac{1}{2}(\lambda_{G_0,r} + \lambda_{G_1,r})e^{2\pi irk/N} + \frac{1}{2}(\lambda_{G_0,r+N/2} - \lambda_{G_1,r+N/2})e^{2\pi i(r+N/2)k/N}. \quad (6.16)
\end{aligned}$$

Now, if we combine equations (6.15) and (6.16), we get

$$\begin{aligned}
&GH(e^{2\pi irk/N})\\
&= \frac{1}{2}(\lambda_{H_0,r} + \lambda_{H_1,r})G(e^{2\pi irk/N}) + \frac{1}{2}(\lambda_{H_0,r} - \lambda_{H_1,r})G(e^{2\pi i(r+N/2)k/N})\\
&= \frac{1}{2}(\lambda_{H_0,r} + \lambda_{H_1,r})\left(\frac{1}{2}(\lambda_{G_0,r} + \lambda_{G_1,r})e^{2\pi irk/N} + \frac{1}{2}(\lambda_{G_0,r+N/2} - \lambda_{G_1,r+N/2})e^{2\pi i(r+N/2)k/N})\right)\\
&+ \frac{1}{2}(\lambda_{H_0,r} - \lambda_{H_1,r})\left(\frac{1}{2}(\lambda_{G_0,r+N/2} + \lambda_{G_1,r+N/2})e^{2\pi i(r+N/2)k/N} + \frac{1}{2}(\lambda_{G_0,r} - \lambda_{G_1,r})e^{2\pi irk/N})\right)\\
&= \frac{1}{4}\left((\lambda_{H_0,r} + \lambda_{H_1,r})(\lambda_{G_0,r} + \lambda_{G_1,r}) + (\lambda_{H_0,r} - \lambda_{H_1,r})(\lambda_{G_0,r} - \lambda_{G_1,r})\right)e^{2\pi irk/N}\\
&+ \frac{1}{4}\left((\lambda_{H_0,r} + \lambda_{H_1,r})(\lambda_{G_0,r+N/2} - \lambda_{G_1,r+N/2}) + (\lambda_{H_0,r} - \lambda_{H_1,r})(\lambda_{G_0,r+N/2} + \lambda_{G_1,r+N/2})\right)e^{2\pi i(r+N/2)k/N}\\
&= \frac{1}{2}(\lambda_{H_0,r}\lambda_{G_0,r} + \lambda_{H_1,r}\lambda_{G_1,r})e^{2\pi irk/N} + \frac{1}{2}(\lambda_{H_0,r}\lambda_{G_0,r+N/2} - \lambda_{H_1,r}\lambda_{G_1,r+N/2})e^{2\pi i(r+N/2)k/N}.
\end{aligned}$$

If we also replace with the continuous frequency response, we obtain the following:

Theorem 6.14. *Expression for aliasing.*
We have that

$$\begin{aligned}
GH(e^{2\pi irk/N}) =&\frac{1}{2}(\lambda_{H_0,r}\lambda_{G_0,r} + \lambda_{H_1,r}\lambda_{G_1,r})e^{2\pi irk/N}\\
&+ \frac{1}{2}(\lambda_{H_0,r}\lambda_{G_0,r+N/2} - \lambda_{H_1,r}\lambda_{G_1,r+N/2})e^{2\pi i(r+N/2)k/N}.
\end{aligned} \quad (6.17)$$

In particular, we have alias cancellation if and only if

$$\lambda_{H_0}(\omega)\lambda_{G_0}(\omega+\pi) = \lambda_{H_1}(\omega)\lambda_{G_1}(\omega+\pi). \quad (6.18)$$

We will refer to this as the *alias cancellation condition*. If in addition

$$\lambda_{H_0}(\omega)\lambda_{G_0}(\omega) + \lambda_{H_1}(\omega)\lambda_{G_1}(\omega) = 2, \quad (6.19)$$

we also have perfect reconstruction. We will refer to as the *condition for perfect reconstruction*.

No phase distortion means that we have alias cancellation, and that

$$\lambda_{H_0}(\omega)\lambda_{G_0}(\omega) + \lambda_{H_1}(\omega)\lambda_{G_1}(\omega) \text{ is real.}$$

Now let us turn to how we can construct wavelets/perfect reconstruction systems from FIR-filters (recall from Chapter 3 that FIR filters where filters with a finite number of filter coefficients). We will have use for some theorems which allow us to construct wavelets from prototype filters. In particular we show that, when G_0 and H_0 are given lowpass filters which satisfy a certain common property, we can define unique (up to a constant) highpass filters H_1 and G_1 so that the collection of these four filters can be used to implement a wavelet. We first state the following general theorem.

Theorem 6.15. *Criteria for perfect reconstruction.*
The following statements are equivalent for FIR filters H_0, H_1, G_0, G_1:

- H_0, H_1, G_0, G_1 give perfect reconstruction,
- there exist $\alpha \in \mathbb{R}$ and $d \in \mathbb{Z}$ so that

$$(H_1)_n = (-1)^n \alpha^{-1} (G_0)_{n-2d} \tag{6.20}$$
$$(G_1)_n = (-1)^n \alpha (H_0)_{n+2d} \tag{6.21}$$
$$2 = \lambda_{H_0,n}\lambda_{G_0,n} + \lambda_{H_0,n+N/2}\lambda_{G_0,n+N/2} \tag{6.22}$$

Let us translate this to continuous frequency responses. We first have that

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

We have a similar computation for $\lambda_{G_1}(\omega)$. We can thus state the following:

Theorem 6.16. *Criteria for perfect reconstruction.*
The following statements are equivalent for FIR filters H_0, H_1, G_0, G_1:

- H_0, H_1, G_0, G_1 give perfect reconstruction,
- there exist $\alpha \in \mathbb{R}$ and $d \in \mathbb{Z}$ so that

$$\lambda_{H_1}(\omega) = \alpha^{-1} e^{-2id\omega} \lambda_{G_0}(\omega + \pi) \tag{6.23}$$
$$\lambda_{G_1}(\omega) = \alpha e^{2id\omega} \lambda_{H_0}(\omega + \pi) \tag{6.24}$$
$$2 = \lambda_{H_0}(\omega)\lambda_{G_0}(\omega) + \lambda_{H_0}(\omega + \pi)\lambda_{G_0}(\omega + \pi) \tag{6.25}$$

Proof. Let us prove first that equations (6.23)- (6.25) for a FIR filter implies that we have perfect reconstruction. Equations (6.23)-(6.24) mean that the alias cancellation condition (6.18) is satisfied, since

$$\begin{aligned}\lambda_{H_1}(\omega)\lambda_{G_1}(\omega+\pi) =& \alpha^{-1}e^{-2id\omega}\lambda_{G_0}(\omega+\pi)(\alpha)e^{2id(\omega+\pi}\lambda_{H_0}(\omega)\\ =& \lambda_{H_0}(\omega)\lambda_{G_0}(\omega+\pi).\end{aligned}$$

Inserting this in the perfect reconstruction condition (6.25), we get

$$\begin{aligned}2 &= \lambda_{H_0}(\omega)\lambda_{G_0}(\omega)+\lambda_{G_0}(\omega+\pi)\lambda_{H_0}(\omega+\pi)\\ &= \lambda_{H_0}(\omega)\lambda_{G_0}(\omega)+\alpha^{-1}e^{-2id\omega}\lambda_{G_0}(\omega+\pi)\alpha e^{2id\omega}\lambda_{H_0}(\omega+\pi)\\ &= \lambda_{H_0}(\omega)\lambda_{G_0}(\omega)+\lambda_{H_1}(\omega)\lambda_{G_1}(\omega),\end{aligned}$$

which is Equation (6.19), so that equations (6.23)- (6.25) imply perfect reconstruction. We therefore only need to prove that any set of FIR filters which give perfect reconstruction, also satisfy these equations. Due to the calculation above, it is enough to prove that equations (6.23)-(6.24) are satisfied. The proof of this will wait till Section 8.1, since it uses some techniques we have not introduced yet. □

When constructing a wavelet 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 from above. If the filters are symmetric we just saw that $d = 0$. If G_0, G_1 are known, it follows from from equations (6.20) and(6.21) that

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

so that $\alpha = \sum_n (-1)^n(G_0)_n(G_1)_n$. On the other hand, if H_0, H_1 are known instead, we must have that

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

so that $\alpha = 1/(\sum_n (-1)^n(H_0)_n(H_1)_n)$. Let us use these observations to state the filters for the alternative wavelet of piecewise linear functions, which is the only wavelet we have gone through we have not computed the filters and the frequency response for.

Let us use Theorem 6.16 to compute the filters H_0 and H_1 for the alternative piecewise linear wavelet. These filters are also symmetric, since G_0, G_1 were. We get that

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

We now get

$$\begin{aligned}(H_0)_n &= \alpha^{-1}(-1)^n (G_1)_n = 2(-1)^n (G_1)_n \\ (H_1)_n &= \alpha^{-1}(-1)^n (G_0)_n = 2(-1)^n (G_0)_n,\end{aligned} \tag{6.26}$$

so that

$$\begin{aligned}H_0 &= \sqrt{2}\{-1/8, 1/4, \underline{3/4}, 1/4, -1/8\} \\ H_1 &= \sqrt{2}\{-1/2, \underline{1}, -1/2\}.\end{aligned} \tag{6.27}$$

Note that, even though conditions (6.23) and (6.24) together ensure that the alias cancellation condition is satisfied, alias cancellation can occur also if these conditions are not satisfied. Conditions (6.23) and (6.24) thus give a stronger requirement than alias cancellation. We will be particularly concerned with wavelets where the filters are symmetric, for which we can state the following corollary.

Corollary 6.17. *Criteria for perfect reconstruction* .
The following statements are equivalent:

- H_0, H_1, G_0, G_1 are the filters of a symmetric wavelet,
- $\lambda_{H_0}(\omega), \lambda_{H_1}(\omega), \lambda_{G_0}(\omega), \lambda_{G_1}(\omega)$ are real functions, and

$$\lambda_{H_1}(\omega) = \alpha^{-1}\lambda_{G_0}(\omega + \pi) \tag{6.28}$$
$$\lambda_{G_1}(\omega) = \alpha\lambda_{H_0}(\omega + \pi) \tag{6.29}$$
$$2 = \lambda_{H_0}(\omega)\lambda_{G_0}(\omega) + \lambda_{H_0}(\omega + \pi)\lambda_{G_0}(\omega + \pi). \tag{6.30}$$

Thw delay d is thus 0 for symmetric wavelets.

Proof. Since H_0 is symmetric, $(H_0)_n = (H_0)_{-n}$, and from equations (6.20) and (6.21) it follows that

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

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

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

which proves Equation (6.28). Equation (6.28) follows similarly. □

In the literature, two particular cases of filter banks have been important. They are both refered to as *Quadrature Mirror Filter banks*, or QMF filter banks, and some confusion exist between the two. Let us therefore make precise definitions of the two.

Definition 6.18. *Classical QMF filter banks.*

In the classical definition of a QMF filter banks it is required that $G_0 = H_0$ and $G_1 = H_1$ (i.e. the filters in the forward and reverse transforms are equal), and that

$$\lambda_{H_1}(\omega) = \lambda_{H_0}(\omega+\pi). \tag{6.31}$$

It is straightforward to check that, for a classical QMF filter bank, the forward and reverse transforms are equal (i.e. $G = H$). It is easily checked that conditions (6.23) and (6.24) are satisfied with $\alpha = 1, d = 0$ for a classical QMF filter bank. In particular, the alias cancellation condition is satisfied. The perfect recontruction condition can be written as

$$2 = \lambda_{H_0}(\omega)\lambda_{G_0}(\omega) + \lambda_{H_1}(\omega)\lambda_{G_1}(\omega) = \lambda_{H_0}(\omega)^2 + \lambda_{H_0}(\omega+\pi)^2. \tag{6.32}$$

Unfortunately, it is impossible to find non-trivial FIR-filters which satisfy this quadrature formula (Exercise 6.23). Therefore, classical QMF filter banks which give perfect reconstruction do not exist. Nevertheless, one can construct such filter banks which give close to perfect reconstruction [23], and this together with the fulfillment of the alias cancellation condition still make them useful. In fact, we will see in Section 8.3 that the MP3 standard take use of such filters, and this explains our previous observation that the MP3 standard does not give perfect reconstruction. Note, however, that if the filters in a classical QMF filter bank are symmetric (so that $\lambda_{H_0}(\omega)$ is real), we have no phase distortion.

The second type of QMF filter bank is defined as follows.

Definition 6.19. *Alternative QMF filter banks.*

In the alternative definition of a QMF filter bank it is required that $G_0 = (H_0)^T$ and $G_1 = (H_1)^T$ (i.e. the filter coefficients in the forward and reverse transforms are reverse of oneanother), and that

$$\lambda_{H_1}(\omega) = \overline{\lambda_{H_0}(\omega+\pi)}. \tag{6.33}$$

The perfect reconstruction condition for an alternative QMF filter bank can be written as

$$2 = \lambda_{H_0}(\omega)\lambda_{G_0}(\omega) + \lambda_{H_1}(\omega)\lambda_{G_1}(\omega) = \lambda_{H_0}(\omega)\overline{\lambda_{H_0}(\omega)} + \overline{\lambda_{H_0}(\omega+\pi)}\lambda_{H_0}(\omega+\pi)$$
$$= |\lambda_{H_0}(\omega)|^2 + |\lambda_{H_0}(\omega+\pi)|^2.$$

We see that the perfect reconstruction property of the two definitions of QMF filter banks only differ in that the latter take absolute values. It turns out that the latter also has many interesting solutions, as we will see in Chapter 7. If we in in condition (6.23) substitute $G_0 = (H_0)^T$ we get

$$\lambda_{H_1}(\omega) = \alpha^{-1}e^{-2id\omega}\lambda_{G_0}(\omega+\pi) = \alpha^{-1}e^{-2id\omega}\overline{\lambda_{H_0}(\omega+\pi)}.$$

If we set $\alpha = 1, d = 0$, we get equality here. A similar computation follows for Condition (6.24). In other words, also alternative QMF filter banks satisfy the alias cancellation condition. In the literature, a wavelet is called *orthonormal* if $G_0 = (H_0)^T, G_1 = (H_1)^T$. From our little computation it follows that alternative QMF filter banks with perfect reconstruction are examples of orthonormal wavelets, and correpond to orthonormal wavelets which satisfy $\alpha = 1, d = 0$.

For the Haar wavelet it is easily checked that $G_0 = (H_0)^T, G_1 = (H_1)^T$, but it does not satisfy the relation $\lambda_{H_1}(\omega) = \overline{\lambda_{H_0}(\omega+\pi)}$. Instead it satsifies the relation $\lambda_{H_1}(\omega) = -\overline{\lambda_{H_0}(\omega+\pi)}$. In other words, the Haar wavelet is not an alternative QMF filter bankthe way we have defined them. The difference lies only in a sign, however. This is the reason why the Haar wavelet is still listed as an alternative QMF filter bank in the literature. The additional sign leads to orthonormnal wavelets which satisfy $\alpha = -1, d = 0$ instead.

The following is clear for orthonormal wavelets.

Theorem 6.20. *Orthogonality og the DWT matrix.*

A DWT matrix is orthogonal (i.e. the IDWT equals the transpose of the DWT) if and only if the filters satisfy $G_0 = (H_0)^T, G_1 = (H_1)^T$, i.e. if and only if the MRA equals the dual MRA.

This can be proved simply by observing that, if we transpose the DWT-matrix, Theorem 6.23 says that we get an IDWT matrix with filters $(H_0)^T, (H_1)^T$, and this is equal to the IDWT if and only if $G_0 = (H_0)^T, G_1 = (H_1)^T$. It follows that QMF filter banks with perfect reconstruction give rise to orthonormal wavelets.

Exercise 6.23: Finding FIR filters

Show that it is impossible to find a non-trivial FIR-filter which satisfies Equation (6.32).

Exercise 6.24: The Haar wavelet as an alternative QMF filter bank

Show that the Haar wavelet satisfies $\lambda_{H_1}(\omega) = -\overline{\lambda_{H_0}(\omega + \pi)}$, and $G_0 = (H_0)^T$, $G_1 = (H_1)^T$. The Haar wavelet can thus be considered as an alternative QMF filter bank.

6.3 A generalization of the filter representation, and its use in audio coding

It turns out that the filter representation, which we now have used for an alternative representation of a wavelet transformation, can be generalized in such a way that it also is useful for audio coding. In this section we will first define this generalization. We will then state how the MP3 standard encodes and decodes audio, and see how our generalization is connected to this. Much literature fails to elaborate on this connection. We will call our generalizations *filter bank transforms*, or simply *filter banks*. Just as for wavelets, filters are applied differently for the forward and reverse transforms. The code for this section can be found in a module called `mp3funcs`.

We start by defining the forward filter bank transform and its filters.

Definition 6.21. *Forward filter bank transform.*

Let $H_0, H_1, \ldots, H_{M-1}$ be $N \times N$-filters. A *forward filter bank transform* H produces output $\boldsymbol{z} \in \mathbb{R}^N$ from the input $\boldsymbol{x} \in \mathbb{R}^N$ in the following way:

- $z_{iM} = (H_0\boldsymbol{x})_{iM}$ for any i so that $0 \leq iM < N$.
- $z_{iM+1} = (H_1\boldsymbol{x})_{iM+1}$ for any i so that $0 \leq iM + 1 < N$.
- ...
- $z_{iM+(M-1)} = (H_{M-1}\boldsymbol{x})_{iM+(M-1)}$ for any i so that $0 \leq iM + (M-1) < N$.

In other words, the output of a forward filter bank transform is computed by applying filters $H_0, H_1, \ldots, H_{M-1}$ to the input, and by downsampling and assembling these so that we obtain the same number of output samples as input samples (also in this more general setting this is called critical sampling). $H_0, H_1, \ldots, H_{M-1}$ are also called *analysis filter components*, the output of filter H_i is called channel i *channel*, and M is called the number of channels. The output samples z_{iM+k} are also called the *subband samples* of channel k.

Clearly this definition generalizes the DWT and its analysis filters, since these can be obtained by setting $M = 2$. The DWT is thus a 2-channel forward filter bank transform. While the DWT produces the output $\begin{pmatrix} \boldsymbol{c}_{m-1} \\ \boldsymbol{w}_{m-1} \end{pmatrix}$ from the input $\boldsymbol{c}_m$, an M-channel forward filter bank transform splits the output into M components, instead of 2. Clearly, in the matrix of a forward filter bank

transform the rows repeat cyclically with period M, similarly to MRA-matrices. In practice, the filters in a forward filter bank transform are chosen so that they concentrate on specific frequency ranges. This parallels what we saw for the filters of a wavelet, where one concentrated on high frequencies, one on low frequencies. Using a filter bank to split a signal into frequency components is also called *subband coding*. But the filters in a filter bank are usually not ideal bandpass filters. There exist a variety of different filter banks, for many different purposes [47, 39]. In Chapter 7 we will say more on how one can construct filter banks which can be used for subband coding.

Let us now turn to reverse filter bank transforms.

Definition 6.22. *Reverse filter bank transforms.*

Let $G_0, G_1, \ldots, G_{M-1}$ be $N \times N$-filters. An *reverse filter bank transform* G produces $\boldsymbol{x} \in \mathbb{R}^N$ from $\boldsymbol{z} \in \mathbb{R}^N$ in the following way:

Define $\boldsymbol{z}_k\mathbb{R}^N$ as the vector where $(\boldsymbol{z}_k)_{iM+k} = \boldsymbol{z}_{iM+k}$ for all i so that $0 \leq iM + k < N$, and $(\boldsymbol{z}_k)_s = 0$ for all other s.

$$\boldsymbol{x} = G_0\boldsymbol{z}_0 + G_1\boldsymbol{z}_1 + \ldots + G_{M-1}\boldsymbol{z}_{M-1}. \tag{6.34}$$

$G_0, G_1, \ldots, G_{M-1}$ are also called *synthesis filter components*.

Again, this generalizes the IDWT and its synthesis filters, and the IDWT can be seen as a 2-channel reverse filter bank transform. Also, in the matrix of a reverse filter bank transform, the columns repeat cyclically with period M, similarly to MRA-matrices. Also in this more general setting the filters G_i are in general different from the filters H_i. But we will see that, just as we saw for the Haar wavelet, there are important special cases where the analysis and synthesis filters are equal, and where their frequency responses are simply shifts of oneanother. It is clear that definitions 6.21 and 6.22 give the diagram for computing forward and reverse filter bank transforms shown in Figure 6.7:

Here $\downarrow_M$ and $\uparrow_M$ means that we extract every M'th element in the vector, and add $M-1$ zeros between the elements, respectively, similarly to how we previously defined $\downarrow_2$ and $\uparrow_2$. Comparing Figure 6.3 with Figure 6.7 makes the similarities between wavelet transformations and the transformation used in the MP3 standard very visible: Although the filters used are different, they are subject to the same kind of processing, and can therefore be subject to the same implementations.

In general it may be that the synthesis filters do not invert exactly the analysis filters. If the synthesis system exactly inverts the analysis system, we say that we have a *perfect reconstruction filter bank*. Since the analysis system introduces undesired frequencies in the different channels, these have to cancel in the inverse transform, in order to reconstruct the input exactly.

We will have use for the following simple connection between forward and reverse filter bank transforms, which follows imemdiately from the definitions.

Theorem 6.23. *Connection between forward and reverse filter bank transforms.*

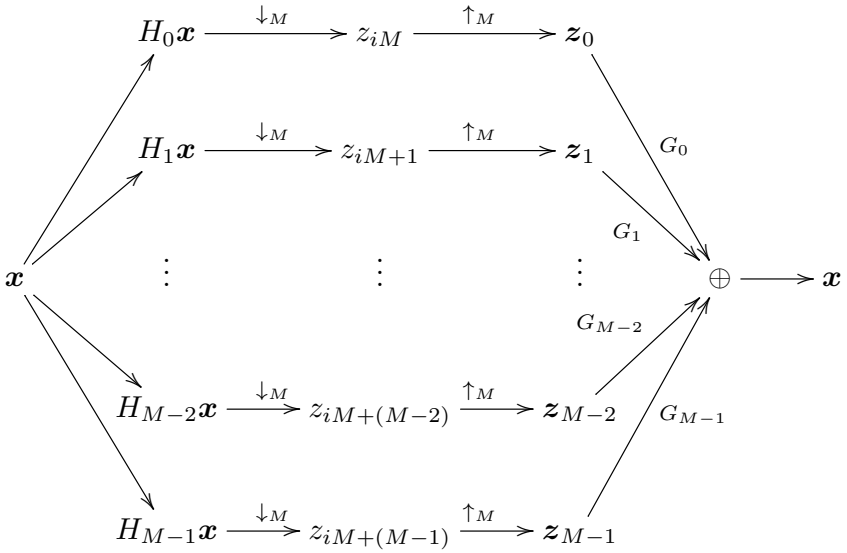

Figure 6.7: Illustration of forward and reverse filter bank transforms.

Assume that H is a forward filter bank transform with filters $H_0, \ldots, H_{N-1}$. Then H^T is a reverse filter bank transform with filters $G_0 = (H_0)^T, \ldots, G_{N-1} = (H_{N-1})^T$.

6.3.1 Forward filter bank transform in the MP3 standard

Now, let us turn to the MP3 standard. The MP3 standard document states that it applies a filter bank, and explains the following procedure for applying this filter bank, see p. 67 of the standard document (the procedure is slightly modified with mathematical terminology adapted to this book):

- Input 32 audio samples at a time.
- Build an input sample vector $X \in \mathbb{R}^{512}$, where the 32 new samples are placed first, all other samples are delayed with 32 elements. In particular the 32 last samples are taken out.
- Multiply X componentwise with a vector C (this vector is defined through a table in the standard), to obtain a vector $Z \in \mathbb{R}^{512}$. The standard calls this *windowing*.
- Compute the vector $Y \in \mathbb{R}^{64}$ where $Y_i = \sum_{j=0}^{7} Z_{i+64j}$. The standard calls this a *partical calculation*.

- Calculate $S = MY \in \mathbb{R}^{32}$, where M is the 32×64- matrix where $M_{ik} = \cos((2i+1)(k-16)\pi/64)$. S is called the vector of output samples, or output subband samples. The standard calls this *matrixing*.

The standard does not motivate these steps, and does not put them into the filter bank transform framework which we have established. Also, the standard does not explain how the values in the vector C have been constructed.

Let us start by proving that the steps above really corresponds to applying a forward filter bank transform, and let us state the corresponding filters of this transform. The procedure computes 32 outputs in each iteration, and each of them is associated with a subband. Therefore, from the standard we would guess that we have $M = 32$ channels, and we would like to find the corresponding 32 filters $H_0, H_1, \ldots, H_{31}$.

It may seem strange to use the name matrixing here, for something which obviously is matrix multiplication. The reason for this name must be that the at the origin of the procedure come from outside a linear algebra framework. The name windowing is a bit strange, too. This really does not correspond to applying a window to the sound samples as we explained in Section 3.3.1. We will see that it rather corresponds to applying a filter coefficient to a sound sample. A third and final thing which seems a bit strange is that the order of the input samples is reversed, since we are used to having the first sound samples in time with lowest index. This is perhaps more usual to do in an engineering context, and not so usual in a mathematical context. FIFO.

Clearly, the procedure above defines a linear transformation, and we need to show that this linear transformation coincides with the procedure we defined for a forward filter bank transform, for a set of 32 filters. The input to the transformation are the audio samples, which we will denote by a vector $\boldsymbol{x}$. At iteration s of the procedure above the input audio samples are $x_{32s-512}, x_{32s-510}, \ldots, x_{32s-1}$, and $X_i = x_{32s-i-1}$ due to the reversal of the input samples. The output to the transformation at iteration s of the procedure are the $S_0, \ldots, S_{31}$. We assemble these into a vector $\boldsymbol{z}$, so that the output at iteration s are $z_{32(s-1)} = S_0$, $z_{32(s-1)+1} = S_1, \ldots, z_{32(s-1)+31} = S_{31}$.

We will have use for the following cosine-properties, which are easily verified:

$$\cos\left(2\pi(n+1/2)(k+2Nr)/(2N)\right) = (-1)^r \cos\left(2\pi(n+1/2)k/(2N)\right) \tag{6.35}$$

$$\cos\left(2\pi(n+1/2)(2N-k)/(2N)\right) = -\cos\left(2\pi(n+1/2)k/(2N)\right). \tag{6.36}$$

With the terminology above and using Property (6.35) the transformation can be written as

$$\begin{aligned}
z_{32(s-1)+n} &= \sum_{k=0}^{63} \cos((2n+1)(k-16)\pi/64) Y_k = \sum_{k=0}^{63} \cos((2n+1)(k-16)\pi/64) \sum_{j=0}^{7} Z_{k+64j} \\
&= \sum_{k=0}^{63} \sum_{j=0}^{7} (-1)^j \cos((2n+1)(k+64j-16)\pi/64) Z_{k+64j} \\
&= \sum_{k=0}^{63} \sum_{j=0}^{7} \cos((2n+1)(k+64j-16)\pi/64)(-1)^j C_{k+64j} X_{k+64j} \\
&= \sum_{k=0}^{63} \sum_{j=0}^{7} \cos((2n+1)(k+64j-16)\pi/64)(-1)^j C_{k+64j} x_{32s-(k+64j)-1}.
\end{aligned}$$

Now, if we define $\{h_r\}_{r=0}^{511}$ by $h_{k+64j} = (-1)^j C_{k+64j}$, $0 \leq j < 8, 0 \leq k < 64$, and $h^{(n)}$ as the filter with coefficients $\{\cos((2n+1)(k-16)\pi/64) h_k\}_{k=0}^{511}$, the above can be simplified as

$$\begin{aligned}
z_{32(s-1)+n} &= \sum_{k=0}^{511} \cos((2n+1)(k-16)\pi/64) h_k x_{32s-k-1} = \sum_{k=0}^{511} (h^{(n)})_k x_{32s-k-1} \\
&= (h^{(n)} \boldsymbol{x})_{32s-1} = (E_{n-31} h^{(n)} \boldsymbol{x})_{32(s-1)+n}.
\end{aligned}$$

This means that the output of the procedure stated in the MP3 standard can be computed as a forward filter bank transform, and that we can choose the analysis filters as $H_n = E_{n-31} h^{(n)}$.

Theorem 6.24. *Forward filter bank transform for the MP3 standard.*

Define $\{h_r\}_{r=0}^{511}$ by $h_{k+64j} = (-1)^j C_{k+64j}$, $0 \leq j < 8, 0 \leq k < 64$, and $h^{(n)}$ as the filter with coefficients $\{\cos((2n+1)(k-16)\pi/64) h_k\}_{k=0}^{511}$. If we define $H_n = E_{n-31} h^{(n)}$, the procedure stated in the MP3 standard corresponds to applying the corresponding forward filter bank transform.

The filters H_n were shown in Example 3.34 as examples of filters which concentrate on specific frequency ranges. The h_k are the filter coefficients of what is called a prototype filter. This kind of filter bank is also called a *cosine-modulated filter*. The multiplication with $\cos\left(2\pi(n+1/2)(k-16)/(2N)\right) h_k$, modulated the filter coefficients so that the new filter has a frequency response which is simply shifted in frequency in a symmetric manner: In Exercise 3.44, we saw that, by multiplying with a cosine, we could contruct new filters with real filter coefficients, which also corresponded to shifting a prototype filter in frequency. Of course, multiplication with a complex exponential would also shift the frequency response (such filter banks are called *DFT-modulated filter banks*), but the problem with this is that the new filter has complex coefficients: It will turn out that cosine-modulated filter banks can also be constructed so that they are invertible, and that one can find such filter banks where the inverse is easily found.

The effect of the delay in the definition of H_n is that, for each n, the multiplications with the vector $\boldsymbol{x}$ are "aligned", so that we can save a lot of multiplications by performing this multiplication first, and summing these. We actually save even more multiplications in the sum where j goes from 0 to 7, since we here multiply with the same cosines. The steps defined in the MP3 standard are clearly motivated by the desire to reduce the number of multiplications due to these facts. A simple arithmetic count illutrates these savings: For every 32 output samples, we have the following number of multiplications:

- The first step computes 512 multiplications.
- The second step computes 64 sums of 8 elements each, i.e. a total of $7 \times 64 = 448$ additions (note that $q = 512/64 = 8$).

The standard says nothing about how the matrix multiplication in the third step can be implemented. A direct multiplication would yield $32 \times 64 = 2048$ multiplications, leaving a total number of multiplications at 2560. In a direct implementation of the forward filter bank transform, the computation of 32 samples would need $32 \times 512 = 16384$ multiplications, so that the procedure sketched in the standard gives a big reduction.

The standard does not mention all possibilities for saving multiplications, however: We can reduce the number of multiplications even further, since clearly a DCT-type implementation can be used for the matrixing operation. We already have an efficient implementation for multiplication with a 32×32 type-III cosine matrix (this is simply the IDCT). We have seen that this implementation can be chosen to reduce the number of multiplications to $N \log_2 N/2 = 80$, so that the total number of multiplications is $512 + 80 = 592$. Clearly then, when we use the DCT, the first step is the computationally most intensive part.

6.3.2 Reverse filter bank transform in the MP3 standard

Let us now turn to how decoding is specified in the MP3 standard, and see that we can associate this with a reverse filter bank transform. The MP3 standard also states the following procedure for decoding:

- Input 32 new subband samples as the vector S.
- Change vector $V \in \mathbb{R}^{512}$, so that all elements are delayed with 64 elements. In particular the 64 last elements are taken out.
- Set the first 64 elements of V as $NS \in \mathbb{R}^{64}$, where N is the 64×32-matrix where $N_{ik} = \cos((16 + i)(2k + 1)\pi/64)$. The standard also calls this matrixing.
- Build the vector $U \in \mathbb{R}^{512}$ from V from the formulas $U_{64i+j} = V_{128i+j}$, $U_{64i+32+j} = V_{128i+96+j}$ for $0 \leq i \leq 7$ and $0 \leq j \leq 31$, i.e. U is the vector where V is first split into segments of length 132, and U is constructed by assembling the first and last 32 elements of each of these segments.

- Multiply U componentwise with a vector D (this vector is defined in the standard), to obtain a vector $W \in \mathbb{R}^{512}$. The standard also calls this windowing.
- Compute the 32 next sound samples as $\sum_{i=0}^{15} W_{32i+j}$.

To interpret this also in terms of filters, rewrite first steps 4 to 6 as

$$\begin{aligned} x_{32(s-1)+j} &= \sum_{i=0}^{15} W_{32i+j} = \sum_{i=0}^{15} D_{32i+j} U_{32i+j} \\ &= \sum_{i=0}^{7} D_{64i+j} U_{64i+j} + \sum_{i=0}^{7} D_{64i+32+j} U_{64i+32+j} \\ &= \sum_{i=0}^{7} D_{64i+j} V_{128i+j} + \sum_{i=0}^{7} D_{64i+32+j} V_{128i+96+j}. \end{aligned} \tag{6.37}$$

The elements in V are obtained by "matrixing" different segments of the vector $\boldsymbol{z}$. More precisely, at iteration s we have that

$$\begin{pmatrix} V_{64r} \\ V_{64r+1} \\ \vdots \\ V_{64r+63} \end{pmatrix} = N \begin{pmatrix} z_{32(s-r-1)} \\ z_{32(s-r-1)+1} \\ \vdots \\ z_{32(s-r-1)+31} \end{pmatrix},$$

so that

$$V_{64r+j} = \sum_{k=0}^{31} \cos((16+j)(2k+1)\pi/64) z_{32(s-r-1)+k}$$

for $0 \leq j \leq 63$. Since also

$$V_{128i+j} = V_{64(2i)+j} \qquad V_{128i+96+j} = V_{64(2i+1)+j+32},$$

we can rewrite Equation (6.37) as

$$\begin{aligned} &\sum_{i=0}^{7} D_{64i+j} \sum_{k=0}^{31} \cos((16+j)(2k+1)\pi/64) z_{32(s-2i-1)+k} \\ &+ \sum_{i=0}^{7} D_{64i+32+j} \sum_{k=0}^{31} \cos((16+j+32)(2k+1)\pi/64) z_{32(s-2i-2))+k}. \end{aligned}$$

Again using Relation (6.35), this can be written as

$$\sum_{k=0}^{31}\sum_{i=0}^{7}(-1)^i D_{64i+j}\cos((16+64i+j)(2k+1)\pi/64)z_{32(s-2i-1)+k}$$
$$+\sum_{k=0}^{31}\sum_{i=0}^{7}(-1)^i D_{64i+32+j}\cos((16+64i+j+32)(2k+1)\pi/64)z_{32(s-2i-2)+k}.$$

Now, if we define $\{g_r\}_{r=0}^{511}$ by $g_{64i+s}=(-1)^i C_{64i+s}$, $0\le i<8, 0\le s<64$, and $g^{(k)}$ as the filter with coefficients $\{\cos((r+16)(2k+1)\pi/64)g_r\}_{r=0}^{511}$, the above can be simplified as

$$\begin{aligned}
&\sum_{k=0}^{31}\sum_{i=0}^{7}(g^{(k)})_{64i+j}z_{32(s-2i-1)+k}+\sum_{k=0}^{31}\sum_{i=0}^{7}(g^{(k)})_{64i+j+32}z_{32(s-2i-2)+k}\\
=&\sum_{k=0}^{31}\left(\sum_{i=0}^{7}(g^{(k)})_{32(2i)+j}z_{32(s-2i-1)+k}+\sum_{i=0}^{7}(g^{(k)})_{32(2i+1)+j}z_{32(s-2i-2)+k}\right)\\
=&\sum_{k=0}^{31}\sum_{r=0}^{15}(g^{(k)})_{32r+j}z_{32(s-r-1)+k},
\end{aligned}$$

where we observed that $2i$ and $2i+1$ together run through the values from 0 to 15 when i runs from 0 to 7. Since $\boldsymbol{z}$ has the same values as $\boldsymbol{z}_k$ on the indices $32(s-r-1)+k$, this can be written as

$$\begin{aligned}
&=\sum_{k=0}^{31}\sum_{r=0}^{15}(g^{(k)})_{32r+j}(\boldsymbol{z}_k)_{32(s-r-1)+k}\\
&=\sum_{k=0}^{31}(g^{(k)}\boldsymbol{z}_k)_{32(s-1)+j+k}=\sum_{k=0}^{31}((E_{-k}g^{(k)})\boldsymbol{z}_k)_{32(s-1)+j}.
\end{aligned}$$

By substituting a general s and j we see that $\boldsymbol{x}=\sum_{k=0}^{31}(E_{-k}g^{(k)})\boldsymbol{z}_k$. We have thus proved the following.

Theorem 6.25. *Reverse filter bank transform for the MP3 standard.*

Define $\{g_r\}_{r=0}^{511}$ by $g_{64i+s}=(-1)^i C_{64i+s}$, $0\le i<8, 0\le s<64$, and $g^{(k)}$ as the filter with coefficients $\{\cos((r+16)(2k+1)\pi/64)g_r\}_{r=0}^{511}$. If we define $G_k=E_{-k}g^{(k)}$, the procedure stated in the MP3 standard corresponds to applying the corresponding reverse filter bank transform.

In other words, both procedures for encoding and decoding stated in the MP3 standard both correspond to filter banks: A forward filter bank transform for the encoding, and a reverse filter bank transform for the decoding. Moreover, both filter banks can be constructed by cosine-modulating prototype filters, and the coefficients of these prototype filters are stated in the MP3 standard (up to

multiplication with an alternating sign). Note, however, that the two prototype filters may be different. When we compare the two tables for these coefficients in the standard they do indeed seem to be different. At closer inspection, however, one sees a connection: If you multiply the values in the D-table with 32, and reverse them, you get the values in the C-table. This indicates that the analysis and synthesis prototype filters are the same, up to multiplication with a scalar. This connection will be explained in Section 8.3.

While the steps defined in the MP3 standard for decoding seem a bit more complex than the steps for encoding, they are clearly also motivated by the desire to reduce the number of multiplications. In both cases (encoding and decoding), the window tables (C and D) are in direct connection with the filter coefficients of the prototype filter: one simply adds a sign which alternates for every 64 elements. The standard document does not mention this connection, and it is perhaps not so simple to find this connection in the literature (but see [35]).

The forward and reverse filter bank transforms are clearly very related. The following result clarifies this.

Theorem 6.26. *Connection between the forward and reverse filter bank transforms in the MP3 standard.*

Assume that a forward filter bank transform has filters on the form $H_i = E_{i-31}h^{(i)}$ for a prototype filter h. Then $G = E_{481}H^T$ is a reverse filter bank transform with filters on the form $G_k = E_{-k}g^{(k)}$, where g is a prototype filter where the elements equal the reverse of those in h. Vice versa, $H = E_{481}G^T$.

Proof. From Theorem 6.23 we know that H^T is a reverse filter bank transform with filters

$$(H_i)^T = (E_{i-31}h^{(i)})^T = E_{31-i}(h^{(i)})^T.$$

$(h^{(i)})^T$ has filter coefficients $\cos((2i+1)(-k-16)\pi/64))h_{-k}$. If we delay all $(H_i)^T$ with $481 = 512 - 31$ elements as in the theorem, we get a total delay of $512 - 31 + 31 - i = 512 - i$ elements, so that we get the filter

$$\begin{aligned}
&E_{512-i}\{\cos((2i+1)(-k-16)\pi/64))h_{-k}\}_k \\
&= E_{-i}\{\cos((2i+1)(-(k-512)-16)\pi/64))h_{-(k-512)}\}_k \\
&= E_{-i}\{\cos((2i+1)(k+16)\pi/64))h_{-(k-512)}\}_k.
\end{aligned}$$

Now, we define the prototype filter g with elements $g_k = h_{-(k-512)}$. This has, just as h, its support on $[1, 511]$, and consists of the elements from h in reverse order. If we define $g^{(i)}$ as the filter with coefficients $\cos((2i+1)(k+16)\pi/64))g_k$, we see that $E_{481}H^T$ is a reverse filter bank transform with filters $E_{-i}g^{(i)}$. Since $g^{(k)}$ now has been defined as for the MP3 standard, and its elements are the reverse of those in h, the result follows. □

We will have use for this result in Section 8.3, when we find conditions on the prototype filter in order for the reverse transform to invert the forward

transform. Preferably, the reverse filter bank transform inverts exactly the forward filter bank transform. In Exercise 6.26 we construct examples which show that this is not the case. In the same exercise we also find many examples where the reverse transform does what we would expect. These examples will also be explained in Section 8.3, where we also will see how one can get around this so that we obtain a system with perfect reconstruction. It may seem strange that the MP3 standard does not do this.

In the MP3 standard, the output from the forward filter bank transform is processed further, before the result is compressed using a lossless compression method.

Exercise 6.25: Plotting frequency responses

The values C_q, D_q can be found by calling the functions `mp3ctable`, `mp3dtable` which can be found on the book's webpage.

a) Use your computer to verify the connection we stated between the tables C and D, i.e. that $D_i = 32C_i$ for all i.

b) Plot the frequency responses of the corresponding prototype filters, and verify that they both are lowpass filters. Use the connection from Theorem (6.24) to find the prototype filter coefficients from the C_q.

Exercise 6.26: Implementing forward and reverse filter bank transforms

It is not too difficult to make implementations of the forward and reverse steps as explained in the MP3 standard. In this exercise we will experiment with this. In your code you can for simplicity assume that the input and output vectors to your methods all have lengths which are multiples of 32. Also, use the functions `mp3ctable`, `mp3dtable` mentioned in the previous exercise.

a) Write a function `mp3forwardfbt` which implements the steps in the forward direction of the MP3 standard.

b) Write also a function `mp3reversefbt` which implements the steps in the reverse direction.

6.4 Summary

We started this chapter by noting that, by reordering the target base of the DWT, the change of coordinate matrix took a particular form. From this form we understood that the DWT could be realized in terms of two filters H_0 and H_1, and that the IDWT could be realized in a similar way in terms of two filters G_0 and G_1. This gave rise to what we called the filter representation of wavelets. The filter representation gives an entirely different view on wavelets: instead of constructing function spaces with certain properties and deducing corresponding

filters from these, we can instead construct filters with certain properties (such as alias cancellation and perfect reconstruction), and attempt to construct corresponding mother wavelets, scaling functions, and function spaces. This strategy, which replaces problems from function theory with discrete problems, will be the subject of the next chapter. In practice this is what is done.

We stated what is required for filter bank matrices to invert each other: The frequency responses of the lowpass filters needed to satisfy a certain equation, and once this is satsified the highpass filters can easily be obtained in the same way we previously obtained highpass filters from lowpass filters. We will return to this equation in the next chapter.

A useful consequence of the filter representation was that we could reuse existing implementations of filters to implement the DWT and the IDWT, and reuse existing theory, such as symmetric extensions. For wavelets, symmetric extensions are applied in a slightly different way, when compared to the developments which lead to the DCT. We looked at the frequency responses of the 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 why this is typically the case for other wavelets as well. The filter reprersentation was also easily generalized from 2 to $M > 2$ filters, and such transformations had a similar interpretation in terms of splitting the input into a uniform set of frequencies. Such transforms were generally called filter bank transforms, and we saw that the processing performed by the MP3 standard could be interpreted as a certain filter bank transform, called a cosine-modulated filter bank. This is just one of many possible filter banks. In fact, the filter bank of the MP3 standard is largely outdated, since it is too simple, and as we will see it does not even give perfect reconstruction (only alias cancellation and no phase distortion). It is merely chosen here since it is the simplest to present theoretically, and since it is perhaps the best known standard for compression of sound. Other filters banks with better properties have been constructed, and they are used in more recent standards. In many of these filter banks, the filters do not partition frequencies uniformly, and have been adapted to the way the human auditory system handles the different frequencies. Different contruction methods are used to construct such filter banks. The motivation behind filter bank transforms is that their output is more suitable for further processing, such as compression, or playback in an audio system, and that they have efficient implementations.

We mentioned that the MP3 standard does not say how the prototype filters were chosen. We will have more to say on what dictates their choice in Section 8.3.

There are several differences between the use of wavelet transformations in wavelet theory, and the use of filter bank transforms in signal processing theory One is that wavelet transforms are typically applied in stages, while filter bank transforms often are not. Nevertheless, such use of filter banks also has theoretical importance, and this gives rise to what is called *tree-structured filter banks* [47]. Another difference lies in the use of the term perfect reconstruction system. In wavelet theory this is a direct consequence of the wavelet construction, since the DWT and the IDWT correspond to change of coordinates to and from the same bases. The alternative QMF filter bank was used as an example

of a filter bank which stems from signal processing, and which also shows up in wavelet transformation. In signal processing theory, one has a wider perspective, since one can design many useful systems with fast implementations when one replaces the perfect reconstruction requirement with a near perfect reconstruction requirement. One instead requires that the reverse transform gives alias cancellation. The classical QMF filter banks were an example of this. The original definition of classical QMF filter banks are from [9], and differ only in a sign from how they are defined here.

All filters we encounter in wavelets and filter banks in this book are FIR. This is just done to limit the exposition. Much useful theory has been developed using IIR-filters.

What you should have learned in this chapter.

- How one can find the filters of a wavelet transformation by considering its matrix and its inverse.
- Forward and reverse filter bank transforms.
- How one can implement the DWT and the IDWT with the help of the filters.
- Plot of the frequency responses for the filters of the wavelets we have considered, and their interpretation as low-pass and high-pass filters.

Chapter 7

Constructing interesting wavelets

In the previous chapter, from an MRA with corresponding scaling function and mother wavelet, we defined what we called a forward filter bank transform. We also defined a reverse filter bank transform, but we did not state an MRA connected to this, or prove if any such association could be made. In this chapter we will address this. We will also see, if we start with a forward and reverse filter bank transform, how we can construct corresponding MRA's, and for which transforms we can make this construction. We will see that there is a great deal of flexibility in the filter bank transforms we can construct (as this is a discrete problem). Actually it is so flexible that we can construct scaling functions/mother wavelets with any degree of regularity, and well suited for approximation of functions. This will also explain our previous interest in vanishing moments, and explain how we can find the simplest filters which give rise to a given number of vanishing moments, or a given degree of differentiability.. Answers to these questions certainly transfer much more theory between wavelets and filters. Several of these filters enjoy a widespread use in applications. We will look at two of these. These are used for lossless and lossy compression in JPEG2000, which is a much used standard. These wavelets all have symmetric filters. We end the chapter by looking at a family of orthonormal wavelets with different number of vanishing moments.

7.1 From filters to scaling functions and mother wavelets

From Theorem 6.9 it follows that the support sizes of these dual functions are are 4 and 3, respectively, so that their supports should be $[-2, 2]$ and $[-1, 2]$, respectively. This is the reason why we have plotted the functions over $[-2, 2]$. The plots seem to confirm the support sizes we have computed.

In our first examples of wavelets in Chapter 5, we started with some bases og functions $\boldsymbol{\phi}_m$, and deduced filters G_0 and G_1 from these. If we instead start with the filters G_0 and G_1, what properties must they fulfill in order for us to make an association the opposite way? We should thus demand that there exist functions ϕ, ψ so that

$$\phi(t) = \sum_{n=0}^{2N-1} (G_0)_{n,0}\phi_{1,n}(t) \tag{7.1}$$

$$\psi(t) = \sum_{n=0}^{2N-1} (G_1)_{n,1}\phi_{1,n}(t) \tag{7.2}$$

Using Equation (7.1), the Fourier transform of ϕ is

$$\begin{aligned}\hat{\phi}(\omega) &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}\phi(t)e^{-i\omega t}dt = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}\left(\sum_n (G_0)_{n,0}\sqrt{2}\phi(2t-n)\right)e^{-i\omega t}dt \\ &= \frac{1}{\sqrt{2}\sqrt{2\pi}}\sum_n\int_{-\infty}^{\infty}(G_0)_{n,0}\phi(t)e^{-i\omega(t+n)/2}dt \\ &= \frac{1}{\sqrt{2}}\left(\sum_n (G_0)_{n,0}e^{-i\omega n/2}\right)\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}\phi(t)e^{-i(\omega/2)t)}dt = \frac{\lambda_{G_0}(\omega/2)}{\sqrt{2}}\hat{\phi}(\omega/2).\end{aligned} \tag{7.3}$$

Clearly this expression can be continued recursively. We can thus state the following result.

Theorem 7.1. g_N.
Define

$$g_N(\omega) = \prod_{s=1}^{N}\frac{\lambda_{G_0}(\omega/2^s)}{\sqrt{2}}\chi_{[0,2\pi]}(2^{-N}\omega). \tag{7.4}$$

Then on $[0, 2\pi 2^N]$ we have that $\hat{\phi}(\nu) = g_N(\nu)\hat{\phi}(\nu/2^N)$.

We can now prove the following.

Lemma 7.2. *$g_N(\nu)$ converges.*
Assume that $\sum_n (G_0)_n = \sqrt{2}$ (i.e. $\lambda_{G_0}(0) = \sqrt{2}$), and that G_0 is a FIR-filter. Then $g_N(\nu)$ converges pointwise as $N \to \infty$ to an infinitely differentiable function.

Proof. We need to verify that the infinite product $\prod_{s=1}^{\infty}\frac{\lambda_{G_0}(2\pi\nu/2^s)}{\sqrt{2}}$ converges. Taking logarithms we get $\sum_s \ln\left(\frac{\lambda_{G_0}(2\pi\nu/2^s)}{\sqrt{2}}\right)$. To see if this series converges, we consider the ratio between two successive terms:

$$\frac{\ln\left(\frac{\lambda_{G_0}(2\pi\nu/2^{s+1})}{\sqrt{2}}\right)}{\ln\left(\frac{\lambda_{G_0}(2\pi\nu/2^{s})}{\sqrt{2}}\right)}.$$

Since $\sum_n (G_0)_n = \sqrt{2}$, we see that $\lambda_{G_0}(0) = \sqrt{2}$. Since $\lim_{\nu\to 0}\lambda_{G_0}(\nu) = \sqrt{2}$, both the numerator and the denominator above tends to 0 (to one inside the logarithms), so that we can use L'hospital's rule on $\frac{\ln\left(\frac{\lambda_{G_0}(\nu/2)}{\sqrt{2}}\right)}{\ln\left(\frac{\lambda_{G_0}(\nu)}{\sqrt{2}}\right)}$ to obtain

$$\frac{\lambda_{G_0}(\nu)}{\lambda_{G_0}(\nu/2)}\frac{\sum_n (G_0)_n(-in)e^{-in\nu/2}/2}{\sum_n (G_0)_n(-in)e^{-in\nu}} \to \frac{1}{2} < 1$$

as $\nu \to 0$. It follows that the product converges for any ν. Clearly the convergence is absolute and uniform on compact sets, so that the limit is infinitely differentiable. □

It follows that $\hat{\phi}$, when ϕ exists, must be an infinitely differentiable function also. Similarly we get

$$\begin{aligned}\hat{\psi}(\omega) &= \frac{1}{\sqrt{2}}\left(\sum_n (G_1)_{n-1,0}e^{-i\omega n/2}\right)\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}\phi(t)e^{-i(\omega/2)t)}dt \\ &= \frac{1}{\sqrt{2}}\left(\sum_n (G_1)_{n,0}e^{-i\omega(n+1)/2}\right)\hat{\phi}(\omega/2) = e^{-i\omega/2}\frac{\lambda_{G_1}(\omega/2)}{\sqrt{2}}\hat{\phi}(\omega/2).\end{aligned}$$

It follows in the same way that $\hat{\psi}$ must be an infinitely differentiable function also.

Now consider the dual filter bank transform,as defined in Chapter 6. Its synthesis filter are $(H_0)^T$ and $(H_1)^T$. If there exist a scaling function $\tilde{\phi}$ and a mother wavelet $\tilde{\psi}$ for the dual transform, they must in the same way be infinitely differentiable. Moreover, $\hat{\phi}, \hat{\psi}, \hat{\tilde{\phi}}, \hat{\tilde{\psi}}$ can be found as infinite products of the known frequency responses. If these functions are in $L^2\mathbb{R}$, then we can find unique functions $\phi, \psi, \tilde{\phi}, \tilde{\psi}$ with these as Fourier transforms.

So, our goal is to find filters so that the derived infinite products of the frequency responses lie in $L^2(\mathbb{R})$, and so that the constructed functions $\phi, \psi, \tilde{\phi}, \tilde{\psi}$ give rise to "nice" wavelet bases. Some more technical requirements will be needed in order for this. In order to state these we should be clear on what we mean by a "nice" basis in this context. First of all, the bases should together span all of $L^2(\mathbb{R})$. But our bases are not orthogonal, so we should have some substitute for this. We will need the following definitions.

Definition 7.3. *Frame.*

Let $\mathcal{H}$ be a Hilbert space. A set of vectors $\{u_n\}_n$ is called a frame of $\mathcal{H}$ if there exist constants $A > 0$ and $B > 0$ so that, for any $f \in \mathcal{H}$,

$$A\|f\|^2 \leq \sum_n |\langle f, u_n\rangle|^2 \leq B\|f\|^2.$$

If $A = B$, the frame is said to be tight.

Note that, for a frame of $\mathcal{H}$, any $f \in \mathcal{H}$ is uniquely characterized by the inner products $\langle f, u_n\rangle$. Indeed, if both $a, b \in \mathcal{H}$ have the same inner products, then $a - b \in \mathcal{H}$ have inner products 0, which implies that $a = b$ from the left inequality.

For every frame one can find a dual frame $\{\tilde{u_n}\}_n$ which satisfies

$$\frac{1}{B}\|f\|^2 \leq \sum_n |\langle f, \tilde{u_n}\rangle|^2 \leq \frac{1}{A}\|f\|^2,$$

and

$$f = \sum_n \langle f, u_n\rangle \tilde{u_n} = \sum_n \langle f, \tilde{u_n}\rangle u_n. \tag{7.5}$$

Thus, if the frame is tight, the dual frame is also tight.

A frame is called a Riesz basis if all its vectors also are linearly independent. One can show that the vectors in the dual frame of a Riesz basis also are linearly independent, so that the dual frame of a Riesz basis also is a Riesz basis. It is also called the dual Riesz basis. We will also need the following definition.

Definition 7.4. *Biorthogonal bases.*

We say that two bases $\{\boldsymbol{f}_n\}_n$, $\{\boldsymbol{g}_m\}_m$ are *biorthogonal* if $\langle \boldsymbol{f}_n, \boldsymbol{g}_m\rangle = 0$ whenever $n \neq m$, and 1 if $n = m$.

From Equation (7.5) and linear independence, it is clear that the vectors in a Riesz basis and in its dual Riesz basis are biorthogonal. In the absence of orthonormal bases for $L^2(\mathbb{R})$, the best we can hope for is dual Riesz bases for $L^2(\mathbb{R})$. The following result explains how we can obtain this from the filters.

Proposition 7.5. *Biorthogonality.*

Assume that the frequency responses λ_{G_0} and λ_{H_0} can be written as.

$$\frac{\lambda_{G_0}(\omega)}{\sqrt{2}} = \left(\frac{1 + e^{-i\omega}}{2}\right)^L \mathcal{F}(\omega) \qquad \frac{\lambda_{H_0}(\omega)}{\sqrt{2}} = \left(\frac{1 + e^{-i\omega}}{2}\right)^{\tilde{L}} \tilde{\mathcal{F}}(\omega), \tag{7.6}$$

where $\mathcal{F}$ and $\tilde{\mathcal{F}}$ are trigonometric polynomials of finite degree. Assume also that, for some $k, \tilde{k} > 0$,

$$B_k = \max_\omega \left|\mathcal{F}(\omega)\cdots\mathcal{F}(2^{k-1}\omega)\right|^{1/k} < 2^{L-1/2} \tag{7.7}$$

$$\tilde{B_k} = \max_\omega \left|\tilde{\mathcal{F}}(\omega)\cdots\tilde{\mathcal{F}}(2^{\tilde{k}-1}\omega)\right|^{1/\tilde{k}} < 2^{\tilde{L}-1/2} \tag{7.8}$$

Then the following hold:

- $\phi, \tilde{\phi} \in L^2(\mathbb{R})$, and the corresponding bases $\boldsymbol{\phi}_0$ and $\tilde{\boldsymbol{\phi}}_0$ are biorthogonal.
- $\boldsymbol{\psi}_{m,n}$ is a Riesz basis of $L^2(\mathbb{R})$.
- $\tilde{\boldsymbol{\psi}}_{m,n}$ is the dual Riesz basis of $\boldsymbol{\psi}_{m,n}$. Thus, $\boldsymbol{\psi}_m, n$ and $\tilde{\boldsymbol{\psi}}_{m,n}$ are biorthogonal bases, and for any $f \in L^2(\mathbb{R})$,

$$f = \sum_{m,n} \langle f, \tilde{\psi}_{m,n} \rangle \psi_{m,n} = \sum_{m,n} \langle f, \psi_{m,n} \rangle \tilde{\psi}_{m,n}. \tag{7.9}$$

If also

$$B_k < 2^{L-1-m} \qquad\qquad \tilde{B}_k < 2^{\tilde{L}-1-\tilde{m}}, \tag{7.10}$$

then

- ϕ, ψ are m times differentiable and $\tilde{\psi}$ has $m+1$ vanishing moments,
- $\tilde{\phi}, \tilde{\psi}$ are $\tilde{m}$ times differentiable and ψ has $\tilde{m}+1$ vanishing moments.

The proof for Proposition 7.5 is long, technical, and split in many stages. The entire proof can be found in [7], and we will not go through all of it, only address some simple parts of it in the following subsections. After that we will see how we can find G_0, H_0 so that equations (7.6), (7.7), (7.8) are fulfilled. Before we continue on this path, several comments are in order.

1. The paper [7] much more general conditions for when filters give rise to a Riesz basis as stated here. The conditions (7.7), (7.8) are simply chosen because they apply or the filters we consider.

2. From Equation (7.6) it follows that the flatness in the frequency responses close to π explains how good the bases are for approximations, since the number of vanishing moments is infered from the multiplictity of the zero at π for the frequency response.

3. From the result we obtain an MRA (with scaling function ϕ), and a dual MRA (with scaling function $\tilde{\phi}$), as well as mother wavelets (ψ and $\tilde{\psi}$), and we can define the resolution spaces V_m and the detail spaces W_m as before, as well as the "dual resolution spaces" $\tilde{V}_m$, (the spaces spanned by $\tilde{\boldsymbol{\phi}}_m = \{\tilde{\phi}_{m,n}\}_n$) and "dual detail spaces" $\tilde{W}_m$ (the spaces spanned by $\tilde{\boldsymbol{\psi}}_m = \{\tilde{\psi}_{m,n}\}_n$). In general V_m is different from $\tilde{V}_m$ (except when $\phi = \tilde{\phi}$), and W_m is in general different from the orthogonal complement of V_{m-1} in V_m (except when $\phi = \tilde{\phi}$, when all bases are orthonormal), although constructed so that $V_m = V_{m-1} \oplus W_{m-1}$. Our construction thus involves two MRA's

$$V_0 \subset V_1 \subset V_2 \subset \cdots \subset V_m \subset \cdots \qquad \tilde{V}_0 \subset \tilde{V}_1 \subset \tilde{V}_2 \subset \cdots \subset \tilde{V}_m \subset \cdots$$

where there are different scaling functions, satisfying a biorthogonality relationship. This is also called a dual multiresolution analysis.

4. The DWT and IDWT are defined as before, so that the same change of coordinates can be applied, as dictated by the filter coefficients. As will be seen below, while proving Proposition 7.5 it also follows that the bases $\boldsymbol{\phi}_0 \oplus \boldsymbol{\psi}_0 \oplus \boldsymbol{\psi}_1 \cdots \boldsymbol{\psi}_{m-1}$ and $\tilde{\boldsymbol{\phi}}_0 \oplus \tilde{\boldsymbol{\psi}}_0 \oplus \tilde{\boldsymbol{\psi}}_1 \cdots \tilde{\boldsymbol{\psi}}_{m-1}$ are biorthogonal (in addition to that $\boldsymbol{\phi}_m$ and $\tilde{\phi}_m$ are biorthogonal, as stated). For $f \in V_m$ this means that

$$f(t) = \sum_n \langle f(t), \tilde{\phi}_{m,n}\rangle \phi_{m,n} = \sum_n \langle f(t), \tilde{\phi}_{0,n}\rangle \phi_{0,n} + \sum_{m'<m,n} \langle f(t), \tilde{\psi}_{m',n}\rangle \psi_{m',n},$$

since this relationship is fulfilled for any linear combination of the $\{\phi_{m,n}\}_n$, or for any of the $\{\phi_{0,n}, \psi_{m',n}\}_{m'<m,n}$, due to biorthogonality. Similarly, for $\tilde{f} \in \tilde{V}_m$

$$\tilde{f}(t) = \sum_n \langle \tilde{f}(t), \phi_{m,n}\rangle \tilde{\phi}_{m,n} = \sum_n \langle \tilde{f}(t), \phi_{0,n}\rangle \tilde{\phi}_{0,n} + \sum_{m'<m,n} \langle \tilde{f}(t), \psi_{m',n}\rangle \tilde{\psi}_{m',n}.$$

It follows that for $f \in V_m$ and for $\tilde{f} \in \tilde{V}_m$ the DWT and the IDWT and their duals can be expressed in terms of inner products as follows.

- The input to the DWT is $c_{m,n} = \langle f, \tilde{\phi}_{m,n}\rangle$. The output of the DWT is $c_{0,n} = \langle f, \tilde{\phi}_{0,n}\rangle$ and $w_{m',n} = \langle f, \tilde{\psi}_{m',n}\rangle$
- The input to the dual DWT is $\tilde{c}_{m,n} = \langle \tilde{f}, \phi_{m,n}\rangle$. The output of the dual DWT is $\tilde{c}_{0,n} = \langle \tilde{f}, \phi_{0,n}\rangle$ and $\tilde{w}_{m',n} = \langle \tilde{f}, \psi_{m',n}\rangle$.
- in the DWT matrix, column k has entries $\langle \phi_{1,k}, \tilde{\phi}_{0,l}\rangle$, and $\langle \phi_{1,k}, \tilde{\psi}_{0,l}\rangle$ (with a similar expression for the dual DWT).
- in the IDWT matrix, column $2k$ has entries $\langle \phi_{0,k}, \tilde{\phi}_{1,l}\rangle$, and column $2k+1$ has entries $\langle \psi_{0,k}, \tilde{\phi}_{1,l}\rangle$ (with a similar expression for the dual IDWT).

Equation (7.9) comes from eliminating the $\phi_{m,n}$ by letting $m \to \infty$.

5. When $\phi = \tilde{\phi}$ (orthonormal MRA's), the approximations (finite sums) above coincide with projections onto the spaces $V_m, \tilde{V}_m, W_m, \tilde{W}_m$. When $\phi \neq \tilde{\phi}$, however, there are no reasons to believe that these approximations equal the best approximations to f from V_m. In this case we have no procedure for computing best approximations. When f is not in $V_m, \tilde{V}_m$ we can, however, consider the approximations

$$\sum_n \langle f(t), \tilde{\phi}_{m,n}\rangle \phi_{m,n}(t) \in V_m \text{ and } \sum_n \langle f(t), \phi_{m,n}\rangle \tilde{\phi}_{m,n}(t) \in \tilde{V}_m$$

(when the MRA is orthonormal, this coincides with the best approximation). Now, we can choose m so large that $f(t) = \sum_n c_n \phi_{m,n}(t) + \epsilon(t)$, with $\epsilon(t)$ a small function. The first approximation can now be written

$$\begin{aligned}\sum_n \langle \sum_{n'} c_{n'} \phi_{m,n'}(t) + \epsilon(t), \tilde{\phi}_{m,n}\rangle \phi_{m,n}(t) &= \sum_n c_n \phi_{m,n}(t) + \sum_n \langle \epsilon(t), \tilde{\phi}_{m,n}\rangle \phi_{m,n}(t) \\ &= f(t) + \sum_n \langle \epsilon(t), \tilde{\phi}_{m,n}\rangle \phi_{m,n}(t) - \epsilon(t).\end{aligned}$$

Clearly, the difference $\sum_n \langle \epsilon(t), \tilde{\phi}_{m,n} \rangle \phi_{m,n}(t) - \epsilon(t)$ from f is small. It may. however, be hard to compute the c_n above, so that instead, as in Theorem 5.33, one uses $\frac{2^{-m}}{\int_0^N \phi_{m,0}(t)dt} f(n/2^m)\phi_{m,n}(t)$ as an approximation to f (i.e. use sample values as c_n) also in this more general setting.

6. Previously we were taught to think in a periodic or folded way, so that we could restrict to an interval $[0, N]$, and to bases of finite dimensions $(\{\phi_{0,n}\}_{n=0}^{N-1})$. But the results above are only stated for wavelet bases of infinite dimension. Let us therefore say something on how the results carry over to our finite dimensional setting. If $f \in L^2(\mathbb{R})$ we can define the function

$$f^{per}(t) = \sum_k f(t + kN) \qquad f^{fold}(t) = \sum_k f(t + 2kN) + \sum_k f(2kN - t).$$

f^{per} and f^{fold} are seen to be periodic with periods N and $2N$. It is easy to see that the restriction of f^{per} to $[0, N]$ is in $L^2([0, N])$, and that the restriction of f^{fold} to $[0, 2N]$ is in $L^2([0, 2N])$. In [6] it is shown that the result above extends to a similar result for the periodized/folded basis (i.e. $\psi_{m,n}^{fold}$), so that we obtain dual Riesz bases for $L^2([0, N])$ and $L^2([0, 2N])$ instead of $L^2(\mathbb{R})$. The result on the vanishing moments does not extend, however. One can, however, alter some of the basis functions so that one achieves this. This simply changes some of the columns in the DWT/IDWT matrices. Note that our extension strategy is not optimal. The extension is usually not differentiable at the boundary, so that the corresponding wavelet coefficients may be large, even though the wavelet has many vanishing moments. The only way to get around this would be to find an extension strategy which gave a more regular extension. However, natural images may not have high regularity, which would make such an extension strategy useless.

Sketch of proof for the biorthogonality in Proposition 7.5 (1). We first show that $\boldsymbol{\phi}_0$ and $\tilde{\boldsymbol{\phi}}_0$ are biorthogonal. Recall that definition (7.4) said that $g_N(\omega) = \prod_{s=1}^N \frac{\lambda_{G_0}(\omega/2^s)}{\sqrt{2}} \chi_{[0,2\pi]}(2^{-N}\omega)$. Let us similarly define $h_N(\omega) = \prod_{s=1}^N \frac{\lambda_{H_0}(\omega/2^s)}{\sqrt{2}} \chi_{[0,2\pi]}(2^{-N}\omega)$. Recall that $g_N \to \hat{\phi}$ and $h_N \to \hat{\tilde{\phi}}$ pointwise as $N \to \infty$. We have that

$$g_{N+1}(\omega) = \frac{\lambda_{G_0}(\omega/2)}{\sqrt{2}} g_N(\omega/2) \qquad h_{N+1}(\omega) = \frac{\lambda_{H_0}(\omega/2)}{\sqrt{2}} h_N(\omega/2).$$

g_N, h_N are compactly supported, and equal to trigonometric polynomials on their support, so that $g_N, h_N \in L^2(\mathbb{R})$. Since the Fourier transform also is an isomorphism og $L^2(\mathbb{R})$ onto itself, there exist functions $u_N, v_N \in L^2(\mathbb{R})$ so that $g_N = \hat{u_N}$, $h_N = \hat{v_N}$. Since the above relationship equals that of Equation (7.3), with $\hat{\phi}$ replaced with g_N, we must have that

$$u_{N+1}(t) = \sum_n (G_0)_{n,0}\sqrt{2}u_N(2t-n) \quad v_{N+1}(t) = \sum_n (H_0)_{0,n}\sqrt{2}v_N(2t-n).$$

Now, note that $g_0(\omega) = h_0(\omega) = \chi_{[0,1]}(\omega)$. Since $\langle u_0, v_0\rangle = \langle g_0, h_0\rangle$ we get that

$$\int_\infty^{-\infty} u_0(t)\overline{v_0(t-k)}dt = \int_\infty^{-\infty} g_0(\nu)\overline{h_0(\nu)e^{2\pi ik\nu}}d\nu = \int_0^{2\pi} e^{-2\pi ik\nu}d\nu = \delta_{k,0}.$$

Now assume that we have proved that $\langle u_N(t), v_N(t-k)\rangle = \delta_{k,0}$. We then get that

$$\begin{aligned}
\langle u_{N+1}(t), v_{N+1}(t-k)\rangle &= 2\sum_{n_1,n_2} (G_0)_{n_1,0}(H_0)_{0,n_2}\langle u_N(2t-n_1), v_N(2(t-k)-n_2)\rangle \\
&= 2\sum_{n_1,n_2} (G_0)_{n_1,0}(H_0)_{0,n_2}\langle u_N(t), v_N(t+n_1-n_2-2k)\rangle \\
&= \sum_{n_1,n_2|n_1-n_2=2k} (G_0)_{n_1,0}(H_0)_{0,n_2} = \sum_n (H_0)_{0,n-2k}(G_0)_{n,0} \\
&= \sum_n (H_0)_{2k,n}(G_0)_{n,0} = \sum_n H_{2k,n}G_{n,0} = (HG)_{2k,0} = I_{2k,0} = \delta_{k,0}
\end{aligned}$$

where we did the change of variables $u = 2t - n_1$. There is an extra argument to show that $g_N \to_{L^2} \hat{\phi}$ (stronger than pointwise convergence as was stated above), so that also $u_N \to_{L^2} \phi \in L^2(\mathbb{R})$, since the Fourier transform is an isomorphism of $L^2(\mathbb{R})$ onto itself. It follows that

$$\langle \phi_{m,k}, \tilde{\phi}_{m,l}\rangle = \lim_{N\to\infty} \langle u_N(t-k), v_N(t-l)\rangle = \delta_{k,l}.$$

While proving this one also establishes that

$$|\hat{\phi}(\omega)| \le C(1+|\omega|)^{-1/2-\epsilon} \qquad |\hat{\tilde{\phi}}(\omega)| \le C(1+|\omega|)^{-1/2-\epsilon}, \tag{7.11}$$

where $\epsilon = L - 1/2 - \log B_k/\log 2 > 0$ due to Assumption (7.7). In the paper it is proved that this condition implies that the bases constitute dual frames. The biorthogonality is used to show that they also are dual Riesz bases (i.e. that they also are linearly independent).

Sketch of proof for the biorthogonality of in Proposition 7.5 (2). The biorthogonality of $\boldsymbol{\psi}_{m,n}$ and $\tilde{\boldsymbol{\psi}}_{m,n}$ can be deduced from the biorthogonality of $\boldsymbol{\phi}_0$ and $\tilde{\boldsymbol{\phi}}_0$ as follows. We have that

$$\begin{aligned}\langle \psi_{0,k}, \tilde{\psi}_{0,l} \rangle &= \sum_{n_1,n_2} (G_1)_{n_1,1}(H_1)_{1,n_2} \langle \phi_{1,n_1+2k}(t) \tilde{\phi}_{1,n_2+2l}(t) \rangle \\ &= \sum_n (G_1)_{n,1}(H_1)_{1,n+2(k-l)} = \sum_n (H_1)_{1+2(l-k),n}(G_1)_{n,1} = \sum_n H_{1+2(l-k),n} G_{n,1} \\ &= (HG)_{1+2(l-k),1} = \delta_{k,0}.\end{aligned}$$

Similarly,

$$\begin{aligned}\langle \psi_{0,k} \tilde{\phi}_{0,l} \rangle &= \sum_{n_1,n_2} (G_1)_{n_1,1}(H_0)_{0,n_2} \langle \phi_{1,n_1+2k}(t) \tilde{\phi}_{1,n_2+2l}(t) \rangle = \sum_n (G_1)_{n,1}(H_0)_{0,n+2(k-l)} \\ &= \sum_n (H_0)_{2(l-k),n}(G_1)_{n,1} = \sum_n H_{2(l-k),n} G_{n,1} = (HG)_{2(l-k),1} = 0 \\ \langle \phi_{0,k} \tilde{\psi}_{0,l} \rangle &= \sum_{n_1,n_2} (G_0)_{n_1,0}(H_1)_{1,n_2} \langle \phi_{1,n_1+2k}(t) \tilde{\phi}_{1,n_2+2l}(t) \rangle = \sum_n (G_0)_{n,0}(H_1)_{1,n+2(k-l)} \\ &= \sum_n (H_1)_{1+2(l-k),n}(G_0)_{n,0} = \sum_n H_{1+2(l-k),n} G_{n,0} = (HG)_{1+2(l-k),0} = 0.\end{aligned}$$

From this we also get with a simple change of coordinates that

$$\langle \psi_{m,k}, \tilde{\psi}_{m,l} \rangle = \langle \psi_{m,k}, \tilde{\phi}_{m,l} \rangle = \langle \phi_{m,k}, \tilde{\psi}_{m,l} \rangle = 0.$$

Finally, if $m' < m$, $\phi_{m',k'}, \psi_{m',k}$ can be written as a linear combination of $\phi_{m,l}$, so that $\langle \phi_{m',k}, \tilde{\psi}_{m,l} \rangle = \langle \psi_{m',k}, \tilde{\psi}_{m,l} \rangle = 0$ due to what we showed above. Similarly, $\langle \tilde{\phi}_{m',k}, \psi_{m,l} \rangle = \langle \tilde{\psi}_{m',k}, \psi_{m,l} \rangle = 0$.

Regularity and vanishing moments. Now assume also that $B_k < 2^{L-1-m}$, so that $\log B_k < L - 1 - m$. We have that $\epsilon = L - 1/2 - \log B_k / \log 2 > L - 1/2 - L + 1 + m = m + 1/2$, so that $|\hat{\phi}(\omega)| < C(1 + |\omega|)^{-1/2-\epsilon} = C(1 + |\omega|)^{-m-1-\delta}$ for some $\delta > 0$. This implies that $\hat{\phi}(\omega)(1 + |\omega|)^m < C(1 + |\omega|)^{-1-\delta} \in L^1$. An important property of the Fourier transform is that $\hat{\phi}(\omega)(1 + |\omega|)^m \in L^1$ if and only if ϕ is m times differentiable. This property implies that ϕ, and thus ψ is m times differentiable. Similarly, $\tilde{\phi}$, $\tilde{\psi}$ are $\tilde{m}$ times differentiable.

In [7] it is also proved that if

- $\psi_{m,n}$ and $\tilde{\psi}_{m,n}$ are biorthogonal bases,
- ψ is m times differentiable with all derivatives $\psi^{(l)}(t)$ of order $l \leq m$ bounded, and
- $\tilde{\psi}(t) < C(1 + |t|)^{m+1}$,

then $\tilde{\psi}$ has $m + 1$ vanisning moments. In our case we have that ψ and $\tilde{\psi}$ have compact support, so that these conditions are satisfied. It follows that $\tilde{\psi}$ has $m + 1$ vanisning moments.

In the next section we will construct a wide range of forward and reverse filter bank transforms which invert each other, and which give rise to wavelets.

In [7] one checks that many of these wavelets satisfy (7.7) and (7.8) (implying that they give rise to dual Riesz bases for $L^2(\mathbb{R})$), or the more general (7.10) (implying a certain regularity and a certain number of vanishing moments). Requirements on the filters lengths in order to obtain a given number of vanishing moments are also stated.

Exercise 7.1: Implementation of the cascade algorithm

a) In the code above, we turned off symmetric extensions (the `symm`-argument is 0). Attempt to use symmetric extensions instead, and observe the new plots you obtain. Can you explain why these new plots do not show the correct functions, while the previous plots are correct?

Exercise 7.2: Using the cascade algorithm

In Exercise 6.20 we constructed a new mother wavelet $\hat{\psi}$ for piecewise linear functions by finding constants $\alpha, \beta, \gamma, \delta$ so that

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

Use the cascade algorithm to plot $\hat{\psi}$. Do this by using the wavelet kernel for the piecewise linear wavelet (do not use the code above, since we have not implemented kernels for this wavelet yet).

7.2 Vanishing moments

The scaling functions and mother wavelets we constructed in Chapter 5 were very simple. They were however, enough to provide scaling functions which were differentiable. This may clearly be important for signal approximation, at least in cases where we know certain things about the regularity of the functions we approximate. However, there seemed to be nothing which dictated how the mother wavelet should be chosen in order to be useul. To see that this may pose a problem, consider the mother wavelet we hose for piecewise linear functions. 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.

Idea 7.6. *Approximation.*

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 7.9 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.

In this section we will address a property which the mother wavelet must fulfill in order to be useful in this respect. To motivate this property, let us first use decompose $f \in V_m$ as

$$f = \sum_{n=0}^{N-1} \langle f, \tilde{\phi}_{0,n} \rangle \phi_{0,n} + \sum_{r=0}^{m-1} \sum_{n=0}^{2^r N-1} \langle f, \tilde{\psi}_{r,n} \rangle \psi_{r,n}. \tag{7.12}$$

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 (7.12) 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 7.7. *Vanishing moments.*
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 5.14. It is an exercise to see that the Haar wavelet has only one vanishing moment, i.e. $\int_0^N t\psi(t)dt \neq 0$.

Theorem 7.8. *Vanishing moments.*

Assume that the filters are chosen so that the scaling functions exist. Then the following hold

- The number of vanishing moments of $\tilde{\psi}$ equals the multiplicity of a zero at $\omega = \pi$ for $\lambda_{G_0}(\omega)$.
- The number of vanishing moments of ψ equals the multiplicity of a zero at $\omega = \pi$ for $\lambda_{H_0}(\omega)$.

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 functions are for approximation of functions. 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. We have that

$$\lambda_{s_{-\tilde{\psi}(-t)}}(\nu) = -\int_{-\infty}^{\infty} \tilde{\psi}(-t) e^{-2\pi i \nu t} dt. \tag{7.13}$$

By differentiating this expression k times w.r.t. ν (differentiate under the integral sign) we get

$$(\lambda_{s_{-\tilde{\psi}(-t)}})^{(k)}(\nu) = -\int (-2\pi i t)^k \tilde{\psi}(t) e^{-2\pi i \nu t} dt. \tag{7.14}$$

Evaluating this at $\nu = 0$ gives

$$(\lambda_{s_{-\tilde{\psi}(-t)}})^{(k)}(0) = -\int (-2\pi i t)^k \tilde{\psi}(t) dt. \tag{7.15}$$

From this expression it is clear that the number of vanishing moments of $\tilde{\psi}$ equals the multiplicity of a zero at $\nu = 0$ for $\lambda_{s_{-\tilde{\psi}(-t)}}(\nu)$, which we have already shown equals the multiplicity of a zero at $\omega = 0$ for $\lambda_{H_1}(\omega)$. Similarly it follows that the number of vanishing moments of ψ equals the multiplicity of a zero at $\omega = 0$ for $\lambda_{G_1}(\omega)$. Since we know that $\lambda_{G_0}(\omega)$ has the same number of zeros at π as $\lambda_{H_1}(\omega)$ has at 0, and $\lambda_{H_0}(\omega)$ has the same number of zeros at π as $\lambda_{G_1}(\omega)$ has at 0, the result follows. □

These results explain how we can construct ϕ, ψ, $\tilde{\phi}$, $\tilde{\psi}$ from FIR-filters H_0, H_1, G_0, G_1 satisfying the perfect reconstruction condition. Also, the results explain how we can obtain such functions with as much differentiability and as many vanishing moments as we want. We will use these results in the next

section to construct interesting wavelets. There we will also cover how we can construct the simplest possible such filters.

There are some details which have been left out in this section: We have not addressed why the wavelet bases we have constructed are linearly independent, and why they span $L^2(\mathbb{R})$. Dual Riesz bases. These details are quite technical, and we refer to [7] for them. Let us also express what we have found in terms of analog filters.

Observation 7.9. *Analog filters.*
Let

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

$c_{m,n}$ and $w_{m,n}$ can be computed by sampling the output of an analog filter. To be more precise,

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

In other words, $c_{m,n}$ can be obtained by sampling $s_{-\tilde{\phi}_{m,0}(-t)}(f(t))$ at the points $2^{-m}n$, $w_{m,n}$ by sampling $s_{-\tilde{\psi}_{m,0}(-t)}(f(t))$ at $2^{-m}n$, where the analog filters $s_{-\tilde{\phi}_{m,0}(-t)}$, $s_{-\tilde{\psi}_{m,0}(-t)}$ were defined in Theorem 1.25, i.e.

$$s_{-\tilde{\phi}_{m,0}(-t)}(f(t)) = \int_0^N (-\tilde{\phi}_{m,0}(-s))f(t-s)ds \tag{7.16}$$
$$s_{-\tilde{\psi}_{m,0}(-t)}(f(t)) = \int_0^N (-\tilde{\psi}_{m,0}(-s))f(t-s)ds. \tag{7.17}$$

A similar statement can be made for $\tilde{f} \in \tilde{V}_m$. 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.

7.3 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 will state the results 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.

7.3.1 Symmetric filters

The main result when the filters are symmetric looks as follows.

Theorem 7.10. *Wavelet criteria.*
Assume that H_0, H_1, G_0, G_1 are the filters of a wavelet, and that

- the filters are symmetric,
- λ_{H_0} has a zero of multiplicity N_1 at π,
- λ_{G_0} has a zero of multiplicity N_2 at π.

Then N_1 and N_2 are even, and there exist a polynomial Q which satisfies

$$u^{(N_1+N_2)/2}Q(1-u) + (1-u)^{(N_1+N_2)/2}Q(u) = 2. \tag{7.18}$$

so that $\lambda_{H_0}(\omega), \lambda_{G_0}(\omega)$ can be written on the form

$$\lambda_{H_0}(\omega) = \left(\frac{1}{2}(1+\cos\omega)\right)^{N_1/2} Q_1\left(\frac{1}{2}(1-\cos\omega)\right) \tag{7.19}$$

$$\lambda_{G_0}(\omega) = \left(\frac{1}{2}(1+\cos\omega)\right)^{N_2/2} Q_2\left(\frac{1}{2}(1-\cos\omega)\right), \tag{7.20}$$

where $Q = Q_1 Q_2$.

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 equations (7.19) and (7.20). With $Q = Q_1Q_2$, Equation (6.25) can now be rewritten as

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

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 Equation (7.18). 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 Section 7.3.3, states that the equation

$$u^N q_1(u) + (1-u)^N q_2(u) = 1 \tag{7.21}$$

has unique solutions q_1, q_2 with $\deg(q_1), \deg(q_2) < (N_1+N_2)/2$. To find these solutions, substituting $1-u$ for u gives the following equations:

$$\begin{aligned}u^N q_1(u) + (1-u)^N q_2(u) &= 1 \\ u^N q_2(1-u) + (1-u)^N q_1(1-u) &= 1,\end{aligned}$$

and uniqueness in Bezouts theorem gives that $q_1(u) = q_2(1-u)$, and $q_2(u) = q_1(1-u)$. Equation (7.21) can thus be stated as

$$u^N q_2(1-u) + (1-u)^N q_2(u) = 1,$$

and comparing with Equation (7.18) (set $N = (N_1 + N_2)/2$) we see that $Q(u) = 2q_2(u)$. $u^N q_1(u) + (1-u)^N q_2(u) = 1$ now gives

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

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

$$Q(u) = 2q_2(u) = 2\sum_{k=0}^{N-1} \binom{N+k-1}{k} u^k. \tag{7.22}$$

Define $Q^{(N)}(u) = 2\sum_{k=0}^{N-1} \binom{N+k-1}{k} u^k$. The first $Q^{(N)}$ are

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

for which we compute

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

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 into polynomials Q_1 and Q_2, and distribute these 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

$$
\begin{aligned}
Q^{(1)}\left(\frac{1}{2}(1-\cos\omega)\right) &= 2 \\
Q^{(2)}\left(\frac{1}{2}(1-\cos\omega)\right) &= \frac{1}{3.7321}(e^{i\omega}-3.7321)(e^{-i\omega}-3.7321) \\
Q^{(3)}\left(\frac{1}{2}(1-\cos\omega)\right) &= \frac{3}{4}\frac{1}{9.4438}(e^{2i\omega}-5.4255e^{i\omega}+9.4438) \\
&\quad \times (e^{-2i\omega}-5.4255e^{-i\omega}+9.4438) \\
Q^{(4)}\left(\frac{1}{2}(1-\cos\omega)\right) &= \frac{5}{8}\frac{1}{3.0407}\frac{1}{7.1495}(e^{i\omega}-3.0407)(e^{2i\omega}-4.0623e^{i\omega}+7.1495) \\
&\quad \times (e^{-i\omega}-3.0407)(e^{-2i\omega}-4.0623e^{-i\omega}+7.1495), \quad (7.23)
\end{aligned}
$$

The factors in these factorizations can be distributed as factors in the frequency responses of $\lambda_{H_0}(\omega)$, and $\lambda_{G_0}(\omega)$. 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. When a frequency response absorbs more factors, the corresponding filter gets more filter coefficients. In the following examples, both factor distribution strategies will be encountered. Note that it is straightforward to use your computer to factor Q into a product of polynomials Q_1 and Q_2. First the `roots` function can be used to find the roots in the polynomials. Then the `conv` function can be used to multiply together factors corresponding to different roots, to obtain the coefficients in the polynomials Q_1 and Q_2.

7.3.2 Orthonormal wavelets

Now we turn to the case of orthonormal wavelets, i.e. where $G_0 = (H_0)^T$, $G_1 = (H_1)^T$. For simplicity we will assume $d = 0, \alpha = -1$ in conditions (6.23) and (6.24) (this corresponded to requiring $\lambda_{H_1}(\omega) = -\overline{\lambda_{H_0}(\omega+\pi)}$ in the definition of alternative QMF filter banks). We will also assume for simplicity that G_0 is *causal*, meaning that $t_{-1}, t_{-2}, \ldots$ all are zero (the other solutions can be derived from this). We saw that the Haar wavelet was such an orthonormal wavelet. We have the following result:

Theorem 7.11. *Criteria for perfect reconstruction.*

Assume that H_0, H_1, G_0, G_1 are the filters of an orthonormal wavelet (i.e. $H_0 = (G_0)^T$ and $H_1 = (G_1)^T$) which also is an alternative QMF filter bank (i.e. $\lambda_{H_1}(\omega) = -\overline{\lambda_{H_0}(\omega+\pi)}$). Assume also that $\lambda_{G_0}(\omega)$ has a zero of multiplicity N at π and that G_0 is causal. Then there exists a polynomial Q which satisfies

$$u^N Q(1-u) + (1-u)^N Q(u) = 2, \tag{7.24}$$

so that if f is another polynomial which satisfies $f(e^{i\omega})f(e^{-i\omega}) = Q\left(\frac{1}{2}(1-\cos\omega)\right)$, $\lambda_{G_0}(\omega)$ can be written on the form

$$\lambda_{G_0}(\omega) = \left(\frac{1+e^{-i\omega}}{2}\right)^N f(e^{-i\omega}), \tag{7.25}$$

We avoided stating $\lambda_{H_0}(\omega)$ in this result, since the relation $H_0 = (G_0)^T$ gives that $\lambda_{H_0}(\omega) = \overline{\lambda_{G_0}(\omega)}$. In particular, $\lambda_{H_0}(\omega)$ also has a zero of multiplicity N at π. That G_0 is causal is included to simplify the expression further.

Proof. The proof is very similar to the proof of Theorem 7.10. N vanishing moments and that G_0 is causal 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 real polynomial. Also

$$\lambda_{H_0}(\omega) = \overline{\lambda_{G_0}(\omega)} = (\cos(\omega/2))^N e^{iN\omega/2} f(e^{i\omega}).$$

Condition (6.25) now says that

$$\begin{aligned} 2 &= \lambda_{G_0}(\omega)\lambda_{H_0}(\omega) + \lambda_{G_0}(\omega+\pi)\lambda_{H_0}(\omega+\pi) \\ &= (\cos^2(\omega/2))^N f(e^{i\omega}) f(e^{-i\omega}) + (\sin^2(\omega/2))^N f(e^{i(\omega+\pi)}) f(e^{-i(\omega+\pi)}). \end{aligned}$$

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

$$\begin{aligned} 2 &= (\cos^2(\omega/2))^N P(\cos\omega) + (\sin^2(\omega/2))^N P(\cos(\omega+\pi)) \\ &= (\cos^2(\omega/2))^N P(1-2\sin^2(\omega/2)) + (\sin^2(\omega/2))^N P(1-2\cos^2(\omega/2)). \end{aligned}$$

If we as in the proof of Theorem 7.10 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 (7.18) 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

$$a_1, \ldots, a_n, \overline{a_1}, \ldots, \overline{a_n} \text{ and } 1/a_1, \ldots, 1/a_n, \overline{1/a_1}, \ldots, \overline{1/a_n},$$

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 the polynomial f by

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

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 pair the roots in many different ways. The new algorithm is thus as follows:

- As before, write $Q\left(\frac{1}{2}(1-\cos\omega)\right)$ as a polynomial in $e^{i\omega}$, and find the roots.
- 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 as above.

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

Clearly the filters obtained with this strategy are not symmetric since f is not symmetric. In Section 7.6 we will take a closer look at wavelets constructed in this way.

7.3.3 The proof of Bezouts theorem

Theorem 7.12. *Existence of polynomials.*

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

$$p_1(x)q_1(x) + p_2(x)q_2(x) = 1. \tag{7.26}$$

Proof. We first establish the existence of q_1, q_2 satisfying Equation (7.26). 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

$$b_{n-1}(x) = a_{n+1}(x)b_n(x) + b_{n+1}(x), \tag{7.27}$$

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 Equation (7.27), 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

$$b_N = a^{(1)}_{N,N-1} b_1 + a^{(2)}_{N,N-1} b_0 = a^{(1)}_{N,N-1} p_2 + a^{(2)}_{N,N-1} p_1, \tag{7.28}$$

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 Equation (7.28) it follows that $q_1 = a^{(2)}_{N,N-1}/b_N$ and $q_2 a^{(1)}_{N,N-1}/b_N$ satisfies Equation (7.26), 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 Equation (7.26). 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. □

Exercise 7.3: Compute filters

Compute the filters H_0, G_0 in Theorem 7.10 when $N = N_1 = N_2 = 4$, and $Q_1 = Q^{(4)}$, $Q_2 = 1$. Compute also filters H_1, G_1 so that we have perfect reconstruction (note that these are not unique).

7.4 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 frequency responses of the filters in the filter factorization are

$$\begin{aligned} \lambda_{H_0}(\omega) &= \left(\frac{1}{2}(1+\cos\omega)\right)^{N_1/2} Q^{(N)}\left(\frac{1}{2}(1-\cos\omega)\right) \\ \lambda_{G_0}(\omega) &= \left(\frac{1}{2}(1+\cos\omega)\right)^{N_2/2}, \end{aligned} \tag{7.29}$$

where $N = (N_1 + N_2)/2$. Since $Q^{(N)}$ has degree $N-1$, 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 (7.3) write

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

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 $\chi_{[-1/2,1/2)}(t)$

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

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 7.5 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 7.13. *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.

7.4.1 The Spline 5/3 wavelet

For the case of $N_1 = N_2 = 2$ when the first design strategy is used, equations (7.19) and (7.20) take the form

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

The filters G_0, H_0 are thus

$$G_0 = \left\{\frac{1}{4}, \underline{\frac{1}{2}}, \frac{1}{4}\right\} \qquad H_0 = \left\{-\frac{1}{4}, \frac{1}{2}, \underline{\frac{3}{2}}, \frac{1}{2}, -\frac{1}{4}\right\}$$

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 6.3. Now, how do we find the filters (G_1, H_1)? Previously we saw how to find the constant α in Theorem 6.16 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 high-pass filters. We thus can set

$$G_1 = \left\{-\frac{1}{4}, -\frac{1}{2}, \underline{\frac{3}{2}}, -\frac{1}{2}, -\frac{1}{4}\right\} \qquad H_1 = \left\{-\frac{1}{4}, \underline{\frac{1}{2}}, -\frac{1}{4}\right\}.$$

We have now found all the filters. It is clear that the forward and reverse filter bank transforms here differ only by multiplication with a constant from those of the the alternative piecewise linear wavelet, so that this gives the same scaling function and mother wavelet as that wavelet.

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. The particular Spline wavelet from Example 7.4.1 is used for lossless compression in the JPEG2000 standard.

Exercise 7.4: Viewing the frequency response

In this exercise we will see how we can view the frequency responses, scaling functions and mother wavelets for any spline wavelet.

a) Plot the frequency responses of the filters of some of the spline wavelets in this section.

Exercise 7.5: Wavelets based on higher degree polynomials

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.

7.5 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 7.6: The CDF 9/7 wavelet

We choose $N_1 = N_2 = 4$. In Equation (7.23) 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

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

The corresponding frequency responses are plotted in Figure 7.1.

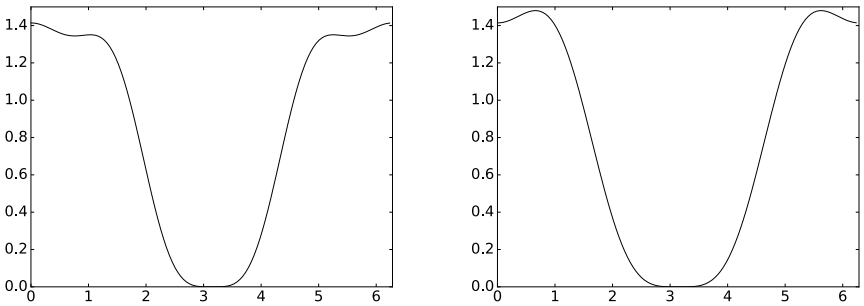

Figure 7.1: The frequency responses $\lambda_{H_0}(\omega)$ (left) and $\lambda_{G_0}(\omega)$ (right) for the CDF 9/7 wavelet.

It is seen that both filters are low-pass 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 6.3 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 cascade algorithm to plot these functions. Note first that since G_0 has 7 filter coefficients, and G_1 has 9 filter coefficients, it follows from Theorem 6.9 that $\text{supp}(\phi) = [-3, 3]$, $\text{supp}(\psi) = [-3, 4]$, $\text{supp}(\tilde{\phi}) = [-4, 4]$, and $\text{supp}(\tilde{\psi}) = [-3, 4]$. The scaling functions and mother wavelets over these supports are shown in Figure 7.2. Again they have irregular shapes, but now at least the functions and dual functions more resemble each other.

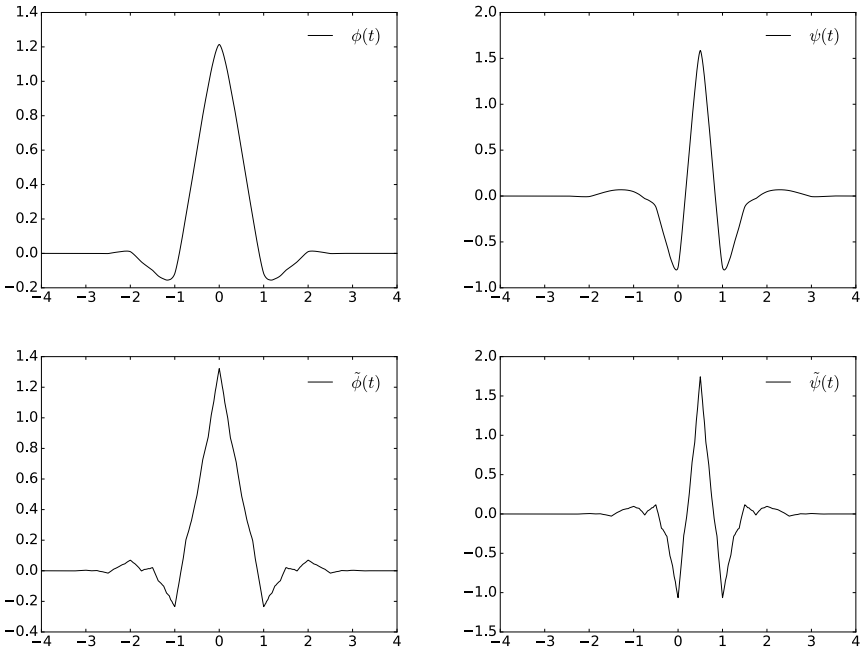

Figure 7.2: Scaling functions and mother wavelets for the CDF 9/7 wavelet.

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:

- Compute the polynomial Q, and find its roots.
- Pair complex conjugate roots into real second degree polynomials, and form polynomials Q_1, Q_2.
- Compute the coefficients in equations (7.19) and (7.20).

7.6 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.

In Theorem 7.11 we characterized orthonormal wavelets where G_0 was causal. All our filters have an even number, say $2L$, of filter coefficients. We can also find an orthonormal wavelet where H_0 has a minimum possible overweight of filter coefficients with negative indices, H_1 a minimum possible overweight of positive indices, i.e. that the filters can be written with the following compact notation:

$$H_0 = \{t_{-L}, \ldots, t_{-1}, \underline{t_0}, t_1, \ldots, t_{L-1}\} \quad H_1 = \{s_{-L+1}, \ldots, s_{-1}, \underline{s_0}, s_1, \ldots, s_L\}. \tag{7.30}$$

To see why, Theorem 6.16 says that we first can shift the filter coefficients of H_0 so that it has this form (we then need to shift G_0 in the opposite direction). H_1, G_1 then can be defined by $\alpha = 1$ and $d = 0$. We will follow this convention for the orthonormal wavelets we look at.

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 (7.23) can be factored further as

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

which gives that $f(e^{i\omega}) = \sqrt{\frac{5}{8}\frac{1}{3.0407}\frac{1}{7.1495}}(e^{3i\omega} - 7.1029e^{2i\omega} + 19.5014^{i\omega} - 21.7391)$. This factorization is not unique, however. This gives the frequency response $\lambda_{G_0}(\omega) = \left(\frac{1+e^{-i\omega}}{2}\right)^N f(e^{-i\omega})$ as

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

which gives the filters

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

so that we get 2, 4, 6 and 8 filter coefficients in $G_0 = (H_0)^T$. We see that the filter coefficients when $N = 1$ are those of the Haar wavelet. The three next filters we have not seen before. The filter $G_1 = (H_1)^T$ can be obtained from the relation $\lambda_{G_1}(\omega) = -\overline{\lambda_{G_0}(\omega + \pi)}$, i.e. by reversing the elements and adding an alternating sign, plus an extra minus sign, so that

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

Frequency responses are shown in Figure 7.3 for $N = 1$ to $N = 6$. It is seen that the frequency responses get increasingly flatter as N increases. The frequency responses are now complex, so their magnitudes are plotted.

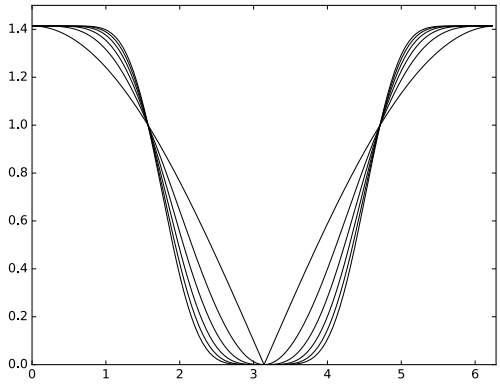

Figure 7.3: The magnitudes $|\lambda_{G_0}(\omega)| = |\lambda_{H_0}(\omega)|$ for the first orthonormal wavelets.

Clearly these filters have low-pass characteristic. We also see that the high-pass characteristics resemble the low-pass characteristics. We also see that the

frequency response gets flatter near the high and low frequencies, as N increases. One can verify that this is the case also when N is increased further. The shapes for higher N are very similar to the frequency responses of those filters used in the MP3 standard (see Figure 3.10). One difference is that the support of the latter is concentrated on a smaller set of frequencies.

The way we have defined the filters, one can show in the same way as in the proof of Theorem 6.9 that, when all filters have $2N$ coefficients, $\phi = \tilde{\phi}$ has support $[-N+1, N]$, $\psi = \tilde{\psi}$ has support $[-N+1/2, N-1/2]$ (i.e. the support of ψ is symmetric about the origin). In particular we have that

- for $N = 2$: $\text{supp}(\phi) = \text{supp}(\psi) = [-1, 2]$,
- for $N = 3$: $\text{supp}(\phi) = \text{supp}(\psi) = [-2, 3]$,
- for $N = 4$: $\text{supp}(\phi) = \text{supp}(\psi) = [-3, 4]$.

The scaling functions and mother wavelets are shown in Figure 7.4. All functions have been plotted over $[-4, 4]$, so that all these support sizes can be verified. Also here we have used the cascade algorithm to approximate the functions.

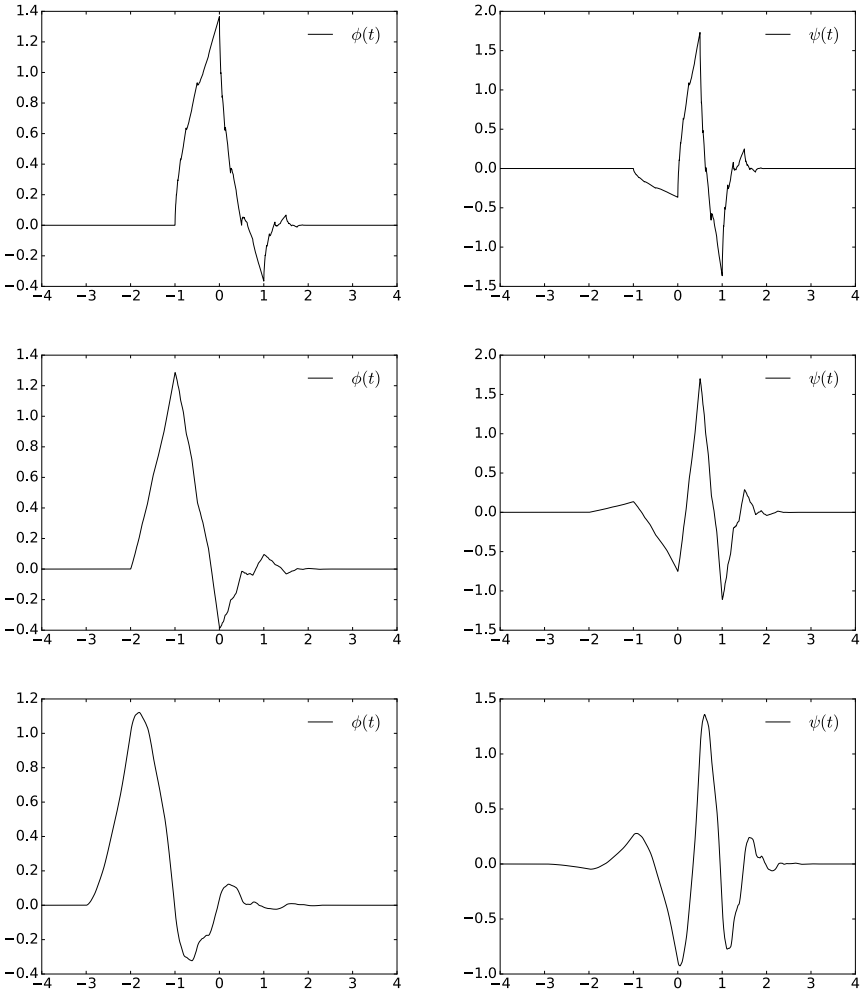

Figure 7.4: The scaling functions and mother wavelets for orthonormal wavelets with N vanishing moments, for different values of N.

7.7 Summary

We started the section by showing how filters from filter bank matrices can give rise to scaling functions and mother wavelets. We saw that we obtained dual function pairs in this way, which satisfied a mutual property called biorthogonality. We then saw how differentiable scaling functions or mother wavelets with vanishing moments could be constructed, and we saw how we could construct the simplest such. These could be found in terms of the frequency responses

of the involved filters. Finally we studied some examples with applications to image compression.

For the wavelets we constructed in this chapter, we also plotted the corresponding scaling functions and mother wavelets (see figures 7.2, 7.4). The importance of these functions are that they are particularly suited for approximation of regular functions, and providing a compact representation of these 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.

In the literature, the orthonormal wavelets with compact support we have constructed were first constructed in [12]. Biorthogonal wavelets were first constructed in [7].

Chapter 8

The polyphase representation and wavelets

In Chapter 6 we saw that we could express wavelet transformations and more general transformations in terms of filters. Through this we obtained intuition for what information the different parts of a wavelet transformation represent, in terms of lowpass and highpass filters. We also obtained some insight into the filters used in the transformation used in the MP3 standard. We expressed the DWT and IDWT implementations in terms of what we called kernel transformations, and these were directly obtained from the filters of the wavelet.

We have looked at many wavelets, however, but have only stated the kernel transformation for the Haar wavelet. In order to use these wavelets in sound and image processing, or in order to use the cascade algorithm to plot the corresponding scaling functions and mother wavelets, we need to make these kernel transformations. This will be one of the goals in this chapter. This will be connected to what we will call the *polyphase representation* of the wavelet. This representation will turn out to be useful for different reasons than the filter representation as well. First of all, with the polyphase representation, transformations can be viewed as block matrices where the blocks are filters. This allows us to prove results in a different way than for filter bank transforms, since we can prove results through block matrix manipulation. There will be two major results we will prove in this way.

First, in Section 8.1 we obtain a factorization of a wavelet transformation into sparse matrices, called elementary lifting matrices. We will show that this factorization reduces the number of arithmetic operations, and also enables us to compute the DWT in-place, in a similar way to how the FFT could be computed in-place after a bit-reversal. This is important: recall that we previously factored a filter into a product of smaller filters which is useful for efficient hardware implementations. But this did not address the fact that only every second component of the filters needs to be evaluated in the DWT, something any efficient implementation of the DWT should take into account.

The factorization into sparse matrices will be called the *lifting factorization*, and it will be clear from this factorization how the wavelet kernels and their duals can be implemented. We will also see how we can use the polyphase representation to prove the remaining parts of Theorem 6.16.

Secondly, in Section 8.3 we will use the polyphase representation to analyze how the forward and reverse filter bank transforms from the MP3 standard can be chosen in order for us to have perfect or near perfect reconstruction. Actually, we will obtain a factorization of the polyphase representation into block matrices also here, and the conditions we need to put on the prototype filters will be clear from this.

8.1 The polyphase representation and the lifting factorization

Let us start by defining the basic concepts in the polyphase representation.

Definition 8.1. *Polyphase components and representation.*

Assume that S is a matrix, and that M is a number. By the *polyphase components* of S we mean the matrices $S^{(i,j)}$ defined by $S^{(i,j)}_{r_1,r_2} = S_{i+r_1M,j+r_2M}$, i.e. the matrices obtained by taking every M'th component of S. By the *polyphase representation* of S we mean the block matrix with entries $S^{(i,j)}$.

The polyphase representation applies in particular for vectors. Since a vector $\boldsymbol{x}$ only has one column, we write $\boldsymbol{x}^{(p)}$ for its polyphase components. As an examples consider the 6×6 MRA-matrix

$$S = \begin{pmatrix} 2 & 3 & 0 & 0 & 0 & 1 \\ 4 & 5 & 6 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 & 0 \\ 0 & 0 & 4 & 5 & 6 & 0 \\ 0 & 0 & 0 & 1 & 2 & 3 \\ 6 & 0 & 0 & 0 & 4 & 5 \end{pmatrix}. \tag{8.1}$$

The polyphase components of S are

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

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

We will mainly be concerned with polyphase representations of MRA matrices. For such matrices we have the following result (this result can be stated more generally for any filter bank transform).

Theorem 8.2. *Similarity.*

When S is an MRA-matrix, the polyphase components $S^{(i,j)}$ are filters (in general different from the filters considered in Chapter 6), i.e. the polyphase representation is a 2×2-block matrix where all blocks are filters. Also, S is similar to its polyphase representation, through a permutation matrix P which places the even-indexed elements first.

To see why, note that when P is the permutation matrix defined above, then PS consists of S with the even-indexed rows grouped first, and since also $SP^T = (PS^T)^T$, SP^T groups the even-indexed columns first. From these observations it is clear that PSP^T is the polyphase representation of S, so that S is similar to its polyphase representation.

We also have the following result on the polyphase representation. This result is easily proved from manipulation with block matrices, and is therefore left to the reader.

Theorem 8.3. *Products and transpose.*

Let A and B be (forward or reverse) filter bank transforms, and denote the corresponding polyphase components by $A^{(i,j)}$, $B^{(i,j)}$. The following hold

- $C = AB$ is also a filter bank transform, with polyphase components $C^{(i,j)} = \sum_k A^{(i,k)} B^{(k,j)}$.
- A^T is also a filter bank transform, with polyphase components $((A^T)^{(i,j)})_{k,l} = (A^{(j,i)})_{l,k}$.

Also, the polyphase components of the identity matrix is the $M \times M$-block matrix with the identity matrix on the diagonal, and $\mathbf{0}$ elsewhere.

To see an application of the polyphase representation, let us prove the final ingredient of Theorem 6.16. We need to prove the following:

Theorem 8.4. *Criteria for perfect reconstruction.*

For any set of FIR filters H_0, H_1, G_0, G_1 which give perfect reconstruction, there exist $\alpha \in \mathbb{R}$ and $d \in \mathbb{Z}$ so that

$$\lambda_{H_1}(\omega) = \alpha^{-1} e^{-2id\omega} \lambda_{G_0}(\omega + \pi) \tag{8.2}$$

$$\lambda_{G_1}(\omega) = \alpha e^{2id\omega} \lambda_{H_0}(\omega + \pi). \tag{8.3}$$

Proof. Let $H^{(i,j)}$ be the polyphase components of H, $G^{(i,j)}$ the polyphase components of G. $GH = I$ means that

$$\begin{pmatrix} G^{(0,0)} & G^{(0,1)} \\ G^{(1,0)} & G^{(1,1)} \end{pmatrix} \begin{pmatrix} H^{(0,0)} & H^{(0,1)} \\ H^{(1,0)} & H^{(1,1)} \end{pmatrix} = \begin{pmatrix} I & \mathbf{0} \\ \mathbf{0} & I \end{pmatrix}.$$

If we here multiply with $\begin{pmatrix} G^{(1,1)} & -G^{(0,1)} \\ -G^{(1,0)} & G^{(0,0)} \end{pmatrix}$ on both sides to the left, or with $\begin{pmatrix} H^{(1,1)} & -H^{(0,1)} \\ -H^{(1,0)} & H^{(0,0)} \end{pmatrix}$ on both sides to the right, we get

$$\begin{pmatrix} G^{(1,1)} & -G^{(0,1)} \\ -G^{(1,0)} & G^{(0,0)} \end{pmatrix} = \begin{pmatrix} (G^{(0,0)}G^{(1,1)} - G^{(1,0)}G^{(0,1)})H^{(0,0)} & (G^{(0,0)}G^{(1,1)} - G^{(1,0)}G^{(0,1)})H^{(0,1)} \\ (G^{(0,0)}G^{(1,1)} - G^{(1,0)}G^{(0,1)})H^{(1,0)} & (G^{(0,0)}G^{(1,1)} - G^{(1,0)}G^{(0,1)})H^{(1,1)} \end{pmatrix}$$
$$\begin{pmatrix} H^{(1,1)} & -H^{(0,1)} \\ -H^{(1,0)} & H^{(0,0)} \end{pmatrix} = \begin{pmatrix} (H^{(0,0)}H^{(1,1)} - H^{(1,0)}H^{(0,1)})G^{(0,0)} & (H^{(0,0)}H^{(1,1)} - H^{(1,0)}H^{(0,1)})G^{(0,1)} \\ (H^{(0,0)}H^{(1,1)} - H^{(1,0)}H^{(0,1)})G^{(1,0)} & (H^{(0,0)}H^{(1,1)} - H^{(1,0)}H^{(0,1)})G^{(1,1)} \end{pmatrix}$$

Now since $G^{(0,0)}G^{(1,1)} - G^{(1,0)}G^{(0,1)}$ and $H^{(0,0)}H^{(1,1)} - H^{(1,0)}H^{(0,1)}$ also are circulant Toeplitz matrices, the expressions above give that

$$\begin{aligned} l(H^{(0,0)}) \leq l(G^{(1,1)}) \leq l(H^{(0,0)}) \\ l(H^{(0,1)}) \leq l(G^{(0,1)}) \leq l(H^{(0,1)}) \\ l(H^{(1,0)}) \leq l(G^{(1,0)}) \leq l(H^{(1,0)}) \end{aligned}$$

so that we must have equality here, and with both

$$G^{(0,0)}G^{(1,1)} - G^{(1,0)}G^{(0,1)} \text{ and } H^{(0,0)}H^{(1,1)} - H^{(1,0)}H^{(0,1)}$$

having only one nonzero diagonal. In particular we can define the diagonal matrix $D = H^{(0,0)}H^{(1,1)} - H^{(0,1)}H^{(1,0)} = \alpha^{-1}E_d$ (for some α, d), and we have that

$$\begin{pmatrix} G^{(0,0)} & G^{(0,1)} \\ G^{(1,0)} & G^{(1,1)} \end{pmatrix} = \begin{pmatrix} \alpha E_{-d}H^{(1,1)} & -\alpha E_{-d}H^{(0,1)} \\ -\alpha E_{-d}H^{(1,0)} & \alpha E_{-d}H^{(0,0)} \end{pmatrix}.$$

The first columns here state a relation between G_0 and H_1, while the second columns state a relation between G_1 and H_0. It is straightforward to show that these relations imply equation (8.2)-(8.3). The details for this can be found in Exercise 8.1. □

In the following we will find factorizations of 2×2-block matrices where the blocks are filters, into simpler such matrices. The importance of Theorem 8.2 is then that MRA-matrices can be written as a product of simpler MRA matrices. These simpler MRA matrices will be called elementary lifting matrices, and will be of the following type.

Definition 8.5. *Elementary lifting matrices.*
A matrix on the form

$$\begin{pmatrix} I & S \\ 0 & I \end{pmatrix}$$

where S is a filter is called an *elementary lifting matrix of even type.* A matrix on the form

$$\begin{pmatrix} I & 0 \\ S & I \end{pmatrix}$$

is called an *elementary lifting matrix of odd type.*

The following are the most useful properties of elementary lifting matrices:

Lemma 8.6. *Lifting lemma.*
The following hold:

$$\begin{pmatrix} I & S \\ 0 & I \end{pmatrix}^T = \begin{pmatrix} I & 0 \\ S^T & I \end{pmatrix}, \text{ and } \begin{pmatrix} I & 0 \\ S & I \end{pmatrix}^T = \begin{pmatrix} I & S^T \\ 0 & I \end{pmatrix},$$

$$\begin{pmatrix} I & S_1 \\ 0 & I \end{pmatrix}\begin{pmatrix} I & S_2 \\ 0 & I \end{pmatrix} = \begin{pmatrix} I & S_1+S_2 \\ 0 & I \end{pmatrix}, \text{ and } \begin{pmatrix} I & 0 \\ S_1 & I \end{pmatrix}\begin{pmatrix} I & 0 \\ S_2 & I \end{pmatrix} = \begin{pmatrix} I & 0 \\ S_1+S_2 & I \end{pmatrix},$$

$$\begin{pmatrix} I & S \\ 0 & I \end{pmatrix}^{-1} = \begin{pmatrix} I & -S \\ 0 & I \end{pmatrix}, \text{ and } \begin{pmatrix} I & 0 \\ S & I \end{pmatrix}^{-1} = \begin{pmatrix} I & 0 \\ -S & I \end{pmatrix}$$

These statements follow directly from Theorem 8.3. 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 states why elementary lifting matrices can be used to factorize general MRA-matrices:

Theorem 8.7. *Multiplying.*
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 with a finite numer of filter coefficients, can be written on the form

$$\Lambda_1 \cdots \Lambda_n \begin{pmatrix} \alpha_0 E_p & 0 \\ 0 & \alpha_1 E_q \end{pmatrix}, \tag{8.4}$$

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

$$\begin{pmatrix} \alpha_0^{-1} E_{-p} & 0 \\ 0 & \alpha_1^{-1} E_{-q} \end{pmatrix} (\Lambda_n)^{-1} \cdots (\Lambda_1)^{-1}. \tag{8.5}$$

Note that $(\Lambda_i)^{-1}$ can be computed with the help of Property 3 of Lemma 8.6.

Proof. The proof will use the concept of the length of a filter, as defined in Definition 3.3. 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 S_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 3.3. If Λ_i is of even type, then the first column in $\Lambda_i S$ is

$$\begin{pmatrix} I & S_i \\ 0 & I \end{pmatrix}\begin{pmatrix} S^{(0,0)} \\ S^{(1,0)} \end{pmatrix} = \begin{pmatrix} S^{(0,0)} + S_i S^{(1,0)} \\ S^{(1,0)} \end{pmatrix}. \tag{8.6}$$

S_i can now be chosen so that $l(S^{(0,0)} + S_i S^{(1,0)}) < l(S^{(1,0)})$. To see how, recall that we in Section 3.1 stated that multiplying filters corresponds to multiplying polynomials. S_i can thus be found from polynomial division with remainder: when we divide $S^{(0,0)}$ by $S^{(1,0)}$, we actually find polynomials S_i and P with $l(P) < l(S^{(1,0)})$ so that $S^{(0,0)} = S_i S^{(1,0)} + P$, so that the length of $P = S^{(0,0)} - S_i S^{(1,0)}$ 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

$$\begin{pmatrix} I & 0 \\ -I & I \end{pmatrix} \begin{pmatrix} I & I \\ 0 & I \end{pmatrix} \begin{pmatrix} 0 & X \\ Y & Z \end{pmatrix} = \begin{pmatrix} Y & X+Z \\ 0 & -X \end{pmatrix}$$

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

$$\Lambda_{n+1} = \begin{pmatrix} I & I \\ 0 & I \end{pmatrix} \text{ and } \Lambda_{n+2} = \begin{pmatrix} I & 0 \\ -I & I \end{pmatrix},$$

so that we always can assume that the second element in the first column is 0, i.e.

$$\Lambda_n \cdots \Lambda_1 S = \begin{pmatrix} P & Q \\ 0 & R \end{pmatrix},$$

for some matrices P, Q, R. From the proof of Theorem 6.16 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

$$\begin{pmatrix} P & Q \\ 0 & R \end{pmatrix}$$

is also invertible, we must thus have that PR must be on the form αE_n. When the filters have a finite number of filter coefficients, the only possibility for this to happen is when $P = \alpha_0 E_p$ 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

$$S = (\Lambda_1)^{-1} \cdots (\Lambda_n)^{-1} \begin{pmatrix} \alpha_0 E_p & Q \\ 0 & \alpha_1 E_q \end{pmatrix}, \tag{8.7}$$

Noting that

$$\begin{pmatrix} \alpha_0 E_p & Q \\ 0 & \alpha_1 E_q \end{pmatrix} = \begin{pmatrix} 1 & \frac{1}{\alpha_1} E_{-q} Q \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \alpha_0 E_p & 0 \\ 0 & \alpha_1 E_q \end{pmatrix},$$

we can rewrite Equation (8.7) as

$$S = (\Lambda_1)^{-1} \cdots (\Lambda_n)^{-1} \begin{pmatrix} 1 & \frac{1}{\alpha_1} E_{-q} Q \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \alpha_0 E_p & 0 \\ 0 & \alpha_1 E_q \end{pmatrix},$$

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 Equation (8.5). This completes the proof. □

Factorizations on the form given by Equation (8.4) will be called *lifting factorizations*. Assume that we have applied Theorem 8.7 in order to get a factorization of the polyphase representation of the DWT kernel of the form

$$\Lambda_n \cdots \Lambda_2 \Lambda_1 H = \begin{pmatrix} \alpha & \mathbf{0} \\ \mathbf{0} & \beta \end{pmatrix}. \tag{8.8}$$

Theorem 8.6 then immediately gives us the following factorizations.

$$H = (\Lambda_1)^{-1} (\Lambda_2)^{-1} \cdots (\Lambda_n)^{-1} \begin{pmatrix} \alpha & \mathbf{0} \\ \mathbf{0} & \beta \end{pmatrix} \tag{8.9}$$

$$G = \begin{pmatrix} 1/\alpha & \mathbf{0} \\ \mathbf{0} & 1/\beta \end{pmatrix} \Lambda_n \cdots \Lambda_2 \Lambda_1 \tag{8.10}$$

$$H^T = \begin{pmatrix} \alpha & \mathbf{0} \\ \mathbf{0} & \beta \end{pmatrix} ((\Lambda_n)^{-1})^T ((\Lambda_{n-1})^{-1})^T \cdots ((\Lambda_1)^{-1})^T \tag{8.11}$$

$$G^T = (\Lambda_1)^T (\Lambda_2)^T \cdots (\Lambda_n)^T \begin{pmatrix} 1/\alpha & \mathbf{0} \\ \mathbf{0} & 1/\beta \end{pmatrix}. \tag{8.12}$$

Since H^T and G^T are the kernel transformations of the dual IDWT and the dual DWT, respectively, these formulas give us recipes for computing the DWT, IDWT, dual IDWT, and the dual DWT, respectively. All in all, everything can be computed by combining elementary lifting steps.

In practice, one starts with a given wavelet with certain proved properties such as the ones from Chapter 7, and applies an algorithm to obtain a lifting factorization of the polyphase representation of the kernels. The algorihtm can easily be written down from the proof of Theorem 8.7. 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. In particular this means that $S^{(0,0)}$ is a symmetric matrix, and that $S^{(1,0)}$ is symmetric about $-1/2$ (see Exercise 8.8).

Assume that we in the proof of Theorem 8.7 add an elementary lifting of even type. At this step we then compute $S^{(0,0)} + S_i S^{(1,0)}$ in the first entry of the first column. Since $S^{(0,0)}$ is now assumed symmetric, $S_i S^{(1,0)}$ must also be symmetric in order for the length to be reduced. And since the filter coefficients of $S^{(1,0)}$ are assumed symmetric about $-1/2$, S_i must be chosen with symmetry around $1/2$.

For most of our wavelets we will consider in the following examples it will turn out the filters in the first column differ in the number of filter coefficients by 1 at all steps. When this is the case, we can choose a filter of length 2 to reduce the length by 2, so that the S_i in an even lifting step can be chosen on the form $S_i = \lambda_i\{\underline{1}, 1\}$. Similarly, for an odd lifting step, S_i can be chosen on the form $S_i = \lambda_i\{1, \underline{1}\}$. Let us summarize this as follows:

Theorem 8.8. *Differing by* 1.

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 and odd type take the simplified form

$$\begin{pmatrix} I & \lambda_i\{\underline{1}, 1\} \\ 0 & I \end{pmatrix} \text{ and } \begin{pmatrix} I & 0 \\ \lambda_i\{1, \underline{1}\} & I \end{pmatrix},$$

respectively.

The lifting steps mentioned in this theorem are quickly computed due to their simple structure.

Each lifting step leaves every second element unchanged, while for the remaining elements, we simply add the two neighbours. Clearly these computations can be computed in-place, without the need for extra memory allocations. From this it is also clear how we can compute the entire DWT/IDWT in-place. We simply avoid the reorganizing into the $(\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1})$-basis until after all the lifting steps. After the application of the matrices above, we have coordinates in the $\mathcal{C}_m$-basis. Here only the coordinates with indices $(0, 2, 4, \ldots)$ need to be further transformed, so the next step in the algorithm should work directly on these. After the next step only the coordinates with indices $(0, 4, 8, \ldots)$ need to be further transformed, and so on. From this it is clear that

- the $\boldsymbol{\psi}_{m-k}$ coordinates are found at indices $2^{k-1} + r2^k$, i.e. the last k bits are 1 followed by $k-1$ zeros.
- the $\boldsymbol{\phi}_0$ coordinates are found at indices $r2^m$, i.e. the last m bits are 0.

If we place the last k bits of the $\boldsymbol{\psi}_{m-k}$-coordinates in front in reverse order, and the the last m bits of the $\boldsymbol{\phi}_0$-coordinates in front, the coordinates have the same order as in the $(\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1})$-basis. This is also called a partial bit-reverse, and is related to the bit-reversal performed in the FFT algorithm.

Clearly, these lifting steps are also MRA-matrices with symmetric filters, so that our procedure factorizes an MRA-matrix with symmetric filters into simpler MRA-matrices which also have symmetric filters.

8.1.1 Reduction in the number of arithmetic operations

The number of arithmetic operations needed to apply matrices on the form stated in Equation (6.10) is easily computed. The number of multiplications is $N/2$ if symmetry is exploited as in Observation 4.20 (N if symmetry is not

exploited). Similarly, the number of additions is N. Let K be the total number of filter coefficients in H_0, H_1. In the following we will see that each lifting step can be chosen to reduce the number of filter coefficients in the MRA matrix by 4, so that a total number of $K/4$ lifting steps are required. Thus, a total number of $KN/8$ ($KN/4$) multiplications, and $KN/4$ additions are required when a lifting factorization is used. In comparison, a direct implementation would require $KN/4$ ($KN/2$) multiplications, and $KN/2$ additions. For the examples we will consider, we therefore have the following result.

Theorem 8.9. *Reducing arithmetic operations.*

The lifting factorization approximately halves the number of additions and multiplications needed, when compared with a direct implementation (regardless of whether symmetry is exploited or not).

Exercise 8.1: The frequency responses of the polyphase components

Let H and G be MRA-matrices for a DWT/IDWT, with corresponding filters H_0, H_1, G_0, G_1, and polyphase components $H^{(i,j)}$, $G^{(i,j)}$.

a) Show that

$$\begin{aligned}
\lambda_{H_0}(\omega) &= \lambda_{H^{(0,0)}}(2\omega) + e^{i\omega}\lambda_{H^{(0,1)}}(2\omega) \\
\lambda_{H_1}(\omega) &= \lambda_{H^{(1,1)}}(2\omega) + e^{-i\omega}\lambda_{H^{(1,0)}}(2\omega) \\
\lambda_{G_0}(\omega) &= \lambda_{G^{(0,0)}}(2\omega) + e^{-i\omega}\lambda_{G^{(1,0)}}(2\omega) \\
\lambda_{G_1}(\omega) &= \lambda_{G^{(1,1)}}(2\omega) + e^{i\omega}\lambda_{G^{(0,1)}}(2\omega).
\end{aligned}$$

b) In the proof of the last part of Theorem 6.16, we defered the last part, namely that equations (8.2)-(8.3) follow from

$$\begin{pmatrix} G^{(0,0)} & G^{(0,1)} \\ G^{(1,0)} & G^{(1,1)} \end{pmatrix} = \begin{pmatrix} \alpha E_{-d} H^{(1,1)} & -\alpha E_{-d} H^{(0,1)} \\ -\alpha E_{-d} H^{(1,0)} & \alpha E_{-d} H^{(0,0)} \end{pmatrix}.$$

Prove this based on the result from a).

Exercise 8.2: Finding new filters

Let S be a filter. Show that

a)

$$G \begin{pmatrix} I & \mathbf{0} \\ S & I \end{pmatrix}$$

is an MRA matrix with filters $\tilde{G}_0, G_1$, where

$$\lambda_{\tilde{G}_0}(\omega) = \lambda_{G_0}(\omega) + \lambda_S(2\omega)e^{-i\omega}\lambda_{G_1}(\omega),$$

b)

$$G\begin{pmatrix} I & S \\ \mathbf{0} & I \end{pmatrix}$$

is an MRA matrix with filters $G_0, \tilde{G}_1$, where

$$\lambda_{\tilde{G}_1}(\omega) = \lambda_{G_1}(\omega) + \lambda_S(2\omega)e^{i\omega}\lambda_{G_0}(\omega),$$

c)

$$\begin{pmatrix} I & \mathbf{0} \\ S & I \end{pmatrix} H$$

is an MRA-matrix with filters $H_0, \tilde{H}_1$, where

$$\lambda_{\tilde{H}_1}(\omega) = \lambda_{H_1}(\omega) + \lambda_S(2\omega)e^{-i\omega}\lambda_{H_0}(\omega).$$

d)

$$\begin{pmatrix} I & S \\ \mathbf{0} & I \end{pmatrix} H$$

is an MRA-matrix with filters $\tilde{H}_0, H_1$, where

$$\lambda_{\tilde{H}_0}(\omega) = \lambda_{H_0}(\omega) + \lambda_S(2\omega)e^{i\omega}\lambda_{H_1}(\omega).$$

In summary, this exercise shows that one can think of the steps in the lifting factorization as altering one of the filters of an MRA-matrix in alternating order.

Exercise 8.3: Relating to the polyphase components

Show that S is a filter of length kM if and only if the entries $\{S^{i,j}\}_{i,j=0}^{M-1}$ in the polyphase representation of S satisfy $S^{(i+r) \bmod M,(j+r) \bmod M} = S_{i,j}$. In other words, S is a filter if and only if the polyphase representation of S is a "block-circulant Toeplitz matrix". This implies a fact that we will use: GH is a filter (and thus provides alias cancellation) if blocks in the polyphase representations repeat cyclically as in a Toeplitz matrix (in particular when the matrix is block-diagonal with the same block repeating on the diagonal).

Exercise 8.4: QMF filter banks

Recall from Definition 6.18 that we defined a classical QMF filter bank as one where $M = 2$, $G_0 = H_0$, $G_1 = H_1$, and $\lambda_{H_1}(\omega) = \lambda_{H_0}(\omega + \pi)$. Show that the forward and reverse filter bank transforms of a classical QMF filter bank take the form

$$H = G = \begin{pmatrix} A & -B \\ B & A \end{pmatrix}$$

Exercise 8.5: Alternative QMF filter banks

Recall from Definition 6.19 that we defined an alternative QMF filter bank as one where $M = 2$, $G_0 = (H_0)^T$, $G_1 = (H_1)^T$, and $\lambda_{H_1}(\omega) = \overline{\lambda_{H_0}(\omega + \pi)}$. Show that the forward and reverse filter bank transforms of an alternative QMF filter bank take the form.

$$H = \begin{pmatrix} A^T & B^T \\ -B & A \end{pmatrix} \qquad G = \begin{pmatrix} A & -B^T \\ B & A^T \end{pmatrix} = \begin{pmatrix} A^T & B^T \\ -B & A \end{pmatrix}^T.$$

Exercise 8.6: Alternative QMF filter banks with additional sign

Consider alternative QMF filter banks where we take in an additional sign, so that $\lambda_{H_1}(\omega) = -\overline{\lambda_{H_0}(\omega + \pi)}$ (the Haar wavelet was an example of such a filter bank). Show that the forward and reverse filter bank transforms now take the form

$$H = \begin{pmatrix} A^T & B^T \\ B & -A \end{pmatrix} \qquad G = \begin{pmatrix} A & B^T \\ B & -A^T \end{pmatrix} = \begin{pmatrix} A^T & B^T \\ B & -A \end{pmatrix}^T.$$

It is straightforward to check that also these satisfy the alias cancellation condition, and that the perfect reconstruction condition also here takes the form $|\lambda_{H_0}(\omega)|^2 + |\lambda_{H_0}(\omega + \pi)|^2 = 2$.

8.2 Examples of lifting factorizations

We have seen that the polyphase representations of wavelet kernels can be factored into a product of elementary lifting matrices. In this section we will compute the exact factorizations for the wavelets we have considered. In the exercises we will then complete the implementations, so that we can make actual experiments, such as listening to the low-resolution approximations in sound, or using the cascade algorithm to plot scaling functions and mother wavelets. We will omit the Haar wavelet. One can easily write down a lifting factorization for this as well, but there is little to save in this factorization when compared to the direct form of this we already have considered.

First we will consider the two piecewise linear wavelets we have looked at. It turns out that their lifting factorizations can be obtained in a direct way by considering the polyphase representations as a change of coordinates. To see how, we first define

$$\mathcal{D}_m = \{\phi_{m,0}, \phi_{m,2}, \phi_{m,4} \ldots, \phi_{m,1}, \phi_{m,3}, \phi_{m,5}, \ldots\}, \tag{8.13}$$

$P_{\mathcal{D}_m \leftarrow \boldsymbol{\phi}_m}$ is clearly the permutation matrix P used in the similarity between a matrix and its polyphase representation. Let now H and G be the kernel transformations of a wavelet. The polyphase representation of H is

$$PHP^T = P_{\mathcal{D}_m \leftarrow \boldsymbol{\phi}_m} P_{\mathcal{C}_m \leftarrow \boldsymbol{\phi}_m} P_{\boldsymbol{\phi}_m \leftarrow \mathcal{D}_m} = P_{(\boldsymbol{\phi}_1, \boldsymbol{\psi}_1) \leftarrow \boldsymbol{\phi}_m} P_{\boldsymbol{\phi}_m \leftarrow \mathcal{D}_m} = P_{(\boldsymbol{\phi}_1, \boldsymbol{\psi}_1) \leftarrow \mathcal{D}_m}.$$

Taking inverses here we obtain that $PGP^T = P_{\mathcal{D}_m \leftarrow (\boldsymbol{\phi}_1, \boldsymbol{\psi}_1)}$. We therefore have the following result:

Theorem 8.10. *The polyphase representation.*

The polyphase representation of H equals the change of coordinates matrix $P_{(\boldsymbol{\phi}_1, \boldsymbol{\psi}_1) \leftarrow \mathcal{D}_m}$, and the polyphase representation of G equals the change of coordinates matrix $P_{\mathcal{D}_m \leftarrow (\boldsymbol{\phi}_1, \boldsymbol{\psi}_1)}$.

8.2.1 The piecewise linear wavelet

The polyphase representation of G is $\frac{1}{\sqrt{2}} \begin{pmatrix} I & \mathbf{0} \\ \frac{1}{2}\{1, \underline{1}\} & I \end{pmatrix}$. Due to Theorem 8.6, the polyphase representation of H is $\sqrt{2} \begin{pmatrix} I & \mathbf{0} \\ -\frac{1}{2}\{1, \underline{1}\} & I \end{pmatrix}$ We can summarize that the polyphase representations of the kernels H and G for the piecewise linear wavelet are

$$\sqrt{2} \begin{pmatrix} I & \mathbf{0} \\ -\frac{1}{2}\{1, \underline{1}\} & I \end{pmatrix} \text{ and } \frac{1}{\sqrt{2}} \begin{pmatrix} I & \mathbf{0} \\ \frac{1}{2}\{1, \underline{1}\} & I \end{pmatrix}, \tag{8.14}$$

respectively.

Example 8.7: Lifting factorization of the alternative piecewise linear wavelet

The polyphase representation of H is

$$\sqrt{2} \begin{pmatrix} I & \frac{1}{4}\{\underline{1}, 1\} \\ \mathbf{0} & I \end{pmatrix} \begin{pmatrix} I & \mathbf{0} \\ -\frac{1}{2}\{1, \underline{1}\} & I \end{pmatrix}.$$

In this case we required one additional lifting step. We can thus conclude that the polyphase representations of the kernels H and G for the alternative piecewise linear wavelet are

$$\sqrt{2} \begin{pmatrix} I & \frac{1}{4}\{\underline{1}, 1\} \\ \mathbf{0} & I \end{pmatrix} \begin{pmatrix} I & \mathbf{0} \\ -\frac{1}{2}\{1, \underline{1}\} & I \end{pmatrix} \text{ and } \frac{1}{\sqrt{2}} \begin{pmatrix} I & \mathbf{0} \\ \frac{1}{2}\{1, \underline{1}\} & I \end{pmatrix} \begin{pmatrix} I & -\frac{1}{4}\{\underline{1}, 1\} \\ \mathbf{0} & I \end{pmatrix}, \tag{8.15}$$

respectively.

8.2.2 The Spline 5/3 wavelet

Let us consider the Spline 5/3 wavelet, which we defined in Example 7.4.1. Let us start by looking at, and we recall that

$$H_0 = \left\{-\frac{1}{4}, \frac{1}{2}, \underline{\frac{3}{2}}, \frac{1}{2}, -\frac{1}{4}\right\} \qquad H_1 = \left\{-\frac{1}{4}, \underline{\frac{1}{2}}, -\frac{1}{4}\right\}.$$

from which we see that the polyphase components of H are

$$\begin{pmatrix} H^{(0,0)} & H^{(0,1)} \\ H^{(1,0)} & H^{(1,1)} \end{pmatrix} = \begin{pmatrix} \{-\frac{1}{4}, \underline{\frac{3}{2}}, -\frac{1}{4}\} & \frac{1}{2}\{\underline{1}, 1\} \\ -\frac{1}{4}\{1, \underline{1}\} & \frac{1}{2}I \end{pmatrix}$$

We see here that the upper filter has most filter coefficients in the first column, so that we must start with an elementary lifting of even type. We need to find a filter S_1 so that $S_1\{-1/4, \underline{-1/4}\} + \{-1/4, \underline{3/2}, -1/4\}$ has fewer filter coefficients than $\{-1/4, \underline{3/2}, -1/4\}$. It is clear that we can choose $S_1 = \{\underline{-1}, -1\}$, and that

$$\Lambda_1 H = \begin{pmatrix} I & \{\underline{-1}, -1\} \\ \mathbf{0} & I \end{pmatrix} \begin{pmatrix} \{-\frac{1}{4}, \underline{\frac{3}{2}}, -\frac{1}{4}\} & \frac{1}{2}\{\underline{1}, 1\} \\ -\frac{1}{4}\{1, \underline{1}\} & \frac{1}{2}I \end{pmatrix} = \begin{pmatrix} 2I & 0 \\ -\frac{1}{4}\{1, \underline{1}\} & \frac{1}{2}I \end{pmatrix}$$

Now we need to apply an elementary lifting of odd type, and we need to find a filter S_2 so that $S_2 I - \frac{1}{4}\{1, \underline{1}\} = \mathbf{0}$. Clearly we can choose $S_2 = \{1/8, \underline{1/8}\}$, and we get

$$\Lambda_2 \Lambda_1 H = \begin{pmatrix} I & \mathbf{0} \\ \frac{1}{8}\{1, \underline{1}\} & I \end{pmatrix} \begin{pmatrix} 2I & 0 \\ -\frac{1}{4}\{1, \underline{1}\} & \frac{1}{2}I \end{pmatrix} = \begin{pmatrix} 2I & \mathbf{0} \\ \mathbf{0} & \frac{1}{2}I \end{pmatrix}.$$

Multiplying with inverses of elementary lifting steps, we now obtain that the polyphase representations of the kernels for the Spline 5/3 wavelet are

$$H = \begin{pmatrix} I & \{\underline{1}, 1\} \\ \mathbf{0} & I \end{pmatrix} \begin{pmatrix} I & \mathbf{0} \\ -\frac{1}{8}\{1, \underline{1}\} & I \end{pmatrix} \begin{pmatrix} 2I & \mathbf{0} \\ \mathbf{0} & \frac{1}{2}I \end{pmatrix}$$

and

$$G = \begin{pmatrix} \frac{1}{2}I & \mathbf{0} \\ \mathbf{0} & 2I \end{pmatrix} \begin{pmatrix} I & \mathbf{0} \\ \frac{1}{8}\{1, \underline{1}\} & I \end{pmatrix} \begin{pmatrix} I & \{\underline{-1}, -1\} \\ \mathbf{0} & I \end{pmatrix},$$

respectively. Two lifting steps are thus required. We also see that the lifting steps involve only dyadic fractions, just as the filter coefficients did. This means that the lifting factorization also can be used for lossless operations.

8.2.3 The CDF 9/7 wavelet

For the wavelet we considered in Example 7.6, 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 8.7. 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 that the polyphase representations of the kernels H and G for the CDF 9/7 wavelet are

$$\begin{pmatrix} I & 0.5861\{\underline{1},1\} \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0.6681\{1,\underline{1}\} & I \end{pmatrix} \begin{pmatrix} I & -0.0700\{\underline{1},1\} \\ 0 & I \end{pmatrix} \times \begin{pmatrix} I & 0 \\ -1.2002\{1,\underline{1}\} & I \end{pmatrix} \begin{pmatrix} -1.1496 & 0 \\ 0 & -0.8699 \end{pmatrix} \text{ and}$$

$$\begin{pmatrix} -0.8699 & 0 \\ 0 & -1.1496 \end{pmatrix} \begin{pmatrix} I & 0 \\ 1.2002\{1,\underline{1}\} & I \end{pmatrix} \begin{pmatrix} I & 0.0700\{\underline{1},1\} \\ 0 & I \end{pmatrix} \times \begin{pmatrix} I & 0 \\ -0.6681\{1,\underline{1}\} & I \end{pmatrix} \begin{pmatrix} I & -0.5861\{\underline{1},1\} \\ 0 & I \end{pmatrix},$$

respectively. In this case four lifting steps were required.

Perhaps more important than the reduction in the number of arithmetic operations is the fact that the lifting factorization splits the DWT and IDWT into simpler components, each very attractive for hardware implementations since a lifting step only requires the additional value λ_i from Theorem 8.8. Lifting actually provides us with a complete implementation strategy for the DWT and IDWT, in which the λ_i are used as precomputed values.

Finally we will find a lifting factorization for orthonormal wavelets. Note that here the filters H_0 and H_1 are not symmetric, and each of them has an even number of filter coefficients. There are thus a different number of filter coefficients with positive and negative indices, and in Section 7.6 we defined the filters so that the filter coefficients were as symmetric as possible when it came to the number of nonzero filter coefficients with positive and negative indices.

8.2.4 Orthonormal wavelets

We will attempt to construct a lifting factorization where the following property is preserved after each lifting step:

P1: $H^{(0,0)}$, $H^{(1,0)}$ have a minimum possible overweight of filter coefficients with negative indices.

This property stems from the assumption in Section 7.6 that H_0 is assumed to have a minimum possible overweight of filter coefficients with negative indices. To see that this holds at the start, assume as before that all the filters have $2L$ nonzero filter coefficients, so that $H0$ and H_1 are on the form given by Equation (7.30). Assume first that L is even. It is clear that

$$
\begin{aligned}
H^{(0,0)} &= \{t_{-L}, \ldots, t_{-2}, \underline{t_0}, t_2, \ldots, t_{L-2}\} \\
H^{(0,1)} &= \{t_{-L+1}, \ldots, t_{-3}, \underline{t_{-1}}, t_1, \ldots, t_{L-1}\} \\
H^{(1,0)} &= \{s_{-L+1}, \ldots, s_{-1}, \underline{s_1}, s_3, \ldots, s_{L-1}\} \\
H^{(1,1)} &= \{s_{-L+2}, \ldots, s_{-2}, \underline{s_0}, s_2, \ldots, s_L\}.
\end{aligned}
$$

Clearly P1 holds. Assume now that L is odd. It is now clear that

$$
\begin{aligned}
H^{(0,0)} &= \{t_{-L+1}, \ldots, t_{-2}, \underline{t_0}, t_2, \ldots, t_{L-1}\} \\
H^{(0,1)} &= \{t_{-L}, \ldots, t_{-3}, \underline{t_{-1}}, t_1, \ldots, t_{L-2}\} \\
H^{(1,0)} &= \{s_{-L+2}, \ldots, s_{-1}, \underline{s_1}, s_3, \ldots, s_L\} \\
H^{(1,1)} &= \{s_{-L+1}, \ldots, s_{-2}, \underline{s_0}, s_2, \ldots, s_{L-1}\}.
\end{aligned}
$$

In this case it is seen that all filters have equally many filter coefficients with positive and negative indices, so that P1 holds also here.

Now let us turn to the first lifting step. We will choose it so that the number of filter coefficients in the first column is reduced with 1, and so that $H^{(0,0)}$ has an odd number of coefficients. If L is even, we saw that $H^{(0,0)}$ and $H^{(1,0)}$ had an even number of coefficients, so that the first lifting step must be even. To preserve P1, we must cancel t_{-L}, so that the first lifting step is

$$
\Lambda_1 = \begin{pmatrix} I & -t_{-L}/s_{-L+1} \\ \mathbf{0} & I \end{pmatrix}.
$$

If L is odd, we saw that $H^{(0,0)}$ and $H^{(1,0)}$ had an odd number of coefficients, so that the first lifting step must be odd. To preserve P1, we must cancel s_L, so that the first lifting step is

$$
\Lambda_1 = \begin{pmatrix} I & \mathbf{0} \\ -s_L/t_{L-1} & I \end{pmatrix}.
$$

Now that we have a difference of one filter coefficent in the first column, we will reduce the entry with the most filter coefficients with two with a lifting step, until we have $H^{(0,0)} = \{\underline{K}\}$, $H^{(1,0)} = 0$ in the first column.

Assume first that $H^{(0,0)}$ has the most filter coefficients. We then need to apply an even lifting step. Before an even step, the first column has the form

$$
\begin{pmatrix} \{t_{-k}, \ldots, t_{-1}, \underline{t_0}, t_1, \ldots, t_k\} \\ \{s_{-k}, \ldots, s_{-1}, s_0, s_1, \ldots, s_{k-1}\} \end{pmatrix}.
$$

We can then choose $\Lambda_i = \begin{pmatrix} I & \underline{\{-t_{-k}/s_{-k}, -t_k/s_{k-1}\}} \\ \mathbf{0} & I \end{pmatrix}$ as a lifting step.

Assume then that $H^{(1,0)}$ has the most filter coefficients. We then need to apply an odd lifting step. Before an odd step, the first column has the form

$$\begin{pmatrix} \{t_{-k}, \ldots, t_{-1}, \underline{t_0}, t_1, \ldots, t_k\} \\ \{s_{-k-1}, \ldots, s_{-1}, s_0, s_1, \ldots, s_k\} \end{pmatrix}.$$

We can then choose $\Lambda_i = \begin{pmatrix} I & \mathbf{0} \\ \{-s_{-k-1}/t_{-k}, \underline{-s_k/t_k}\} & I \end{pmatrix}$ as a lifting step.

If L is even we end up with a matrix on the form $\begin{pmatrix} \alpha & \{\underline{0}, K\} \\ \mathbf{0} & \beta \end{pmatrix}$, and we can choose the final lifting step as $\Lambda_n = \begin{pmatrix} I & \{\underline{0}, -K/\beta\} \\ \mathbf{0} & I \end{pmatrix}$.

If L is odd we end up with a matrix on the form

$$\begin{pmatrix} \alpha & K \\ \mathbf{0} & \beta \end{pmatrix},$$

and we can choose the final lifting step as $\Lambda_n = \begin{pmatrix} I & -K/\beta \\ \mathbf{0} & I \end{pmatrix}$. Again using equations (8.9)-(8.10), this gives us the lifting factorizations.

In summary we see that all even and odd lifting steps take the form $\begin{pmatrix} I & \{\underline{\lambda_1}, \lambda_2\} \\ \mathbf{0} & I \end{pmatrix}$ and $\begin{pmatrix} I & \mathbf{0} \\ \lambda_1, \underline{\lambda_2}\} & I \end{pmatrix}$. We see that symmetric lifting steps correspond to the special case when $\lambda_1 = \lambda_2$. The even and odd lifting matrices now used are

$$\begin{pmatrix} 1 & \lambda_1 & 0 & 0 & \cdots & 0 & 0 & \lambda_2 \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \lambda_2 & 1 & \lambda_1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \lambda_2 & 1 & \lambda_1 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix} \text{ and } \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \lambda_2 & 1 & \lambda_1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ \lambda_1 & 0 & 0 & 0 & \cdots & 0 & \lambda_2 & 1 \end{pmatrix}, \tag{8.16}$$

respectively. We note that when we reduce elements to the left and right in the upper and lower part of the first column, the same type of reductions must occur in the second column, since the determinant $H^{(0,0)}H^{(1,1)} - H(0,1)H^{(1,0)}$ is a constant after any number of lifting steps.

This example explains the procedure for finding the lifting factorization into steps of the form given in Equation (8.16). You will be spared the details of writing an implementation which applies this procedure. In order to use orthornormal wavelets in implementations, we have implemented a function `liftingfactortho`, which takes N as input, and computes the steps in a lifting factorization so that (8.8) holds. These are written to file, and read from file when needed (you need not call `liftingfactortho` yourself, this is handled behind the curtains). In the exercises, you will be asked to implement both these non-symmetric elementary lifting steps, as well as the full kernel transformations for orthonormal wavelets.

Exercise 8.8: Polyphase components for symmetric filters

Assume that the filters H_0, H_1 of a wavelet are symmetric, and denote by $S^{(i,j)}$ the polyphase components of the corresponding MRA-matrix H. Show that $S^{(0,0)}$ and $S^{(1,1)}$ are symmetric filters, that the filter coefficients of $S^{(1,0)}$ has symmetry about $-1/2$, and that $S^{(0,1)}$ has symmetry about $1/2$. Also show a similar statement for the MRA-matrix G of the inverse DWT.

Exercise 8.9: Implementing kernels transformations using lifting

Up to now in this chapter we have obtained lifting factorizations for four different wavelets where the filters are symmetric. Let us now implement the kernel transformations for these wavelets. Your functions should call the functions from Exercise 5.33 and Exercise 5.37. in order to compute the individual lifting steps. Recall that the kernel transformations should take the input vector `x`, `symm` (i.e. whether symmetric extension should be applied), and `dual` (i.e. whether the dual wavelet transform should be applied) as input. You will need equations (8.9)-(8.12) here, in order to complete the kernels for bot the transformations and the dual transformations.

a) Write functions

```
dwt_kernel_53(x, bd_mode)
idwt_kernel_53(x, bd_mode)
```

which implement the DWT and IDWT kernel transformations for the Spline 5/3 wavelet. Use the lifting factorization obtained in Example 8.2.2.

b) Write functions

```
dwt_kernel_97(x, bd_mode)
idwt_kernel_97(x, bd_mode)
```

which implement the DWT and IDWT kernel transformations for the CDF 9/7 wavelet. Use the lifting factorization obtained in Example 8.2.3.

c) In Chapter 5, we listened to the low-resolution approximations and detail components in sound for three different wavelets. Repeat these experiments with the Spline 5/3 and the CDF 9/7 wavelet, using the new kernels we have implemented in this exercise.

d) Plot all scaling functions and mother wavelets for the Spline 5/3 and the CDF 9/7 wavelets, using the cascade algorithm and the kernels you have implemented.

Exercise 8.10: Lifting orthonormal wavelets

In this exercise we will implement the kernel transformations for orthonormal wavelets.

a) Write functions

```
lifting_even(lambda1, lambda2, x, bd_mode)
lifting_odd(lambda1, lambda2, x, bd_mode)
```

which apply the elementary lifting matrices (8.16) to $\boldsymbol{x}$. Assume that N is even.

b) Write functions

```
dwt_kernel_ortho(x, filters, bd_mode)
idwt_kernel_ortho(x, filters, bd_mode)
```

which apply the DWT and IDWT kernel transformations for orthonormal wavelets to $\boldsymbol{x}$. You should call the functions `lifting_even` and `lifting_odd`. You can assume that you can access the lifting steps so that the lifting factorization (8.8) holds, through the object `filters` by writing `filters.lambdas`, `filters.alpha`, and `filters.beta`. `filters.lambdas` is an $n \times 2$-matrix so that the filter coefficients $\{\underline{\lambda_1}, \lambda_2\}$ or $\{\lambda_1, \underline{\lambda_2}\}$ in the i'th lifting step is found in row i. Recall that the last lifting step was even.

Due to the `filters` object, the functions `dwt_kernel_ortho` and `idwt_kernel_ortho` do not abide to the signature we have required for kernel functions up to now. The code base creates such functions based on the functions above in the following way:

```
filters = ...
dwt_kernel = lambda x, bd_mode: dwt_kernel_ortho(x, filters, bd_mode)
```

c) Listen to the low-resolution approximations and detail components in sound for orthonormal wavelets for $N = 1, 2, 3, 4$.

d) Plot all scaling functions and mother wavelets for the orthonormal wavelets for $N = 1, 2, 3, 4$, using the cascade algorithm. Since the wavelets are orthonormal, we should have that $\phi = \tilde{\phi}$, and $\psi = \tilde{\psi}$. In other words, you should see that the bottom plots equal the upper plots.

Exercise 8.11: 4 vanishing moments

In Exercise 5.39 we found constants $\alpha, \beta, \gamma, \delta$ which give the coordinates of $\hat{\psi}$ in $(\boldsymbol{\phi}_1, \hat{\boldsymbol{\psi}}_1)$, where $\hat{\psi}$ had four vanishing moments, and where we worked with the multiresolution analysis of piecewise constant functions.

a) Show that the polyphase representation of G when $\hat{\psi}$ is used as mother wavelet can be factored as

$$\frac{1}{\sqrt{2}} \begin{pmatrix} I & \mathbf{0} \\ \{1/2, \underline{1/2}\} & I \end{pmatrix} \begin{pmatrix} I & \{-\gamma, \underline{-\alpha}, -\beta, -\delta\} \\ \mathbf{0} & I \end{pmatrix}. \tag{8.17}$$

You here need to reconstruct what you did in the lifting factorization for the alternative piecewise linear wavelet, i.e. write

$$P_{\mathcal{D}_1 \leftarrow (\phi_1, \hat{\psi}_1)} = P_{\mathcal{D}_1 \leftarrow (\phi_1, \psi_1)} P_{(\phi_1, \psi_1) \leftarrow (\phi_1, \hat{\psi}_1)}.$$

By inversion, find also a lifting factorization of H.

Exercise 8.12: Wavelet based on piecewise quadratic scaling function

In Exercise 7.3 you should have found the filters

$$\begin{aligned}
H_0 &= \frac{1}{128}\{-5, 20, -1, -96, 70, \underline{280}, 70, -96, -1, 20, -5\} \\
H_1 &= \frac{1}{16}\{1, -4, \underline{6}, -4, 1\} \\
G_0 &= \frac{1}{16}\{1, 4, \underline{6}, 4, 1\} \\
G_1 &= \frac{1}{128}\{5, 20, 1, -96, -70, \underline{280}, -70, -96, 1, 20, 5\}.
\end{aligned}$$

a) Show that

$$\begin{pmatrix} I & -\frac{1}{128}\{5, \underline{-29}, -29, 5\} \\ \mathbf{0} & \underline{I} \end{pmatrix} \begin{pmatrix} I & \mathbf{0} \\ -\{1, \underline{1}\} & I \end{pmatrix} \begin{pmatrix} I & -\frac{1}{4}\{\underline{1}, 1\} \\ \mathbf{0} & I \end{pmatrix} G = \begin{pmatrix} \frac{1}{4} & \mathbf{0} \\ \mathbf{0} & 4 \end{pmatrix}.$$

From this we can easily derive the lifting factorization of G.

b) Listen to the low-resolution approximations and detail components in sound for this wavelet.

c) Plot all scaling functions and mother wavelets for this wavelet, using the cascade algorithm.

8.3 Cosine-modulated filter banks and the MP3 standard

Previously we saw that the MP3 standard used a certain filter bank, called a cosine-modulated filter bank. We also illustrated that, surprisingly for a much used international standard, the synthesis system did not exactly invert the analysis system, i.e. we do not have perfect reconstruction, only "near-perfect reconstruction". In this section we will first explain how this filter bank can be constructed, and why it can not give perfect reconstruction. In particular it will be clear how the prototype filter can be constructed. We will then construct a very similar filter bank, which actually can give perfect reconstruction. It may seem very surprising that the MP3 standard does not use this filter bank instead due to this. The explanation may lie in that the MP3 standard was established at about the same time as these filter banks were developed, so that the standard did not capture this very similar filter bank with perfect reconstruction.

8.3.1 Polyphase representations of the filter bank transforms

The main idea is to find the polyphase representations of the forward and reverse filter bank transforms of the MP3 standard. We start with the expression

$$z_{32(s-1)+n} = \sum_{k=0}^{511} \cos((n+1/2)(k-16)\pi/32) h_k x_{32s-k-1}, \tag{8.18}$$

which lead to the expression of the forward filter bank transform (Theorem 6.24). Using that any $k < 512$ can be written uniquely on the form $k = m + 64r$, where $0 \leq m < 64$, and $0 \leq r < 8$, we can rewrite this as

$$\begin{aligned} &= \sum_{m=0}^{63} \sum_{r=0}^{7} (-1)^r \cos\left(2\pi(n+1/2)(m-16)/64\right) h_{m+64r} x_{32s-(m+64r)-1} \\ &= \sum_{m=0}^{63} \cos\left(2\pi(n+1/2)(m-16)/64\right) \sum_{r=0}^{7} (-1)^r h_{m+32\cdot 2r} x_{32(s-2r)-m-1}. \end{aligned}$$

Here we also used Property (6.35). If we write

$$V^{(m)} = \{\underline{(-1)^0 h_m}, 0, (-1)^1 h_{m+64}, 0, (-1)^2 h_{m+128}, \ldots, (-1)^7 h_{m+7\cdot 64}, 0\}, \tag{8.19}$$

for $0 \leq m \leq 63$, and we can write the expression above as

$$\begin{aligned} &\sum_{m=0}^{63} \cos\left(2\pi(n+1/2)(m-16)/64\right) \sum_{r=0}^{15} V_r^{(m)} x_{32(s-r)-m-1} \\ &= \sum_{m=0}^{63} \cos\left(2\pi(n+1/2)(m-16)/64\right) \sum_{r=0}^{15} V_r^{(m)} \boldsymbol{x}_{s-1-r}^{(32-m-1)} \\ &= \sum_{m=0}^{63} \cos\left(2\pi(n+1/2)(m-16)/64\right) (V^{(m)} \boldsymbol{x}^{(32-m-1)})_{s-1}, \end{aligned}$$

where we recognized $x_{32(s-r)-m-1}$ in terms of the polyphase components of $\boldsymbol{x}$, and the inner sum as a convolution. We remark that the inner terms $\{(V^{(m)}\boldsymbol{x}^{(32-m-1)})_{s-1}\}_{m=0}^{63}$ here are what the standard calls partial calculations (windowing refers to multiplication with the combined set of filter coefficients of the $V^{(m)}$), and that matrixing here represents the multiplication with the cosine entries. Since $\boldsymbol{z}^{(n)} = \{z_{32(s-1)+n}\}_{s=0}^{\infty}$ is the n'th polyphase component of $\boldsymbol{z}$, this can be written as

$$\boldsymbol{z}^{(n)} = \sum_{m=0}^{63} \cos\left(2\pi(n+1/2)(m-16)/64\right) I V^{(m)} \boldsymbol{x}^{(32-m-1)}.$$

In terms of matrices this can be written as

$$\boldsymbol{z} = \begin{pmatrix} \cos\left(2\pi(0+1/2)\cdot(-16)/64\right) I & \cdots & \cos\left(2\pi(0+1/2)\cdot(47)/64\right) I \\ \vdots & \ddots & \vdots \\ \cos\left(2\pi(31+1/2)\cdot(-16)/64\right) I & \cdots & \cos\left(2\pi(31+1/2)\cdot(47)/64\right) I \end{pmatrix} \times \begin{pmatrix} V^{(0)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & V^{(1)} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & V^{(62)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & V^{(63)} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}^{(31)} \\ \boldsymbol{x}^{(30)} \\ \vdots \\ \boldsymbol{x}^{(-32)} \end{pmatrix}.$$

If we place the 15 first columns in the cosine matrix last using Property (6.35) (we must then also place the 15 first rows last in the second matrix), we obtain

$$\boldsymbol{z} = \begin{pmatrix} \cos\left(2\pi(0+1/2)\cdot(0)/64\right) I & \cdots & \cos\left(2\pi(0+1/2)\cdot(63)/64\right) I \\ \vdots & \ddots & \vdots \\ \cos\left(2\pi(31+1/2)\cdot(0)/64\right) I & \cdots & \cos\left(2\pi(31+1/2)\cdot(63)/64\right) I \end{pmatrix} \times \begin{pmatrix} \mathbf{0} & \cdots & \mathbf{0} & V^{(16)} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & V^{(63)} \\ -V^{(0)} & \cdots & \cdots & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \mathbf{0} \\ \mathbf{0} & \cdots & -V^{(15)} & \mathbf{0} & \cdots & \mathbf{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}^{(31)} \\ \boldsymbol{x}^{(30)} \\ \vdots \\ \boldsymbol{x}^{(-32)} \end{pmatrix}.$$

Using Equation (6.36) to combine column k and $64-k$ in the cosine matrix (as well as row k and $64-k$ in the second matrix), we can write this as

$$\begin{pmatrix} \cos\left(2\pi(0+1/2)\cdot(0)/64\right) I & \cdots & \cos\left(2\pi(0+1/2)\cdot(31)/64\right) I \\ \vdots & \ddots & \vdots \\ \cos\left(2\pi(31+1/2)\cdot(0)/64\right) I & \cdots & \cos\left(2\pi(31+1/2)\cdot(31)/64\right) I \end{pmatrix} \begin{pmatrix} A' & B' \end{pmatrix} \begin{pmatrix} \boldsymbol{x}^{(31)} \\ \boldsymbol{x}^{(30)} \\ \vdots \\ \boldsymbol{x}^{(-32)} \end{pmatrix}.$$

where

$$A' = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & V^{(16)} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & V^{(15)} & \mathbf{0} & V^{(17)} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \mathbf{0} \\ \mathbf{0} & V^{(1)} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & V^{(31)} \\ V^{(0)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \end{pmatrix}$$

$$B' = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ V_{(32)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & V^{(33)} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & -V^{(63)} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & V^{(47)} & \mathbf{0} & -V^{(49)} & \cdots & \mathbf{0} \end{pmatrix}.$$

Using Equation (4.3), the cosine matrix here can be written as

$$\sqrt{\frac{M}{2}}(D_M)^T \begin{pmatrix} \sqrt{2} & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}.$$

The above can thus be written as

$$4(D_{32})^T \begin{pmatrix} A & B \end{pmatrix} \begin{pmatrix} \boldsymbol{x}^{(31)} \\ \boldsymbol{x}^{(30)} \\ \vdots \\ \boldsymbol{x}^{(-32)} \end{pmatrix},$$

where A and B are the matrices A', B' with the first row multiplied by $\sqrt{2}$ (i.e. replace $V^{(16)}$ with $\sqrt{2}V^{(16)}$ in the matrix A'). Using that $\boldsymbol{x}^{(-i)} = E_1\boldsymbol{x}_i$ for $1 \leq i \leq 32$, we can write this as

$$4(D_{32})^T \begin{pmatrix} A & B \end{pmatrix} \begin{pmatrix} \boldsymbol{x}^{(31)} \\ \vdots \\ \boldsymbol{x}^{(0)} \\ E_1\boldsymbol{x}^{(31)} \\ \vdots \\ E_1\boldsymbol{x}^{(0)} \end{pmatrix} = 4(D_{32})^T \left(A \begin{pmatrix} \boldsymbol{x}^{(31)} \\ \vdots \\ \boldsymbol{x}^{(0)} \end{pmatrix} + B \begin{pmatrix} E_1\boldsymbol{x}^{(31)} \\ \vdots \\ E_1\boldsymbol{x}^{(0)} \end{pmatrix} \right),$$

which can be written as

$$4(D_{32})^T \begin{pmatrix} \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \sqrt{2}V^{(16)} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & V^{(15)} & \mathbf{0} & V^{(17)} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \\ \mathbf{0} & V^{(1)} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & V^{(31)} \\ V^{(0)} + E_1V^{(32)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & E_1V^{(33)} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & -E_1V^{(63)} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & E_1V^{(47)} & \mathbf{0} & -E_1V^{(49)} & \cdots & \mathbf{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}^{(31)} \\ \vdots \\ \boldsymbol{x}^{(0)} \end{pmatrix},$$

which also can be written as

$$4(D_{32})^T \begin{pmatrix} \mathbf{0} & \cdots & \mathbf{0} & \sqrt{2}V^{(16)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \cdots & V^{(17)} & \mathbf{0} & V^{(15)} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ V^{(31)} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & V^{(1)} & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & V^{(0)} + E_1V^{(32)} \\ -E_1V^{(63)} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & E_1V^{(33)} & \mathbf{0} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \cdots & -E_1V^{(49)} & \mathbf{0} & E_1V^{(47)} & \cdots & \mathbf{0} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}^{(0)} \\ \vdots \\ \boldsymbol{x}^{(31)} \end{pmatrix}.$$

We have therefore proved the following result.

Theorem 8.11. *Polyphase factorization of a forward filter bank transform based on a prototype filter.*

The polyphase form of a forward filter bank transform based on a prototype filter can be factored as

$$4(D_{32})^T \begin{pmatrix} \mathbf{0} & \cdots & \mathbf{0} & \sqrt{2}V^{(16)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \cdots & V^{(17)} & \mathbf{0} & V^{(15)} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ V^{(31)} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & V^{(1)} & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & V^{(0)} + E_1V^{(32)} \\ -E_1V^{(63)} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & E_1V^{(33)} & \mathbf{0} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \cdots & -E_1V^{(49)} & \mathbf{0} & E_1V^{(47)} & \cdots & \mathbf{0} & \mathbf{0} \end{pmatrix} \tag{8.20}$$

Due to Theorem 6.26, it is also very simple to write down the polyphase factorization of the reverse filter bank transform as well. Since $E_{481}G^T$ is a

forward filter bank transform where the prototype filter has been reversed, $E_{481}G^T$ can be factored as above, with $V^{(m)}$ replaced by $W^{(m)}$, with $W^{(m)}$ being the filters derived from the synthesis prototype filter in reverse order. This means that the polyphase form of G can be factored as

$$4\begin{pmatrix} \mathbf{0} & \mathbf{0} & \cdots & (W^{(31)})^T & \mathbf{0} & -E_{-1}(W^{(63)})^T & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \mathbf{0} & (W^{(17)})^T & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & -E_{-1}(W^{(49)})^T \\ \sqrt{2}(W^{(16)})^T & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & (W^{(15)})^T & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & E_{-1}(W^{(47)})^T \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & (W^{(1)})^T & \mathbf{0} & E_{-1}(W^{(33)})^T & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & (W^{(0)})^T + E_{-1}(W^{(32)})^T & \mathbf{0} & \cdots & \mathbf{0} \end{pmatrix} \times D_{32}E_{481}. \tag{8.21}$$

Now, if we define $U^{(m)}$ as the filters derived from the synthesis prototype filter itself, we have that

$$(W^{(k)})^T = -E_{-14}V^{(64-k)},\ 1 \le k \le 15 \qquad (W^{(0)})^T = E_{-16}V^{(0)}.$$

Inserting this in Equation (8.21) we get the following result:

Theorem 8.12. *Polyphase factorization of a reverse filter bank transform based on a prototype filter.*

Assume that G is a reverse filter filter bank transform based on a prototype filter, and that $U^{(m)}$ are the filters derived from this prototype filter. Then the polyphase form of G can be factored as

$$4\begin{pmatrix} \mathbf{0} & \mathbf{0} & \cdots & -U^{(33)} & \mathbf{0} & E_{-1}U^{(1)} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \mathbf{0} & -U^{(47)} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & E_{-1}U^{(15)} \\ -\sqrt{2}U^{(48)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & -U^{(49)} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & -E_{-1}U^{(17)} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & -U^{(63)} & \mathbf{0} & -E_{-1}U^{(31)} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & E_{-2}U^{(0)} - E_{-1}U^{(32)} & \mathbf{0} & \cdots & \mathbf{0} \end{pmatrix} \times D_{32}E_{33}. \tag{8.22}$$

Now, consider the matrices

$$\begin{pmatrix} V^{(32-i)} & V^{(i)} \\ -E_1V^{(64-i)} & E_1V^{(32+i)} \end{pmatrix} \text{ and } \begin{pmatrix} -U^{(32+i)} & E_{-1}U^{(i)} \\ -U^{(64-i)} & -E_{-1}U^{(32-i)} \end{pmatrix}. \tag{8.23}$$

for $1 \leq i \leq 15$. These make out submatrices in the matrices in equations (8.20) and (8.22). Clearly, only the product of these matrices influence the result. Since

$$\begin{aligned}
&\begin{pmatrix} -U^{(32+i)} & E_{-1}U^{(i)} \\ -U^{(64-i)} & -E_{-1}U^{(32-i)} \end{pmatrix} \begin{pmatrix} V^{(32-i)} & V^{(i)} \\ -E_1V^{(64-i)} & E_1V^{(32+i)} \end{pmatrix} \\
&= \begin{pmatrix} -U^{(32+i)} & U^{(i)} \\ -U^{(64-i)} & -U^{(32-i)} \end{pmatrix} \begin{pmatrix} V^{(32-i)} & V^{(i)} \\ -V^{(64-i)} & V^{(32+i)} \end{pmatrix}
\end{aligned} \tag{8.24}$$

we have the following result.

Theorem 8.13. *Filter bank transforms.*

Let H, G be forward and reverse filter bank transforms defined from analysis and synthesis prototype filters. Let also $V^{(k)}$ be the prototype filter of H, and $U^{(k)}$ the reverse of the prototype filter of G. If

$$\begin{aligned}
\begin{pmatrix} -U^{(32+i)} & U^{(i)} \\ -U^{(64-i)} & -U^{(32-i)} \end{pmatrix} \begin{pmatrix} V^{(32-i)} & V^{(i)} \\ -V^{(64-i)} & V^{(32+i)} \end{pmatrix} &= c \begin{pmatrix} E_d & \mathbf{0} \\ \mathbf{0} & E_d \end{pmatrix} \\
(\sqrt{2}V^{(16)})(-\sqrt{2}U^{(48)}) &= cE_d \\
(V^{(0)} + E_1V^{(32)})(E_{-2}U^{(0)} - E_{-1}U^{(32)}) &= cE_d
\end{aligned} \tag{8.25}$$

for $1 \leq i \leq 15$, then $GH = 16cE_{33+32d}$.

This result is the key ingredient we need in order to construct forward and reverse systems which together give perfect reconstuction. In Exercise 8.15 we go through how we can use lifting in order to express a wide range of possible (U, V) matrix pairs which satisfy Equation (8.25). This turns the problem of constructing cosine-modulated filter banks which are useful for audio coding into an optimization problem: the optimization variables are values λ_i which characterize lifting steps, and the objective function is the deviation of the corresponding prototype filter from an ideal bandpass filter. This optimization problem has been subject to a lot of research, and we will not go into details on this.

8.3.2 The prototype filters

Now, let us return to the MP3 standard. We previously observed that in this standard the coefficients in the synthesis prototype filter seemed to equal 32 times the analysis prototype filter. This indicates that $U^{(k)} = 32V^{(k)}$. A closer inspection also yields that there is a symmetry in the values of the prototype filter: We see that $C_i = -C_{512-i}$ (i.e. antisymmetry) for most values of i. The only exception is for $i = 64, 128, \ldots, 448$, for which $C_i = C_{512-i}$ (i.e. symmetry). The antisymmetry can be translated to that the filter coefficients of $V^{(k)}$ equal those of $V^{(64-k)}$ in reverse order, with a minus sign. The symmetry can be translated to that $V^{(0)}$ is symmetric. These observations can be rewritten as

$$V^{(64-k)} = -E_{14}(V^{(k)})^T, 1 \leq k \leq 15. \tag{8.26}$$

$$V^{(0)} = E_{16}(V^{(0)})^T. \tag{8.27}$$

Inserting first that $U^{(k)} = 32V^{(k)}$ in Equation (8.24) gives

$$\begin{aligned}&\begin{pmatrix} -U^{(32+i)} & U^{(i)} \\ -U^{(64-i)} & -U^{(32-i)} \end{pmatrix} \begin{pmatrix} V^{(32-i)} & V^{(i)} \\ -V^{(64-i)} & V^{(32+i)} \end{pmatrix} \\ =&32 \begin{pmatrix} -V^{(32+i)} & V^{(i)} \\ -V^{(64-i)} & -V^{(32-i)} \end{pmatrix} \begin{pmatrix} V^{(32-i)} & V^{(i)} \\ -V^{(64-i)} & V^{(32+i)} \end{pmatrix}.\end{aligned}$$

Substituting for $V^{(32+i)}$ and $V^{(64-i)}$ after what we found by inspection now gives

$$\begin{aligned}&32 \begin{pmatrix} E_{14}(V^{(32-i)})^T & V^{(i)} \\ E_{14}(V^{(i)})^T & -V^{(32-i)} \end{pmatrix} \begin{pmatrix} V^{(32-i)} & V^{(i)} \\ E_{14}(V^{(i)})^T & -E_{14}(V^{(32-i)})^T \end{pmatrix} \\ =&32 \begin{pmatrix} E_{14} & \mathbf{0} \\ \mathbf{0} & E_{14} \end{pmatrix} \begin{pmatrix} (V^{(32-i)})^T & V^{(i)} \\ (V^{(i)})^T & -V^{(32-i)} \end{pmatrix} \begin{pmatrix} V^{(32-i)} & V^{(i)} \\ (V^{(i)})^T & -(V^{(32-i)})^T \end{pmatrix} \\ =&32 \begin{pmatrix} E_{14} & \mathbf{0} \\ \mathbf{0} & E_{14} \end{pmatrix} \begin{pmatrix} V^{(32-i)} & V^{(i)} \\ (V^{(i)})^T & -(V^{(32-i)})^T \end{pmatrix}^T \begin{pmatrix} V^{(32-i)} & V^{(i)} \\ (V^{(i)})^T & -(V^{(32-i)})^T \end{pmatrix} \\ =&32 \begin{pmatrix} E_{14} & \mathbf{0} \\ \mathbf{0} & E_{14} \end{pmatrix} \begin{pmatrix} V^{(i)}(V^{(i)})^T + V^{(32-i)}(V^{(32-i)})^T & \mathbf{0} \\ \mathbf{0} & V^{(i)}(V^{(i)})^T + V^{(32-i)}(V^{(32-i)})^T \end{pmatrix}.\end{aligned} \tag{8.28}$$

Due to Exercise 8.6 (set $A = (V^{(32-i)})^T, B = (V^{(i)})^T$), with

$$H = \begin{pmatrix} V^{(32-i)} & V^{(i)} \\ (V^{(i)})^T & -(V^{(32-i)})^T \end{pmatrix} \qquad G = \begin{pmatrix} (V^{(32-i)})^T & V^{(i)} \\ (V^{(i)})^T & -V^{(32-i)} \end{pmatrix}$$

we recognize an alternative QMF filter bank. We thus have alias cancellation, with perfect reconstruction only if $|\lambda_{H_0}(\omega)|^2 + |\lambda_{H_0}(\omega + \pi)|^2$. For the two remaining filters we compute

$$\begin{aligned}&(\sqrt{2}V^{(16)})(-\sqrt{2}U^{(48)}) \\ &= -64V^{(16)}V^{(48)} = 64E_{14}V^{(16)}(V^{(16)})^T = 32E_{14}(V^{(16)}(V^{(16)})^T + V^{(16)}(V^{(16)})^T)\end{aligned} \tag{8.29}$$

and

$$\begin{aligned}&(V^{(0)} + E_1V^{(32)})(E_{-2}U^{(0)} - E_{-1}U^{(32)}) \\ &= 32(V^{(0)} + E_1V^{(32)})(E_{-2}V^{(0)} - E_{-1}V^{(32)}) = 32E_{-2}(V^{(0)} + E_1V^{(32)})(V^{(0)} - E_1V^{(32)}) \\ &= 32E_{-2}(V^{(0)})^2 - (V^{(32)})^2) = 32E_{14}((V^{(0)}(V^{(0)})^T + V^{(32)}(V^{(32)})^T)).\end{aligned} \tag{8.30}$$

We see that the filters from equations (8.28)-(8.30) are similar, and that we thus can combine them into

$$\{V^{(i)}(V^{(i)})^T + V^{(32-i)}(V^{(32-i)})^T\}_{i=0}^{16}. \tag{8.31}$$

All of these can be the identity, expect for $1024V^{(16)}(V^{(16)})^T$, since we know that the product of two FIR filters is never the identity, except when both are delays (And all $V^{(m)}$ are FIR, since the prototype filters defined by the MP3 standard are FIR). This single filter is thus what spoils for perfect reconstruction, so that we can only hope for alias cancellation, and this happens when the filters from Equation (8.31) all are equal. Ideally this is close to cI for some scalar c, and we then have that

$$GH = 16 \cdot 32cE_{33+448} = 512cE_{481}I.$$

This explains the observation from the MP3 standard that GH seems to be close to E_{481}. Since all the filters $V^{(i)}(V^{(i)})^T + V^{(32-i)}(V^{(32-i)})^T$ are symmetric, GH is also a symmetric filter due to Theorem 8.3, so that its frequency response is real, so that we have no phase distortion. We can thus summarize our findings as follows.

Observation 8.14. *MP3 standard.*

The prototype filters from the MP3 standard do not give perfect reconstruction. They are found by choosing 17 filters $\{V^{(k)}\}_{k=0}^{16}$ so that the filters from Equation (8.31) are equal, and so that their combination into a prototype filter using equations (8.19) and (8.26) is as close to an ideal bandpass filter as possible. When we have equality the alias cancellation condition is satisfied, and we also have no phase distortion. When the common value is close to $\frac{1}{512}I$, GH is close to E_{481}, so that we have near-perfect reconstruction.

This states clearly the optimization problem which the values stated in the MP3 standard solves.

8.3.3 Perfect reconstruction

How can we overcome the problem that $1024V^{(16)}(V^{(16)})^T \neq I$, which spoiled for perfect reconstruction in the MP3 standard? It turns out that we can address this a simple change in our procedure. In Equation (8.18) we replace with

$$z_{32(s-1)+n} = \sum_{k=0}^{511} \cos((n+1/2)(k+1/2-16)\pi/32)h_k x_{32s-k-1}, \tag{8.32}$$

i.e. $1/2$ is added inside the cosine. We now have the properties

$$\cos\left(2\pi(n+1/2)(k+64r+1/2)/(2N)\right) = (-1)^r \cos\left(2\pi(n+1/2)(k+1/2)/(2N)\right) \tag{8.33}$$

$$\cos\left(2\pi(n+1/2)(2N-k-1+1/2)/(2N)\right) = -\cos\left(2\pi(n+1/2)(k+1/2)/(2N)\right). \tag{8.34}$$

Due to the first property, we can deduce as before that

$$\boldsymbol{z}^{(n)} = \sum_{m=0}^{63} \cos\left(2\pi(n+1/2)(m+1/2-16)/64\right) IV^{(m)} \boldsymbol{x}^{(32-m-1)},$$

where the filters $V^{(m)}$ are defined as before. As before placing the 15 first columns of the cosine-matrix last, but instead using Property (8.34) to combine columns k and $64-k-1$ of the cosine-matrix, we can write this as

$$\begin{pmatrix} \cos\left(2\pi(0+1/2)\cdot(0+1/2)/64\right) I & \cdots & \cos\left(2\pi(0+1/2)\cdot(31+1/2)/64\right) I \\ \vdots & \ddots & \vdots \\ \cos\left(2\pi(31+1/2)\cdot(0+1/2)/64\right) I & \cdots & \cos\left(2\pi(31+1/2)\cdot(31+1/2)/64\right) I \end{pmatrix} \begin{pmatrix} A & B \end{pmatrix} \begin{pmatrix} \boldsymbol{x}^{(31)} \\ \vdots \\ \boldsymbol{x}^{(-32)} \end{pmatrix}$$

where

$$A = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \cdots & V^{(15)} & V^{(16)} & \cdots & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & V^{(1)} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & V^{(30)} & \mathbf{0} \\ V^{(0)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & \cdots & V^{(31)} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \end{pmatrix}$$

$$B = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ V_{(32)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & -V^{(63)} \\ \mathbf{0} & V^{(33)} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & -V^{(62)} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & V^{(47)} & -V^{(48)} & \cdots & \cdots & \mathbf{0} \end{pmatrix}.$$

Since the cosine matrix can be written as $\sqrt{\frac{M}{2}} D_M^{(iv)}$, the above can be written as

$$4D_M^{(iv)} \begin{pmatrix} A & B \end{pmatrix} \begin{pmatrix} \boldsymbol{x}^{(31)} \\ \vdots \\ \boldsymbol{x}^{(-32)} \end{pmatrix}.$$

As before we can rewrite this as

$$4D_M^{(iv)}\begin{pmatrix}A & B\end{pmatrix}\begin{pmatrix}\boldsymbol{x}^{(31)}\\ \vdots\\ \boldsymbol{x}^{(0)}\\ E_1\boldsymbol{x}^{(31)}\\ \vdots\\ E_1\boldsymbol{x}^{(0)}\end{pmatrix} = 4D_M^{(iv)}\left(A\begin{pmatrix}\boldsymbol{x}^{(31)}\\ \vdots\\ \boldsymbol{x}^{(0)}\end{pmatrix} + B\begin{pmatrix}E_1\boldsymbol{x}^{(31)}\\ \vdots\\ E_1\boldsymbol{x}^{(0)}\end{pmatrix}\right),$$

which can be written as

$$4D_M^{(iv)}\begin{pmatrix}\mathbf{0} & \mathbf{0} & \cdots & V^{(15)} & V^{(16)} & \cdots & \cdots & \mathbf{0}\\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots\\ \mathbf{0} & V^{(1)} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & V^{(30)} & \mathbf{0}\\ V^{(0)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & \cdots & V^{(31)}\\ E_1V_{(32)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & \cdots & -E_1V^{(63)}\\ \mathbf{0} & E_1V^{(33)} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & -E_1V^{(62)} & \mathbf{0}\\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots\\ \mathbf{0} & \mathbf{0} & \cdots & E_1V^{(47)} & -E_1V^{(48)} & \cdots & \cdots & \mathbf{0}\end{pmatrix}\begin{pmatrix}\boldsymbol{x}^{(31)}\\ \vdots\\ \boldsymbol{x}^{(0)}\end{pmatrix},$$

which also can be written as

$$4D_M^{(iv)}\begin{pmatrix}\mathbf{0} & \mathbf{0} & \cdots & V^{(16)} & V^{(15)} & \cdots & \cdots & \mathbf{0}\\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots\\ \mathbf{0} & V^{(30)} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & V^{(1)} & \mathbf{0}\\ V^{(31)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & \cdots & V^{(0)}\\ -E_1V_{(63)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & \cdots & E_1V^{(32)}\\ \mathbf{0} & -E_1V^{(62)} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & E_1V^{(33)} & \mathbf{0}\\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots\\ \mathbf{0} & \mathbf{0} & \cdots & -E_1V^{(48)} & E_1V^{(47)} & \cdots & \cdots & \mathbf{0}\end{pmatrix}\begin{pmatrix}\boldsymbol{x}^{(0)}\\ \vdots\\ \boldsymbol{x}^{(31)}\end{pmatrix}.$$

We therefore have the following result

Theorem 8.15. *Polyphase factorization of a forward filter bank transform based on a prototype filter, modified version.*

The modified version of the polyphase form of a forward filter bank transform based on a prototype filter can be factored as

$$4D_M^{(iv)} \begin{pmatrix} \mathbf{0} & \mathbf{0} & \cdots & V^{(16)} & V^{(15)} & \cdots & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & V^{(30)} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & V^{(1)} & \mathbf{0} \\ V^{(31)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & \cdots & V^{(0)} \\ -E_1 V_{(63)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & \cdots & E_1 V^{(32)} \\ \mathbf{0} & -E_1 V^{(62)} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & E_1 V^{(33)} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & -E_1 V^{(48)} & E_1 V^{(47)} & \cdots & \cdots & \mathbf{0} \end{pmatrix} \tag{8.35}$$

Clearly this factorization avoids having two blocks of filters: There are now 16 2×2-polyphase matrices, and as we know, each of them can be invertible, so that the full matrix can be inverted in a similar fashion as before. It is therefore now possible to obtain perfect reconstruction. Although we do not state recipes for implementing this, one has just as efficient implementations as in the MP3 standard.

Since we ended up with the 2×2 polyphase matrices M_k, we can apply the lifting factorization in order to halve the number of multiplications/additions. This is not done in practice, since a lifting factorization requires that we compute all outputs at once. In audio coding it is required that we compute the output progressively, due to the large size of the input vector. The procedure above is therefore mostly useful for providing the requirements for the filters, while the preceding comments can be used for the implementation.

Exercise 8.13: Run forward and reverse transform

Run the forward and then the reverse transform from Exercise 6.26 on the vector $(1, 2, 3, \ldots, 8192)$. Verify that there seems to be a delay on 481 elements, as promised by Therorem 8.14. Do you get the exact same result back?

Exercise 8.14: Verify statement of filters

Use your computer to verify the symmetries we have stated for the symmetries in the prototype filters, i.e. that

$$C_i = \begin{cases} -C_{512-i} & i \neq 64, 128, \ldots, 448 \\ C_{512-i} & i = 64, 128, \ldots, 448. \end{cases}$$

Explain also that this implies that $h_i = h_{512-i}$ for $i = 1, \ldots, 511$. In other words, the prototype filter has symmetry around $(511+1)/2 = 256$, so that it has linear phase.

Exercise 8.15: Lifting

We mentioned that we could use the lifting factorization to construct filters on the form stated in Equation (8.19), so that the matrices on the form given by Equation (8.23), i.e.

$$\begin{pmatrix} V^{(32-i)} & V^{(i)} \\ -V^{(64-i)} & V^{(32+i)} \end{pmatrix},$$

are invertible. Let us see what kind of lifting steps produce such matrices.

a) Show that the lifting steps

$$\begin{pmatrix} I & \lambda E_2 \\ \mathbf{0} & I \end{pmatrix} \text{ and } \begin{pmatrix} I & \mathbf{0} \\ \lambda I & I \end{pmatrix}$$

applied in alternating order to a matrix on the form given by Equation (8.23), where the filters are on the from given by Equation (8.19), again produces matrices and filters on these forms. This explains how we can parametrize a larger number of such matrices with the help of lifting steps.It also explain why the inverse matrix is on the form stated in Equation (8.23) with filters on the same form, since the inverse lifting steps are of the same type.

b) Explain that 16 numbers $\{\lambda_i\}_{i=1}^{16}$ are needed (together with what we start with on the diagonal in the lifting construction), in order to construct filters so that the prototype filter has 512 coefficients. Since there are 15 submatrices, this gives 240 optimization variables.

Lifting gives the following strategy for finding a corresponding synthesis prototype filter which gives perfect reconstruction: First compute matrices V, W which are inverses of oneanother using lifting (using the lifting steps of this exercise ensures that all filters will be on the form stated in Equation (8.19)), and write

$$\begin{aligned} VW &= \begin{pmatrix} V^{(1)} & V^{(2)} \\ -V^{(3)} & V^{(4)} \end{pmatrix} \begin{pmatrix} W^{(1)} & -W^{(3)} \\ W^{(2)} & W^{(4)} \end{pmatrix} = \begin{pmatrix} V^{(1)} & V^{(2)} \\ -V^{(3)} & V^{(4)} \end{pmatrix} \begin{pmatrix} (W^{(1)})^T & (W^{(2)})^T \\ -(W^{(3)})^T & (W^{(4)})^T \end{pmatrix}^T \\ &= \begin{pmatrix} V^{(1)} & V^{(2)} \\ -V^{(3)} & V^{(4)} \end{pmatrix} \begin{pmatrix} E_{15}(W^{(1)})^T & E_{15}(W^{(2)})^T \\ -E_{15}(W^{(3)})^T & E_{15}(W^{(4)})^T \end{pmatrix}^T \begin{pmatrix} E_{15} & \mathbf{0} \\ \mathbf{0} & E_{15} \end{pmatrix} = I. \end{aligned}$$

Now, the matrices $U^{(i)} = E_{15}(W^{(i)})^T$ are on the form stated in Equation (8.19), and we have that

$$\begin{pmatrix} V^{(1)} & V^{(2)} \\ -V^{(3)} & V^{(4)} \end{pmatrix} \begin{pmatrix} U^{(1)} & U^{(2)} \\ -U^{(3)} & U^{(4)} \end{pmatrix} = \begin{pmatrix} E_{-15} & \mathbf{0} \\ \mathbf{0} & E_{-15} \end{pmatrix}$$

We can now conclude from Theorem 8.13 that if we define the synthesis prototype filter as therein, and set $c = 1, d = -15$, we have that $GH = 16E_{481-32\cdot 15} = 16E_1$.

8.4 Summary

We defined the polyphase representation of a matrix, and proved some useful properties. For filter bank transforms, the polyphase representation was a block matrix where the blocks are filters, and these blocks/filters were called polyphase components. In particular, the filter bank transforms of wavelets were 2×2-block matrices of filters. We saw that, for wavelets, the polyphase representation could be realized through a rearrangement of the wavelet bases, and thus paralleled the development in Chapter 6 for expressing the DWT in terms of filters, where we instead rearranged the target base of the DWT.

We showed with two examples that factoring the polyphase representation into simpler matrices (also refered to as a polyphase factorization) could be a useful technique. First, for wavelets ($M = 2$), we established the lifting factorization. This is useful not only since it factorizes the DWT and the IDWT into simpler operations, but also since it reduces the number of arithmetic operations in these. The lifting factorization is therefore also used in practical implementations, and we applied it to some of the wavelets we constructed in Chapter 7. The JPEG2000 standard document [21] explains a procedure for implementing some of these wavelet transforms using lifting, and the values of the lifting steps used in the standard thus also appear here.

The polyphase representation was also useful for proving the characterization of wavelets we encountered in Chapter 7, which we used to find expressions for many useful wavelets.

The polyphase representation was also useful to explain how the prototype filters of the MP3 standard should be chosen, in order for the reverse filter bank transform to invert the forward filter bank transform. Again this was attacked by factoring the polyphase representation of the forward and reverse filter bank transforms. The parts of the factorization which represented the prototype filters were represented by a sparse matrix, and it was clear from this matrix what properties we needed to put on the prototype filter, in order to have alias cancellation, and no phase distortion. In fact, we proved that the MP3 standard could not possible give perfect reconstruction, but it was very clear from our construction how the filter bank could be modified in order for the overall system to provide perfect reconstruction.

The lifting scheme as introduced here was first proposed by Sweldens [45]. How to use lifting for in-place calculation for the DWT was also suggested by Sweldens [44].

This development concludes the one-dimensional aspect of wavelets in this book. In the following we will extend our theory to also apply for images. Images will be presented in Chapter 9. After that we will define the tensor product concept, which will be the key ingredient to apply wavelets to two-dimensional objects such as images.

Chapter 9

Digital images

Upto now we have presented wavelets in a one-dimensional setting. Images, however, are two-dimensional by nature. This poses another challenge, which we did not encounter in the case of sound signals. In this chapter we will establish the mathematics to handle this, but first we will present some basics on images, as well as how they can be represented and manipulated with simple mathematics. Images are a very important type of digital media, and this material is thus useful, general knowledge for anyone with a digital camera and a computer. For many scientists this material is also an essential tool. As an example, in astrophysics data from both satellites and distant stars and galaxies is collected in the form of images, and information is extracted from the images with advanced image processing techniques. As another example, 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.

We will see how filter-based operations extend naturally to the two-dimensional setting of images. Smoothing and edge detections are the two main examples of filter-based operations we will concider for images. The key mathematical concept in this extension is the *tensor product*, which can be thought of as a general tool for constructing two-dimensional objects from one-dimensional counterparts. We will also see that the tensor product allows us to establish an efficient implementation of filtering for images, efficient meaning a complexity substantially less than what is required by general linear transformations.

We will finally consider useful coordinate changes for images. Recall that 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 two-dimensional data like images. It turns out that the tensor product can also be used to extend changes of coordinates to a two-dimensional setting.

Functionality for accessing images are collected in a module called `images`.

9.1 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.

Light.

Fact 9.1. *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 ($\approx 3 \times 10^8$ 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.

Digital output media. Our focus will be on objects that emit light, for example a computer display. A computer monitor consists of a matrix 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 colors. In this way the color at each set of three dots can be controlled, and a color 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, colors by mixing red, green and blue. In addition, different computer monitors use slightly different red, green and blue colors, and unless this is taken into consideration, colors will look different on the two monitors. This also means that some colors 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 colors. Instead as many as 7–8 different inks (or similar substances) are mixed to the right color. This makes it possible to produce a wide range of colors, but not all, and the problem of matching a color from another device like a monitor is at least as difficult as matching different colors 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 colors equally.

The quality of a device is closely linked to the density of the dots.

Fact 9.2. *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).

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 matrix of (possibly colored) dots. As for printers, an important measure of quality is the number of dots per inch.

Fact 9.3. *Printers.*

The resolution of a scanner usually varies in the range 75 dpi to 9600 dpi, and the color 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 9.4. *Pixels.*

The number of pixels recorded by a digital camera usually varies in the range 320×240 to 6000×4000 with 24 bits of color 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 color 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.

Definition of digital image. We have already talked about digital images, but we have not yet been precise about what they are. From a mathematical point of view, an image is quite simple.

Fact 9.5. *Digital image.*

A digital image P is a matrix 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 color 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 referring 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 color 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 color 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 Example 9.3 below).

Figure 9.1: Our test image.

In Figure 9.1 we have shown the test image we will work with, called the *Lena image*. It is named after the girl in the image. This image is also used as a test image in many textbooks on image processing.

In Figure 9.2 we have shown the corresponding black and white, and grey-level versions of the test image.

Fact 9.6. *Intensity.*

In these notes the intensity values $p_{i,j}$ are assumed to be real numbers in the interval $[0, 1]$. For color images, each of the red, green, and blue intensity values are assumed to be real numbers in $[0, 1]$.

Figure 9.2: Black and white (left), and grey-level (right) versions of the image in Figure 9.1.

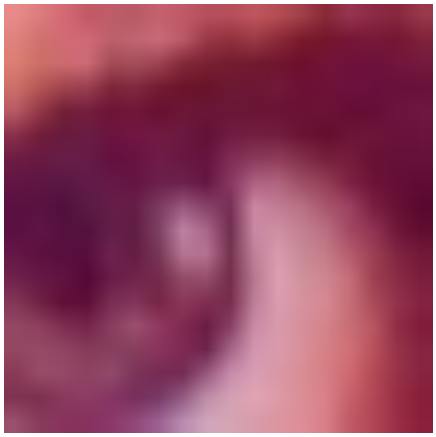

Figure 9.3: 18×18 pixels excerpt of the color image in Figure 9.1. The grid indicates the borders between the pixels.

If we magnify the part of the color image in Figure 9.1 around one of the eyes, we obtain the images in figures 9.3-9.4. 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 centers, as indicated by the grids in figures 9.3-9.4.

Fact 9.7. *Shape of pixel.*

Figure 9.4: 50×50 pixels excerpt of the color image in Figure 9.1.

The pixels of an image are assumed to be square with sides of length one, with the pixel with value $p_{i,j}$ centered at the point (i, j).

9.2 Some simple operations on images with Python

Images are two-dimensional matrices 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 will go through first).

In order to perform these operations, we need to be able to use images with a programming environment.

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 color image, we need 3 matrices, one for each color component. We will also view this as a 3-dimensional matrix. In the following, operations on images will be implemented in such a way that they are applied to each color component simultaneously. This is similar to the FFT and the DWT, where the operations were applied to each sound channel simultaneously.

Since images are viewed as 2-dimensional or 3-dimensional matrices, we can use any linear algebra software in 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 previous 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

```
X = double(imread('filename.fmt', 'fmt'))
```

the image with the given path and format is read, and stored in the matrix which we call `X`. 'fmt' can be 'jpg','tif', 'gif', 'png', and so on. This parameter is optional: If it is not present, the program will attempt to determine the format from the first bytes in the file, and from the filename. 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`). To perform operations on the image, we must first convert the entries to the data type `double`, as shown above. Similarly, the function `imwrite`

can be used to write the image represented by a matrix to file. If we write

```
imwrite(uint8(X), 'filename.fmt', 'fmt')
```

the image represented by the matrix `X` 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. 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(uint8(X))` displays the matrix `X` as an image in a separate window. Also here we needed to convert the samples using the function `uint8`.

The following examples go through some much used operations on images.

Example 9.1: 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.

```
def mapto01(X):
    minval, maxval = X.min(), X.max()
    X -= minval
    X /= (maxval-minval)

def contrastadjust(X,epsilon):
    """
    Assumes that the values are in [0,255]
    """
    X /= 255.
    X += epsilon
    log(X, X)
    X -= log(epsilon)
    X /= (log(1+epsilon)-log(epsilon))
    X *= 255

def contrastadjust0(X,n):
    """
    Assumes that the values are in [0,255]
    """
    X /= 255.
    X -= 1/2.
    X *= n
    arctan(X, X)
    X /= (2*arctan(n/2.))
    X += 1/2.0
    X *= 255 # Maps the values back to [0,255]
```

Several examples of using this function will be shown below. A good question here is why the functions `min` and `max` are called three times in succession. The reason is that there is a third "dimension" in play, besides the spatial x- and y-directions. This dimension describes the color components in each pixel, which are usually the red-, green-, and blue color components.

Example 9.2: Extracting the different colors

If we have a color image

$$P = (r_{i,j}, g_{i,j}, b_{i,j})_{i,j=1}^{m,n},$$

it is often useful to manipulate the three color 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 color components, respectively. Let us take the image `lena.png` used in Figure 9.1. When the image is read (first line below), the returned object has three dimensions. The first two dimensions represent the spatial directions (the row-index and column-index). The third dimension represents the color component. One can therefore view images representing the different color components with the help of the following code:

```
X1 = zeros_like(img)
X1[:, :, 0] = img[:, :, 0]

X2 = zeros_like(img)
X2[:, :, 1] = img[:, :, 1]

X3 = zeros_like(img)
X3[:, :, 2] = img[:, :, 2]
```

The resulting images are shown in Figure 9.5.

Figure 9.5: The red, green, and blue components of the color image in Figure 9.1.

Example 9.3: Converting from color to grey-level

If we have a color 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 color 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 color 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 color components as a measure of the intensity, i.e, to set $p = \max(r, g, b)$. 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. A third possibility is to think of the intensity of (r, g, b) as the length of the color vector, in analogy with points in space, and set $p = \sqrt{r^2 + g^2 + b^2}$. 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.

Let us sum this up as follows: A color 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:

- Set $q_{i,j} = \max(r_{i,j}, g_{i,j}, b_{i,j})$ for all i and j.
- Compute $\hat{q}_{i,j} = r_{i,j} + g_{i,j} + b_{i,j}$ for all i and j.
- Transform all the values to the interval $[0, 1]$ by setting

$$q_{i,j} = \frac{\hat{q}_{i,j}}{\max_{k,l} \hat{q}_{k,l}}.$$

- Compute $\hat{q}_{i,j} = \sqrt{r_{i,j}^2 + g_{i,j}^2 + b_{i,j}^2}$ for all i and j.
- Transform all the values to the interval $[0, 1]$ by setting

$$q_{i,j} = \frac{\hat{q}_{i,j}}{\max_{k,l} \hat{q}_{k,l}}.$$

In practice one of the last two methods are preferred, perhaps with a preference for the last method, but the actual choice depends on the application. These can be implemented as follows.

```
mx = maximum(img[:, :, 0], img[:, :, 1])
X1 = maximum(img[:, :, 2], mx)

X2 = img[:, :, 0] + img[ :, :, 1] + img[ :, :, 2]
mapto01(X2)
X2 *= 255

X3 = sqrt(img[:,:,0]**2 + img[:, :, 1]**2 + img[:, :, 2]**2)
mapto01(X3)
X3 *= 255
```

The results of applying these three operations can be seen in Figure 9.6.

Figure 9.6: Alternative ways to convert the color image in Figure 9.1 to a grey level image.

Example 9.4: 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' $1 - p$. This is also easily translated to code as above. The resulting image is shown in Figure 9.7.

Figure 9.7: The negative versions of the corresponding images in Figure 9.6.

Example 9.5: 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 monotone function f which maps $[0, 1]$ onto $[0, 1]$. If we choose f so that its derivative is large in the area where many intensity values are concentrated, we obtain the desired effect. We will consider two such families of functions:

$$f_n(x) = \frac{\arctan(n(x - 1/2))}{2\arctan(n/2)} + \frac{1}{2} \tag{9.1}$$

$$g_\epsilon(x) = \frac{\ln(x + \epsilon) - \ln\epsilon}{\ln(1 + \epsilon) - \ln\epsilon}. \tag{9.2}$$

The first type of functions 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 second type of functions have a large derivative near $x = 0$ and will therefore increase the contrast in images with a large proportion of small intensity values, i.e., very dark images. Figure 9.8 shows some examples of these functions. The three functions in the left plot in Figure 9.8 are f_4, f_{10}, and f_{100}, the ones shown in the plot are $g_{0.1}$, $g_{0.01}$, and $g_{0.001}$.

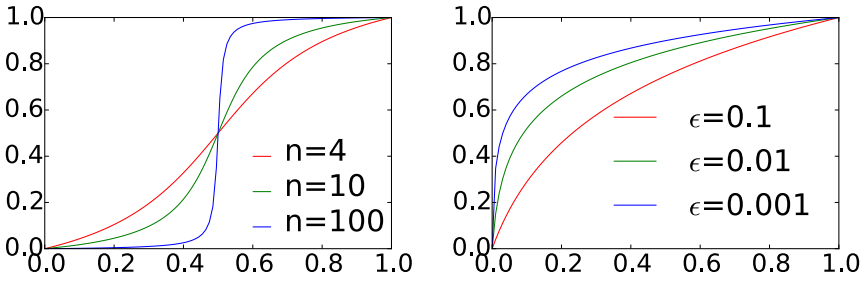

Figure 9.8: Some functions that can be used to improve the contrast of an image.

Figure 9.9: The result after applying f_{10} and $g_{0.01}$ to the test image.

In Figure 9.9 f_{10} and $g_{0.01}$ have been applied to the image in the right part of Figure 9.6. Since the image was quite well balanced, f_{10} made the dark areas too dark and the bright areas too bright. $g_{0.01}$ on the other hand has made the image as a whole too bright.

Increasing the contrast is easy to implement. The following function uses the contrast adjusting function from Equation (9.2), with ϵ as parameter.

```
def contrastadjust(X,epsilon):
    """
    Assumes that the values are in [0,255]
    """
    X /= 255.
    X += epsilon
    log(X, X)
    X -= log(epsilon)
    X /= (log(1+epsilon)-log(epsilon))
    X *= 255
```

```
def contrastadjust0(X,n):
    """
    Assumes that the values are in [0,255]
    """
    X /= 255.
    X -= 1/2.
    X *= n
    arctan(X, X)
    X /= (2*arctan(n/2.))
    X += 1/2.0
    X *= 255 # Maps the values back to [0,255]
```

This has been used to generate the right image in Figure 9.9.

Exercise 9.6: Generate black and white images

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 the right part of Figure 9.2.

Exercise 9.7: Adjust contrast in images

a) Write a function `contrastadjust0` which instead uses the function from Equation (9.1) to increase the contrast. n should be a parameter to the function.

b) Generate the left and right images in Figure 9.9 on your own by writing code which uses the two functions `contrastadjust0` and `contrastadjust`.

Exercise 9.8: Adjust contrast with another function

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 $n \to \infty$.

b) The color image `secret.jpg`,shown in Figure 9.10, 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.

Figure 9.10: Secret message.

9.3 Filter-based operations on images

The next examples of operations on images we consider will use filters. These examples define what it means to apply a filter to two-dimensional data. We start with the following definition of a computational molecule. This term stems from image processing, and seems at the outset to be unrelated to filters.

Definition 9.8. *Computational molecules.*

We say that an operation S on an image X is given by the *computational molecule*

$$A = \begin{pmatrix} \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & a_{-1,-1} & a_{-1,0} & a_{-1,1} & \cdots \\ \cdots & a_{0,-1} & \underline{a_{0,0}} & a_{0,1} & \cdots \\ \cdots & a_{1,-1} & a_{1,0} & a_{1,1} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

if we have that

$$(SX)_{i,j} = \sum_{k_1,k_2} a_{k_1,k_2} X_{i-k_1,j-k_2}. \tag{9.3}$$

In the molecule, indices are allowed to be both positive and negative, we underline the element with index $(0,0)$ (the center of the molecule), and assume that $a_{i,j}$ with indices falling outside those listed in the molecule are zero (as for compact filter notation).

In Equation (9.3), it is possible for the indices $i-k_1$ and $j-k_2$ to fall outside the legal range for X. We will solve this case in the same way as we

did for filters, namely that we assume that X is extended (either periodically or symmetrically) in both directions. The interpretation of a computational molecule is that we place the center of the molecule on a pixel, multiply the pixel and its neighbors by the corresponding weights $a_{i,j}$ in reverse order, and finally sum up in order to produce the resulting value. This type of operation will turn out to be particularly useful for images. The following result expresses how computational molecules and filters are related. It states that, if we apply one filter to all the columns, and then another filter to all the rows, the end result can be expressed with the help of a computational molecule.

Theorem 9.9. *Filtering and computational molecules.*

Let S_1 and S_2 be filters with compact filter notation $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$, respectively, and consider the operation S where S_1 is first applied to the columns in the image, and then S_2 is applied to the rows in the image. Then S is an operation which can be expressed in terms of the computational molecule $a_{i,j} = (\boldsymbol{t}_1)_i(\boldsymbol{t}_2)_j$.

Proof. Let $X_{i,j}$ be the pixels in the image. When we apply S_1 to the columns of X we get the image Y defined by

$$Y_{i,j} = \sum_{k_1} (t_1)_{k_1} X_{i-k_1,j}.$$

When we apply S_2 to the rows of Y we get the image Z defined by

$$\begin{aligned} Z_{i,j} &= \sum_{k_2} (t_2)_{k_2} Y_{i,j-k_2} = \sum_{k_2} (t_2)_{k_2} \sum_{k_1} (t_1)_{k_1} X_{i-k_1,j-k_2} \\ &= \sum_{k_1} \sum_{k_2} (t_1)_{k_1} (t_2)_{k_2} X_{i-k_1,j-k_2}. \end{aligned}$$

Comparing with Equation (9.3) we see that S is given by the computational molecule with entries $a_{i,j} = (\boldsymbol{t}_1)_i(\boldsymbol{t}_2)_j$. □

Note that, when we filter an image with S_1 and S_2 in this way, the order does not matter: since computing S_1X is the same as applying S_1 to all columns of X, and computing $Y(S_2)^T$ is the same as applying S_2 to all rows of Y, the combined filtering operation, denoted S, takes the form

$$S(X) = S_1 X (S_2)^T, \tag{9.4}$$

and the fact that the order does not matter simply boils down to the fact that it does not matter which of the left or right multiplications we perform first. Applying S_1 to the columns of X is what we call a *vertical filtering operation*, while applying S_2 to the rows of X is what we call a *horizontal filtering operation*. We can thus state the following.

Observation 9.10. *Order of vertical and horizontal filtering.*

The order of vertical and horizontal filtering of an image does not matter.

Most computational molecules we will consider in the following can be expressed in terms of filters as in this theorem, but clearly there exist also computational molecules which are not on this form, since the matrix A with entries $a_{i,j} = (\boldsymbol{t}_1)_i(\boldsymbol{t}_2)_j$ has rank one, and a general computational molecule can have any rank. In most of the examples the filters are symmetric.

Assume that the image is stored as the matrix `X`. In Exercise 9.13 you will be asked to implement a function `tensor_impl` which computes the transformation $S(X) = S_1 X (S_2)^T$, where `X`, `S1`, and `S2` are input. If the computational molecule is obtained by applying the filter S_1 to the columns, and the filter S_2 to the rows, we can compute it with the following code: (we have assumed that the filter lengths are odd, and that the middle filter coefficient has index 0):

```
def S1(x):
    filterS(S1, x, True)

def S2(x):
    filterS(S2, x, True)

tensor_impl(X, S1, S2)
```

We have here used the function `filterS` to implement the filtering, so that we assume that the image is periodically or symmetrically extended. The above code uses symmetric extension, and can thus be used for symmetric filters. If the filter is non-symmetric, we should use a periodic extension instead, for which the last parameter to `filterS` should be changed.

9.3.1 Tensor product notation for operations on images

Filter-based operations on images can be written compactly using what we will call *tensor product notation*. This is part of a very general tensor product framework, and we will review parts of this framework for the sake of completeness. Let us first define the tensor product of vectors.

Definition 9.11. *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 $(\boldsymbol{x} \otimes \boldsymbol{y})_{i,j} = x_i y_j$. In other words, $\boldsymbol{x} \otimes \boldsymbol{y} = \boldsymbol{x}\boldsymbol{y}^T$.

The tensor product $\boldsymbol{x}\boldsymbol{y}^T$ is also called the *outer product* of $\boldsymbol{x}$ and $\boldsymbol{y}$ (contrary to the inner product $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \boldsymbol{x}^T \boldsymbol{y}$). In particular $\boldsymbol{x} \otimes \boldsymbol{y}$ is a matrix of rank 1, which means that most matrices cannot be written as a tensor product of two vectors. 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 9.12. *Standard basis for $L_{M,N}(\mathbb{R})$.*

Let $\mathcal{E}_M = \{\boldsymbol{e}_i\}_{i=0}^{M-1}$ $\mathcal{E}_N = \{\boldsymbol{e}_i\}_{i=0}^{N-1}$ 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 $L_{M,N}(\mathbb{R})$.

The standard basis thus consists of rank 1-matrices. An image can simply be thought of as a matrix in $L_{M,N}(\mathbb{R})$, and a computational molecule is simply a special type of linear transformation from $L_{M,N}(\mathbb{R})$ to itself. Let us also define the tensor product of matrices.

Definition 9.13. *Tensor product of matrices.*
If $S_1 : \mathbb{R}^M \to \mathbb{R}^M$ and $S_2 : \mathbb{R}^N \to \mathbb{R}^N$ are matrices, we define the linear mapping $S_1 \otimes S_2 : L_{M,N}(\mathbb{R}) \to L_{M,N}(\mathbb{R})$ by linear extension of $(S_1 \otimes S_2)(\boldsymbol{e}_i \otimes \boldsymbol{e}_j) = (S_1\boldsymbol{e}_i) \otimes (S_2\boldsymbol{e}_j)$. The linear mapping $S_1 \otimes S_2$ is called the tensor product of the matrices S_1 and S_2.

A couple of remarks are in order. First, from linear algebra we know that, when S is linear mapping from V and $S(\boldsymbol{v}_i)$ is known for a basis $\{\boldsymbol{v}_i\}_i$ of V, S 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 $S_1 \otimes S_2$ so that $(S_1 \otimes S_2)(\boldsymbol{e}_i \otimes \boldsymbol{e}_j) = (S_1\boldsymbol{e}_i) \otimes (S_2\boldsymbol{e}_j)$. This unique linear transformation is what we call the linear extension from the values in the given basis. Clearly, by linearity, also $(S_1 \otimes S_2)(\boldsymbol{x} \otimes \boldsymbol{y}) = (S_1\boldsymbol{x}) \otimes (S_2\boldsymbol{y})$, since

$$\begin{aligned}(S_1 \otimes S_2)(\boldsymbol{x} \otimes \boldsymbol{y}) &= (S_1 \otimes S_2)((\sum_i x_i\boldsymbol{e}_i) \otimes (\sum_j y_j\boldsymbol{e}_j)) = (S_1 \otimes S_2)(\sum_{i,j} x_iy_j(\boldsymbol{e}_i \otimes \boldsymbol{e}_j)) \\ &= \sum_{i,j} x_iy_j(S_1 \otimes S_2)(\boldsymbol{e}_i \otimes \boldsymbol{e}_j) = \sum_{i,j} x_iy_j(S_1\boldsymbol{e}_i) \otimes (S_2\boldsymbol{e}_j) \\ &= \sum_{i,j} x_iy_jS_1\boldsymbol{e}_i((S_2\boldsymbol{e}_j))^T = S_1(\sum_i x_i\boldsymbol{e}_i)(S_2(\sum_j y_j\boldsymbol{e}_j))^T \\ &= S_1\boldsymbol{x}(S_2\boldsymbol{y})^T = (S_1\boldsymbol{x}) \otimes (S_2\boldsymbol{y}).\end{aligned}$$

Here we used the result from Exercise 9.17. We can now prove the following.

Theorem 9.14. *Compact filter notation and computational molecules.*
If $S_1 : \mathbb{R}^M \to \mathbb{R}^M$ and $S_2 : \mathbb{R}^N \to \mathbb{R}^N$ are matrices of linear transformations, then $(S_1 \otimes S_2)X = S_1X(S_2)^T$ for any $X \in L_{M,N}(\mathbb{R})$. In particular $S_1 \otimes S_2$ is the operation which applies S_1 to the columns of X, and S_2 to the resulting rows. In other words, if S_1, S_2 have compact filter notations $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$, respectively, then $S_1 \otimes S_2$ has computational molecule $\boldsymbol{t}_1 \otimes \boldsymbol{t}_2$.

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 $(0, 0)$.

Proof. We have that

$$\begin{aligned}(S_1 \otimes S_2)(\boldsymbol{e}_i \otimes \boldsymbol{e}_j) &= (S_1\boldsymbol{e}_i) \otimes (S_2\boldsymbol{e}_j) = S_1\boldsymbol{e}_i((S_2\boldsymbol{e}_j))^T \\ &= S_1\boldsymbol{e}_i(\boldsymbol{e}_j)^T(S_2)^T = S_1(\boldsymbol{e}_i \otimes \boldsymbol{e}_j)(S_2)^T.\end{aligned}$$

This means that $(S_1 \otimes S_2)X = S_1 X (S_2)^T$ for any $X \in L_{M,N}(\mathbb{R})$ also, since equality holds on the basis vectors $\boldsymbol{e}_i \otimes \boldsymbol{e}_j$. Since the matrix A with entries $a_{i,j} = (\boldsymbol{t}_1)_i (\boldsymbol{t}_2)_j$ also can be written as $\boldsymbol{t}_1 \otimes \boldsymbol{t}_2$, the result follows. □

We have thus shown that we alternatively can write $S_1 \otimes S_2$ for the operations we have considered. This notation also makes it easy to combine several two-dimensional filtering operations:

Corollary 9.15. *Composing tensor products.*
We have that $(S_1 \otimes T_1)(S_2 \otimes T_2) = (S_1 S_2) \otimes (T_1 T_2)$.

Proof. By Theorem 9.14 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 $S_1 \otimes S_2$ to an image. We can factorize $S_1 \otimes S_2$ as

$$S_1 \otimes S_2 = (S_1 \otimes I)(I \otimes S_2) = (I \otimes S_2)(S_1 \otimes I). \tag{9.5}$$

Moreover, since

$$(S_1 \otimes I)X = S_1 X \qquad (I \otimes S_2)X = X(S_2)^T = (S_2 X^T)^T,$$

$S_1 \otimes I$ is a vertical filtering operation, and $I \otimes S_2$ is a horizontal filtering operation in this factorization. For filters we have an even stronger result: If S_1, S_2, S_3, S_4 all are filters, we have from Corollary 9.15 that $(S_1 \otimes S_2)(S_3 \otimes S_4) = (S_3 \otimes S_4)(S_1 \otimes S_2)$, since all filters commute. This does not hold in general since general matrices do not commute.

We will now consider two important examples of filtering operations on images: smoothing and edge detection/computing partical derivatives. For all examples we will use the tensor product notation for these operations.

Example 9.9: Smoothing an image

When we considered filtering of digital sound, low-pass filters dampened high frequencies. We will here similarly see that an image can be smoothed by applying a low-pass filters to the rows and the columns. Let us consider such computational molecules. In particular, let us as before take filter coefficients taken from Pascal's triangle. If we use the filter $S = \frac{1}{4}\{1, \underline{2}, 1\}$ (row 2 from Pascal's triangle), Theorem 9.9 gives the computational molecule

$$A = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & \underline{4} & 2 \\ 1 & 2 & 1 \end{pmatrix}. \tag{9.6}$$

If the pixels in the image are $p_{i,j}$, this means that we compute the new pixels by

$$\hat{p}_{i,j} = \frac{1}{16}\big(4p_{i,j} + 2(p_{i,j-1} + p_{i-1,j} + p_{i+1,j} + p_{i,j+1}) + p_{i-1,j-1} + p_{i+1,j-1} + p_{i-1,j+1} + p_{i+1,j+1}\big).$$

If we instead use the filter $S = \frac{1}{64}\{1, 6, 15, \underline{20}, 15, 6, 1\}$ (row 6 from Pascal's triangle), we get the computational molecule

$$\frac{1}{4096}\begin{pmatrix} 1 & 6 & 15 & 20 & 15 & 6 & 1 \\ 6 & 36 & 90 & 120 & 90 & 36 & 6 \\ 15 & 90 & 225 & 300 & 225 & 90 & 15 \\ 20 & 120 & 300 & \underline{400} & 300 & 120 & 20 \\ 15 & 90 & 225 & 300 & 225 & 90 & 15 \\ 6 & 36 & 90 & 120 & 90 & 36 & 6 \\ 1 & 6 & 15 & 20 & 15 & 6 & 1 \end{pmatrix}. \tag{9.7}$$

We anticipate that both molecules give a smoothing effect, but that the second molecules provides more smoothing. The result of applying the two molecules in (9.6) and (9.7) to our greyscale-image is shown in the two right images in Figure 9.11. With the help of the function `tensor_impl`, smoothing with the first molecule (9.6) above can be obtained by writing

```
def S(x):
    filterS([1., 2., 1.]/4., x, True);

tensor_impl(X, S, S)
```

To make the smoothing effect visible, we have zoomed in on the face in the image. The smoothing effect is clarly best visible in the second image.

Figure 9.11: The two right images show the effect of smoothing the left image.

Smoothing effects are perhaps more visible if we use a simple image, as the one in the left part of Figure 9.12.

Again we have used the filter $S = \frac{1}{4}\{1, \underline{2}, 1\}$. Here we also have shown what happens if we only smooth the image in one of the directions. In the right

Figure 9.12: The results of smoothing the simple image to the left in three different ways.

image we have smoothed in both directions. We clearly see the union of the two one-dimensional smoothing operations then.

Example 9.10: Edge detection

Another operation on images which can be expressed in terms of computational molecules 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 since a grey-level image is a scalar function of two variables, we have a perfect situation for applying numerical differentiation techniques.

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 used when plotting functions. This means that the familiar symmetric Newton quotient approximation for the partial derivative [31] takes the form

$$\frac{\partial P}{\partial x}(i,j) \approx \frac{p_{i,j+1} - p_{i,j-1}}{2}, \tag{9.8}$$

where we have used the convention $h = 1$ which means that the derivative is measured in terms of 'intensity per pixel'. This corresponds to applying the bass-reducing filter $S = \frac{1}{2}\{1, \underline{0}, -1\}$ to all the rows (alternatively, applying the tensor product $I \otimes S$ to the image). We can thus express this in terms of the computational molecule.

$$\frac{1}{2}\begin{pmatrix} 0 & 0 & 0 \\ 1 & \underline{0} & -1 \\ 0 & 0 & 0 \end{pmatrix}.$$

We have included the two rows of 0s just to make it clear how the computational molecule is to be interpreted when we place it over the pixels. Let us first apply this molecule to the usual excerpt of the Lena image. This gives the first image in Figure 9.13. This images shows many artefacts since the pixel values

lie outside the legal range: many of the intensities are in fact negative. More specifically, the intensities turn out to vary in the interval $[-0.424, 0.418]$. Let us therefore normalise and map all intensities to $[0, 1]$. This gives the second image in Figure 9.13. The predominant color 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 (9.1) to each intensity value. The result is shown in the third image in Figure 9.13. This does indeed show more detail.

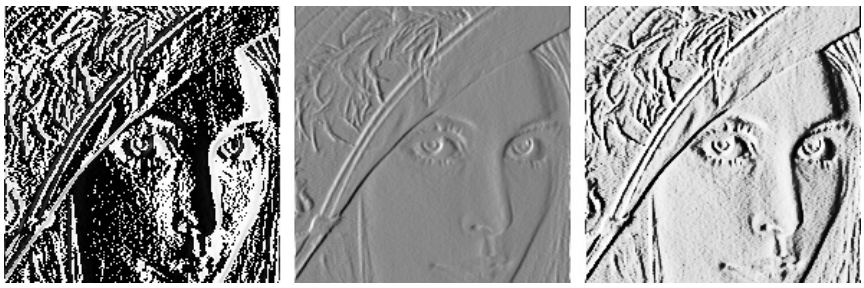

Figure 9.13: Experimenting with the partial derivative in the x-direction for the image in 9.6. The left image has artefacts, since the pixel values are outside the legal range. We therefore normalize the intensities to lie in $[0, 255]$ (middle), before we increase the contrast (right).

It is important to understand the colors 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 the left image in Figure 9.13 corresponds to no change in intensity between the two pixels.

When the values are mapped to the interval $[0, 1]$ in the second image, 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 the third image these values have just been emphasised even more.

The third image 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.

Partial derivative in y-direction. The partial derivative $\partial P/\partial y$ can be computed analogously to $\partial P/\partial x$, i.e. we apply the filter $-S = \frac{1}{2}\{-1, \underline{0}, 1\}$ to

all columns of the image (alternatively, apply the tensor product $-S \otimes I$ to the image), where S is the filter which we used for edge detection in the x-direction. 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. We can express this in terms of the computational molecule

$$\frac{1}{2}\begin{pmatrix} 0 & 1 & 0 \\ 0 & \underline{0} & 0 \\ 0 & -1 & 0 \end{pmatrix}.$$

Let us compare the partial derivatives in both directions. The result is shown in Figure 9.14.

Figure 9.14: The first-order partial derivatives in the x- and y-direction, respectively. In both images, the computed numbers have been normalised and the contrast enhanced.

The intensities have been normalised and the contrast enhanced by the function f_{50} from Equation (9.1).

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. Note that, as for the first order derivatives, it is possible for the length of the gradient to be outside the legal range of values. The computed gradient values, the gradient mapped to the legal range, and the gradient with contrast adjusted, are shown in Figure 9.15.

Figure 9.15: The computed gradient (left). In the middle the intensities have been normalised to the $[0, 255]$, and to the right the contrast has been increased.

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 the parts of the image that have virtually constant intensity (partial derivatives close to 0) are colored black. In the images of the partial derivatives these values ended up in the middle of the range of intensity values, with a final color of grey, since there were both positive and negative values. To enhance the contrast for this image we should thus do something different from what was done in the other images, since we now have a large number of intensities near 0. The solution was to apply a function like the ones shown in the right plot in Figure 9.8. Here we have used the function $g_{0.01}$.

Figure 9.14 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.

Example 9.11: Second-order derivatives

To compute the three second order derivatives we can combine the two computational molecules which we already have described. For the mixed second

order derivative we get $(I \otimes S)((-S) \otimes I) = -S \otimes S$. For the last two second order derivative $\frac{\partial^2 P}{\partial x^2}$, $\frac{\partial^2 P}{\partial y^2}$, we can also use the three point approximation to the second derivative [31]

$$\frac{\partial P}{\partial x^2}(i,j) \approx p_{i,j+1} - 2p_{i,j} + p_{i,j-1} \tag{9.9}$$

(again we have set $h = 1$). This gives a smaller molecule than if we combine the two molecules for order one differentiation (i.e. $(I \otimes S)(I \otimes S) = (I \otimes S^2)$ and $((-S) \otimes I)((-S) \otimes I) = (S^2 \otimes I)$), since $S^2 = \frac{1}{2}\{1, \underline{0}, -1\}\frac{1}{2}\{1, \underline{0}, -1\} = \frac{1}{4}\{1, 0, \underline{-2}, 0, 1\}$. The second order derivatives of an image P can thus be computed by applying the computational molecules

$$\frac{\partial^2 P}{\partial x^2}: \quad \begin{pmatrix} 0 & 0 & 0 \\ 1 & \underline{-2} & 1 \\ 0 & 0 & 0 \end{pmatrix}, \tag{9.10}$$

$$\frac{\partial^2 P}{\partial y \partial x}: \quad \frac{1}{4}\begin{pmatrix} -1 & 0 & 1 \\ 0 & \underline{0} & 0 \\ 1 & 0 & -1 \end{pmatrix}, \tag{9.11}$$

$$\frac{\partial^2 P}{\partial y^2}: \quad \begin{pmatrix} 0 & 1 & 0 \\ 0 & \underline{-2} & 0 \\ 0 & 1 & 0 \end{pmatrix}. \tag{9.12}$$

With these molecules it is quite easy to compute the second-order derivatives. The results are shown in Figure 9.16.

Figure 9.16: The second-order partial derivatives in the xx-, xy-, and yy-directions, respectively. In all images, the computed numbers have been normalised and the contrast enhanced.

The computed derivatives were first normalised and then the contrast enhanced with the function f_{100} in each image, see equation (9.1).

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.

This procedure can be generalized to higher order derivatives also. To apply $\frac{\partial^{k+l} P}{\partial x^k \partial y^l}$ to an image we can compute $S_l \otimes S_k$ where S_r corresponds to any point method for computing the r'th order derivative. We can also compute $(S^l) \otimes (S^k)$, where we iterate the filter $S = \frac{1}{2}\{1, \underline{0}, -1\}$ for the first derivative, but this gives longer filters.

Example 9.12: Chess pattern image

Let us apply the molecules for differentiation to a chess pattern test image. In Figure 9.17 we have applied $S \otimes I$, $I \otimes S$, and $S \otimes S$, $I \otimes S^2$, and $S^2 \otimes I$ to the example image shown in the upper left.

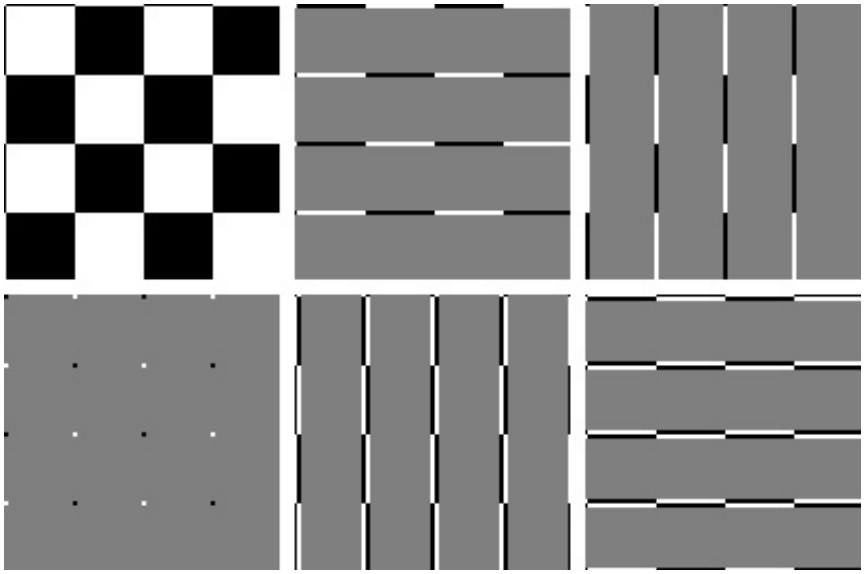

Figure 9.17: Different tensor products applied to the simple chess pattern image shown in the upper left.

These images make it is clear that $S \otimes I$ detects all horizontal edges, $I \otimes S$ detects all vertical edges, and that $S \otimes S$ detects all points where abrupt changes appear in both directions. We also see that the second order partial derivative detects exactly the same edges which the first order partial derivative found. Note that the edges detected with $I \otimes S^2$ are wider than the ones detected with $I \otimes S$. The reason is that the filter S^2 has more filter coefficients than S. Also, edges are detected with different colors. This reflects whether the difference between the neighbouring pixels is positive or negative. The values after we have applied the tensor product may thus not lie in the legal range

of pixel values (since they may be negative). The figures have taken this into account by mapping the values back to a legal range of values, as we did in Chapter 9. Finally, we also see additional edges at the first and last rows/edges in the images. The reason is that the filter S is defined by assuming that the pixels repeat periodically (i.e. it is a circulant Toeplitz matrix). Due to this, we have additional edges at the first/last rows/edges. This effect can also be seen in Chapter 9, although there we did not assume that the pixels repeat periodically.

Defining a two-dimensional filter by filtering columns and then rows is not the only way we can define a two-dimensional filter. Another possible way is to let the $MN \times MN$-matrix itself be a filter. Unfortunately, this is a bad way to define filtering of an image, since there are some undesirable effects near the boundaries between rows: in the vector we form, the last element of one row is followed by the first element of the next row. These boundary effects are unfortunate when a filter is applied.

Exercise 9.13: Implement a tensor product

Implement a function `tensor_impl` which takes a matrix `X`, and functions `S1` and `S2` as parameters, and applies `S1` to the columns of `X`, and `S2` to the rows of `X`.

Exercise 9.14: Generate images

Write code which calls the function `tensor_impl` with appropriate filters and which generate the following images:

a) The right image in Figure 9.11.

b) The right image in Figure 9.13.

c) The images in Figure 9.15.

d) The images in Figure 9.14.

e) The images in Figure 9.16.

Exercise 9.15: Interpret tensor products

Let the filter S be defined by $S = \{\underline{-1}, 1\}$.

a) Let X be a matrix which represents the pixel values in an image. What can you say about how the new images $(S \otimes I)X$ og $(I \otimes S)X$ look? What are the interpretations of these operations?

b) Write down the $4 \otimes 4$-matrix $X = (1,1,1,1) \otimes (0,0,1,1)$. Compute $(S \otimes I)X$ by applying the filters to the corresponding rows/columns of X as we have learned, and interpret the result. Do the same for $(I \otimes S)X$.

Exercise 9.16: Computational molecule of moving average filter

Let S be the moving average filter of length $2L+1$, i.e. $T = \frac{1}{L}\{\underbrace{1, \cdots, 1, \underline{1}, 1, \cdots, 1}_{2L+1 \text{ times}}\}$.

What is the computational molecule of $S \otimes S$?

Exercise 9.17: Bilinearity of the tensor product

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 $F(\boldsymbol{x}, \alpha \boldsymbol{y}_1 + \beta \boldsymbol{y}_2) = \alpha F(\boldsymbol{x}, \boldsymbol{y}_1) + \beta F(\boldsymbol{x}, \boldsymbol{y}_2)$.

Exercise 9.18: Attempt to write as tensor product

Attempt to find matrices $S_1 : \mathbb{R}^M \to \mathbb{R}^M$ and $S_2 : \mathbb{R}^N \to \mathbb{R}^N$ so that the following mappings from $L_{M,N}(\mathbb{R})$ to $L_{M,N}(\mathbb{R})$ can be written on the form $X \to S_1 X (S_2)^T = (S_1 \otimes S_2)X$. In all the cases, it may be that no such S_1, S_2 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.

Exercise 9.19: Computational molecules

Let the filter S be defined by $S = \{1, \underline{2}, 1\}$.

a) Write down the computational molecule of $S \otimes S$.

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 $(S \otimes S)A$ by applying the filter S to every row and column in the matrix the way we have learned. If the matrix A was more generally an image, what can you say about how the new image will look?

Exercise 9.20: Computational molecules 2

Let $S = \frac{1}{4}\{1, \underline{2}, 1\}$ be a filter.

a) What is the effect of applying the tensor products $S \otimes I$, $I \otimes S$, and $S \otimes S$ on an image represented by the matrix X?

b) Compute $(S \otimes S)(\boldsymbol{x} \otimes \boldsymbol{y})$, where $\boldsymbol{x} = (4, 8, 8, 4)$, $\boldsymbol{y} = (8, 4, 8, 4)$ (i.e. both $\boldsymbol{x}$ and $\boldsymbol{y}$ are column vectors).

Exercise 9.21: Comment on code

Suppose that we have an image given by the $M \times N$-matrix `X`, and consider the following code:

```
for n in range(N):
    X[0, n] = 0.25*X[N-1, n] + 0.5*X[0, n] + 0.25*X[1, n]
    X[1:(N-1), n] = 0.25*X[0:(N-2), n] + 0.5*X[1:(N-1), n] \
    + 0.25*X[2:N, n]
    X[N-1, n] = 0.25*X[N-2, n] + 0.5*X[N-1, n] + 0.25*X[0, n]
for m in range(m):
    X[m, 0] = 0.25*X[m, M-1] + 0.5*X[m, 0] + 0.25*X[m, 1]
    X[m, 1:(M-1)] = 0.25*X[m, 0:(M-2)] + 0.5*X[m, 1:(M-1)] \
    + 0.25*X[m, 2:M]
    X[m, M-1] = 0.25*X[m, M-2] + 0.5*X[m, M-1] + 0.25*X[m, 0]
```

Which tensor product is applied to the image? Comment what the code does, in particular the first and third line in the inner `for`-loop. What effect does the code have on the image?

Exercise 9.22: Eigenvectors of tensor products

Let $\boldsymbol{v}_A$ be an eigenvector of A with eigenvalue λ_A, and $\boldsymbol{v}_B$ an eigenvector of B with eigenvalue λ_B. Show that $\boldsymbol{v}_A \otimes \boldsymbol{v}_B$ is an eigenvector of $A \otimes B$ with eigenvalue $\lambda_A \lambda_B$. Explain from this why $\|A \otimes B\| = \|A\|\|B\|$, where $\|\cdot\|$ denotes the operator norm of a matrix.

Exercise 9.23: The Kronecker product

The *Kronecker tensor product* of two matrices A and B, written $A \otimes^k B$, is defined as

$$A \otimes^k B = \begin{pmatrix} a_{1,1}B & a_{1,2}B & \cdots & a_{1,M}B \\ a_{2,1}B & a_{2,2}B & \cdots & a_{2,M}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{p,1}B & a_{p,2}B & \cdots & a_{p,M}B \end{pmatrix},$$

where the entries of A are $a_{i,j}$. The tensor product of a $p \times M$-matrix, and a $q \times N$-matrix is thus a $(pq) \times (MN)$-matrix. Note that this tensor product in particular gives meaning for vectors: if $\boldsymbol{x} \in \mathbb{R}^M$, $\boldsymbol{y} \in \mathbb{R}^N$ are column vectors, then $\boldsymbol{x} \otimes^k \boldsymbol{y} \in \mathbb{R}^{MN}$ is also a column vector. In this exercise we will investigate how the Kronecker tensor product is related to tensor products as we have defined them in this section.

a) Explain that, if $\boldsymbol{x} \in \mathbb{R}^M$, $\boldsymbol{y} \in \mathbb{R}^N$ are column vectors, then $\boldsymbol{x} \otimes^k \boldsymbol{y}$ is the column vector where the rows of $\boldsymbol{x} \otimes \boldsymbol{y}$ have first been stacked into one large row vector, and this vector transposed. The linear extension of the operation defined by

$$\boldsymbol{x} \otimes \boldsymbol{y} \in \mathbb{R}^{M,N} \to \boldsymbol{x} \otimes^k \boldsymbol{y} \in \mathbb{R}^{MN}$$

thus stacks the rows of the input matrix into one large row vector, and transposes the result.

b) Show that $(A \otimes^k B)(\boldsymbol{x} \otimes^k \boldsymbol{y}) = (A\boldsymbol{x}) \otimes^k (B\boldsymbol{y})$. We can thus use any of the defined tensor products $\otimes$, $\otimes^k$ to produce the same result, i.e. we have the commutative diagram shown in Figure 9.18, where the vertical arrows represent stacking the rows in the matrix, and transposing, and the horizontal arrows represent the two tensor product linear transformations we have defined. In particular, we can compute the tensor product in terms of vectors, or in terms of matrices, and it is clear that the Kronecker tensor product gives the matrix of tensor product operations.

$$\begin{array}{ccc} \boldsymbol{x} \otimes \boldsymbol{y} & \xrightarrow{A \otimes B} & (A\boldsymbol{x}) \otimes (B\boldsymbol{y}) \\ \downarrow & & \downarrow \\ \boldsymbol{x} \otimes^k \boldsymbol{y} & \xrightarrow{A \otimes^k B} & (A\boldsymbol{x}) \otimes^k (B\boldsymbol{y}), \end{array}$$

Figure 9.18: Tensor products

c) Using the Euclidean inner product on $L(M, N) = \mathbb{R}^{MN}$, i.e.

$$\langle X, Y \rangle = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} X_{i,j} \overline{Y_{i,j}}.$$

and the correspondence in a) we can define the inner product of $\boldsymbol{x}_1 \otimes \boldsymbol{y}_1$ and $\boldsymbol{x}_2 \otimes \boldsymbol{y}_2$ by

$$\langle \boldsymbol{x}_1 \otimes \boldsymbol{y}_1, \boldsymbol{x}_2 \otimes \boldsymbol{y}_2 \rangle = \langle \boldsymbol{x}_1 \otimes^k \boldsymbol{y}_1, \boldsymbol{x}_2 \otimes^k \boldsymbol{y}_2 \rangle.$$

Show that

$$\langle \boldsymbol{x}_1 \otimes \boldsymbol{y}_1, \boldsymbol{x}_2 \otimes \boldsymbol{y}_2 \rangle = \langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle \langle \boldsymbol{y}_1, \boldsymbol{y}_2 \rangle.$$

Clearly this extends linearly to an inner product on $L_{M,N}$.

d) Show that the FFT factorization can be written as

$$\begin{pmatrix} F_{N/2} & D_{N/2} F_{N/2} \\ F_{N/2} & -D_{N/2} F_{N/2} \end{pmatrix} = \begin{pmatrix} I_{N/2} & D_{N/2} \\ I_{N/2} & -D_{N/2} \end{pmatrix} (I_2 \otimes^k F_{N/2}).$$

Also rewrite the sparse matrix factorization for the FFT from Equation (2.18) in terms of tensor products.

9.4 Change of coordinates in tensor products

Filter-based operations were not the only operations we considered for sound. We also considered the DFT, the DCT, and the wavelet transform, which were changes of coordinates which gave us useful frequency- or time-frequency information. We would like to define similar changes of coordinates for images, which also give useful such information. Tensor product notation will also be useful in this respect, and we start with the following result.

Theorem 9.16. *The basis $\mathcal{B}_1 \otimes \mathcal{B}_2$.*

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 9.17). This means that

$$\boldsymbol{0} = \sum_{(i,j)=(0,0)}^{(M-1,N-1)} \alpha_{i,j}(\boldsymbol{v}_i \otimes \boldsymbol{w}_j) = \sum_{i=0}^{M-1} \boldsymbol{v}_i \otimes \boldsymbol{h}_i = \sum_{i=0}^{M-1} \boldsymbol{v}_i \boldsymbol{h}_i^T.$$

Column k in this matrix equation says $\boldsymbol{0} = \sum_{i=0}^{M-1} h_{i,k}\boldsymbol{v}_i$, 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 $\sum_{j=0}^{N-1} \alpha_{i,j}\boldsymbol{w}_j = \boldsymbol{0}$ 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 9.17. *Coordinate matrix.*

Let $\mathcal{B} = \{\boldsymbol{b}_i\}_{i=0}^{M-1}$, $\mathcal{C} = \{\boldsymbol{c}_j\}_{j=0}^{N-1}$ be bases for $\mathbb{R}^M$ and $\mathbb{R}^N$, and let $A \in L_{M,N}(\mathbb{R})$. By the coordinate matrix of A in $\mathcal{B} \otimes \mathcal{C}$ we mean the $M \times N$-matrix X (with components X_{kl}) such that $A = \sum_{k,l} X_{k,l}(\boldsymbol{b}_k \otimes \boldsymbol{c}_l)$.

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 9.18. *Change of coordinates in tensor products.*
Assume that

- $\mathcal{B}_1, \mathcal{C}_1$ are bases for $\mathbb{R}^M$, and that S_1 is the change of coordinates matrix from $\mathcal{B}_1$ to $\mathcal{C}_1$,
- $\mathcal{B}_2, \mathcal{C}_2$ are bases for $\mathbb{R}^N$, and that S_2 is the change of coordinates matrix from $\mathcal{B}_2$ to $\mathcal{C}_2$.

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 $\mathcal{B}_1 \otimes \mathcal{B}_2$, and Y the coordinate matrix in $\mathcal{C}_1 \otimes \mathcal{C}_2$, then the change of coordinates from $\mathcal{B}_1 \otimes \mathcal{B}_2$ to $\mathcal{C}_1 \otimes \mathcal{C}_2$ can be computed as

$$Y = S_1 X (S_2)^T. \tag{9.13}$$

Proof. Denote the change of coordinates from $\mathcal{B}_1 \otimes \mathcal{B}_2$ to $\mathcal{C}_1 \otimes \mathcal{C}_2$ by S. Since any change of coordinates is linear, it is enough to show that $S(\boldsymbol{e}_i \otimes \boldsymbol{e}_j) = S_1(\boldsymbol{e}_i \otimes \boldsymbol{e}_j)(S_2)^T$ for any i, j. We can write

$$\begin{aligned}
\boldsymbol{b}_{1i} \otimes \boldsymbol{b}_{2j} &= \left(\sum_k (S_1)_{k,i} \boldsymbol{c}_{1k} \right) \otimes \left(\sum_l (S_2)_{l,j} \boldsymbol{c}_{2l} \right) = \sum_{k,l} (S_1)_{k,i} (S_2)_{l,j} (\boldsymbol{c}_{1k} \otimes \boldsymbol{c}_{2l}) \\
&= \sum_{k,l} (S_1)_{k,i} ((S_2)^T)_{j,l} (\boldsymbol{c}_{1k} \otimes \boldsymbol{c}_{2l}) = \sum_{k,l} (S_1 \boldsymbol{e}_i (\boldsymbol{e}_j)^T (S_2)^T)_{k,l} (\boldsymbol{c}_{1k} \otimes \boldsymbol{c}_{2l}) \\
&= \sum_{k,l} (S_1 (\boldsymbol{e}_i \otimes \boldsymbol{e}_j) (S_2)^T)_{k,l} (\boldsymbol{c}_{1k} \otimes \boldsymbol{c}_{2l})
\end{aligned}$$

This shows that the coordinate matrix of $\boldsymbol{b}_{1i} \otimes \boldsymbol{b}_{2j}$ in $\mathcal{C}_1 \otimes \mathcal{C}_2$ is $S_1(\boldsymbol{e}_i \otimes \boldsymbol{e}_j)(S_2)^T$. Since the coordinate matrix of $\boldsymbol{b}_{1i} \otimes \boldsymbol{b}_{2j}$ in $\mathcal{B}_1 \otimes \mathcal{B}_2$ is $\boldsymbol{e}_i \otimes \boldsymbol{e}_j$, this shows that $S(\boldsymbol{e}_i \otimes \boldsymbol{e}_j) = S_1(\boldsymbol{e}_i \otimes \boldsymbol{e}_j)(S_2)^T$. The result follows. □

In both cases of filtering and change of coordinates in tensor products, we see that we need to compute the mapping $X \to S_1 X (S_2)^T$. As we have seen, this amounts to a row/column-wise operation, which we restate as follows:

Observation 9.19. *Change of coordinates in tensor products.*
The change of coordinates from $\mathcal{B}_1 \otimes \mathcal{B}_2$ to $\mathcal{C}_1 \otimes \mathcal{C}_2$ can be implemented as follows:

- For every column in the coordinate matrix in $\mathcal{B}_1 \otimes \mathcal{B}_2$, perform a change of coordinates from $\mathcal{B}_1$ to $\mathcal{C}_1$.
- For every row in the resulting matrix, perform a change of coordinates from $\mathcal{B}_2$ to $\mathcal{C}_2$.

We can again use the function `tensor_impl` in order to implement change of coordinates for a tensor product. We just need to replace the filters with the functions `S1` and `S2` for computing the corresponding changes of coordinates:

```
tensor_impl(X, S1, S2)
```

The operation $X \to (S_1)X(S_2)^T$, which we now have encountered in two different ways, is one particular type of linear transformation from $\mathbb{R}^{N^2}$ to itself (see Exercise 9.23 for how the matrix of this linear transformation can be constructed). While a general such linear transformation requires N^4 multiplications (i.e. when we perform a full matrix multiplication), $X \to (S_1)X(S_2)^T$ can be implemented generally with only $2N^3$ multiplications (since multiplication of two $N \times N$-matrices require N^3 multiplications in general). The operation $X \to (S_1)X(S_2)^T$ is thus computationally simpler than linear transformations in general. In practice the operations S_1 and S_2 are also computationally simpler, since they can be filters, FFT's, or wavelet transformations, so that the complexity in $X \to (S_1)X(S_2)^T$ can be even lower.

In the following examples, we will interpret the pixel values in an image as coordinates in the standard basis, and perform a change of coordinates.

Example 9.24: Change of coordinates with the DFT

The DFT is one particular change of coordinates which we have considered. It was the change of coordinates from the standard basis to the Fourier basis. A corresponding change of coordinates in a tensor product is obtained by substituting the DFT as the functions S_1, S_2 for implementing the changes of coordinates above. 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 the image is split into smaller squares of appropriate size, called blocks, and a change of coordinates is performed independently for each block. In this example we have split the image into blocks of size 8×8.

Recall that 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. Let us introduce a neglection threshold in the same way as in Example 2.17, to view the image after we set certain frequencies to zero. As for sound, this 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 *= (abs(X) >= threshold)
```

`abs(X)>=threshold` now instead returns a *threshold matrix* with 1 and 0 of the same size as `X`.

In Figure 9.19 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 76.6% of the samples have been zeroed out. A blocking effect at the block boundaries is clearly visible.

Figure 9.19: The effect on an image when it is transformed with the DFT, and the DFT-coefficients below a certain threshold are zeroed out. The threshold has been increased from left to right, from 100, to 200, and 400. The percentage of pixel values that were zeroed out are 76.6, 89.3, and 95.3, respectively.

Example 9.25: 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. Change of coordinates in tensor products between the standard basis and the DCT basis is obtained by substituting with the DCT and the IDCT for the changes of coordinates S_1, S_2 above.

The DCT is used more than the DFT in image processing. In particular, the JPEG standard applies a two-dimensional DCT, rather than a two-dimensional DFT. With the JPEG standard, the blocks are always 8×8, as in the previous example. It is of course not a coincidence that a power of 2 is chosen here: the DCT, as the DFT, has an efficient implementation for powers of 2.

If we follow the same strategy for the DCT as for the DFT example, so that we zero out DCT-coefficients which are below a given threshold [1], and use the same block sizes, we get the images shown in Figure 9.20. We see similar effects as with 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 9.21, 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.

Exercise 9.26: Implement DFT and DCT on blocks

In this section we have used functions which apply the DCT and the DFT either to subblocks of size 8×8, or to the full image. Implement functions `DFTImpl8`,

[1] The JPEG standard does not do exactly the kind of thresholding described here. Rather it performs what is called a quantization.

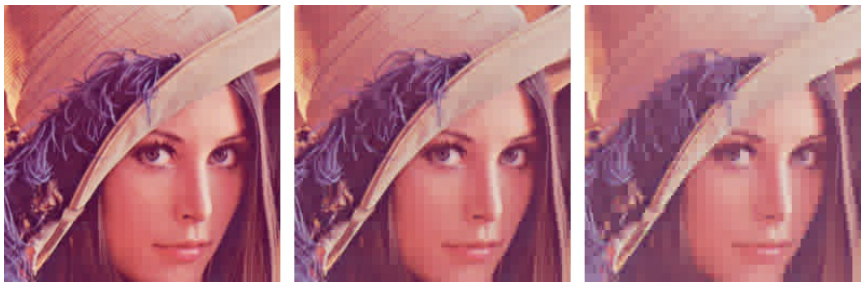

Figure 9.20: The effect on an image when it is transformed with the DCT, and the DCT-coefficients below a certain threshold are zeroed out. The threshold has been increased from left to right, from 30, to 50, and 100. The percentage of pixel values that were zeroed out are 93.2, 95.8, and 97.7, respectively.

Figure 9.21: The effect on an image when it is transformed with the DCT, and the DCT-coefficients below a certain threshold are zeroed out. The image has not been split into blocks here, and the same thresholds as in Figure 9.20 were used. The percentage of pixel values that were zeroed out are 93.2, 96.6, and 98.8, respectively.

`IDFTImpl8`, `DCTImpl8`, and `IDCTImpl8` which apply the DFT, IDFT, DCT, and IDCT, to consecutive segments of length 8.

Exercise 9.27: Implement two-dimensional FFT and DCT

Implement functions `DFTImplFull`, `IDFTImplFull`, `DCTImplFull`, and `IDCTImplFull` which applies the DFT, IDFT, DCT, and IDCT, to the entire vector, and use these to implement. FFT2, IFFT2, DCT2, and IDCT2 on an image, with the help of the function `tensor_impl`.

Exercise 9.28: Zeroing out DCT coefficients

The function `forw_comp_rev_DCT2` in the module `forw_comp_rev` applies the DCT to a part of our sample image, and sets DCT coefficients below a certain threshold to be zero. This is very similar to what the JPEG standard does.

Run `forw_comp_rev_DCT2` for different threshold parameters, and with the functions `DCTImpl8`, `IDCTImpl8`, `DCTImplFull`, and `IDCTImplFull` as parameters. Check that this reproduces the DCT test images of this section, and that the correct numbers of values which have been neglected (i.e. which are below the threshold) are printed on screen.

Exercise 9.29: Comment code

Suppose that we have given an image by the matrix `X`. Consider the following code:

```
threshold = 30
[M, N] = shape(X)[0:2]
for n in range(N):
    FFTImpl(X[:, n], FFTKernelStandard)
for m in range(M):
    FFTImpl(X[m, :], FFTKernelStandard)

X = X.*(abs(X) >= threshold)

for n in range(N):
    FFTImpl(X[:, n], FFTKernelStandard, 0)
for m in range(M):
    FFTImpl((X[m, :], FFTKernelStandard, 0)
```

Comment what the code does. Comment in particular on the meaning of the parameter `threshold`, and what effect this has on the image.

9.5 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. Many of these operations could be described in terms of a row/column-wise application of filters, and more generally in term of what we called computational molecules. We defined the tensor product, and saw how our operations could be expressed within this framework. The tensor product framework could also be used to state change of coordinates for images, so that we could consider changes of coordinates such as the DFT and the DCT also for images. The algorithm for computing filtering operations or changes of coordinates for images turned out to be similar, in the sense that the one-dimensional counterparts were simply assplied to the rows and the columns in the image.

In introductory image processing textbooks, many other image processing methods are presented. We have limited to the techniques presented here, since

our interest in images is mainly for transformation operations which are useful for compression. An excellent textbook on image processing which uses Matlab is [18]. This contains important topics such as image restoration and reconstruction, geometric transformations, morphology, and object recognition. None of these are considered in this book.

In much literature, one only mentions that filtering can be extended to images by performing one-dimensional filtering for the rows, followed by one-dimensional filtering for the columns, without properly explaining why this is the natural thing to do. The tensor product may be the most natural concept to explain this, and a concept which is firmly established in mathematical literature. Tensor products are usually not part of beginning courses in linear algebra. We have limited the focus here to an introduction to tensor products, and the theory needed to explain filtering an image, and computing the two-dimensional wavelet transform. Some linear algebra books (such as [30]) present tensor products in exercise form only, and often only mentions the Kronecker tensor product, as we defined it.

Many international standards exist for compression of images, and we will take a closer look at two of them in this book. The JPEG standard, perhaps the most popular format for images on the Internet, applies a change of coordinates with a two-dimensional DCT, as described in this chapter. 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. JPEG is usually lossy, but may also be lossless and has become. The standard defines both the algorithms for encoding and decoding and the storage format. The extension of a JPEG-file is `.jpg` or `.jpeg`. JPEG is short for *Joint Photographic Experts Group*, and was approved as an international standard in 1994. A more detailed description of the standard can be found in [36].

The second standard we will consider is JPEG2000. It was developed to address some of the shortcomings of JPEG, and is based on wavelets. The standard document for this [21] does not focus on explaining the theory behind the standard. As the MP3 standard document, it rather states step-by-step procedures for implementing the standard.

The theory we present related to these image standards concentrate on transforming the image (either with a DWT or a DCT) to obtain something which is more suitable for (lossless or lossy) compression. However, many other steps are also needed in order to obtain a full image compression system. One of these is *quantization*. In the simplest form of quantization, every resulting sample from the transformation is rounded to a fixed number of bits. Quantization can also be done in more advanced ways than this: We have already mentioned that the MP3 standard may use different number of bits for values in the different subbands, depending on the importance of the samples for the human perception. The JPEG2000 standard quantizes in such a way that there is bigger interval around 0 which is quantized to 0, i.e. the rounding error is allowed to be bigger in an interval around 0. Standards which are lossless do not apply quantization, since this always leads to loss.

Somewhere in the image processing or sound processing pipeline, we also need a step which actually achieves compression of the data. Different standards use different lossless coding techniques for this. JPEG2000 uses an advances type of arithmetic coding for this. JPEG can also use arithmetic coding, but also Huffman coding.

Besides transformation, quantization, and coding, many other steps are used, which have different tasks. Many standards preprocess the pixel values before a transform is applied. Preprocessing may mean to center the pixel values around a certain value (JPEG2000 does this), or extracting the different image components before they are processed separately. Also, the image is often split into smaller parts (often called *tiles*), which are processed separately. For big images this is very important, since it allows users to zoom in on a small part of the image, without processing larger uninteresting parts of the image. Independent processing of the separate tiles makes the image compression what we call *error-resilient*, to errors such as transmission errors, since errors in one tile does not propagate to errors in the other tiles. It is also much more memory-friendly to process the image in several smaller parts, since it is not required to have the entire image in memory at any time. It also gives possibilities for parallel computing. For standards such as JPEG and JPEG2000, 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.

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, where the image was taken (such as GPS coordinates), and similar information. Metadata can also tell us how the color in the image are represented. As we have already seen, in most color images the color 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 color information in cases where each of the three color components needs 8 bits, it is common to create a table of up to 256 colors with which a given image could be represented quite well. Instead of storing the 24 bits, one then just stores a color 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 color*, and the table is called a *look-up* table or *palette*. For large photographs, however, 256 colors 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.

What you should have learned in this chapter.

- How to read, write, and show images on your computer.
- How to extract different color components.
- How to convert from color to grey-level images.
- How to use functions for adjusting the contrast.

- The operation $X \to S_1 X (S_2)^T$ can be used to define operations on images, based on one-dimensional operations S_1 and S_2. This amounts to applying S_1 to all columns in the image, and then S_2 to all rows in the result. You should know how this operation can be conveniently expressed with tensor product notation, and that in the typical case when S_1 and S_2 are filters, this can equivalently be expressed in terms of computational molecules.
- The case when the S_i are smoothing filters gives rise to smoothing operations on images.
- A simple highpass filter, corresponding to taking the derivative, gives rise to edge-detection operations on images.
- The operation $X \to S_1 X (S_2)^T$ can also be used to facilitate change of coordinates in images, in addition to filtering images. In other words, change of coordinates is done first column by column, then row by row. The DCT and the DFT are particular changes of coordinates used for images.

Chapter 10

Using tensor products to apply wavelets to images

Previously we have used the theory of wavelets to analyze sound. We would also like to use wavelets in a similar way to analyze images. Since the tensor product concept constructs two dimensional objects (matrices) from one-dimensional objects (vectors), we are lead to believe that tensor products can also be used to apply wavelets to images. In this chapter we will see that this can indeed be done. The vector spaces we V_m encountered for wavelets were function spaces, however. What we therefore need first is to establish a general definition of tensor products of function spaces. This will be done in the first section of this chapter. In the second section we will then specialize the function spaces to the spaces V_m we use for wavelets, and interpret the tensor product of these and the wavelet transform applied to images more carefully. Finally we will look at some examples on this theory applied to some example images.

The examples in this chapter can be run from the notebook `applinalgnbchap10.ipynb`.

10.1 Tensor product of function spaces

In the setting of functions, it will turn out that the tensor product of two univariate functions can be most intiutively defined as a function in two variables. This seems somewhat different from the strategy of Chapter 9, but we will see that the results we obtain will be very similar.

Definition 10.1. *Tensor product of function spaces.*

Let U_1 and U_2 be 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 $f_1 \otimes f_2$, is the function in two variables defined on $[0, M) \times [0, N)$ by

$$(f_1 \otimes f_2)(t_1, t_2) = f_1(t_1) f_2(t_2).$$

$f_1 \otimes f_2$ is also called the separable extension of f_1 and f_2 to two variables. The tensor product of the spaces $U_1 \otimes U_2$ is the vector space spanned by the two-variable functions $\{f_1 \otimes f_2\}_{f_1 \in U_1, f_2 \in U_2}$.

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,

$$\langle f, g \rangle = \int_0^N \int_0^M f(t_1, t_2) g(t_1, t_2) dt_1 dt_2. \tag{10.1}$$

In particular, this says that

$$\begin{aligned} \langle f_1 \otimes f_2, g_1 \otimes g_2 \rangle &= \int_0^N \int_0^M f_1(t_1) f_2(t_2) g_1(t_1) g_2(t_2) dt_1 dt_2 \\ &= \int_0^M f_1(t_1) g_1(t_1) dt_1 \int_0^N f_2(t_2) g_2(t_2) dt_2 = \langle f_1, g_1 \rangle \langle f_2, g_2 \rangle. \end{aligned} \tag{10.2}$$

This means that for tensor products, a double integral can be computed as the product of two one-dimensional integrals. This formula also ensures that inner products of tensor products of functions obey the same rule as we found for tensor products of vectors in Exercise 9.23.

The tensor product space defined in Definition 10.1 is useful for approximation of functions of two variables if each of the two spaces of univariate functions have good approximation properties.

Idea 10.2. *Using tensor products for approximation.*

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.

10.1.1 Tensor products of polynomials

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.

10.1.2 Tensor products of Fourier spaces

Let $U_1 = U_2 = V_{N,T}$ be the Nth order Fourier space which is spanned by the functions

$$e^{-2\pi i N t/T}, \ldots, e^{-2\pi i t/T}, 1, e^{2\pi i t/T}, \ldots, e^{2\pi i N t/T}$$

The tensor product space $U_1 \otimes U_1$ now consists of all functions on the form

$$\sum_{k,l=-N}^{N} \alpha_{k,l} e^{2\pi i k t_1/T} e^{2\pi i l t_2/T}.$$

One can show that this space has approximation properties similar to $V_{N,T}$ for functions in two variables. 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 9.16:

Theorem 10.3. *Bases for tensor products of function spaces.*

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 9.16: 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 (10.2). □

We can now define the tensor product of two bases of functions as before, and coordinate matrices as before:

Definition 10.4. *Coordinate matrix.*

if $\mathcal{B} = \{f_i\}_{i=0}^{M-1}$ and $\mathcal{C} = \{g_j\}_{j=0}^{N-1}$, we define $\mathcal{B} \otimes \mathcal{C}$ as the basis $\{f_i \otimes g_j\}_{(i,j)=(0,0)}^{(M-1,N-1)}$ for $U_1 \otimes U_2$. We say that X is the coordinate matrix of f if $f(t_1, t_2) = \sum_{i,j} X_{i,j}(f_i \otimes g_j)(t_1, t_2)$, where $X_{i,j}$ are the elements of X.

Theorem 9.18 can also be proved in the same way in the context of function spaces. We state this as follows:

Theorem 10.5. *Change of coordinates in tensor products of function spaces.*

Assume that U_1 and U_2 are function spaces, and that

- $\mathcal{B}_1, \mathcal{C}_1$ are bases for U_1, and that S_1 is the change of coordinates matrix from $\mathcal{B}_1$ to $\mathcal{C}_1$,
- $\mathcal{B}_2, \mathcal{C}_2$ are bases for U_2, and that S_2 is the change of coordinates matrix from $\mathcal{B}_2$ to $\mathcal{C}_2$.

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 $\mathcal{B}_1 \otimes \mathcal{B}_2$, Y the coordinate matrix in $\mathcal{C}_1 \otimes \mathcal{C}_2$, then the change of coordinates from $\mathcal{B}_1 \otimes \mathcal{B}_2$ to $\mathcal{C}_1 \otimes \mathcal{C}_2$ can be computed as

$$Y = S_1 X (S_2)^T. \tag{10.3}$$

10.2 Tensor product of function spaces in a wavelet setting

We will now specialize the spaces U_1, U_2 from Definition 10.1 to the resolution spaces V_m and the detail spaces W_m, arising 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

- 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),
- 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 $(\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1})$ also is a basis for V_m, we get the following corollary to Theorem 10.3:

Corollary 10.6. *Bases for tensor products.*

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

$$\begin{aligned}
&(\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1}) \otimes (\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1}) \\
&= \{\phi_{m-1,n_1} \otimes \phi_{m-1,n_2}, \\
&\qquad \phi_{m-1,n_1} \otimes \psi_{m-1,n_2}, \\
&\qquad \psi_{m-1,n_1} \otimes \phi_{m-1,n_2}, \\
&\qquad \psi_{m-1,n_1} \otimes \psi_{m-1,n_2}\}_{n_1,n_2}
\end{aligned}$$

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. These vector spaces deserve individual names:

Definition 10.7. *Tensor product spaces.*
We define the following tensor product spaces:

- The space $W_m^{(0,1)}$ spanned by $\{\phi_{m,n_1} \otimes \psi_{m,n_2}\}_{n_1,n_2}$,
- The space $W_m^{(1,0)}$ spanned by $\{\psi_{m,n_1} \otimes \phi_{m,n_2}\}_{n_1,n_2}$,
- The space $W_m^{(1,1)}$ spanned by $\{\psi_{m,n_1} \otimes \psi_{m,n_2}\}_{n_1,n_2}$.

Since these spaces are linearly independent, we can write

$$V_m \otimes V_m = (V_{m-1} \otimes V_{m-1}) \oplus W_{m-1}^{(0,1)} \oplus W_{m-1}^{(1,0)} \oplus W_{m-1}^{(1,1)}. \tag{10.4}$$

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, $W_{m-1}^{(0,1)}$, $W_{m-1}^{(1,0)}$, and $W_{m-1}^{(1,1)}$, are refered to as detail spaces. The coordinate matrix of

$$\sum_{n_1,n_2=0}^{2^{m-1}N} (c_{m-1,n_1,n_2}(\phi_{m-1,n_1} \otimes \phi_{m-1,n_2}) + w_{m-1,n_1,n_2}^{(0,1)}(\phi_{m-1,n_1} \otimes \psi_{m-1,n_2}) + w_{m-1,n_1,n_2}^{(1,0)}(\psi_{m-1,n_1} \otimes \phi_{m-1,n_2}) + w_{m-1,n_1,n_2}^{(1,1)}(\psi_{m-1,n_1} \otimes \psi_{m-1,n_2})) \tag{10.5}$$

in the basis $(\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1}) \otimes (\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1})$ is

$$\left(\begin{array}{ccc|ccc} c_{m-1,0,0} & \cdots & & w_{m-1,0,0}^{(0,1)} & \cdots & \\ \vdots & \vdots & & \vdots & \vdots & \\ \hline w_{m-1,0,0}^{(1,0)} & \cdots & & w_{m-1,0,0}^{(1,1)} & \cdots & \\ \vdots & \vdots & & \vdots & \vdots & \end{array}\right). \tag{10.6}$$

The coordinate matrix is thus 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 Equation (10.6).
- The $w_{m-1}^{(0,1)}$-values, i.e. the coordinates for $W_{m-1}^{(0,1)}$. This is the upper right corner in Equation (10.6).
- The $w_{m-1}^{(1,0)}$-values, i.e. the coordinates for $W_{m-1}^{(1,0)}$. This is the lower left corner in Equation (10.6).

- The $w_{m-1}^{(1,1)}$-values, i.e. the coordinates for $W_{m-1}^{(1,1)}$. This is the lower right corner in Equation (10.6).

The $w_{m-1}^{(i,j)}$-values are as in the one-dimensional situation often refered to as wavelet coefficients. Let us consider the Haar wavelet as an example.

Example 10.1: Piecewise constant functions

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 (10.4) 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

$$\langle \phi_{1,k_1} \otimes \phi_{1,k_2}, \phi_{0,n_1} \otimes \phi_{0,n_2} \rangle = \langle \phi_{1,k_1}, \phi_{0,n_1} \rangle \langle \phi_{1,k_2}, \phi_{0,n_2} \rangle = \frac{\sqrt{2}}{2}\frac{\sqrt{2}}{2} = \frac{1}{2}$$

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

$$
\begin{aligned}
&\phi_{1,k_1} \otimes \phi_{1,k_2} - \mathrm{proj}_{V_0 \otimes V_0}(\phi_{1,k_1} \otimes \phi_{1,k_2}) \\
&= \phi_{1,k_1} \otimes \phi_{1,k_2} - \frac{1}{2}(\phi_{0,k_1/2} \otimes \phi_{0,k_2/2}) \\
&= \left(\frac{1}{\sqrt{2}}(\phi_{0,k_1/2} + \psi_{0,k_1/2})\right) \otimes \left(\frac{1}{\sqrt{2}}(\phi_{0,k_2/2} + \psi_{0,k_2/2})\right) - \frac{1}{2}(\phi_{0,k_1/2} \otimes \phi_{0,k_2/2}) \\
&= \frac{1}{2}(\phi_{0,k_1/2} \otimes \phi_{0,k_2/2}) + \frac{1}{2}(\phi_{0,k_1/2} \otimes \psi_{0,k_2/2}) \\
&+ \frac{1}{2}(\psi_{0,k_1/2} \otimes \phi_{0,k_2/2}) + \frac{1}{2}(\psi_{0,k_1/2} \otimes \psi_{0,k_2/2}) - \frac{1}{2}(\phi_{0,k_1/2} \otimes \phi_{0,k_2/2}) \\
&= \frac{1}{2}(\phi_{0,k_1/2} \otimes \psi_{0,k_2/2}) + \frac{1}{2}(\psi_{0,k_1/2} \otimes \phi_{0,k_2/2}) + \frac{1}{2}(\psi_{0,k_1/2} \otimes \psi_{0,k_2/2}).
\end{aligned}
$$

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 10.1, together with the function $\phi \otimes \phi \in V_0 \otimes V_0$.

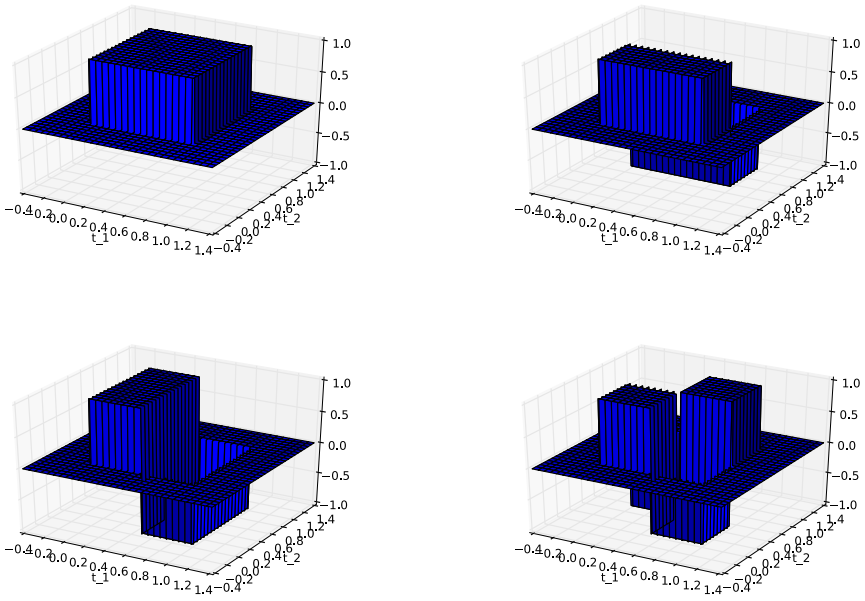

Figure 10.1: The functions $\phi \otimes \phi$, $\phi \otimes \psi$, $\psi \otimes \phi$, $\psi \otimes \psi$, which are bases for $(V_0 \otimes V_0) \oplus W_0^{(0,1)} \oplus W_0^{(1,0)} \oplus W_0^{(1,1)}$ for the Haar wavelet.

Example 10.2: Piecewise linear functions

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 10.2 for the alternative piecewise linear wavelet.

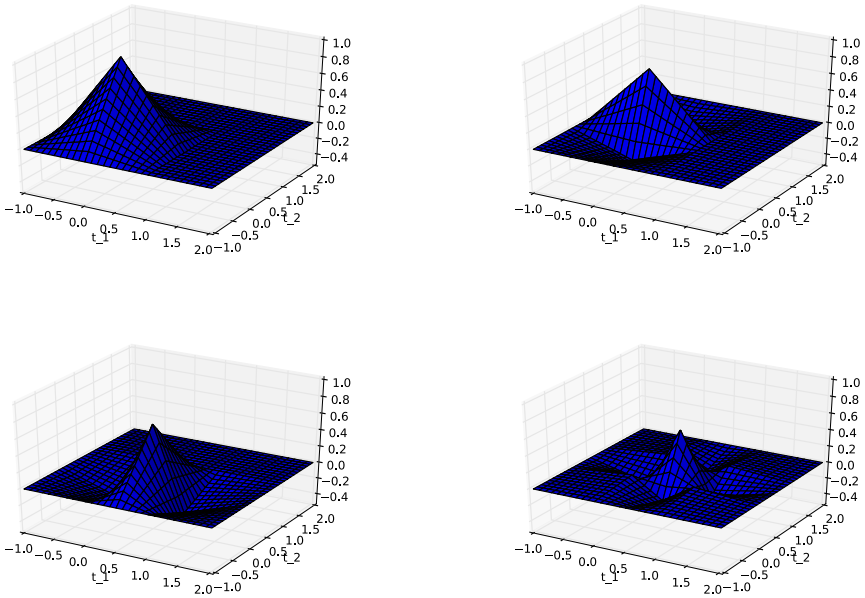

Figure 10.2: The functions $\phi \otimes \phi$, $\phi \otimes \psi$, $\psi \otimes \phi$, $\psi \otimes \psi$, which are bases for $(V_0 \otimes V_0) \oplus W_0^{(0,1)} \oplus W_0^{(1,0)} \oplus W_0^{(1,1)}$ for the alternative piecewise linear wavelet.

10.2.1 Interpretation

An immediate corollary of Theorem 10.5 is the following:

Corollary 10.8. *Implementing tensor product.*
Let

$$A_m = P_{(\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1}) \leftarrow \boldsymbol{\phi}_m}$$
$$B_m = P_{\boldsymbol{\phi}_m \leftarrow (\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1})}$$

be the stages in the DWT and the IDWT, and let

$$X = (c_{m,i,j})_{i,j} \qquad Y = \begin{pmatrix} (c_{m-1,i,j})_{i,j} & (w^{(0,1)}_{m-1,i,j})_{i,j} \\ (w^{(1,0)}_{m-1,i,j})_{i,j} & (w^{(1,1)}_{m-1,i,j})_{i,j} \end{pmatrix} \tag{10.7}$$

be the coordinate matrices in $\boldsymbol{\phi}_m \otimes \boldsymbol{\phi}_m$, and $(\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1}) \otimes (\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1})$, respectively. Then

$$Y = A_m X A_m^T \tag{10.8}$$

$$X = B_m Y B_m^T \tag{10.9}$$

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.

It is straightforward to make implementations of DWT2 and IDWT2, in the same way we implemented `DWTImpl` and `IDWTImpl`. In Exercise 10.7 you will be asked to program functions `DW2TImpl` and `IDW2TImpl` for this. Each stage in DWT2 and IDWT2 can now be implemented by substituting the matrices A_m, B_m above into the code following Theorem 9.18. 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 10.3, where the different types of coordinates which appear in the first two stages in a DWT2 are indicated.

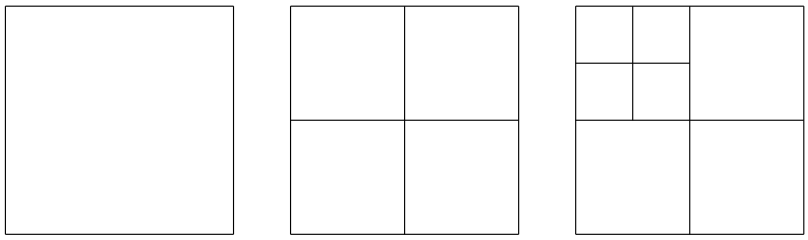

Figure 10.3: Illustration of the different coordinates in a two level DWT2 before the first stage is performed (left), after the first stage (middle), and after the second stage (right).

It is instructive to see what information the different types of coordinates in an image represent. In the following examples we will discard some types of coordinates, and view the resulting image. Discarding a type of coordinates will be illustrated by coloring the corresponding regions from Figure 10.3 black. As an example, 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 10.4 illustrates first the collection of all coordinates, and then the resulting collection of coordinates after removing subbands at the first level successively.

Figure 10.5 illustrates in the same way incremental removal of the subbands at the second level.

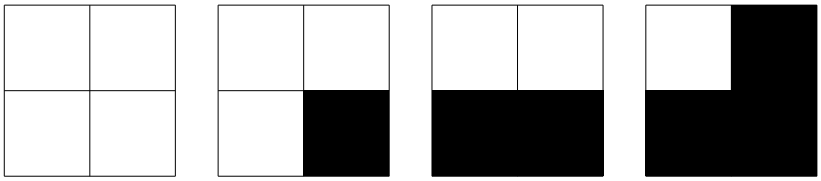

Figure 10.4: Graphical representation of neglecting the wavelet coefficients at the first level. After applying DWT2, the wavelet coefficients are split into four parts, as shown in the left figure. In the following figures we have removed coefficients from $W_1^{(1,1)}$, $W_1^{(1,0)}$, and $W_1^{(0,1)}$, in that order.

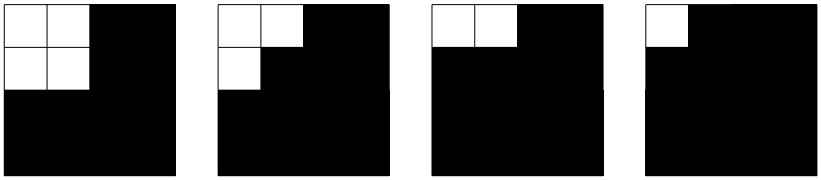

Figure 10.5: Graphical representation of neglecting the wavelet coefficients at the second level. After applying the second stage in DWT2, the wavelet coefficients from the upper left corner are also split into four parts, as shown in the left figure. In the following figures we have removed coefficients from $W_2^{(1,1)}$, $W_2^{(1,0)}$, and $W_2^{(0,1)}$, in that order.

Before we turn to experiments on images using wavelets, we would like to make another interpretation on the corners in the matrices after the DWT2, which correspond to the different coordinates $(c_{m-1,i,j})_{i,j}$, $(w^{(0,1)})_{m-1,i,j}$, $(w^{(1,0)})_{m-1,i,j}$, and $(w^{(1,1)})_{m-1,i,j}$. It turns out that these corners have natural interpretations in terms of the filter characterization of wavelets, as given in Chapter 6. Recall again that in a DWT2, the DWT is first applied to the columns in the image, then to the rows in the image. Recall first that the DWT2 applies first the DWT to all columns, and then to all rows in the resulting matrix.

First the DWT is applied to all columns in the image. Since the first half of the coordinates in a DWT are outputs from a lowpass filter H_0 (Theorem 6.3), the upper half after the DWT has now been subject to a lowpass filter to the columns. Similarly, the second half of the coordinates in a DWT are outputs from a highpass filter H_1 (Theorem 6.3 again), so that the bottom half after the DWT has been subject to a highpass filter to the columns.

Then the DWT is applied to all rows in the image. Similarly as when we applied the DWT to the columns, the left half after the DWT has been subject

to the same lowpass filter to the rows, and the right half after the DWT has been subject to the same highpass filter to the rows.

These observations split the resulting matrix after DWT2 into four blocks, with each block corresponding to a combination of lowpass and highpass filters. The following names are thus given to these blocks:

- 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 type of filters which have been applied (L=lowpass, H=highpass). The first letter indicates the type of 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. In conclusion, a block in the matrix after the DWT2 corresponds to applying a combination of lowpass/higpass filters to the rows of the columns of the image. Due to this, and since lowpass filters extract slow variations, highpass filters abrupt changes, the following holds:

Observation 10.9. *Visual interpretation of the DWT2.*

After the DWT2 has been applied to an image, we expect to see the following:

- In the upper left corner, slow variations in both the vertical and horizontal directions are captured, i.e. this is a low-resolution version of the image.
- In the upper right corner, slow variations in the vertical direction are captured, together with abrupt changes in the horizontal direction.
- In the lower left corner, slow variations in the horizontal direction are captured, together with abrupt changes in the vertical direction.
- In the lower right corner, abrupt changes in both directions appear are captured.

These effects will be studied through examples in the next section.

10.3 Experiments with images using wavelets

In this section we will make some experiments with images using the wavelets we have considered. The wavelet theory is applied to images in the following way: We first visualize the pixels in the image as coordinates in the basis $\boldsymbol{\phi}_m \otimes \boldsymbol{\phi}_m$ (so that the image has size $(2^m M) \times (2^m N)$). As in the case for sound, this will represent a good approximation wehn m is large. We then perform a change of coordinates with the DWT2. As we did for sound, we can then either set

the detail components from the $W_k^{(i,j)}$-spaces to zero, or the low-resolution approximation from $V_0 \otimes V_0$ to zero, depending on whether we want to inspect the detail components or the low-resolution approximation. Finally we apply the IDWT2 to end up with coordinates in $\boldsymbol{\phi}_m \otimes \boldsymbol{\phi}_m$ again, and display the new image with pixel values equal to these coordinates.

Example 10.3: Applying the Haar wavelet to a very simple example image

Let us apply the Haar wavelet to the sample chess pattern example image from Figure 9.17. 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. Clearly, abrupt changes in the vertical and horizontal directions appear here only at the edges in the chess pattern, and abrupt changes in both directions appear only at the grid points in the chess pattern. Due to Observation 10.9, after a DWT2 we expect to see the following:

- In the upper left corner, we should see a low-resolution version of the image.
- In the upper right corner, only the vertical edges in the chess pattern should be visible.
- In the lower left corner, only the horizontal edges in the chess pattern should be visible.
- In the lower right corner, only the grid points in the chess pattern should be visible.

These effects are seen clearly if we apply one level of the DWT2 to the chess pattern example image. The result of this can be seen in Figure 10.6.

Example 10.4: Creating thumbnail images

Let us apply the Haar wavelet to our sample image. After the DWT2, the upper left 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 $m = 1$:

```
DWT2Impl(X, 1, 'Haar')
X = X[0:(shape(X)[0]/2), 0:(shape(X)[1]/2)]
mapto01(X); X *= 255
```

Note that here it is necessary to map the result back to $[0, 255]$.

In Figure 10.7 the results are shown up to 4 resolutions. In Figure 10.8 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.

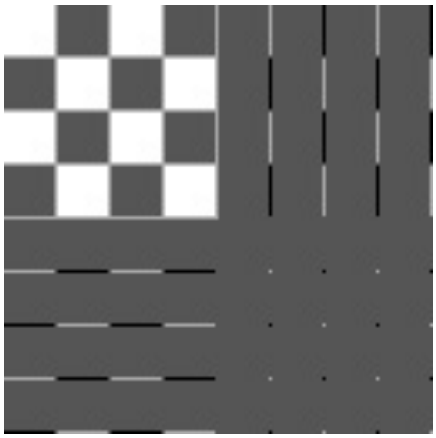

Figure 10.6: The chess pattern example image after application of the DWT2. The Haar wavelet was used.

Figure 10.7: The corresponding thumbnail images for the Image of Lena, obtained with a DWT of 1, 2, 3, and 4 levels.

Example 10.5: Detail and low-resolution approximations for different wavelets

Let us take a closer look at the images generated when we use different wavelets. Above we viewed the low-resolution approximation as a smaller image. Let us compare with the image resulting from setting 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 10.4 and Figure 10.5. We should expect that the lower order resolution approximations from V_0 are worse when m increase.

Figure 10.9 confirms this for the lower order resolution approximations for the Haar wavelet. Alternatively, we should see that the higher order detail spaces contain more information as m increases. The result is shown in Figure 10.10.

Figures 10.11 and 10.12 confirms the same for the CDF 9/7 wavelet, which also shows some improvement in the low resolution approximations. The black

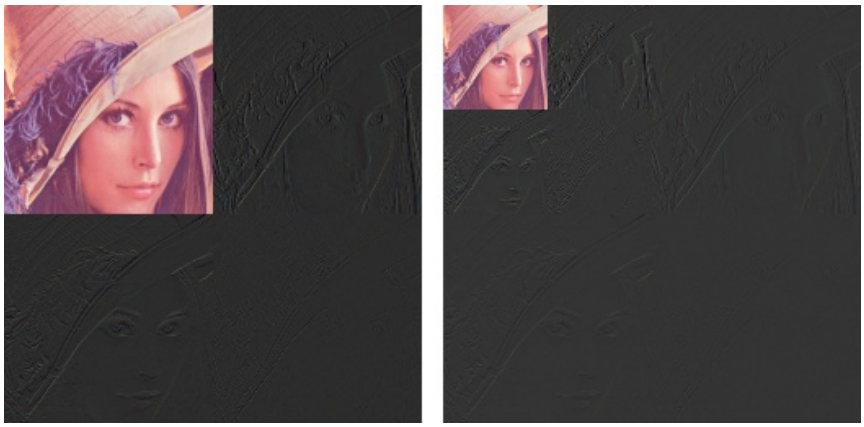

Figure 10.8: The corresponding image resulting from a wavelet transform with the Haar-wavelet for $m = 1$ and $m = 2$.

color indicates values which are close to 0. In other words, most of the coefficients are close to 0.

Example 10.6: The Spline 5/3 wavelet and removing bands in the detail spaces

Since the detail components in images are split into three bands, another thing we can try is to neglect only parts of the detail components (i.e.e some of $W_m^{(1,1)}$, $W_m^{(1,0)}$, $W_m^{(0,1)}$), 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 10.4 are removed are shown in Figure 10.13. The corresponding plot for the second level is shown in Figure 10.14.

The image is seen still to resemble the 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 10.15 we have also shown the resulting image after the third and fourth levels 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 10.16, we have shown the corresponding detail for DWT's with 1, 2, 3, and 4 levels. Clearly, more detail can be seen in the image when more of the detail is included.

Figure 10.9: Low resolution approximations of the Lena image, for the Haar wavelet.

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 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

Figure 10.10: Detail of the Lena image, for the Haar wavelet.

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`.

Exercise 10.7: Implement two-dimensional DWT

Implement functions

```
DWT2Impl_internal(x, nres, f, bd_mode)
IDWT2Impl_internal(x, nres, f, bd_mode)
```

which perform the m-level DWT2 and the IDWT2, respectively, on an image. The arguments are the as those in `DWTImpl_internal` and `IDWTImpl_internal`, with the input vector `x` replaced with a two-dimensional object/image. The

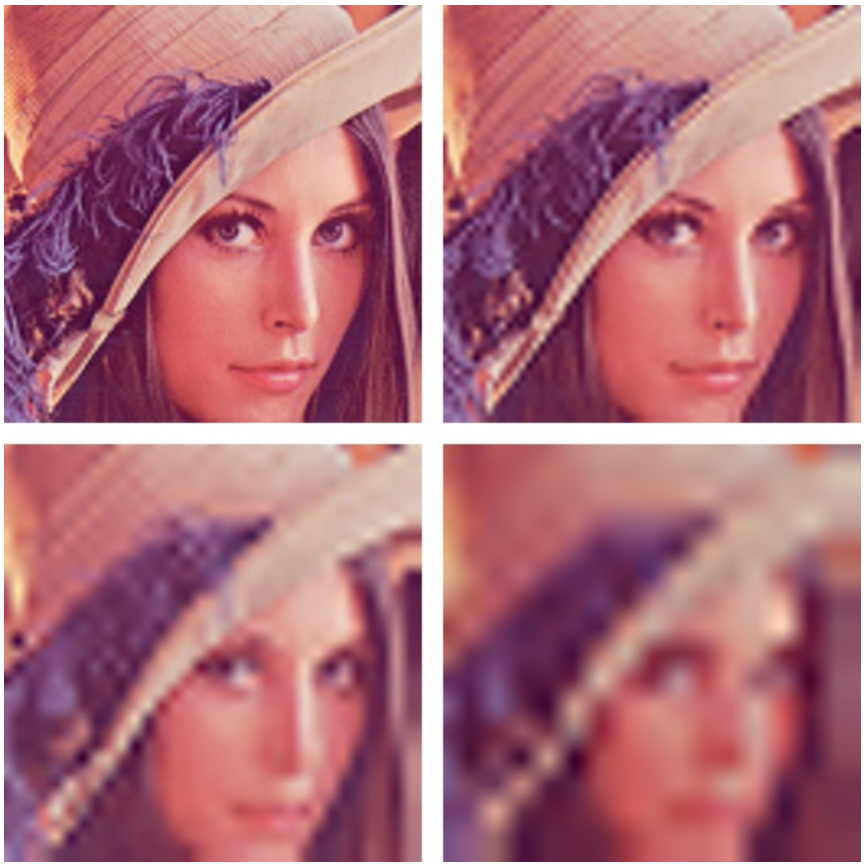

Figure 10.11: Low resolution approximations of the Lena image, for the CDF 9/7 wavelet.

functions should at each stage apply the kernel function `f` to the appropriate rows and columns. If the image has several color components, the functions should be applied to each color component (there are three color components in the test image 'lena.png').

Exercise 10.8: Comment code

Assume that we have an image represented by the $M \times N$-matrix `X`, and consider the following code:

```
for n in range(N):
    c = (X[0:M:2, n] + X[1:M:2, n])/sqrt(2)
    w = (X[0:M:2, n] - X[1:M:2, n])/sqrt(2)
```

Figure 10.12: Detail of the Lena image, for the CDF 9/7 wavelet.

```
    X[:, n] = concatenate([c, w])

for m in range(M):
    c = (X[m, 0:N:2] + X[m, 1:N:2])/sqrt(2)
    w = (X[m, 0:N:2] - X[m, 1:N:2])/sqrt(2)
    X[m, :] = concatenate([c,w])
```

a) Comment what the code does, and explain what you will see if you display `X` as an image after the code has run.

b) The code above has an inverse transformation, which reproduce the original image from the transformed values which we obtained. Assume that you zero out the values in the lower left and the upper right corner of the matrix `X` after the code above has run, and that you then reproduce the image by applying this inverse transformation. What changes can you then expect in the image?

Figure 10.13: Image of Lena, with various bands of detail at the first level zeroed out. From left to right, the detail at $W_1^{(1,1)}$, $W_1^{(1,0)}$, $W_1^{(0,1)}$, as illustrated in Figure 10.4. The Spline 5/3 wavelet was used.

Figure 10.14: Image of Lena, with various bands of detail at the second level zeroed out. From left to right, the detail at $W_2^{(1,1)}$, $W_2^{(1,0)}$, $W_2^{(0,1)}$, as illustrated in Figure 10.5. The Spline 5/3 wavelet was used.

Exercise 10.9: Comment code

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`:

```
M, N = shape(X)
for n in range(N):
    c = X[0:M:2, n] + X[1:M:2, n]
    w = X[0:M:2, n] - X[1:M:2, n]
    X[:, n] = vstack([c,w])

for m in range(M):
```

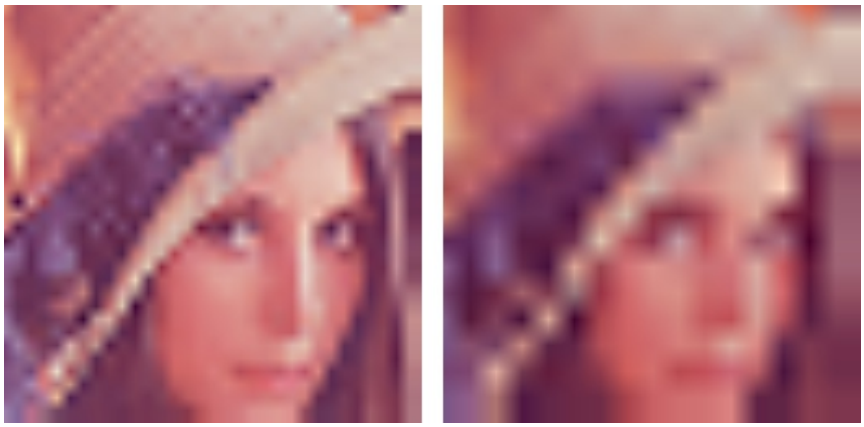

Figure 10.15: Image of Lena, with detail including level 3 and 4 zeroed out. The Spline 5/3 wavelet was used.

```
c = X[m, 0:N:2] + X[m, 1:N:2]
w = X[m, 0:N:2] - X[m, 1:N:2]
X[m, :] = hstack([c,w])
```

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 $(G_1 \otimes G_1)X$?

Exercise 10.10: Experiments on a test image

In Figure 10.17 we have applied the DWT2 with the Haar wavelet to an image very similar to the one you see in Figure 10.6. You see here, however, that there seems to be no detail components, which is very different from what you saw in Example 10.3, even though the images are very similar. Attempt to explain what causes this to happen.

Hint. Compare with Exercise 5.21.

Figure 10.16: The corresponding detail for the image of Lena. The Spline 5/3 wavelet was used.

Figure 10.17: A simple image before and after one level of the DWT2. The Haar wavelet was used.

10.4 An application to the FBI standard for compression of fingerprint images

In the beginning of the 1990s, the FBI 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 9.4. 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).

Figure 10.18: A typical fingerprint image.

Fingerprint images are a very specific type of images, as seen in Figure 10.18. 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 regularity (as many 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 10.3) 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. We will split this process into five stages. The subband structures and the resulting images after stage 1-4 are illustrated in Figure 10.19 and in Figure 10.20, respectively.

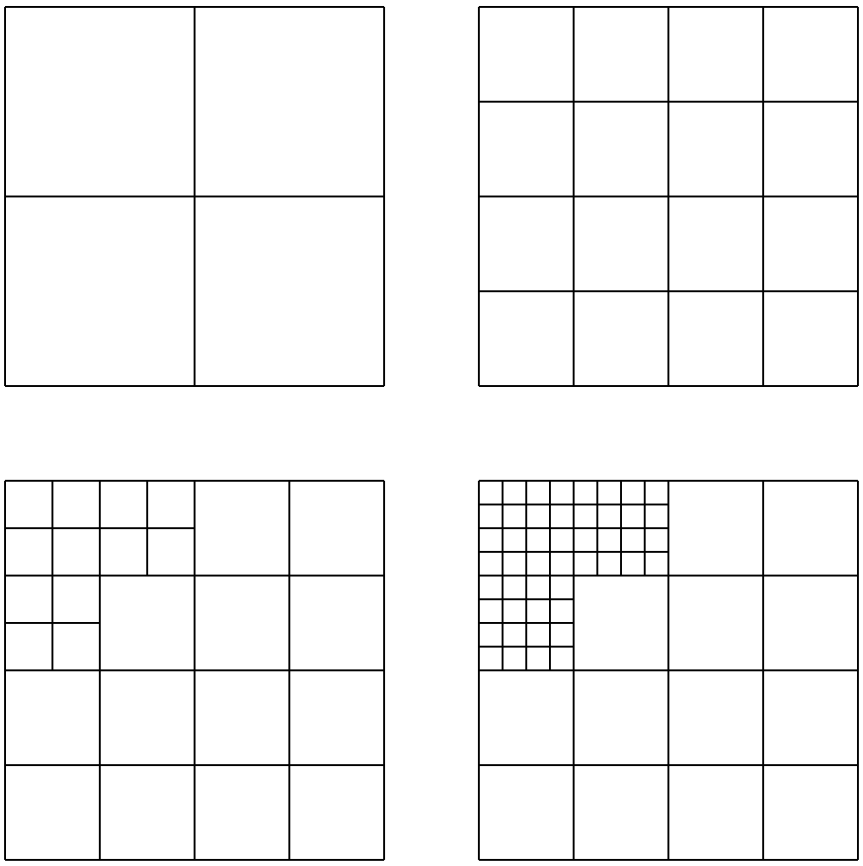

Figure 10.19: Subband structure after the different stages of the wavelet applications in the FBI fingerprint compression scheme.

1. First apply the first stage in a DWT2. This gives the upper left corners in the two figures.
2. Then apply a DWT2 to all four resulting subbands. This is different from the DWT2, which only continues on the upper left corner. This gives the upper right corners in the two figures.

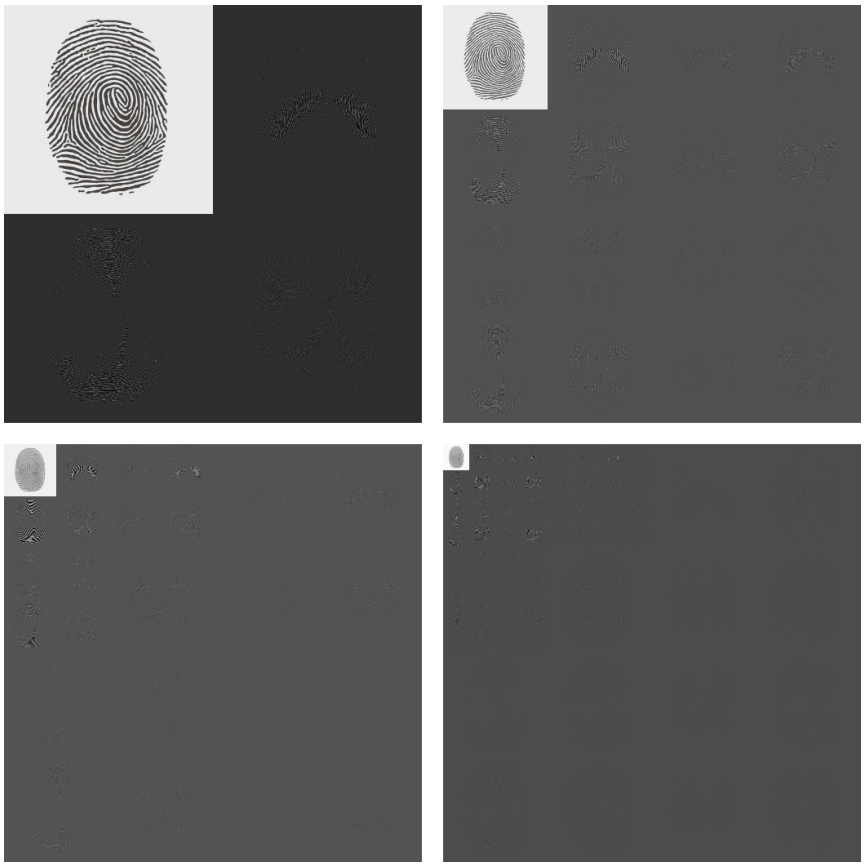

Figure 10.20: The fingerprint image after several DWT's.

3. Then apply a DWT2 in three of the four resulting subbands. This gives the lower left corners.

4. In all remaining subbands, the DWT2 is again applied. This gives the lower right corners.

Now for the last stage. A DWT2 is again applied, but this time only to the upper left corner. The subbands are illustrated in Figure 10.21, and in Figure 10.22 the resulting image is shown.

When establishing the standard for compression of fingerprint images, the FBI chose this subband decomposition. In Figure 10.23 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.

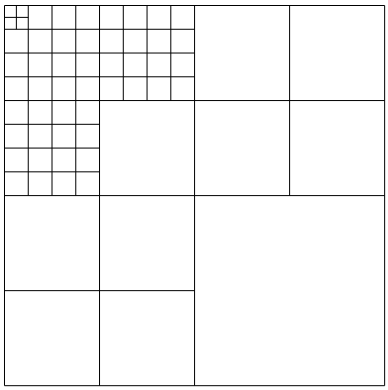

Figure 10.21: Subbands structure after all stages.

Figure 10.22: The resulting image obtained with the subband decomposition employed by the FBI.

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

Figure 10.23: The low-resolution approximation and the detail obtained by the FBI standard for compression of fingerprint images, when applied to our sample fingerprint image.

need to decompress all images in the database to perform matching. We will not go into details on this here.

Exercise 10.11: Implement the fingerprint compression scheme

Write code which generates the images shown in figures 10.20, 10.22, and 10.23. Use the functions `DW2TImpl` and `IDW2TImpl` with the CDF 9/7 wavelet kernel functions as input.

10.5 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 [21]. In [46], 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 [15]. In [4], the theory is described. The book [16] uses the application to compression of fingerprint images as an example of the usefulness of recent developments in wavelet theory.

What you should have learned in this chapter.

- The special interpretation of DWT2 applied to an image as splitting into four types of coordinates (each being one corner of the image), which represent lowpass/highpass combinations in the horizontal/vertical directions.
- How to call functions which perform different wavelet transformations on an image.
- Be able to interpret the detail components and low-resolution approximations in what you see.