Part II

Wavelets and applications to image processing

Chapter 5

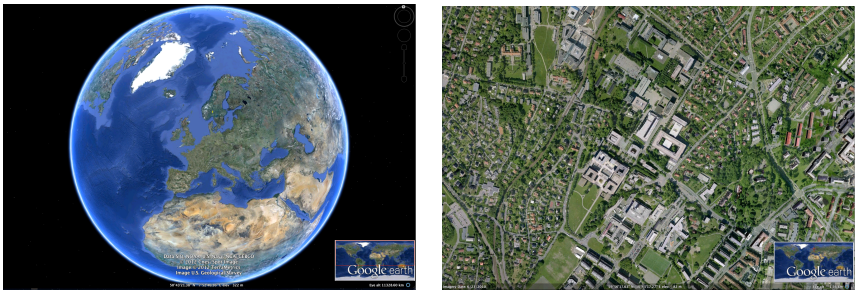
Wavelets

In Part I on Fourier analysis our focus was to approximate periodic functions or vectors in terms of trigonometric functions. We saw that the Discrete Fourier transform could be used to obtain the representation of a vector in terms of such functions, and the computations could be done efficiently with the FFT algorithm. This was useful for analyzing, filtering, and compression of sound and other discrete data. However, Fourier series and the DFT also have some serious limitations:

1. First of all, the functions used in the approximation are periodic with short periods. In contrast, for most functions encountered in applications the frequency content changes with time. Although Fourier analysis tools also exist for analyzing non-periodic functions, these tools mostly have a theoretical significance and are rarely used in practice, because of lack of efficient implementations.

2. Secondly, all components of the Fourier basis vectors are nonzero — in fact they all have absolute value 1 at all points. This means that, in order to compute a value using the representation in the Fourier basis, we must for each instance in time sum over all N vectors in the basis. This is time-consuming when N is large.

In this chapter we are going to introduce the basic properties of an alternative to Fourier analysis, namely wavelets. Similar to Fourier analysis, wavelets are also based on the idea of transforming a function to a different basis. But in contrast to Fourier analysis, where the basis is fixed, wavelets provide a general framework with many different types of bases. In this chapter, we introduce the framework via the simplest wavelets. We then discuss some general wavelet concepts before we consider a second example.



(a) (b)

Figure 5.1: A view of Earth from space (a) and a zoomed in view (b).

5.1 Why wavelets?

Figure 5.1 shows two views of the Earth. The one on the left is the startup image in Google Earth, a program for viewing satellite images, maps and other geographic information. The right image is a zoomed-in view of a small part of the Earth. There is clearly an amazing amount of information available behind a program like Google Earth, with images detailed enough to differentiate between buildings and even trees or cars all over the Earth. So when the Earth is spinning in the opening screen, all the Earth's buildings appear to be spinning with it! If this was the case, the Earth would not be spinning on the screen. There would just be so much information to process that a laptop would not be able to display a rotating Earth.

There is a simple reason that the globe can be shown spinning in spite of the huge amounts of information that need to be handled. We are going to see later that a digital image is just a rectangular array of numbers that represent the colour at a dense set of points. As an example, the images in Figure 5.1 are both made up of a grid with 1576 points in the horizontal direction and 1076 points in the vertical direction, for a total of 1 695 776 points. The colour at a point is represented by three eight-bit integers, which means that the image file contains a total of 5 087 328 bytes. So regardless of how close to the surface of the Earth our viewpoint is, the resulting image always contains the same number of points. This means that when we are far away from the Earth we can use a very coarse model of the geographic information that is being displayed, but as we zoom in, we need to display more details and therefore need a more accurate model.

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

A consequence of Observation 5.1 is that for applications like Google Earth we should use a mathematical model that makes it easy to switch between

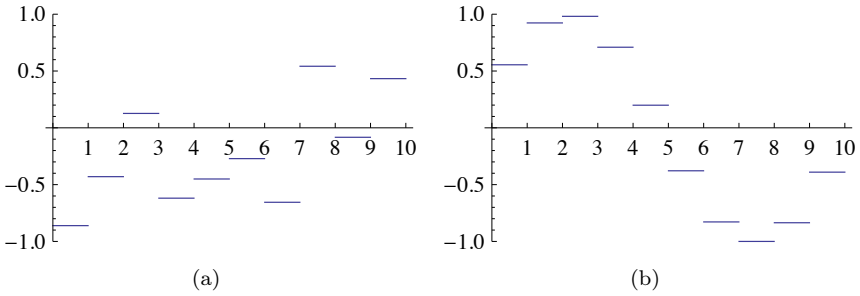


Figure 5.2: Two examples of piecewise constant functions.

different levels of detail, or different resolutions. Such models are called *multiresolution models*, and wavelets are prominent examples of this kind of models.

5.2 Wavelets constructed from piecewise constant functions

There are many different kinds of wavelets that all share certain standard properties. In this section we will introduce the simplest wavelets and through this also the general framework for constructing wavelets. The construction goes in two steps: First we introduce the resolution spaces, and then the detail spaces and wavelets.

5.2.1 Resolution spaces

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

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

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

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

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

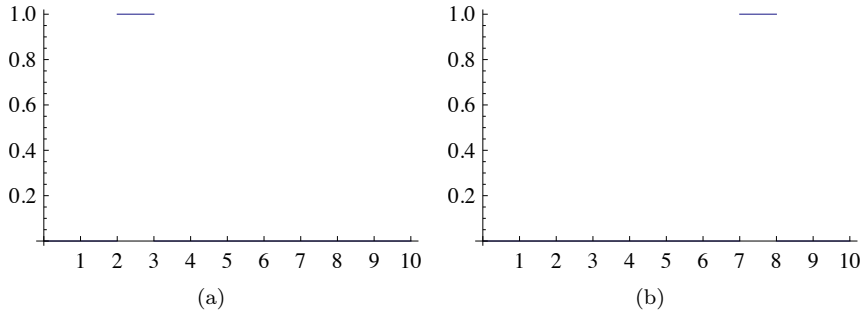


Figure 5.3: The functions ϕ_2 (a) and ϕ_7 (b) in V_0.

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

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

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

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

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

Two examples of the basis functions defined in Lemma 5.5 are shown in Figure 5.3.

Proof. Two functions ϕ_{n_1} and ϕ_{n_2} with $n_1 \neq n_2$ clearly satisfy $\int \phi_{n_1}(t)\phi_{n_2}(t)dt = 0$ since $\phi_{n_1}(t)\phi_{n_2}(t) = 0$ for all values of x. It is also easy to check that $\|\phi_n\| = 1$ for all n. Finally, any function in V_0 can be written as a linear combination the functions ϕ_0, ϕ_1, …, ϕ_{N-1}, so the conclusion of the lemma follows. □

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

In Fourier analysis we obtained a linear space of possible approximations by including sines of frequency 1, 2, 3, …, up to some maximum. We use a similar

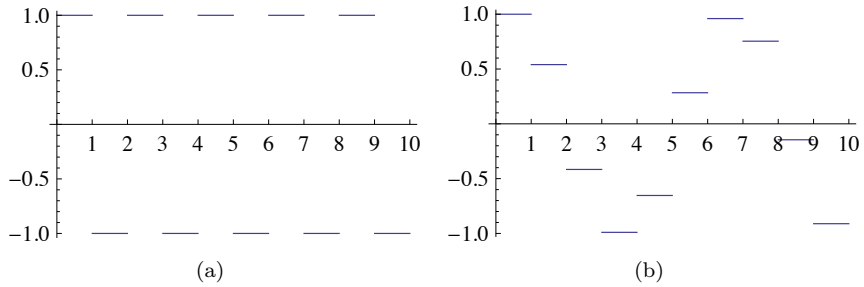


Figure 5.4: The square wave in V_0 (a) and an approximation to $\cos t$ from V_0.

approach for constructing wavelets, but we double the frequency each time and label the spaces as V_0, V_1, V_2, ...

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

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

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

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

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

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

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

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

Proof. The functions given in (5.25) are exactly the characteristic functions of the subintervals $[n/2^m, (n+1)/2^m)$ which we referred to in Definition 5.4, so the proof is very similar to the proof of Lemma 5.5. The one mysterious thing

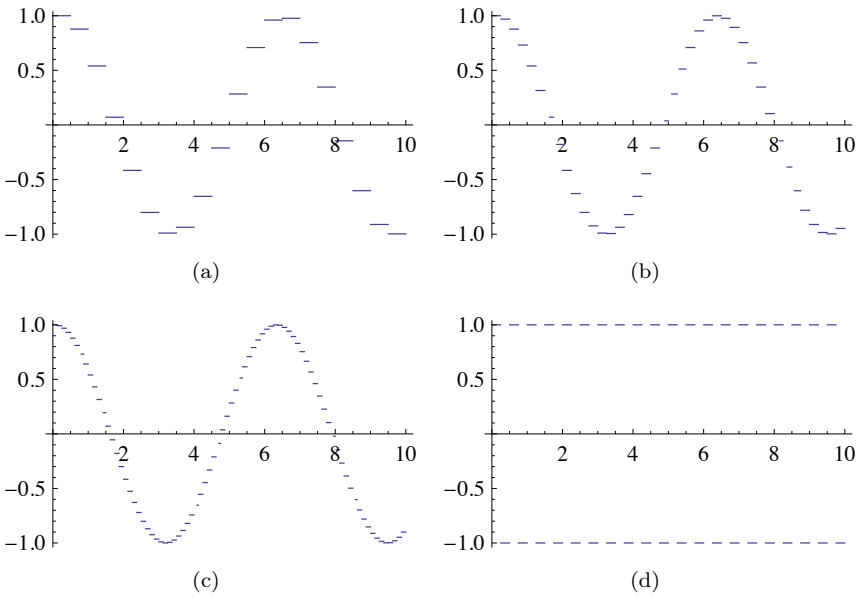


Figure 5.5: Piecewise constant approximations to $\cos t$ on the interval $[0, 10]$ in the spaces V_1 (a), V_2 (b), and V_3 (c). The plot in (d) shows the square wave in V_2.

may be the normalisation factor $2^{-m/2}$. This comes from the fact that

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

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

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

5.2.2 Function approximation property

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

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

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

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

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

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

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

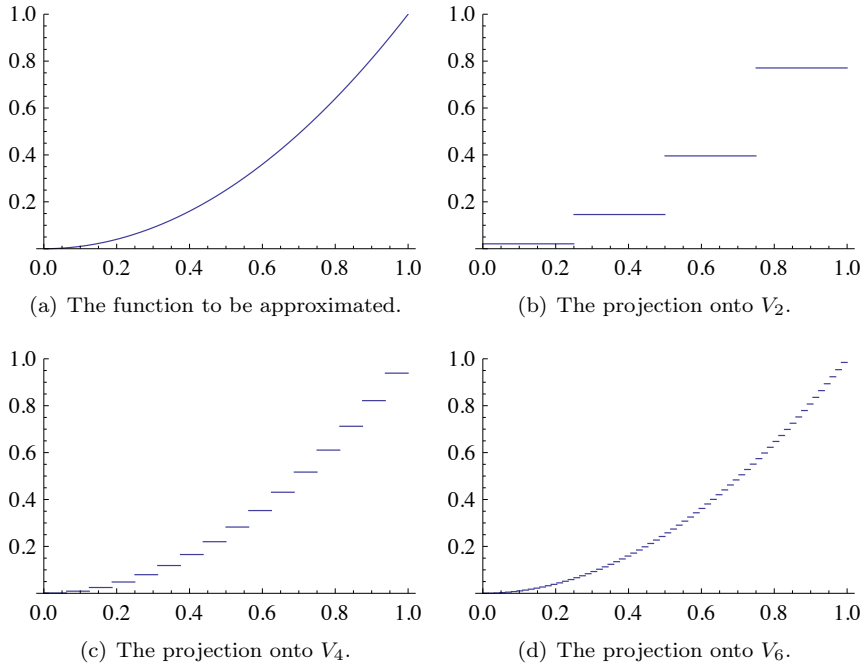
$$t_{m,n+1/2} = (n + 1/2)2^{-m}.$$

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

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

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

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



(a) The function to be approximated.

(b) The projection onto V_2.

(c) The projection onto V_4.

(d) The projection onto V_6.

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

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

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

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

5.2.3 Detail spaces and wavelets

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

We start by observing that since

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

we have

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

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

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

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

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

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

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

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

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

If $g_1 \in V_1$ is given by

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

then

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

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

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

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

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

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

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

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

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

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

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

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

which proves (5.8) □

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

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

The name detail space is used since $e_0 \in W_0$ can be considered as the detail which is left out when considering g_0 instead of g_1 (due to the expression $g_1 = g_0 + e_0$). We will write $V_1 = V_0 \oplus W_0$ to say that any element in V_1 can be written uniquely as a sum of an element in V_0, and an element in the orthogonal complement W_0. $\oplus$ here denotes what is called a direct sum, which can be more generally defined as follows for any vector spaces which are linearly independent:

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

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

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

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

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

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

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

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

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

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

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

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

onto V_0. The error is then

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

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

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

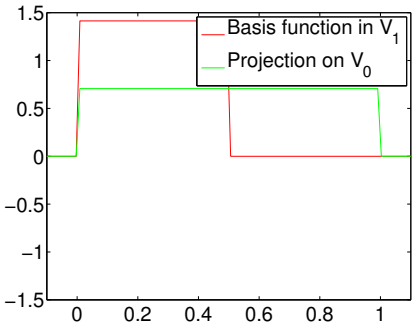
For a general function g_1 we first split the sum into even and odd terms as in (5.10) and then project each part onto W_0,

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

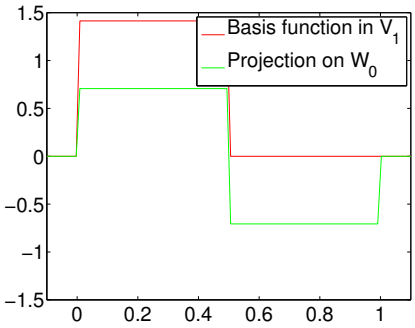
which is (5.12) □

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

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



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



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

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

Moreover, $\text{proj}_{W_0}(f)$ is the function in W_0 which is $(f_{n,1} - f_{n,2})/2$ on $[n, n + 1/2)$, and $-(f_{n,1} - f_{n,2})/2$ on $[n + 1/2, n + 1)$.

In other words, the projection on V_0 is constructed by averaging on two subintervals, while the projection on W_0 is constructed by taking the difference from the mean. This sounds like a reasonable candidate for the least-squares approximations. In the exercise we generalize these observations.

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

Lemma 5.14. Define the function ψ by

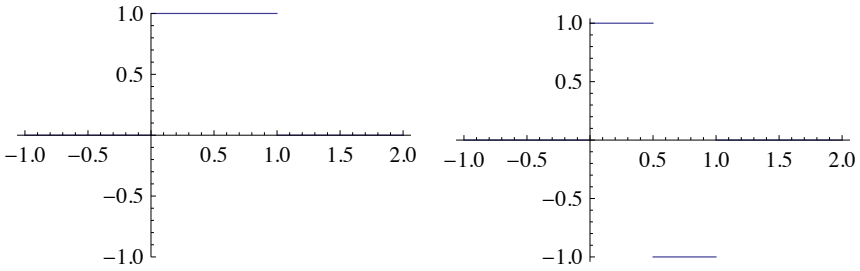
$$\psi(t) = \big(\phi_{1,0}(t) - \phi_{1,1}(t)\big)/\sqrt{2} = \phi(2t) - \phi(2t-1) \tag{5.13}$$

and set

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

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

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



(a) The function ϕ (a) (b) The function ψ

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

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

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

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

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

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

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

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

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

$$c_{0,n} = (c_{1,2n} + c_{1,2n+1})/\sqrt{2} \tag{5.15}$$
$$w_{0,n} = (c_{1,2n} - c_{1,2n+1})/\sqrt{2}. \tag{5.16}$$

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

$$c_{1,2n} = (c_{0,n} + w_{0,n})/\sqrt{2} \tag{5.17}$$

$$c_{1,2n+1} = (c_{0,n} - w_{0,n})/\sqrt{2}. \tag{5.18}$$

Proof. Most of this theorem has already been established. In particular, the formulas (5.15)–(5.16) are just (5.8) and (5.12). What remains is to prove the formulas (5.17)–(5.18). For this we note from (5.5) and (5.14) that

$$g_0 + e_0 = \sum_{n=0}^{N-1} c_{0,n}\phi_{0,n} + \sum_{n=0}^{N-1} w_{0,n}\psi_{0,n} \tag{5.19}$$

$$= \sum_{n=0}^{N-1} c_{0,n}(\phi_{1,2n} + \phi_{1,2n+1})/\sqrt{2} + \sum_{n=0}^{N-1} w_{0,n}(\phi_{1,2n} - \phi_{1,2n+1})/\sqrt{2} \tag{5.20}$$

$$= \sum_{n=0}^{N-1} (c_{0,n} + w_{0,n})\phi_{1,2n}/\sqrt{2} + (c_{0,n} - w_{0,n})\phi_{1,2n+1}/\sqrt{2}. \tag{5.21}$$

□

It is common to reorder the basis vectors in $(\boldsymbol{\phi}_0, \boldsymbol{\psi}_0)$ to

$$\mathcal{C}_1 = \{\phi_{0,0}, \psi_{0,0}, \phi_{0,1}, \psi_{0,1}, \cdots, \phi_{0,N-1}, \psi_{0,N-1}\}. \tag{5.22}$$

The subscript 1 is used since $\mathcal{C}_1$ is a basis for V_1. This reordering of the basis functions is useful since it makes it easier to write down the change of coordinates matrices. To be more precise, from formulas (5.17)–(5.18) it is apparent that $P_{\boldsymbol{\phi}_1 \leftarrow \mathcal{C}_1}$ is the matrix where

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

is repeated along the main diagonal N times. Also, from formulas (5.15)–(5.16) it is apparent that $P_{\mathcal{C}_1 \leftarrow \boldsymbol{\phi}_1}$ is the same matrix. Such matrices are called *block diagonal matrices*. This particular block diagonal matrix is clearly orthogonal, since it transforms one orthonormal base to another.

Exercises for Section 5.2

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

Ex. 2 — Show that

$$\mathrm{proj}_{V_0}(f) = \sum_{n=0}^{N-1} \left(\int_n^{n+1} f(t)dt \right) \phi_{0,n}(t) \tag{5.23}$$

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

Ex. 3 — Show that

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

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

Ex. 4 — Consider the projection T of V_1 onto V_0.

a. Show that $T(\phi) = \phi$ and $T(\psi) = 0$.
b. Show that the matrix of T relative to $\mathcal{C}_1$ is given by the diagonal matrix where 1 and 0 are repeated alternatingly on the diagonal, N times (i.e. 1 at the even indices, 0 at the odd indices).
c. Show in a similar way that the projection of V_1 onto W_0 has a matrix relative to $\mathcal{C}_1$ given by the diagonal matrix where 1 and 0 also are repeated alternatingly on the diagonal, but with the opposite order.

Ex. 5 — Use lemma 5.9 to write down the matrix for the linear transformation $\mathrm{proj}_{V_0} : V_1 \to V_0$ relative to the bases $\boldsymbol{\phi}_1$ and $\boldsymbol{\phi}_0$. Also, use lemma 5.12 to write down the matrix for the linear transformation $\mathrm{proj}_{W_0} : V_1 \to W_0$ relative to the bases $\boldsymbol{\phi}_1$ and $\boldsymbol{\psi}_0$.

Ex. 6 — Show that

$$\mathrm{proj}_{W_0}(f) = \sum_{n=0}^{N-1} \left(\int_n^{n+1/2} f(t)dt - \int_{n+1/2}^{n+1} f(t)dt \right) \psi_{0,n}(t) \tag{5.24}$$

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

5.3 Multiresolution analysis for piecewise constant functions

In the Section 5.2 we introduced the important decomposition $V_1 = V_0 \oplus W_0$ which lets us rewrite a function in V_1 as an approximation in V_0 and the corresponding error in W_0 which is orthogonal to the approximation. The resolution spaces V_m were in fact defined for all integers $m \geq 0$. It turns out that all these resolution spaces can be decomposed in the same way as V_1.

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

The first question we will try to answer is how we can, for $f \in V_m$, extract the corresponding detail in W_{m-1}.

5.3.1 Extraction of details at higher resolutions

We first need to define $\psi_{m,n}$ in terms of ψ, similarly to how we defined $\phi_{m,n}$ in terms of ϕ,

$$\psi_{m,n}(t) = 2^{m/2}\psi(2^m t - n), \quad \text{for } n = 0, 1, \ldots, 2^m N - 1. \tag{5.25}$$

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

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

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

and

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

If

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

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

$$c_{m-1,n} = (c_{m,2n} + c_{m,2n+1})/\sqrt{2}, \tag{5.26}$$

$$w_{m-1,n} = (c_{m,2n} - c_{m,2n+1})/\sqrt{2}. \tag{5.27}$$

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

$$c_{m,2n} = (c_{m-1,n} + w_{m-1,n})/\sqrt{2}, \tag{5.28}$$

$$c_{m,2n+1} = (c_{m-1,n} - w_{m-1,n})/\sqrt{2}. \tag{5.29}$$

We will omit the proof of Theorem 5.18, and only remark that it can be proved by making the substitution $t \to 2^m u$ in Lemma 5.9 and Lemma 5.12, and then following the proof of Theorem 5.16. Clearly, we can now find the change of coordinate matrices as before, and as before this is most easily expressed if we reorder the basis vectors for $(\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1})$ again as in Equation (5.22), i.e. we define

$$\mathcal{C}_m = \{\phi_{m-1,0}, \psi_{m-1,0}, \phi_{m-1,1}, \psi_{m-1,1}, \cdots, \phi_{m-1,2^{m-1}N-1}, \psi_{m-1,2^{m-1}N-1}\}. \tag{5.30}$$

The bases $\boldsymbol{\phi}_m$ and $\mathcal{C}_m$ are both referred to as *wavelet bases*. It is now apparent that both change of coordinates matrices $P_{\boldsymbol{\phi}_m \leftarrow \mathcal{C}_m}$, $P_{\mathcal{C}_m \leftarrow \boldsymbol{\phi}_m}$ can be obtained by repeating the matrix

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

along the diagonal, but this time it is repeated $2^{m-1}N$ times. In mathematical statements in the following, we will always express a change of coordinates in terms of the wavelet bases $\boldsymbol{\phi}_m$ and $\mathcal{C}_m$, due to the nice expression this matrix then has. In implementations, however, we also need to reorder $\mathcal{C}_m$ to $(\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1})$, in order to prepare for successive changes of coordinates, as we will now describe.

Let us return to our interpretation of the Discrete Wavelet Transform as writing a function $g_1 \in V_1$ as a sum of a function $g_0 \in V_0$ at low resolution, and a detail function $e_0 \in W_0$. Theorem 5.18 states similarly how we can write $g_m \in V_m$ as a sum of a function $g_{m-1} \in V_{m-1}$ at lower resolution, and a detail function $e_{m-1} \in W_{m-1}$. The same decomposition can of course be applied to g_{m-1} in V_{m-1}, then to the resulting approximation g_{m-2} in V_{m-2}, and so on,

$$\begin{aligned} V_m &= V_{m-1} \oplus W_{m-1} \\ &= V_{m-2} \oplus W_{m-2} \oplus W_{m-1} \\ &\vdots \\ &= V_0 \oplus W_0 \oplus W_1 \oplus \cdots \oplus W_{m-2} \oplus W_{m-1}. \end{aligned} \tag{5.31}$$

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

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

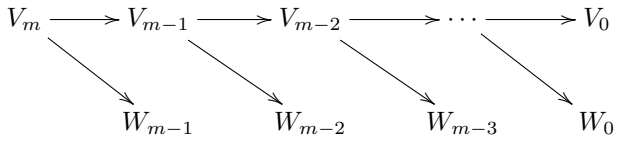
When we above say "basis for $V_0 \oplus W_0 \oplus W_1 \oplus \cdots \oplus W_{m-2} \oplus W_{m-1}$", we really mean the canonical basis for this space. In general, the Discrete Wavelet Transform is used to denote a change of coordinates from ϕ_m to the canonical basis, for any m.

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

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

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

Clearly, this generalizes the Discrete Wavelet Transform defined in Section 5.2. At each level in a DWT, V_k is split into one part from V_{k-1}, and one part from W_{k-1}. We can visualize this with the following figure, where the arrows represent changes of coordinates:



The part from W_{k-1} is not subject to further transformation. This is seen in the figure since W_{m-1} is a leaf node, i.e. there are no arrows going out from W_{m-1}. In a similar illustration for the IDWT, the arrows would go the opposite way. The Discrete Wavelet Transform is the analogue in a wavelet setting to the Discrete Fourier transform. When applying the DFT to a vector of length N, one starts by viewing this vector as coordinates relative to the standard basis. When applying the DWT to a vector of length N, one instead views the vector as coordinates relative to the basis $\boldsymbol{\phi}_m$. This makes sense in light of Exercise 5.2.1.

The DWT is what is used in practice when transforming a signal using wavelets, and it is straightforward to implement: One simply needs to iterate(5.26)-(5.27) for $m, m-1, \ldots, 1$, also at each step, the coordinates in $\boldsymbol{\phi}_{m-1}$ should be placed before the ones in $\boldsymbol{\psi}_{m-1}$, due to the order of the basis vectors in the canonical basis of the direct sum. At each step, only the first coordinates are further transformed. The following function, called `DWTHaarImpl`, follows this procedure. It takes as input the number of levels m, as well as the input vector $\boldsymbol{x}$, runs the m-level DWT on $\boldsymbol{x}$, and returns the result:

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

Note that this implementation is not recursive, contrary to the FFT. The for-loop here runs through the different resolutions. Inside the loop we perform the change of coordinates from $\boldsymbol{\phi}_k$ to $(\boldsymbol{\phi}_{k-1}, \boldsymbol{\psi}_{k-1})$ by applying equations (5.26)-(5.27). This works on the first coordinates, since the coordinates from $\boldsymbol{\phi}_k$ are stored first in

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

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

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

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

Here the steps are simply performed in the reverse order, and by iterating equations (5.28)-(5.29).

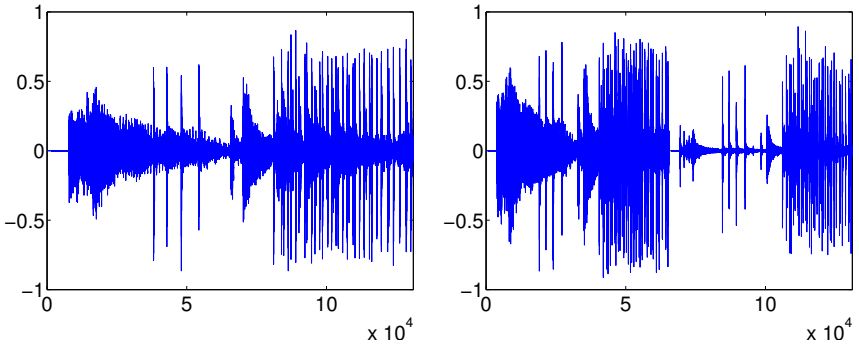
You may be puzzled by the names `DWTHaarImpl` and `IDWTHaarImpl`. In the next sections we will consider other cases, where the underlying function ϕ may be a different function, not necessarily piecewise constant. It will turn out that much of the analysis we have done makes sense for other functions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

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

It is of interest to plot the samples of our test audio file `castanets.wav`, and compare it with the first order DWT coefficients of the same samples. This is shown in Figure 5.9. The first part half of the plot represents the low-resolution approximation of the sound, the second part represents the detail/error. We see that the detail is quite significant in this case. This means that the first order wavelet approximation does not give a very good approximation to the sound. In the exercises we will experiment more on this.

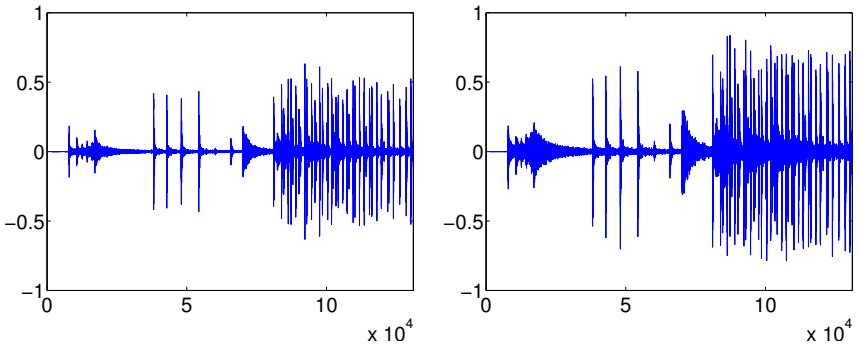
It is also interesting to plot only the detail/error in the sound, for different resolutions. For this, we must perform a DWT so that we get a representation in the basis $V_0 \oplus W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}$, set the coefficicents from V_0 to sero, and transform back with the IDWT.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

The previous example illustrates that wavelets as well may be used to perform operations on sound. As we will see later, however, our main application for wavelets will be images, where they have found a more important role than for sound. Images typically display variations which are less abrupt than the ones found in sound. Just as the functions above had smaller errors in the corresponding resolution spaces than the sound had, images are thus more suited for for use with wavelets. The main idea behind why wavelets are so useful comes



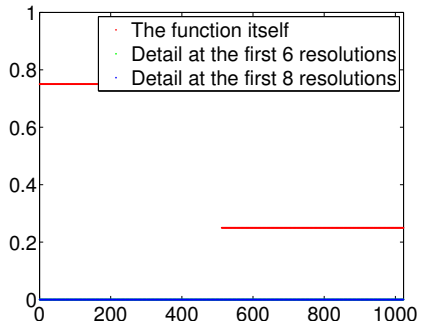
(a) The 2^{17} first sound samples (b) The first order DWT coefficients

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

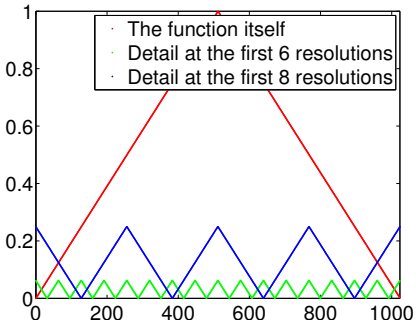


(a) $m = 1$ (b) $m = 2$

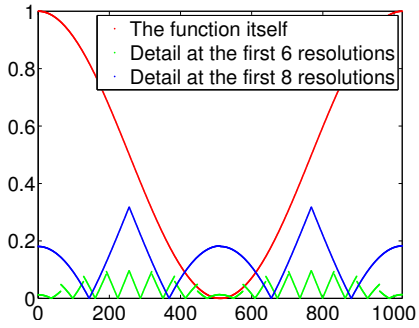
Figure 5.10: The error (i.e. the contribution from $W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}$) in the sound file `castanets.wav`, for different values of m.



(a) A square wave



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



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

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

from the fact that the detail, i.e., wavelet coefficients corresponding to the spaces W_k, are often very small. After a DWT one is therefore often left with a couple of significant coefficients, while most of the coefficients are small. The approximation from V_0 can be viewed as a good approximation, even though it contains much less information. This gives another reason why wavelets are popular for images: Detailed images can be very large, but when they are downloaded to a web browser, the browser can very early show a low-resolution of the image, while waiting for the rest of the details in the image to be downloaded. When we later look at how wavelets are applied to images, we will need to handle one final hurdle, namely that images are two-dimensional.

Example 5.22. Above we plotted the DWT coefficients of a sound, as well as the detail/error. We can also experiment with samples generated from a mathematical function. Figure 5.11 plots the error for different functions, with $N = 1024$. In these cases, we see that we require large m before the detail/error becomes significant. We see also that there is no error for the square wave. The reason is that the square wave is a piecewise constant function, so that it can be represented exactly by the ϕ-functions. For the other functions, however, this

is not the case, so we here het an error.

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

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

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

Moreover $\psi_{m,n}$ is nonzero only on $[2^{-m}n, 2^{-m}(n+1))$, and is 1 on $[2^{-m}n, 2^{-m}(n+1/2))$, and -1 on $[2^{-m}(n+1/2), 2^{-m}(n+1))$. We can therefore write

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

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

5.3.2 Matrix factorization of the DWT

In this section we will write down a matrix factorization of the DWT. This factorization is not used much in mathematical statements, since one typically hides this in implementations of the DWT. This is very similar to the case for the

FFT, where the matrix factorizations grow increasingly complex when $N = 2^n$ is large, but where the algorithms are still very compact. We need the concept of a direct sum of matrices before we can write down the DWT factorization:

Definition 5.24 (Direct sum of matrices)**.** Let $T_1, T_2, \ldots, T_n$ be square matrices. By the direct sum of $T_1, \ldots, T_n$, denoted $T_1 \oplus T_2 \oplus \cdots \oplus T_n$, we mean the block-diagonal matrix where the matrices T_1, T_2,...,T_n are placed along the diagonal, with zeros everywhere else.

We can now establish the matrix factorization of the DWT and IDWT in terms of the direct sum of matrices:

Theorem 5.25 (Matrix of the m-level DWT)**.** Define the $(2^m N) \times (2^m N)$ change of coordinate matrices

$$G_m = (P_{\boldsymbol{\phi}_m \leftarrow \mathcal{C}_m})^T$$
$$H_m = P_{\mathcal{C}_m \leftarrow \boldsymbol{\phi}_m}$$

The m-level DWT and IDWT can be expressed as

$$\begin{aligned} F_m &= (P_{2^1 N} H_1 \oplus I_{2^m N - 2^1 N})(P_{2^2 N} H_2 \oplus I_{2^m N - 2^2 N}) \\ &\quad \cdots (P_{2^{m-1} N} H_{m-1} \oplus I_{2^m N - 2^{m-1} N}) P_{2^m N} H_m, \\ (F_m)^{-1} &= (P_{2^m N} G_m)^T ((P_{2^{m-1} N} G_{m-1})^T \oplus I_{2^m N - 2^{m-1} N}) \\ &\quad \cdots ((P_{2^2 N} G_2)^T \oplus I_{2^m N - 2^2 N})((P_{2^1 N} G_1)^T \oplus I_{2^m N - 2^1 N}). \end{aligned}$$

where P_N is the matrix we used to group a vector into its even- and odd indexed samples in Section 4.1 (i.e. $P_N \boldsymbol{x} = (\boldsymbol{x}^{(e)}, \boldsymbol{x}^{(o)})$).

Proof. The m level DWT performs m changes of coordinates in order. For $k = 0, 1, \ldots, m-1$, these steps are (in this order), the change of coordinates from the canonical basis of $V_{m-k} \oplus_{r=m-k}^{m-1} W_r$ to the canonical basis of $V_{m-k-1} \oplus_{r=m-k-1}^{m-1} W_r$. This change of coordinates only transforms the coordinates from V_{m-k}, and there are $2^{m-k} N$ such coordinates. The remaining $2^m N - 2^{m-k} N$ coordinates are left unchanged, which corresponds to

$$0, 2^m N - 2^{m-1} N, \ldots, 2^m N - 2^{m-(m-2)} N, ..., 2^m N - 2^{m-(m-1)} N$$

coordinates for $k = 0, 1, \ldots, m-1$, which explain the I_0, $I_{2^m N - 2^{m-1} N}$,..., $I_{2^m N - 2^2 N}$, $I_{2^m N - 2^1 N}$ matrices above from right to left. The change of coordinates from V_{m-k} to $V_{m-k-1} \oplus W_{m-k-1}$ is implemented with the change of coordinates matrix H_{m-k}, followed by a reoredering of the coordinates so that the even-indexed ones come first. It is clear that this can be implemented as $P_{2^{m-k} N} H_{m-k}$, where $P_{2^{m-k} N}$ is defined as in Section 4.1. This explains the matrices $P_{2^m N} H_m$, $P_{2^{m-1} N} H_{m-1}$,..., $P_{2^2 N} H_2$, $P_{2^1 N} H_1$ above, from right to left.

The m-level IDWT is the product of the inverse matrices in the opposite order. We have that

$$\begin{aligned}(P_{2^{m-k}N}H_{m-k} \oplus I_{2^m N - 2^{m-k}N})^{-1} &= (P_{2^{m-k}N}H_{m-k})^{-1} \oplus I_{2^m N - 2^{m-k}N} \\ &= (G_{m-k})^T (P_{2^{m-k}N})^T \oplus I_{2^m N - 2^{m-k}N} \\ &= (P_{2^{m-k}N}G_{m-k})^T \oplus I_{2^m N - 2^{m-k}N}\end{aligned}$$

where we used Exercise 5. The result now follows. □

A good question is why we use the transpose of G_m in its definition. We will discuss this later.

5.3.3 Summary

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

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

We also showed that continuous functions could be approximated arbitrarily well from V_m, as long as m was chosen large enough. Moreover it is clear that the space V_0 is closed under all translates, at least if we view the functions in V_0 as periodic with period N, defined as previously on the period $[0, N)$ (translating with N then means that we get the same function back). In the following we will always identify a function with this periodic extension, just as we did in Fourier analysis. When performing this identification, it is also clear that $f(t) \in V_m$ if and only if $g(t) = f(2t) \in V_{m+1}$. We have therefore shown that the scaling funtion ϕ fits in with the following general framework.

Definition 5.26. A Multiresolution analysis, or MRA, is a nested sequence of function spaces

$$V_0 \subset V_1 \subset V_2 \subset \cdots \subset V_m \subset \cdots \tag{5.32}$$

so that

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

Note that, while the basis function we have seen up to now have been orthogonal, we state here that we allow them to be simply a basis as well. The reason is that it will turn out that the assumption of orthogonality may be too

strict, in that it makes it difficult to construct interesting wavelets. We will return to this. The concept of Multiresolution Analysis is much used, and one can find a wide variety of functions ϕ (not only piecewise constant functions), which gives rise to a Multiresolution Analysis. With a multiresolution analysis there is another important thing we also need: We need to be able to efficiently compute the decomposition of $g_m \in V_m$ into the low resolution approximation $g_{m-1} \in V_{m-1}$ and the detail $e_{m-1} \in W_{m-1}$. This requires that we have a simple expression for the corresponding projections. In particular, we need to find a basis for W_m, which hopefully also is orthonormal. Once we have this, the orthogonal decomposition formula can be used to compute the projections, i.e. we can compute the detail and low resolution approximations. Let us summarize this in the following recipe for constructing wavelets:

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

1. Find a function ϕ which gives rise to a multiresolution analysis, and so that we easily can compute the projection from V_1 onto V_0.
2. Find a function ψ so that $\{\psi(t-n)\}_{0\leq n<N}$ is an orthonormal basis for W_0, and so that we easily can compute the projection from V_1 onto W_0.

If we can achieve this, the m-level Discrete Wavelet Transform can be defined and computed similarly as in the case when ϕ is a piecewise constant function, with the obvious replacements.

In the next sections we will follow this recipe in order to contruct other wavelets. Along the way we will run into other questions which are interesting. One of them is, given the resolution spaces, is there a unique choice of ϕ, ψ? If not, are any choices of ϕ, ψ better than others? How can we quantify how good such a choice is?

Exercises for Section 5.3

Ex. 1 — Generalize exercise 5.2.4 to the projections from V_{m+1} onto V_m amd W_m.

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

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

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

when $\boldsymbol{c}_1 \in C_1$, $\boldsymbol{c}_2 \in C_2$, ..., $\boldsymbol{c}_n \in C_n$. Show that, if $\mathcal{B}_i$ is a basis for C_i then

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

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

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

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

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

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

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

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

a. Write a function

```
function playDWTlower(m)
```

which

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

b. Run the function `playDWTlower` for different values of m. For which m can you hear that the sound gets degraded? How does it get degraded? Compare with what you heard through the function `playDFTlower` in Example 3.15, where you performed a DFT on the sound sample instead, and set some of the DFT coefficients to zero.

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

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

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

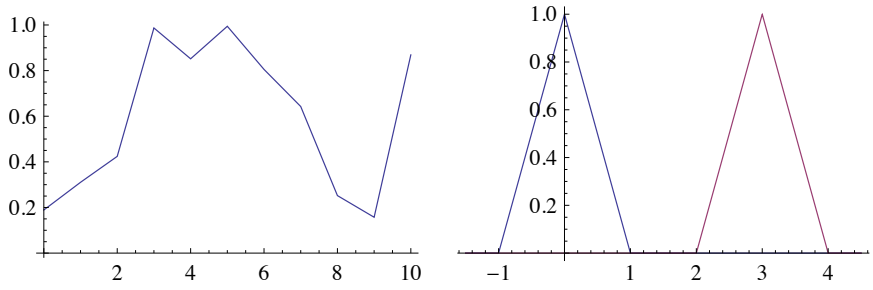
Ex. 12 — Compute the wavelet detail coefficients analytically for the functions in Example 5.22, i.e. compute the quantities $w_{m,n} = \int_0^N f(t)\psi_{m,n}(t)dt$ similarly to how this was done in Example 5.23.

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

5.4 Wavelets constructed from piecewise linear functions

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

In many situations, piecewise constant functions are too simple, and in this section we are going to extend the construction of wavelets to piecewise linear functions. The advantage is that piecewise linear functions are better for approximating smooth functions and data than piecewise constants, which should translate into smaller components (errors) in the detail spaces in many practical situations. As an example, this would be useful if we are interested in



(a) A piecewise linear function. (b) The two functions $\phi(t)$ and $\phi(t-3)$.

Figure 5.12: Some piecewise linear functions.

compression.

5.4.1 Multiresolution analysis

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

Definition 5.28 (Resolution spaces of piecewise linear functions)**.** The space V_m is the subspace of continuous functions on $\mathbb{R}$ which are periodic with period N, and linear on each subinterval of the form $[n2^{-m}, (n+1)2^{-m})$.

Any $f \in V_m$ is uniquely determined by its values on $[0, N)$. Figure 5.12 (a) shows an example of a piecewise linear function in V_0 on the interval $[0, 10]$. We note that a piecewise linear function in V_0 is completely determined by its value at the integers, so the functions that are 1 at one integer and 0 at all others are particularly simple and therefore interesting, see Figure 5.12 (b). These simple functions are all translates of each other and can therefore be built from one scaling function, as is required for a multiresolution analysis.

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

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

Lemma 5.29. Let the function ϕ be defined by

$$\phi(t) = \begin{cases} 1+t, & \text{if } -1 \leq t < 0; \\ 1-t, & \text{if } 0 \leq t < 1; \\ 0, & \text{otherwise;} \end{cases} \tag{5.33}$$

and for any $m \geq 0$ set

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

or in vector notation

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

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

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

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

$$f(t) = \sum_{n=0}^{N-1} f(n)\phi_{0,n}(t) \tag{5.34}$$

since this function agrees with f at the integers in the interval $[0, N]$ and is linear on each subinterval between two neighbouring integers. This means that V_0 is spanned by the functions $\{\phi_{0,n}\}_{n=0}^{N-1}$. On the other hand, if f is identically 0, all the coefficients in (5.34) are also 0, so $\{\phi_{0,n}\}_{n=0}^{N-1}$ are linearly independent and therefore a basis for V_0.

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

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

A formula like (5.34) is also valid for functions in V_m.

Lemma 5.30. A function $f \in V_m$ may be written as

$$f(t) = \sum_{n=0}^{2^m N-1} f(n/2^m)\phi_{m,n}(t). \tag{5.35}$$

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

Lemma 5.31. The piecewise linear resolution spaces are nested,

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

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

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

Let us sum up our findings so far.

Observation 5.32. The spaces V_0, V_1, …, V_m, … form a multiresolution analysis generated by the scaling function ϕ.

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

Lemma 5.33. The function $\phi_{0,n}$ satisfies the relation

$$\phi_{0,n} = \frac{1}{2}\phi_{1,2n-1} + \phi_{1,2n} + \frac{1}{2}\phi_{1,2n+1}. \tag{5.36}$$

A general function g_0 in V_0 is also in V_1, and if

$$g_0 = \sum_{n=0}^{N-1} c_{0,n}\phi_{0,n} = \sum_{n=0}^{2N-1} c_{1,n}\phi_{1,n}$$

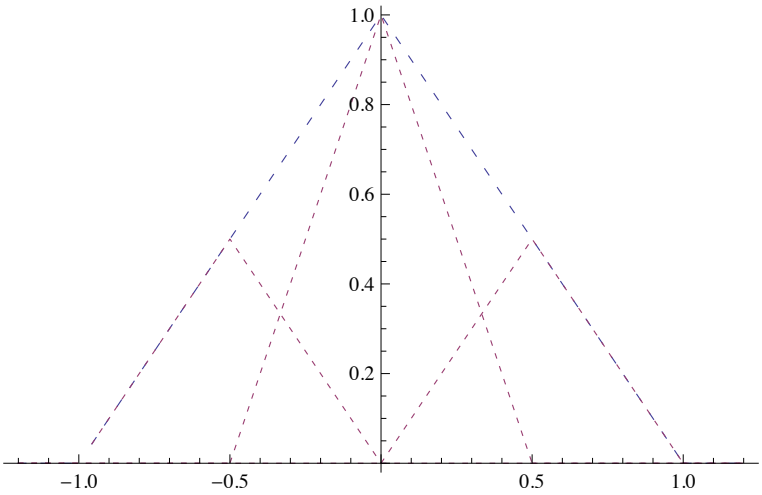
then

$$c_{1,2n} = c_{0,n}, \qquad \text{for } n = 0, 1, \ldots, N-1; \tag{5.37}$$

$$c_{1,2n+1} = (c_{0,n} + c_{0,(n+1) \bmod N})/2, \qquad \text{for } n = 0, 1, \ldots, N-1. \tag{5.38}$$

Proof. Since $\phi_{0,n}$ is in V_0 it may be expressed in the basis $\boldsymbol{\phi}_1$ with formula (5.35),

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



The relation (5.36) now follows since

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

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

To prove (5.37) and (5.38), we use (5.36),

$$\begin{aligned} g_0 &= \sum_{n=0}^{N-1} c_{0,n}\phi_{0,n} \\ &= \sum_{n=0}^{N-1} c_{0,n}(\phi_{1,2n-1}/2 + \phi_{1,2n} + \phi_{1,2n+1}/2) \\ &= \sum_{n=0}^{N-1} c_{0,n}\phi_{1,2n} + \sum_{n=0}^{N-1} c_{0,(n+1) \bmod N}\phi_{1,2n+1}/2 + \sum_{n=0}^{N-1} c_{0,n}\phi_{1,2n+1}/2 \\ &= \sum_{n=0}^{N-1} c_{0,n}\phi_{1,2n} + \sum_{n=0}^{N-1} (c_{0,n} + c_{0,(n+1) \bmod N})\phi_{1,2n+1}/2, \end{aligned}$$

where we have performed a substitution of the form $n \to n+1$. The result now follows by comparing with $\sum_{n=0}^{2N-1} c_{1,n}\phi_{1,n}$. □

The relations in Lemma 5.33 can also be expressed in matrix form. If we set

$$\boldsymbol{c}_1 = (c_{1,n})_{n=0}^{2N-1}, \quad \boldsymbol{c}_1^e = (c_{1,2n})_{n=0}^{N-1}, \quad \boldsymbol{c}_1^o = (c_{1,2n+1})_{n=0}^{N-1},$$

we may write the equations (5.37) and (5.38) as

$$\begin{pmatrix} \boldsymbol{c}_1^e \\ \boldsymbol{c}_1^o \end{pmatrix} = \begin{pmatrix} I \\ A_0 \end{pmatrix} \boldsymbol{c}_0, \tag{5.39}$$

where I is the $N \times N$ identity matrix and A_0 is the $N \times N$ circulant Toeplitz matrix given by

$$A_0 = \frac{1}{2}\begin{pmatrix} 1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 1 \end{pmatrix}. \tag{5.40}$$

The formulas (5.37)–(5.38), or alternatively (5.39), show how a function in V_0 and can be represented in V_1. Analogous formulas let us rewrite a function in V_k in terms of the basis for V_{k+1}.

5.4.2 Detail spaces and wavelets

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

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

Definition 5.34. Let g_1 be a function in V_1 given by

$$g_1 = \sum_{n=0}^{2N-1} c_{1,n}\ \phi_{1,n}. \tag{5.41}$$

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

$$g_0(n) = g_1(n), \quad n = 0,\, 1,\, \ldots,\, N-1. \tag{5.42}$$

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

Lemma 5.35. Let g_1 be given by (5.41) and suppose that $g_0 = \sum_{n=0}^{N-1} c_{0,n}\phi_{0,n}$ in V_0 interpolates g_1 at the integers in $0, \ldots, N-1$. Then the coefficients are given by

$$c_{0,n} = c_{1,2n}, \quad \text{for } n = 0,\, 1,\, \ldots,\, N-1, \tag{5.43}$$

and

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

Once the method of approximation is determined, it is straightforward to determine the detail space as the space of error functions. With the notation from Definition 5.34, the error is given by $e_0 = g_1 - g_0$. Since g_0 interpolates g_1 at the integers, the error is 0 there,

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

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

Lemma 5.36. Suppose the function g_0 in V_0 interpolates a function g_1 in V_1 at the integers. Then the error $e_0 = g_1 - g_0$ lies in the space W_0 defined by

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

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

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

If $g_1 = \sum_{n=0}^{2N-1} c_{1,n}\ \phi_{1,n}$ is approximated by g_0 in V_0 as in Lemma 5.35, the error is given by $e_0 = \sum_{n=0}^{N-1} w_{0,n}\psi_{0,n}$ where

$$w_{0,n} = c_{1,2n+1} - \frac{1}{2}(c_{1,2n} + c_{1,(2n+2) \bmod 2N}). \tag{5.44}$$

Proof. We must show that any error function can be written in terms of the wavelets. First of all we note that the wavelets are linearly independent since their supports do not intersect. Let $g_1 \in V_1$ be as in (5.41) and let the $g_0 \in V_0$

be the approximation described by Lemma 5.35. Then the error is given by

$$
\begin{aligned}
e_0 &= g_1 - g_0 \\
&= \sum_{n=0}^{2N-1} c_{1,n}\phi_{1,n} - \sum_{n=0}^{N-1} c_{0,n}\phi_{0,n} \\
&= \sum_{n=0}^{N-1} c_{1,2n}\phi_{1,2n} + \sum_{n=0}^{N-1} c_{1,2n+1}\phi_{1,2n+1} - \sum_{n=0}^{N-1} c_{1,2n}\phi_{0,n} \\
&= \sum_{n=0}^{N-1} c_{1,2n}\phi_{1,2n} + \sum_{n=0}^{N-1} c_{1,2n+1}\phi_{1,2n+1} \\
&\qquad - \frac{1}{2}\sum_{n=0}^{N-1} c_{1,2n}\phi_{1,2n-1} - \sum_{n=0}^{N-1} c_{1,2n}\phi_{1,2n} - \frac{1}{2}\sum_{n=0}^{N-1} c_{1,2n}\phi_{1,2n+1} \\
&= \sum_{n=0}^{N-1} c_{1,2n}\phi_{1,2n} + \sum_{n=0}^{N-1} c_{1,2n+1}\phi_{1,2n+1} \\
&\qquad - \frac{1}{2}\sum_{n=0}^{N-1} c_{1,(2n+2) \bmod 2N}\phi_{1,2n+1} - \sum_{n=0}^{N-1} c_{1,2n}\phi_{1,2n} - \frac{1}{2}\sum_{n=0}^{N-1} c_{1,2n}\phi_{1,2n+1} \\
&= \sum_{n=0}^{N-1} \Big(c_{1,2n+1} - \frac{1}{2}(c_{1,2n} + c_{1,(2n+2) \bmod 2N}) \Big) \phi_{1,2n+1}.
\end{aligned}
$$

In the third equation we split the sum in g_1 into even and odd terms and used the definition of g_0 in (5.43). In the next step we then rewrote g_0 in V_1 using formula (5.37), and finally we rewrote $\phi_{0,n}$ using formula (5.36). □

We now have all the ingredients to formulate an analog of Theorem 5.18 that describes how V_m can be expressed as a direct sum of V_{m-1} and W_{m-1}. The formulas for $m = 1$ generalise without change, except that the upper bound on the summation indices must be adjusted.

Theorem 5.37. The space V_m can be decomposed as the direct sum $V_m = V_{m-1} \oplus W_{m-1}$ where W_{m-1} is the space of all functions in V_m that are zero at the points $\{n/2^{m-1}\}_{n=0}^{N2^{m-1}-1}$. The space V_m has the two bases

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

and

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

If $g_m \in V_m$, $g_{m-1} \in V_{m-1}$, and $e_{m-1} \in W_{m-1}$ are given by

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

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

$$\begin{aligned} c_{m-1,n} &= c_{m,2n}, \\ w_{m-1,n} &= c_{m,2n+1} - (c_{m,2n} + c_{m,(2n+2) \bmod 2N})/2. \end{aligned}$$

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

$$c_{m,2n} = c_{m-1,n}, \tag{5.45}$$

$$c_{m,2n+1} = w_{m-1,n} + (c_{m-1,n} + c_{m-1,n+1})/2. \tag{5.46}$$

The matrix notation in (5.39) may be generalised to cover Theorem 5.37. With the natural extension of the notation in (5.39) we see that IDWT given by (5.45) and (5.46) can be expressed as

$$\begin{pmatrix} \boldsymbol{c}_m^e \\ \boldsymbol{c}_m^o \end{pmatrix} = \begin{pmatrix} I & 0 \\ A_{m-1} & I \end{pmatrix} \begin{pmatrix} \boldsymbol{c}_{m-1} \\ \boldsymbol{w}_{m-1} \end{pmatrix} \tag{5.47}$$

where both identity matrices have dimension $N2^{m-1}$. The matrix A_{m-1} is a $\left(N2^{m-1}\right) \times \left(N2^{m-1}\right)$ matrix which is the natural generalisation of the matrix A_0 defined in (5.40). The DWT is simply the inverse of (5.39) and is given by

$$\begin{pmatrix} \boldsymbol{c}_{m-1} \\ \boldsymbol{w}_{m-1} \end{pmatrix} = \begin{pmatrix} I & 0 \\ -A_{m-1} & I \end{pmatrix} \begin{pmatrix} \boldsymbol{c}_m^e \\ \boldsymbol{c}_m^o \end{pmatrix}. \tag{5.48}$$

There is another simple expression for the DWT we will have use for. From Equation (5.36) and from the definition of ψ we have

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

Again, it is custom to use the normalized functions $\phi_{m,n}(t) = 2^{1/2}\phi(2^m t - n)$.

Using these instead, the two equations above take the form

$$\begin{aligned}\phi_{0,n} &= \frac{1}{2\sqrt{2}}\phi_{1,2n-1} + \frac{1}{\sqrt{2}}\phi_{1,2n} + \frac{1}{2\sqrt{2}}\phi_{1,2n+1}\\ \psi_{0,n} &= \frac{1}{\sqrt{2}}\phi_{1,2n+1}.\end{aligned} \tag{5.49}$$

These two relations together give all columns in the change of coordinate matrix $P_{\boldsymbol{\phi}_1 \leftarrow \mathcal{C}_1}$, when the spaces $\boldsymbol{\phi}_m$, $\mathcal{C}_m$ instead are defined in terms of the function ψ, and the normalized ϕ. In particular, the first two columns in this matrix are

$$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 0 \\ 1/2 & 1 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 1/2 & 0 \end{pmatrix}. \tag{5.50}$$

The remaining columns are obtained by shifting this, as in a circulant Toeplitz matrix. Similarly we can compute the change of coordinate matrix the opposite way, $P_{\mathcal{C}_1 \leftarrow \boldsymbol{\phi}_1}$: Equations (5.49) can be written

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

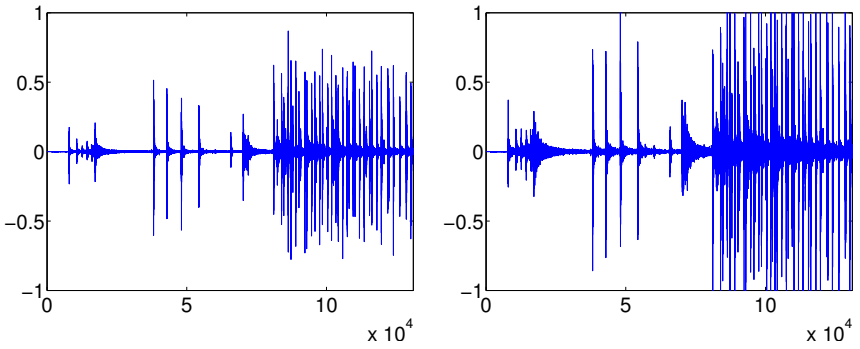
from which it follows that

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

which in the same way as above give the following two first columns in the change of coordinate matrix $P_{\mathcal{C}_1 \leftarrow \boldsymbol{\phi}_1}$:

$$\sqrt{2}\begin{pmatrix} 1 & 0 \\ -1/2 & 1 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ -1/2 & 0 \end{pmatrix}. \tag{5.51}$$

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



(a) $m = 1$ (b) $m = 2$

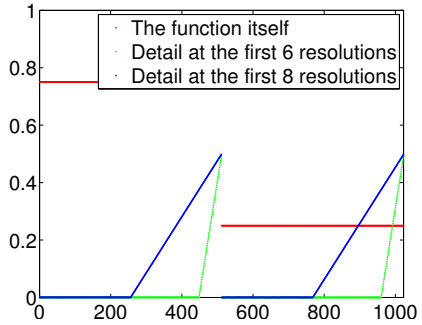
Figure 5.13: The error (i.e. the contribution from $W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}$) in the sound file `castanets.wav`, for different values of m.

Example 5.38. In Section 5.6 we will construct an algorithm which performs DWT/IDWT, for a general wavelet. In particular, this algorithm can be used for the wavelet we constructed in this section. Let us also for this wavelet plot the detail/error in the test audio file `castanets.wav` for different resolutions, as we did in Example 5.21. The result is shown in Figure 5.13. When comparing with Figure 5.10 we see much of the same, but it seems here that the error is bigger than before. In the next section we will try to explain why this is the case, and construct another wavelet based on piecewise linear functions which remedies this.

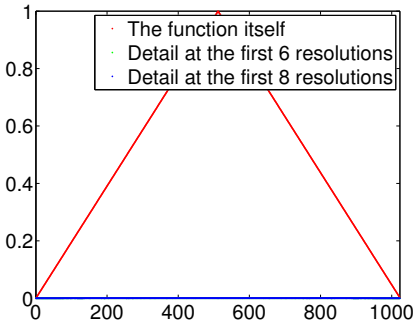
Example 5.39. Let us also repeat Exercise 5.22, where we plotted the detail/error at different resolutions, for the samples of a mathematical function. Figure 5.14 shows the new plot. With the square wave we see now that there is an error. The reason is that a piecewise constant function can not be represented exactly by piecewise linear functions, due to discontinuity. For the second function we see that there is no error. The reason is that this function is piecewise constant, so there is no error when we represent the function from the space V_0. With the third function, hoewever, we see an error.

Exercises for Section 5.4

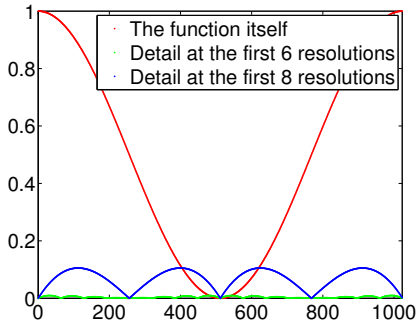
Ex. 1 — Show that, for $f \in V_0$ we have that $[f]_{\phi_0} = (f(0), f(1), \ldots, f(N-1))$. This generalizes the result for piecewise constant functions.



(a) A square wave



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



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

Figure 5.14: The error (i.e. the contribution from $W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}$) for $N = 1024$ for different functions $f(t)$, for different values of m.

Ex. 2 — Show that

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

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

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

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

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

5.5 Alternative wavelets for piecewise linear functions

The direct sum decomposition that we derived in Section 5.4 was very simple, but also has its shortcomings. To see this, set $N = 1$ and consider the space V_{10}, which has dimension 2^{10}, which in most cases will mean that the function g_{10} will 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.

To address this shortcoming, let us return to the piecewise constant wavelet, and assume that $f \in V_m$. By the orthogonal decomposition theorem we have

$$f = \sum_{n=0}^{N-1} \langle f, \phi_{0,n} \rangle \phi_{0,n} + \sum_{r=0}^{m-1} \sum_{n=0}^{2^r N - 1} \langle f, \psi_{r,n} \rangle \psi_{r,n}. \tag{5.52}$$

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 $\langle t^k, \psi \rangle = 0$, for $k = 1, \ldots, s$, we have also that $\langle t^k, \psi_{r,t} \rangle = 0$ for $r \leq s$, so that $\langle P_s, \psi_{r,t} \rangle = 0$

also. This means that (5.52) can be written

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

Here the first sum lies in V_0. We see that the wavelet coefficients from W_r are $\langle Q_s, \psi_{r,n}\rangle$, 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 5.40 (Vanishing moments)**.** We say that ψ has k vanishing moments if the integrals $\int_{-\infty}^{\infty} t^l \psi(t) dt = 0$ for all $0 \leq l \leq k-1$. If a function $f \in V_m$ is r times differentiable, and ψ has r vanishing moments, then f can be approximated well from V_0. Moreover, the quality of this approximation improves when r increases.

It is also clear from the argument that if f is a polynomial of degree less than or equal to $k-1$ and ψ has k vanishing moments, then the wavelet detail coefficients are exactly 0. This theorem at least says what we have to aim for when the wavelet basis is orthonormal. The concept of vanishing moments also makes sense when the wavelet is not orthonormal, however. One can show that it is desirable to have many vanishing moments for other wavelets also. We we will not go into this (the wavelets used in practice turn out to be "almost" orthonormal, although they are not actually orthonormal. In such cases the computations above serve as good approximations, so that it is desirable to have many vanishing moments also here).

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

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

Idea 5.41. Adjust the wavelet construction in Theorem 5.37 so that the new wavelets $\{\hat{\psi}_{m-1,n}\}_{n=0}^{N2^m-1}$ in W_{m-1} satisfy

$$\int_0^N \hat{\psi}_{m-1,n}(t)\, dt = \int_0^N t\hat{\psi}_{m-1,n}(t)\, dt = 0, \tag{5.53}$$

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

As usual, it is sufficient to consider what happens when V_1 is written as a direct sum of V_0 and W_0. From Idea 5.41 we see that we need to enforce two conditions for each wavelet function. If we adjust the wavelets in Theorem 5.37 by adding multiples of the two neighbouring hat functions, we have two free parameters,

$$\hat{\psi}_{0,n} = \psi_{0,n} - \alpha\phi_{0,n} - \beta\phi_{0,n+1} \tag{5.54}$$

that we may determine so that the two conditions in (5.53) are enforced. If we do this, we get the following result:

Lemma 5.42. The function

$$\hat{\psi}_{0,n}(t) = \psi_{0,n}(t) - \frac{1}{4}\big(\phi_{0,n}(t) + \phi_{0,n+1}(t)\big) \tag{5.55}$$

satisfies the conditions

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

Using Equation (5.36), which stated that

$$\phi_{0,n}(t) = \frac{1}{2}\phi_{1,2n-1} + \phi_{1,2n} + \frac{1}{2}\phi_{1,2n+1} \tag{5.56}$$

we get

$$\begin{aligned}
\hat{\psi}_{0,n}(t) &= \psi_{0,n}(t) - \frac{1}{4}\big(\phi_{0,n}(t) + \phi_{0,n+1}(t)\big) \\
&= \phi_{1,2n+1}(t) - \frac{1}{4}\big(\frac{1}{2}\phi_{1,2n-1} + \phi_{1,2n} + \frac{1}{2}\phi_{1,2n+1} + \frac{1}{2}\phi_{1,2n+1} + \phi_{1,2n+2} + \frac{1}{2}\phi_{1,2n+3}\big) \\
&= -\frac{1}{8}\phi_{1,2n-1} - \frac{1}{4}\phi_{1,2n} + \frac{3}{4}\phi_{1,2n+1} - \frac{1}{4}\phi_{1,2n+2} - \frac{1}{8}\phi_{1,2n+3}.
\end{aligned} \tag{5.57}$$

Note that what we did here is equivalent to finding the coordinates of $\hat{\psi}$ in the basis $\boldsymbol{\phi}_1$: Equation (5.55) says that

$$[\hat{\psi}]_{\boldsymbol{\phi}_0 \oplus \boldsymbol{\psi}_0} = (-1/4, -1/4, 0, \ldots, 0) \oplus (1, 0, \ldots, 0). \tag{5.58}$$

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

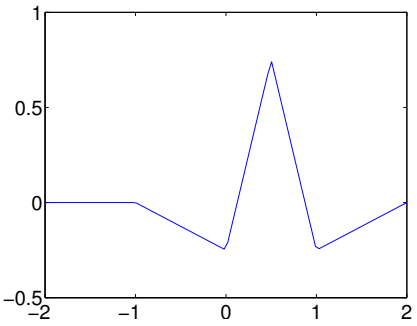


Figure 5.15: The function ψ we constructed as an alternative wavelet for piecewise linear functions.

Again, it is custom to use the normalized functions $\phi_{m,n}(t) = 2^{1/2}\phi(2^m t - n)$. Using these instead, the two equations above take the form

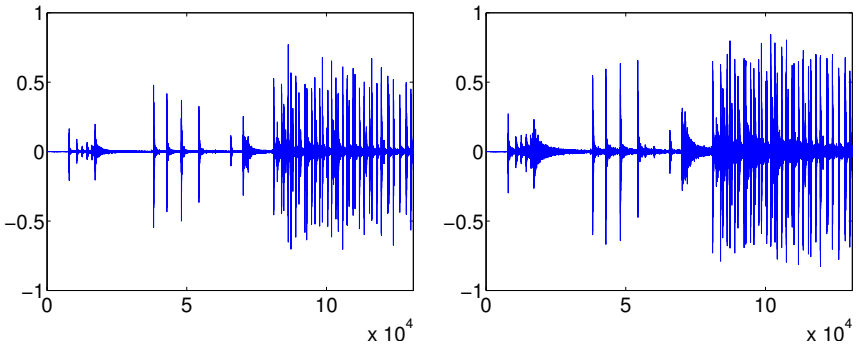
$$\phi_{0,n}(t) = \frac{1}{2\sqrt{2}}\phi_{1,2n-1} + \frac{1}{\sqrt{2}}\phi_{1,2n} + \frac{1}{2\sqrt{2}}\phi_{1,2n+1}$$
$$\hat{\psi}_{0,n}(t) = -\frac{1}{8\sqrt{2}}\phi_{1,2n-1} - \frac{1}{4\sqrt{2}}\phi_{1,2n} + \frac{3}{4\sqrt{2}}\phi_{1,2n+1} - \frac{1}{4\sqrt{2}}\phi_{1,2n+2} - \frac{1}{8\sqrt{2}}\phi_{1,2n+3}.$$

These two relations together give all columns in the change of coordinate matrix $P_{\boldsymbol{\phi}_1 \leftarrow \mathcal{C}_1}$, when the spaces $\boldsymbol{\phi}_m$, $\mathcal{C}_m$ instead are defined in terms of the function $\hat{\psi}$, and the normalized ϕ. In particular, the first two columns in this matrix are

$$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -1/4 \\ 1/2 & 3/4 \\ 0 & -1/4 \\ 0 & -1/8 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 1/2 & -1/8 \end{pmatrix}. \tag{5.59}$$

The first column is the same as before, since there was no change in the definition of ϕ. The remaining columns are obtained by shifting this, as in a circulant Toeplitz matrix. Similarly we could compute the change of coordinate matrix the opposite way, $P_{\mathcal{C}_1 \leftarrow \boldsymbol{\phi}_1}$. We will explain how this can be done in the next section. The function ψ is plotted in Figure 5.15.

Example 5.43. Let us also plot the detail/error in the test audio file `castanets.wav` for different resolutions for our alternative wavelet, as we did in Example 5.21. The result is shown in Figure 5.16. Again, when comparing with Figure 5.10 we see much of the same. It is difficult to see an improvement from this figure. However, this figure also clearly shows a smaller error than the wavelet of



(a) $m = 1$ (b) $m = 2$

Figure 5.16: The error (i.e. the contribution from $W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}$) in the sound file `castanets.wav`, for different values of m.

the preceding section. A partial explanation is that the wavelet we now have constructed has two vanishing moments.

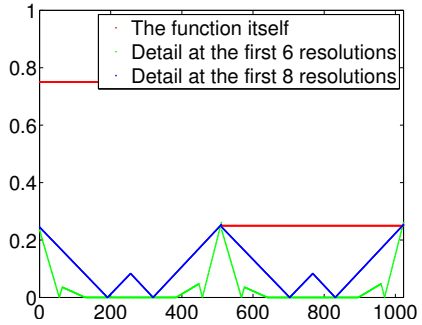
Example 5.44. Let us also repeat Exercise 5.22 for our alternative wavelet, where we plotted the detail/error at different resolutions, for the samples of a mathematical function. Figure 5.17 shows the new plot. Again for the square wave there is an error, which seems to be slightly lower than for the previous wavelet. For the second function we see that there is no error, as before. The reason is the same as before, since the function is piecewise constant. With the third function there is an error. The error seems to be slightly lower than for the previous wavelet, which fits well with the number of vanishing moments.

Exercises for Section 5.5

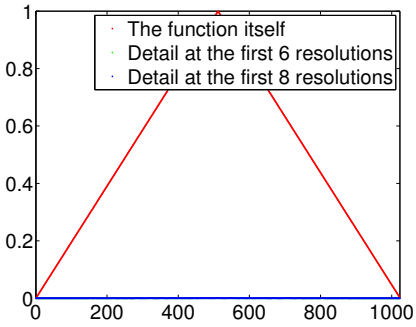
Ex. 1 — In this exercise we will show that there is a unique function on the form (5.54) which has two vanishing moments.

a. Show that, when $\hat{\psi}$ is defined by (5.54), we have that

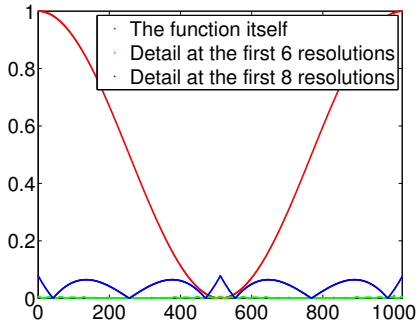
$$\hat{\psi}(t) = \begin{cases} -\alpha t - \alpha & \text{for } -1 \le t < 0 \\ (2 + \alpha - \beta)t - \alpha & \text{for } 0 \le t < 1/2 \\ (\alpha - \beta - 2)t - \alpha + 2 & \text{for } 1/2 \le t < 1 \\ \beta t - 2\beta & \text{for } 1 \le t < 2 \\ 0 & \text{for all other } t \end{cases}$$



(a) A square wave



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



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

Figure 5.17: The error (i.e. the contribution from $W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}$) for $N = 1024$ for different functions $f(t)$, for different values of m.

b. Show that

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

c. Explain why there is a unique function on the form (5.54) which has two vanishing moments, and that this function is given by Equation (5.55).

Ex. 2 — In the previous exercise we ended up with a lot of calculations to find α, β in Equation (5.54). Let us try to make a program which does this for us, and which also makes us able to generalize the result.

a. Define

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

for $k \geq 0$. Explain why finding α, β so that we have two vanishing moments in Equation 5.54 is equivalent to solving the following equation:

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

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

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

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

$$\hat{\psi} = \psi_{0,0} - \alpha\phi_{0,0} - \beta\phi_{0,1} - \gamma\phi_{0,-1} - \delta\phi_{0,2}. \tag{5.60}$$

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

$$g_k = \int_{-2}^{0} t^k(1 - |t+1|)dt$$
$$d_k = \int_1^3 t^k(1 - |t-2|)dt$$

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

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

and solve this with Matlab.

c. Plot the function defined by (5.60), which you found in b.
Hint: If `t` is the vector of t-values, and you write `(t>=0).*(t<=1).*(1-2*abs(t-0.5))`, you get the points $\phi_{1,1}(t)$.

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

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

Hint: you can also compare with Equation (5.58) here. The placement of $-\gamma$ may seem a bit strange here, and has to with that $\phi_{0,-1}$ is not one of the basis functions $\{\phi_{0,n}\}_{n=0}^{N-1}$. However, we have that $\phi_{0,-1} = \phi_{0,N-1}$, i.e. $\phi(t+1) = \phi(t-N+1)$, since we always assume that the functions we work with have period N.

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

Ex. 3 — It is also possible to add more vanishing moments to the Haar wavelet. Define

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

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

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

$$\begin{pmatrix} c_{0,0} & c_{0,1} & \cdots & c_{0,k-1} \\ c_{1,0} & c_{1,1} & \cdots & c_{1,k-1} \\ \vdots & \vdots & \vdots & \vdots \\ c_{k-1,0} & c_{k-1,1} & \cdots & c_{k-1,k-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{k-1} \end{pmatrix} = \begin{pmatrix} e_0 \\ e_1 \\ \vdots \\ e_{k-1} \end{pmatrix} \tag{5.61}$$

b. Write a function

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

which solves Equation 5.61, and returns $a_0, a_1, \ldots, a_{k-1}$ in the vector `a`.

5.6 Wavelets and filters

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

Definition 5.45 (MRA-matrices)**.** An $N \times N$-matrix T, with N even, is called an MRA-matrix if the columns are translates of the first two columns in alternating order, in the same way as the columns of a circulant Toeplitz matrix.

From our previous calculations it is clear that, once ϕ and ψ are given through an MRA, the corresponding change of coordinate matrices will always be MRA-matrices. The MRA-matrices is our connection between matrices and wavelets. We would also like to state a similar connection with filters, i.e. show how the DWT could be implemented in terms of filters. We start with the following definition:

Definition 5.46. We denote by H_0 the (unique) filter with the same first row as $P_{\mathcal{C}_m \leftarrow \boldsymbol{\phi}_m}$, and by H_1 the (unique) filter with the same second row as $P_{\mathcal{C}_m \leftarrow \boldsymbol{\phi}_m}$.

Using this definition it is clear that

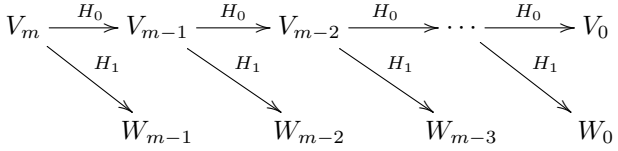
$$(P_{\mathcal{C}_m \leftarrow \boldsymbol{\phi}_m} \boldsymbol{c}_m)_k = (H_0 \boldsymbol{c}_m)_k \qquad \text{when } k \text{ is even}$$
$$(P_{\mathcal{C}_m \leftarrow \boldsymbol{\phi}_m} \boldsymbol{c}_m)_k = (H_1 \boldsymbol{c}_m)_k \qquad \text{when } k \text{ is odd}$$

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

Theorem 5.47 (DWT expressed in terms of filters)**.** Let $\boldsymbol{c}_m$ be the coordinates in $\boldsymbol{\phi}_m$, and let H_0, H_1 be defined as above. Any stage in a DWT can ble implemented in terms of filters as follows:

1. Compute $H_0 \boldsymbol{c}_m$. The even-indexed entries in the result are the cordinates $\boldsymbol{c}_{m-1}$ in $\boldsymbol{\phi}_{m-1}$.
2. Compute $H_1 \boldsymbol{c}_m$. The odd-indexed entries in the result are the coordinates $\boldsymbol{w}_{m-1}$ in $\boldsymbol{\psi}_{m-1}$.

Note that this corresponds to applying two filters, and throwing away half of the results (since we only keep even-indexed and odd-indexed entries, respectively). In practice we do not compute the full application of the filter due to this. We can now complement Figure 5.3.1 by giving names to the arrows as follows:



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

Definition 5.48. We denote by G_0 the (unique) filter with the same first column as $P_{\boldsymbol{\phi}_m \leftarrow \mathcal{C}_m}$, and by G_1 the (unique) filter with the same second column as $P_{\boldsymbol{\phi}_m \leftarrow \mathcal{C}_m}$.

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

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

$$= P_{\boldsymbol{\phi}_m \leftarrow \mathcal{C}_m} \begin{pmatrix} c_{m-1,0} \\ 0 \\ c_{m-1,1} \\ 0 \\ \cdots \\ c_{m-1,2^{m-1}N-1} \\ 0 \end{pmatrix} + P_{\boldsymbol{\phi}_m \leftarrow \mathcal{C}_m} \begin{pmatrix} 0 \\ w_{m-1,0} \\ 0 \\ w_{m-1,1} \\ \cdots \\ 0 \\ w_{m-1,2^{m-1}N-1} \end{pmatrix}$$

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

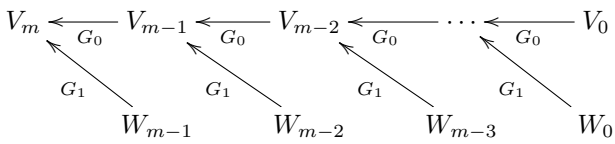
Here we have split a vector into its even-indexed and odd-indexed elements, which correspond to the coefficients from $\boldsymbol{\phi}_{m-1}$ and $\boldsymbol{\psi}_{m-1}$, respectively. In the last equation, we replaced with G_0, G_1, since the multiplications with $P_{\boldsymbol{\phi}_m \leftarrow \mathcal{C}_m}$ depend only on the even and odd columns in that matrix (due to the zeros

inserted), and these columns are equal in G_0, G_1. We can now state the following characterization of the inverse Discrete Wavelet transform:

Theorem 5.49 (IDWT expressed in terms of filters)**.** Let G_0, G_1 be defined as above. Any stage in an IDWT can be implemented in terms of filters as follows:

$$\boldsymbol{c}_m = G_0 \begin{pmatrix} c_{m-1,0} \\ 0 \\ c_{m-1,1} \\ 0 \\ \cdots \\ c_{m-1,2^{m-1}N-1} \\ 0 \end{pmatrix} + G_1 \begin{pmatrix} 0 \\ w_{m-1,0} \\ 0 \\ w_{m-1,1} \\ \cdots \\ 0 \\ w_{m-1,2^{m-1}N-1} \end{pmatrix}. \tag{5.62}$$

We can now also complement Figure 5.3.1 for the IDWT with named arrows as follows:



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

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

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

$$\phi(t) = \sum_{n=0}^{2N-1} (G_0)_{n,0} \phi_{1,n}(t) \tag{5.63}$$

$$\psi(t) = \sum_{n=0}^{2N-1} (G_1)_{n,1} \psi_{1,n}(t)., \tag{5.64}$$

where $(G_k)_{i,j}$ are the entries in the matrix G_k. Similar equations are true for $\phi(t-k), \psi(t-k)$. Due to (5.63), the filter G_0 should have lowpass filter characteristics, since it extracts the information at lower frequencies. G_1 should have highpass filter characteristics du to (5.64). Let us verify this for the different wavelets we have defined up to now.

5.6.1 Frequency response for the Haar Wavelet

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

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

repeated along the diagonal. From this it is clear that

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

We have seen these filters previously: G_0 is a movinge average filter (of two elements), while G_1 is a bass-reducing filter (up to multiplication with a constant). We compute their frequency response as

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

The magnitude of these are plotted in Figure 5.18, where the lowpass/highpass characteristics are clearly seen. The two frequency responses seem also to be the same, except for a shift by π in frequency. We will show later that this is not coincidental. In this case we also have that

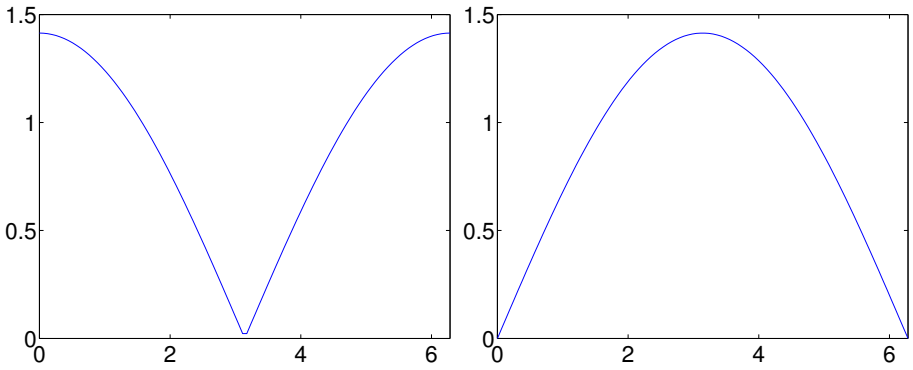
$$H_0 = \{\underline{1/\sqrt{2}}, 1/\sqrt{2}\}$$
$$H_1 = \{1/\sqrt{2}, \underline{-1/\sqrt{2}}\},$$

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

5.6.2 Frequency responses for wavelets of piecewise linear functions

For the first wavelet for piecewise linear functions we looked at in the previous section, Equation (5.50) gives that

$$G_0 = \frac{1}{\sqrt{2}}\{1/2, \underline{1}, 1/2\}$$
$$G_1 = \frac{1}{\sqrt{2}}\{\underline{1}\}. \tag{5.65}$$



(a) $\lambda_{G_0}(\omega)$ (b) $\lambda_{G_1}(\omega)$

Figure 5.18: The frequency responses for the MRA of piecewise constant functions.

G_0 is again a filter we have seen before: Up to multiplication with a constant, it is the treble-reducing filter with values from row 2 of Pascal's triangle. The frequency responses are thus

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

$\lambda_{G_1}(\omega)$ thus has magnitude $\frac{1}{\sqrt{2}}$ at all points. The magnitude of $\lambda_{G_0}(\omega)$ is plotted in Figure 5.19. Comparing with Figure 5.18 we see that here also the frequency response has a zero at π. The frequency response seems also to be flatter around π. For the DWT, Equation (5.51) gives us

$$H_0 = \sqrt{2}\{\underline{1}\}$$
$$H_1 = \sqrt{2}\{-1/2, \underline{1}, -1/2\}. \tag{5.66}$$

We see that, up to a constant, H_1 is obtained from G_0 by adding an alternating sign. We know from before that this turns a lowpass filter into a highpass filter, so that H_1 is a highpass filter (it is a bass-reducing filter with values taken from row 2 of Pascals triangle).

Let us compare with the alternative wavelet we used for piecewise linear functions. In Equation (5.59) we wrote down the first two columns in $P_{\boldsymbol{\phi}_m \leftarrow \mathcal{C}_m}$. This gives us that the two filters are

$$G_0 = \frac{1}{\sqrt{2}}\{1/2, \underline{1}, 1/2\}$$
$$G_1 = \frac{1}{\sqrt{2}}\{-1/8, -1/4, \underline{3/4}, -1/4, -1/8\}. \tag{5.67}$$

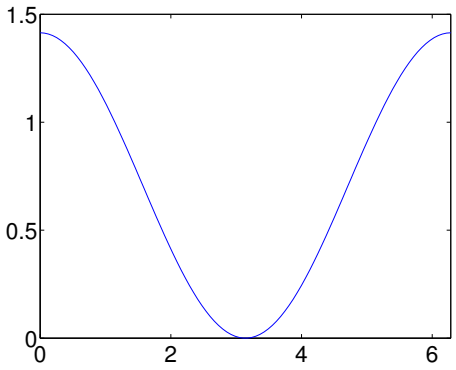


Figure 5.19: The frequency response $\lambda_{G_0}(\omega)$ for the first choice of wavelet for piecewise linear functions

Here G_0 was as before since we use the same scaling function, but G_1 was changed. We also have to find the filters H_0, H_1. It can be shown that (although we do not prove this), if $g_{0,n}$, $g_{1,n}$, $h_{0,n}$, $h_{1,n}$ are the filter coefficients for the filters, then

$$\begin{aligned} h_{0,n} &= \alpha(-1)^n g_{1,n} \\ h_{1,n} &= \alpha(-1)^n g_{0,n}, \end{aligned} \tag{5.68}$$

where $\alpha = \frac{1}{\sum_n g_{0,n} g_{1,n}}$. In other words, the filters are the same as G_0, G_1 up to multiplication by a constant, and an alternating sign. This means, more generally, that H_1 is a highpass filter when G_0 is a lowpass filter, and that H_0 is a lowpass filter when G_1 is a highpass filter. In this case, this means that

$$\alpha = \frac{1}{\frac{1}{2}\left(-\frac{1}{2}\left(-\frac{1}{4}\right) + 1 \cdot \frac{3}{4} - \frac{1}{2}\left(-\frac{1}{4}\right)\right)} = 2,$$

so that

$$\begin{aligned} H_0 &= \sqrt{2}\{-1/8, 1/4, \underline{3/4}, 1/4, -1/8\} \\ H_1 &= \sqrt{2}\{-1/2, \underline{1}, -1/2\}. \end{aligned} \tag{5.69}$$

We now have that

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

The magnitude of $\lambda_{G_1}(\omega)$ is plotted in Figure 5.20. Clearly, G_1 now has highpass characteristics, while the lowpass characteristic of G_0 has been preserved. The filters G_0, G_1, H_0, H_1 are particularly important in applications: Apart from the

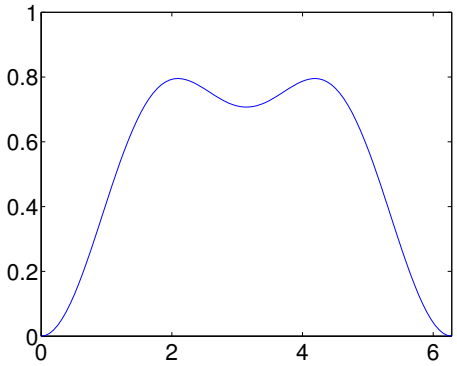


Figure 5.20: The frequency response $\lambda_{G_1}(\omega)$ for the alternative wavelet for piecewise linear functions.

scaling factors $1/\sqrt{2}$, $\sqrt{2}$ in front, we see that the filter coefficients are all dyadic fractions, i.e. they are on the form $\beta/2^j$. Arithmetic operations with dyadic fractions can be carried out exactly on a computer, due to representations as binary numbers in computers. These filters are thus important in applications, since they can be used as transformations for lossless coding. The same argument can be made for the Haar wavelet, but this wavelet had one less vanishing moment.

5.6.3 Filter-based algorithm for the DWT and the IDWT

From the analysis in this section, we see that we can implement DWT/IDWT based on expressions for the corresponding filters. This opens up for other opportunities also, in that we can start altogether by providing filters G_0, G_1, H_0, H_1, and construct MRA-matrices with the same even/odd-indexed rows/-columns as these filters (as in Theorem 5.47 and Theorem 5.49). If we can find such filters so that the corresponding MRA-matrices invert each other, we can use them in implementations of the DWT/IDWT, even though we have no idea what the underlying functions ϕ, ψ may be, or if such functions exist at all.

This approach will be made in the following. To be more precise, we will provide filters G_0, G_1, H_0, H_1 which are used in practice, and where it is known that the corresponding MRA-matrices invert each other, and apply these in the algorithms sketched in Theorem 5.47 and Theorem 5.49. We will restrict ourself to the case where the filters G_0, G_1, H_0, H_1 all are symmetric. As can be seen from the filter expressions, this was the case for all filters we have looked at, except the Haar wavelet. We have already implemented the Haar wavelet, however. Symmetric filters are also very common in practice. A reason for this is that MRA-matrices based on symmetric filters also can be shown to preserve symmetric vectors, so that they share some of the desirable properties of symmetric filters (which lead us to the definition of the DCT).

The algorithm for the DWT/IDWT is essentially a matrix/vector multiplication, and this can be computed entry by entry once we have the rows of the matrices. With the DWT we have these rows, since the rows in $P_{\mathcal{C}_m \leftarrow \boldsymbol{\phi}_m}$ are given by the rows of H_0, H_1. In Exercise 3.4.3, we implemented a symmetric filter, which took the filter coefficients as input. Another thing we need is to extend this implementation so that it works for the case where the first two rows, instead of only the first row, are given. In Exercise 3 we take you through the steps in finding the formulas for this. This enables us to write an algorithm for multiplying with an MRA matrix based on symmetric filters. You will be spared writing this algorithm, and you can assume that the function `y=rowsymmmratrans(a0,a1,x)` performs this task. Here `x` represents the vector we want to multiply with the MRA-matrix, and `y` represents the result. The parameters `a0,a1` require som explanation: They represent the filter coefficients for the filters which have the same first/second rows as the MRA-matrix, respectively. This is most easily explained for the MRA-matrix for the DWT, since here these filters are H_0 and H_1. Since H_0, H_1 are symmetric filters, they can be written on the form

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

Since the negative-indexed filter coefficients are equal to the positive-indexed filter coefficients, there is no need to provide them to the function `rowsymmmratrans`. It will therefore be assumed that the input to `rowsymmmratrans` is

$$\texttt{a0} = (h_{0,0}, h_{0,1}, \ldots, h_{0,k_0})$$
$$\texttt{a1} = (h_{1,0}, h_{1,1}, \ldots, h_{1,k_1}),$$

i.e. only the filter coefficients with index ≥ 0 are included in `a0` and `a1`. For the IDWT the situation is different: here only the columns of $P_{\boldsymbol{\phi}_m \leftarrow \mathcal{C}_m}$ are given (and in terms of the columns of G_0, G_1). We therefore first need a step where we translate the column representation of $P_{\boldsymbol{\phi}_m \leftarrow \mathcal{C}_m}$ to a row representation of the same matrix. This is a straightforward task, however a bit tedious. You will be spared writing this code also, and can take for granted that the function `[a0,a1]=changecolumnrows(g0,g1)` performs this task. The parameters `g0,g1` are best explained in terms of the IDWT: If G_0, G_1 are the symmetric filters in the IDWT, they are first written on the form

$$G_0 = \{g_{0,-l_1}, \ldots, g_{0,-1}\underline{g_{0,0}}, g_{0,1}, \ldots, g_{0,l_0}\}$$
$$G_1 = \{g_{1,-l_1}, \ldots, g_{1,-1}\underline{g_{1,0}}, g_{1,1}, \ldots, g_{1,l_1}\},$$

and we write as above

$$\texttt{g0} = (g_{0,0}, g_{0,1}, \ldots, g_{0,l_0})$$
$$\texttt{g1} = (g_{1,0}, g_{1,1}, \ldots, g_{1,l_1}).$$

The vectors `a0,a1` returned by `changecolumnrows` are then the row representation which is accepted by the unction `rowsymmmratrans`.

Once we have the functions `rowsymmmratrans` and `changecolumnrows`, the DWT and the IDWT can easily be computed. Exercises 4 and 5 will talk you through the steps in this process. There is a very good reason for encapsulating the filtering operations inside the function `rowsymmmratrans`: it can hide the details of highly optimized implementations of different types of filters.

Example 5.50. In Exercise 8 you will be asked to implement a function `playDWTfilterslower` which plays the low-resolution approximations to our audio test file, for any type of wavelet, using the functions we have described. With this function we can play the result for all the wavelets we have considered up to now, in succession, and at a given resolution, with the following code:

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

The call to `playDWTlower` first plays the result, using the Haar wavelet. The code then moves on to the piecewise linear wavelet. From Equation (5.66) we first see that

$$\texttt{h0} = (h_{0,0}, h_{0,1}, \ldots, h_{0,k_0}) = (\sqrt{2}) \tag{5.70}$$

$$\texttt{h1} = (h_{1,0}, h_{1,1}, \ldots, h_{1,k_1}) = (\sqrt{2}, -\sqrt{2}/2), \tag{5.71}$$

and from Equation (5.65) we see that

$$\texttt{g0} = (g_{0,0}, g_{0,1}, \ldots, g_{0,l_0}) = (1/\sqrt{2}, 1/(2\sqrt{2})) \tag{5.72}$$

$$\texttt{g1} = (g_{1,0}, g_{1,1}, \ldots, g_{1,l_1}) = (1/\sqrt{2}). \tag{5.73}$$

These explain the parameters to the call to `playDWTfilterslower` for the piecewise linear wavelet. The code then moves to the alternative piecewise linear wavelet. From Equation (5.69) we see that

$$\texttt{h0} = (h_{0,0}, h_{0,1}, \ldots, h_{0,k_0}) = (3\sqrt{2}/4, \sqrt{2}/4, -\sqrt{2}/8)$$

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

and from Equation (5.67) we see that

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

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

Exercises for Section 5.6

Ex. 1 — Find two symmetric filters, so that the corresponding MRA-matrix, constructed with alternating rows from these two filters, is not symmetric.

Ex. 2 — Assume that an MRA-matrix is symmetric. Show that the corresponding filters are also symmetric.

Ex. 3 — Assume that G is an MRA-matrix where the rows repeated are $\boldsymbol{a}^{(0)}$, $\boldsymbol{a}^{(1)}$ (symmetric around 0). Assume that their supports are $[-E_0, E_0]$ and $[-E_1, E_1]$, respectively. Show that $y_n = (G\boldsymbol{x})_n$ can be computed as follows, depending on n:

a. n even: The formulas (3.33)-(3.35) you derived in Exercise 3.4.3 can be used, with $T_{0,k}$ replaced with $\boldsymbol{a}^{(0)}$, E replaced by E_0.
b. n odd: The formulas (3.33)-(3.35) you derived in Exercise 3.4.3 can be used, with $T_{0,k}$ replaced with $\boldsymbol{a}^{(1)}$, E replaced by E_1.

Ex. 4 — Write a function

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

which takes a signal `x` of length N, computes the transforms $F_1, ..., F_{m-1}$, and computes the coordinate of `x` in the basis $V_0 \oplus W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}$. Your function should call the function `rowsymmmratrans` to achieve this. Remember that you have to sort the even and odd outputs after calling that function, before you apply the next step. You can assume that the signal `x` has length 2^m.

Ex. 5 — Write a function

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

which recovers the coordinates in the basis V_m from those in the basis $V_0 \oplus W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}$. Your function should call the function `changecolumnrows`, and the function `rowsymmmratrans`.

Ex. 6 — In this exercise we will practice setting up the parameters `h0,h1,g0,g1` which are used in the calls to `DWTImpl` and `IDWTImpl`.

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

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

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

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

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

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

Ex. 7 — Let us also practice on writing down the change of coordinate matrices from the parameters `h0,h1,g0,g1`.

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

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

Ex. 8 — Write a function

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

which reimplements the function `playDWTlower` from Exercise 5.3.9 so that it takes as input the positive parts of the four different filters as in Example 5.50. Listen to the result using the different wavelets we have encountered and for different m, using the code from Example 5.50. Can you hear any difference from the Haar wavelet? If so, which wavelet gives the best sound quality?

Ex. 9 — In this exercise we will change the code in Example 5.50 so that it instead only plays the contribution from the detail spaces (i.e. $W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}$).

a. Reimplement the function you made in Exercise 8 so that it instead plays the contribution from the detail spaces. Call the new function `playDWTfilterslowerdifference`.

b. In Exercise 5.3.11 we implemented a function `playDWTlowerdifference` for listening to the detail/error when the Haar wavelet is used. In the function `playDWTall` from Example 5.50, replace `playDWTlower` and `playDWTfilterslower` with `playDWTlowerdifference` and `playDWTfilterslowerdifference`. Describe the sounds you hear for different m. Try to explain why the sound seems to get louder when you increase m.

Ex. 10 — Let us return to the piecewise linear wavelet from Exercise 5.5.2.

a. With $\hat{\psi}$ as defined as in Exercise 5.5.2 b., compute the coordinates of $\hat{\psi}$ in the basis $\boldsymbol{\phi}_1$ (i.e. $[\hat{\psi}]_{\boldsymbol{\phi}_1}$) with $N = 8$, i.e. compute the IDWT of

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

which is the coordinate vector you computed in Exercise 5.5.2 d.. For this, you should use the function `IDWTImpl` from Exercise 5, with parameters being the filters G_0, G_1, given as described by `g0,g1` by equations (5.72)-(5.73) in Example 5.50.

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

c. Write code which uses Equation (5.68) to find H_0, H_1 from G_0, G_1, and state these filters with our compact filter notation. Also, state the forms

`h0,h1`, which should be used in calls to `DWTImpl` for our new wavelet. These replace the forms from equations (5.70)-(5.71) in Example 5.50, which we found for the first piecewise linear wavelet.
Hint: Note that the filter G_0 is unchanged from that of the first piecewise linear wavelet (since ϕ is unchanged when compared to the other wavelets for piecewise linear functions).

d. The filters you have found above should be symmetric, so that we can follow the procedure from Example 5.50 to listen to sound which has been wavelet-transformed by this wavelet. Write a program which plays our audio test file as in Example 5.50 for $m = 1, 2, 3, 4$ (i.e. plays the part in V_0), as well as the difference as in Exercise 9 (i.e. play the part from $W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}$), where the new filters you have found are used. Listen to the sounds.

Ex. 11 — Repeat the previous exercise for the Haar wavelet as in exercise 3, and plot the corresponding frequency responses for $k = 2, 4, 6$.

5.7 Summary

We started this chapter by motivating the theory of wavelets as a different function approximation scheme, which solved some of the shortcomings of Fourier series. While one approximates functions with trigonometric functions in Fourier theory, with wavelets one instead approximates a function in several stages, where one at each stage attempts to capture information at a given resolution, using a function prototype. We first considered the Haar wavelet, which is a function approximation scheme based on piecewise constant functions. We then moved on to a scheme with piecewise linear functions, where we saw that we had several degrees of freedom in constructing wavelets. Just as the DFT and the DCT, we interpreted a wavelet transformation as a change of basis, and found that the corresponding change of coordinate matrices had a particular form, which we studied. We denoted the change of basis in a wavelet transformation by the Discrete Wavelet Transform (DWT), and we showed how we could interpret and implement the DWT in terms of filters in such a way that a wide range of usable wavelets could be used as input to this implementation. We will use this implementation in the coming sections, in order to analyze images.

Chapter 6

Digital images

The theory on wavelets has been presented as a one-dimensional theory upto now. Images, however, are two-dimensional by nature. This poses another challenge, contrary to the case for sound. In the next chapter we will establish the mathematics to handle this, but first we will present some basics on images. Images are a very important type of digital media, and we will go through how they can be represented and manipulated with simple mathematics. This is useful general knowledge for anyone who has a digital camera and a computer, but for many scientists, it is an essential tool. In astrophysics, data from both satellites and distant stars and galaxies is collected in the form of images, and information extracted from the images with advanced image processing techniques. Medical imaging makes it possible to gather different kinds of information in the form of images, even from the inside of the body. By analysing these images it is possible to discover tumours and other disorders.

6.1 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.

6.1.1 Light

Fact 6.1 (What is light?)**.** Light is electromagnetic radiation with wavelengths in the range 400–700 nm (1 nm is 10^{-9} m): Violet has wavelength 400 nm and red has wavelength 700 nm. White light contains roughly equal amounts of all wave lengths.

Other examples of electromagnetic radiation are gamma radiation, ultraviolet and infrared radiation and radio waves, and all electromagnetic radiation travel at the speed of light (3×10^8 m/s). Electromagnetic radiation consists of waves

and may be reflected and refracted, just like sound waves (but sound waves are not electromagnetic waves).

We can only see objects that emit light, and there are two ways that this can happen. The object can emit light itself, like a lamp or a computer monitor, or it reflects light that falls on it. An object that reflects light usually absorbs light as well. If we perceive the object as red it means that the object absorbs all light except red, which is reflected. An object that emits light is different; if it is to be perceived as being red it must emit only red light.

6.1.2 Digital output media

Our focus will be on objects that emit light, for example a computer display. A computer monitor consists of a rectangular array of small dots which emit light. In most technologies, each dot is really three smaller dots, and each of these smaller dots emit red, green and blue light. If the amounts of red, green and blue is varied, our brain merges the light from the three small light sources and perceives light of different colours. In this way the colour at each set of three dots can be controlled, and a colour image can be built from the total number of dots.

It is important to realise that it is possible to generate most, but not all, colours by mixing red, green and blue. In addition, different computer monitors use slightly different red, green and blue colours, and unless this is taken into consideration, colours will look different on the two monitors. This also means that some colours that can be displayed on one monitor may not be displayable on a different monitor.

Printers use the same principle of building an image from small dots. On most printers however, the small dots do not consist of smaller dots of different colours. Instead as many as 7–8 different inks (or similar substances) are mixed to the right colour. This makes it possible to produce a wide range of colours, but not all, and the problem of matching a colour from another device like a monitor is at least as difficult as matching different colours across different monitors.

Video projectors builds an image that is projected onto a wall. The final image is therefore a reflected image and it is important that the surface is white so that it reflects all colours equally.

The quality of a device is closely linked to the density of the dots.

Fact 6.2 (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).

6.1.3 Digital input media

The two most common ways to acquire digital images is with a digital camera or a scanner. A scanner essentially takes a photo of a document in the form of a rectangular array of (possibly coloured) dots. As for printers, an important measure of quality is the number of dots per inch.

Fact 6.3. The resolution of a scanner usually varies in the range 75 dpi to 9600 dpi, and the colour is represented with up to 48 bits per dot.

For digital cameras it does not make sense to measure the resolution in dots per inch, as this depends on how the image is printed (its size). Instead the resolution is measured in the number of dots recorded.

Fact 6.4. The number of pixels recorded by a digital camera usually varies in the range 320×240 to 6000×4000 with 24 bits of colour information per pixel. The total number of pixels varies in the range 76 800 to 24 000 000 (0.077 megapixels to 24 megapixels).

For scanners and cameras it is easy to think that the more dots (pixels), the better the quality. Although there is some truth to this, there are many other factors that influence the quality. The main problem is that the measured colour information is very easily polluted by noise. And of course high resolution also means that the resulting files become very big; an uncompressed 6000×4000 image produces a 72 MB file. The advantage of high resolution is that you can magnify the image considerably and still maintain reasonable quality.

6.1.4 Definition of digital image

We have already talked about digital images, but we have not yet been precise about what it is. From a mathematical point of view, an image is quite simple.

Fact 6.5 (Digital image)**.** A digital image P is a rectangular array of *intensity values* $\{p_{i,j}\}_{i,j=1}^{m,n}$. For grey-level images, the value $p_{i,j}$ is a single number, while for colour images each $p_{i,j}$ is a vector of three or more values. If the image is recorded in the rgb-model, each $p_{i,j}$ is a vector of three values,

$$p_{i,j} = (r_{i,j}, g_{i,j}, b_{i,j}),$$

that denote the amount of red, green and blue at the point (i, j).

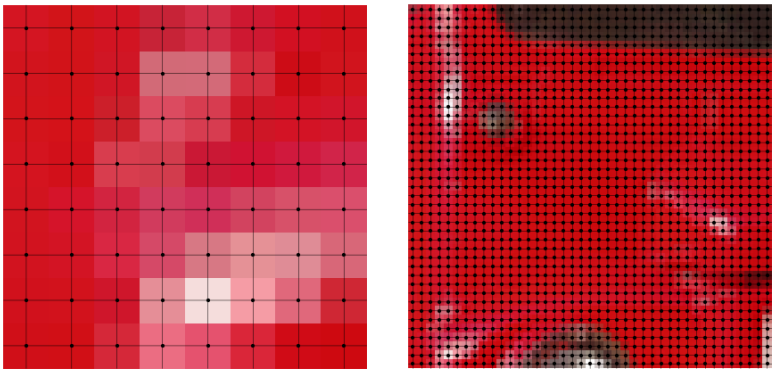
Note that, when refering to the coordinates (i, j) in an image, i will refer to row index, j to column index, in the same was as for matrices. In particular, the top row in the image have coordinates $\{(0, j)\}_{j=0}^{N-1}$, while the left column in the image has coordinates $\{(i, 0)\}_{i=0}^{M-1}$. With this notation, the dimension of the image is $M \times N$. The value $p_{i,j}$ gives the colour information at the point (i, j).

(a)

(b)

(c)

Figure 6.1: Different version of the same image; black and white (a), grey-level (b), and colour (c).



(a) (b)

Figure 6.2: Two excerpt of the colour image in figure 6.1. The dots indicate the position of the points (i, j).

It is important to remember that there are many formats for this. The simplest case is plain black and white images in which case $p_{i,j}$ is either 0 or 1. For grey-level images the intensities are usually integers in the range 0–255. However, we will assume that the intensities vary in the interval $[0, 1]$, as this sometimes simplifies the form of some mathematical functions. For colour images there are many different formats, but we will just consider the rgb-format mentioned in the fact box. Usually the three components are given as integers in the range 0–255, but as for grey-level images, we will assume that they are real numbers in the interval $[0, 1]$ (the conversion between the two ranges is straightforward, see section 6.11 below). Figure 6.1 shows an image in different formats.

Fact 6.6. In these notes the intensity values $p_{i,j}$ are assumed to be real numbers in the interval $[0, 1]$. For colour images, each of the red, green, and blue intensity values are assumed to be real numbers in $[0, 1]$.

If we magnify a small part of the colour image in figure 6.1, we obtain the image in figure 6.2 (the black lines and dots have been added). A we can see, the pixels have been magnified to big squares. This is a standard representation used by many programs — the actual shape of the pixels will depend on the output medium. Nevertheless, we will consider the pixels to be square, with integer coordinates at their centres, as indicated by the grids in figure 6.2.

Fact 6.7 (Shape of pixel)**.** The pixels of an image are assumed to be square with sides of length one, with the pixel with value $p_{i,j}$ centred at the point (i, j).

Figure 6.3: The grey-level image in figure 6.1 plotted as a surface. The height above the (x, y)-plane is given by the intensity value.

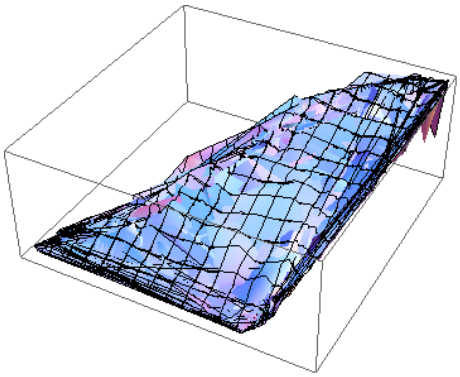


Figure 6.4: A colour image viewed as a parametric surface in space.

6.1.5 Images as surfaces

Recall from your previous calculus courses that a function $f : \mathbb{R}^2 \mapsto \mathbb{R}$ can be visualised as a surface in space. A grey-level image is almost on this form. If we define the set of integer pairs by

$$\mathbb{Z}_{m,n} = \big\{(i,j) \mid 1 \leq i \leq m \text{ and } 1 \leq j \leq n\big\},$$

we can consider a grey-level image as a function $P : \mathbb{Z}_{m,n} \mapsto [0,1]$. In other words, we may consider an image to be a sampled version of a surface with the intensity value denoting the height above the (x,y)-plane, see figure 6.3.

Fact 6.8 (Grey-level image as a surface)**.** Let $P = (p)_{i,j=1}^{m,n}$ be a grey-level image. Then P can be considered a sampled version of the piecewise constant surface

$$F_P : [1/2, m+1/2] \times [1/2, n+1/2] \mapsto [0,1]$$

which has the constant value $p_{i,j}$ in the square (pixel)

$$[i-1/2, i+1/2] \times [j-1/2, j+1/2]$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, n$.

What about a colour image P? Then each $p_{i,j} = (r_{i,j}, g_{i,j}, b_{i,j})$ is a triple of numbers so we have a mapping

$$P : \mathbb{Z}_{m,n} \mapsto \mathbb{R}^3.$$

If we examine this expression, we see that this corresponds to a sampled version of a parametric surface if we consider the colour values $(r_{i,j}, g_{i,j}, b_{i,j})$ to be x-, y-, and z-coordinates. This may be useful for computations in certain settings, but visually it does not make much sense, see figure 6.4

6.2 Operations on images

Images are two-dimensional arrays of numbers, contrary to the sound signals we considered in the previous section. In this respect it is quite obvious that we can manipulate an image by performing mathematical operations on the numbers. In this section we will consider some of the simpler operations. In later sections we will go through more advanced operations, and explain how the theory for these can be generalized from the corresponding theory for one-dimensional (sound) signals (which we wil go through first).

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

6.2.1 Images and MATLAB

An image can also be thought of as a matrix, by associating each pixel with an element in a matrix. The matrix indices thus correspond to positions in the pixel grid. Black and white images correspond to matrices where the elements are natural numbers between 0 and 255. To store a colour image, we need 3 matrices, one for each colour component. This enables us to use linear algebra packages, such as MATLAB, on order to work with images. After we now have made the connection with matrices, we can create images from mathematical formulas, just as we could with sound in the previuos sections. But what we also need before we go through operations on images, is, as in the sections on sound, means of reading an image from a file so that its contents are accessible as a matrix, and write images represented by a matrix which we have constructed ourself to file. Reading a function from file can be done with help of the function `imread`. If we write

```
A = double(imread('filename.fmt','fmt'));
```

the image with the given path and format is read, and stored in the matrix `A`. 'fmt' can be 'jpg','tif', 'gif', 'png',... You should consult the MATLAB help pages to see which formats are supported. After the call to `imread`, we have a matrix where the entries represent the pixel values, and of integer data type (more precisely, the data type `uint8` in Matlab). To perform operations on the image, we must first convert the entries to the data type `double`. This is done with a call to the Matlab function `double`. Similarly, the function `imwrite` can be used to write the image represented by a matrix to file. If we write

```
imwrite(uint8(A), 'filename.fmt','fmt')
```

the image represented by the matrix `A` is written to the given path, in the given format. Before the image is written to file, you see that we have converted the matrix values back to the integer data type with the help of the function `uint8`. In other words: `imread` and `imwrite` both assume integer matrix entries, while operations on matrices assume double matrix entries. If you want to print images you have created yourself, you can use this function first to write the image

to a file, and then send that file to the printer using another program. Finally, we need an alternative to playing a sound, namely displaying an image. The function `imageview(A)` displays the matrix `A` as an image in a separate window. Unfortunately, you can't print the image from the menus in this window.

The following examples go through some much used operations on images.

Example 6.9 (Normalising the intensities)**.** We have assumed that the intensities all lie in the interval $[0, 1]$, but as we noted, many formats in fact use integer values in the range 0–255. And as we perform computations with the intensities, we quickly end up with intensities outside $[0, 1]$ even if we start out with intensities within this interval. We therefore need to be able to *normalise* the intensities. This we can do with the simple linear function

$$g(x) = \frac{x - a}{b - a}, \quad a < b,$$

which maps the interval $[a, b]$ to $[0, 1]$. A simple case is mapping $[0, 255]$ to $[0, 1]$ which we accomplish with the scaling $g(x) = x/255$. More generally, we typically perform computations that result in intensities outside the interval $[0, 1]$. We can then compute the minimum and maximum intensities $p_{\min}$ and $p_{\max}$ and map the interval $[p_{\min}, p_{\max}]$ back to $[0, 1]$. Below we have shown a function `mapto01` which achieves this task.

```
function newimg=mapto01(img)
  minval = min(min(img));
  maxval = max(max(img));
  newimg = (img - minval)/(maxval-minval);
```

Several examples of using this function will be shown below.

Example 6.10 (Extracting the different colours)**.** If we have a colour image $P = (r_{i,j}, g_{i,j}, b_{i,j})_{i,j=1}^{m,n}$ it is often useful to manipulate the three colour components separately as the three images

$$P_r = (r_{i,j})_{i,j=1}^{m,n}, \quad P_r = (g_{i,j})_{i,j=1}^{m,n}, \quad P_r = (b_{i,j})_{i,j=1}^{m,n}.$$

As an example, let us first see how we can produce three separate images, showing the R,G, and B colour components, respectively. Let us take the image `bus-small-rgb.png` used in Figure 6.1. When the image is read, three matrices are returned, one for each colour component, and we can generate new files for the different colour components with the following code:

```
img=double(imread('bus-small-rgb.png','png'));
newimg=zeros(size(img));
newimg(:,:,1)=img(:,:,1);
imwrite(uint8(newimg),'gr.png','png');
newimg=zeros(size(img));
newimg(:,:,2)=img(:,:,2);
imwrite(uint8(newimg),'ggg.png','png');
```

(a) (b) (c)

Figure 6.5: The red (a), green (b), and blue (c) components of the colour image in Figure 6.1.

```
newimg=zeros(size(img));
newimg(:,:,3)=img(:,:,3);
imwrite(uint8(newimg),'gb.png','png');
```

The resulting image files are shown in Figure 6.5.

Example 6.11 (Converting from colour to grey-level)**.** If we have a colour image we can convert it to a grey-level image. This means that at each point in the image we have to replace the three colour values (r, g, b) by a single value p that will represent the grey level. If we want the grey-level image to be a reasonable representation of the colour image, the value p should somehow reflect the intensity of the image at the point. There are several ways to do this.

It is not unreasonable to use the largest of the three colour components as a measure of the intensity, i.e, to set $p = \max(r, g, b)$. The result of this can be seen in Figure 6.6(a).

An alternative is to use the sum of the three values as a measure of the total intensity at the point. This corresponds to setting $p = r + g + b$. Here we have to be a bit careful with a subtle point. We have required each of the r, g and b values to lie in the range $[0, 1]$, but their sum may of course become as large as 3. We also require our grey-level values to lie in the range $[0, 1]$ so after having computed all the sums we must normalise as explained above. The result can be seen in Figure 6.6(b).

A third possibility is to think of the intensity of (r, g, b) as the length of the colour vector, in analogy with points in space, and set $p = \sqrt{r^2 + g^2 + b^2}$. Again, we may end up with values in the range $[0, \sqrt{3}]$ so we have to normalise like we did in the second case. The result is shown in Figure 6.6(c).

Let us sum this up as an algorithm.

A colour image $P = (r_{i,j}, g_{i,j}, b_{i,j})_{i,j=1}^{m,n}$ can be converted to a grey level image $Q = (q_{i,j})_{i,j=1}^{m,n}$ by one of the following three operations:

1. Set $q_{i,j} = \max(r_{i,j}, g_{i,j}, b_{i,j})$ for all i and j.
2. (a) Compute $\hat{q}_{i,j} = r_{i,j} + g_{i,j} + b_{i,j}$ for all i and j.
   (b) Transform all the values to the interval $[0, 1]$ by setting
   $$q_{i,j} = \frac{\hat{q}_{i,j}}{\max_{k,l} \hat{q}_{k,l}}.$$
3. (a) Compute $\hat{q}_{i,j} = \sqrt{r_{i,j}^2 + g_{i,j}^2 + b_{i,j}^2}$ for all i and j.
   (b) Transform all the values to the interval $[0, 1]$ by setting
   $$q_{i,j} = \frac{\hat{q}_{i,j}}{\max_{k,l} \hat{q}_{k,l}}.$$

This can be implemented by using most of the code from the previous example, and replacing with the lines

```
newimg1=max(img,[],3);
newvals=img(:,:,1)+img(:,:,2)+img(:,:,3);
newimg2=newvals/max(max(newvals))*255;
newvals=sqrt(img(:,:,1).^2+img(:,:,2).^2+img(:,:,3).^2);
newimg3=newvals/max(max(newvals))*255;
```

respectively. In practice one of the last two methods are usually preferred, perhaps with a preference for the last method, but the actual choice depends on the application. These resuliting images are visualised as grey-level images in Figure 6.5.

Example 6.12 (Computing the negative image)**.** In film-based photography a negative image was obtained when the film was developed, and then a positive image was created from the negative. We can easily simulate this and compute a negative digital image.

Suppose we have a grey-level image $P = (p_{i,j})_{i,j=1}^{m,n}$ with intensity values in the interval $[0, 1]$. Here intensity value 0 corresponds to black and 1 corresponds to white. To obtain the negative image we just have to replace an intensity p by its 'mirror value' $1 - p$.

Fact 6.13 (Negative image)**.** Suppose the grey-level image $P = (p_{i,j})_{i,j=1}^{m,n}$ is given, with intensity values in the interval $[0, 1]$. The negative image $Q = (q_{i,j})_{i,j=1}^{m,n}$ has intensity values given by $q_{i,j} = 1 - p_{i,j}$ for all i and j.

(a) (b) (c)

Figure 6.6: Alternative ways to convert the colour image in Figure 6.1 to a grey level image. In (a) each colour triple has been replaced by its maximum, in (b) each colour triple has been replaced by its sum and the result mapped to $[0, 1]$, while in (c) each triple has been replaced by its length and the result mapped to $[0, 1]$.

This is also easily translated to code as above. The resulting image is shown in Figure 6.7.

Example 6.14 (Increasing the contrast)**.** A common problem with images is that the contrast often is not good enough. This typically means that a large proportion of the grey values are concentrated in a rather small subinterval of $[0, 1]$. The obvious solution to this problem is to somehow spread out the values. This can be accomplished by applying a function f to the intensity values, i.e., new intensity values are computed by the formula

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

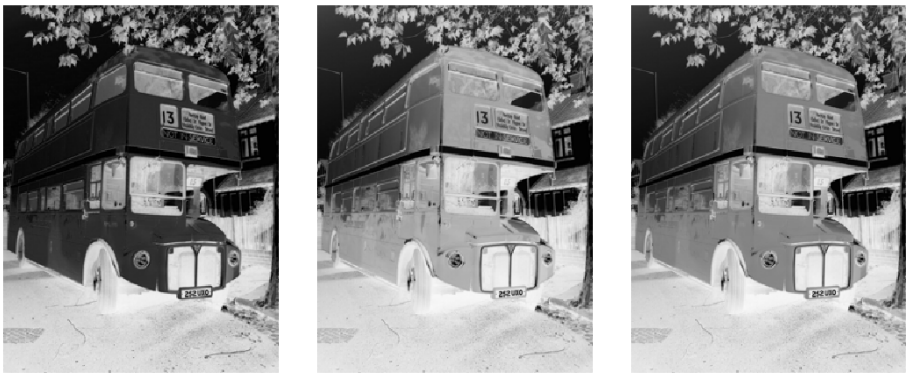
for all i and j. If we choose f so that its derivative is large in the area where many intensity values are concentrated, we obtain the desired effect.

Figure 6.8 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

$$f_n(x) = \frac{\arctan\big(n(x - 1/2)\big)}{2\arctan(n/2)} + \frac{1}{2}. \tag{6.1}$$

For any $n \neq 0$ these functions satisfy the conditions $f_n(0) = 0$ and $f_n(1) = 1$. The three functions in figure 6.8a correspond to $n = 4$, 10, and 100.

Functions of the kind shown in figure 6.8b have a large derivative near $x = 0$ and will therefore increase the contrast in an image with a large proportion of



(a) (b) (c)

Figure 6.7: The negative versions of the corresponding images in figure 6.6.

small intensity values, i.e., very dark images. The functions are given by

$$g_\epsilon(x) = \frac{\ln(x+\epsilon) - \ln\epsilon}{\ln(1+\epsilon) - \ln\epsilon}, \tag{6.2}$$

and the ones shown in the plot correspond to $\epsilon = 0.1$, 0.01, and 0.001.

In figure 6.8c the middle function in (a) has been applied to the image in figure 6.6c. Since the image was quite well balanced, this has made the dark areas too dark and the bright areas too bright. In figure 6.8d the function in (b) has been applied to the same image. This has made the image as a whole too bright, but has brought out the details of the road which was very dark in the original.

Observation 6.15. Suppose a large proportion of the intensity values $p_{i,j}$ of a grey-level image P lie in a subinterval I of $[0,1]$. Then the contrast of the image can be improved by computing new intensities $\hat{p}_{i,j} = f(p_{,j})$ where f is a function with a large derivative in the interval I.

Increasing the contrast is easy to implement. The following function has been used to generate the image in Figure 6.8(d):

```
function newimg=contrastadjust(img)
  epsilon = 0.001;  % Try also 0.1, 0.01, 0.001
  newimg = img/255; % Maps the pixel values to [0,1]
  newimg = (log(newimg+epsilon) - log(epsilon))/...
           (log(1+epsilon)-log(epsilon));
  newimg = newimg*255; % Maps the values back to [0,255]
```

We will see more examples of how the contrast in an image can be enhanced when we try to detect edges below.

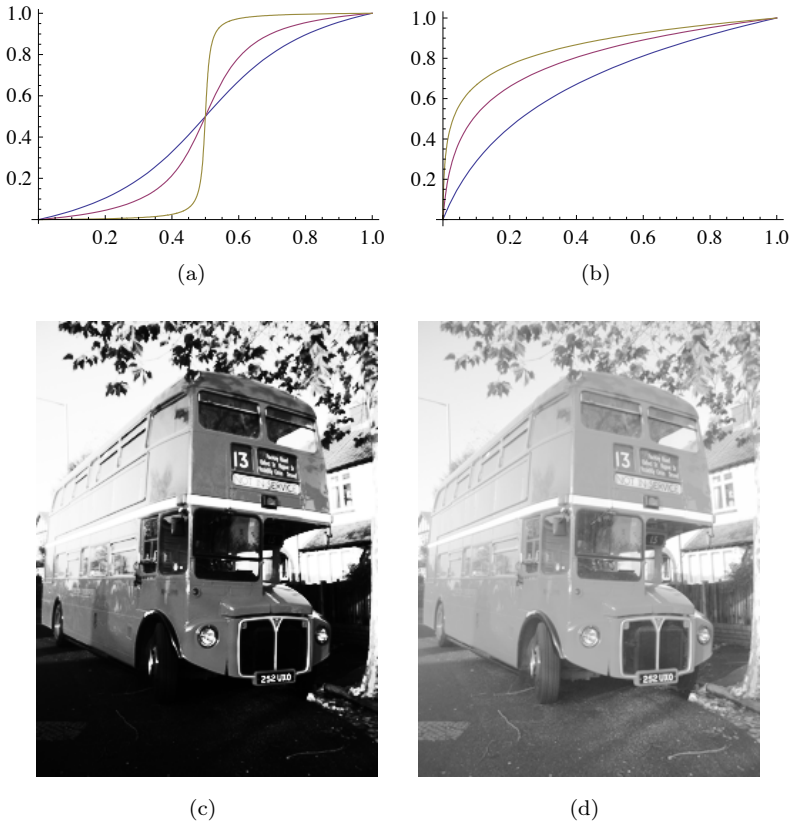


Figure 6.8: The plots in (a) and (b) show some functions that can be used to improve the contrast of an image. In (c) the middle function in (a) has been applied to the intensity values of the image in figure 6.6c, while in (d) the middle function in (b) has been applied to the same image.

(a) (b) (c)

Figure 6.9: The images in (b) and (c) show the effect of smoothing the image in (a).

Example 6.16 (Smoothing an image)**.** When we considered filtering of digital sound, we observed that replacing each sample of a sound by an average of the sample and its neighbours dampened the high frequencies of the sound. We can do a similar operation on images.

Consider the array of numbers given by

$$\frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}. \tag{6.3}$$

We can smooth an image with this array by placing the centre of the array on a pixel, multiplying the pixel and its neighbours by the corresponding weights, summing up and dividing by the total sum of the weights. More precisely, we would compute the new pixels by

$$\hat{p}_{i,j} = \frac{1}{16}\big(4p_{i,j} + 2(p_{i,j-1} + p_{i-1,j} + p_{i+1,j} + p_{i,j+1}) + p_{i-1,j-1} + p_{i+1,j-1} + p_{i-1,j+1} + p_{i+1,j+1}\big).$$

Since the weights sum to one, the new intensity value $\hat{p}_{i,j}$ is a weighted average of the intensity values on the right. The array of numbers in (6.3) is in fact an example of a computational molecule. As in the section on sound, we could have used equal weights for all pixels, but it seems reasonable that the weight of a pixel should be larger the closer it is to the centre pixel. For the onedimensional case on sound, we used the values of Pascal's triangle here, since these weights are known to give a very good smoothing effect. We will return to how we can generalize the use of Pascal's triangle to obtain computational molecules for use in images.

A larger filter is given by the array

$$\frac{1}{4096}\begin{pmatrix} 1 & 6 & 15 & 20 & 15 & 6 & 1 \\ 6 & 36 & 90 & 120 & 90 & 36 & 6 \\ 15 & 90 & 225 & 300 & 225 & 90 & 15 \\ 20 & 120 & 300 & 400 & 300 & 120 & 20 \\ 15 & 90 & 225 & 300 & 225 & 90 & 15 \\ 6 & 36 & 90 & 120 & 90 & 36 & 6 \\ 1 & 6 & 15 & 20 & 15 & 6 & 1 \end{pmatrix}. \tag{6.4}$$

These numbers are taken from row six of Pascal's triangle. More precisely, the value in row k and column l is given by the product $\binom{6}{k}\binom{6}{l}$. The scaling $1/4096$ comes from the fact that the sum of all the numbers in the table is $2^{6+6} = 4096$.

The result of applying the two filters in (6.3) and (6.4) to our greyscale-image is shown in Figure 6.9(b) and -(c) respectively. The smoothing effect is clearly visible.

Observation 6.17. An image P can be smoothed out by replacing the intensity value at each pixel by a weighted average of the intensity at the pixel and the intensity of its neighbours.

It is straightforward to write a function which performs smoothing. Assume that the image is stored as the matrix `img`, and the computational molecule is stored as the matrix `compmolecule`. The following function will return the smoothed image:

```
function newimg=smooth(img,compmolecule)
  [m,n]=size(img);
  [k,k1] = size(compmolecule); % We need k==k1, and odd
  sc = (k+1)/2;
  for m1=1:m
    for n1=1:n
      slidingwdw = zeros(k,k);
        % slidingwdw is the part of the picture which
        % compmolecule is applied to pixel (m1,n1)
      slidingwdw(max(sc+1-m1,1):min(sc+m-m1,2*sc-1) , ...
                 max(sc+1-n1,1):min(sc+n-n1,2*sc-1)) = ...
      img(max(1,m1-(sc-1)):min(m,m1+(sc-1)) , ...
          max(1,n1-(sc-1)):min(n,n1+(sc-1)));
      newimg(m1,n1) = sum(sum(compmolecule .* slidingwdw));
    end
  end
```

What makes this code difficult to write is the fact that the computational molecule may extend outside the borders of the image, when we are close to these borders. With this function, the first smoothing above can be performed by writing

```
smooth(img,(1/16)*[ 1 2 1; 2 4 2; 1 2 1]);
```

Example 6.18 (Detecting edges)**.** The final operation on images we are going to consider is edge detection. An edge in an image is characterised by a large change in intensity values over a small distance in the image. For a continuous function this corresponds to a large derivative. An image is only defined at isolated points, so we cannot compute derivatives, but we have a perfect situation for applying numerical differentiation. Since a grey-level image is a scalar function of two variables, numerical differentiation techniques can be applied.

Partial derivative in x-direction. Let us first consider computation of the partial derivative $\partial P/\partial x$ at all points in the image. Note first that it is the second coordinate in an image which refers to the x-direction we are used to from plotting functions. This means that the familiar approximation for the partial derivative takes the form

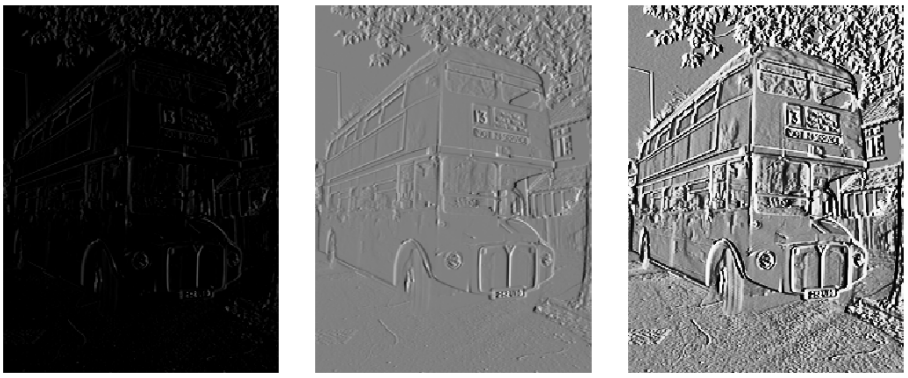
$$\frac{\partial P}{\partial x}(i,j) = \frac{p_{i,j+1} - p_{i,j-1}}{2}, \tag{6.5}$$

where we have used the convention $h = 1$ which means that the derivative is measured in terms of 'intensity per pixel'. We can run through all the pixels in the image and compute this partial derivative, but have to be careful for $j = 1$ and $j = m$ where the formula refers to non-existing pixels. We will adapt the simple convention of assuming that all pixels outside the image have intensity 0. The result is shown in figure 6.10a.

This image is not very helpful since it is almost completely black. The reason for this is that many of the intensities are in fact negative, and these are just displayed as black. More specifically, the intensities turn out to vary in the interval $[-0.424, 0.418]$. We therefore normalise and map all intensities to $[0, 1]$. The result of this is shown in (b). The predominant colour of this image is an average grey, i.e, an intensity of about 0.5. To get more detail in the image we therefore try to increase the contrast by applying the function f_{50} in equation 6.1 to each intensity value. The result is shown in figure 6.10c which does indeed show more detail.

It is important to understand the colours in these images. We have computed the derivative in the x-direction, and we recall that the computed values varied in the interval $[-0.424, 0.418]$. The negative value corresponds to the largest average decrease in intensity from a pixel $p_{i-1,j}$ to a pixel $p_{i+1,j}$. The positive value on the other hand corresponds to the largest average increase in intensity. A value of 0 in figure 6.10a corresponds to no change in intensity between the two pixels.

When the values are mapped to the interval $[0, 1]$ in figure 6.10b, the small values are mapped to something close to 0 (almost black), the maximal values are mapped to something close to 1 (almost white), and the values near 0 are mapped to something close to 0.5 (grey). In figure 6.10c these values have just been emphasised even more.



(a) (b) (c)

Figure 6.10: The image in (a) shows the partial derivative in the x-direction for the image in 6.6. In (b) the intensities in (a) have been normalised to $[0, 1]$ and in (c) the contrast as been enhanced with the function f_{50}, equation 6.1.

Figure 6.10c tells us that in large parts of the image there is very little variation in the intensity. However, there are some small areas where the intensity changes quite abruptly, and if you look carefully you will notice that in these areas there is typically both black and white pixels close together, like down the vertical front corner of the bus. This will happen when there is a stripe of bright or dark pixels that cut through an area of otherwise quite uniform intensity.

Since we display the derivative as a new image, the denominator is actually not so important as it just corresponds to a constant scaling of all the pixels; when we normalise the intensities to the interval $[0, 1]$ this factor cancels out.

We sum up the computation of the partial derivative by giving its computational molecule.

Observation 6.19. 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

$$\frac{1}{2}\begin{pmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}. \tag{6.6}$$

As we remarked above, the factor $1/2$ can usually be ignored. We have included the two rows of 0s just to make it clear how the computational molecule is to be interpreted; it is obviously not necessary to multiply by 0.

Partial derivative in y-direction. The partial derivative $\partial P/\partial y$ can be computed analogously to $\partial P/\partial x$. Note that the positive direction of this axis

in an image is opposite to the direction of the y-axis we use when plotting functions.

Observation 6.20. 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

$$\frac{1}{2}\begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}. \tag{6.7}$$

The result is shown in figure 6.12b. The intensities have been normalised and the contrast enhanced by the function f_{50} in (6.1).

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 6.11c.

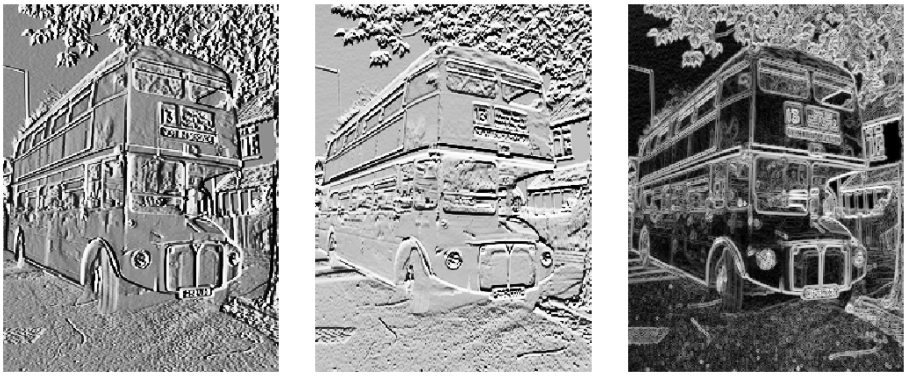
The image of the gradient looks quite different from the images of the two partial derivatives. The reason is that the numbers that represent the length of the gradient are (square roots of) sums of squares of numbers. This means that the parts of the image that have virtually constant intensity (partial derivatives close to 0) are coloured black. In the images of the partial derivatives these values ended up in the middle of the range of intensity values, with a final colour of grey, since there were both positive and negative values.

Figure 6.11a shows the computed values of the gradient. Although it is possible that the length of the gradient could become larger than 1, the maximum value in this case is about 0.876. By normalising the intensities we therefore increase the contrast slightly and obtain the image in figure 6.11b.

To enhance the contrast further we have to do something different from what was done in the other images since we now have a large number of intensities near 0. The solution is to apply a function like the ones shown in figure 6.8b to the intensities. If we use the function $g_{0.01}$ defined in equation(6.2) we obtain the image in figure 6.11c.

(a) (b) (c)

Figure 6.11: Computing the gradient. The image obtained from the computed gradient is shown in (a) and in (b) the numbers have been normalised. In (c) the contrast has been enhanced with a logarithmic function.



(a) (b) (c)

Figure 6.12: The first-order partial derivatives in the x-direction (a) and y-direction (b), and the length of the gradient (c). In all images, the computed numbers have been normalised and the contrast enhanced.

6.2.2 Comparing the first derivatives

Figure 6.12 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.

6.2.3 Second-order derivatives

To compute the three second order derivatives we apply the corresponding computational molecules which we already have described.

Observation 6.21 (Second order derivatives of an image)**.** The second order derivatives of an image P can be computed by applying the computational molecules

$$\frac{\partial^2 P}{\partial x^2}: \quad \begin{pmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \tag{6.8}$$

$$\frac{\partial^2 P}{\partial y \partial x}: \quad \frac{1}{4}\begin{pmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{pmatrix}, \tag{6.9}$$

$$\frac{\partial^2 P}{\partial y^2}: \quad \begin{pmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{pmatrix}. \tag{6.10}$$

With the information in observation 6.21 it is quite easy to compute the second-order derivatives, and the results are shown in figure 6.13. The computed derivatives were first normalised and then the contrast enhanced with the function f_{100} in each image, see equation 6.1.

As for the first derivatives, the xx-derivative seems to emphasise vertical edges and the yy-derivative horizontal edges. However, we also see that the second derivatives are more sensitive to noise in the image (the areas of grey are less uniform). The mixed derivative behaves a bit differently from the other two, and not surprisingly it seems to pick up both horizontal and vertical edges.

(a) (b) (c)

Figure 6.13: The second-order partial derivatives in the x-direction (a) and xy-direction (b), and the y-direction (c). In all images, the computed numbers have been normalised and the contrast enhanced.

Exercises for section 6.2

Ex. 1 — Black and white images can be generated from greyscale images (with values between 0 and 255) by replacing each pixel value with the one of 0 and 255 which is closest. Use this strategy to generate the black and white image shown in Figure 6.1(a).

Ex. 2 — Generate the image in Figure 6.8(d) on your own by writing code which uses the function `contrastadjust`.

Ex. 3 — Let us also consider the second way we mentioned for increasing the contrast.

a. Write a function `contrastadjust2` which instead uses the function 6.1 to increase the contrast.

b. Generate the image in Figure 6.8(c) on your own by using your code from Exercise 2, and instead calling the function `contrastadjust2`.

Ex. 4 — In this exercise we will look at another function for increasing the contrast of a picture.

a. Show that the function $f : \mathbb{R} \to \mathbb{R}$ given by

$$f_n(x) = x^n,$$

for all n maps the interval $[0, 1] \to [0, 1]$, and that $f'(1) \to \infty$ as $n \to \infty$.

Figure 6.14: Secret message

b. The color image `secret.jpg`,shown in Figure 6.14, contains some information that is nearly invisible to the naked eye on most computer monitors. Use the function $f(x)$, to reveal the secret message.
Hint: You will first need to convert the image to a greyscale image. You can then use the function `contrastadjust` as a starting point for your own program.

Ex. 5 — Generate the image in Figure 6.9(b) and -(c) by writing code which calls the function `smooth` with the appropriate computational molecules.

Ex. 6 — Generate the image in Figure 6.10 by writing code in the same way. Also generate the images in figures 6.11, 6.12, and 6.13.

6.3 Adaptations to image processing

There are two particular image standards we will consider in these notes. The first is teh JPEG standard. JPEG is short for *Joint Photographic Experts Group*, and is an image format that was approved as an international standard in 1994. JPEG is usually lossy, but may also be lossless and has become a popular format for image representation on the Internet. The standard defines both the algorithms for encoding and decoding and the storage format. JPEG performs a DCT on the image, and neglects DCT-coefficients which are below a given threshold. We will describe this in the next chapter. JPEG codes the remaining DCT-coefficients by a variation of Huffman coding, but it may also use arithmetic coding. The compression level in JPEG images is selected by the user and may result in conspicuous artefacts if set too high. JPEG is especially prone to artefacts in areas where the intensity changes quickly from pixel to pixel. The extension of a JPEG-file is `.jpg` or `.jpeg`.

The second standard we will consider is JPEG2000. It was developed to address some of the shortcomings of JPEG, and is based on wavelets.

There are several extensions, additions and modifications to the theory we will present in the later chapters, which are needed in order for us to have a full image compression system: the wavelet transform, the DCT, and the DFT were addressed because these are linked to relevant mathematics, and because they are important ingredients in many standards for sound and images. In this section we will mention many other extensions which also are important.

6.3.1 Lossless coding

Somewhere in the image processing or sound processing pipeline we need a step which actually achieves compression of the data. We have not mentioned anything about this step, since the output from the wavelet transform or the DCT is simply another set of data of the same size, called the *transformed data*. What we need to do is to apply a coding algorithm to this data to achieve compression. This may be Huffman coding, arithmetic coding, or any other algorithm. These methods are applied to the transformed data, since the effect of the wavelet transform is that it exploits the data so that it can be represented with data with lower entropy, so that it can be compressed more efficiently with these techniques.

6.3.2 Quantization

Coding as we have learnt previously is a lossless operation. As we saw, for certain wavelets the transform can also be performed in a lossless manner. These wavelets are, however, quite restrictive, which is why there is some loss involved with most wavelets used in practical applications. When there is some loss inherent in the transform, a quantization of the transformed data is also performed before the coding takes place. This quantization is typically done with a fixed number of bits, but may also be more advanced.

6.3.3 Preprocessing

Image compression as performed for certain image standards also often preprocess the pixel values before any transform is applied. The preprocessing may be centering the pixel values around a certain value, or extracting the different image components before they are processed separately.

6.3.4 Tiles, blocks, and error resilience

We have presented the wavelet transform as something which transforms the entire image. In practice this is not the case. The image is very often split into smaller parts, often called tiles. The tiles in an image are processed independently, so that errors whih occur within one tile do not affect the appearance of

parts in the image which correspond to other tiles. This makes the image compression what we call *error-resilient*, to errors such as transmission errors. The second reason for splitting into tiles has to do with that it may be more efficient to perform many transforms on smaller parts, rather than one big transform for the entire image. Performing one big transform would force us to have a big part of the image in memory during each computation, as well as remove possibilities for parallel computing. Often the algorithm itself also requires more computations when the size is bigger. For some standards, tiles are split into even smaller parts, called blocks, where parts of the processing within each block also is performed independently. This makes the possibilities for parallel computing even bigger. As an example, we mentioned that the JPEG standard performs the two-dimensional DCT on blocks as small as size 8×8.

6.3.5 Metadata

An image standard also defines how to store metadata about an image, and what metadata is accepted, like resolution, time when the image was taken, or where the image was taken (GPS coordinates) and similar information. Metadata can also tell us how the colour in the image are represented. As we have already seen, in most colour images the colour of a pixel is represented in terms of the amount of red, green and blue or (r, g, b). But there are other possibilities as well: Instead of storing all 24 bits of colour information in cases where each of the three colour components needs 8 bits, it is common to create a table of 256 colours with which a given image could be represented quite well. Instead of storing the 24 bits, one then just stores a colour table in the metadata, and at each pixel, the eight bits corresponding to the correct entry in the table. This is usually referred to as eight-bit colour and the table is called a *look-up* table or *palette*. For large photographs, however, 256 colours is far from sufficient to obtain reasonable colour reproduction.

Metadata is usually stored in the beginning of the file, formatted in a very specific way.

6.4 Summary

We first discussed the basic question what an image is, and took a closer look at digital images. We went through several operations which give meaning for digital images, and showed how to implement these.

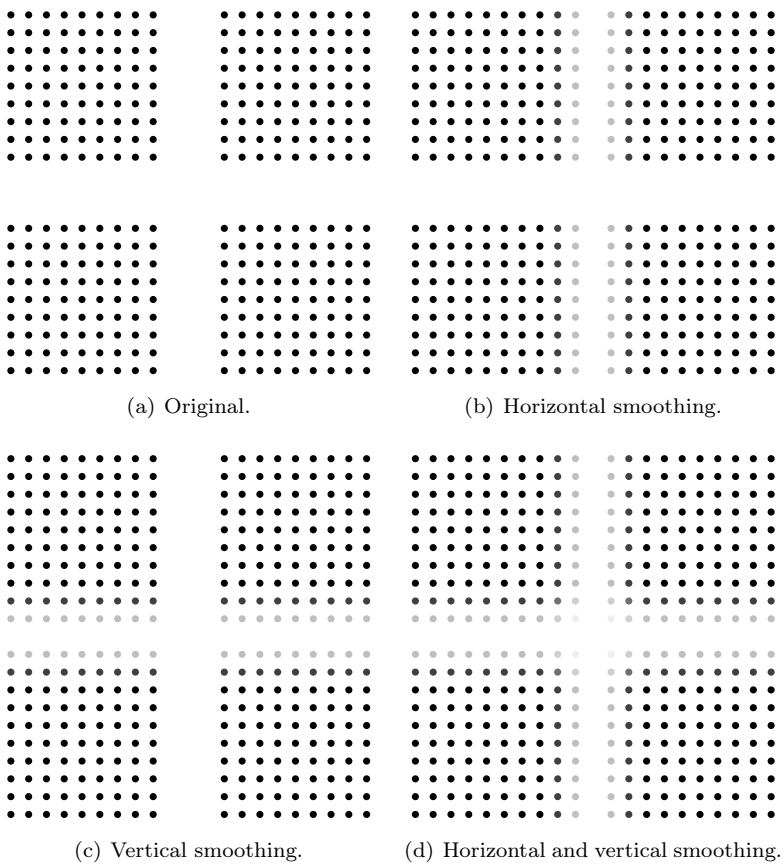
Chapter 7

Definition and properties of tensor products

The DFT, the DCT, and the wavelet transform were all defined as changes of basis for vectors or functions of one variable and therefore cannot be directly applied to higher dimensional data like images. In this chapter we will introduce a simple recipe for extending such one-dimensional schemes to two (and higher) dimensions. The basic ingredient is the *tensor product* construction. This is a general tool for constructing two-dimensional functions and filters from one-dimensional counterparts. This will allow us to generalise the filtering and compression techniques for audio to images, and we will also recognise some of the basic operations for images introduced in Chapter 6 as tensor product constructions.

A two-dimensional discrete function on a rectangular domain, like for example an image, is conveniently represented in terms of a matrix X with elements $X_{i,j}$, and with indices in the ranges $0 \leq i \leq M-1$ and $0 \leq j \leq N-1$. One way to apply filters to X would be to rearrange the matrix into a long vector, column by column. We could then apply a one-dimensional filter to this vector and then split the resulting vector into columns that can be reassembled back into a matrix again. This approach may have some undesirable effects near the boundaries between columns. In addition, the resulting computations may be rather ineffective. Consider for example the case where X is an $N \times N$ matrix so that the long vector has length N^2. Then a linear transformation applied to X involves multiplication with an $N^2 \times N^2$-matrix. Each such matrix multiplication may require as many as N^4 multiplications which is substantial when N is large.

The concept of tensor products can be used to address these problems. Using tensor products, one can construct operations on two-dimensional functions which inherit properties of one-dimensional operations. Tensor products also turn out to be computationally efficient.



(a) Original. (b) Horizontal smoothing.

(c) Vertical smoothing. (d) Horizontal and vertical smoothing.

Figure 7.1: The results of smoothing an image with the filter $\{1/4, \underline{1/2}, 1/4\}$ horizontally, vertically, and both. The pixels are shown as disks with intensity corresponding to the pixel values.

7.1 The tensor product of vectors

In Chapter 6 we saw examples of several filters applied to images. The filters of special interest to us now are those that determined a new image by combining neighbouring pixels, like the smoothing filter in Example 6.16 and the edge detection filter in Example 6.18. Our aim now is to try and apply filters like these as a repeated application of one-dimensional filters rather than using a computational molecule like in Chapter 6. It will be instructive to do this with an example before we describe the general construction, so let us revisit Example 6.16.

Figure 7.1 (a) shows an example of a simple image. We want to smooth this image X with the one-dimensional filter T given by $y_n = (T(\boldsymbol{x}))_n = (x_{n-1} + 2x_n + x_{n+1})/4$, or equivalently $T = \{1/4, \underline{1/2}, 1/4\}$. There are two obvious

one-dimensional operations we can do:

1. Apply the filter to each row in the image.
2. Apply the filter to each column in the image.

The problem is of course that these two operations will only smooth the image either horizontally or vertically as can be seen in the image in (b) and (c) of Figure 7.1.

So what else can we do? We can of course first smooth all the rows of the image and then smooth the columns of the resulting image. The result of this is shown in Figure 7.1 (d). Note that we could have performed the operations in the opposite order: first vertical smoothing and then horizontal smoothing, and currently we do not know if this is the same. We will show that these things actually are the same, and that computational molecules, as we saw in Chapter 6, naturally describe operations which are performed both vertically and horizontally. The main ingredient in this will be the tensor product construction. We start by defining the tensor product of two vectors.

Definition 7.1 (Tensor product of vectors)**.** If $\boldsymbol{x}, \boldsymbol{y}$ are vectors of length M and N, respectively, their tensor product $\boldsymbol{x} \otimes \boldsymbol{y}$ is defined as the $M \times N$-matrix defined by $(\boldsymbol{x} \otimes \boldsymbol{y})_{ij} = x_i y_j$. In other words, $\boldsymbol{x} \otimes \boldsymbol{y} = \boldsymbol{x}\boldsymbol{y}^T$.

In particular $\boldsymbol{x} \otimes \boldsymbol{y}$ is a matrix of rank 1, which means that most matrices cannot be written as tensor products. The special case $\boldsymbol{e}_i \otimes \boldsymbol{e}_j$ is the matrix which is 1 at (i, j) and 0 elsewhere, and the set of all such matrices forms a basis for the set of $M \times N$-matrices.

Observation 7.2 (Interpretation of tensor products for vectors)**.** Let

$$\mathcal{E}_M = \{\boldsymbol{e}_i\}_{i=0}^{M-1} \quad \text{and} \quad \mathcal{E}_N = \{\boldsymbol{e}_i\}_{i=0}^{N-1}$$

be the standard bases for $\mathbb{R}^M$ and $\mathbb{R}^N$. Then

$$\mathcal{E}_{M,N} = \{\boldsymbol{e}_i \otimes \boldsymbol{e}_j\}_{(i,j)=(0,0)}^{(M-1,N-1)}$$

is a basis for $L_{M,N}(\mathbb{R})$, the set of $M \times N$-matrices. This basis is often referred to as the standard basis for $L_{M,N}(\mathbb{R})$.

An image can simply be thought of as a matrix in $L_{M,N}(\mathbb{R})$. With this definition of tensor products, we can define operations on images by extending the one-dimensional filtering operations defined earlier.

Definition 7.3 (Tensor product of matrices)**.** If $S : \mathbb{R}^M \to \mathbb{R}^M$ and $T : \mathbb{R}^N \to \mathbb{R}^N$ are matrices, we define the linear mapping $S \otimes T : L_{M,N}(\mathbb{R}) \to L_{M,N}(\mathbb{R})$ by linear extension of $(S \otimes T)(\boldsymbol{e}_i \otimes \boldsymbol{e}_j) = (S\boldsymbol{e}_i) \otimes (T\boldsymbol{e}_j)$. The linear mapping $S \otimes T$ is called the tensor product of the matrices S and T.

A couple of remarks are in order. First, from linear algebra we know that, when T is linear mapping from V and $T(\boldsymbol{v}_i)$ is known for a basis $\{\boldsymbol{v}_i\}_i$ of V, T is uniquely determined. In particular, since the $\{\boldsymbol{e}_i \otimes \boldsymbol{e}_j\}_{i,j}$ form a basis, there exists a unique linear transformation $S \otimes T$ so that $(S \otimes T)(\boldsymbol{e}_i \otimes \boldsymbol{e}_j) = (S\boldsymbol{e}_i) \otimes (T\boldsymbol{e}_j)$. This unique linear transformation is what we call the linear extension from the values in the given basis.

Secondly $S \otimes T$ also satisfies $(S \otimes T)(\boldsymbol{x} \otimes \boldsymbol{y}) = (S\boldsymbol{x}) \otimes (T\boldsymbol{y})$. This follows from

$$\begin{aligned}(S \otimes T)(\boldsymbol{x} \otimes \boldsymbol{y}) &= (S \otimes T)((\sum_i x_i \boldsymbol{e}_i) \otimes (\sum_j y_j \boldsymbol{e}_j)) = (S \otimes T)(\sum_{i,j} x_i y_j (\boldsymbol{e}_i \otimes \boldsymbol{e}_j)) \\ &= \sum_{i,j} x_i y_j (S \otimes T)(\boldsymbol{e}_i \otimes \boldsymbol{e}_j) = \sum_{i,j} x_i y_j (S\boldsymbol{e}_i) \otimes (T\boldsymbol{e}_j) \\ &= \sum_{i,j} x_i y_j S\boldsymbol{e}_i ((T\boldsymbol{e}_j))^T = S(\sum_i x_i \boldsymbol{e}_i)(T(\sum_j y_j \boldsymbol{e}_j))^T \\ &= S\boldsymbol{x}(T\boldsymbol{y})^T = (S\boldsymbol{x}) \otimes (T\boldsymbol{y}).\end{aligned}$$

Here we used the result from Exercise 5. Linear extension is necessary anyway, since only rank 1 matrices have the form $\boldsymbol{x} \otimes \boldsymbol{y}$.

Example 7.4 (Smoothing an image)**.** Assume that S and T are both filters, and that $S = T = \{1/4, 1/2, 1/4\}$. Let us set $M = 5$ and $N = 7$, and let us compute $(S \otimes T)(\boldsymbol{e}_2 \otimes \boldsymbol{e}_3)$. We have that

$$(S \otimes T)(\boldsymbol{e}_2 \otimes \boldsymbol{e}_3) = (S\boldsymbol{e}_2)(T\boldsymbol{e}_3)^T = (\mathrm{col}_2 S)(\mathrm{col}_3 T)^T.$$

Since

$$S = \frac{1}{4}\begin{pmatrix} 2 & 1 & 0 & 0 & 1 \\ 1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 2 & 1 \\ 1 & 0 & 0 & 1 & 2 \end{pmatrix} \qquad T = \frac{1}{4}\begin{pmatrix} 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 2 \end{pmatrix},$$

we find that

$$\frac{1}{4}\begin{pmatrix} 0 \\ 1 \\ 2 \\ 1 \\ 0 \end{pmatrix} \frac{1}{4}\begin{pmatrix} 0 & 0 & 1 & 2 & 1 & 0 & 0 \end{pmatrix} = \frac{1}{16}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 2 & 4 & 2 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

We recognize here the computational molecule from Example 6.16 for smoothing an image. More generally it is not hard to see that $(S \otimes T)(\boldsymbol{e}_i \otimes \boldsymbol{e}_j)$ is the matrix where the same computational molecule is placed with its centre at (i, j).

Clearly then, the linear extension $S \otimes T$ is obtained by placing the computational molecule over all indices, multiplying by the value at that index, and summing everything together. This is equivalent to the procedure for smoothing we learnt in Example 6.16. One can also write down component formulas for this as well. To achieve this, one starts with writing out the operation for tensor products of vectors:

$$\begin{aligned}
&((T \otimes T)(\boldsymbol{x} \otimes \boldsymbol{y}))_{i,j} \\
&= ((T\boldsymbol{x}) \otimes (T\boldsymbol{y}))_{i,j} = (T\boldsymbol{x})(T\boldsymbol{y})^T)_{i,j} = (T\boldsymbol{x})_i (T\boldsymbol{y})_j \\
&= \frac{1}{4}(x_{i-1} + 2x_i + x_{i+1})\frac{1}{4}(y_{j-1} + 2y_j + y_{j+1}) \\
&= \frac{1}{16}(x_{i-1}y_{j-1} + 2x_{i-1}y_j + x_{i-1}y_{j+1} \\
&\quad + 2x_i y_{j-1} + 4x_i y_j + 2x_i y_{j+1} + x_{i+1}y_{j-1} + 2x_{i+1}y_j + x_{i+1}y_{j+1}) \\
&= \frac{1}{16}((\boldsymbol{x} \otimes \boldsymbol{y})_{i-1,j-1} + 2(\boldsymbol{x} \otimes \boldsymbol{y})_{i-1,j} + (\boldsymbol{x} \otimes \boldsymbol{y})_{i-1,j+1} \\
&\quad 2(\boldsymbol{x} \otimes \boldsymbol{y})_{i,j-1} + 4(\boldsymbol{x} \otimes \boldsymbol{y})_{i,j} + 2(\boldsymbol{x} \otimes \boldsymbol{y})_{i,j+1} \\
&\quad (\boldsymbol{x} \otimes \boldsymbol{y})_{i+1,j-1} + 2(\boldsymbol{x} \otimes \boldsymbol{y})_{i+1,j} + (\boldsymbol{x} \otimes \boldsymbol{y})_{i+1,j+1}).
\end{aligned}$$

Since this formula is valid when applied to any tensor product of two vectors, it is also valid when applied to any matrix:

$$\begin{aligned}
&((T \otimes T)X)_{i,j} \\
&= \frac{1}{16}(X_{i-1,j-1} + 2X_{i,j-1} + 2X_{i-1,j} + 4X_{i,j} + 2X_{i,j+1} \\
&\quad + 2X_{i+1,j} + X_{i+1,j+1})
\end{aligned}$$

This again confirms that the computational molecule given by Equation 6.3 in Example 6.16 is the tensor product of the filter $\{1/4, \underline{1/2}, 1/4\}$ with itself.

While we have seen that the computational molecules from Chapter 1 can be written as tensor products, not all computational molecules can be written as tensor products: we need of course that the molecule is a rank 1 matrix, since matrices which can be written as a tensor product always have rank 1.

The tensor product can be expressed explicitly in terms of matrix products.

Theorem 7.5. If $S : \mathbb{R}^M \to \mathbb{R}^M$ and $T : \mathbb{R}^N \to \mathbb{R}^N$ are matrices, the action of their tensor product on a matrix X is given by $(S \otimes T)X = SXT^T$ for any $X \in L_{M,N}(\mathbb{R})$.

Proof. We have that

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

This means that $(S \otimes T)X = SXT^T$ for any $X \in L_{M,N}(\mathbb{R})$, since equality holds on the basis vectors $\boldsymbol{e}_i \otimes \boldsymbol{e}_j$. □

This leads to the following implementation for the tensor product of matrices:

Theorem 7.6 (Implementation of a tensor product of matrices)**.** If $S : \mathbb{R}^M \to \mathbb{R}^M$, $T : \mathbb{R}^N \to \mathbb{R}^N$ are matrices, and $X \in L_{M,N}(\mathbb{R})$, we have that $(S \otimes T)X$ can be computed as follows:

1. Apply S to every column of X.
2. Transpose the resulting matrix.
3. Apply T to every column in the resulting matrix.
4. Transpose the resulting matrix.

This recipe for computing $(S \otimes T)X$ is perhaps best seen if we write

$$(S \otimes T)X = SXT^T = (T(SX)^T)^T. \tag{7.1}$$

In the first step above we compute SX, in the second step $(SX)^T$, in the third step $T(SX)^T$, in the fourth step $(T(SX)^T)^T$. The reason for writing the tensor product this way, as an operation column by column, has to do with with that S and T are mostly filters for our purposes, and that we want to reuse efficient implementations instead of performing full matrix multiplications, just as we decided to express a wavelet transformation in terms of filters. The reason for using columns instead of rows has to do with that we have expressed filtering as a matrix by column multiplication. Note that this choice of using columns instead of rows should be influenced by how the computer actually stores values in a matrix. If these values are stored column by column, performing operations columnwise may be a good idea, since then the values from the matrix are read in the same order as they are stored. If matrix values are stored row by row, it may be a good idea to rewrite the procedure above so that operations are performed row by row also (see Exercise 7).

Theorem 7.6 leads to the following algorithm for computing the tensor product of matrices:

```
[M,N]=size(X);
for col=1:N
  X(:,col)=S*X(:,col);
end
X=X'
for col=1:M
  X(:,col)=T*X(:,col);
end
X=X';
```

This algorithm replaces the rows and columns in X at each step. In the following, $S = T$ in most cases. In this case we can replace with the following algorithm, which is even simpler:

```
for k=1:2
  for col=1:size(X,2)
    X(:,col)=S*X(:,col);
  end
  X=X';
end
```

In an efficient algorithm, we would of course replace the matrix multiplications with S and T with efficient implementations.

If we want to apply a sequence of tensor products of filters to a matrix, the order of the operations does not matter. This will follow from the next result:

Corollary 7.7. If $S_1 \otimes T_1$ and $S_2 \otimes T_2$ are two tensor products of one dimensional filters, then $(S_1 \otimes T_1)(S_2 \otimes T_2) = (S_1 S_2) \otimes (T_1 T_2)$.

Proof. By Theorem 7.5 we have that

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

for any $X \in L_{M,N}(\mathbb{R})$. This proves the result. □

Suppose that we want to apply the operation $S \otimes T$ to an image. We can write

$$S \otimes T = (S \otimes I)(I \otimes T) = (I \otimes T)(S \otimes I). \tag{7.2}$$

Moreover, from Theorem 7.5 it follows that

$$\begin{aligned}(S \otimes I)X &= SX \\ (I \otimes T)X &= XT^T = (TX^T)^T.\end{aligned}$$

This means that $S \otimes I$ corresponds to applying S to each column in X, and $I \otimes T$ corresponds to applying T to each row in X. When S and T are smoothing filters, this is what we refered to as vertical smoothing and horizontal smoothing, respectively. The relations in Equation (7.2) thus have the following interpretation (alternatively note that the order of left or right multiplication does not matter).

Observation 7.8. The order of vertical and horizontal smoothing does not matter, and any tensor product of filters $S \otimes T$ can be written as a horizontal filtering operation $I \otimes T$ followed by a vertical filtering operation $S \otimes I$.

In fact, the order of any vertical operation $S \otimes I$ and horizontal operation $I \otimes T$ does not matter: it is not required that the operations are filters. For filters we have a stronger result: If S_1, T_1, S_2, T_2 all are filters, we have from Corollary 7.7 that $(S_1 \otimes T_1)(S_2 \otimes T_2) = (S_2 \otimes T_2)(S_1 \otimes T_1)$, since all filters commute. This does not hold in general since general matrices do not commute.

Example 7.9 (Detecting edges)**.** Consider the bass reducing filter $T = \{1/2, \underline{0}, -1/2\}$, i.e. $(T(\boldsymbol{x}))_n = \frac{1}{2}(x_{n+1} - x_{n-1})$. We compute the vertical filtering operation $T \otimes I$ as

$$
\begin{aligned}
&((T \otimes I)(\boldsymbol{x} \otimes \boldsymbol{y}))_{i,j}] = (T\boldsymbol{x})_i y_j \\
&= \frac{1}{2}(x_{i+1} - x_{i-1}) y_j = \frac{1}{2} x_{i+1} y_j - \frac{1}{2} x_{i-1} y_j = \frac{1}{2}(\boldsymbol{x} \otimes \boldsymbol{y})_{i+1,j} - \frac{1}{2}(\boldsymbol{x} \otimes \boldsymbol{y})_{i-1,j}.
\end{aligned}
$$

This shows as above that $T \otimes I$ is the transformation where the computational molecule given by Equation 6.7 in Example 6.18 is placed over the image samples. This tensor product can thus be used for detecting vertical edges in images.

Exercises for Section 7.1

Ex. 1 — With $T = \{1/2, \underline{0}, -1/2\}$, show that $I \otimes T$ is the transformation where the computational molecule is given by Equation 6.6 in Example 6.18. This tensor product can thus be used for detecting horizontal edges in images.

Ex. 2 — With $T = \{1/2, \underline{0}, -1/2\}$, show that $T \otimes T$ corresponds to the computational molecule given by Equation 6.9 in Example 6.18.

Ex. 3 — Let T be the moving average filter of length $2L + 1$, i.e. $T = \frac{1}{L}\{\underbrace{1, \cdots, 1, \underline{1}, 1, \cdots, 1}_{2L+1 \text{ times}}\}$. As in Example 7.4, find the computational molecule of $T \otimes T$.

Ex. 4 — Verify that the computational molecule given by Equation 6.4 in Example 6.18 is the same as that of $T \otimes T$, where $T = \{\frac{1}{64}, \frac{6}{64}, \frac{15}{64}, \underline{\frac{20}{64}}, \frac{15}{64}, \frac{6}{64}, \frac{1}{64}\}$ (the coefficients come from row 6 of Pascals triangle).

Ex. 5 — 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 $F(\boldsymbol{x}, \alpha \boldsymbol{y}_1 + \beta \boldsymbol{y}_2) = \alpha F(\boldsymbol{x}, \boldsymbol{y}_1) + \beta F(\boldsymbol{x}, \boldsymbol{y}_2)$.

Ex. 6 — Attempt to find matrices $S : \mathbb{R}^M \to \mathbb{R}^M$ and $T : \mathbb{R}^N \to \mathbb{R}^N$ so that the following mappings from $L_{M,N}(\mathbb{R})$ to $L_{M,N}(\mathbb{R})$ can be written on the form

$X \to SXT^T = (S \otimes T)X$. In all the cases, it may be that no such S, T can be found. If this is the case, prove it.

a. The mapping which reverses the order of the rows in a matrix.

b. The mapping which reverses the order of the columns in a matrix.

c. The mapping which transposes a matrix.

Ex. 7 — Find an alternative form for Equation (7.1) and an accompanying reimplementation of Theorem 7.6 which is adapted to the case when we want all operations to be performed row by row, instead of column by column.

7.2 Change of bases in tensor products

In this section we will prove a specialization of our previous result to the case where S and T are change of coordinate matrices. We start by proving the following:

Theorem 7.10. 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 7.1.5). 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 7.11 (Coordinate matrix)**.** Let $\{\boldsymbol{v}_i\}_{i=0}^{M-1}, \{\boldsymbol{w}_j\}_{j=0}^{N-1}$ be bases for $\mathbb{R}^M$ and $\mathbb{R}^N$. By the coordinate matrix of $\sum_{k,l} \alpha_{k,l}(\boldsymbol{v}_k \otimes \boldsymbol{w}_l)$ we will mean the $M \times N$-matrix X with entries $X_{kl} = \alpha_{k,l}$.

We will have use for the following theorem, which shows how change of coordinates in $\mathbb{R}^M$ and $\mathbb{R}^N$ translate to a change of coordinates in the tensor product:

Theorem 7.12 (Change of coordinates in tensor products)**.** Assume that $\mathcal{B}_1, \mathcal{C}_1$ are bases for $\mathbb{R}^M$, and $\mathcal{B}_2, \mathcal{C}_2$ are bases for $\mathbb{R}^N$, and that S is the change of coordinates matrix from $\mathcal{C}_1$ to $\mathcal{B}_1$, and that T is the change of coordinates matrix from $\mathcal{C}_2$ to $\mathcal{B}_2$. Both $\mathcal{B}_1 \otimes \mathcal{B}_2$ and $\mathcal{C}_1 \otimes \mathcal{C}_2$ are bases for $L_{M,N}(\mathbb{R})$, and if X is the coordinate matrix in $\mathcal{C}_1 \otimes \mathcal{C}_2$, and Y the coordinate matrix in $\mathcal{B}_1 \otimes \mathcal{B}_2$, then

$$Y = SXT^T. \tag{7.3}$$

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

$$SXT^T = S(\boldsymbol{e}_i \otimes \boldsymbol{e}_j)T^T = \mathrm{col}_i(S)\mathrm{row}_j(T^T).$$

On the other hand, since column i in S is the coordinates of $\boldsymbol{c}_{1i}$ in the basis $\mathcal{B}_1$, and column j in T is the coordinates of $\boldsymbol{c}_{2j}$ in the basis $\mathcal{B}_2$, we can write

$$\begin{aligned}\boldsymbol{c}_{1i} \otimes \boldsymbol{c}_{2j} &= \left(\sum_k S_{k,i}\boldsymbol{b}_{1k}\right) \otimes \left(\sum_l T_{l,j}\boldsymbol{b}_{2l}\right) = \sum_{k,l} S_{k,i}T_{l,j}(\boldsymbol{b}_{1k} \otimes \boldsymbol{b}_{2l}) \\ &= \sum_{k,l} S_{k,i}(T^T)_{j,l}(\boldsymbol{b}_{1k} \otimes \boldsymbol{b}_{2l}) = \sum_{k,l} (\mathrm{col}_i(S)\mathrm{row}_j(T^T))_{k,l}(\boldsymbol{b}_{1k} \otimes \boldsymbol{b}_{2l})\end{aligned}$$

we see that the coordinate vector of $\boldsymbol{c}_{1i} \otimes \boldsymbol{c}_{2j}$ in the basis $\mathcal{B}_1 \otimes \mathcal{B}_2$ is $\mathrm{col}_i(S)\mathrm{row}_j(T^T)$. In other words, change of coordinates coincides with $X \to SXT^T$, and the proof is done. □

In both cases of tensor products of matrices and change of coordinates in tensor products, we see that we need to compute the mapping $X \to SXT^T$. This means that we can restate Theorem 7.6 for change of coordinates as follows:

Theorem 7.13 (Implementation of change of coordinates in tensor products)**.** The change of coordinates from $\mathcal{C}_1 \otimes \mathcal{C}_2$ to $\mathcal{B}_1 \otimes \mathcal{B}_2$ can be implemented as follows:

1. For every column in the coordinate matrix in $\mathcal{C}_1 \otimes \mathcal{C}_2$, perform a change of coordinates from $\mathcal{C}_1$ to $\mathcal{B}_1$.
2. Transpose the resulting matrix.
3. For every column in the resulting matrix, perform a change of coordinates from $\mathcal{C}_2$ to $\mathcal{B}_2$.
4. Transpose the resulting matrix.

We can reuse the algorithm from the previous section to implement this. In the following operations on images, we will visualize the pixel values in an image as coordinates in the standard basis, and perform a change of coordinates.

Example 7.14 (Change of coordinates with the DFT)**.** The DFT is one particular change of coordinates which we have considered. The DFT was the change of coordinates from the standard basis to the Fourier basis. The corresponding change of coordinates in a tensor product is obtained by substituting with the DFT as the function implementing change of coordinates in Theorem 7.13. The change of coordinates in the opposite direction is obtained by using the IDFT instead of the DFT.

Modern image standards do typically not apply a change of coordinates to the entire image. Rather one splits the image into smaller squares of appropriate size, called blocks, and perform change of coordinates independently for each block. With the JPEG standard, the blocks are always 8×8. It is of course not a coincidence that a power of 2 is chosen here, since the DFT takes a simplified form in case of powers of 2.

The DFT values express frequency components. The same applies for the two-dimensional DFT and thus for images, but frequencies are now represented in two different directions. The thing which actually provides compression in many image standards is that frequency components which are small are set to 0. This corresponds to neglecting frequencies in the image which have small contributions. This type of lossy compression has little effect on the human perception of the image, if we use a suitable neglection threshold.

In Figure 7.3 we have applied the two-dimensional DFT to our test image. We have then neglected DFT coefficients which are below certain thresholds, and transformed the samples back to reconstruct the image. When increasing the threshold, the image becomes more and more unclear, but the image is quite clear in the first case, where as much as more than 90% of the samples have been neglected. The blocking effect at the block boundaries is clearly visible.

Example 7.15 (Change of coordinates with the DCT)**.** Similarly to the DFT, the DCT was the change of coordinates from the standard basis to what we

(a) Threshold 30. 91.3% of the DFT-values were neglected

(b) Threshold 50. 94.9% of the DFT-values were neglected

(c) Threshold 100. 97.4% of the DFT-values were neglected

Figure 7.2: The effect on an image when it is transformed with the DFT, and the DFT-coefficients below a certain threshold were neglected.

called the DCT basis. The DCT is used more than the DFT in image processing. Change of coordinates in tensor products between the standard basis and the DCT basis is obtained by substituting with the DCT and the IDCT in Theorem 7.13. The JPEG standard actually applies a two-dimensional DCT to the blocks of size 8×8, it does not apply the two-dimensional DFT.

If we follow the same strategy for the DCT as for the DFT, so that we neglect DCT-coefficients which are below a given threshold, we get the images shown in Figure 7.3. We see similar effects as with the DFT, but it seems that the latter images are a bit clearer, verifying that the DCT is a better choice than the DFT. It is also interesting to compare with what happens when we drop splitting the image into blocks. Of course, when we neglect many of the DCT-coefficients, we should see some artifacts, but there is no reason to believe that these should be at the old block boundaries. The new artifacts can be seen in Figure 7.4, where the same thresholds as before have been used. Clearly, the new artifacts take a completely different shape.

In the exercises you will be asked to implement functions which generate the images shown in these examples.

Exercises for Section 7.2

Ex. 1 — Implement functions

```
function newx=FFT2Impl(x)
function x=IFF2Impl(newx)
function newx=DCT2Impl(x)
function x=IDCT2Impl(newx)
```

(a) Threshold 30. 93.9% of the DCT-values were neglected

(b) Threshold 50. 96.0% of the DCT-values were neglected

(c) Threshold 100. 97.6% of the DCT-values were neglected

Figure 7.3: The effect on an image when it is transformed with the DCT, and the DCT-coefficients below a certain threshold were neglected.

(a) Threshold 30. 93.7% of the DCT-values were neglected

(b) Threshold 50. 96.7% of the DCT-values were neglected

(c) Threshold 100. 98.5% of the DCT-values were neglected

Figure 7.4: The effect on an image when it is transformed with the DCT, and the DCT-coefficients below a certain threshold were neglected. The image has not been split into blocks here.

which implement the two-dimensional DCT, FFT, and their inverses as described in this section. Base your code on the algorithm at the end of Section 7.1.

Ex. 2 — Implement functions

```
function samples=transform2jpeg(x)
function samples=transform2invjpeg(x)
```

which splits the image into blocks of size 8×8, and performs the DCT2/IDCT2 on each block. Finally run the code

```
function showDCThigher(threshold)
  img = double(imread('mm.gif','gif'));
  newimg=transform2jpeg(img);
  thresholdmatr=(abs(newimg)>=threshold);
  zeroedout=size(img,1)*size(img,2)-sum(sum(thresholdmatr));
  newimg=transform2invjpeg(newimg.*thresholdmatr);
  imageview(abs(newimg));
  fprintf('%i percent of samples zeroed out\n',...
      100*zeroedout/(size(img,1)*size(img,2)));
```

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

7.3 Summary

We defined the tensor product, and saw how this could be used to define operations on images in a similar way to how we defined operations on sound. It turned out that the tensor product construction could be used to construct some of the operations on images we looked at in the previous chapter, which now could be factorized into first filtering the columns in the image, and then filtering the rows in the image. We went through an algorithm for computing the tensor product, and established how we could perform change of coordinates in tensor products. This enables us to define two-dimensional extensions of the DCT and the DFT and their inverses, and we used these extensions to experiment on images.

Chapter 8

Tensor products in a wavelet setting

In Chapter 7 we defined tensor products in terms of vectors, and we saw that the tensor product of two vectors is in fact a matrix. The same construction can be applied to other vector spaces, in particular to vector spaces that are function spaces. As we will see, the tensor product of two univariate function spaces will be a space of functions in two variables. Recall that wavelets are defined in terms of function spaces, so this construction will allow us to define tensor products of wavelets. Through this we will be able to define wavelet transforms that can be applied to images.

Definition 8.1 (Tensor product of function spaces)**.** Let V and W be two vector spaces of functions defined on the intervals $[0, M)$ and $[0, N)$, respectively, and suppose that $f_1 \in V$ and $f_2 \in W$. The tensor product of f_1 and f_2, denoted $f_1 \otimes f_2$, denotes the function in two variables defined on $[0, M) \times [0, N)$ given by $f_1(t_1)f_2(t_2)$. The function $f_1 \otimes f_2$ is also referred to as the separable extension of f_1 and f_2 to two variables. The tensor product of the two spaces $V \otimes W$ denotes the set of all functions in two variables defined on $[0, M) \times [0, N)$ and on the form $f_1(t_1)f_2(t_2)$, where $f_1 \in V$ and $f_2 \in W$.

We will always assume that the spaces V and W consist of functions which are at least integrable. In this case $V \otimes W$ is also an inner product space, with the inner product given by a double integral,

$$\langle f, g \rangle = \int_0^N \int_0^M f(t_1, t_2) g(t_1, t_2) dt_1 dt_2. \tag{8.1}$$

In particular, this says that

$$\begin{aligned}\langle f_1 \otimes f_2, g_1 \otimes g_2 \rangle &= \int_0^N \int_0^M f_1(t_1) f_2(t_2) g_1(t_1) g_2(t_2) dt_1 dt_2 \\ &= \int_0^M f_1(t_1) g_1(t_1) dt_1 \int_0^N f_2(t_2) g_2(t_2) dt_2 = \langle f_1, g_1 \rangle \langle f_2, g_2 \rangle.\end{aligned} \tag{8.2}$$

This means that for tensor products, a double integral can be computed as the product of two one-dimensional integrals.

The tensor product space defined in Definition 8.1 is useful for approximation of functions of two variables if each of the two spaces of univariate functions have good approximation properties.

Idea 8.2. If the spaces V and W can be used to approximate functions in one variable, then $V \otimes W$ can be used to approximate functions in two variables.

We will not state this precisely, but just consider some important examples.

Example 8.3. Let $V = W$ be the space of all polynomials of finite degree. We know that V can be used for approximating many kinds of functions, such as continuous functions, for example by Taylor series. The tensor product $V \otimes V$ 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 8.4. Let $V = W = V_{N,T}$ 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 $V \otimes V$ now consists of all functions on the form $\sum_{k,l=0}^{N} \alpha_{k,l} e^{2\pi i k t_1/T} e^{2\pi i l t_2/T}$. One can show that this space has approximation properties similar to $V_{N,T}$. This is the basis for the theory of Fourier series in two variables.

In the following we think of $V \otimes W$ 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 7.10:

Theorem 8.5. If $\{f_i\}_{i=0}^{M-1}$ is a basis for V and $\{g_j\}_{j=0}^{N-1}$ is a basis for W, then $\{f_i \otimes g_j\}_{(i,j)=(0,0)}^{(M-1,N-1)}$ is a basis for $V \otimes W$. Moreover, if the bases for V and W are orthogonal/orthonormal, then the basis for $V \otimes W$ is orthogonal/orthonormal.

Proof. The proof is similar to that of Theorem 7.10: 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 8.2. □

We can now define the tensor product of two bases of functions as before: if $\mathcal{B} = \{f_i\}_{i=0}^{M-1}$ and $\mathcal{C} = \{g_j\}_{j=0}^{N-1}$, we set $\mathcal{B} \otimes \mathcal{C} = \{f_i \otimes g_j\}_{(i,j)=(0,0)}^{(M-1,N-1)}$. Coordinate matrices can also be defined as before: if $f(t_1, t_2) = \sum X_{i,j}(f_i \otimes g_j)(t_1, t_2)$, the coordinate matrix of f is the matrix X with elements $X_{i,j}$. Theorem 7.12 can also be proved in the same way in the context of function spaces. We state this as follows:

Theorem 8.6 (Change of coordinates in tensor products of function spaces)**.** Assume that $\mathcal{B}_1, \mathcal{C}_1$ are bases for V, and $\mathcal{B}_2, \mathcal{C}_2$ are bases for W, and that S is the change of coordinates matrix from $\mathcal{C}_1$ to $\mathcal{B}_1$, and that T is the change of coordinates matrix from $\mathcal{C}_2$ to $\mathcal{B}_2$. Both $\mathcal{B}_1 \otimes \mathcal{B}_2$ and $\mathcal{C}_1 \otimes \mathcal{C}_2$ are bases for $V \otimes W$, and if X is the coordinate matrix in $\mathcal{C}_1 \otimes \mathcal{C}_2$, and Y the coordinate matrix in $\mathcal{B}_1 \otimes \mathcal{B}_2$, then

$$Y = SXT^T. \tag{8.3}$$

8.1 Adopting the tensor product terminology to wavelets

In the remaining part of this chapter we will apply the tensor product construction to wavelets. In particular the spaces V, W from Definition 8.1 are defined from function spaces V_m, W_m, constructed from a given wavelet. We can in particular form the tensor products $\phi_{0,n_1} \otimes \boldsymbol{\phi}_{0,n_2}$. We will assume that

1. the first component ϕ_{0,n_1} has period M (so that $\{\phi_{0,n_1}\}_{n_1=0}^{M-1}$ is a basis for the first component space),

2. the second component ϕ_{0,n_2} has period N (so that $\{\phi_{0,n_2}\}_{n_2=0}^{N-1}$ is a basis for the second component space).

When we speak of $V_0 \otimes V_0$ we thus mean an MN-dimensional space with basis $\{\phi_{0,n_1} \otimes \boldsymbol{\phi}_{0,n_2}\}_{(n_1,n_2)=(0,0)}^{(M-1,N-1)}$, where the coordinate matrices are $M \times N$. This difference in the dimension of the two components is done to allow for images where the number of rows and columns may be different. In the following we

will implicitly assume that the component spaces have dimension M and N, to ease notation. If we use that $\boldsymbol{\phi}_{m-1} \oplus \boldsymbol{\psi}_{m-1}$ also is a basis for V_m, we get the following corollary to Theorem 8.5:

Corollary 8.7. Let ϕ, ψ be a scaling function and a mother wavelet. Then the two sets of tensor products given by

$$\boldsymbol{\phi}_m \otimes \boldsymbol{\phi}_m = \{\phi_{m,n_1} \otimes \phi_{m,n_2}\}_{n_1,n_2}$$

and

$$\begin{aligned}(\boldsymbol{\phi}_{m-1} \oplus \boldsymbol{\psi}_{m-1}) \otimes (\boldsymbol{\phi}_{m-1} \oplus \boldsymbol{\psi}_{m-1}) \\ = \{&\phi_{m-1,n_1} \otimes \phi_{m-1,n_2}, \\ &\phi_{m-1,n_1} \otimes \psi_{m-1,n_2}, \\ &\psi_{m-1,n_1} \otimes \phi_{m-1,n_2}, \\ &\psi_{m-1,n_1} \otimes \psi_{m-1,n_2}\}_{n_1,n_2}\end{aligned}$$

are both bases for $V_m \otimes V_m$. This second basis is orthogonal/orthonormal whenever the first basis is.

From this we observe that while the one-dimensional wavelet decomposition splits V_m into a direct sum of the two vector spaces V_{m-1} and W_{m-1}, the corresponding two-dimensional decomposition splits $V_m \otimes V_m$ into a direct sum of four tensor product vector spaces which deserve individual names.

Definition 8.8. We define the following tensor product spaces:

1. The space $W_m^{(0,1)}$ spanned by $\{\phi_{m,n_1} \otimes \psi_{m,n_2}\}_{n_1,n_2}$. This is also called the 01-subband, or the LH-subband,
2. The space $W_m^{(1,0)}$ spanned by $\{\psi_{m,n_1} \otimes \phi_{m,n_2}\}_{n_1,n_2}$. This is also called the 10-subband, or the HL-subband,
3. The space $W_m^{(1,1)}$ spanned by $\{\psi_{m,n_1} \otimes \psi_{m,n_2}\}_{n_1,n_2}$. This is also called the 11-subband, or the HH-subband.

The names L and H stand for *Low-pass filters* and *High-pass filters*, reflecting the interpretation of the corresponding filters G_0, G_1, H_0, H_1 as lowpass/high-pass filters. The use of the term subbands comes from the interpretation of these filters as being selective on a certain frequency band. The splitting of $V_m \otimes V_m$ into a direct sum of vector spaces can now be summed up as

$$V_m \otimes V_m = (V_{m-1} \otimes V_{m-1}) \oplus W_{m-1}^{(0,1)} \oplus W_{m-1}^{(1,0)} \oplus W_{m-1}^{(1,1)}. \tag{8.4}$$

Also in the setting of tensor products we refer to $V_{m-1} \otimes V_{m-1}$ as the space of low-resolution approximations. The remaining parts, $W_{m-1}^{(0,1)} \oplus W_{m-1}^{(1,0)} \oplus W_{m-1}^{(1,1)}$,

is refered to as the detail space. Note that the coordinate matrix of

$$\sum_{n_1,n_2=0}^{2^{m-1}N} (c_{m-1,n_1,n_2}(\phi_{m-1,n_1}\otimes\phi_{m-1,n_2}) + w^{(0,1)}_{m-1,n_1,n_2}(\phi_{m-1,n_1}\otimes\psi_{m-1,n_2}) + w^{(1,0)}_{m-1,n_1,n_2}(\psi_{m-1,n_1}\otimes\phi_{m-1,n_2}) + w^{(1,1)}_{m-1,n_1,n_2}(\psi_{m-1,n_1}\otimes\psi_{m-1,n_2})) \tag{8.5}$$

in the basis $(\boldsymbol{\phi}_{m-1}\oplus\boldsymbol{\psi}_{m-1})\otimes(\boldsymbol{\phi}_{m-1}\oplus\boldsymbol{\psi}_{m-1})$ is

$$\left(\begin{array}{ccc|ccc} c_{m-1,0,0} & \cdots & & w^{(0,1)}_{m-1,0,0} & \cdots & \\ \vdots & \vdots & & \vdots & \vdots & \\ \hline w^{(1,0)}_{m-1,0,0} & \cdots & & w^{(1,1)}_{m-1,0,0} & \cdots & \\ \vdots & \vdots & & \vdots & \vdots & \end{array}\right). \tag{8.6}$$

We see that the coordinate matrix is split into four submatrices:

- The c_{m-1}-values, i.e. the coordinates for $V_{m-1}\oplus V_{m-1}$. This is the upper left corner in (8.6), and is also called the 00-subband, or the LL-subband.
- The $w^{(0,1)}_{m-1}$-values, i.e. the coordinates for $W^{(0,1)}_{m-1}$. This is the upper right corner in (8.6), and corresponds to the LH-subband.
- The $w^{(1,0)}_{m-1}$-values, i.e. the coordinates for $W^{(1,0)}_{m-1}$. This is the lower left corner in (8.6), and corresponds to the HL-subband.
- The $w^{(1,1)}_{m-1}$-values, i.e. the coordinates for $W^{(1,1)}_{m-1}$. This is the lower right corner in (8.6), and corresponds to the HH-subband.

The $w^{(i,j)}_{m-1}$-values are as in the one-dimensional situation often refered to as wavelet coefficients. Let us consider the Haar wavelet as an example.

Example 8.9. If V_m is the vector space of piecewise constant functions on any interval of the form $[k2^{-m},(k+1)2^{-m})$ (as in the piecewise constant wavelet), $V_m\otimes V_m$ is the vector space of functions in two variables which are constant on any square of the form $[k_12^{-m},(k_1+1)2^{-m})\times[k_22^{-m},(k_2+1)2^{-m})$. 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 (8.4) are orthonormal, and we can thus use the orthogonal decomposition formula to find this projection. Clearly $\phi_{1,k_1}\otimes\phi_{1,k_2}$ has different support from all except one of $\phi_{0,n_1}\otimes\phi_{0,n_2}$. Since

$$\langle\phi_{1,k_1}\otimes\phi_{1,k_2},\phi_{0,n_1}\otimes\phi_{0,n_2}\rangle = 1/2$$

when the supports intersect, we obtain

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

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 8.1, together with the function $\phi\otimes\phi\in V_0\otimes V_0$.

Example 8.10. If we instead use any of the wavelets for piecewise linear functions, the wavelet basis functions are not orthogonal anymore, just as in the one-dimensional case. The new basis functions are shown in Figure 8.2 for the alternative piecewise linear wavelet.

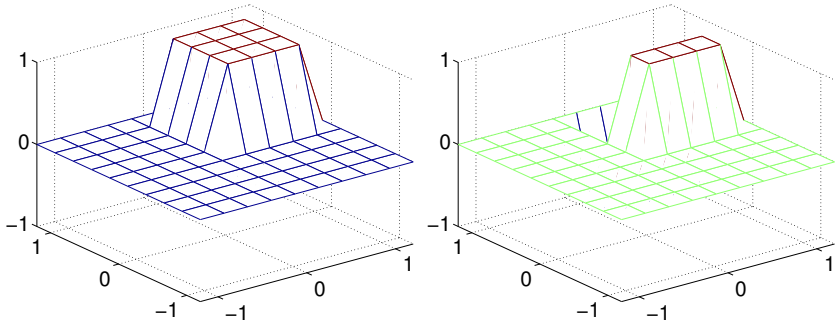
An immediate corollary of Theorem 8.6 is the following:

Corollary 8.11. Let

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

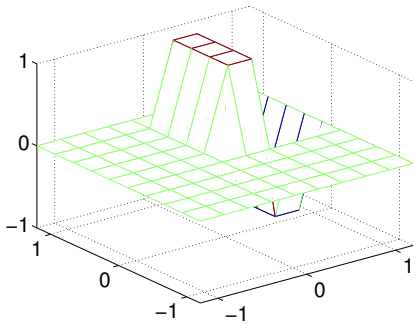
be the stages in the DWT and the IDWT, and let

$$X = (c_{m,i,j})_{i,j} \qquad Y = \begin{pmatrix} (c_{m-1,i,j})_{i,j} & (w^{(0,1)}_{m-1,i,j})_{i,j} \\ (w^{(1,0)}_{m-1,i,j})_{i,j} & (w^{(1,1)}_{m-1,i,j})_{i,j} \end{pmatrix} \tag{8.7}$$

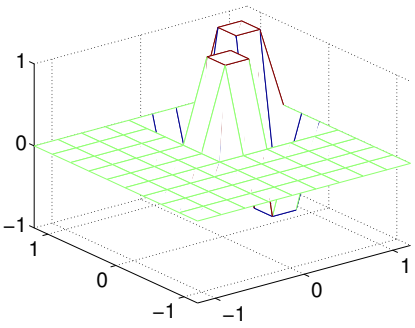


(a) The function $\phi \otimes \phi$

(b) The function $\phi \otimes \psi$

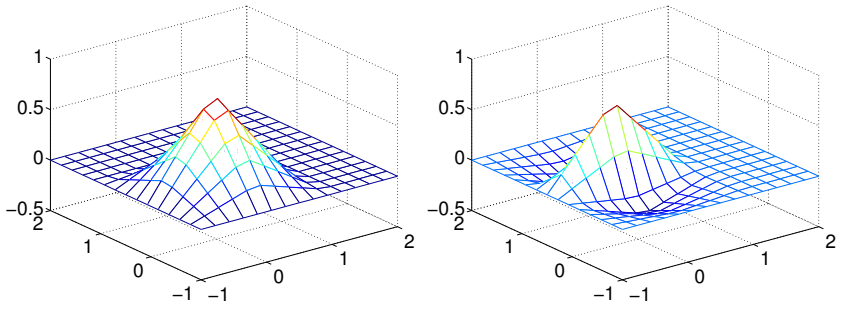


(c) The function $\psi \otimes \phi$



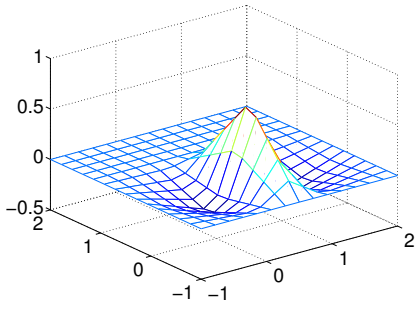
(d) The function $\psi \otimes \psi$

Figure 8.1: The basis functions for $(V_0 \otimes V_0) \oplus W_0^{(0,1)} \oplus W_0^{(1,0)} \oplus W_0^{(1,1)}$ for the Haar wavelet.

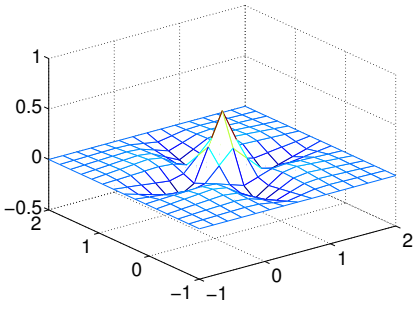


(a) The function $\phi \otimes \phi$

(b) The function $\phi \otimes \psi$



(c) The function $\psi \otimes \phi$



(d) The function $\psi \otimes \psi$

Figure 8.2: The basis functions for $(V_0 \otimes V_0) \oplus W_0^{(0,1)} \oplus W_0^{(1,0)} \oplus W_0^{(1,1)}$ for the alternative piecewise linear wavelet.

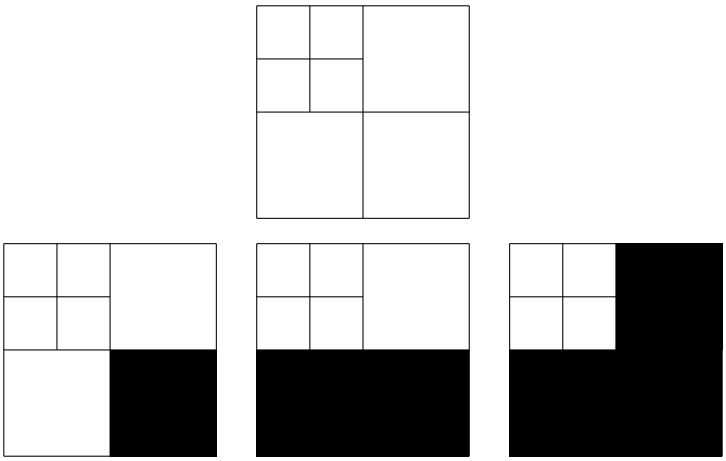


Figure 8.3: Graphical representation of neglecting the wavelet coefficients at the first level. In the first figure, all coefficients are present. Then we remove the ones from $W_1^{(1,1)}$, $W_1^{(1,0)}$, and $W_1^{(0,1)}$, respectively.

be the coordinate matrices in $\boldsymbol{\phi}_m \otimes \boldsymbol{\phi}_m$, and $(\boldsymbol{\phi}_{m-1} \oplus \boldsymbol{\psi}_{m-1}) \otimes (\boldsymbol{\phi}_{m-1} \oplus \boldsymbol{\psi}_{m-1})$, respectively. Then

$$Y = A_m X A_m^T \tag{8.8}$$

$$X = B_m Y B_m^T \tag{8.9}$$

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.

Each stage in DWT2 and IDWT2 can now be implemented by substituting the matrices A_m, B_m above into Theorem 7.13. This implementation can reuse an efficient implementation of the one-dimensional DWT/IDWT. When using many levels of the DWT2, the next stage is applied only to the upper left corner of the matrix, just as the DWT at the next stage only is applied to the first part of the coordinates. At each stage, the upper left corner of the coordinate matrix (which gets smaller at each iteration), is split into four equally big parts. To illustrate this, assume that we have a coordinate matrix, and that we perform the change of basis at two levels, i.e. we start with a coordinate matrix in the basis $\boldsymbol{\phi}_2 \otimes \boldsymbol{\phi}_2$. Figure 8.3 illustrates first the collection of all coordinates, and then the resulting collection of coordinates after removing subbands at the first level successively. The subbands which have been removed are indicated with a black colour. Figure 8.4 illustrates in the same way incremental removal of the subbands at the second level.

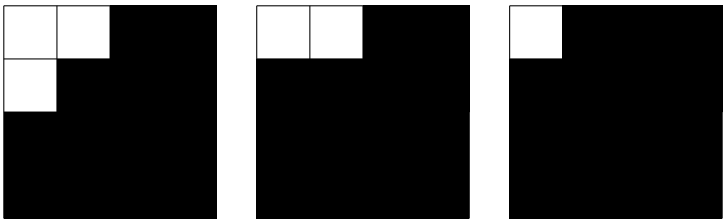


Figure 8.4: Graphical representation of neglecting the wavelet coefficients at the second level. We remove the ones from $W_2^{(1,1)}$, $W_2^{(1,0)}$, and $W_2^{(0,1)}$, respectively.

Figure 8.5: Image of Marilyn Monroe, used in our experiments.

Let us round off this section with some experiments with images using the wavelets we have considered[1]. Our theory is applied to images in the following way: We visualize the pixels in the image as coordinates in the basis $\boldsymbol{\phi}_m \otimes \boldsymbol{\phi}_m$ (so that the image has size $(2^m M) \times (2^m N)$), and perform change of coordinates with the DWT2. We can then, just as we did for sound, and for the DCT/DFT-values in images, either set the the part from the $W_k^{(i,j)}$-spaces (the detail) to zero, or the part from $V_0 \otimes V_0$ (the $M \times N$-low-resolution approximation) to zero, depending on whether we want to inspect the detail or the low-resolution approximation in the image. Finally we apply the IDWT2 to end up with coordinates in $\boldsymbol{\phi}_m \otimes \boldsymbol{\phi}_m$ again, and display the image with pixel values being these coordinates.

Example 8.12 (Creating thumbnail images)**.** Let us take the sample image of Marilyn Monroe, shown in Figure 8.5, and first use the Haar wavelet. In Exercise 1 you will be asked to implement a function which compute DWT2 for the Haar wavelet. 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 $m = 1$:

[1]Note also that Matlab has a wavelet toolbox which could be used for these purposes, but we will not go into the usage of this.

(a) (b) (c) (d)

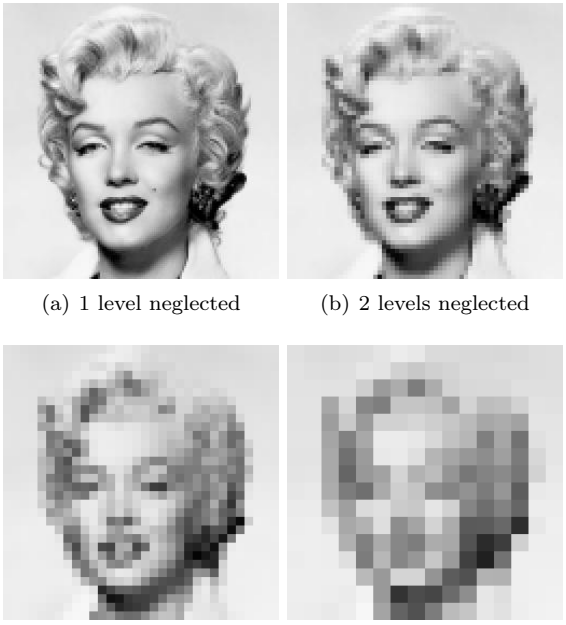
Figure 8.6: The corresponding thumbnail images for the Image of Marilyn Monroe, obtained with a DWT of 1, 2, 3, and 4 levels.

```
img = double(imread('mm.gif','gif'));
[l1,l2]=size(img);
x=DWT2HaarImpl(img,1);
x=x(1:(l1/2),1:(l2/2));
imwrite(uint8(x),'mm1thumbnail.jpg','jpg');
```

In Figure 8.6 the results are shown up to 4 resolutions.

Example 8.13 (Detail and low-resolution approximations with the Haar wavelet)**.** In Exercise 2 you will be asked to implement a function `showDWTlower` which displays the low-resolution approximations to our image test file `mm.gif`, for the Haar wavelet, using functions we implement in the exercises. Let us take a closer look at the images generated. Above we viewed the low-resolution approximation as a smaller image. Let us compare with the image resulting from setting 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 8.3 and Figure 8.4. 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 8.7 confirms this for the lower order resolution approximations. Alternatively, we should see that the higher order detail spaces contain more information. In Exercise 3 you will be asked to implement a function `showDWTlowerdifference` which displays the detail components in the image for a given resolution m for the Haar wavelet. The new images when this function is used are shown in Figure 8.8. The black colour indicates values which are close to 0. In other words, most of the coefficients are close to 0, which reflects one of the properties of the wavelet.

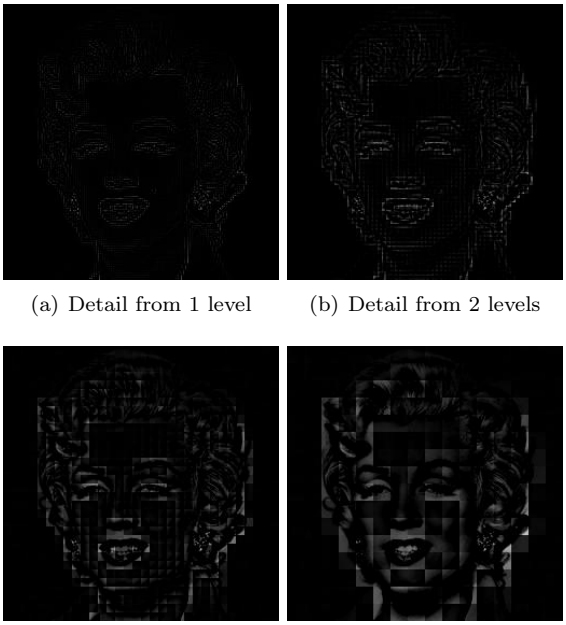
Example 8.14 (The alternative piecewise linear wavelet, and neglecting bands in the detail spaces)**.** In Exercise 5 you will be asked to implement a function `showDWTfilterslower` which displays the low-resolution approximations to our image test file `mm.gif`, for any type of wavelet, using functions we implement in the exercises. With this function we can display the result for all the wavelets we have considered up to now, in succession, and at a given resolution, with the following code:



(a) 1 level neglected (b) 2 levels neglected

(c) 3 levels neglected (d) 4 levels neglected

Figure 8.7: Image of Marilyn Monroe, with higher levels of detail neglected for the Haar wavelet.



(a) Detail from 1 level (b) Detail from 2 levels

(c) Detail from 3 levels (d) Detail from 4 levels

Figure 8.8: The corresponding detail for the images in Figure 8.7, with the Haar wavelet.

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

The call to `showDWTlower` first displays the result, using the Haar wavelet. The code then moves on to the piecewise linear wavelet and the alternative piecewise linear wavelet. In Example 5.50 we found the parameters `h0,h1,g0,g1` to use for these wavelets. These are then sent as parameters to the function `showDWTfilterslower`, which displays the corresponding image when a wavelet with a given set of filter coefficients are used.

Let us use some other filter than the Haar wavelet. Once the filters are known, and the image has been read from file, the functions `DWT2Impl` and `IDWT2Impl` from Exercise 4, a new image with detail at m levels neglected can be produced with the following code:

```
x=DWT2Impl(h0,h1,img,m);
newx=zeros(size(x));
newx(1:(l1/2^m),1:(l2/2^m))=x(1:(l1/2^m),1:(l2/2^m));
x=IDWT2Impl(g0,g1,newx,m);
```

We can repeat this for various number of levels, and compare the different images. We can also neglect only parts of the detail, since it at each level is grouped into three bands ($W_m^{(1,1)}$, $W_m^{(1,0)}$, $W_m^{(0,1)}$), contrary to the one-dimensional case. Let us use the alternative piecewise linear wavelet. This is used in the JPEG2000 standard for lossless compression, since the filter coefficients here turned out to be dyadic fractions, which are suitable for lossless operations. The resulting images when the bands on the first level indicated in Figure 8.3 are removed are shown in Figure 8.9. The resulting images when the bands on the second level indicated in Figure 8.4 are removed are shown in Figure 8.10. 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 8.11 we have also shown the resulting image after the third and fourth level of detail have been neglected. Although we still can see details in the image, the quality in the image is definitely poorer. Although the quality is poorer when we neglect levels of wavelet coefficients, all information is kept if

(a) The image unaltered

(b) Resulting image after neglecting detail in $W_1^{(1,1)}$, as illustrated in Figure 8.3(b)

(c) Resulting image after neglecting also detail in $W_1^{(1,0)}$, as illustrated in Figure 8.3(c).

(d) Resulting image after neglecting also detail in $W_1^{(0,1)}$, as illustrated in Figure 8.3(d).

Figure 8.9: Image of Marilyn Monroe, with various bands of detail at the first level neglected. The alternative piecewise linear wavelet was used.

(a) Resulting image after also neglecting detail in $W_2^{(1,1)}$, as illustrated in Figure 8.10(a).

(b) Resulting image after also neglecting detail in $W_2^{(1,0)}$, as illustrated in Figure 8.10(b).

(c) Resulting image after also neglecting detail in $W_2^{(0,1)}$, as illustrated in Figure 8.10(c).

Figure 8.10: Image of Marilyn Monroe, with various bands of detail at the second level neglected. The alternative piecewise linear wavelet was used.

(a) 3 levels neglected (b) 4 levels neglected

Figure 8.11: Image of Marilyn Monroe, with higher levels of detail neglected. The alternative piecewise linear wavelet was used.

we additionally include the detail/bands. In Figure 8.12, we have shown the corresponding detail for Figure 8.9(d), Figure 8.10(c), and Figure 8.11. Clearly, more detail can be seen in the image when more of the detail is included.

Example 8.15. The JPEG2000 standard uses a more advanced wavelet for lossy compression than the piecewise linear wavelet used for lossless compression. This uses a more advanced scaling function ϕ than the ones we have used, and adds more vanishing momemnts to it, similarly, to how we did for the alternative piecewise linear wavelet. We will not deduce the expression for this wavelet[2], only state that it is determined by the filters

$$H_0 = \{0.1562, -0.0985, -0.4569, 1.5589, \underline{3.5221}, 1.5589, -0.4569, -0.0985, 0.1562\}$$
$$H_1 = \{0.0156, -0.0099, \underline{-0.1012}, 0.1909, -0.1012, -0.0099, 0.0156\}.$$

for the DWT, and the filters

$$G_0 = \{-0.0156, -0.0099, 0.1012, \underline{0.1909}, 0.1012, -0.0099, -0.0156\}$$
$$G_1 = \{0.1562, 0.0985, -0.4569, \underline{-1.5589}, 3.5221, -1.5589, -0.4569, 0.0985, 0.1562\}$$

for the IDWT. The length of the filters are 9 and 7 in this case, so that this wavelet is called the CDF 9/7 wavelet (CDF represents the first letters in the names of the inventors of the wavelet). The corresponding frequency responses are

$$\lambda_{G_0}(\omega) = -0.0312\cos(3\omega), -0.0198\cos(2\omega), 0.2024\cos\omega + 0.1909$$
$$\lambda_{H_0}(\omega) = 0.3124\cos(4\omega) - 0.1970\cos(3\omega) - 0.9138\cos(2\omega), 3.1178\cos\omega + 3.5221.$$

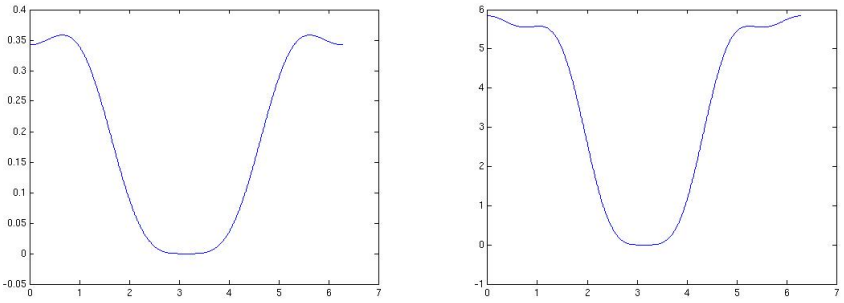
In Figure 8.13 we have plotted these. It is seen that both filters are lowpass

[2]it can be obtained by hand, but is more easily automated on a computer

(a) Detail from 1 level (b) Detail from 2 levels

(c) Detail from 3 levels (d) Detail from 4 levels

Figure 8.12: The corresponding detail for the image of Marilyn Monroe. The alternative piecewise linear wavelet was used.



(a) $\lambda_{G_0}(\omega)$ (b) $\lambda_{H_0}(\omega)$

Figure 8.13: The frequency responses for the filters used in lossy compression with JPEG2000.

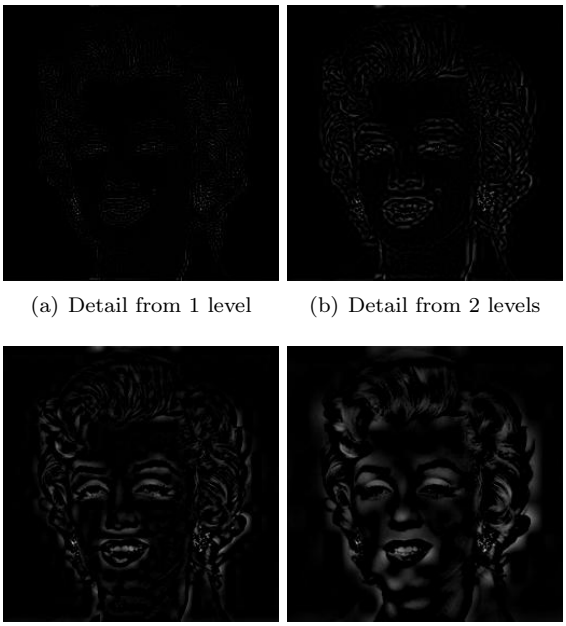
(a) 1 level neglected (b) 2 levels neglected

(c) 3 levels neglected (d) 4 levels neglected

Figure 8.14: Image of Marilyn Monroe, with higher levels of detail neglected. The CDF 9/7 wavelet was used.

filters also here, and that they are closer to an ideal bandpass filter. Although there exist wavelets with better properties when it comes to compression, it can be shown that this wavelet has a good tradeoff between complexity and compression, and is therefore much used. With the CDF 9/7 wavelet, we should see improved images when we discared the detail in the images. Figure 8.14 confirms this for the lower resolution spaces, while Figure 8.15 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 core software should be available free of charge, and JPEG2000 is part of most



(a) Detail from 1 level (b) Detail from 2 levels

(c) Detail from 3 levels (d) Detail from 4 levels

Figure 8.15: The corresponding detail for the image of Marilyn Monroe. The CDF 9/7 wavelet was used.

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

Exercises for Section 8.1

Ex. 1 — Implement functions

```
function xnew=DWT2HaarImpl(x,m)
function x=IDWT2HaarImpl(xnew,m)
```

which implements DWT2 and IDWT2 for the Haar-wavelet.

Ex. 2 — In this exercise we will experiment with applying an m-level DWT to a sound file.

a. Write a function

```
function showDWTlower(m)
```

which

1. reads the image file `mm.gif`,
2. performs an m-level DWT2 to the image samples using the function `DWT2HaarImpl`,
3. sets all wavelet coefficients representing detail to zero (i.e. keep only wavelet coefficients from $V_0 \otimes V_0$),
4. performs an IDWT2 on the resulting coefficients using the function `IDWT2HaarImpl`,
5. displays the resuting image.

b. Run the function `showDWTlower` for different values of m. Describe what you see for different m. degraded? Compare with what you saw with the function `showDCThigher` in Exercise 2, where you performed a DCT on the image samples instead, and set DCT coefficients below a given threshold to zero.

c. Do the image samples returned by `showDWTlower` lie in $[0, 255]$?

Ex. 3 — Repeat Exercise 2, but this time instead keep only wavelet coefficients from the detail spaces. Call the new function `showDWTlowerdifference`. What kind of image do you see? Can you recognize the original image in what you see?

Ex. 4 — Implement functions

```
function xnew=DWT2Impl(h0,h1,x,m)
function x=IDWT2Impl(g0,g1,xnew,m)
```

where `DWT2Impl` performs the m-level DWT2 on the image given by `x`, and `IDWT2Impl` performs the m-level IDWT2. The functions should at each stage call `DWTImpl` and `IDWTImpl` with $m = 1$, which you implemented in exercises 4 and 5, and each call to these functions should alter the appropriate upper left submatrix in the coordinate matrix. You can assume that the number of rows and columns both are powers of 2 in this matrix (but they may not be equal).

Ex. 5 — Write a function

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

which reimplements the function `showDWTlower` from Exercise 2 so that it takes as input the positive parts of the four different filters as in Section 5.6. Look at the result using the different wavelets we have encountered and for different m, using the code from Example 8.13. Can you see any difference from the Haar wavelet? If so, which wavelet gives the best image quality?

Ex. 6 — In this exercise we will change the code in Example 8.13 so that it instead only shows the contribution from the detail spaces.

a. Reimplement the function you made in Exercise 5 so that it instead shows the contribution from the detail spaces. Call the new function `showDWTfilterslowerdifference`.

b. In Exercise 3 we implemented a function `showDWTlowerdifference` for looking at the detail/error when the Haar wavelet is used. In the function `showDWTall` from Example 8.13, replace `showDWTlower` and `showDWTfilterslower` with `showDWTlowerdifference` and `showDWTfilterslowerdifference`. Describe the images you see for different m. Try to explain why the images seem to get clearer when you increase m.

8.2 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 alternative piecewise linear wavelet, and a new wavelet which is much used in lossy ompression of images. The experiments confirmed what we previously have proved, that wavelets with many vanishing moments are better suited for compression purposes.