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

Example 7.1. *The alternative piecewise linear wavelet.*

Let us return to the alternative piecewise linear wavelet. In Example 6.19 we found the filters H_0, H_1 for this wavelet, and these determine the dual scaling function and the dual mother wavelet. We already know how the scaling function

and the mother wavelet look, but how do the dual functions look? It turns out that there is usually no way to find analytical expressions for these dual functions (as is the case for the scaling function and the mother wavelet itself in most cases), but that there still is an algorithm we can apply in order to see how these functions look. This algorithm is called the *cascade algorithm*, and works essentially by computing the coordinates of ϕ, ψ (or $\tilde{\phi}, \tilde{\psi}$) in $\boldsymbol{\phi}_m$ (or $\tilde{\boldsymbol{\phi}}_m$). By increasing m, we have previously argued that these coordinates are good approximations to the samples of the functions.

To be more specific, we start with the following observation for the dual functions (similar observations hold for the scaling function and the mother wavelet also):

- the coordinates of $\tilde{\phi}$ in $(\tilde{\boldsymbol{\phi}}_0, \tilde{\boldsymbol{\psi}}_0, \tilde{\boldsymbol{\psi}}_1 \ldots)$ is the vector with 1 first, followed by only zeros,
- the coordinates of $\tilde{\psi}$ in $(\tilde{\boldsymbol{\phi}}_0, \tilde{\boldsymbol{\psi}}_0, \tilde{\boldsymbol{\psi}}_1 \ldots)$ is the vector with N zeros first, then a 1, and then only zeros.

The length of these vectors is $N2^m$. The coordinates in $\tilde{\boldsymbol{\phi}}_m$ for $\tilde{\phi}$ and $\tilde{\psi}$ can be obtained by applying the m-level IDWT for the dual wavelet (i.e. the filters $(H_0)^T$, $(H_1)^T$ are used) to these vectors. In Exercise 7.1 we will study code which uses this approach to approximate the scaling function and mother wavelet. In Figure 7.1 we have plotted the resulting coordinates in $\tilde{\boldsymbol{\phi}}_{10}$, and thus a good approximation to $\tilde{\phi}$ and $\tilde{\psi}$. We see that these functions look very irregular. Also, they are very different from the original scaling function and mother wavelet. We will later argue that this is bad, it would be much better if $\phi \approx \tilde{\phi}$ and $\psi \approx \tilde{\psi}$.

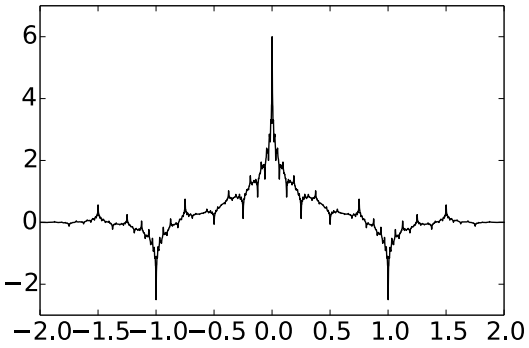

Figure 7.1: Dual scaling function $\tilde{\phi}$ (left) and dual mother wavelet $\tilde{\psi}$ (right) for the alternative piecewise linear wavelet.

From Theorem 6.11 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.

Let us formalize the cascade algorithm from the previous example as follows.

Definition 7.2. *The cascade algorithm.*

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

7.2 Turning things around: How to construct useful wavelet bases from filters

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 7.3. 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 7.4. *$g_N(\nu)$ 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 7.5. *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 7.6. *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 7.7. *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

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

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 7.7 is long, technical, and split in many stages. The entire proof can be found in [5], 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 [5] 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 7.7 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 5.40, 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 [4] 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.

7.2.1 Sketch of proof for the biorthogonality in Proposition 7.7 (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

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

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

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

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).

7.2.2 Sketch of proof for the biorthogonality of in Proposition 7.7 (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, $\langle \tilde{\phi}_{m',k}, \psi_{m,l} \rangle = \langle \tilde{\psi}_{m',k}, \psi_{m,l} \rangle = 0$.

7.2.3 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 [5] 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 [5] 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

Let us consider the following code, which shows how the cascade algorithm can be used to plot the scaling functions and the mother wavelet of a wavelet and its dual wavelet with given kernels, over the interval $[a, b]$.

```
def plotwaveletfunctions(invf, a, b):
    """
    Plot the scaling functions and mother wavelets of a wavelet
    and its dual wavelet using the cascade algorithm.

    invf: the IDWT kernel
    a: the left point of the plot interval.
    b: the right point of the plot interval.
    """
    nres = 10
    t = linspace(a, b, (b-a)*2**nres)

    coordsvm = zeros((b-a)*2**nres)
    coordsvm[0] = 1
    IDWTImpl(coordsvm, nres, invf, 0, 0)
    coordsvm *= 2**(nres/2)
    plt.subplot(2, 2, 1)
    coordsvm = concatenate([coordsvm[(b*2**nres):((b-a)*2**nres)], \
                            coordsvm[0:(b*2**nres)]])
    plt.plot(t, coordsvm)
    plt.title('$\phi$')
```

```
coordsvm = zeros((b-a)*2**nres)
coordsvm[b - a] = 1
IDWTImpl(coordsvm, nres, invf, 0, 0)
coordsvm *= 2**(nres/2)
plt.subplot(2, 2, 2)
coordsvm = concatenate([coordsvm[(b*2**nres):((b-a)*2**nres)], \
                        coordsvm[0:(b*2**nres)]])
plt.plot(t, coordsvm)
plt.title('$\psi$')

coordsvm = zeros((b-a)*2**nres)
coordsvm[0] = 1
IDWTImpl(coordsvm, nres, invf, 0, 1)
coordsvm *= 2**(nres/2)
plt.subplot(2, 2, 3)
coordsvm = concatenate([coordsvm[(b*2**nres):((b-a)*2**nres)], \
                        coordsvm[0:(b*2**nres)]])
plt.plot(t, coordsvm)
plt.title('Dual $\phi$')

coordsvm = zeros((b-a)*2**nres)
coordsvm[b - a] = 1
IDWTImpl(coordsvm, nres, invf, 0, 1)
coordsvm *= 2**(nres/2)
plt.subplot(2, 2, 4)
coordsvm = concatenate([coordsvm[(b*2**nres):((b-a)*2**nres)], \
                        coordsvm[0:(b*2**nres)]])
plt.plot(t, coordsvm)
plt.title('Dual $\psi$')
plt.show()
```

a) Run the function `plotwaveletfunctions` with the three different kernels `IDWTKernelHaar`, `IDWTKernelpwl0`, and 'IDWTKernelpwl2' to plot all scaling functions and mother wavelets for the Haar wavelet and the two piecewise linear wavelets we have encountered. This should verify the different plots for these we have seen previously in the book.

b) Explain that the input to `IDWTImpl` in the code above are the coordinates of $\phi_{0,0}$, $\psi_{0,0}$, $\tilde{\phi}_{0,0}$, and $\tilde{\psi}_{0,0}$ in the basis $(\boldsymbol{\phi}_0, \boldsymbol{\psi}_0, \boldsymbol{\psi}_1, \boldsymbol{\psi}_2, \cdots, \boldsymbol{\psi}_{m-1})$, respectively.

c) In the code above, we wanted the functions to be plotted on $[a, b]$. Explain from this why the `coordsvm`-vector have been rearranged as on the line where the `plot`-command is called.

d) 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?

e) In the code you see that all values are scaled with the factor $2^{m/2}$ before they are plotted. Can you think out an explanation to why this is done?

Exercise 7.2: Using the cascade algorithm

In Exercise 6.10 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).

Exercise 7.3: Implementing the traspose transforms

Since the dual of a wavelet is constructed by transposing filters, one may suspect that taking the dual is the same as taking the transpose. However, show that the DWT, the dual DWT, the transpose of the DWT, and the transpose of the dual DWT, can be computed as follows:

```
DWTImpl(x,  m, DWTkernel, 1, 0) # DWT
DWTImpl(x,  m, DWTkernel, 1, 1) # Dual DWT
IDWTImpl(x, m, IDWTkernel, 1, 1) # Transpose of the DWT
IDWTImpl(x, m, IDWTkernel, 1, 0) # Transpose of the dual DWT
```

Similar statements hold for the IDWT as well.

7.3 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 7.8. *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 7.11 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 7.9. *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 7.10. *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 [5] for them. Let us also express what we have found in terms of analog filters.

Observation 7.11. *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 1.39, 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.

7.4 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.

7.4.1 Symmetric filters

The main result when the filters are symmetric looks as follows.

Theorem 7.12. *Wavelet crirteria.*
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 $Q = Q_1 Q_2$, Equation (6.20) can now be rewritten as

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

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 7.4.3, 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.

7.4.2 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 (6.18) and (6.19) (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 7.13. *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 7.12. 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 (6.20) 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 7.12 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 7.7 we will take a closer look at wavelets constructed in this way.

7.4.3 The proof of Bezouts theorem

Theorem 7.14. *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 7.4: Compute filters

Compute the filters H_0, G_0 in Theorem 7.12 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).

7.5 A design strategy suitable for lossless compression

We choose $Q_1 = Q$, $Q_2 = 1$. In this case there is no need to find factors in Q. The 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

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

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

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

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

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

Due to this $\left(\frac{\sin(\pi f)}{\pi f}\right)^{N_2}$ is the frequency response of $*_{k=1}^{N_2}\chi_{[-1/2,1/2)}(t)$. By the uniqueness of the frequency response we have that $\phi(t) = \hat{\phi}(0)*_{k=1}^{N_2}\chi_{[-1/2,1/2)}(t)$. In Exercise 7.6 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 7.15. *Resolution spaces of piecewise polynomials.*

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

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

Example 7.16. *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 6.19. Now, how do we find the filters (G_1, H_1)? Previously we saw how to find the constant α in Theorem 6.17 when we knew one of the two pairs (G_0, G_1), (H_0, H_1). This was the last part of information we needed in order to construct the other two filters. Here we know (G_0, H_0) instead. In this case it is even easier to find (G_1, H_1) since we can set $\alpha = 1$. This means that (G_1, H_1) can be obtained simply by adding alternating signs to (G_0, H_0), i.e. they are the corresponding highpass filters. We thus can set

$$G_1 = \left\{-\frac{1}{4}, -\frac{1}{2}, \underline{\frac{3}{2}}, -\frac{1}{2}, -\frac{1}{4}\right\} \qquad H_1 = \left\{-\frac{1}{4}, \underline{\frac{1}{2}}, -\frac{1}{4}\right\}.$$

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 7.16 is used for lossless compression in the JPEG2000 standard.

Exercise 7.5: 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) Write a function which takes N_1 and N_2 as input, computes the filter coefficients of H_0 and G_0 using equation (7.29) in the compendium, and plots the frequency responses of G_0 and H_0. Recall that the frequency response can be obtained from the filter coefficients by taking a DFT. You will have use for the `conv` function here, and that the frequency response $(1+\cos\omega)/2$ corresponds to the filter with coefficients $\{1/4, \underline{1/2}, 1/4\}$.

b) Recall that in Exercise 6.12 we implemented DWT and IDWT kernels, which worked for any set of symmetric filters. Combine these kernels with your computation of the filter coefficients from a), and use the function `plotwaveletfunctions` to plot the corresponding scaling functions and mother wavelets for different N_1 and N_2.

Exercise 7.6: 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.

7.6 A design strategy suitable for lossy compression

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

Example 7.17. *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

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

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 7.2.

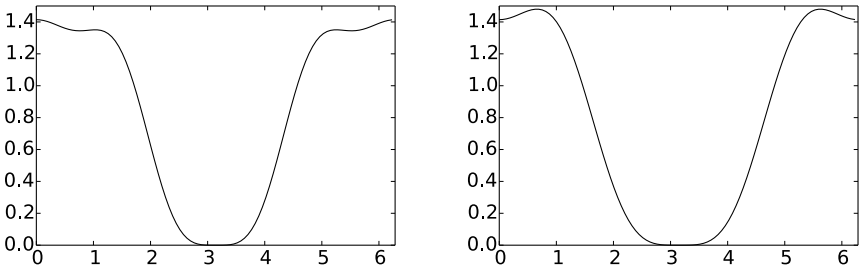

Figure 7.2: The frequency responses $\lambda_{H_0}(\omega)$ (left) and $\lambda_{G_0}(\omega)$ (right) for the CDF 9/7 wavelet.

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

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

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

In Example 6.19 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, as we did in Example 7.1 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 6.11 that $\text{supp}(\phi) = [-3, 3]$, $\text{supp}(\psi) = [-3, 4]$, $\text{supp}(\tilde{\phi}) = [-4, 4]$, and $\text{supp}(\tilde{\psi}) = [-3, 4]$. Plotting the scaling functions and mother wavelets over these supports using the cascade algorithm gives the plots in Figure 7.3. Again they have irregular shapes, but now at least the functions and dual functions more resemble each other.

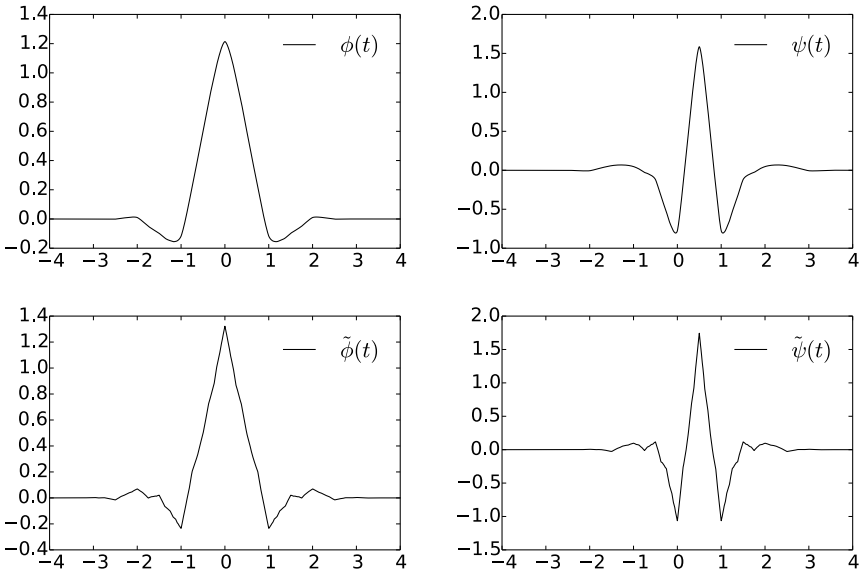

Figure 7.3: Scaling functions and mother wavelets for the CDF 9/7 wavelet.

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).

Exercise 7.7: Generate plots

Generate the plots from Figure 7.3 using the cascade algorithm. Reuse the code from Exercise 7.1 in order to achieve this.

7.7 Orthonormal wavelets

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

In Theorem 7.13 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:

$$H_0 = \{t_{-L}, \dots, t_{-1}, \underline{t_0}, t_1, \dots, t_{L-1}\} \quad H_1 = \{s_{-L+1}, \dots, s_{-1}, \underline{s_0}, s_1, \dots, s_L\}. \tag{7.30}$$

To see why, Theorem 6.17 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

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

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

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

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

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

Frequency responses are shown in Figure 7.4 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.

Clearly these filters have lowpass characteristic. We also see that the high-pass characteristics resemble the lowpass 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 3.12). 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 6.11 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

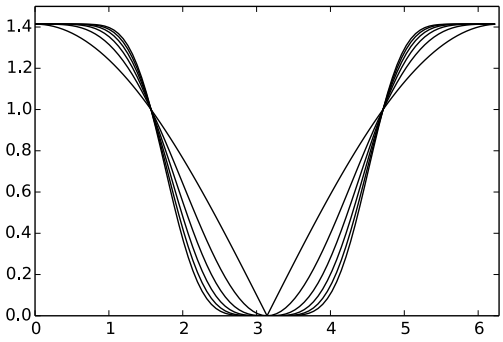

Figure 7.4: The magnitudes $|\lambda_{G_0}(\omega)| = |\lambda_{H_0}(\omega)|$ for the first orthonormal wavelets.

- for $N = 2$: $\text{supp}(\phi) = [-1, 2]$, $\text{supp}(\psi) = [-3/2, 3/2]$,
- for $N = 3$: $\text{supp}(\phi) = [-2, 3]$, $\text{supp}(\psi) = [-5/2, 5/2]$,
- for $N = 4$: $\text{supp}(\phi) = [-3, 4]$, $\text{supp}(\psi) = [-7/2, 7/2]$.

The scaling functions and mother wavelets are shown in Figure 7.5. 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.

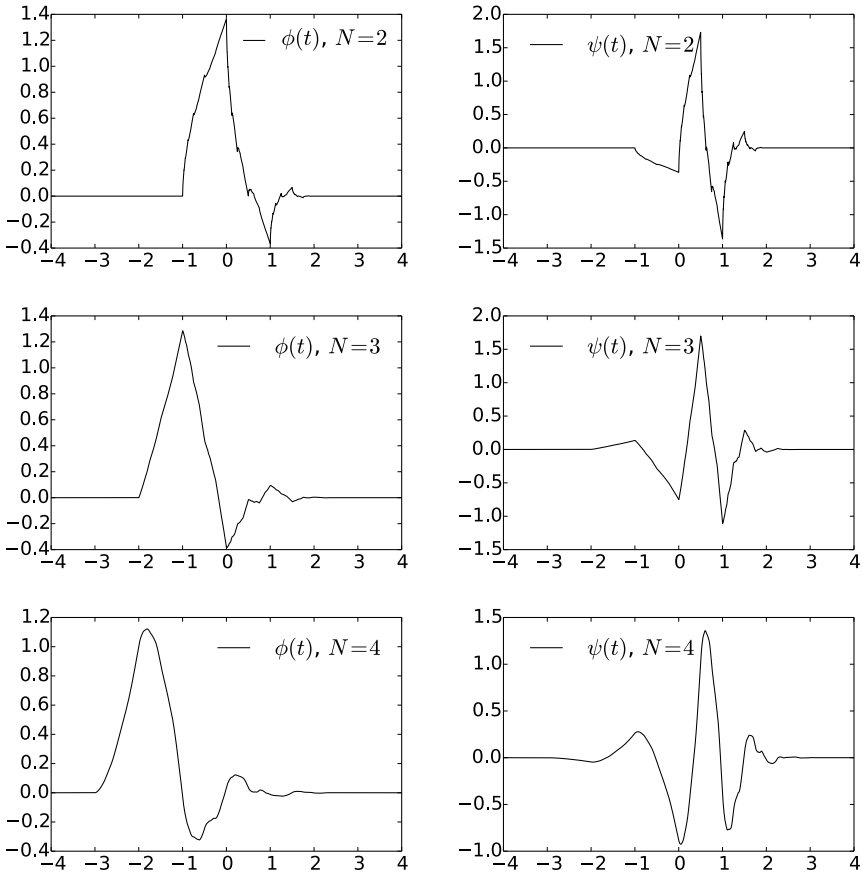

Figure 7.5: The scaling functions and mother wavelets for orthonormal wavelets with N vanishing moments, for different values of N.

7.8 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 7.1, 7.3, 7.5). 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 [8]. Biorthogonal wavelets were first constructed in [5].

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 6.17.

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.

Example 8.2. *A 6×6 MRA-matrix.*

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 8.3. *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 8.4. *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 6.17. We need to prove the following:

Theorem 8.5. *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 8.3 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 8.6. *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 8.7. *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 8.4. 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 8.8. *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 8.7.

Proof. The proof will use the concept of the length of a filter, as defined in Definition 3.5. 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 3.5. 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 $\Lambda_{n+1} = \begin{pmatrix} I\&I \\ 0\&I \end{pmatrix}$, 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 6.17 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 8.8 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 8.7 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 8.8. 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 8.7).

Assume that we in the proof of Theorem 8.8 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 8.9. *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. The corresponding MRA matrices are

$$\begin{pmatrix} 1 & \lambda & 0 & 0 & \cdots & 0 & 0 & \lambda \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \lambda & 1 & \lambda & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \lambda & 1 & \lambda \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix} \text{ and } \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \lambda & 1 & \lambda & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ \lambda & 0 & 0 & 0 & \cdots & 0 & \lambda & 1 \end{pmatrix},$$

respectively, or

$$\begin{pmatrix} 1 & 2\lambda & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \lambda & 1 & \lambda & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \lambda & 1 & \lambda \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix} \text{ and } \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \lambda & 1 & \lambda & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 2\lambda & 1 \end{pmatrix} \tag{8.13}$$

if we use symmetric extensions as defined by Definition 5.42 (we have used Theorem 5.43). 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 with the lifting factorization

The number of arithmetic operations needed to apply matrices on the form stated in Equation (8.13) is easily computed. The number of multiplications is $N/2$ if symmetry is exploited as in Observation 4.21 (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 8.10. *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 6.17, we defered the last part, namely that equations (7.29) in the compendium-(8.3) in the compendium 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 6.20 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 6.21 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

$$\mathcal{D}_m = \{\phi_{m,0}, \phi_{m,2}, \phi_{m,4} \ldots, \phi_{m,1}, \phi_{m,3}, \phi_{m,5}, \ldots\}, \tag{8.14}$$

$P_{\mathcal{D}_m \leftarrow \boldsymbol{\phi}_m}$ 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 8.11. *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 $P_{\mathcal{D}_m \leftarrow (\boldsymbol{\phi}_1, \boldsymbol{\psi}_1)}$.

Example 8.12. *Lifting factorization of the piecewise linear wavelet.*

Let us consider the piecewise linear wavelet from Section 5.4, for which we found that the change of coordinate matrix G was given by Equation (6.1). In the four different polyphase components of G, let us underline the corresponding elements:

$$\frac{1}{\sqrt{2}} \begin{pmatrix} \underline{1} & 0 \\ 1/2 & 1 \\ \underline{0} & 0 \\ \vdots & \vdots \\ \underline{0} & 0 \\ 1/2 & 0 \end{pmatrix}, \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & \underline{0} \\ 1/2 & 1 \\ 0 & \underline{0} \\ \vdots & \vdots \\ 0 & \underline{0} \\ 1/2 & 0 \end{pmatrix}, \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 \\ \underline{1/2} & 1 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ \underline{1/2} & 0 \end{pmatrix}, \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 \\ 1/2 & \underline{1} \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 1/2 & \underline{0} \end{pmatrix}. \tag{8.15}$$

we get the following:

- The upper left polyphase component $G^{(0,0)}$ is $\frac{1}{\sqrt{2}}I$.
- The upper right polyphase component $G^{(0,1)}$ is $\mathbf{0}$.
- The lower left polyphase component $G^{(1,0)}$ is $\frac{1}{\sqrt{2}}S_1$, where S_1 is the filter $\{1/2, \underline{1/2}\}$.
- The lower right polyphase component $G^{(1,1)}$ is $\frac{1}{\sqrt{2}}I$.

In other words, 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 8.7, 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

$$\sqrt{2}\begin{pmatrix} I & \mathbf{0} \\ -\frac{1}{2}\{1,\underline{1}\} & I \end{pmatrix} \text{ and } \frac{1}{\sqrt{2}}\begin{pmatrix} I & \mathbf{0} \\ \frac{1}{2}\{1,\underline{1}\} & I \end{pmatrix}, \tag{8.16}$$

respectively.

Example 8.13. *Lifting factorization of the alternative piecewise linear wavelet.*

Let us now consider the alternative piecewise linear wavelet. In this case, Equation (6.3) shows that $P_{\mathcal{D}_1 \leftarrow (\boldsymbol{\phi}_1, \hat{\boldsymbol{\psi}}_1)}$ (the polyphase representation of H) is not on the form

$$\begin{pmatrix} I & \mathbf{0} \\ S_1 & I \end{pmatrix}$$

for some filter S_1, since there is more than one element in every column. Recall, however, that the alternative piecewise linear wavelet was obtained by constructing a new mother wavelet $\hat{\psi}$ from the old ψ. $\hat{\psi}$ is defined in Section 5.5 by Equation (5.38), which said that

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

From this equation it is clear that

$$P_{(\boldsymbol{\phi}_1, \boldsymbol{\psi}_1) \leftarrow (\boldsymbol{\phi}_1, \hat{\boldsymbol{\psi}}_1)} = \begin{pmatrix} I & S_2 \\ 0 & I \end{pmatrix},$$

where $S_2 = -\frac{1}{4}\{\underline{1}, 1\}$. We can now write the polyphase representation of G as

$$P_{\mathcal{D}_1 \leftarrow (\boldsymbol{\phi}_1, \hat{\boldsymbol{\psi}}_1)} = P_{\mathcal{D}_1 \leftarrow (\boldsymbol{\phi}_1, \boldsymbol{\psi}_1)} P_{(\boldsymbol{\phi}_1, \boldsymbol{\psi}_1) \leftarrow (\boldsymbol{\phi}_1, \hat{\boldsymbol{\psi}}_1)}. = \frac{1}{\sqrt{2}}\begin{pmatrix} I & \mathbf{0} \\ \frac{1}{2}\{1,\underline{1}\} & I \end{pmatrix}\begin{pmatrix} I & -\frac{1}{4}\{\underline{1},1\} \\ \mathbf{0} & I \end{pmatrix}.$$

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

$$\sqrt{2}\begin{pmatrix} I & \frac{1}{4}\{\underline{1},1\} \\ \mathbf{0} & I \end{pmatrix}\begin{pmatrix} I & \mathbf{0} \\ -\frac{1}{2}\{1,\underline{1}\} & I \end{pmatrix} \text{ and } \frac{1}{\sqrt{2}}\begin{pmatrix} I & \mathbf{0} \\ \frac{1}{2}\{1,\underline{1}\} & I \end{pmatrix}\begin{pmatrix} I & -\frac{1}{4}\{\underline{1},1\} \\ \mathbf{0} & I \end{pmatrix}, \tag{8.17}$$

respectively.

Example 8.14. *Lifting factorization of the Spline 5/3 wavelet.*

Let us consider the Spline 5/3 wavelet, which we defined in Example 7.16. 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.

Example 8.15. *Lifting factorization of the CDF 9/7 wavelet.*

For the wavelet we considered in Example 7.17, 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 8.8. 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 8.9. 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 7.7 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.

Example 8.16. *Lifting of 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 7.7 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

$$\begin{pmatrix} \{t_{-k},\dots,t_{-1},\underline{t_0},t_1,\dots,t_k\} \\ \{s_{-k},\dots,s_{-1},s_0,s_1,\dots,s_{k-1}\} \end{pmatrix}.$$

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

$$\begin{pmatrix} \{t_{-k},\dots,t_{-1},\underline{t_0},t_1,\dots,t_k\} \\ \{s_{-k-1},\dots,s_{-1},s_0,s_1,\dots,s_k\} \end{pmatrix}.$$

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

$$\begin{pmatrix} 1 & \lambda_1 & 0 & 0 & \cdots & 0 & 0 & \lambda_2 \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \lambda_2 & 1 & \lambda_1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \lambda_2 & 1 & \lambda_1 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix} \text{ and } \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \lambda_2 & 1 & \lambda_1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ \lambda_1 & 0 & 0 & 0 & \cdots & 0 & \lambda_2 & 1 \end{pmatrix}, \tag{8.18}$$

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 (8.18). 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 sets global variables `lambdas`, `alpha`, and `beta`, so that the factorization (8.8) holds. `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 of `lambdas`. In the exercises, you will be asked to implement both these nonsymmetric elementary lifting steps, as well as kernel transformations for orthonormal wavelets, which assume that these global variables have been set, and describe the lifting steps of the wavelet (Exercise 8.10).

Exercise 8.7: Polyphase components for symetric 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 8.8: Implement elementary lifting steps

Write functions `liftingstepevensymm` and `liftingstepoddsymm` which take λ, a vector $\boldsymbol{x}$, and `symm` as input, and apply the elementary lifting matrices as in Equation (8.13) in the compendium, respectively, to $\boldsymbol{x}$. The parameter `symm` should indicate whether symmetric extensions shall be applied. Your code should handle both when N is odd, and when N is even (as noted previously, when symmetric extensions are not applied, we assume that N is even). The function should not perform matrix multiplication, and apply as few multiplications as possible.

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 8.8 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 (8.13) in the compendium-(8.12) in the compendium here, in order to complete the kernels for bot the transformations and the dual transformations.

a) Write the DWT and IDWT kernel transformations for the piecewise linear wavelet. Your functions should use the lifting factorizations in (8.16) in the compendium. Call your functions `DWTKernelpwl0` and `IDWTKernelpwl0`.

b) Write the DWT and IDWT kernel transformations for the alternative piecewise linear wavelet. The lifting factorizations are now given by (8.17) in the compendium instead. Call your functions `DWTKernelpwl2` and `IDWTKernelpwl2`.

c) Write the DWT and IDWT kernel transformations for the Spline 5/3 wavelet, using the lifting factorization obtained in Example 8.14. Call your functions `DWTKernel53` and `IDWTKernel53`.

d) Write the DWT and IDWT kernel transformations for the CDF 9/7 wavelet, using the lifting factorization obtained in Example 8.15. Call your functions `DWTKernel97` and `IDWTKernel97`.

e) In Chapter 5, we listened to the low-resolution approximations and detail components in sound for three different wavelets, using the function `playDWT`. Repeat these experiments with the Spline 5/3 and the CDF 9/7 wavelet, using the new kernels we have implemented in this exercise.

f) Use the function `plotwaveletfunctions` from Exercise 7.1 to plot all scaling functions and mother wavelets for the Spline 5/3 and the CDF 9/7 wavelets, using 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 `liftingstepeven` and `liftingstepodd` which take λ_1, λ_2 and a vector $\boldsymbol{x}$ as input, and apply the elementary lifting matrices (8.18) in the compendium, respectively, to $\boldsymbol{x}$. Assume that N is even.

b) Write functions `DWTKernelOrtho` and `IDWTKernelOrtho` which take a vector $\boldsymbol{x}$ as input, and apply the DWT and IDWT kernel transformations for orthonormal wavelets to $\boldsymbol{x}$. You should call the functions `liftingstepeven` and `liftingstepodd`. As mentioned, assume that global variables `lambdas`, `alpha`, and `beta` have been set, so that the lifting factorization (8.8) in the compendium holds, where `lambdas` is a $n \times 2$-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 of `lambdas`. Recall that the last lifting step was even.

c) Listen to the low-resolution approximations and detail components in sound for orthonormal wavelets for $N = 1, 2, 3, 4$, again using the function `playDWT`. You need to call the function `liftingfactortho` in order to set the kernel for the different values of N.

d) Use the function `plotwaveletfunctions` from Exercise 7.1 to plot all scaling functions and mother wavelets for orthonormal wavelets for $N = 1, 2, 3, 4$. 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 5.31 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

$$\frac{1}{\sqrt{2}}\begin{pmatrix} I & \mathbf{0} \\ \{1/2, \underline{1/2}\} & I \end{pmatrix}\begin{pmatrix} I & \{-\gamma, \underline{-\alpha}, -\beta, -\delta\} \\ \mathbf{0} & I \end{pmatrix}. \tag{8.19}$$

You here need to reconstruct what you did in the lifting factorization for the alternative piecewise linear wavelet, i.e. write

$$P_{\mathcal{D}_1 \leftarrow (\boldsymbol{\phi}_1, \hat{\boldsymbol{\psi}}_1)} = P_{\mathcal{D}_1 \leftarrow (\boldsymbol{\phi}_1, \boldsymbol{\psi}_1)} P_{(\boldsymbol{\phi}_1, \boldsymbol{\psi}_1) \leftarrow (\boldsymbol{\phi}_1, \hat{\boldsymbol{\psi}}_1)}.$$

By inversion, find also a lifting factorization of H.

Exercise 8.12: Wavelet based on piecewise quadratic scaling function

In Exercise 7.4 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

$$\begin{pmatrix} I & -\frac{1}{128}\{5, \underline{-29}, -29, 5\} \\ \mathbf{0} & I \end{pmatrix}\begin{pmatrix} I & \mathbf{0} \\ -\{1, \underline{1}\} & I \end{pmatrix}\begin{pmatrix} I & -\frac{1}{4}\{\underline{1}, 1\} \\ \mathbf{0} & I \end{pmatrix} G = \begin{pmatrix} \frac{1}{4} & \mathbf{0} \\ \mathbf{0} & 4 \end{pmatrix}.$$

From this we can easily derive the lifting factorization of G.

b) Implement the kernels of the wavelet of this exercise using what you did in Exercise 6.12.

c) Listen to the low-resolution approximations and detail components in sound for this wavelet.

d) Use the function `plotwaveletfunctions` from Exercise 7.1 to plot all scaling functions and mother wavelets for this wavelet.

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 of the MP3 standard

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

$$z_{32(s-1)+n} = \sum_{k=0}^{511} \cos((n+1/2)(k-16)\pi/32) h_k x_{32s-k-1}, \tag{8.20}$$

which lead to the expression of the forward filter bank transform (Theorem 6.26). 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 (6.31). If we write

$$V^{(m)} = \{\underline{(-1)^0 h_m}, 0, (-1)^1 h_{m+64}, 0, (-1)^2 h_{m+128}, \ldots, (-1)^7 h_{m+7\cdot 64}, 0\}, \tag{8.21}$$

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

$$\begin{aligned}
\boldsymbol{z} = &\begin{pmatrix}
\cos\left(2\pi(0+1/2)\cdot(-16)/64\right) I & \cdots & \cos\left(2\pi(0+1/2)\cdot(47)/64\right) I \\
\vdots & \ddots & \vdots \\
\cos\left(2\pi(31+1/2)\cdot(-16)/64\right) I & \cdots & \cos\left(2\pi(31+1/2)\cdot(47)/64\right) I
\end{pmatrix} \\
&\times \begin{pmatrix}
V^{(0)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & V^{(1)} & \cdots & \mathbf{0} & \mathbf{0} \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
\mathbf{0} & \mathbf{0} & \cdots & V^{(62)} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & V^{(63)}
\end{pmatrix}
\begin{pmatrix}
\boldsymbol{x}^{(31)} \\ \boldsymbol{x}^{(30)} \\ \vdots \\ \boldsymbol{x}^{(-32)}
\end{pmatrix}.
\end{aligned}$$

If we place the 15 first columns in the cosine matrix last using Property (6.31) (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 (6.32) 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

$$4(D_{32})^T \begin{pmatrix} \mathbf{0} & \cdots & \mathbf{0} & \sqrt{2}V^{(16)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \cdots & V^{(17)} & \mathbf{0} & V^{(15)} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ V^{(31)} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & V^{(1)} & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & V^{(0)} + E_1 V^{(32)} \\ -E_1 V^{(63)} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & E_1 V^{(33)} & \mathbf{0} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \cdots & -E_1 V^{(49)} & \mathbf{0} & E_1 V^{(47)} & \cdots & \mathbf{0} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}^{(0)} \\ \vdots \\ \boldsymbol{x}^{(31)} \end{pmatrix}.$$

We have therefore proved the following result.

Theorem 8.17. *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

$$4(D_{32})^T \begin{pmatrix} \mathbf{0} & \cdots & \mathbf{0} & \sqrt{2}V^{(16)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \cdots & V^{(17)} & \mathbf{0} & V^{(15)} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ V^{(31)} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & V^{(1)} & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & V^{(0)} + E_1 V^{(32)} \\ -E_1 V^{(63)} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & E_1 V^{(33)} & \mathbf{0} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \cdots & -E_1 V^{(49)} & \mathbf{0} & E_1 V^{(47)} & \cdots & \mathbf{0} & \mathbf{0} \end{pmatrix} \tag{8.22}$$

Due to Theorem 6.28, 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

$$4\begin{pmatrix} \mathbf{0} & \mathbf{0} & \cdots & (W^{(31)})^T & \mathbf{0} & -E_{-1}(W^{(63)})^T & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \mathbf{0} & (W^{(17)})^T & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & -E_{-1}(W^{(49)})^T \\ \sqrt{2}(W^{(16)})^T & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & (W^{(15)})^T & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & E_{-1}(W^{(47)})^T \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & (W^{(1)})^T & \mathbf{0} & E_{-1}(W^{(33)})^T & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & (W^{(0)})^T + E_{-1}(W^{(32)})^T & \mathbf{0} & \cdots & \mathbf{0} \end{pmatrix} \times D_{32}E_{481}. \tag{8.23}$$

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 (8.23) we get the following result:

Theorem 8.18. *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

$$4\begin{pmatrix} \mathbf{0} & \mathbf{0} & \cdots & -U^{(33)} & \mathbf{0} & E_{-1}U^{(1)} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \mathbf{0} & -U^{(47)} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & E_{-1}U^{(15)} \\ -\sqrt{2}U^{(48)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & -U^{(49)} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & -E_{-1}U^{(17)} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & -U^{(63)} & \mathbf{0} & -E_{-1}U^{(31)} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & E_{-2}U^{(0)} - E_{-1}U^{(32)} & \mathbf{0} & \cdots & \mathbf{0} \end{pmatrix} \times D_{32}E_{33}. \tag{8.24}$$

Now, consider the matrices

$$\begin{pmatrix} V^{(32-i)} & V^{(i)} \\ -E_1V^{(64-i)} & E_1V^{(32+i)} \end{pmatrix} \text{ and } \begin{pmatrix} -U^{(32+i)} & E_{-1}U^{(i)} \\ -U^{(64-i)} & -E_{-1}U^{(32-i)} \end{pmatrix}. \tag{8.25}$$

for $1 \le i \le 15$. These make out submatrices in the matrices in equations (8.22) and (8.24). Clearly, only the product of these matrices influence the result. Since

$$\begin{pmatrix} -U^{(32+i)} & E_{-1}U^{(i)} \\ -U^{(64-i)} & -E_{-1}U^{(32-i)} \end{pmatrix} \begin{pmatrix} V^{(32-i)} & V^{(i)} \\ -E_1V^{(64-i)} & E_1V^{(32+i)} \end{pmatrix}$$
$$= \begin{pmatrix} -U^{(32+i)} & U^{(i)} \\ -U^{(64-i)} & -U^{(32-i)} \end{pmatrix} \begin{pmatrix} V^{(32-i)} & V^{(i)} \\ -V^{(64-i)} & V^{(32+i)} \end{pmatrix} \tag{8.26}$$

we have the following result.

Theorem 8.19. *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

$$\begin{pmatrix} -U^{(32+i)} & U^{(i)} \\ -U^{(64-i)} & -U^{(32-i)} \end{pmatrix} \begin{pmatrix} V^{(32-i)} & V^{(i)} \\ -V^{(64-i)} & V^{(32+i)} \end{pmatrix} = c \begin{pmatrix} E_d & \mathbf{0} \\ \mathbf{0} & E_d \end{pmatrix}$$
$$(\sqrt{2}V^{(16)})(-\sqrt{2}U^{(48)}) = cE_d$$
$$(V^{(0)} + E_1V^{(32)})(E_{-2}U^{(0)} - E_{-1}U^{(32)}) = cE_d \tag{8.27}$$

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 (8.27). 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 chosen in the MP3 standard

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

$$V^{(64-k)} = -E_{14}(V^{(k)})^T, 1 \leq k \leq 15. \tag{8.28}$$
$$V^{(0)} = E_{16}(V^{(0)})^T. \tag{8.29}$$

Inserting first that $U^{(k)} = 32V^{(k)}$ in Equation (8.26) 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

$$\begin{aligned}&32 \begin{pmatrix} E_{14}(V^{(32-i)})^T & V^{(i)} \\ E_{14}(V^{(i)})^T & -V^{(32-i)} \end{pmatrix} \begin{pmatrix} V^{(32-i)} & V^{(i)} \\ E_{14}(V^{(i)})^T & -E_{14}(V^{(32-i)})^T \end{pmatrix} \\ =&32 \begin{pmatrix} E_{14} & \mathbf{0} \\ \mathbf{0} & E_{14} \end{pmatrix} \begin{pmatrix} (V^{(32-i)})^T & V^{(i)} \\ (V^{(i)})^T & -V^{(32-i)} \end{pmatrix} \begin{pmatrix} V^{(32-i)} & V^{(i)} \\ (V^{(i)})^T & -(V^{(32-i)})^T \end{pmatrix} \\ =&32 \begin{pmatrix} E_{14} & \mathbf{0} \\ \mathbf{0} & E_{14} \end{pmatrix} \begin{pmatrix} V^{(32-i)} & V^{(i)} \\ (V^{(i)})^T & -(V^{(32-i)})^T \end{pmatrix}^T \begin{pmatrix} V^{(32-i)} & V^{(i)} \\ (V^{(i)})^T & -(V^{(32-i)})^T \end{pmatrix} \\ =&32 \begin{pmatrix} E_{14} & \mathbf{0} \\ \mathbf{0} & E_{14} \end{pmatrix} \begin{pmatrix} V^{(i)}(V^{(i)})^T + V^{(32-i)}(V^{(32-i)})^T & \mathbf{0} \\ \mathbf{0} & V^{(i)}(V^{(i)})^T + V^{(32-i)}(V^{(32-i)})^T \end{pmatrix}.\end{aligned} \tag{8.30}$$

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

$$\begin{aligned}&(\sqrt{2}V^{(16)})(-\sqrt{2}U^{(48)}) \\ &= -64V^{(16)}V^{(48)} = 64E_{14}V^{(16)}(V^{(16)})^T = 32E_{14}(V^{(16)}(V^{(16)})^T + V^{(16)}(V^{(16)})^T)\end{aligned} \tag{8.31}$$

and

$$\begin{aligned}&(V^{(0)} + E_1V^{(32)})(E_{-2}U^{(0)} - E_{-1}U^{(32)}) \\ &= 32(V^{(0)} + E_1V^{(32)})(E_{-2}V^{(0)} - E_{-1}V^{(32)}) = 32E_{-2}(V^{(0)} + E_1V^{(32)})(V^{(0)} - E_1V^{(32)}) \\ &= 32E_{-2}(V^{(0)})^2 - (V^{(32)})^2) = 32E_{14}((V^{(0)}(V^{(0)})^T + V^{(32)}(V^{(32)})^T)).\end{aligned} \tag{8.32}$$

We see that the filters from equations (8.30)-(8.32) are similar, and that we thus can combine them into

$$\{V^{(i)}(V^{(i)})^T + V^{(32-i)}(V^{(32-i)})^T\}_{i=0}^{16}. \tag{8.33}$$

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 (8.33) 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 8.4, so that its frequency response is real, so that we have no phase distortion. We can thus summarize our findings as follows.

Observation 8.20. *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 (8.33) are equal, and so that their combination into a prototype filter using equations (8.21) and (8.28) 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 How can we obtain 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 (8.20) we replace with

$$z_{32(s-1)+n} = \sum_{k=0}^{511} \cos((n+1/2)(k+1/2-16)\pi/32)h_k x_{32s-k-1}, \tag{8.34}$$

i.e. $1/2$ is added inside the cosine. We now have the properties

$$\cos\left(2\pi(n+1/2)(k+64r+1/2)/(2N)\right) = (-1)^r \cos\left(2\pi(n+1/2)(k+1/2)/(2N)\right) \tag{8.35}$$

$$\cos\left(2\pi(n+1/2)(2N-k-1+1/2)/(2N)\right) = -\cos\left(2\pi(n+1/2)(k+1/2)/(2N)\right). \tag{8.36}$$

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 (8.36) 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 8.21. *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

$$4D_M^{(iv)} \begin{pmatrix} \mathbf{0} & \mathbf{0} & \cdots & V^{(16)} & V^{(15)} & \cdots & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & V^{(30)} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & V^{(1)} & \mathbf{0} \\ V^{(31)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & \cdots & V^{(0)} \\ -E_1 V_{(63)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & \cdots & E_1 V^{(32)} \\ \mathbf{0} & -E_1 V^{(62)} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & E_1 V^{(33)} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & -E_1 V^{(48)} & E_1 V^{(47)} & \cdots & \cdots & \mathbf{0} \end{pmatrix} \tag{8.37}$$

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 6.16 on the vector $(1, 2, 3, \ldots, 8192)$. Verify that there seems to be a delay on 481 elements, as promised by Therorem 8.20. 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 (8.21) in the compendium, so that the matrices on the form given by Equation (8.25) in the compendium, 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 (8.25) in the compendium, where the filters are on the from given by Equation (8.21) in the compendium, 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 (8.25) in the compendium 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 (8.21) in the compendium), 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 (8.21) in the compendium, 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 8.19 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 [17] 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 [35]. How to use lifting for in-place calculation for the DWT was also suggested by Sweldens [34].

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.

9.1.1 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.

9.1.2 Digital output media

Our focus will be on objects that emit light, for example a computer display. A computer monitor consists of a 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).

9.1.3 Digital input media

The two most common ways to acquire digital images is with a digital camera or a scanner. A scanner essentially takes a photo of a document in the form of a 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.

9.1.4 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 9.10 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.*

Figure 9.2: Black and white (left), and grey-level (right) versions of the image in Figure 9.1.

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 9.3: 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.

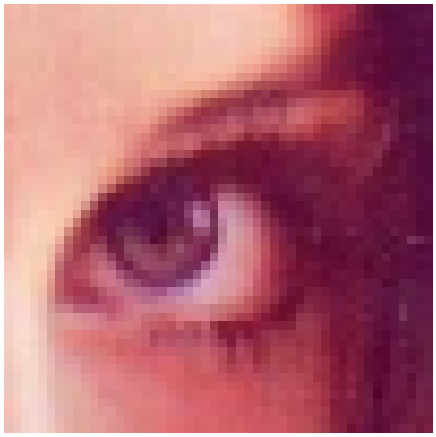

Figure 9.4: 50×50 pixels excerpt of the color image in Figure 9.1.

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.*

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

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 wil go through first).

In order to perform these operations, we need to be able to use images with a programming environment.

9.2.1 Images and Python

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 previuos sections. But what we also need before we go through operations on images, is, as in the sections on sound, means of reading an image from a file so that its contents are accessible as a matrix, and write images represented by a matrix which we have constructed ourself to file. Reading a function from file can be done with help of the function `imread`. If we write

```
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 9.8. *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 coulor components in each pixel, which are usually the red-, green-, and blue color components.

Example 9.9. *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 9.10. *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 $p = \max(r, g, b)$. The result of this can be seen in the left image of Figure 9.6.

An alternative is to use the sum of the three values as a measure of the total intensity at the point. This corresponds to setting $p = r + g + b$. Here we have to be a bit careful with a subtle point. We have required each of the r, g and b values to lie in the range $[0, 1]$, but their sum may of course become as large as 3. We also require our grey-level values to lie in the range $[0, 1]$ so after having computed all the sums we must normalise as explained above. The result can be seen in the middle images of Figure 9.6.

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. The result is shown in the right image of Figure 9.6.

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

$$q_{i,j} = \frac{\hat{q}_{i,j}}{\max_{k,l} \hat{q}_{k,l}}.$$

If `img` is an $M \times N$ image, this can be implemented by using most of the code from the previous example, and replacing with the lines

```
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
```

respectively. In practice one of the last two methods are usually preferred, perhaps with a preference for the last method, but the actual choice depends on the application.

Example 9.11. *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 9.6: Alternative ways to convert the color image in Figure 9.1 to a grey level image. The result is mapped to $(0, 1)$.

Figure 9.7: The negative versions of the corresponding images in Figure 9.6.

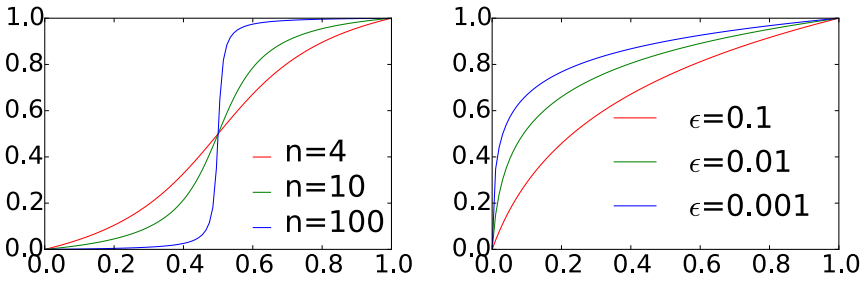

Figure 9.8: Some functions that can be used to improve the contrast of an image.

Example 9.12. *Increasing the contrast.*

A common problem with images is that the contrast often is not good enough. This typically means that a large proportion of the grey values are concentrated in a rather small subinterval of $[0, 1]$. The obvious solution to this problem is to somehow spread out the values. This can be accomplished by applying a function f to the intensity values, i.e., new intensity values are computed by the

Figure 9.9: The middle functions in Figure 9.8 have been applied to a grey-level version of the test image.

formula

$$\hat{p}_{i,j} = f(p_{i,j})$$

for all i and j. If we choose f so that its derivative is large in the area where many intensity values are concentrated, we obtain the desired effect.

Figure 9.8 shows some examples. The functions in the left plot have quite large derivatives near $x = 0.5$ and will therefore increase the contrast in images with a concentration of intensities with value around 0.5. The functions are all on the form

$$f_n(x) = \frac{\arctan\bigl(n(x - 1/2)\bigr)}{2\arctan(n/2)} + \frac{1}{2}. \tag{9.1}$$

For any $n \neq 0$ these functions satisfy the conditions $f_n(0) = 0$ and $f_n(1) = 1$. The three functions in the left plot in Figure 9.8 correspond to $n = 4$, 10, and 100. In the left plot in Figure 9.9 the middle function has been applied to the image in Figure 9.6(c). Since the image was quite well balanced, this has made the dark areas too dark and the bright areas too bright.

Functions of the kind shown in the right plot have a large derivative near $x = 0$ and will therefore increase the contrast in an image with a large proportion of small intensity values, i.e., very dark images. These functions are given by

$$g_\epsilon(x) = \frac{\ln(x + \epsilon) - \ln\epsilon}{\ln(1 + \epsilon) - \ln\epsilon}, \tag{9.2}$$

and the ones shown in the plot correspond to $\epsilon = 0.1$, 0.01, and 0.001. In the right plot in Figure 9.9 the middle function has been applied to the same image. This has made the image as a whole too bright, but has brought out the details of the road which was very dark in the original.

Increasing the contrast is easy to implement. The following function uses the contrast adjusting function from Equation (9.2), with ϵ as in that equation 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.

What you should have learned in this section.

- 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.

Exercise 9.1: 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 Figure 9.2(b).

Exercise 9.2: Adjust contrast in images 1

Generate the right image in Figure 9.9 on your own by writing code which uses the function `contrastadjust`.

Exercise 9.3: Adjust contrast in images 2

Let us also consider the second way we mentioned for increasing the contrast.

a) Write a function `contrastadjust0` which instead uses the function from Equation (9.1) in the compendium to increase the contrast. n should be a parameter to the function.

b) Generate the left image in Figure 9.9 on your own by using your code from Exercise 9.2, and instead calling the function `contrastadjust0`.

Exercise 9.4: Adjust contrast in images 3

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.

Figure 9.10: 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.

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 9.13. *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 9.14. *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

$$Y_{i,j} = \sum_{k_1} \sum (t_1)_{k_1} X_{i-k_1,j}.$$

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 $S_1 X$ 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 9.15. *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 9.5 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 9.16. *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 9.17. *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 9.18. *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 9.9. We can now prove the following.

Theorem 9.19. *Comact filter notationa 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

$$\begin{aligned}(S_1 \otimes S_2)(\boldsymbol{e}_i \otimes \boldsymbol{e}_j) &= (S_1\boldsymbol{e}_i) \otimes (S_2\boldsymbol{e}_j) \\ &= (\mathrm{col}_i(S_1)) \otimes (\mathrm{col}_j(S_2)) = \mathrm{col}_i(S_1)(\mathrm{col}_j(S_2))^T \\ &= \mathrm{col}_i(S_1)\mathrm{row}_j((S_2)^T) = S_1(\boldsymbol{e}_i \otimes \boldsymbol{e}_j)(S_2)^T.\end{aligned}$$

This means that $(S_1 \otimes S_2)X = S_1X(S_2)^T$ 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 9.20. *Composing tensor products.*
We have that $(S_1 \otimes T_1)(S_2 \otimes T_2) = (S_1S_2) \otimes (T_1T_2)$.

Proof. By Theorem 9.19 we have that

$$(S_1 \otimes T_1)(S_2 \otimes T_2)X = S_1(S_2XT_2^T)T_1^T = (S_1S_2)X(T_1T_2)^T = ((S_1S_2) \otimes (T_1T_2))X.$$

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 9.20 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 9.21. *Smoothing an image.*

When we considered filtering of digital sound, we observed that replacing each sample of a sound by an average of the sample and its neighbours dampened the high frequencies of the sound. Let us consider the computational molecules where such a filter is applied to both the rows and the columns. For the one-dimensional case on sound, we argued that filter coefficients taken from Pascal's triangle give good smoothing effects. The same can be argued for images. If we use the filter $S = \frac{1}{4}\{1, \underline{2}, 1\}$ (row 2 from Pascal's triangle), Theorem 9.14 says that we obtain the computational molecule

$$A = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & \underline{4} & 2 \\ 1 & 2 & 1 \end{pmatrix}. \tag{9.6}$$

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

$$\frac{1}{4096}\begin{pmatrix} 1 & 6 & 15 & 20 & 15 & 6 & 1 \\ 6 & 36 & 90 & 120 & 90 & 36 & 6 \\ 15 & 90 & 225 & 300 & 225 & 90 & 15 \\ 20 & 120 & 300 & \underline{400} & 300 & 120 & 20 \\ 15 & 90 & 225 & 300 & 225 & 90 & 15 \\ 6 & 36 & 90 & 120 & 90 & 36 & 6 \\ 1 & 6 & 15 & 20 & 15 & 6 & 1 \end{pmatrix}. \tag{9.7}$$

For both molecules the weights sum to one, so that the new intensity values $\hat{p}_{i,j}$ are weighted averages of the intensity values on the right. 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.

Figure 9.12: The results of smoothing the simple image to the left with the filter $\frac{1}{4}\{1,\underline{2},1\}$ horizontally, vertically, and both, respectively.

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. The smoothing effects are then only seen in one of the vertical or horizontal directions. In the right image we have smoothed in both directions. We clearly see the union of the two one-dimensional smoothing operations then.

Let us summarize from this example as follows.

Observation 9.22. *Smoothing an image.*
An image P can be smoothed by applying a smoothing filter to the rows, and then to the columns.

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 we have a perfect situation for applying numerical differentiation. Since a grey-level image is a scalar function of two variables, numerical differentiation techniques can be applied.

Partial derivative in x-direction. Let us first consider computation of the partial derivative $\partial P/\partial x$ at all points in the image. Note first that it is the second coordinate in an image which refers to the x-direction used when plotting functions. This means that the familiar symmetric Newton quotient approximation for the partial derivative [23] 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 computational molecules as follows.

Observation 9.23. *The partial derivative $\partial P/\partial x$.*
Let $P = (p_{i,j})_{i,j=1}^{m,n}$ be a given image. The partial derivative $\partial P/\partial x$ of the image can be computed with the computational molecule

$$\frac{1}{2}\begin{pmatrix} 0 & 0 & 0 \\ 1 & \underline{0} & -1 \\ 0 & 0 & 0 \end{pmatrix}. \tag{9.9}$$

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. If we apply the `smooth`-function to the same excerpt of the Lena image with this molecule, we obtain the left image in Figure 9.13. It 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 $[-0.424, 0.418]$. We therefore normalise and map all intensities to $[0, 1]$. The result of this is shown in the middle image. 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

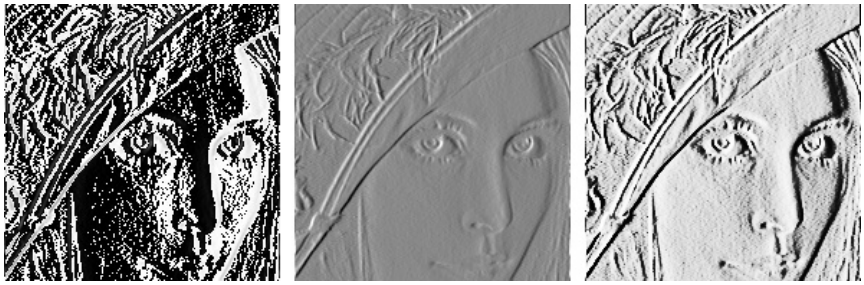

Figure 9.13: Experimenting with the partial derivative in the x-direction for the image in 9.6. The left image has artefacts, since the pixel values are outside the legal range. We therefore normalize the intensities to lie in $[0, 25]$ (middle), before we increase the contrast (right).

each intensity value. The result is shown in the right image in Figure 9.13 which does indeed show more detail.

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 middle image in Figure 9.13, 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 right image in Figure 9.13 these values have just been emphasised even more.

The right image in Figure 9.13 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.

Observation 9.24. *The partial derivative $\partial P/\partial y$.*

Let $P = (p_{i,j})_{i,j=1}^{m,n}$ be a given image. The partial derivative $\partial P/\partial y$ of the image can be computed with the computational molecule

$$\frac{1}{2}\begin{pmatrix} 0 & 1 & 0 \\ 0 & \underline{0} & 0 \\ 0 & -1 & 0 \end{pmatrix}. \tag{9.10}$$

The result is shown in Figure 9.15(b). 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; the resulting is shown as an image in Figure 9.14c.

Figure 9.14: 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.

The left image in Figure 9.14 shows the computed values of the gradient. Note that it is possible that the length of the gradient could be outside the legal range of values. This would have been seen as artefacts in this image. In the middle image the intensities have been mapped to the legal range. To enhance the contrast further we have to do something different from what was done in the other images since we now have a large number of intensities near 0. The solution is to apply a function like the ones shown in the right plot in Figure 9.8 to the intensities. If we use the function $g_{0.01}$ defined in equation(9.2) we obtain the right image in Figure 9.14.

Figure 9.15: 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.

9.3.2 Comparing the first derivatives

Figure 9.15 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.

9.3.3 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 [23]

$$\frac{\partial P}{\partial x^2}(i,j) \approx p_{i,j+1} - 2p_{i,j} + p_{i,j-1} \tag{9.11}$$

to the second derivative (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 $S^2 = \frac{1}{2}\{1, \underline{0}, -1\}\frac{1}{2}\{1, \underline{0}, -1\} = \frac{1}{4}\{1, 0, \underline{-2}, 0, 1\}$.

Observation 9.25. *Second order derivatives of an image.*

The second order derivatives of an image P can be computed by applying the computational molecules

$$\frac{\partial^2 P}{\partial x^2}: \quad \begin{pmatrix} 0 & 0 & 0 \\ 1 & \underline{-2} & 1 \\ 0 & 0 & 0 \end{pmatrix}, \tag{9.12}$$

$$\frac{\partial^2 P}{\partial y \partial x}: \quad \frac{1}{4}\begin{pmatrix} -1 & 0 & 1 \\ 0 & \underline{0} & 0 \\ 1 & 0 & -1 \end{pmatrix}, \tag{9.13}$$

$$\frac{\partial^2 P}{\partial y^2}: \quad \begin{pmatrix} 0 & 1 & 0 \\ 0 & \underline{-2} & 0 \\ 0 & 1 & 0 \end{pmatrix}. \tag{9.14}$$

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.

With the information in Observation 9.25 it is quite easy to compute the second-order derivatives, and the results are shown in Figure 9.16. 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.

Let us also 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.

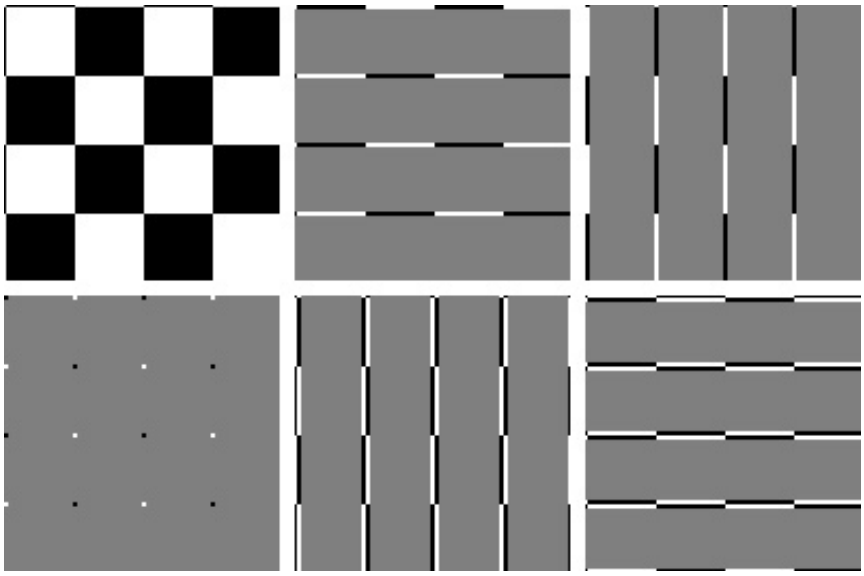

Figure 9.17: Different tensor products representing partial derivatives applied to a simple chess pattern example image (upper left). The tensor products are $S \otimes I$, $I \otimes S$, $S \otimes S$, $I \otimes S^2$, and $S^2 \otimes I$.

These images make it is clear that $S \otimes I$ detects all horizontal edges, that $I \otimes S$ 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.

What you should have learned in this section.

- 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.

Exercise 9.5: 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 9.6: 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 figures 9.14.

d) The images in Figure 9.15.

e) The images in Figure 9.16.

Exercise 9.7: 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 learnt, and interpret the result. Do the same for $(I \otimes S)X$.

Exercise 9.8: 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 9.9: 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 9.10: 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 9.11: 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 learnt. If the matrix A was more generally an image, what can you say about how the new image will look?

Exercise 9.12: Computational molecules

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 9.13: 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 9.14: 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 9.15: 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 the compendium 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 9.26. *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 9.9). 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 9.27. *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 9.28. *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

$$Y = S_1 X (S_2)^T. \tag{9.15}$$

Proof. Let $\boldsymbol{c}_{ki}$ be the i'th basis vector in $\mathcal{C}_k$, $\boldsymbol{b}_{ki}$ the i'th basis vector in $\mathcal{B}_k$, $k = 1, 2$. Since any change of coordinates is linear, it is enough to show that it coincides with $X \to S_1 X (S_2)^T$ for any basis vector in $\mathcal{B}_1 \otimes \mathcal{B}_2$. The basis vector $\boldsymbol{b}_{1i} \otimes \boldsymbol{b}_{2j}$ has coordinate vector $X = \boldsymbol{e}_i \otimes \boldsymbol{e}_j$ in $\mathcal{B}_1 \otimes \mathcal{B}_2$. With the mapping $X \to S_1 X (S_2)^T$ this is sent to

$$S_1 X (S_2)^T = S_1(\boldsymbol{e}_i \otimes \boldsymbol{e}_j)(S_2)^T = S_1 \boldsymbol{e}_i (\boldsymbol{e}_j)^T (S_2)^T = S_1 \boldsymbol{e}_i (S_2 \boldsymbol{e}_j)^T = \mathrm{col}_i(S_1)(\mathrm{col}_j(S_2))^T.$$

On the other hand, since column i in S_1 is the coordinates of $\boldsymbol{b}_{1i}$ in the basis $\mathcal{C}_1$, and column j in S_2 is the coordinates of $\boldsymbol{b}_{2j}$ in the basis $\mathcal{C}_2$, we can write

$$\begin{aligned}
\boldsymbol{b}_{1i} \otimes \boldsymbol{b}_{2j} &= \left(\sum_k (S_1)_{k,i} \boldsymbol{c}_{1k}\right) \otimes \left(\sum_l (S_2)_{l,j} \boldsymbol{c}_{2l}\right) = \sum_{k,l} (S_1)_{k,i} (S_2)_{l,j} (\boldsymbol{c}_{1k} \otimes \boldsymbol{c}_{2l}) \\
&= \sum_{k,l} (\mathrm{col}_i(S_1))_k (\mathrm{col}_j(S_2))_l (\boldsymbol{c}_{1k} \otimes \boldsymbol{c}_{2l}) \\
&= \sum_{k,l} (\mathrm{col}_i(S_1)(\mathrm{col}_j(S_2))^T)_{k,l} (\boldsymbol{c}_{1k} \otimes \boldsymbol{c}_{2l}).
\end{aligned}$$

This shows that the coordinate matrix of $\boldsymbol{b}_{1i} \otimes \boldsymbol{b}_{2j}$ in $\mathcal{C}_1 \otimes \mathcal{C}_2$ is $\text{col}_i(S_1)(\text{col}_j(S_2))^T$. This means that the change of coordinates coincides with the mapping $X \to S_1 X (S_2)^T$ for any vector in $\mathcal{B}_1 \otimes \mathcal{B}_2$, so that the change of coordinates is given by $X \to S_1 X (S_2)^T$ for all vectors also. □

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 9.29. *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 funtion `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 9.15 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 9.30. *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 2.28, 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 9.31. *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.

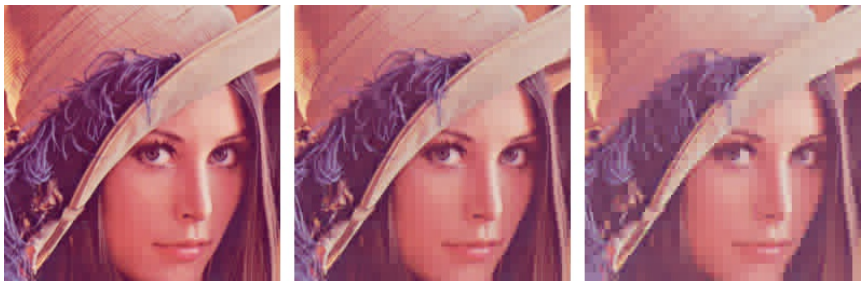

Figure 9.20: The effect on an image when it is transformed with the DCT, and the DCT-coefficients below a certain threshold are zeroed out. The threshold has been increased from left to right, from 30, to 50, and 100. The percentage of pixel values that were zeroed out are 93.2, 95.8, and 97.7, respectively.

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.

What you should have learned in this section.

- 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.

[1] The JPEG standard does not do exactly the kind of thresholding described here. Rather it performs what is called a quantization.

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.

Exercise 9.16: 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 which applies the DFT, IDFT, DCT, and IDCT, to consecutive segments of length 8.

Exercise 9.17: Implement two-dimensional FFT and DCT

Write down code for running FFT2, IFFT2, DCT2, and IDCT2 on an image, using the function `tensor_impl`.

Exercise 9.18: Zeroing out DCT coefficients

The following function `showDCThigher` applies the DCT to an image in the same way as the JPEG standard does. The function takes a threshold parameter, and sets DCT coefficients below this value to zero:

```
def showDCThigher(threshold):
    img = imread('images/lena.png', 'png').astype(float)
    zeroedout = 0
    tensor_impl(img, DCTImpl8, DCTImpl8)
    thresholdmatr = (abs(img) >= threshold)
    zeroedout += prod(shape(img)) - sum(thresholdmatr)
    img *= thresholdmatr
    tensor_impl(img, IDCTImpl8, IDCTImpl8)
    mapto01(img)
    imshow(uint8(255*img))
    print '%i percent of samples zeroed out\n' \
    % 100*zeroedout/prod(shape(img))
```

a) Explain this code line by line.

b) Run `showDCThigher` for different threshold parameters, and check that this reproduces the test images of this section, and prints the correct numbers of values which have been neglected (i.e. which are below the threshold) on screen.

Exercise 9.19: 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 [15]. 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 [22]) 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 [27].

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 [17] 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.

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 9.15.

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.

Example 10.3. *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.

Example 10.4. *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 9.26:

Theorem 10.5. *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 9.26: 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 10.6. *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 9.28 can also be proved in the same way in the context of function spaces. We state this as follows:

Theorem 10.7. *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 10.5:

Corollary 10.8. *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 10.9. *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

$$\begin{aligned}\sum_{n_1,n_2=0}^{2^{m-1}N} &(c_{m-1,n_1,n_2}(\phi_{m-1,n_1} \otimes \phi_{m-1,n_2}) + w_{m-1,n_1,n_2}^{(0,1)}(\phi_{m-1,n_1} \otimes \psi_{m-1,n_2}) + \\ &w_{m-1,n_1,n_2}^{(1,0)}(\psi_{m-1,n_1} \otimes \phi_{m-1,n_2}) + w_{m-1,n_1,n_2}^{(1,1)}(\psi_{m-1,n_1} \otimes \psi_{m-1,n_2}))\end{aligned} \tag{10.5}$$

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 10.10. *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

$$\mathrm{proj}_{V_0 \otimes V_0}(\phi_{1,k_1} \otimes \phi_{1,k_2}) = \begin{cases} \frac{1}{2}(\phi_{0,k_1/2} \otimes \phi_{0,k_2/2}) & \text{when } k_1, k_2 \text{ are even} \\ \frac{1}{2}(\phi_{0,k_1/2} \otimes \phi_{0,(k_2-1)/2}) & \text{when } k_1 \text{ is even, } k_2 \text{ is odd} \\ \frac{1}{2}(\phi_{0,(k_1-1)/2} \otimes \phi_{0,k_2/2}) & \text{when } k_1 \text{ is odd, } k_2 \text{ is even} \\ \frac{1}{2}(\phi_{0,(k_1-1)/2} \otimes \phi_{0,(k_2-1)/2}) & \text{when } k_1, k_2 \text{ are odd} \end{cases}$$

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 $\phi \otimes \phi \in V_0 \otimes V_0$.

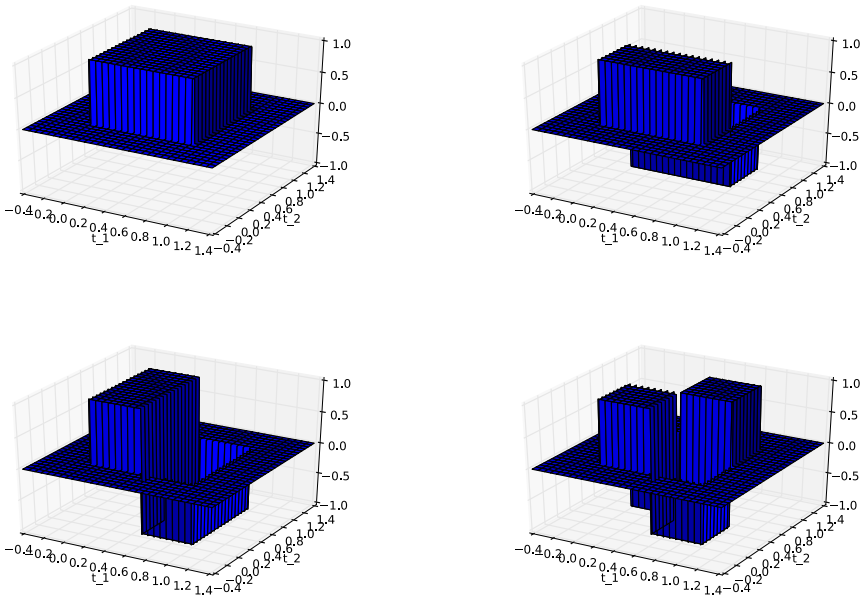

Figure 10.1: The functions $\phi \otimes \phi$, $\phi \otimes \psi$, $\psi \otimes \phi$, $\psi \otimes \psi$, which are bases for $(V_0 \otimes V_0) \oplus W_0^{(0,1)} \oplus W_0^{(1,0)} \oplus W_0^{(1,1)}$ for the Haar wavelet.

Example 10.11. *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.

An immediate corollary of Theorem 10.7 is the following:

Corollary 10.12. *Implementing tensor product.*

Let

$$A_m = P_{(\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1}) \leftarrow \boldsymbol{\phi}_m}$$
$$B_m = P_{\boldsymbol{\phi}_m \leftarrow (\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1})}$$

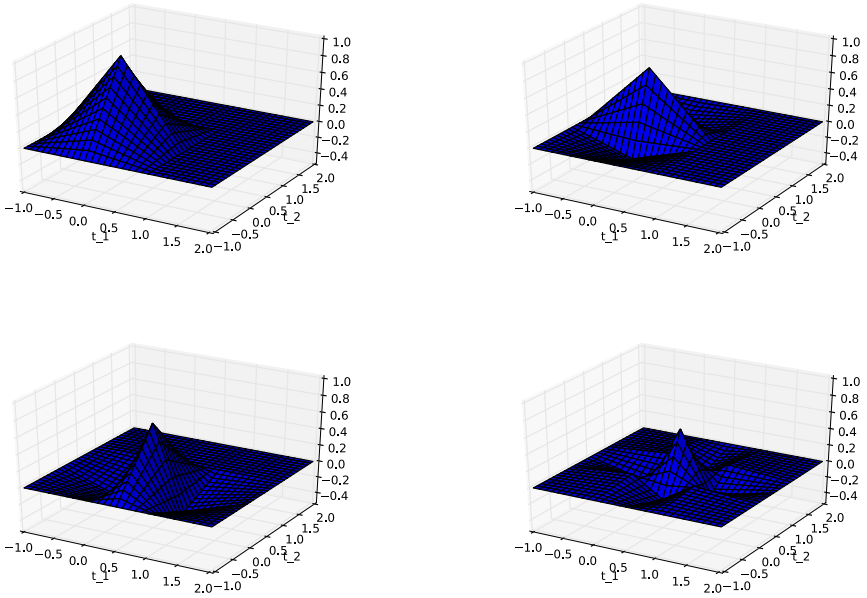

Figure 10.2: The functions $\phi \otimes \phi$, $\phi \otimes \psi$, $\psi \otimes \phi$, $\psi \otimes \psi$, which are bases for $(V_0 \otimes V_0) \oplus W_0^{(0,1)} \oplus W_0^{(1,0)} \oplus W_0^{(1,1)}$ for the alternative piecewise linear wavelet.

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 10.1 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 9.28. 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.

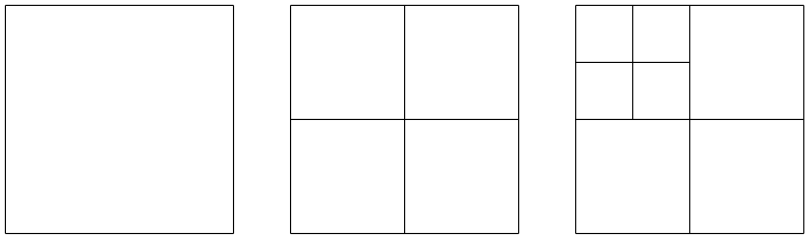

Figure 10.3: Illustration of the different coordinates in a two level DWT2 before the first stage is performed (left), after the first stage (middle), and after the second stage (right).

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.

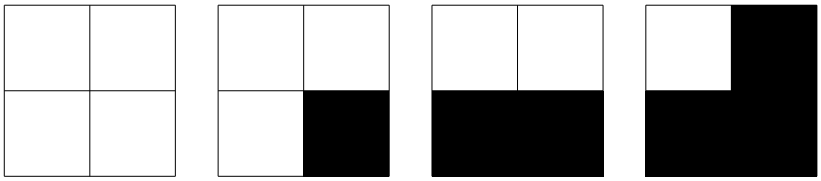

Figure 10.4: Graphical representation of neglecting the wavelet coefficients at the first level. After applying DWT2, the wavelet coefficients are split into four parts, as shown in the left figure. In the following figures we have removed coefficients from $W_1^{(1,1)}$, $W_1^{(1,0)}$, and $W_1^{(0,1)}$, in that order.

Figure 10.5 illustrates in the same way incremental removal of the subbands at the second level.

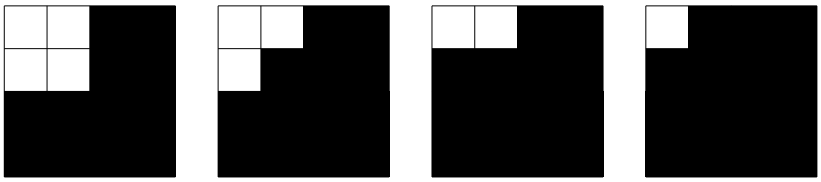

Figure 10.5: Graphical representation of neglecting the wavelet coefficients at the second level. After applying the second stage in DWT2, the wavelet coefficients from the upper left corner are also split into four parts, as shown in the left figure. In the following figures we have removed coefficients 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 10.13. *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.

What you should have learned in this section.

- 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.

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 10.14. *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 10.13, 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.

In Figure 10.6 we have applied one level of the DWT2 to the chess pattern example image, and all these effects are seen clearly here.

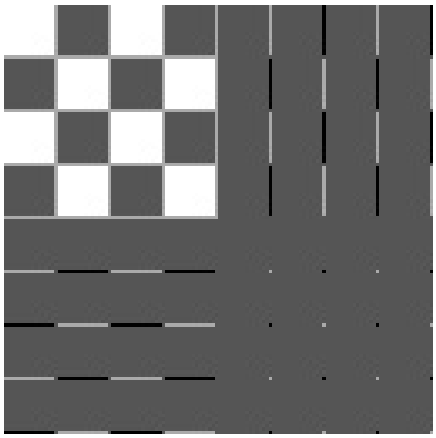

Figure 10.6: The chess pattern example image after application of the DWT2. The Haar wavelet was used.

Example 10.15. *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, DWTKernelHaar)
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 $[0, 255]$.

Figure 10.7: The corresponding thumbnail images for the Image of Lena, obtained with a DWT of 1, 2, 3, and 4 levels.

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.

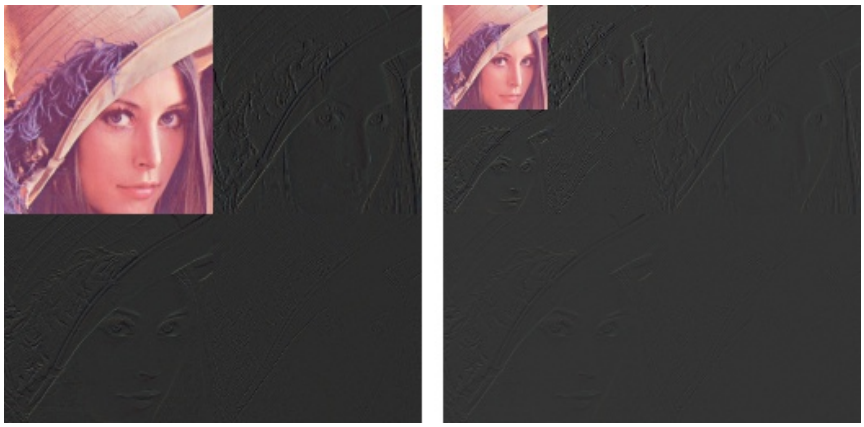

Figure 10.8: The corresponding image resulting from a wavelet transform with the Haar-wavelet for $m = 1$ and $m = 2$.

Example 10.16. *Detail and low-resolution approximations with the Haar wavelet.*

In Exercise 10.4 you will be asked to implement a function `showDWT` which displays the low-resolution approximations or the detail components for our test

image for any wavelet, using functions we have previously implemented. Let us take a closer look at the images generated when the Haar wavelet is used. 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. Since the Haar wavelet has few vanishing moments, we should expect that the lower order resolution approximations from V_0 are worse when m increase. Figure 10.9 confirms this for the lower order resolution approximations.

Figure 10.9: Image of Lena, with detail at the first 1, 2, 3, and 4 levels zeroed out, respectively, for the Haar wavelet.

Alternatively, we should see that the higher order detail spaces contain more information. The new images when `showDWT` is used to display the detail components for the Haar wavelet are shown in Figure 10.10.

Figure 10.10: The corresponding detail for the images in Figure 10.9, with the Haar wavelet.

The black color indicates values which are close to 0. In other words, most of the coefficients are close to 0, which reflects one of the properties of the wavelet.

Example 10.17. *Experimenting with different wavelets.*

Using the function `showDWT`, we can display the low-resolution approximations at a given resolution of our image test file `lena.png`, for the Spline 5/3 and CDF 9/7 wavelets in addition to the Haar wavelet, with the following code:

```
showDWT(m, DWTKernelHaar, IDWTKernelHaar, True)
showDWT(m, DWTKernel53, IDWTKernel53, True)
showDWT(m, DWTKernel97, IDWTKernel97, True)
```

The first call to `showDWT` displays the result using the Haar wavelet. The second call to `showDWT` moves to the Spline 5/3 wavelet, and the third call uses the CDF

9/7 wavelet. We can repeat this for various number of levels m, and compare the different images.

Example 10.18. *The Spline 5/3 wavelet and removing bands in the detail spaces.*

Since the detail components now 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 10.11.

Figure 10.11: 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.

The resulting images when the bands on the second level indicated in Figure 10.5 are removed are shown in Figure 10.12.

Figure 10.12: 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.

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 10.13 we have also shown the resulting image after the third and fourth level of detail have been neglected. Although we still can see details in the image, the quality in the image is definitely poorer.

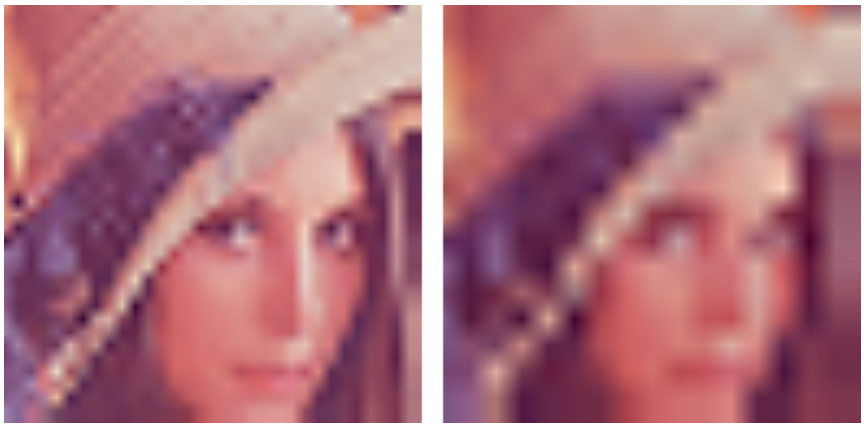

Figure 10.13: Image of Lena, with detail including level 3 and 4 zeroed out. The Spline 5/3 wavelet was used.

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 10.14, we have shown the corresponding detail for the two right images in Figure 10.11, and Figure 10.13. Clearly, more detail can be seen in the image when more of the detail is included.

Example 10.19. *the CDF 9/7 wavelet.*

Let us repeat the previous example for the CDF 9/7 wavelet, using the function `showDWT` you implemented in Exercise 10.4. We should now see improved images when we discard the detail in the images. Figure 10.15 confirms this for the lower resolution spaces,

while Figure 10.16 confirms this for the higher order detail spaces.

As mentioned, the procedure developed in this section for applying a wavelet transform to an image with the help of the tensor product construction, is adopted in the JPEG2000 standard. This lossy (can also be used as lossless) image format was developed by the Joint Photographic Experts Group and published in 2000. After significant processing of the wavelet coefficients, the final coding with JPEG2000 uses an advanced version of arithmetic coding. At the cost of increased encoding and decoding times, JPEG2000 leads to as much as 20 % improvement in compression ratios for medium compression rates, possibly more for high or low compression rates. The artefacts are less visible than in JPEG and appear at higher compression rates. Although a number of components in JPEG2000 are patented, the patent holders have agreed that the

Figure 10.14: The corresponding detail for the image of Lena. The Spline 5/3 wavelet was used.

core software should be available free of charge, and JPEG2000 is part of most Linux distributions. However, there appear to be some further, rather obscure, patents that have not been licensed, and this may be the reason why JPEG2000 is not used more. The extension of JPEG2000 files is `.jp2`.

What you should have learned in this section.

- 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.

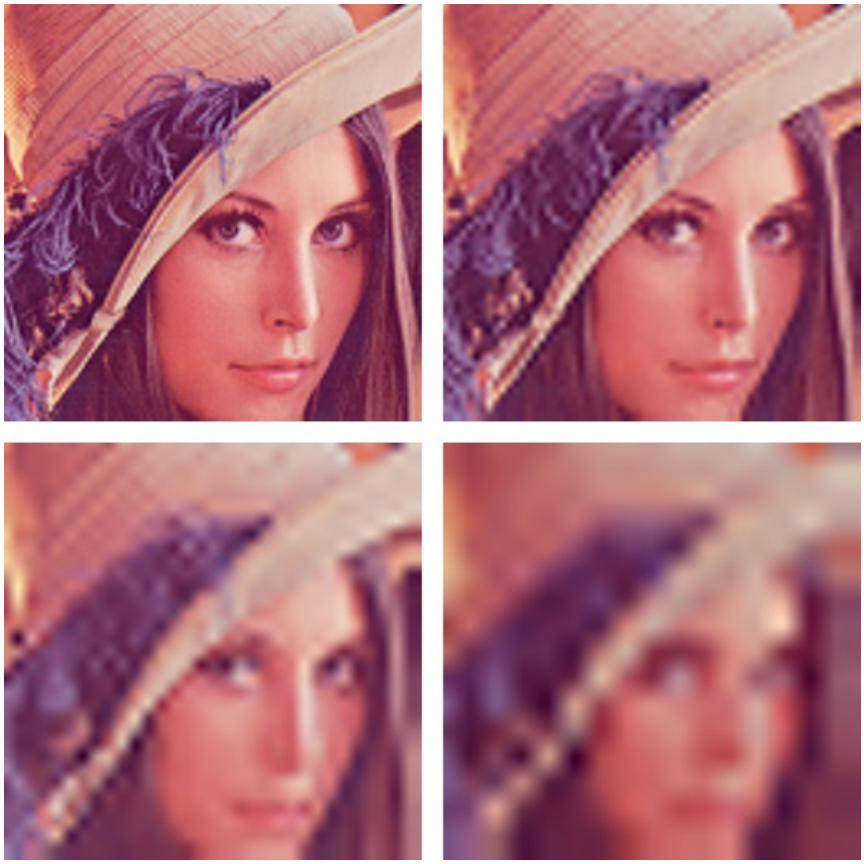

Figure 10.15: Image of Lena, with higher levels of detail neglected. The CDF 9/7 wavelet was used.

Exercise 10.1: Implement two-dimensional DWT

Implement functions `DW2TImpl` and `IDW2TImpl` which perform the m-level DWT2 and the IDWT2, respectively, on an image. The functions should take the same input as `DWTImpl` and `IDWTImpl`, with the input vector replaced with a two-dimensional object. The functions should at each stage call `DWTImpl` and `IDWTImpl` with $m = 1$, and each call to these functions should alter the appropriate upper left submatrix in the coordinate matrix. 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'.

Figure 10.16: The corresponding detail for the image of Lena. The CDF 9/7 wavelet was used.

Exercise 10.2: 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[0:M:2, n])/sqrt(2)
    w = (X[0:M:2, n] - X[0:M:2, n])/sqrt(2)
    X[:, n] = vstack([c, w])

for m in range(M):
    c = (X[m, 0:N:2] + X[m, 0:N:2])/sqrt(2)
    w = (X[m, 0:N:2] - X[m, 0:N:2])/sqrt(2)
    X[m, :] = hstack([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?

Exercise 10.3: 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):
    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 10.4: Zeroint out DWT coefficients

In this exercise we will experiment with applying the m-level DWT2 to an image.

a) Write a function `showDWT`, which takes m, a DWT kernel `f`, an IDWT kernel `invf`, and a variable `lowres` as input, and

- reads the image file `lena.png`,
- performs an m-level DWT2 on the image samples using the function `DW2TImpl`, with DWT kernel `f`
- sets all wavelet coefficients representing detail to zero if `lowres` is true (i.e. keep only the low-resolution coordinates from $\boldsymbol{\phi}_0 \otimes \boldsymbol{\phi}_0$),

- sets all low-resolution coordinates to zero if `lowres` is false (i.e. keep only the detail coordinates),
- performs an IDWT2 on the resulting coefficients using the function `IDW2TImpl`, with IDWT kernel `invf`,
- displays the resulting image.

b) Do the image samples returned by `showDWT` lie in $[0, 255]$?

c) Run the function `showDWT` for different values of m for the Haar wavelet, with `lowres` set to true. Describe what you see for different m. For which m can you see that the image gets degraded? How does it get degraded? Compare with what you saw with the function `showDCThigher` in Exercise 9.18, where you performed a DCT on the image samples instead, and set DCT coefficients below a given threshold to zero.

d) Repeat what you did in c., but this time with `lowres` set to false instead. What kind of image do you see now? Can you recognize the original image in what you see? Try to explain why the images seem to get clearer when you increase m.

e) In the code in Example 10.17, set `lowres` to false in the call to `showDWT` also for the other wavelets. and repeat what you did in d..

Exercise 10.5: 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 Figure 10.6, even though the images are very similar. Attempt to explain what causes this to happen.

Hint. Compare with Exercise 5.17.

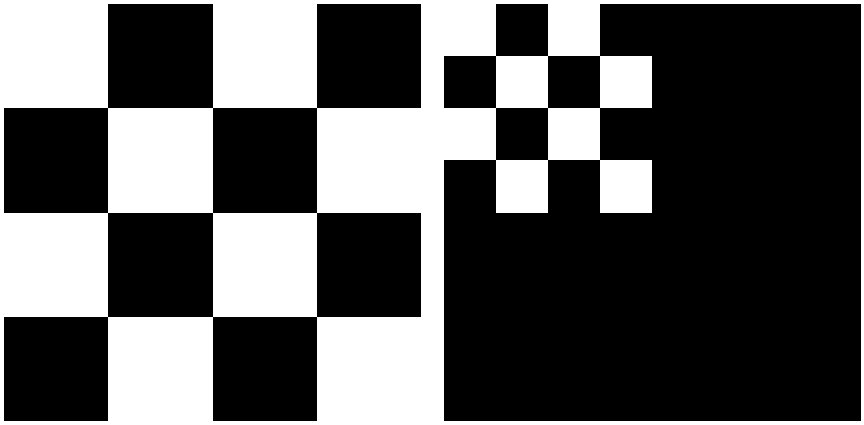

Figure 10.17: A simple image before and after one level of the DWT2. The Haar wavelet was used.

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).

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

Figure 10.18: A typical fingerprint image.

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.

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.

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.

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

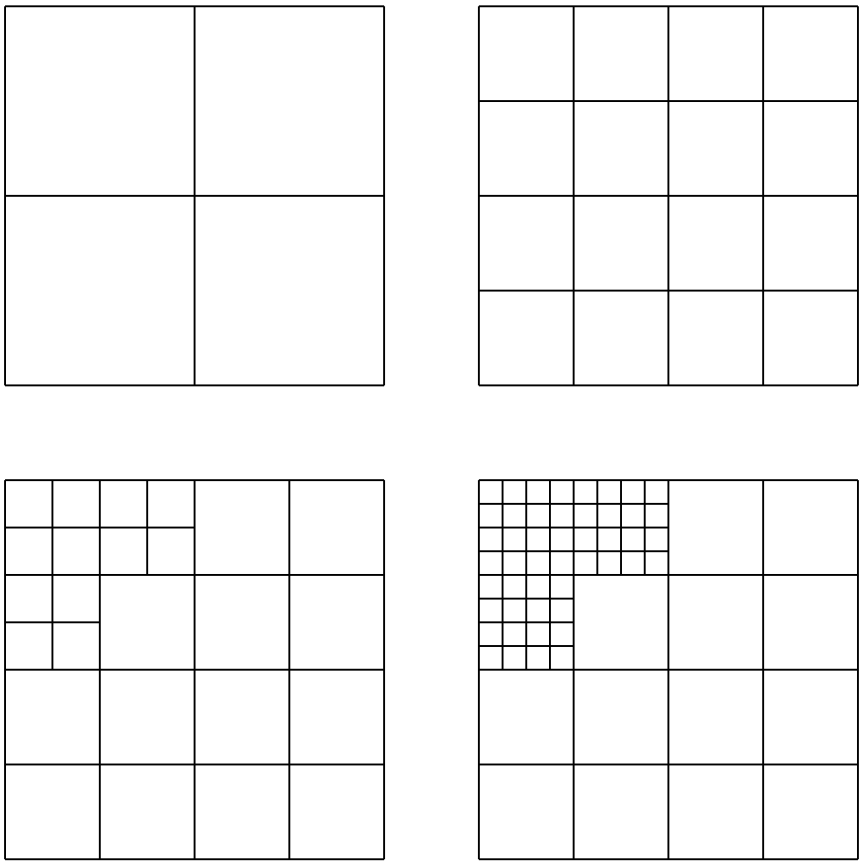

Figure 10.19: Subband structure after the different stages of the wavelet applications in the FBI fingerprint compression scheme.

standard, which makes the two standards more compatible. IN FBI's system, there are also other important parts besides the actual compression strategy, such as *fingerprint pattern matching*: In order to match a fingerprint quickly with the records in the database, several characteristics of the fingerprints are stored, such as the number of lines in the fingerprint, and points where the lines split or join. When the database is indexed with this information, one may not need to decompress all images in the database to perform matching. We will not go into details on this here.

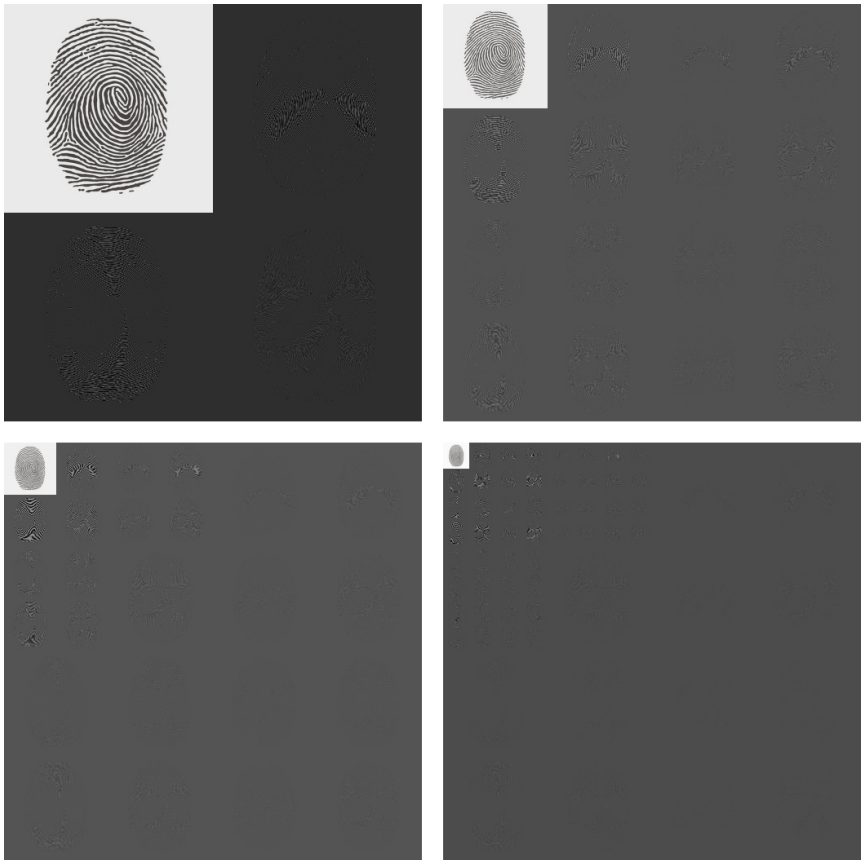

Figure 10.20: The fingerprint image after several DWT's.

Exercise 10.6: 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.

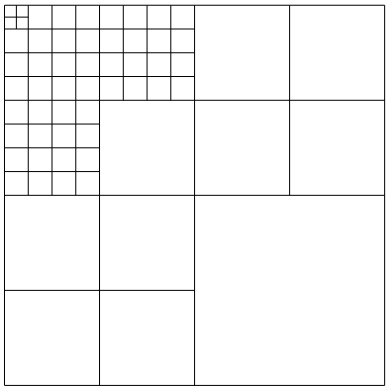

Figure 10.21: Subbands structure after all stages.

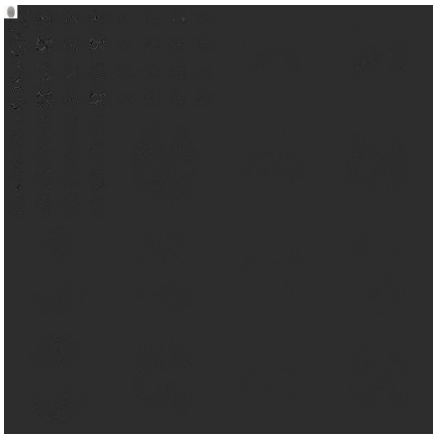

Figure 10.22: The resulting image obtained with the subband decomposition employed by the FBI.

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.

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 [17]. In [36], 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 [11]. In [2], the theory is described. The book [13] uses the application to compression of fingerprint images as an example of the usefulness of recent developments in wavelet theory.

Appendix A

Basic Linear Algebra

This book assumes that the student has taken a beginning course in linear algebra at university level. In this appendix we summarize the most important concepts one needs to know from linear algebra. Note that what is listed here should not be considered as a substitute for such a course: It is important for the student to go through a full course in linear algebra, in order to get good intuition for these concepts through extensive exercises. Such exercises are omitted here.

A.1 Matrices

An $m \times n$-matrix is simply a set of mn numbers, stored in m rows and n columns. We write a_{kn} for the entry in row k and column n of the matrix A. The zero matrix, denoted $\mathbf{0}$ is the matrix with all zeroes. A square matrix (i.e. where $m = n$) is said to be diagonal if $a_{kn} = 0$ whenever $k \neq n$. The identity matrix, denoted I, or I_n to make the dimension of the matrix clear, is the diagonal matrix where the entries on the diagonal are 1, the rest zeroes. If A is a matrix we will denote the transpose of A by A^T. If A is invertible we denote its inverse by A^{-1}. We say that a matrix A is *orthogonal* if $A^T A = AA^T = I$. A matrix is called sparse if most of the entries in the matrix are zero.

A.2 Vector spaces

A set of vectors V is called a vector space if ... We say that the vectors $\{\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{n-1}\}$ are *linearly independent* if, whenever $\sum_{i=0}^{n-1} c_i \boldsymbol{v}_i = \mathbf{0}$, we must have that all $c_i = 0$. We will say that a set of vectors $\mathcal{B} = \{\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{n-1}\}$ from V is a *basis* for V if the vectors are linearly independent, and span V.

Subspaces of $\mathbb{R}^N$, and function spaces.

A.3 Inner products and orthogonality

Most vector spaces in this book are inner product spaces. A (real) inner product on a vector space is a binary operation, written as $(\boldsymbol{u}, \boldsymbol{v}) \to \langle \boldsymbol{u}, \boldsymbol{v} \rangle$, which fulfills the following properties for any vectors $\boldsymbol{u}$, $\boldsymbol{v}$, and $\boldsymbol{w}$:

- $\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \langle \boldsymbol{v}, \boldsymbol{u} \rangle$
- $\langle \boldsymbol{u} + \boldsymbol{v}, \boldsymbol{w} \rangle = \langle \boldsymbol{u}, \boldsymbol{w} \rangle + \langle \boldsymbol{v}, \boldsymbol{w} \rangle$
- $\langle c\boldsymbol{u}, \boldsymbol{v} \rangle = c\langle \boldsymbol{u}, \boldsymbol{v} \rangle$ for any scalar c
- $\langle \boldsymbol{u}, \boldsymbol{u} \rangle \geq 0$, and $\langle \boldsymbol{u}, \boldsymbol{u} \rangle = 0$ if and only if $\boldsymbol{u} = 0$.

$\boldsymbol{u}$ and $\boldsymbol{v}$ are said to be *orthogonal* if $\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \boldsymbol{0}$. In this book we have seen two important examples of inner product spaces. First of all the Euclidean inner product, which is defined by

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \sum_{i=0}^{n-1} u_i v_i \tag{A.1}$$

for any $\boldsymbol{u}$, $\boldsymbol{v}$ in $\mathbb{R}^n$. For functions we have seen examples which are variants of the following form:

$$\langle f, g \rangle = \int f(t)g(t)dt. \tag{A.2}$$

Any set of mutually orthogonal elements are also linearly independent. A basis where all basis vectors are mutually orthogonal is called an *orthogonal basis*. If additionally the vectors all have length 1, we say that the basis is *orthonormal*. If $\boldsymbol{x}$ is in a vector space with an orthogonal basis $\mathcal{B} = \{\boldsymbol{v}_k\}_{k=0}^{n-1}$, we can express $\boldsymbol{x}$ as

$$\frac{\langle \boldsymbol{x}, \boldsymbol{v}_0 \rangle}{\langle \boldsymbol{v}_0, \boldsymbol{v}_0 \rangle} \boldsymbol{v}_0 + \frac{\langle \boldsymbol{x}, \boldsymbol{v}_1 \rangle}{\langle \boldsymbol{v}_1, \boldsymbol{v}_1 \rangle} \boldsymbol{v}_1 + \cdots + \frac{\langle \boldsymbol{x}, \boldsymbol{v}_{n-1} \rangle}{\langle \boldsymbol{v}_{n-1}, \boldsymbol{v}_{n-1} \rangle} \boldsymbol{v}_{n-1}. \tag{A.3}$$

In other words, the weights in linear combinations are easily found when the basis is orthogonal. This is also called the *orthogonal decomposition theorem*.

By the *projection* of a vector $\boldsymbol{x}$ onto a subspace U we mean the vector $\boldsymbol{y} = \text{proj}_U \boldsymbol{x}$ which minimizes the distance $\|\boldsymbol{y} - \boldsymbol{x}\|$. If $\boldsymbol{v}_i$ is an orthogonal basis for U, we have that $\text{proj}_U \boldsymbol{x}$ can be written by Equation (A.3).

A.4 Coordinates and change of coordinates

If $\mathcal{B} = \{\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{n-1}\}$ is a basis for a vector space, and $\boldsymbol{x} = \sum_{i=0}^{n-1} x_i \boldsymbol{v}_i$, we say that $(x_0, x_1, \ldots, x_{n-1})$ is the *coordinate vector* of $\boldsymbol{x}$ w.r.t. the basis $\mathcal{B}$. We also write $[\boldsymbol{x}]_{\mathcal{B}}$ for this coordinate vector.

If $\mathcal{B}$ and $\mathcal{C}$ are two different bases for the same vector space, we can write down the two coordinate vectors $[\boldsymbol{x}]_{\mathcal{B}}$ and $[\boldsymbol{x}]_{\mathcal{C}}$. A useful operation is to transform the coordinates in $\mathcal{B}$ to those in $\mathcal{C}$, i.e. apply the transformation which sends $[\boldsymbol{x}]_{\mathcal{B}}$ to $[\boldsymbol{x}]_{\mathcal{C}}$. This is a linear transformation, and we will denote the $n \times n$-matrix of this linear transformation by $P_{\mathcal{C}\leftarrow\mathcal{B}}$, and call this the *change of coordinate matrix* from $\mathcal{B}$ to $\mathcal{C}$ In other words, the change of coordinate matrix is defined by requiring that

$$P_{\mathcal{C}\leftarrow\mathcal{B}}[\boldsymbol{x}]_{\mathcal{B}} = [\boldsymbol{x}]_{\mathcal{C}}. \tag{A.4}$$

It is straightforward to show that $P_{\mathcal{C}\leftarrow\mathcal{B}} = (P_{\mathcal{B}\leftarrow\mathcal{C}})^{-1}$, so that matrix inversion can be used to compute the change of coordinate matrix the opposite way. It is also straightforward to show that the columns in the change of coordinate matrix can be obtained by expressing the old basis in terms of the new basis, i.e. finding the vectors $[P_{\mathcal{B}\leftarrow\mathcal{C}}(\boldsymbol{v}_i)]_{\mathcal{C}}$.

If L is a linear transformation between the spaces V and W, and $\mathcal{B}$ is a basis for V, $\mathcal{C}$ a basis for W, we can consider the operation which sends the coordinates of $\boldsymbol{v} \in V$ in the basis $\mathcal{B}$ to the coordinates of $L\boldsymbol{v} \in W$ in the basis $\mathcal{C}$. This is represented by a matrix, called *the matrix of L relative to the bases $\mathcal{B}$ and $\mathcal{C}$*. Similarly to change of coordinate matrices, the columns of the matrix of L relative to the bases $\mathcal{B}$ and $\mathcal{C}$ are given by $[L(\boldsymbol{v}_i)]_{\mathcal{C}}$.

A.5 Eigenvectors and eigenvalues

If A is a linear transformation from a vector space to itself, a vector $\boldsymbol{v}$ is called an *eigenvector* if there exists a scalar λ so that $A\boldsymbol{v} = \lambda\boldsymbol{v}$. λ is called the corresponding eigenvalue.

If the matrix A is symmetric, the following hold:

- The eigenvalues of A are real,
- the eigenspaces of A are orthonormal,
- any vector can be decomposed as a sum of eigenvectors from A.

Fo non-symmetric matrices, these results do not hold in general. But for filters, clearly the second and third property always hold, regardless of whether the filter is symmetric or not.

A.6 Diagonalization

One can show that, for a symmetric matrix, $A = PDP^T$ where D is a digonal matrix and the eigenvalues of A are the values on the diagonal of D, and P is a matrix where the columns are the eigenvectors of A, with corresponding eigenvalue appearing in the same column in D.

Appendix B

Signal processing and linear algebra: a translation guide

This book should not be considered as a standard signal processing textbook. There are several reasons for this. First of all, much signal processing literature is written for people with an engineering background. This book is written for people with a basic linear algebra background. Secondly, the book does not give a comprehensive treatment of all basic signal processing concepts. Signal processing concepts are introduced whenever they are needed to encompass the mathematical exposition. In order to learn more about the different signal processing concepts, the reader can consult many excellent textbooks, such as [28, 1, 25, 32]. The translation guide of this chapter may be of some help in this respect, when one tries to unify material presented here with material from these signal processing textbooks. The translation guide handles both differences in notation between this book and signal processing literature, and topical differences. Most topical differences are also elaborated further in the summaries of the different chapters. The book has adopted most of its notation and concepts from mathematical literature.

B.1 Complex numbers

There are several differences between engineering literature and mathematics. In mathematics literature, i is used for the imaginary complex number which satisfies $i^2 = -1$. In engineering literature, the name j is used instead.

B.2 Functions

What in signal processing are refered to as continuous-time signals, are here refered to as functions. Usually we refer to a function by the letter f, according

to the mathematical tradition. The variable is mostly time, represented by the symbol t.

In signal processing, one often uses capital letters to denote a function which is the Fourier transform of another function, so that the Fourier ransform of x would be denoted by X. Here we simply denote a periodic function by its Fourier coefficients y_n, and we avoid the CTFT. We use analog filters, however, which also work in continuous time. Analog filters preserve frequencies, and we have used ν to denote frequency (variations per second), and not used angular frequency ω. In signal processing literature it is common to junp between the two.

B.3 Vectors

Discrete-time signals, as they are used in signal processing, are here mostly refered to as vectors. To as big extent as possible, we have attempted to keep vectors finite-dimensional. Vectors are in boldface (i.e. $\boldsymbol{x}$), but its elements are not in boldface, and with subscripts (i.e. x_n). Superscripts are also used to differ between vectors with the same base name (i.e. $\boldsymbol{x}^{(1)}$, $\boldsymbol{x}^{(2)}$ etc.), so that this does not interfer with the vector indices. In signal processing literature the corresponding notation would be x for the signal, and $x[n]$ for its elements, and signals with equal base names could be named like $x_1[n], x_2[n]$.

We have sometimes denoted the Fourier transform of $\boldsymbol{x}$ by $\widehat{\boldsymbol{x}}$, according to the mathematical tradition. More often we have distuinguished between a vector and its Discrete Fourier transform by using $\boldsymbol{x}$ for the first, and $\boldsymbol{y}$ for the latter. This also makes us distuinguish between the input and output to a filter, where we instead use $\boldsymbol{z}$ for the latter. Much signal processing literature write (capital) X for the DFT of the vector x.

B.4 Inner products and orthogonality

Throughout the book we have defined inner products for functions (for Fourier analysis and wavelets), and we have also used the standard inner product of $\mathbb{R}^N$. from this we have deduced the orthogonality of several basis functions used in signal processing theory. That the functions are orthogonal, as well as the inner product itself are, however, often not commented on in signal processing literature. As an unfortunate consequence, one has to explain the expression for the Fourier series using other means than the orthogonal decomposition formula and the least squares method. Also, one does not mention that the DFT is a unitary transformation.

B.5 Matrices and filters

Boldface notation is not used for matrices, according to the mathematical tradition. In signal processing, it is not common to formulate matrix equations,

such as for the DFT and DCT, or matrix factorizations. Instead one typically writes down each equation, one equation for each row in $\boldsymbol{y} = A\boldsymbol{x}$, i.e. not recognizing matrix/vector multiplication. We have sticked to the name filtering operations, but made it clear that this is nothing but a linear transformation with a Toeplitz matrix as its matrix. In particular, we alternately use the terms filtering and multiplication with a Toeplitz matrix. The characterization of filters as circulant Toeplitz matrices is usually not done in signal processing literature (but see [13]). In this text we allow for matrices also to be of infinite dimensions, expanding on the common use in linear algebra. When infinite dimensions are assumed, infinite in both directions is assumed. Matrices are scaled if necessary to make them unitary, in particular the DCT and the DFT. This scaling is usually not done in signal processing literature.

Representing a filter in terms of a finite matrix and restriction of a filter to a finite signal. This is usually omitted in signal processing literature.

One of the most important statements in signal processing is that convolution in time is equivalent to multiplication in frequency. We have presented a compelling interpretation of this in linear algebra terms. Since the frequency response simply are eigenvalues of the filter, and convolution simply is matrix factorization, multiplication in frequency simply means to multiply two diagonal matrices to obtain the frequency response of the product. Moreover, the Fourier basis vectors can be interpreted as eigenvectors.

B.6 Convolution

While we have defined the concept of convolution, readers familiar with signal processing may have noticed that this concept has not been used much. The reason is that we have wanted to present convolution as a matrix multiplication (to adapt to mathematical tradition), and that we have used the concept of filtering often instead. In signal processing literature one defines convolution in terms of vectors of infinite length. We have avoided this, since in practice vectors always need to be truncated to finite lengths. Due to this, we also have analyzed how a finite vector may be turned into a periodic vector (periodic or symmetric extension), and how this affects our analysis. Also we have concentrated on FIR-filters, and this makes us avoid convergence issues. Note that we do not present matrix multiplication as a method of implementing filtering, due to the special structure of this operation. We do not suggest other methods for implementation than applying the convolution formula in a brute-force way, or factoring the filter in simpler components.

B.7 Polyphase factorizations and lifting

In signal processing literature, it is not common to associate polyphase components with matrices, but rather with Laurent polynomials generated from the corresponding filter. The Laurent polynomial is nothing else than the Z-

transform of the associated filter. Associating polyphase components with blocks in a block matrix makes this book fit with block matrix methods in linear algebra textbooks.

The polyphase factorization serves two purposes in this book. Firstly, the lifting factorization (as used for wavelets) is derived from it, and put in a linear algebra framework as a factorization into sparse matrices, similarly to the FFT factorization. Thereby it fits together with many of the matrix factorization results from classical linear algebra, where also sparsity is what makes the factorization good for computation.

Secondly, the polyphase factorization of the filter bank transforms in the MP3 standard are derived (also as a sparse matrix factorization), and from this it is apparent what properties to put on the prototype filters in order to obtain useful transforms. In fact, from this factorization it became apparent that the MP3 filter bank transforms could be expressed in terms of alternative QMF filter banks (i.e. $M = 2$).

These two topics (lifting and the MP3 filter bank transform polyphase factorization) are usually not presented in a unified way in textbooks. we see here that there is a big advantage of doing this, since the second can build on theory from the first.

B.8 Transforms in general

In signal processing, one often refers to the forward and reverse filter bank transforms as analysis and synthesis, respectively, and for obvious reasons. In mathematical literature, one instead often use the term change of coordinates in a wavelet setting. These terms are not normally used in mathematical literature, where the term basis vectors/change of coordinate matrices would be used instead. Also, the output from a forward filter bank transform is often refered to as the *transformed vector*, and the result we get when we apply the reverse filter bank transform to this is called the *reconstructed vector*.

This exposition takes extra care in presenting how the DCT is derived naturally from the DFT. In particular both the DFT and the DCT are derived as matrices of eigenvectors for finite-dimensional filters. The DCT is derived from the DFT in that one restricts to a certain subset of vectors. The orthogonality of these matrices follows from the orthogonality of distinct eigenspaces.

B.9 Perfect reconstruction systems

The term biorthogonality is not used to describe a mutual property of the filters of wavelets. Borthogonality corresponds simply to two matrices being inverses of oneanother. For the same reason, the term perfect reconstruction is not used much. Much wavelet theory refer to a property called delay normalization. This terms has been avoided by mostly considering wavelets with symmetric filters,

for which delay-normalization is automatic. There are, however, many examples of wavelets where this term is important.

B.10 Z-transform and frequency response

The Z-transform and the frequency response are much used in signal processing literaure, and are important concepts for filter design. We have deliberately dropped the Z-transform. Due to this, much signal processing has of course been left out, since placements of poles and zeroes are not performed outside or inside the unit circle, since the frequency response only captures the values on the unit circle. Placement of poles and circles is perhaps the most-used design feature in filter design. The focus here is on implementing filters, not designing them, however.

In signal processing literature, the DTFT and the Z-transform is used, assuming that the inputs and outputs are vectors of infinite length. In practice of course, some truncation is needed, since only finite-dimensional arithmetic is performed by the computer. How this truncation is to be done without affecting the computations is thus never mentioned in signal processing, although it is always performed somehow. This exposition shows that this truncation can be taken as part of the theory, without seriously affecting the results.

Nomenclature

symbol	definition
T	Period of a function
ν	Frequency
f_N	Nth order Fourier series of f
$V_{N,T}$	Nth order Fourier space
$\mathcal{D}_{N,T}$	Order N real Fourier basis for $V_{N,T}$
$\mathcal{F}_{N,T}$	Order N complex Fourier basis for $V_{N,T}$
$\check{f}$	Symmetric extension of the function f
$\lambda_s(\nu)$	Frequency response of an analog filter
f_s	Sampling frequency
T_s	Sampling period
N	Number of points in a DFT/DCT
$\mathcal{F}_N = \{\boldsymbol{\phi}_0, \boldsymbol{\phi}_1, \cdots, \boldsymbol{\phi}_{N-1}\}$	Fourier basis for $\mathbb{R}^N$
F_N	$NimesN$-Fourier matrix
$\hat{\boldsymbol{x}}$	DFT of the vector $\boldsymbol{x}$
$\overline{A}$	Conjugate of a matrix
A^H	Conjugate transpose of a matrix
$\boldsymbol{x}^{(e)}$	Vector of even samples
$\boldsymbol{x}^{(o)}$	Vector of odd samples
$O(N)$	Order of an algorithm
$l(S)$	Length of a filter
$\boldsymbol{x} * \boldsymbol{y}$	Convolution of vectors
$\lambda_{S,n}$	Vector frequency response of a digital filter
E_d	Filter which delays with d samples
ω	Angular frequency
$\lambda_S(\omega)$	Continuous frequency response of a digital filter
$\check{\boldsymbol{x}}$	Symmetric extension of a vector
S_r	Symmetric restriction of S
S^f	Matrix with the columns reversed
$\mathcal{D}_N = \{\boldsymbol{d}_0, \boldsymbol{d}_1, \cdots, \boldsymbol{d}_{N-1}\}$	N-point DCT basis for $\mathbb{R}^N$
DCT_N	$N \times N$-DCT matrix

symbol	definition
ϕ	Scaling function
V_m	Resolution space
r$\boldsymbol{\phi}_m$	Basis for V_m
$c_{m,n}$	Coordinates in $\boldsymbol{\phi}_m$
W_m	Detail space
r$U \oplus V$	Direct sum of vector spaces
$\boldsymbol{\psi}_m$	Basis for W_m
$w_{m,n}$	Coordinates in $\boldsymbol{\psi}_m$
$\mathcal{C}_m$	Reordering of $(\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1})$
$\tilde{\phi}$	Dual scaling function
$\tilde{\psi}$	Dual mother wavelet
$\tilde{V}_m$	Dual resolution space
$\tilde{W}_m$	Dual detail space
$\mathcal{D}_m$	Reordering of $\boldsymbol{\phi}_m$
$\mathcal{E}_N = \{\boldsymbol{e}_0, \boldsymbol{e}_1, \cdots, \boldsymbol{e}_{N-1}\}$	Standard basis for $\mathbb{R}^N$
$\otimes$	Tensor product
$W_m^{(0,1)}$	Resolution m Complementary wavelet space, LH
$W_m^{(1,0)}$	Resolution m Complementary wavelet space, HL
$W_m^{(1,1)}$	Resolution m Complementary wavelet space, HH
A^T	Transpose of a matrix
A^{-1}	Inverse of a matrix
$\langle \boldsymbol{u}, \boldsymbol{v} \rangle$	Inner product
$[\boldsymbol{x}]_{\mathcal{B}}$	Coordinate vector of $\boldsymbol{x}$ relative to the basis $\mathcal{B}$
$P_{\mathcal{C} \leftarrow \mathcal{B}}$	Change of coordinate matrix from $\mathcal{B}$ to $\mathcal{C}$

Bibliography

[1] A. Ambardar. *Digital Signal Processing: a Modern Introduction*. Cengage Learning, 2006.

[2] C. M. Brislawn. Fingerprints go digital. *Notices of the AMS*, 42(11):1278–1283, 1995.

[3] B. A. Cipra. The best of the 20th century: Editors name top 10 algorithms. *SIAM News*, 33(4), 2000. http://www.uta.edu/faculty/rcli/TopTen/topten.pdf.

[4] A. Cohen and I. Daubechies. Wavelets on the interval and fast wavelet transforms. *Applied and computational harmonic analysis*, 1:54–81, 1993.

[5] A. Cohen, I. Daubechies, and J-C. Feauveau. Biorthogonal bases of compactly supported wavelets. *Communications on Pure and Appl. Math.*, 45(5):485–560, June 1992.

[6] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex fourier series. *Math. Comp.*, 19:297–301, 1965.

[7] A. Croisier, D. Esteban, and C. Galand. Perfect channel splitting by use of interpolation/decimation/tree decomposition techniques. *Int. Conf. on Information Sciences and Systems*, pages 443–446, August 1976.

[8] I. Daubechies. Orthonormal bases of compactly supported wavelets. *Communications on Pure and Appl. Math.*, 41(7):909–996, October 1988.

[9] I. Daubechies. *Ten Lectures on Wavelets*. CBMS-NSF conference series in applied mathematics. SIAM Ed., 1992.

[10] P. Duhamel and H. Hollmann. 'split-radix' FFT-algorithm. *Electronic letters*, 20(1):14–16, 1984.

[11] FBI. WSQ gray-scale fingerprint image compression specification. Technical report, IAFIS-IC, 1993.

[12] G. B. Folland. *Real Analysis. Modern Techniques and Their Applications*. John Wiley and sons, 1984.

[13] M. W. Frazier. *An Introduction to Wavelets Through Linear Algebra.* Springer, 1999.

[14] M. Frigo and S. G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005.

[15] R. C. Gonzalez, R. E. Woods, and S. L. Eddins. *Digital Image Processing Using MATLAB.* Gatesmark publishing, 2009.

[16] ISI/IEC. Information technology - coding of moving pictures and associated audio for digital storage media at up to about 1.5 mbit/s. Technical report, ISO/IEC, 1993.

[17] ISO/IEC. Jpeg2000 part 1 final draft international standard. iso/iec fdis 15444-1. Technical report, ISO/IEC, 2000.

[18] S.G Johnson and M. Frigo. A modified split-radix FFT with fewer arithmetic operations. *IEEE Transactions on Signal Processing*, 54, 2006.

[19] J.D. Johnston. A filter family designed for use in quadrature mirror filter banks. *Proc. Int. Conf. Acoust. Speech and Sig. Proc.*, pages 291–294, 1980.

[20] D. C. Lay. *Linear Algebra and Its Applications (4th Edition).* Addison-Wesley, 2011.

[21] S. Mallat. *A Wavelet Tour of Signal Processing.* Tapir Academic Press, 1998.

[22] C. D. Meyer. *Matrix Analysis and Applied Linear Algebra.* SIAM, 2000.

[23] Knut Mørken. *Numerical Algorithms and Digital Representation.* UIO, 2013.

[24] P. Noll. MPEG digital audio coding. *IEEE Signal processing magazine*, pages 59–81, September 1997.

[25] A. V. Oppenheim and R. W. Schafer. *Discrete-Time Signal Processing.* Prentice Hall, 1989.

[26] D. Pan. A tutorial on MPEG/audio compression. *IEEE Multimedia*, pages 60–74, Summer 1995.

[27] W. B. Pennebaker and J. L. Mitchell. *JPEG Still Image Data Compression Standard.* Van Nostrand Reihnold, 1993.

[28] J. G. Proakis and D. G. Manolakis. *Digital Signal Processing. Principles, Algorithms, and Applications. Fourth Edition.* Pearson, 2007.

[29] C. M. Rader. Discrete Fourier transforms when the number of data samples is prime. *Proceedings of the IEEE*, 56:1107–1108, June 1968.

[30] T. A. Ramstad, S. O. Aase, and J. H. Husøy. *Subband Compression of Images: Principles and Examples: Principles and Examples*, volume 6. Elsevier Science, 1995.

[31] C. E. Shannon. Communication in the presence of noise. *Proc. Institute of Radio Engineers*, 37(1):10–21, Jan. 1949.

[32] P. Stoica and R. Moses. *Spectral Analysis of Signals*. Prentice Hall, 2005.

[33] G. Strang and T. Nguyen. *Wavelets and Filter Banks*. Wellesley - Cambridge Press, 1996.

[34] W. Sweldens. The lifting scheme: a new philosophy in biorthogonal wavelet constructions. *Wavelet Applications in Signal and Image Processing III*, pages 68–79, 1995.

[35] W. Sweldens. The lifting scheme: a custom-design construction of biorthogonal wavelets. *Applied and computational harmonic analysis*, 3:186–200, 1996.

[36] D. S. Taubman and M. W. Marcellin. *JPEG2000. Image Compression. Fundamentals, Standards and Practice.* Kluwer Academic Publishers, 2002.

[37] M. Vetterli and J. Kovacevic. *Wavelets and Subband Coding*. Prentice Hall, 1995.

[38] M. Vetterli and H. J. Nussbaumer. Simple FFT and DCT algorithms with reduced number of operations. *Signal Processing*, 6:267–278, 1984.

[39] S. Winograd. On computing the discrete Fourier transform. *Math. Comp.*, 32:175–199, 1978.

[40] R. Yavne. An economical method for calculating the discrete Fourier transform. *Proc. AFIPS Fall Joint Computer Conf.*, 33:115–125, 1968.

Index