Chapter 5

5.1

5.2

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 $(f(0), f(1), \ldots . f(N-1))$.
Solution: We have that $f(t) = \sum_{n=0}^{N-1} c_n \phi_{0,n}$, where c_n are the coordinates of f in the basis $\{\phi_{0,0}, \phi_{0,1}, \ldots, \phi_{0,N-1}\}$. We now get that

$$f(k) = \sum_{n=0}^{N-1} c_n \phi_{0,n}(k) = c_k,$$

since $\phi_{0,n}(k) = 0$ when $n \neq k$. This shows that $(f(0), f(1), \ldots . f(N-1))$ are the coordinates of f.

2. Prove Proposition 5.12.
Solution: Since f is constant and equal to $f(n)$ on $[n, n+1/2)$, and constant and equal to $f(n+1/2)$ on $[n+1/2, n+1)$, we get that

$$\begin{aligned}
\langle f, \phi_{0,n} \rangle &= \int_0^N f(t)\phi_{0,n}(t)dt = \int_n^{n+1} f(t)dt \\
&= \int_n^{n+1/2} f(t)dt + \int_{n+1/2}^{n+1} f(t)dt \\
&= \int_n^{n+1/2} f(n)dt + \int_{n+1/2}^{n+1} f(n+1/2)dt \\
&= f(n)/2 + f(n+1/2)/2 = (f(n) + f(n+1/2))/2.
\end{aligned}$$

The orthogonal decomposition theorem gives that

$$\text{proj}_{V_0} f = \sum_{n=0}^{N-1} \langle f, \phi_{0,n} \rangle \phi_{0,n} = \sum_{n=0}^{N-1} \frac{f(n) + f(n+1/2)}{2} \phi_{0,n}.$$

Since $\phi_{0,n}$ is 1 on $[n, n+1)$ and 0 elsewhere, $\text{proj}_{V_0} f$ is the piecewise constant function which is equal to $(f(n) + f(n+1/2))/2$ on $[n, n+1)$.

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

a. Consider the projection proj_{V_0} of V_1 onto V_0. Use lemma 5.11 to write down the matrix for proj_{V_0} relative to the bases $\boldsymbol{\phi}_1$ and $\boldsymbol{\phi}_0$.
Solution: From lemma 5.11 it follows that

$$\text{proj}_{V_0}(\phi_{1,2n}) = \phi_{0,n}/\sqrt{2}$$
$$\text{proj}_{V_0}(\phi_{1,2n+1}) = \phi_{0,n}/\sqrt{2}$$

This means that

$$[\text{proj}_{V_0}(\phi_{1,2n})]_{\boldsymbol{\phi}_0} = \boldsymbol{e}_n/\sqrt{2}$$
$$[\text{proj}_{V_0}(\phi_{1,2n+1})]_{\boldsymbol{\phi}_0} = \boldsymbol{e}_n/\sqrt{2}.$$

These are the columns in the matrix for proj_{V_0} relative to the bases $\boldsymbol{\phi}_1$ and $\boldsymbol{\phi}_0$. This matrix is thus

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

b. Similarly, use lemma 5.11 to write down the matrix for $\text{proj}_{W_0} : V_1 \to W_0$ relative to the bases $\boldsymbol{\phi}_1$ and $\boldsymbol{\psi}_0$.
Solution: From lemma 5.11 it follows that

$$\text{proj}_{W_0}(\phi_{1,2n}) = \psi_{0,n}/\sqrt{2}$$
$$\text{proj}_{W_0}(\phi_{1,2n+1}) = -\psi_{0,n}/\sqrt{2}$$

This means that

$$[\text{proj}_{W_0}(\phi_{1,2n})]_{\boldsymbol{\psi}_0} = \boldsymbol{e}_n/\sqrt{2}$$
$$[\text{proj}_{W_0}(\phi_{1,2n+1})]_{\boldsymbol{\psi}_0} = -\boldsymbol{e}_n/\sqrt{2}.$$

These are the columns in the matrix for proj_{W_0} relative to the bases $\boldsymbol{\phi}_1$ and $\boldsymbol{\psi}_0$. This matrix is thus

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

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

a. Explain why $\text{proj}_{V_0}(\phi) = \phi$ and $\text{proj}_{V_0}(\psi) = 0$.
Solution: Since $\phi \in V_0$ we must have that $\text{proj}_{V_0}(\phi) = \phi$. Since ψ is in the orthogonal complement of V_0 in V_1 we must have that $\text{proj}_{V_0}(\psi) = 0$.

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

$$[\text{proj}_{V_0}(\phi_{0,0})]_{(\boldsymbol{\phi}_0,\boldsymbol{\psi}_0)} = [\phi_{0,0}]_{(\boldsymbol{\phi}_0,\boldsymbol{\psi}_0)} = \boldsymbol{e}_0$$
$$[\text{proj}_{V_0}(\phi_{0,1})]_{(\boldsymbol{\phi}_0,\boldsymbol{\psi}_0)} = [\phi_{0,1}]_{(\boldsymbol{\phi}_0,\boldsymbol{\psi}_0)} = \boldsymbol{e}_1$$
$$\vdots$$

The last columns in the matrix of proj_{V_0} relative to $(\boldsymbol{\phi}_0, \boldsymbol{\psi}_0)$ are

$$[\text{proj}_{V_0}(\psi_{0,0})]_{(\boldsymbol{\phi}_0,\boldsymbol{\psi}_0)} = [\mathbf{0}]_{(\boldsymbol{\phi}_0,\boldsymbol{\psi}_0)} = \mathbf{0}$$
$$[\text{proj}_{V_0}(\psi_{0,1})]_{(\boldsymbol{\phi}_0,\boldsymbol{\psi}_0)} = [\mathbf{0}]_{(\boldsymbol{\phi}_0,\boldsymbol{\psi}_0)} = \mathbf{0}$$
$$\vdots$$

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

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

5. Show that

$$\text{proj}_{V_0}(f) = \sum_{n=0}^{N-1} \left(\int_n^{n+1} f(t)\,dt \right) \phi_{0,n}(t) \tag{5.1}$$

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

$$\text{proj}_{V_0}(f) = \sum_{n=0}^{N-1} \left(\int_0^{N} f(t)\phi_{0,n}(t)\,dt \right) \phi_{0,n} = \sum_{n=0}^{N-1} \left(\int_n^{n+1} f(t)\,dt \right) \phi_{0,n},$$

where we have used the orthogonal decomposition formula. Note also that, if $f(t) \in V_1$, and $f_{n,1}$ is the value f attains on $[n, n+1/2)$, and $f_{n,2}$ is the value f attains on $[n+1/2, n+1)$, we have that

$$\begin{aligned}\text{proj}_{V_0}(f) &= \sum_{n=0}^{N-1} \left(\int_n^{n+1} f(t)\,dt \right) \phi_{0,n}(t) \\ &= \sum_{n=0}^{N-1} \left(\frac{1}{2} f_{n,1} + \frac{1}{2} f_{n,2} \right) \phi_{0,n}(t) = \sum_{n=0}^{N-1} \frac{f_{n,1} + f_{n,2}}{2} \phi_{0,n}(t),\end{aligned}$$

which is the function which is $(f_{n,1} + f_{n,2})/2$ on $[n, n+1)$. This proves the first part of Proposition 5.12.

6. Show that

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

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

$$\begin{aligned}\|f - \text{proj}_{V_0}(f)\|^2 &= \langle f - \text{proj}_{V_0}(f), f - \text{proj}_{V_0}(f)\rangle \\ &= \langle f,f\rangle - 2\langle f, \text{proj}_{V_0}(f)\rangle + \langle \text{proj}_{V_0}(f), \text{proj}_{V_0}(f)\rangle\end{aligned}$$

Now, note that

$$\langle \text{proj}_{V_0}(f), \text{proj}_{V_0}(f)\rangle = \sum_{n=0}^{N-1}\left(\int_n^{n+1} f(t)dt\right)^2$$

from what we just showed in Exercise 5 (use that the $\phi_{0,n}$ are orthonormal). This means that the above can be written

$$\begin{aligned}&= \langle f,f\rangle - 2\sum_{n=0}^{N-1}\int_0^N \left(\int_n^{n+1} f(s)ds\right)\phi_{0,n}(t)f(t)dt + \sum_{n=0}^{N-1}\left(\int_n^{n+1} f(t)dt\right)^2 \\ &= \langle f,f\rangle - 2\sum_{n=0}^{N-1}\int_n^{n+1} \left(\int_n^{n+1} f(s)ds\right)f(t)dt + \sum_{n=0}^{N-1}\left(\int_n^{n+1} f(t)dt\right)^2 \\ &= \langle f,f\rangle - 2\sum_{n=0}^{N-1}\left(\int_n^{n+1} f(t)dt\right)^2 + \sum_{n=0}^{N-1}\left(\int_n^{n+1} f(t)dt\right)^2 \\ &= \langle f,f\rangle - \sum_{n=0}^{N-1}\left(\int_n^{n+1} f(t)dt\right)^2.\end{aligned}$$

7. Show that

$$\text{proj}_{W_0}(f) = \sum_{n=0}^{N-1}\left(\int_n^{n+1/2} f(t)dt - \int_{n+1/2}^{n+1} f(t)dt\right)\psi_{0,n}(t) \tag{5.2}$$

for any f. Show also that the second part of Proposition 5.12 follows from this.
Solution: The orthogonal decomposition theorem gives that

$$\begin{aligned}\text{proj}_{W_0}(f) &= \sum_{n=0}^{N-1}\langle f,\psi_{0,n}\rangle\psi_{0,n}(t) = \sum_{n=0}^{N-1}\left(\int_0^N f(t)\psi_{0,n}(t)dt\right)\psi_{0,n}(t) \\ &= \sum_{n=0}^{N-1}\left(\int_n^{n+1} f(t)\psi_{0,n}(t)dt\right)\psi_{0,n}(t) \\ &= \sum_{n=0}^{N-1}\left(\int_n^{n+1/2} f(t)dt - \int_{n+1/2}^{n+1} f(t)dt\right)\psi_{0,n}(t),\end{aligned}$$

where we used that $\psi_{0,n}$ is nonzero only on $[n,n+1)$, and is 1 on $[n,n+1/2)$, and -1 on $[n+1/2,n+1)$. Note also that, if $f(t)\in V_1$, and $f_{n,1}$ is the value f attains on

$[n, n+1/2)$, and $f_{n,2}$ is the value f attains on $[n+1/2, n+1)$, we have that

$$\begin{aligned}\mathrm{proj}_{W_0}(f) &= \sum_{n=0}^{N-1}\left(\int_n^{n+1/2} f(t)dt - \int_{n+1/2}^{n+1} f(t)dt\right)\psi_{0,n}(t)\\ &= \sum_{n=0}^{N-1}\left(\frac{1}{2}f_{n,1} - \frac{1}{2}f_{n,2}\right)\psi_{0,n}(t) = \sum_{n=0}^{N-1}\frac{f_{n,1}-f_{n,2}}{2}\psi_{0,n}(t),\end{aligned}$$

which is the function which is $(f_{n,1}-f_{n,2})/2$ on $[n, n+1/2)$, and $-(f_{n,1}-f_{n,2})/2$ on $[n+1/2, n+1)$. This proves the second part of Proposition 5.12.

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

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

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

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

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

$$c_{1,0} = \frac{1}{\sqrt{2}}(c_{0,0} + w_{0,0} + c_{0,(N+1)/2}) \qquad c_{1,N-1} = \frac{1}{\sqrt{2}}c_{0,(N+1)/2}. \tag{5.3}$$

a. If $N = 3$, the DWT matrix equals $\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1 & 1\\ 1 & -1 & 0\\ 0 & 0 & 1\end{pmatrix}$, and the inverse of this is $\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1 & -1\\ 1 & -1 & -1\\ 0 & 0 & 2\end{pmatrix}$. Explain from this that, when N is odd, the DWT matrix can be constructed by adding a column on the form $\frac{1}{\sqrt{2}}(-1, -1, 0, \ldots, 0, 2)$ to the DWT matrices we had for N even (in the last row zeros are also added). In terms of the coordinates, we thus have the additional formulas

$$c_{0,0} = \frac{1}{\sqrt{2}}(c_{1,0}+c_{1,1}-c_{1,N-1}) \quad w_{0,0} = \frac{1}{\sqrt{2}}(c_{1,0}-c_{1,1}-c_{1,N-1}) \quad c_{0,(N+1)/2} = \frac{1}{\sqrt{2}}2c_{1,N-1}. \tag{5.4}$$

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

5.3

1. Write a function `IDWTKernelHaar` which uses the formulas (5.3) to implement the IDWT, similarly to how the function `DWTKernelHaar` implemented the DWT using the formulas (5.4).
Solution: The following code can be used:

```
def IDWTKernelHaar(x, symm, dual):
    """
    Apply the IDWT kernel transformation for the Haar wavelet to x.

    x: The vector which we apply this kernel transformation to
    symm: Whether to apply symmetric extension to the input
    dual: Whether to apply the wavelet kernel or dual wavelet kernel.
    """
    x /= sqrt(2)
    if mod(len(x), 2)==1:
        a, b = x[0] + x[1]  + x[-1], x[0] - x[1]
        x[0], x[1] = a, b
        for k in range(2,len(x) - 2, 2):
            a, b = x[k] + x[k+1], x[k] - x[k+1]
            x[k], x[k+1] = a, b
    else:
        for k in range(0,len(x) - 1, 2):
            a, b = x[k] + x[k+1], x[k] - x[k+1]
            x[k], x[k+1] = a, b
```

2. Generalize exercise 4 to the projections from V_{m+1} onto V_m amd W_m.
Solution: Since $\phi_{m,n} \in V_m$ we must have that $T(\phi_{m,n}) = \phi_{m,n}$. Since $\psi_{m,n}$ is in the orthogonal complement of V_m in V_{m+1} we must have that $T(\psi_{m,n}) = 0$. The first half of the columns in the matrix of proj_{V_m} relative to $(\boldsymbol{\phi}_m, \boldsymbol{\psi}_m)$ are

$$
\begin{aligned}
[\text{proj}_{V_m}(\phi_{m,0})]_{(\boldsymbol{\phi}_m,\boldsymbol{\psi}_m)} &= [\phi_{m,0}]_{(\boldsymbol{\phi}_m,\boldsymbol{\psi}_m)} = \boldsymbol{e}_0 \\
[\text{proj}_{V_m}(\phi_{m,1})]_{(\boldsymbol{\phi}_m,\boldsymbol{\psi}_m)} &= [\phi_{m,1}]_{(\boldsymbol{\phi}_m,\boldsymbol{\psi}_m)} = \boldsymbol{e}_1 \\
&\vdots
\end{aligned}
$$

The second half of the columns are

$$
\begin{aligned}
[T(\psi_{m,0})]_{(\boldsymbol{\phi}_m,\boldsymbol{\psi}_m)} &= [\boldsymbol{0}]_{(\boldsymbol{\phi}_m,\boldsymbol{\psi}_m)} = \boldsymbol{0} \\
[T(\psi_{m,1})]_{(\boldsymbol{\phi}_m,\boldsymbol{\psi}_m)} &= [\boldsymbol{0}]_{(\boldsymbol{\phi}_m,\boldsymbol{\psi}_m)} = \boldsymbol{0} \\
&\vdots
\end{aligned}
$$

Thus, as before, the matrix of proj_{V_m} relative to $(\boldsymbol{\phi}_m, \boldsymbol{\psi}_m)$ is given by the diagonal matrix where the first half of the diagonal consists of 1's, and the second half consists of 0's. (c) follows in the same way.

3. Show that $f(t) \in V_m$ if and only if $g(t) = f(2t) \in V_{m+1}$.
Solution: If $f \in V_m$ we can write $f(t) = \sum_{n=0}^{2^m N-1} c_{m,n}\phi_{m,n}(t)$. We now get

$$\begin{aligned} g(t) = f(2t) &= \sum_{n=0}^{2^m N-1} c_{m,n}\phi_{m,n}(2t) = \sum_{n=0}^{2^m N-1} c_{m,n}2^{m/2}\phi(2^m 2t - n) \\ &= \sum_{n=0}^{2^m N-1} c_{m,n}2^{-1/2}2^{(m+1)/2}\phi(2^{m+1}t - n) = \sum_{n=0}^{2^m N-1} c_{m,n}2^{-1/2}\phi_{m+1,n}(t). \end{aligned}$$

This shows that $g \in V_{m+1}$. To prove the other way, assume that $g(t) = f(2t) \in V_{m+1}$. This means that we can write $g(t) = \sum_{n=0}^{2^{m+1}N-1} c_{m+1,n}\phi_{m+1,n}(t)$. We now have

$$\begin{aligned} f(t) = g(t/2) &= \sum_{n=0}^{2^{m+1}N-1} c_{m+1,n}\phi_{m+1,n}(t/2) = \sum_{n=0}^{2^{m+1}N-1} c_{m+1,n}2^{(m+1)/2}\phi(2^m t - n) \\ &= \sum_{n=0}^{2^m N-1} c_{m+1,n}2^{(m+1)/2}\phi(2^m t - n) + \sum_{n=2^m N}^{2^{m+1}N-1} c_{m+1,n}2^{(m+1)/2}\phi(2^m t - n) \\ &= \sum_{n=0}^{2^m N-1} c_{m+1,n}2^{(m+1)/2}\phi(2^m t - n) + \sum_{n=0}^{2^m N-1} c_{m+1,n+2^m N}2^{(m+1)/2}\phi(2^m t - n - 2^m N) \\ &= \sum_{n=0}^{2^m N-1} c_{m+1,n}2^{(m+1)/2}\phi(2^m t - n) + \sum_{n=0}^{2^m N-1} c_{m+1,n+2^m N}2^{(m+1)/2}\phi(2^m t - n) \\ &= \sum_{n=0}^{2^m N-1} (c_{m+1,n} + c_{m+1,n+2^m N})2^{1/2}2^{m/2}\phi(2^m t - n) \\ &= \sum_{n=0}^{2^m N-1} (c_{m+1,n} + c_{m+1,n+2^m N})2^{1/2}\phi_{m,n}(t) \in V_m \end{aligned}$$

The thing which made this a bit difficult was that the range of the n-indices here was outside $[0, 2^m N-1]$ (which describe the legal indices in the basis V_m), so that we had to use the periodicty of ϕ.

4. 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,$\ldots$,T_n, written as $T_1 \oplus T_2 \oplus \ldots \oplus T_n$, denotes the linear transformation from $C_1 \oplus C_2 \oplus \cdots \oplus C_n$ to itself defined by

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

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

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

Here two new concepts are used: a direct sum of matrices, and a direct sum of linear transformations.
Solution: By definition, $[T_1]_{\mathscr{B}_1} \oplus [T_2]_{\mathscr{B}_2} \oplus \cdots \oplus [T_n]_{\mathscr{B}_n}$ is a block matrix where the blocks on the diagonal are the matrices $[T_1]_{\mathscr{B}_1}$, $[T_2]_{\mathscr{B}_2}$, and so on. If $\boldsymbol{b}_i$ are the basis

vectors in $\mathscr{B}_i$, the columns in $[T_i]_{\mathscr{B}_i}$ are $[T(\boldsymbol{b}_j)]_{\mathscr{B}_i}$. This means that $[T_1]_{\mathscr{B}_1} \oplus [T_2]_{\mathscr{B}_2} \oplus \cdots \oplus [T_n]_{\mathscr{B}_n}$ has $[T(\boldsymbol{b}_j)]_{\mathscr{B}_i}$ in the j'th block, and $\mathbf{0}$ elsewhere. This means that we can write it as

$$\mathbf{0} \oplus \cdots \mathbf{0} \oplus [T(\boldsymbol{b}_j)]_{\mathscr{B}_i} \oplus \mathbf{0} \cdots \mathbf{0}.$$

On the other hand, $[T_1 \oplus T_2 \oplus \ldots \oplus T_n]_{(\mathscr{B}_1, \mathscr{B}_2, \ldots, \mathscr{B}_n)}$ is a matrix of the same size, and the corresponding column to that of the above is

$$\begin{aligned}
&[(T_1 \oplus T_2 \oplus \ldots \oplus T_n)(\mathbf{0} \oplus \cdots \mathbf{0} \oplus \boldsymbol{b}_j \oplus \mathbf{0} \cdots \mathbf{0})]_{(\mathscr{B}_1, \mathscr{B}_2, \ldots, \mathscr{B}_n)} \\
&= [\mathbf{0} \oplus \cdots \mathbf{0} \oplus T(\boldsymbol{b}_j) \oplus \mathbf{0} \cdots \mathbf{0}]_{(\mathscr{B}_1, \mathscr{B}_2, \ldots, \mathscr{B}_n)} \\
&= \mathbf{0} \oplus \cdots \mathbf{0} \oplus [T(\boldsymbol{b}_j)]_{\mathscr{B}_i} \oplus \mathbf{0} \cdots \mathbf{0}.
\end{aligned}$$

Here $\boldsymbol{b}_j$ occurs as the i'th summand. This is clearly the same as what we computed for the right hand side above.

5. 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?
Solution: Assume that λ is an eigenvalue common to both T_1 and T_2. Then there exists a vector $\boldsymbol{v}_1$ so that $T_1\boldsymbol{v}_1 = \lambda\boldsymbol{v}_1$, and a vector $\boldsymbol{v}_2$ so that $T_2\boldsymbol{v}_2 = \lambda\boldsymbol{v}_2$. We now have that

$$\begin{aligned}
(T_1 \oplus T_2)(\boldsymbol{v}_1 \oplus \boldsymbol{v}_2) &= \begin{pmatrix} T_1 & 0 \\ 0 & T_2 \end{pmatrix} \begin{pmatrix} \boldsymbol{v}_1 \\ \boldsymbol{v}_2 \end{pmatrix} \\
&= \begin{pmatrix} T_1\boldsymbol{v}_1 \\ T_2\boldsymbol{v}_2 \end{pmatrix} = \begin{pmatrix} \lambda\boldsymbol{v}_1 \\ \lambda\boldsymbol{v}_2 \end{pmatrix} \\
&= \lambda \begin{pmatrix} \boldsymbol{v}_1 \\ \boldsymbol{v}_2 \end{pmatrix} = \lambda(\boldsymbol{v}_1 \oplus \boldsymbol{v}_2).
\end{aligned}$$

This shows that λ is an eigenvalue for λ also, and that $\boldsymbol{v}_1 \oplus \boldsymbol{v}_2$ is a corresponding eigenvector.

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

$$\begin{aligned}
(A \oplus B)(A^{-1} \oplus B^{-1}) &= \begin{pmatrix} A & 0 \\ 0 & B \end{pmatrix} \begin{pmatrix} A^{-1} & 0 \\ 0 & B^{-1} \end{pmatrix} \\
&= \begin{pmatrix} AA^{-1} & 0 \\ 0 & BB^{-1} \end{pmatrix} = \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix} = I
\end{aligned}$$

where we have multiplied as block matrices. This proves that $A \oplus B$ is invertible, and states what the inverse is.

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

$$(A \oplus B)(C \oplus D) = \begin{pmatrix} A & 0 \\ 0 & B \end{pmatrix} \begin{pmatrix} C & 0 \\ 0 & D \end{pmatrix} = \begin{pmatrix} AC & 0 \\ 0 & BD \end{pmatrix} = (AC) \oplus (BD)$$

where we again have multiplied as block matrices.

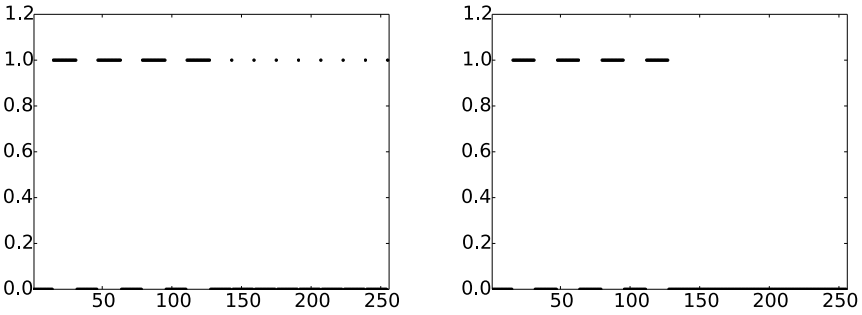

Figure 5.1: 2 vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ which seem equal, but where the DWT's are very different.

8. Assume that you run an m-level DWT on a vector of length r. What value of N does this correspond to? Note that an m-level DWT performs a change of coordinates from $\boldsymbol{\phi}_m$ to $(\boldsymbol{\phi}_0, \boldsymbol{\psi}_0, \boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_{m-2}, \boldsymbol{\psi}_{m-1})$.

9. In Figure 5.1 we have plotted the DWT's of two vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. In both vectors we have 16 ones followed by 16 zeros, and this pattern repeats cyclically so that the length of both vectors is 256. The only difference is that the second vector is obtained by delaying the first vector with one element. You see that the two DWT's are very different: For the first vector we see that there is much detail present (the second part of the plot), while for the second vector there is no detail present. Attempt to explain why this is the case. Based on your answer, also attempt to explain what can happen if you change the point of discontinuity for the piecewise constant function in Figure 5.20(a) to something else.

10. 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.
Solution: The following code achieves this:

```
# Exercise 5.3.10
[x, fs] = audioread('castanets.wav')
DWTImpl(x[0:2**17,0], 2, DWTKernelHaar)
plot(x[0:2**17,0], axis = [0,2**17,-1,1])
```

The values from V_0 corresponds to the first $1/4$ values in the plot, the values from W_0 corresponds to the next $1/4$ values in the plot, while the values from W_1 correspond to the last $1/2$ of the values in the plot.

11. In this exercise we will experiment with applying an m-level DWT to a sound file.
Solution: The following code achieves the task

```
def playDWT(m, f, invf, lowres = True):
    """
    Play a sound after removing either the detail or the lowres part.

    m: The number of resolutions
    f: The DWT kernel
    invf: The IDWT kernel
    lowres: If true, set the detail to 0 and play the lowres part.
            If false, set the lowres part to 0 and play the detail.
    """
    x, fs = audioread('castanets.wav')
    N = 2**17
    x = x[0:N]
    DWTImpl(x, m, f)
    if lowres:
        x[(N/2**m):N] = 0
    else:
        x[0:(N/2**m)] = 0
    IDWTImpl(x, m, invf)
    play(x, fs)
```

a. Write a function `playDWT` which takes m, a DWT kernel, an IDWT kernel, and a variable `lowres` as input, and

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 `DWTImpl` with the given DWT kernel,
3. sets all wavelet coefficients representing detail to zero if `lowres` is true (i.e. keep only the coordinates from $\boldsymbol{\phi}_0$ in the basis $(\boldsymbol{\phi}_0, \boldsymbol{\psi}_0, \boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_{m-2}, \boldsymbol{\psi}_{m-1})$),
4. sets all low-resolution coefficients to seros if `lowres` is false (i.e. zero out the coordinates from $\boldsymbol{\phi}_0$ and keep the others),
5. performs an IDWT on the resulting coefficients using the function `IDWTImpl` with the given IDWT kernel,
6. plays the resulting sound.

b. Run the function `playDWT` with `DWTKernelHaar` and `IDWTKernelHaar` as inputs, and for different values of m, when the low-resolution approximation is chosen. For which m can you hear that the sound gets degraded? How does it get degraded? Compare with what you heard through the function `playDFT` in Example 2.30, where you performed a DFT on the sound sample instead, and set some of the DFT coefficients to zero.
Solution: For $m = 2$ we clearly hear a degradation in the sound. For $m = 4$ and above most of the sound is unrecognizable.

c. Do the sound samples returned by `playDWT` lie in $[-1,1]$?
Solution: There is no reason to believe that sound samples returned by the function lie in $[-1,1]$. you can check this by printing the maximum value in the returned array on screen inside this method.

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

13. Repeat the listening experiment from Exercise 11, but this time instead keep only wavelet coefficients from the detail spaces $W_0, W_1, \ldots$. What kind of sound do you hear? Can you recognize the original sound in what you hear?

14. Compute the wavelet detail coefficients analytically for the functions in Example 5.20, 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 5.21.
Solution: Note first that, similarly to the computation in Exercise 7 in Section 5.2, we have that

$$\int_0^N f(t)\psi_{m,n}(t)\,dt = 2^{m/2}\left(\int_{n2^{-m}}^{(n+1/2)2^{-m}} f(t)\,dt - \int_{(n+1/2)2^{-m}}^{(n+1)2^{-m}} f(t)\,dt\right).$$

With $f(t) = 1 - 2|1/2 - t/N|$ we have two possibilities: when $n < N2^{m-1}$ we have that $[n2^{-m}, (n+1)2^{-m}) \subset [0, N/2]$, so that $f(t) = 2t/N$, and we get

$$\begin{aligned} w_{m,n} &= 2^{m/2}\left(\int_{n2^{-m}}^{(n+1/2)2^{-m}} 2t/N\,dt - \int_{(n+1/2)2^{-m}}^{(n+1)2^{-m}} 2t/N\,dt\right) \\ &= 2^{m/2}[t^2/N]_{n2^{-m}}^{(n+1/2)2^{-m}} - 2^{m/2}[t^2/N]_{(n+1/2)2^{-m}}^{(n+1)2^{-m}} \\ &= \frac{2^{-3m/2}}{N}\left(2(n+1/2)^2 - n^2 - (n+1)^2\right) = -\frac{2^{-3m/2-1}}{N}. \end{aligned}$$

When $n \geq N2^{m-1}$ we have that $f(t) = 2 - 2t/N$, and using that $\int_0^N \psi_{m,n}(t)\,dt = 0$ we must get that $w_{m,n} = \frac{2^{-3m/2-1}}{N}$.

For $f(t) = 1/2 + \cos(2\pi t/N)/2$, note first that this has the same coefficients as $\cos(2\pi t/N)/2$, since $\int_0^N \psi_{m,n}(t)\,dt = 0$. We now get

$$\begin{aligned} w_{m,n} &= 2^{m/2}\left(\int_{n2^{-m}}^{(n+1/2)2^{-m}} \cos(2\pi t/N)/2\,dt - \int_{(n+1/2)2^{-m}}^{(n+1)2^{-m}} \cos(2\pi t/N)/2\,dt\right) \\ &= 2^{m/2}[N\sin(2\pi t/N)/(4\pi)]_{n2^{-m}}^{(n+1/2)2^{-m}} - 2^{m/2}[N\sin(2\pi t/N)/(4\pi)]_{(n+1/2)2^{-m}}^{(n+1)2^{-m}} \\ &= \frac{2^{m/2-2}N}{\pi}\left(2\sin(2\pi(n+1/2)2^{-m}/N) - \sin(2\pi n2^{-m}/N) - \sin(2\pi(n+1)2^{-m}/N)\right). \end{aligned}$$

There seems to be no more possibilities for simplification here.

15. 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 5.21. How do these compare with the coefficients from the Exercise 14?

Solution: We get

$$\begin{aligned} w_{m,n} &= 2^{m/2}\left(\int_{n2^{-m}}^{(n+1/2)2^{-m}} (t/N)^k\,dt - \int_{(n+1/2)2^{-m}}^{(n+1)2^{-m}} (t/N)^k\,dt\right) \\ &= 2^{m/2}[t^{k+1}/((k+1)N^k)]_{n2^{-m}}^{(n+1/2)2^{-m}} - 2^{m/2}[t^{k+1}/((k+1)N^k)]_{(n+1/2)2^{-m}}^{(n+1)2^{-m}} \\ &= \frac{2^{-m(k+1/2)}}{(k+1)N^k}\left(2(n+1/2)^{k+1} - n^{k+1} - (n+1)^{k+1}\right). \end{aligned}$$

The leading term n^{k+1} will here cancel, but the others will not, so there is no room for further simplification here.

16. Suppose that we have the vector $\boldsymbol{x}$ with length $2^{10} = 1024$, defined by $x_n = 1$ for n even, $x_n = -1$ for n odd. What will be the result if you run a 10-level DWT on $\boldsymbol{x}$? Use the function `DWTImpl` to verify what you have found.
Hint: We defined ψ by $\psi(t) = (\phi_{1,0}(t) - \phi_{1,1}(t))/\sqrt{2}$. From this connection it follows that $\psi_{9,n} = (\phi_{10,2n} - \phi_{10,2n+1})/\sqrt{2}$, and thus $\phi_{10,2n} - \phi_{10,2n+1} = \sqrt{2}\psi_{9,n}$. Try to couple this identity with the alternating sign you see in $\boldsymbol{x}$.
Solution: The vector $\boldsymbol{x}$ is the coordinate vector of the function $f(t) = \sum_{n=0}^{1023}(-1)^n\phi_{10,n}$ in the basis $\boldsymbol{\phi}_{10}$ for V_{10}. Since $\phi_{10,2n} - \phi_{10,2n+1} = \sqrt{2}\psi_{9,n}$, we can write $f(t) = \sum_{n=0}^{1023}\sqrt{2}\psi_{9,n}$. Since a 10-level-DWT gives as a result the coordinate vector of f in

$$(\boldsymbol{\phi}_0, \boldsymbol{\psi}_0, \boldsymbol{\psi}_1, \boldsymbol{\psi}_2, \boldsymbol{\psi}_3, \boldsymbol{\psi}_4, \boldsymbol{\psi}_5, \boldsymbol{\psi}_6, \boldsymbol{\psi}_7, \boldsymbol{\psi}_8, \boldsymbol{\psi}_9),$$

(the DWT is nothing but the change of coordinates from $\boldsymbol{\phi}_{10}$ to this basis), and since $f(t) = \sum_{n=0}^{1023}\sqrt{2}\psi_{9,n}$, it is clear that the coordinate vector of f in this basis has $\sqrt{2}$ in the second part (the $\boldsymbol{\psi}_9$-koordinatene), and 0 elsewhere. The 10-level DWT of $\boldsymbol{x}$ therefore gives the vector of length 1024 which is 0 on the first half, and equal to $\sqrt{2}$ on the second half. $m = 10$ is here arbitrarily chosen: The result would have been the same for $m = 1$, $m = 2$, and so on. The following code verifies the result:

```
# Exercise 5.3.16
x = tile([1.,-1.], 512)
DWTImpl(x, 10, DWTKernelHaar)
print x
```

17. Use the results from exercise 8 in Section 5.2 to rewrite the implementations `DWTKernelHaar` and `IDWTKernelHaar` so that they also work in the case when N is odd.
Solution: The following code can be used.

```
def DWTKernelHaar(x):
    x /= sqrt(2)
    if mod(len(x), 2)==1:
        a, b = x[0] + x[1] - x[-1], x[0] - x[1] - x[-1]
        x[0], x[1] = a, b
        x[-1] *= 2
    else:
        a, b = x[0] + x[1], x[0] - x[1]
```

```
        x[0], x[1] = a, b
    for k in range(2,len(x) - 1,2):
        a, b = x[k] + x[k+1], x[k] - x[k+1]
        x[k], x[k+1] = a, b
```

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

5.4

1. Show that, for $f \in V_0$ we have that $[f]_{\boldsymbol{\phi}_0} = (f(0), f(1), \ldots, f(N-1))$. This generalizes the result for piecewise constant functions. Let us write $f(t) = \sum_{n=0}^{N-1} c_n \phi_{0,n}(t)$. If k is an integer we have that

$$f(k) = \sum_{n=0}^{N-1} c_n \phi_{0,n}(k) = \sum_{n=0}^{N-1} c_n \phi(k-n).$$

Clearly the only integer for which $\phi(s) \neq 0$ is $s = 0$ (since $\phi(0) = 1$), so that the only n which contributes in the sum is $n = k$. This means that $f(k) = c_k$, so that $[f]_{\boldsymbol{\phi}_0} = (f(0), f(1), \ldots, f(N-1))$.

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

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

$$\begin{aligned}
\int_n^{n+1} f(t)^2\,dt &= \int_n^{n+1} (x_n + (x_{n+1} - x_n)(t-n))^2\,dt = \int_0^1 (x_n + (x_{n+1} - x_n)t)^2\,dt \\
&= \int_0^1 (x_n^2 + 2x_n(x_{n+1} - x_n)t + (x_{n+1} - x_n)^2 t^2)\,dt \\
&= \left[x_n^2 t + x_n(x_{n+1} - x_n)t^2 + (x_{n+1} - x_n)^2 t^3/3\right]_0^1 \\
&= x_n^2 + x_n(x_{n+1} - x_n) + (x_{n+1} - x_n)^2/3 = \frac{1}{3}(x_n^2 + x_n x_{n+1} + x_{n+1}^2).
\end{aligned}$$

b. Show that

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

Solution: We have that

$$\begin{aligned}
&\int_0^{1/2} (x_0 + (x_1 - x_0)\,t)\phi_{1,1}(t)\,dt \\
&= \int_0^{1/2} (x_0 + (x_1 - x_0)\,t) 2\sqrt{2} t\,dt = 2\sqrt{2}\int_0^{1/2} (x_0\,t + (x_1 - x_0)\,t^2)\,dt \\
&= 2\sqrt{2}\left[\frac{1}{2}x_0\,t^2 + \frac{1}{3}(x_1 - x_0)\,t^3\right]_0^{1/2} = 2\sqrt{2}\left(\frac{1}{8}x_0 + \frac{1}{24}(x_1 - x_0))\right) \\
&= 2\sqrt{2}\left(\frac{1}{12}x_0 + \frac{1}{24}x_1\right).
\end{aligned}$$

In the same way

$$\begin{aligned}
&\int_{1/2}^1 (x_0 + (x_1 - x_0)\,t)\phi_{1,1}(t)\,dt \\
&= \int_{1/2}^1 (x_0 + (x_1 - x_0)\,t) 2\sqrt{2}(1-t)\,t\,dt = 2\sqrt{2}\int_{1/2}^1 (x_0 + (x_1 - 2x_0)\,t - (x_1 - x_0)\,t^2)\,t\,dt \\
&= 2\sqrt{2}\left[x_0\,t + \frac{1}{2}(x_1 - 2x_0)\,t^2 - \frac{1}{3}(x_1 - x_0)\,t^3\right]_{1/2}^1 = 2\sqrt{2}\left(\frac{1}{2}x_0 + \frac{3}{8}(x_1 - 2x_0) - \frac{7}{24}(x_1 - x_0)\right) \\
&= 2\sqrt{2}\left(\frac{1}{24}x_0 + \frac{1}{12}x_1\right).
\end{aligned}$$

c. Use the fact that

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

and a. and b. to find an expression for $\|\phi_{1,1}(t) - \sum_{n=0}^{N-1} x_n \phi_{0,n}(t)\|^2$.
Solution: Using a. and b. we see that the above can be written as

$$\begin{aligned}
&\frac{2}{3} + \sum_{n=0}^{N-1} \frac{1}{3}(x_n^2 + x_n x_{n+1} + x_{n+1}^2) - 2\left(2\sqrt{2}\left(\frac{1}{12}x_0 + \frac{1}{24}x_1\right) - 2\sqrt{2}\left(\frac{1}{24}x_0 + \frac{1}{12}x_1\right)\right) \\
&= \frac{2}{3} + \frac{2}{3}\sum_{n=0}^{N-1} x_n^2 + \frac{1}{3}\sum_{n=0}^{N-1} x_n x_{n+1} - \frac{\sqrt{2}}{2}(x_0 + x_1).
\end{aligned}$$

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

with components $b_0 = b_1 = \sqrt{2}/2$, and $b_k = 0$ for $k \geq 2$.
Solution: We see that the partial derivatives of the function in c. are

$$\begin{aligned}
\frac{\partial f}{\partial x_0} &= \frac{1}{3}x_{N-1} + \frac{4}{3}x_0 + \frac{1}{3}x_1 - \frac{\sqrt{2}}{2} \\
\frac{\partial f}{\partial x_1} &= \frac{1}{3}x_0 + \frac{4}{3}x_1 + \frac{1}{3}x_1 - \frac{\sqrt{2}}{2} \\
\frac{\partial f}{\partial x_i} &= \frac{1}{3}x_{i-1} + \frac{4}{3}x_i + \frac{1}{3}x_{i+1}\ 2 \leq i < N-1 \\
\frac{\partial f}{\partial x_{N-1}} &= \frac{1}{3}x_{N-2} + \frac{4}{3}x_{N-1} + \frac{1}{3}x_0.
\end{aligned}$$

Moving the two terms $\frac{\sqrt{2}}{2}$ over to the right hand side, setting the gradient equal to zero is the same as solving the system $S\boldsymbol{x} = \boldsymbol{b}$ which we stated.

e. Solve the system in d. for some values of N to verify that the projection of $\phi_{1,1}$ onto V_0 is nonzero, and that its support covers the entire $[0, N]$.
Solution: The following code can be used

```
from numpy import *
import matplotlib.pyplot as plt

N = 16
S = zeros((N, N))
S[0,N-1] = 1/3.; S[0,0] = 4/3.; S[0,1] = 1/3.; # First row
for k in range(1,N-1):
    S[k,(k-1):(k+2)] = [1/3., 4/3., 1/3.]
S[N-1,N-2] = 1/3.; S[N-1,N-1]=4/3.; S[N-1,0]=1/3.; # Last row
b=zeros(N); b[0]=sqrt(2)/2; b[1]=sqrt(2)/2;
plt.plot(range(0,N),linalg.solve(S,b)) # Plots the projection
plt.show()
```

3. Show that

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

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

$$\begin{aligned}
\langle \phi_{0,n}, \phi_{0,n} \rangle &= \int_{n-1}^{n+1} (1 - |t-n|)^2 \, dt \\
&= \int_{n-1}^{n+1} \left(1 - 2|t-n| + (t-n)^2\right) dt \\
&= 2 - 2 + \left[\frac{1}{3}(t-n)^3\right]_{n-1}^{n+1} = \frac{2}{3}.
\end{aligned}$$

We also have

$$\begin{aligned}
\langle \phi_{0,n}, \phi_{0,n+1} = \int_n^{n+1} (1-(t-n))(1+(t-n-1))\, dt &= \int_0^1 (1-u)(1+u-1)\, du \\
= \int_0^1 (t-t^2)\, dt = \frac{1}{2} - \frac{1}{3} &= \frac{1}{6}.
\end{aligned}$$

Finally, the supports of $\phi_{0,n}$ and $\phi_{0,n\pm k}$ are disjoint for $k > 1$, so that we must have $\langle \phi_{0,n}, \phi_{0,n\pm k} \rangle = 0$ in that case.

4. 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.
Solution: We have that

$$\chi_{[-1/2,1/2)} * \chi_{[-1/2,1/2)}(x) = \int_{-\infty}^{\infty} \chi_{[-1/2,1/2)}(t)\chi_{[-1/2,1/2)}(x-t)dt.$$

The integrand here is 1 when $-1/2 < t < 1/2$ and $-1/2 < x - t < 1/2$, or in other words when $\max(-1/2, -1/2 + x) < t < \min(1/2, 1/2 + x)$ (else it is 0). When $x > 0$ this happens when $-1/2 + x < t < 1/2$, and when $x < 0$ this happens when $-1/2 < t < 1/2 + x$. This means that

$$\chi_{[-1/2,1/2)} * \chi_{[-1/2,1/2)}(x) = \begin{cases} \int_{-1/2+x}^{1/2} dt = 1 - x & , x > 0 \\ \int_{-1/2}^{1/2+x} dt = 1 + x & , x < 0. \end{cases}$$

But this is by definition ϕ.

5.5

1. In this exercise we will show that there is a unique function on the form (5.32) which has two vanishing moments.

a. Show that, when $\hat{\psi}$ is defined by (5.32), we have that

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

Solution: The function $\hat{\psi}$ is a sum of the functions $\psi = \phi_{1,1}$, ϕ, and $\phi_{0,1}$ (i.e. we have set $n = 0$ in Equation (5.32)). All these are continuous and piecewise

linear, and we can write

$$\phi_{1,1}(t) = \begin{cases} 2t & 0 \le t < 1/2 \\ 2-2t & 1/2 \le t < 1 \\ 0 & \text{elsewhere} \end{cases}$$

$$\phi(t)(t) = \begin{cases} 1+t & -1 \le t < 0 \\ 1-t & 0 \le t < 1 \\ 0 & \text{elsewhere} \end{cases}$$

$$\phi_{0,1}(t) = \begin{cases} t & 0 \le t < 1 \\ 2-t & 1 \le t < 2 \\ 0 & \text{elsewhere} \end{cases}.$$

It follows that $\hat{\psi}(t) = \phi_{1,1}(t) - \alpha\phi(t) - \beta\phi_{1,1}$ is piecewise linear, and linear on the segments $[-1,0]$, $[0,1/2]$, $[1/2,1]$, $[1,2]$.

On the segment $[-1,0]$ only the function ϕ is seen to be nonzero, and since $\phi(t) = 1+t$ here, we have that $\hat{\psi}(t) = -\alpha(1+t) = -\alpha - \alpha t$ here.

On the segment $[0,1/2]$ all three functions are nonzero, and

$$\begin{aligned} \phi_{1,1}(t) &= 2t \\ \phi(t)(t) &= 1-t \\ \phi_{0,1}(t) &= t \end{aligned}$$

on this interval. This means that $\hat{\psi}(t) = 2t - \alpha(1-t) - \beta t = (2+\alpha-\beta)t - \alpha$ on $[0,1/2]$.

On the segment $[0,1/2]$ all three functions are nonzero, and

$$\begin{aligned} \phi_{1,1}(t) &= 2-2t \\ \phi(t)(t) &= 1-t \\ \phi_{0,1}(t) &= t \end{aligned}$$

on this interval. This means that $\hat{\psi}(t) = 2-2t-\alpha(1-t)-\beta t = (\alpha-\beta-2)t-\alpha+2$ on $[1/2,1]$.

On the segment $[1,2]$ only the function $\phi_{0,1}$ is seen to be nonzero, and since $\phi_{0,1}(t) = 2-t$ here, we have that $\hat{\psi}(t) = -\beta(2-t) = \beta t - 2\beta$ here. For all other values of t, $\hat{\psi}$ is zero. This proves the formulas for $\hat{\psi}$ on the different intervals.

b. Show that

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

Solution: We can write

$$\begin{aligned}
\int_0^N \hat{\psi}(t)\,dt &= \int_{-1}^2 \hat{\psi}(t)\,dt = \int_{-1}^0 \hat{\psi}(t)\,dt + \int_0^{1/2} \hat{\psi}(t)\,dt + \int_{1/2}^1 \hat{\psi}(t)\,dt + \int_1^2 \hat{\psi}(t)\,dt \\
&= \int_{-1}^0 (-\alpha-\alpha t)\,dt + \int_0^{1/2} (2+\alpha-\beta)t-\alpha)\,dt \\
&+ \int_{1/2}^1 ((\alpha-\beta-2)t-\alpha+2)\,dt + \int_1^2 (\beta t-2\beta)\,dt \\
&= \left[-\alpha t - \frac{1}{2}\alpha t^2\right]_{-1}^0 + \left[\frac{1}{2}(2+\alpha-\beta)t^2-\alpha t\right]_0^{1/2} \\
&+ \left[\frac{1}{2}(\alpha-\beta-2)t^2+(2-\alpha)t\right]_{1/2}^1 + \left[\frac{1}{2}\beta t^2-2\beta t\right]_1^2 \\
&= -\alpha + \frac{1}{2}\alpha + \frac{1}{8}(2+\alpha-\beta) - \frac{1}{2}\alpha + \frac{3}{8}(\alpha-\beta-2) + \frac{1}{2}(2-\alpha) + \frac{3}{2}\beta - 2\beta \\
&= \frac{1}{2} - \alpha - \beta,
\end{aligned}$$

$\int_0^N t\hat{\psi}(t)\,dt$ is computed similarly, so that we in the end arrive at $\frac{1}{4}-\beta$.

c. Explain why there is a unique function on the form (5.32) which has two vanishing moments, and that this function is given by Equation (5.34).
Solution: The equation system

$$\begin{aligned}
\frac{1}{2} - \alpha - \beta &= 0 \\
\frac{1}{4} - \beta &= 0
\end{aligned}$$

has the unique solution $\alpha = \beta = \frac{1}{4}$, which we already have found.

2. In the previous exercise we ended up with a lot of calculations to find α, β in Equation (5.32). Let us try to make a program which does this for us, and which also makes us able to generalize the result.

a. Define

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

for $k \geq 0$. Explain why finding α, β so that we have two vanishing moments in Equation 5.32 is equivalent to solving the following equation:

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

Write a program which sets up and solves this system of equations, and use this program to verify the values for α, β we previously have found.

Hint: you can integrate functions in Python with the function quad in the package `scipy.integrate`. As an example, the function $\phi(t)$, which is nonzero only on $[-1,1]$, can be integrated as follows:

```
res, err = quad(lambda t: t**k*(1-abs(t)), -1, 1)
```

Solution: In order for ψ to have vanishing moments we must have that $\int \hat{\psi}(t)dt = \int t\hat{\psi}(t)dt = 0$ Substituting $\hat{\psi} = \psi - \alpha\phi_{0,0} - \beta\phi_{0,1}$ we see that, for $k = 0, 1$,

$$\int t^k \left(\alpha\phi_{0,0} + \beta\phi_{0,1}\right) dt = \int t^k \psi(t)dt.$$

The left hand side can here be written

$$\begin{aligned}\int t^k \left(\alpha\phi_{0,0} + \beta\phi_{0,1}\right) dt &= \alpha \int t^k \phi_{0,0} dt + \beta \int t^k \phi_{0,1}(t)dt \\ &= \alpha \int_{-1}^{1} t^k (1-|t|)dt + \beta \int_0^2 t^k (1-|t-1|)dt = \alpha a_k + \beta b_k.\end{aligned}$$

The right hand side is

$$\int t^k \psi(t)dt = \int t^k \phi_{1,1}(t)dt = \int_0^1 (1-2|t-1/2|)dt = e_k.$$

The following program sets up the corresponding equation systems, and solves it by finding α, β.

```
# Exercise 5.5.2a
A = zeros((2, 2))
b = zeros((2, 1))
for k in range(2):
    res1, err1 = quad(lambda t: t**k*(1-abs(t)), -1, 1)
    res2, err2 = quad(lambda t: t**k*(1-abs(t-1)), 0, 2)
    res3, err3 = quad(lambda t: t**k*(1-2*abs(t-1/2.)), 0, 1)
    A[k,:] = [res1, res2]
    b[k] = res3
linalg.solve(A,b)
```

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

$$\hat{\psi} = \psi_{0,0} - \alpha\phi_{0,0} - \beta\phi_{0,1} - \gamma\phi_{0,-1} - \delta\phi_{0,2}. \tag{5.5}$$

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

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

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

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

and solve this with your computer.
Solution: Similarly to a., Equation (5.5) gives that

$$\int t^k \left(\alpha\phi_{0,0} + \beta\phi_{0,1} + \gamma\phi_{0,-1} + \delta\phi_{0,2}\right) dt = \int t^k \psi(t) dt.$$

The correspodning equation system is deduced exactly as in a. The following program sets up the corresponding equation systems, and solves it by finding $\alpha, \beta, \gamma, \delta$.

```
# Exercise 5.5.2b
A=zeros((4, 4))
b=zeros((4, 1))
for k in range(4):
    res1, err1 = quad(lambda t: t**k*(1-abs(t)), -1, 1)
    res2, err2 = quad(lambda t: t**k*(1-abs(t-1)), 0, 2)
    res3, err3 = quad(lambda t: t**k*(1-abs(t+1)), -2, 0)
    res4, err4 = quad(lambda t: t**k*(1-abs(t-2)), 1, 3)
    res5, err5 = quad(lambda t: t**k*(1-2*abs(t-1/2.)), 0, 1)
    A[k,:] = [res1, res2, res3, res4]
    b[k] = res5
coeffs = linalg.solve(A,b)
```

c. Plot the function defined by (5.5), 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 $\phi_{1,1}(t)$.
Solution: The function $\hat{\psi}$ now is supported on $[-2,3]$, and can be plotted as follows:

```
# Exercise 5.5.2c
t=linspace(-2,3,100)
plot(t, (t >= 0)*(t <= 1)*(1-2*abs(t - 0.5)) \
        -coeffs[0]*(t >= -1)*(t <= 1)*(1 - abs(t)) \
        -coeffs[1]*(t >= 0)*(t <= 2)*(1 - abs(t - 1)) \
        -coeffs[2]*(t >= -2)*(t <= 0)*(1 - abs(t + 1)) \
        -coeffs[3]*(t >= 1)*(t <= 3)*(1 - abs(t - 2)))
```

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

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

Hint: You can also compare with Equation (5.37) 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.
Solution: If we define

$$\hat{\psi} = \psi_{0,0} - \sum_{k=0}^{K} \left(\alpha_k \phi_{0,-k} - \beta_k \phi_{0,k+1}\right),$$

we have $2k$ unknowns. These can be determined if we require $2k$ vanishing moments.

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

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

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

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

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

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

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

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

$$\begin{pmatrix} c_{0,0} & c_{0,1} & \cdots & c_{0,k-1} \\ c_{1,0} & c_{1,1} & \cdots & c_{1,k-1} \\ \vdots & \vdots & \vdots & \vdots \\ c_{k-1,0} & c_{k-1,1} & \cdots & c_{k-1,k-1} \end{pmatrix} \begin{pmatrix} a_0 \\ \text{ï¿¡}a_1 \\ \vdots \\ a_{k-1} \end{pmatrix} = \begin{pmatrix} e_0 \\ e_1 \\ \vdots \\ e_{k-1} \end{pmatrix} \tag{5.6}$$

b. Write a function `vanishingmomshaar` which takes k as input, solves Equation 5.6, and returns the vector $\boldsymbol{a} = (a_0, a_1, \ldots, a_{k-1})$.

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

```
# Play detail components for the Haar wavelet
playDWT(m, DWTKernelHaar, IDWTKernelHaar, False)
```

```
# Play detail components for the piecewise linear wavelet
playDWT(m, DWTKernelpwl0, IDWTKernelpwl0, False)
```

```
# Play detail components for the alternative piecewise linear wavelet
playDWT(m, DWTKernelpwl2, IDWTKernelpwl2, False)
```

5.6

1. Prove Theorem 5.43. Use the proof of Theorem 4.9 as a guide.
Solution: We compute

$$S_r\boldsymbol{x} = \begin{pmatrix} S_1 & S_2 \end{pmatrix} \begin{pmatrix} x_0 \\ \vdots \\ x_{N-2} \\ x_{N-1} \\ x_{N-2} \\ \vdots \\ x_1 \end{pmatrix} = S_1 \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix} + S_2 \begin{pmatrix} x_{N-2} \\ \vdots \\ x_1 \end{pmatrix} = S_1 \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix} + (S_2)^f \begin{pmatrix} x_1 \\ \vdots \\ x_{N-2} \end{pmatrix}$$

$$= S_1 \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix} + \begin{pmatrix} 0 & (S_2)^f & 0 \end{pmatrix} \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix} = (S_1 + \begin{pmatrix} 0 & (S_2)^f & 0 \end{pmatrix})\boldsymbol{x},$$

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

2. In this exercise we will establish an orthonormal basis for the symmetric extensions, as defined by Definition 5.42. This parallels Theorem 4.6.

> **a.** Explain why, if $\boldsymbol{x} \in \mathbb{R}^{2N-2}$ is a symmetric extension (according to definition 4.1), then $(\widehat{\boldsymbol{x}})_n = z_n e^{-\pi i n}$, where $\boldsymbol{z}$ is a real vectors which satisfies $z_n = z_{2N-2-n}$
> **Solution**: Using Theorem 4.3 with $d = N-1$ and with $2N-2$ for N, we obtain that
>
> $$(\widehat{\boldsymbol{x}})_n = z_n e^{-2\pi i d n/(2N-2)} = z_n e^{-2\pi i (N-1) n/(2N-2)} = z_n e^{-\pi i n},$$
>
> where $\boldsymbol{z}$ is a real vectors which satisfies $z_n = z_{2N-2-n}$.

b. Show that

$$\left\{\boldsymbol{e}_0, \left\{\frac{1}{\sqrt{2}}(\boldsymbol{e}_i + \boldsymbol{e}_{2N-2-i})\right\}_{n=1}^{N-2}, \boldsymbol{e}_{N-1}\right\} \tag{5.7}$$

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

Solution: Clearly these vectors are an orthonormal basis for the set of vectors where $z_n = z_{2N-2-n}$. The vectors from a. are obtained by multiplying these with $e^{-\pi i n}$. But the orthonormality of these vectors are not affected when we multiply with $e^{-\pi i n}$, so we may skip this.

c. Show that

$$\begin{gathered}\frac{1}{\sqrt{2N-2}}\cos\left(2\pi\frac{0}{2N-2}k\right) \\ \left\{\frac{1}{\sqrt{N-1}}\cos\left(2\pi\frac{n}{2N-2}k\right)\right\}_{n=1}^{N-2} \\ \frac{1}{\sqrt{2N-2}}\cos\left(2\pi\frac{N-1}{2N-2}k\right)\end{gathered} \tag{5.8}$$

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

Solution: We compute the IDFT for all vectors in (b). Since the IDFT is unitary, this will give us an orthonormal basis for the symmetric vectors in $\mathbb{R}^{2N-2}$. Since $(F_N)^H\boldsymbol{\phi}_n = \boldsymbol{e}_n$ we get that

$$\begin{aligned}(F_N)^H\boldsymbol{e}_0 = \boldsymbol{\phi}_0 &= \frac{1}{\sqrt{2N-2}}\cos\left(2\pi\frac{0}{2N-2}k\right) \\ (F_N)^H\left(\frac{1}{\sqrt{2}}(\boldsymbol{e}_n + \boldsymbol{e}_{2N-2-n})\right) &= \frac{1}{\sqrt{2}}\left(\boldsymbol{\phi}_n + \boldsymbol{\phi}_{2N-2-n}\right) \\ &= \frac{1}{\sqrt{2}}\frac{1}{\sqrt{2N-2}}\left(e^{2\pi i k n/(2N-2)} + e^{-2\pi i k n/(2N-2)}\right) \\ &= \frac{1}{\sqrt{N-1}}\cos\left(2\pi\frac{n}{2N-2}k\right) \\ (F_N)^H\boldsymbol{e}_{N-1} = \boldsymbol{\phi}_{N-1} &= \frac{1}{\sqrt{2N-2}}\cos\left(2\pi\frac{N-1}{2N-2}k\right).\end{aligned}$$

These coincide with the vectors listed in the exercise.

d. Assume that S is symmetric. Show that the vectors listed in (5.8) are eigenvectors for S_r, when the vectors are viewed as vectors in $\mathbb{R}^N$, and that they are linearly independent. This shows that S_r is diagonalizable.

Solution: Since S is symmetric, it preserves vectors which are symmetric around $N-1$. In the frequency domain, applying S to a vector listed in (5.8) corresponds to multiplying the vectors listed in 5.7 with the frequency response. Since this does not introduce any more components, it is clear that the new vector must be a multiplum of the same vector, so that these vectors

indeed are eigenvectors. But then the vectors restricted to $\mathbb{R}^N$ are also eigenvectors for S_r, since this is simply S when viewed on the first N elements. Since the vectors in $\mathbb{R}^{2N-2}$ are linearly independent, it is imemdiate that the corresponding vectors in $\mathbb{R}^N$ also are linearly independent, since the second part of the vectors mirror the first part.

3. Let us explain how the matrix S_r can be diagonalized, similarly to how we previously diagonalized using the DCT. In Exercise 2 we showed that the vectors

$$\left\{\cos\left(2\pi\frac{n}{2N-2}k\right)\right\}_{n=0}^{N-1} \tag{5.9}$$

in $\mathbb{R}^N$ is a basis of eigenvectors for S_r when S is symmetric. S_r itself is not symmetric, however, so that this basis can not possibly be orthogonal (S is symmetric if and only if it is orthogonally digonalizable). However, when the vectors are viewed in $\mathbb{R}^{2N-2}$ we showed in Exercise 2.c an orthogonality statement which can be written as

$$\sum_{k=0}^{2N-3}\cos\left(2\pi\frac{n_1}{2N-2}k\right)\cos\left(2\pi\frac{n_2}{2N-2}k\right) = (N-1)\times\begin{cases}2 & \text{if } n_1=n_2\in\{0,N-1\}\\ 1 & \text{if } n_1=n_2\notin\{0,N-1\}\\ 0 & \text{if } n_1\neq n_2\end{cases}. \tag{5.10}$$

a. Show that

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

Hint: Use that $\cos x=\cos(2\pi-x)$ to pair the summands k and $2N-2-k$.
Solution: Using that $\cos x=\cos(2\pi-x)$ we can here pair the summands k and $2N-2-k$ to obtain

$$\begin{aligned}&\sum_{k=0}^{2N-3}\cos\left(2\pi\frac{n_1}{2N-2}k\right)\cos\left(2\pi\frac{n_2}{2N-2}k\right)\\ &=\cos\left(2\pi\frac{n_1}{2N-2}\cdot 0\right)\cos\left(2\pi\frac{n_2}{2N-2}\cdot 0\right)\\ &\quad+2\sum_{k=1}^{N-2}\cos\left(2\pi\frac{n_1}{2N-2}k\right)\cos\left(2\pi\frac{n_2}{2N-2}k\right)\\ &\quad+\cos\left(2\pi\frac{n_1}{2N-2}(N-1)\right)\cos\left(2\pi\frac{n_2}{2N-2}(N-1)\right).\end{aligned}$$

If we divide by 2 and combine these equations we get the result.

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

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

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

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

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

is orthogonal.

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

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

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

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

Chapter 6

6.1

1. Write down the corresponding filters G_0 og G_1 for Exercise 3 in Section 5.5. Plot their frequency responses, and characterize the filters as lowpass- or highpass filters.

2. Find two symmetric filters, so that the corresponding MRA-matrix, constructed with alternating rows from these two filters, is not a symmetric matrix.
Solution: You can set for instance $H_0 = \{1/4, \underline{1/2}, 1/4\}$, and $H_1 = \{\underline{1}\}$ (when you write down the corresponding matrix you will see that $A_{0,1} = 1/2$, $A_{1,0} = 0$, so that the matrix is not symmetric)

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

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

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

Write down the compact form for the corresponding filters H_0, H_1, and compute and plot the frequency responses. Are the filters symmetric?
Solution: We have that $H_0 = \frac{1}{5}\{1, 1, \underline{1}, 1, 1\}$, and $H_1 = \frac{1}{3}\{-1, \underline{1}, -1\}$. The frequency responses are

$$\begin{aligned} \lambda_{H_0}(\omega) &= \frac{1}{5}e^{2i\omega} + \frac{1}{5}e^{i\omega} + \frac{1}{5} + \frac{1}{5}e^{-i\omega}\frac{1}{5}e^{-2i\omega} \\ &= \frac{2}{5}\cos(2\omega) + \frac{2}{5}\cos\omega + \frac{1}{5} \\ \lambda_{H_1}(\omega) &= -\frac{1}{3}e^{i\omega} + \frac{1}{3} - \frac{1}{3}e^{-i\omega} = -\frac{2}{3}\cos\omega + \frac{1}{3}. \end{aligned}$$

Both filters are symmetric.

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

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

Write down the compact form for the filters G_0, G_1, and compute and plot the frequency responses. Are the filters symmetric?
Solution: We have that $G_0 = \{1/4, \underline{1/2}, 1/4\}$, and $G_1 = \{1/16, -1/4, \underline{3/8}, -1/4, 1/16\}$. The frequency responses are

$$\begin{aligned} \lambda_{G_0}(\omega) &= \frac{1}{4}e^{i\omega} + \frac{1}{2} + \frac{1}{4}e^{-i\omega} \\ &= \frac{1}{2}\cos(\omega) + \frac{1}{2} \\ \lambda_{G_1}(\omega) &= \frac{1}{16}e^{2i\omega} - \frac{1}{4}e^{i\omega} + \frac{3}{8} - \frac{1}{4}e^{-i\omega}\frac{1}{16}e^{-2i\omega} \\ &= \frac{1}{8}\cos(2\omega) - \frac{1}{2}\cos\omega + \frac{3}{8}. \end{aligned}$$

Both filters are symmetric.

6. Assume that $H_0 = \{1/16, 1/4, \underline{3/8}, 1/4, 1/16\}$, and $H_1 = \{-1/4, \underline{1/2}, -1/4\}$. Plot the frequency responses of H_0 and H_1, and verify that H_0 is a lowpass filter, and that H_1 is a highpass filter. Also write down the change of coordinate matrix $P_{\mathscr{C}_1 \leftarrow \boldsymbol{\phi}_1}$ for the wavelet corresponding to these filters.
Solution: The frequency responses are

$$\begin{aligned} \lambda_{H_0}(\omega) &= \frac{1}{16}e^{2i\omega} + \frac{1}{4}e^{i\omega} + \frac{3}{8} + \frac{1}{4}e^{-i\omega}\frac{1}{16}e^{-2i\omega} \\ &= \frac{1}{8}\cos(2\omega) + \frac{1}{2}\cos\omega + \frac{3}{8} \\ \lambda_{H_1}(\omega) &= -\frac{1}{4}e^{i\omega} + \frac{1}{2} - \frac{1}{4}e^{-i\omega} \\ &= -\frac{1}{2}\cos(\omega) + \frac{1}{2}. \end{aligned}$$

The two first rows in $P_{\mathscr{C}_1 \leftarrow \boldsymbol{\phi}_1}$ are

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

The remaining rows are obtained by translating these in alternating order.

7. Assume that $G_0 = \frac{1}{3}\{1,\underline{1},1\}$, and $G_1 = \frac{1}{5}\{1,-1,\underline{1},-1,1\}$. Plot the frequency responses of G_0 and G_1, and verify that G_0 is a lowpass filter, and that G_1 is a highpass filter. Also write down the change of coordinate matrix $P_{\boldsymbol{\phi}_1 \leftarrow \mathscr{C}_1}$ for the wavelet corresponding to these filters.
Solution: The frequency responses are

$$\begin{aligned}\lambda_{G_0}(\omega) &= \frac{1}{3}e^{i\omega} + \frac{1}{3} + \frac{1}{3}e^{-i\omega} = \frac{2}{3}\cos\omega + \frac{1}{3} \\ \lambda_{G_1}(\omega) &= \frac{1}{5}e^{2i\omega} - \frac{1}{5}e^{i\omega} + \frac{1}{5} - \frac{1}{5}e^{-i\omega}\frac{1}{5}e^{-2i\omega} \\ &= \frac{2}{5}\cos(2\omega) - \frac{2}{5}\cos\omega + \frac{1}{5}\end{aligned}$$

The two first columns in $P_{\boldsymbol{\phi}_1 \leftarrow \mathscr{C}_1}$ are

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

The remaining columns are obtained by translating these in alternating order.

8. In Exercise 8 in Section 5.3 we computed the DWT of two very simple vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, using the Haar wavelet.

> **a.** Compute $H_0\boldsymbol{x}_1$, $H_1\boldsymbol{x}_1$, $H_0\boldsymbol{x}_2$, and $H_1\boldsymbol{x}_2$, where H_0 and H_1 are the filters used by the Haar wavelet.
>
> **b.** Compare the odd-indexed elements in $H_1\boldsymbol{x}_1$ with the odd-indexed elements in $H_1\boldsymbol{x}_2$. From this comparison, attempt to find an explanation to why the two vectors have very different detail components.

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

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

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

a. Comment the code and explain what happens. Which wavelet is used? What do the vectors `c` and `w` represent? Describe the sound you believe you will hear.
Solution: `c` and `w` represent the coordinates in the wavelet bases $\boldsymbol{\phi}_0$ and $\boldsymbol{\psi}_0$. The code runs a Haar wavelet transform. The sound is normalized so that the sound samples lie in the range between -1 and 1, and the resulting sound is played. The sound is split into two parts, and `c` represents a low-resolution version of the sound (with half the number of samples), so that we first will hear the sound played at double pace. After this we will hear the detail `w` in the sound, also played at double pace. We should also be able to recognize the sound from this detail.

b. Assume that we add lines in the code above which sets the elements in the vector `w` to 0 before we compute the inverse operation. What will you hear if you play the new sound you then get?
Solution: This corresponds to reconstructing a low-resolution approximation of the sound.

10. Let us return to the piecewise linear wavelet from Exercise 2 in Section 5.5.

a. With $\hat{\psi}$ as defined as in Exercise 2 b. in Section 5.5, compute the coordinates of $\hat{\psi}$ in the basis $\boldsymbol{\phi}_1$ (i.e. $[\hat{\psi}]_{\boldsymbol{\phi}_1}$) with $N = 8$, i.e. compute the IDWT of

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

which is the coordinate vector you computed in Exercise 2 d. in Section 5.5. For this, you should use the function `IDWTImpl`, with the kernel of the piecewise linear wavelet without symmetric extension as input. Explain that this gives you the filter coefficients of G_1.
Solution: The code which can be used looks like this:

```
# Exercise 6.1.10a
g1 = array([-coeffs[0],-coeffs[1],-coeffs[3],0,0,0,0,-coeffs[2],\
            1, 0, 0, 0, 0, 0, 0, 0])
IDWTImpl(g1, 1, IDWTKernelpwl0, 0)
g1 = hstack([g1[13:16],g1[0:6]]) # Compact filter notation
```

Note that we have used a kernel which does not make symmetric extensions.

b. Plot the frequency response of G_1.
Solution: The code can look as follows:

```
# Exercise 6.1.10b
omega = linspace(0,2*pi,100)
plot(omega, g1[4] + g1[5]*2*cos(omega) + g1[6]*2*cos(2*omega) \
          + g1[7]*2*cos(3*omega) + g1[8]*2*cos(4*omega))
```

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

12. In Exercise 6 in Section 3.1 we implemented a symmetric filter applied to a vector, i.e. when a periodic extension is assumed. The corresponding function was called `filterS(t, x)`, and used the function `numpy.convolve`.

a. Reimplement the function `filterS` so that it also takes a third parameter `symm`. If `symm` is false a periodic extension of `x` should be performed (i.e. filtering as we have defined it, and as the previous version of `filterS` performs it). If `symm` is true, symmetric extensions should be used (as given by Definition 5.42).
Solution: The code can look like this:

```
def filterS(t, x, symm):
    tlen = len(t)
    N0 = (tlen - 1)/2
    N = len(x)

    if symm:
        y = concatenate([ x[N0:0:(-1)], x, x[(N-2):(N - N0 - 2):(-1)] ])
    else:
        y = concatenate([ x[(N - N0):], x, x[:N0]])
    y = convolve(t, y)
    y = y[(2*N0):(length(y)-2*N0)]
```

b. Implement functions tt DWTKernelFilters(H0, H1, G0, G1, x, symm, dual) and `IDWTKernelFilters(H0, H1, G0, G1, x, symm, dual)` which compute the DWT and IDWT kernels using theorems 6.3 and 6.5, respectively. This function thus bases itself on that the filters of the wavelet are known. The functions should call the function `filterS` from a.. Recall also the definition of the parameter `dual` from this section.
Solution: The code can look like this:

```
def DWTKernelFilters(H0, H1, G0, G1, x, symm, dual):
    f0, f1 = H0, H1
    if dual:
        f0, f1 = G0, G1
    N = len(x)
    x0 = filterS(f0, x, symm)
    x1 = filterS(f1, x, symm)
    x[::2] = x0[::2]
    x[1::2] = x1[1::2]
```

```
def IDWTKernelFilters(H0, H1, G0, G1, x, symm, dual):
    f0, f1 = G0, G1
    if dual:
        f0, f1 = H0, H1
    N = len(x)
    x0 = x.copy(); x0[1::2] = 0
    x1 = x.copy(); x1[::2] = 0
    x0 = filterS(f0, x0, symm)
    x1 = filterS(f1, x1, symm)
    x[:] = x0 + x1
```

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

```
f    = lambda x, symm, dual: DWTKernelFilters(H0,H1,G0,G1,x,symm,dual)
invf = lambda x, symm, dual: IDWTKernelFilters(H0,H1,G0,G1,x,symm,dual)
```

6.2

1. Show that it is impossible to find a non-trivial FIR-filter which satisfies Equation (6.30).

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

6.3

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

> **a.** Use your computer to verify the connection we stated between the tables C and D, i.e. that $D_i = 32C_i$ for all i.
>
> **b.** Plot the frequency responses of the corresponding prototype filters, and verify that they both are lowpass filters. Use the connection from Theorem (6.26) to find the prototype filter coefficients from the C_q.

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

```
def mp3forwardfbt(x):
    N = len(x)
    z = mat(zeros((N,1)))
    C = mp3ctable() # The analysis window;
    x = x[(N-1)::(-1)]
    x = concatenate([x, zeros(512 - 32)])

    # The 32x64 matrix M
    yvec = arange(0, 32, 1, float)
    yvec = yvec.reshape((32, 1))
    xvec = arange(-16, 48, 1, float)
    xvec = xvec.reshape((1, 64))
    M = cos((2*yvec+1)*xvec*pi/64)

    start = len(x) - 512;
```

```
    for n in range(1, N/32 + 1):
        X = x[start:(start + 512)]
        Z = C*X # Pointwise multiplication
        Y = zeros(64)
        for j in range(8):
            Y += Z[(64*j):(64*(j+1))]
        Y = Y.reshape((64, 1))
        z[((n-1)*32):(n*32), 0] = mat(M)*mat(Y)
        start -= 32
    z = array(z).flatten()
    return z
```

```
def mp3reversefbt(z):
    N = len(z)
    z = z. reshape((N,1))
    z = mat(z)
    Ns = N/32
    x = zeros(32*Ns)
    D = mp3dtable()          # The reconstruction window.
    V = mat(zeros((1024,1)))
    # The 64x32 matrix N

    yvec = arange(16, 80, 1, float)
    yvec = yvec.reshape((64, 1))
    xvec = arange(0, 32, 1, float)
    xvec = xvec.reshape((1, 32))
    Nmatr = mat(cos(yvec*(2*xvec + 1)*pi/64))

    U = zeros(512)
    for n in range(1, Ns+1):
        V[64:1024, 0] = V[0:(1024-64), 0]
        V[0:64, 0] = Nmatr*z[((n-1)*32):(n*32), 0]

        for i in range(8):
            U[(i*64):(i*64 + 32)]       = \
            array(V[(i*128):(i*128 + 32), 0]).flatten()
            U[(i*64 + 32):((i + 1)*64)] = \
            array(V[(i*128 + 96):((i+1)*128), 0]).flatten()

        W = U*D
        for i in range(16):
            x[((n-1)*32):(n*32)] += W[32*i:(32*(i + 1))]
    return x
```

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

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

Chapter 7

7.1

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

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

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

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

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

    coordsvm = zeros((b-a)*2**nres)
    coordsvm[0] = 1
```

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

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

If you now run

```
# Plot wavelet functions for alternative piecewise linear wavelet
plotwaveletfunctions(IDWTKernelpwl2, -2, 6)
```

you will see the scaling functions and mother wavelets for the alternative piecewise linear wavelet in Figure 7.1.

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

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

c. In the code above, we turned off symmetric extensions (the `symm`-argument is 0). Attempt to use symmetric extensions instead, and observe the new plots you obtain. Can you explain why these new plots do not show the correct functions, while the previous plots are correct?

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

e. Use the function `plotwaveletfunctions` to plot all scaling functions and mother wavelets for the Haar wavelets and the piecewise linear wavelet also.

2. In Exercise 10 in Section 6.1 we constructed a new mother wavelet $\hat{\psi}$ for piecewise linear functions by finding constants $\alpha, \beta, \gamma, \delta$ so that

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

Use the cascade algorithm to plot $\hat{\psi}$. Do this by using the wavelet kernel for the piecewise linear wavelet (do not use the code above, since we have not implemented kernels for this wavelet yet).
Solution: Assuming that the vector `coeffs` has been set as in Exercise 10 in Section 6.1, the code can look as follows

```
# Exercise 7.1.2
m = 10
t = linspace(-2, 6, 8*2**m)
coordsvm = hstack([[-coeffs[0], -coeffs[1], -coeffs[3], 0, 0, 0, 0,\
                    -coeffs[2], 1, 0, 0, 0, 0, 0, 0], \
                   zeros(8*2**m-16)])
IDWTImpl(coordsvm, m, IDWTKernelpwl0, 0)
coordsvm *= 2**(m/2.)
plot(t,hstack([coordsvm[(6*2**m):(8*2**m+1)], coordsvm[0:(6*2**m)]]))
```

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

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

Similar statements hold for the IDWT as well.
Solution: Assume that the kernel transformations of the DWT and the IDWT are H and G, respectively. The formulas for the DWT and the dual DWT are obvious. For the transpose the point is that, while the kernel transformations of the DWT and the dual DWT are H and G^T, we compose the kernel with a permutation matrix when we compute the DWT. When we transpose, the order of the kernel and the permutation changes, so the transpose must use an IDWT implementation instead.
The kernel for the transpose of the DWT is H^T, which is the kernel of the dual IDWT. This explains the third line.
The kernel for the transpose of the dual DWT is $(G^T)^T = G$, which is the kernel of the IDWT. This explains the fourth line.

7.2

1. Compute the filters H_0, G_0 in Theorem 7.16 when $N = N_1 = N_2 = 4$, and $Q_1 = Q^{(4)}$, $Q_2 = 1$. Compute also filters H_1, G_1 so that we have perfect reconstruction (note that these are not unique).

Solution: We have that

$$\lambda_{H_0}(\omega) = \left(\frac{1}{2}(1+\cos\omega)\right)^{N_1/2} Q_1\left(\frac{1}{2}(1-\cos\omega)\right) = \left(\frac{1}{2}(1+\cos\omega)\right)^2 Q^{(4)}\left(\frac{1}{2}(1-\cos\omega)\right)$$

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

$$= \frac{1}{4}\left(1+\frac{1}{2}e^{i\omega}+\frac{1}{2}e^{-i\omega}\right)^2 = \frac{1}{16}(e^{2i\omega}+4e^{i\omega}+6+4e^{-i\omega}+e^{-2i\omega}).$$

Therefore $G_0 = \frac{1}{16}\{1,4,\underline{6},4,1\}$. We do not recommend to compute H_0 by hand. With the package `sympy` in Python you can do as follows to compute H_0.

```
x = Symbol('x')
z = expand( ((1+x/2+1/(2*x))/2)**2* \
        (2+8*((1-x/2-1/(2*x))/2)+20*((1-x/2-1/(2*x))/2)**2\
        +40*((1-x/2-1/(2*x))/2)**3) )
```

Here we have substituted `x` for $e^{i\omega}$, `1/x` for $e^{-i\omega}$. The first part represents $\left(\frac{1}{2}(1+\cos\omega)\right)^2$, the second part represents $Q^{(4)}(u) = 2+8u+20u^2+40u^3$ with $u = \frac{1}{2}(1-\cos\omega) = \frac{1}{2}\left(1-\frac{1}{2}e^{i\omega}-\frac{1}{2}e^{-i\omega}\right)$. This gives

$$H_0 = \frac{1}{128}\{-5,20,-1,-96,70,\underline{280},70,-96,-1,20,-5\}.$$

Using Theorem 6.16 with $\alpha = 1$, $d = 0$, we get

$$H_1 = \frac{1}{16}\{1,-4,\underline{6},-4,1\}$$

$$G_1 = \frac{1}{128}\{5,20,1,-96,-70,\underline{280},-70,-96,1,20,5\}$$

7.3

1. In this exercise we will see how we can view the frequency responses, scaling functions and mother wavelets for any spline wavelet.

a. Write a function which takes N_1 and N_2 as input, computes the filter coefficients of H_0 and G_0 using equation (7.25), and plots the frequency responses of G_0 and H_0. Recall that the frequency response can be obtained from the filter coefficients by taking a DFT. You will have use for the `conv` function here, and that the frequency response $(1+\cos\omega)/2$ corresponds to the filter with coefficients $\{1/4,\underline{1/2},1/4\}$.

b. Recall that in Exercise 12 in Section 6.1 we implemented DWT and IDWT kernels, which worked for any set of symmetric filters. Combine these kernels with your computation of the filter coefficients from a., and use the function `plotwaveletfunctions` to plot the corresponding scaling functions and mother wavelets for different N_1 and N_2.

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

7.4

1. Generate the plots from Figure 7.3 using the cascade algorithm. Reuse the code from Exercise 1 in Section 7.1 in order to achieve this.

Chapter 8

8.1

1. Let H and G be MRA-matrices for a DWT/IDWT, with corresponding filters H_0, H_1, G_0, G_1, and polyphase components $H^{(i,j)}$, $G^{(i,j)}$.

a. Show that

$$\begin{aligned}
\lambda_{H_0}(\omega) &= \lambda_{H^{(0,0)}}(2\omega) + e^{i\omega}\lambda_{H^{(0,1)}}(2\omega) \\
\lambda_{H_1}(\omega) &= \lambda_{H^{(1,1)}}(2\omega) + e^{-i\omega}\lambda_{H^{(1,0)}}(2\omega) \\
\lambda_{G_0}(\omega) &= \lambda_{G^{(0,0)}}(2\omega) + e^{-i\omega}\lambda_{G^{(1,0)}}(2\omega) \\
\lambda_{G_1}(\omega) &= \lambda_{G^{(1,1)}}(2\omega) + e^{i\omega}\lambda_{G^{(0,1)}}(2\omega).
\end{aligned}$$

Solution: $G^{(0,0)}, G^{(1,1)}$ are the even-indexed filter coefficients of G_0, G_1, respectively, so that $\lambda_{G^{(0,0)}}(2\omega), \lambda_{G^{(1,1)}}(2\omega)$ represents the half of $\lambda_{G_0}(\omega), \lambda_{G_1}(\omega)$, respectively, from the even filter coefficients. $G^{(1,0)}$ are the odd-indexed filter coefficients of G_0. Since coefficient 0 in $G^{(1,0)}$ equals coefficient 1 in G_0, it is clear that $e^{-i\omega}\lambda_{G^{(1,0)}}(2\omega)$ represents the half of $\lambda_{G_0}(\omega)$ from the odd filter coefficients. This proves the first formula. The second formula follows from the same kind of reasoning.

If we transpose H (also in polyphase form), we get an MRA-matrix where the columns are given by the filters $(H_0)^T, (H_1)^T$. Inserting these in the formulas we just proved we get that

$$\begin{aligned}
\lambda_{(H_0)^T}(\omega) &= \lambda_{(H^{(0,0)})^T}(2\omega) + e^{-i\omega}\lambda_{(H^{(0,1)})^T}(2\omega) \\
\lambda_{(H_1)^T}(\omega) &= \lambda_{(H^{(1,1)})^T}(2\omega) + e^{i\omega}\lambda_{(H^{(1,0)})^T}(2\omega).
\end{aligned}$$

If we conjugate these expressions we get

$$\begin{aligned}
\lambda_{H_0}(\omega) &= \lambda_{H^{(0,0)}}(2\omega) + e^{i\omega}\lambda_{H^{(0,1)}}(2\omega) \\
\lambda_{H_1}(\omega) &= \lambda_{H^{(1,1)}}(2\omega) + e^{-i\omega}\lambda_{H^{(1,0)}}(2\omega),
\end{aligned}$$

and the proof is done.

b. In the proof of the last part of Theorem 6.17, we defered the last part, namely that equations (8.2)-(8.3) follow from

$$\begin{pmatrix} G^{(0,0)} & G^{(0,1)} \\ G^{(1,0)} & G^{(1,1)} \end{pmatrix} = \begin{pmatrix} \alpha E_{-d} H^{(1,1)} & -\alpha E_{-d} H^{(0,1)} \\ -\alpha E_{-d} H^{(1,0)} & \alpha E_{-d} H^{(0,0)} \end{pmatrix}.$$

Prove this based on the result from a.
Solution: The first column in the matrix on the left hand side gives the filter G_0. On the right hand side, a. states that the even-indexed columns are taken from the filter with frequency response

$$\begin{aligned} &\lambda_{\alpha E_{-d} H^{(1,1)}}(2\omega) + e^{-i\omega} \lambda_{-\alpha E_{-d} H^{(1,0)}}(2\omega) \\ &= \alpha \lambda_{E_{-d}}(2\omega) \left(\lambda_{H^{(1,1)}}(2\omega) - e^{-i\omega} \lambda_{H^{(1,0)}}(2\omega) \right) \\ &= \alpha e^{2id\omega} \left(\lambda_{H^{(1,1)}}(2(\omega+\pi)) + e^{-i(\omega+\pi)} \lambda_{H^{(1,0)}}(2(\omega+\pi)) \right) = \alpha e^{2id\omega} \lambda_{H_1}(\omega+\pi). \end{aligned}$$

This shows that $\lambda_{G_0}(\omega) = \alpha e^{2id\omega} \lambda_{H_1}(\omega+\pi)$. We obtain Equation (8.2) easily from this. Now, the second column in the matrix on the left hand side gives the filter coefficients of G_1. On the right hand side, a. states that the odd-indexed columns are taken from the filter with frequency response

$$\begin{aligned} &\lambda_{\alpha E_{-d} H^{(0,0)}}(2\omega) + e^{i\omega} \lambda_{-\alpha E_{-d} H^{(0,1)}}(2\omega) \\ &= \alpha \lambda_{E_{-d}}(2\omega) \left(\lambda_{H^{(0,0)}}(2\omega) - e^{i\omega} \lambda_{H^{(0,1)}}(2\omega) \right) \\ &= \alpha e^{2id\omega} \left(\lambda_{H^{(0,0)}}(2(\omega+\pi)) + e^{i(\omega+\pi)} \lambda_{H^{(0,1)}}(2(\omega+\pi)) \right) = \alpha e^{2id\omega} \lambda_{H_0}(\omega+\pi). \end{aligned}$$

This shows that $\lambda_{G_1}(\omega) = \alpha e^{2id\omega} \lambda_{H_0}(\omega+\pi)$, which is Equation (8.3).

2. Let S be a filter. Show that

a. $G \begin{pmatrix} I & \mathbf{0} \\ S & I \end{pmatrix}$ is an MRA matrix with cfilters $\tilde{G_0}, G_1$, where

$$\lambda_{\tilde{G_0}}(\omega) = \lambda_{G_0}(\omega) + \lambda_S(2\omega) e^{-i\omega} \lambda_{G_1}(\omega),$$

Solution: We have that

$$\begin{pmatrix} G^{(0,0)} & G^{(0,1)} \\ G^{(1,0)} & G^{(1,1)} \end{pmatrix} \begin{pmatrix} I & \mathbf{0} \\ S & I \end{pmatrix} = \begin{pmatrix} G^{(0,0)} + SG^{(0,1)} & G^{(0,1)} \\ G^{(1,0)} + SG^{(1,1)} & G^{(1,1)} \end{pmatrix}.$$

Using Exercise 1a., the even-indexed columns in this matrix are taken from the filter with frequency response

$$\begin{aligned} &\lambda_{G^{(0,0)} + SG^{(0,1)}}(2\omega) + e^{-i\omega} \lambda_{G^{(1,0)} + SG^{(1,1)}}(2\omega) \\ &= \lambda_{G^{(0,0)}}(2\omega) + e^{-i\omega} \lambda_{G^{(1,0)}}(2\omega) + \lambda_S(2\omega) \left(\lambda_{G^{(0,1)}}(2\omega) + e^{-i\omega} \lambda_{G^{(1,1)}}(2\omega) \right) \\ &= \lambda_{G_0}(\omega) + \lambda_S(2\omega) e^{-i\omega} \left(\lambda_{G^{(1,1)}}(2\omega) + e^{i\omega} \lambda_{G^{(0,1)}}(2\omega) \right) \\ &= \lambda_{G_0}(\omega) + \lambda_S(2\omega) e^{-i\omega} \lambda_{G_1}(\omega). \end{aligned}$$

b. $G\begin{pmatrix} I & S \\ \mathbf{0} & I \end{pmatrix}$ is an MRA matrix with filters $G_0, \tilde{G}_1$, where

$$\lambda_{\tilde{G}_1}(\omega) = \lambda_{G_1}(\omega) + \lambda_S(2\omega)e^{i\omega}\lambda_{G_0}(\omega),$$

Solution: We have that

$$\begin{pmatrix} G^{(0,0)} & G^{(0,1)} \\ G^{(1,0)} & G^{(1,1)} \end{pmatrix}\begin{pmatrix} I & S \\ \mathbf{0} & I \end{pmatrix} = \begin{pmatrix} G^{(0,0)} & SG^{(0,0)} + G^{(0,1)} \\ G^{(1,0)} & SG^{(1,0)} + G^{(1,1)} \end{pmatrix},$$

so that the odd-indexed columns in this matrix are taken from the filter with frequency response

$$\begin{aligned}
&\lambda_{SG^{(1,0)}+G^{(1,1)}}(2\omega) + e^{i\omega}\lambda_{SG^{(0,0)}+G^{(0,1)}}(2\omega) \\
&= \lambda_{G^{(1,1)}}(2\omega) + e^{i\omega}\lambda_{G^{(0,1)}}(2\omega) + \lambda_S(2\omega)\left(\lambda_{G^{(1,0)}}(2\omega) + e^{i\omega}\lambda_{G^{(0,0)}}(2\omega)\right) \\
&= \lambda_{G_1}(\omega) + \lambda_S(2\omega)e^{i\omega}\left(\lambda_{G^{(0,0)}}(2\omega) + e^{-i\omega}\lambda_{G^{(1,0)}}(2\omega)\right) \\
&= \lambda_{G_1}(\omega) + \lambda_S(2\omega)e^{i\omega}\lambda_{G_0}(\omega).
\end{aligned}$$

c. $\begin{pmatrix} I & \mathbf{0} \\ S & I \end{pmatrix} H$ is an MRA-matrix with filters $H_0, \tilde{H}_1$, where

$$\lambda_{\tilde{H}_1}(\omega) = \lambda_{H_1}(\omega) + \lambda_S(2\omega)e^{-i\omega}\lambda_{H_0}(\omega).$$

Solution: We transpose the expression to obtain $H^T\begin{pmatrix} I & S^T \\ \mathbf{0} & I \end{pmatrix}$ Since H^T has filters $(H_0)^T$ and $(H_1)^T$ in the columns, from b. it follows that $H^T\begin{pmatrix} I & S^T \\ \mathbf{0} & I \end{pmatrix}$ has columns given by $(H_0)^T$ and the filter with frequency response

$$\lambda_{(H_1)^T}(\omega) + \lambda_{S^T}(2\omega)e^{i\omega}\lambda_{(H_0)^T}(\omega) = \overline{\lambda_{H_1}(\omega) + \lambda_S(2\omega)e^{-i\omega}\lambda_{H_0}(\omega)},$$

so that $\begin{pmatrix} I & \mathbf{0} \\ S & I \end{pmatrix} H$ has row filters H_0 and a filter $\tilde{H}_1$ with frequency response

$$\lambda_{\tilde{H}_1}(\omega) = \lambda_{H_1}(\omega) + \lambda_S(2\omega)e^{-i\omega}\lambda_{H_0}(\omega).$$

d. $\begin{pmatrix} I & S \\ \mathbf{0} & I \end{pmatrix} H$ is an MRA-matrix with filters $\tilde{H}_0, H_1$, where

$$\lambda_{\tilde{H}_0}(\omega) = \lambda_{H_0}(\omega) + \lambda_S(2\omega)e^{i\omega}\lambda_{H_1}(\omega).$$

Solution: We transpose the expression to obtain $H^T\begin{pmatrix} I & \mathbf{0} \\ S^T & I \end{pmatrix}$. Since H^T has

filters $(H_0)^T$ and $(H_1)^T$ in the columns, using a. we see that $H^T \begin{pmatrix} I & \mathbf{0} \\ S^T & I \end{pmatrix}$ has columns given by the filter with frequency response

$$\lambda_{(H_0)^T}(\omega) + \lambda_{S^T}(2\omega)e^{-i\omega}\lambda_{(H_1)^T}(\omega) = \overline{\lambda_{H_0}(\omega) + \lambda_S(2\omega)e^{i\omega}\lambda_{H_1}(\omega)},$$

and $(H_1)^T$, so that $\begin{pmatrix} I & S \\ \mathbf{0} & I \end{pmatrix} H$ has a row filter $\tilde{H}_0$ with frequency response

$$\lambda_{\tilde{H}_0}(\omega) = \lambda_{H_0}(\omega) + \lambda_S(2\omega)e^{i\omega}\lambda_{H_1}(\omega),$$

and H_1.

In summary, this exercise shows that one can think of the steps in the lifting factorization as altering one of the filters of an MRA-matrix in alternating order.

3. Show that S is a filter of length kM if and only if the entries $\{S^{i,j}\}_{i,j=0}^{M-1}$ in the polyphase representation of S satisfy $S^{(i+r) \bmod M,(j+r) \bmod M} = S_{i,j}$. In other words, S is a filter if and only if the polyphase representation of S is a "block-circulant Toeplitz matrix". This implies a fact that we will use: GH is a filter (and thus provides alias cancellation) if blocks in the polyphase representations repeat cyclically as in a Toeplitz matrix (in particular when the matrix is block-diagonal with the same block repeating on the diagonal).

Solution: If S is a filter we have that $S^{(i+r+s_1M) \bmod kM,(j+r+s_2M)kM} = S^{(i+s_1M) \bmod M,(j+s_2M) \bmod M}$, $0 \le i, j < M$. But since $S^{(i+s_1M) \bmod kM,(j+s_2M)kM} = S^{(i,j)}_{s_1,s_2}$, it follows that $S^{((i+r) \bmod M,(j+r) \bmod M)}_{s_1,s_2} = S^{(i,j)}_{s_1,s_2}$, so that $S^{(i+r) \bmod M,(j+r) \bmod M} = S_{i,j}$.

4. Recall from Definition 6.20 that we defined a classical QMF filter bank as one where $M = 2$, $G_0 = H_0$, $G_1 = H_1$, and $\lambda_{H_1}(\omega) = \lambda_{H_0}(\omega + \pi)$. Show that the forward and reverse filter bank transforms of a classical QMF filter bank take the form

$$H = G = \begin{pmatrix} A & -B \\ B & A \end{pmatrix}$$

5. Recall from Definition 6.21 that we defined an alternative QMF filter bank as one where $M = 2$, $G_0 = (H_0)^T$, $G_1 = (H_1)^T$, and $\lambda_{H_1}(\omega) = \overline{\lambda_{H_0}(\omega + \pi)}$. Show that the forward and reverse filter bank transforms of an alternative QMF filter bank take the form.

$$H = \begin{pmatrix} A^T & B^T \\ -B & A \end{pmatrix} \qquad G = \begin{pmatrix} A & -B^T \\ B & A^T \end{pmatrix} = \begin{pmatrix} A^T & B^T \\ -B & A \end{pmatrix}^T.$$

6. Consider alternative QMF filter banks where we take in an additional sign, so that $\lambda_{H_1}(\omega) = -\overline{\lambda_{H_0}(\omega + \pi)}$ (the Haar wavelet was an example of such a filter bank). Show that the forward and reverse filter bank transforms now take the form

$$H = \begin{pmatrix} A^T & B^T \\ B & -A \end{pmatrix} \qquad G = \begin{pmatrix} A & B^T \\ B & -A^T \end{pmatrix} = \begin{pmatrix} A^T & B^T \\ B & -A \end{pmatrix}^T.$$

It is straightforward to check that also these satisfy the alias cancellation condition, and that the perfect reconstruction condition also here takes the form $|\lambda_{H_0}(\omega)|^2 + |\lambda_{H_0}(\omega + \pi)|^2 = 2$.

8.2

1. Assume that the filters H_0, H_1 of a wavelet are symmetric, and denote by $S^{(i,j)}$ the polyphase components of the corresponding MRA-matrix H. Show that $S^{(0,0)}$ and $S^{(1,1)}$ are symmetric filters, that the filter coefficients of $S^{(1,0)}$ has symmetry about $-1/2$, and that $S^{(0,1)}$ has symmetry about $1/2$. Also show a similar statement for the MRA-matrix G of the inverse DWT.

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

```
def liftingstepevensymm(lmbda, x, symm):
    """
    Apply an elementary symmetric lifting step of even type to x.

    lmbda: The common value of the two filter coefficients
    x: The vector which we apply the lifting step to
    symm: Whether to apply symmetric extension to the input
    """
    if (not symm) and mod(len(x), 2)!=0:
        raise AssertionError()
    if symm:
        x[0] += 2*lmbda*x[1] # With symmetric extension
    else:
        x[0] += lmbda*(x[1]+x[-1])
    x[2:-1:2] += lmbda*(x[1:-2:2] + x[3::2])
    if mod(len(x), 2)==1 and symm:
        x[-1] += 2*lmbda*x[-2] # With symmetric extension
```

```
def liftingstepoddsymm(lmbda, x, symm):
    """
    Apply an elementary symmetric lifting step of odd type to x.

    lmbda: The common value of the two filter coefficients
    x: The vector which we apply the lifting step to
    symm: Whether to apply symmetric extension to the input
    """
    if (not symm) and mod(len(x), 2)!=0:
        raise AssertionError()
    x[1:-1:2] += lmbda*(x[0:-2:2] + x[2::2])
    if mod(len(x), 2)==0:
        if symm:
            x[-1] += 2*lmbda*x[-2] # With symmetric extension
        else:
            x[-1] += lmbda*(x[0]+x[-2])
```

3. Up to now in this chapter we have obtained lifting factorizations for four different wavelets where the filters are symmetric. Let us now implement the kernel transformations for these wavelets. Your functions should call the functions from Exercise 2 in order to compute the individual lifting steps. Recall that the kernel transformations should take the input vector `x`, `symm` (i.e. whether symmetric extension should be applied), and `dual` (i.e. whether the dual wavelet transform should be applied) as input. You will need equations (8.9)-(8.12) here, in order to complete the kernels for bot the transformations and the dual transformations.

a. Write the DWT and IDWT kernel transformations for the piecewise linear wavelet. Your functions should use the lifting factorizations in (8.16). Call your functions `DWTKernelpwl0` and `IDWTKernelpwl0`.
Solution: The code can look like this:

```
def DWTKernelpwl0(x, symm, dual):
    """
    Apply the DWT kernel transformation for the
    piecewise linear wavelet (i.e. 0 vanishing moments) to x.

    x: The vector which we apply this kernel transformation to
    symm: Whether to apply symmetric extension to the input
    dual: Whether to apply the wavelet kernel or dual wavelet kernel.
    """
    if dual:
        x /= sqrt(2)
        x = liftingstepevensymm(0.5, x, symm)
    else:
        x *= sqrt(2)
        liftingstepoddsymm(-0.5, x, symm)
```

```
def IDWTKernelpwl0(x, symm, dual):
    """
    Apply the IDWT kernel transformation for the
    pieccewise linear wavelet (i.e. 0 vanishing moments) to x.

    x: The vector which we apply this kernel transformation to
    symm: Whether to apply symmetric extension to the input
    dual: Whether to apply the wavelet kernel or dual wavelet kernel.
    """
    if dual:
        x *= sqrt(2)
        liftingstepevensymm(-0.5, x, symm)
    else:
        x /= sqrt(2)
        liftingstepoddsymm(0.5, x, symm)
```

b. Write the DWT and IDWT kernel transformations for the alternative piecewise linear wavelet. The lifting factorizations are now given by (8.17) instead. Call your functions `DWTKernelpwl2` and `IDWTKernelpwl2`.
Solution: The code can look like this:

```
def DWTKernelpwl2(x, symm, dual):
    """
    Apply the DWT kernel transformation for the
    alternative pieccewise linear wavelet (i.e. 2 van. moms.) to x.

    x: The vector which we apply this kernel transformation to
    symm: Whether to apply symmetric extension to the input
    dual: Whether to apply the wavelet kernel or dual wavelet kernel.
    """
    if dual:
        liftingstepevensymm(0.5, x, symm)
        liftingstepoddsymm(-0.25, x, symm)
        x /= sqrt(2)
    else:
        liftingstepoddsymm(-0.5, x, symm)
        liftingstepevensymm(0.25, x, symm)
        x *= sqrt(2)
```

```
def IDWTKernelpwl2(x, symm, dual):
    """
    Apply the IDWT kernel transformation for the
    alternative pieccewise linear wavelet (i.e. 2 van. moms.) to x.

    x: The vector which we apply this kernel transformation to
    symm: Whether to apply symmetric extension to the input
    dual: Whether to apply the wavelet kernel or dual wavelet kernel.
    """
    if dual:
        x *= sqrt(2)
        liftingstepoddsymm(0.25, x, symm)
        liftingstepevensymm(-0.5, x, symm)
    else:
        x /= sqrt(2)
        liftingstepevensymm(-0.25, x, symm)
        liftingstepoddsymm(0.5, x, symm)
```

c. Write the DWT and IDWT kernel transformations for the Spline 5/3 wavelet, using the lifting factorization obtained in Example 8.16. Call your functions `DWTKernel53` and `IDWTKernel53`.
Solution: The code can look like this:

```
def DWTKernel53(x, symm, dual):
    """
    Apply the DWT kernel transformation for the
    Spline 5/3 wavelet to x.

    x: The vector which we apply this kernel transformation to
    symm: Whether to apply symmetric extension to the input
    dual: Whether to apply the wavelet kernel or dual wavelet kernel.
    """
    if dual:
        x[0::2] *= 0.5
        x[1::2] *= 2
        liftingstepevensymm(0.125, x, symm)
```

```
            liftingstepoddsymm(-1, x, symm)
        else:
            x[0::2] *= 2
            x[1::2] *= 0.5
            liftingstepoddsymm(-0.125, x, symm)
            liftingstepevensymm(1, x, symm)
```

```
    def IDWTKernel53(x, symm, dual):
        """
        Apply the IDWT kernel transformation for the
        Spline 5/3 wavelet to x.

        x: The vector which we apply this kernel transformation to
        symm: Whether to apply symmetric extension to the input
        dual: Whether to apply the wavelet kernel or dual wavelet kernel.
        """
        if dual:
            liftingstepoddsymm(1, x, symm)
            liftingstepevensymm(-0.125, x, symm)
            x[0::2] *= 2
            x[1::2] *= 0.5
        else:
            liftingstepevensymm(-1, x, symm)
            liftingstepoddsymm(0.125, x, symm)
            x[0::2] *= 0.5
            x[1::2] *= 2
```

d. Write the DWT and IDWT kernel transformations for the CDF 9/7 wavelet, using the lifting factorization obtained in Example 8.18. Call your functions `DWTKernel97` and `IDWTKernel97`.
Solution: The code can look like this:

```
    def DWTKernel97(x, symm, dual):
        """
        Apply the DWT kernel transformation for the CDF 9/7 wavelet to x.

        x: The vector which we apply this kernel transformation to
        symm: Whether to apply symmetric extension to the input
        dual: Whether to apply the wavelet kernel or dual wavelet kernel.
        """
        lambda1=-0.586134342059950
        lambda2=-0.668067171029734
        lambda3=0.070018009414994
        lambda4=1.200171016244178
        alpha=-1.149604398860250
        beta=-0.869864451624777
        if dual:
            x[0::2] /= alpha
            x[1::2] /= beta
            liftingstepevensymm(lambda4, x, symm)
            liftingstepoddsymm(lambda3, x, symm)
            liftingstepevensymm(lambda2, x, symm)
            liftingstepoddsymm(lambda1, x, symm)
        else:
```

```
            x[0::2] *= alpha
            x[1::2] *= beta
            liftingstepoddsymm(-lambda4, x, symm)
            liftingstepevensymm(-lambda3, x, symm)
            liftingstepoddsymm(-lambda2, x, symm)
            liftingstepevensymm(-lambda1, x, symm)
```

```
def IDWTKernel97(x, symm, dual):
    """
    Apply the IDWT kernel transformation for the CDF 9/7 wavelet to x

    x: The vector which we apply this kernel transformation to
    symm: Whether to apply symmetric extension to the input
    dual: Whether to apply the wavelet kernel or dual wavelet kernel.
    """
    lambda1=-0.586134342059950
    lambda2=-0.668067171029734
    lambda3=0.070018009414994
    lambda4=1.200171016244178
    alpha=-1.149604398860250
    beta=-0.869864451624777
    if dual:
        liftingstepoddsymm(-lambda1, x, symm)
        liftingstepevensymm(-lambda2, x, symm)
        liftingstepoddsymm(-lambda3, x, symm)
        liftingstepevensymm(-lambda4, x, symm)
        x[0::2] *= alpha
        x[1::2] *= beta
    else:
        liftingstepevensymm(lambda1, x, symm)
        liftingstepoddsymm(lambda2, x, symm)
        liftingstepevensymm(lambda3, x, symm)
        liftingstepoddsymm(lambda4, x, symm)
        x[0::2] /= alpha
        x[1::2] /= beta
```

e. In Chapter 5, we listened to the low-resolution approximations and detail components in sound for three different wavelets, using the function `playDWT`. Repeat these experiments with the Spline 5/3 and the CDF 9/7 wavelet, using the new kernels we have implemented in this exercise.
Solution: The following code can be used for listening to the low-resolution approximations for a given value of m.

```
# Play lowres approx for the Spline 5/3 wavelet
playDWT(m, DWTKernel53, IDWTKernel53, True)
```

```
# Play lowres approx for the CDF 9/7 wavelet
playDWT(m, DWTKernel97, IDWTKernel97, True)
```

f. Use the function `plotwaveletfunctions` from Exercise 1 in Section 7.1 to plot all scaling functions and mother wavelets for the Spline 5/3 and the CDF 9/7 wavelets, using the kernels you have implemented.
Solution: The code can look as follows.

```
# Plot wavelet functions for the Spline 5/3 wavelet
plotwaveletfunctions(IDWTKernel53, -4, 4)
```

```
# Plot wavelet functions for the CDF 9/7 wavelet
plotwaveletfunctions(IDWTKernel97, -4, 4)
```

In the plot for the CDF 9/7 wavelet, it is seen that the functions and their dual counterparts are close to being equal. This reflects the fact that this wavelet is close to being orthogonal.

4. In this exercise we will implement the kernel transformations for orthonormal wavelets.

a. Write functions `liftingstepeven` and `liftingstepodd` which take λ_1, λ_2 and a vector $\boldsymbol{x}$ as input, and apply the elementary lifting matrices (8.18), respectively, to $\boldsymbol{x}$. Assume that N is even.
Solution: The code can look like this:

```
def liftingstepeven(lmbda1, lmbda2, x):
    """
    Apply an elementary non-symmetric lifting step of even type to x.

    lmbda1: The first filter coefficient
    lmbda2: The second filter coefficient
    x: The vector which we apply the lifting step to
    """
    if mod(len(x), 2)!=0:
        raise AssertionError()
    x[0] += lmbda1*x[1] + lmbda2*x[-1]
    x[2:-1:2] += lmbda1*x[3::2] + lmbda2*x[1:-2:2]
```

```
def liftingstepodd(lmbda1, lmbda2, x):
    """
    Apply an elementary non-symmetric lifting step of odd type to x.

    lmbda1: The first filter coefficient
    lmbda2: The second filter coefficient
    x: The vector which we apply the lifting step to
    """
    if mod(len(x), 2)!=0:
        raise AssertionError()
    x[1:-2:2] += lmbda1*x[2:-1:2] + lmbda2*x[0:-3:2]
    x[-1] += lmbda1*x[0] + lmbda2*x[-2]
```

b. Write functions `DWTKernelOrtho` and `IDWTKernelOrtho` which take a vector $\boldsymbol{x}$ as input, and apply the DWT and IDWT kernel transformations for orthonormal wavelets to $\boldsymbol{x}$. You should call the functions `liftingstepeven` and `liftingstepodd`. As mentioned, assume that global variables `lambdas`, `alpha`, and `beta` have been set, so that the lifting factorization (8.8) holds, where `lambdas` is a $n\times 2$-matrix so that the filter coefficients $\{\underline{\lambda_1}, \lambda_2\}$ or $\{\lambda_1, \underline{\lambda_2}\}$ in the i'th lifting step is found in row i of `lambdas`. Recall that the last lifting step was even.

Solution: The code can look like this:

```
def DWTKernelOrtho( x, symm, dual):
    """
    Apply the DWT kernel transformation for orthonormal wavelets.
    The number of vanishing moments is stored in global variables.

    x: The vector which we apply this kernel transformation to
    symm: Whether to apply symmetric extension to the input
    dual: Whether to apply the wavelet kernel or dual wavelet kernel.
    """
    global lambdas, alpha, beta
    global beta
    if dual:
        x[0::2] /= alpha
        x[1::2] /= beta
        for stepnr in range(lambdas.shape[0] - 1, 0, -2):
            liftingstepodd(lambdas[stepnr, 1], lambdas[stepnr, 0], x)
            liftingstepeven(lambdas[stepnr -1, 1], lambdas[stepnr - 1, 0], x)
        if mod(lambdas.shape[0], 2)==1:
            liftingstepodd(lambdas[0, 1], lambdas[0, 0], x)
    else:
        x[0::2] *= alpha
        x[1::2] *= beta
        for stepnr in range(lambdas.shape[0] - 1, 0, -2):
            liftingstepeven(-lambdas[stepnr, 0], -lambdas[stepnr, 1], x)
            liftingstepodd(-lambdas[stepnr - 1, 0], -lambdas[stepnr - 1, 1], x)
        if mod(lambdas.shape[0], 2)==1:
            liftingstepeven(-lambdas[0, 0], -lambdas[0, 1], x)
```

```
def IDWTKernelOrtho( x, symm, dual):
    """
    Apply the IDWT kernel transformation for orthonormal wavelets.
    The number of vanishing moments is stored in global variables.

    x: The vector which we apply this kernel transformation to
    symm: Whether to apply symmetric extension to the input
    dual: Whether to apply the wavelet kernel or dual wavelet kernel.
    """
    global lambdas
    global alpha
    global beta
    if dual:
        stepnr = 0
        if mod(lambdas.shape[0], 2) == 1: # Start with an odd step
            liftingstepodd(-lambdas[stepnr, 1], -lambdas[stepnr, 0], x)
```

```
            stepnr += 1
        while stepnr < lambdas.shape[0]:
            liftingstepeven(-lambdas[stepnr, 1], -lambdas[stepnr, 0], x)
            liftingstepodd(-lambdas[stepnr + 1, 1], -lambdas[stepnr + 1, 0], x)
            stepnr += 2
        x[0::2] *= alpha
        x[1::2] *= beta
    else:
        stepnr = 0
        if mod(lambdas.shape[0],2) == 1: # Start with an even step
            liftingstepeven(lambdas[stepnr, 0], lambdas[stepnr, 1], x)
            stepnr += 1
        while stepnr < lambdas.shape[0]:
            liftingstepodd(lambdas[stepnr, 0], lambdas[stepnr, 1], x)
            liftingstepeven(lambdas[stepnr + 1, 0], lambdas[stepnr + 1, 1], x)
            stepnr += 2
        x[0::2] /= alpha
        x[1::2] /=beta
```

c. Listen to the low-resolution approximations and detail components in sound for orthonormal wavelets for $N = 1,2,3,4$, again using the function `playDWT`. You need to call the function `liftingfactortho` in order to set the kernel for the different values of N.
Solution: The following code can be used for listening to the low-resolution approximations for a given value of m.

```
# Play lowres approx for orthonormal wavelets, filters of length 4
liftingfactortho(2)
playDWT(m, DWTKernelOrtho, IDWTKernelOrtho, True)
```

```
# Play lowres approx for orthonormal wavelets, filters of length 6
liftingfactortho(3)
playDWT(m, DWTKernelOrtho, IDWTKernelOrtho, True)
```

```
# Play lowres approx for orthonormal wavelets, filters of length 8
liftingfactortho(4)
playDWT(m, DWTKernelOrtho, IDWTKernelOrtho, True)
```

d. Use the function `plotwaveletfunctions` from Exercise 1 in Section 7.1 to plot all scaling functions and mother wavelets for orthonormal wavelets for $N = 1,2,3,4$. Since the wavelets are orthonormal, we should have that $\phi = \tilde{\phi}$, and $\psi = \tilde{\psi}$. In other words, you should see that the bottom plots equal the upper plots.
Solution: The code can look as follows.

```
    # Plot wavelet functions for orthonormal wavelets, filters length 4
    liftingfactortho(2)
    plotwaveletfunctions(IDWTKernelOrtho, -4, 4)
```

```
    # Plot wavelet functions for orthonormal wavelets, filters length 6
    liftingfactortho(3)
    plotwaveletfunctions(IDWTKernelOrtho, -4, 4)
```

```
    # Plot wavelet functions for orthonormal wavelets, filters length 8
    liftingfactortho(4)
    plotwaveletfunctions(IDWTKernelOrtho, -4, 4)
```

5. Symmetric lifting steps in this chapter have all been on the form $\begin{pmatrix} I & \lambda\{\underline{1},1\} \\ 0 & I \end{pmatrix}$ and $\begin{pmatrix} I & 0 \\ \lambda\{1,\underline{1}\} & I \end{pmatrix}$. In the next exercises, we will see that we also need to consider lifting steps of the form

$$\begin{pmatrix} I & \{\lambda_2,\underline{\lambda_1},\lambda_1,\lambda_2\} \\ 0 & I \end{pmatrix} \text{ and } \begin{pmatrix} I & 0 \\ \{\lambda_2,\lambda_1,\underline{\lambda_1},\lambda_2\} & I \end{pmatrix}. \tag{8.1}$$

Write functions `liftingstepeven2symm` and `liftingstepodd2symm` which take λ_1,λ_2, a vector $\boldsymbol{x}$ and `symm` as input, and apply the lifting matrices (8.1), respectively, to $\boldsymbol{x}$. The functions should perform as few multiplications as possible. Concentrate first on the case where symmetric extensions are made (`symm=1`), and then on the case when they are not made (`symm=0`).

Solution: What complicates this is that there may be two rows which circulate at the beginning and end of the matrix. The code can look as follows.

```
def liftingstepeven2symm( lambda1, lambda2, x, symm):
    """
    Apply a symmetric lifting step with five filter coefficients
    of even type to x.

    lambda1: The common value of the inner filter coefficients
    lambda2: The common value of the outer filter coefficients
    x: The vector which we apply the lifting step to
    symm: Whether to apply symmetric extension to the input
    """
    if (not symm) and mod(len(x), 2)!=0:
        raise AssertionError()
    if symm:
        x[0] += lambda1*2*x[1]
        x[0] += lambda2*2*x[3]
        x[2] += lambda1*(x[3] + x[1])
        x[2] += lambda2*(x[5] + x[3])
        if mod(len(x), 2)==1:
```

```
            x[-1] += lambda1*2*x[-2]
            x[-1] += lambda2*2*x[-4]
            x[-3] += lambda1*(x[-4] + x[-2])
            x[-3] += lambda2*(x[-6] + x[-2])
        else:
            x[-2] += lambda1*(x[-3] + x[-1])
            x[-2] += lambda2*(x[-5] + x[-3])
        x[4:-3:2] += lambda1*(x[5:-2:2] + x[3:-4:2])
        x[4:-3:2] += lambda2*(x[1:-6:2] + x[7::2])
    else:
        x[0]      += lambda1*(x[1] + x[-1])
        x[0]      += lambda2*(x[3] + x[-3])
        x[2]      += lambda1*(x[3] + x[1])
        x[2]      += lambda2*(x[5] + x[-1])
        x[-2]     += lambda1*(x[-3] + x[-1])
        x[-2]     += lambda2*(x[-5] + x[1])
        x[4:-3:2] += lambda1*(x[5:-2:2] + x[3:-4:2])
        x[4:-3:2] += lambda2*(x[7::2] + x[1:-6:2])
```

```
def liftingstepodd2symm( lambda1, lambda2, x, symm):
    """
    Apply a symmetric lifting step with five filter coefficients
    of odd type to x.

    lambda1: The common value of the inner filter coefficients
    lambda2: The common value of the outer filter coefficients
    symm: Whether to apply symmetric extension to the input
    """
    if (not symm) and mod(len(x), 2)!=0:
        raise AssertionError()
    if symm:
        x[1] += lambda1*(x[0] + x[2])
        x[1] += lambda2*(x[4] + x[2])
        if mod(len(x) ,2) == 1:
            x[-2] += lambda1*(x[-3] + x[-1])
            x[-2] += lambda2*(x[-5] + x[-3])
        else:
            x[-1] += lambda1*2*x[-2]
            x[-1] += lambda2*2*x[-4]
            x[-3] += lambda1*(x[-4] + x[-2])
            x[-3] += lambda2*(x[-6] + x[-2])
        x[3:-3:2] += lambda1*(x[4:-2:2] + x[2:-4:2])
        x[3:-3:2] += lambda2*(x[6::2] + x[0:-6:2])
    else:
        x[1]      += lambda1*(x[0] + x[2])
        x[1]      += lambda2*(x[-2] + x[4])
        x[-3]     += lambda1*(x[-4] + x[-2])
        x[-3]     += lambda2*(x[-6] + x[0])
        x[-1]     += lambda1*(x[-2] + x[0])
        x[-1]     += lambda2*(x[-4] + x[2])
        x[3:-4:2] += lambda1*(x[4:-3:2] + x[2:-5:2])
        x[3:-4:2] += lambda2*(x[6:-1:2] + x[0:-7:2])
```

6. In Exercise 2 in Section 5.5 we found constants $\alpha, \beta, \gamma, \delta$ which give the coordinates of $\hat{\psi}$ in $(\boldsymbol{\phi}_1, \hat{\boldsymbol{\psi}}_1)$, where $\hat{\psi}$ had four vanishing moments, and where we worked with the multiresolution analysis of piecewise constant functions.

a. Show that the polyphase representation of G when $\hat{\psi}$ is used as mother wavelet can be factored as

$$\frac{1}{\sqrt{2}}\begin{pmatrix} I & \mathbf{0} \\ \{1/2, \underline{1/2}\} & I \end{pmatrix}\begin{pmatrix} I & \{-\gamma, \underline{-\alpha}, -\beta, -\delta\} \\ \mathbf{0} & I \end{pmatrix}. \tag{8.2}$$

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

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

By inversion, find also a lifting factorization of H.
Solution: We have found constants $\alpha, \beta\gamma\delta$ so that

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

From this it is clear that

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

where $S_2 = \{-\gamma, \underline{-\alpha}, -\beta, -\delta\}$ This gives as before the lifting factorization

$$P_{\mathscr{D}_1 \leftarrow (\boldsymbol{\phi}_1, \hat{\boldsymbol{\psi}}_1)} = \frac{1}{\sqrt{2}}\begin{pmatrix} I & \mathbf{0} \\ S_1 & I \end{pmatrix}\begin{pmatrix} I & \{-\gamma, \underline{-\alpha}, -\beta, -\delta\} \\ \mathbf{0} & I \end{pmatrix}. \tag{8.3}$$

where $S_1 = \{1/2, \underline{1/2}\}$ as before.

b. Implement kernels `DWTKernelpwl4` and `IDWTKernelpwl4` for the DWT and IDWT of this wavelet.
Solution: Note that the matrix

$$\begin{pmatrix} I & S_2 \\ \mathbf{0} & I \end{pmatrix} = \begin{pmatrix} I & \{-\gamma, \underline{-\alpha}, -\beta, -\delta\} \\ \mathbf{0} & I \end{pmatrix} = \begin{pmatrix} I & \{-\gamma, \underline{-\alpha}, -\alpha, -\gamma\} \\ \mathbf{0} & I \end{pmatrix}$$

is an even lifting step on the form given in the previous exercise (we here used that we computed that $\alpha = \beta, \gamma = \delta$). We can therefore use the function `liftingstepeven2symm` from the previous exercise. The values $\alpha, \beta, \gamma, \delta$ can be computed with the code from Exercise 2 in Section 5.5, and the computed values could be purt into the kernels. The kernels `DWTKernelpwl2`, `IDWTKernelpwl2` can therefore easily be changed as follows, in order to implement the new kernels.

```
def DWTKernelpwl4(x, symm, dual):
    """
    Apply the DWT kernel transformation for the
    pieccewise linear wavelet with 4 vanishing moments to x.

    x: The vector which we apply this kernel transformation to
    symm: Whether to apply symmetric extension to the input
    dual: Whether to apply the wavelet kernel or dual wavelet kernel.
```

```
        """
        alpha=0.296875000000000
        gamma=-0.046875000000000
        if dual:
            x /= sqrt(2)
            liftingstepevensymm(0.5, x, symm)
            liftingstepodd2symm(-alpha, -gamma, x, symm)
        else:
            x *= sqrt(2)
            liftingstepoddsymm(-0.5, x, symm)
            liftingstepeven2symm(alpha, gamma, x, symm)
```

```
    def IDWTKernelpwl4(x, symm, dual):
        """
        Apply the IDWT kernel transformation for the
        pieccewise linear wavelet with 4 vanishing moments to x.

        x: The vector which we apply this kernel transformation to
        symm: Whether to apply symmetric extension to the input
        dual: Whether to apply the wavelet kernel or dual wavelet kernel.
        """
        alpha=0.296875000000000
        gamma=-0.046875000000000
        if dual:
            x *= sqrt(2)
            liftingstepodd2symm(alpha, gamma, x, symm)
            liftingstepevensymm(-0.5, x, symm)
        else:
            x /= sqrt(2)
            liftingstepeven2symm(-alpha, -gamma, x, symm)
            liftingstepoddsymm(0.5, x, symm)
```

c. Listen to the low-resolution approximations and detail components in sound for this wavelet.
Solution: The following code can be used for listening to the low-resolution approximations for a given value of m.

```
# Play lowres approx for piecewise linear wavelet, 4 van. moms.
playDWT(m, DWTKernelpwl4, IDWTKernelpwl4, True)
```

d. Use the function `plotwaveletfunctions` from Exercise 1 in Section 7.1 to plot all scaling functions and mother wavelets for this wavelet.
Solution: The code can look as follows.

```
# Plot wavelet functions for piecewise linear wavelet, 4 van. moms.
plotwaveletfunctions(IDWTKernelpwl4, -4, 4)
```

7. In Exercise 1 in Section 7.3 you should have found the filters

$$H_0 = \frac{1}{128}\{-5,20,-1,-96,70,\underline{280},70,-96,-1,20,-5\} \quad H_1 = \frac{1}{16}\{1,-4,\underline{6},-4,1\}$$
$$G_0 = \frac{1}{16}\{1,4,\underline{6},4,1\} \quad G_1 = \frac{1}{128}\{5,20,1,-96,-70,\underline{280},-70,-96,1,20,5\}.$$

a. Show that

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

From this we can easily derive the lifting factorization of G.
Solution: The polyphase factorization of the IDWT is

$$\begin{pmatrix} \frac{1}{16}\{1,\underline{6},1\} & \frac{1}{128}\{5,1,\underline{-70},-70,1,5\} \\ \frac{1}{16}\{4,\underline{4}\} & \frac{1}{128}\{20,-96,\underline{280},-96,20\} \end{pmatrix}.$$

We can first apply an even lifting step:

$$\begin{pmatrix} I & -\frac{1}{4}\{\underline{1},1\} \\ \mathbf{0} & I \end{pmatrix}\begin{pmatrix} \frac{1}{16}\{1,\underline{6},1\} & \frac{1}{128}\{5,1,\underline{-70},-70,1,5\} \\ \frac{1}{16}\{4,\underline{4}\} & \frac{1}{128}\{20,-96,\underline{280},-96,20\} \end{pmatrix} = \begin{pmatrix} \frac{1}{16}\{\underline{4}\} & \frac{1}{128}\{20,\underline{-116},-116,20\} \\ \frac{1}{16}\{4,\underline{4}\} & \frac{1}{128}\{20,-96,\underline{280},-96,20\} \end{pmatrix}.$$

We can now apply an odd lifting step

$$\begin{pmatrix} I & \mathbf{0} \\ -\{1,\underline{1}\} & I \end{pmatrix}\begin{pmatrix} \frac{1}{16}\{\underline{4}\} & \frac{1}{128}\{20,\underline{-116},-116,20\} \\ \frac{1}{16}\{4,\underline{4}\} & \frac{1}{128}\{20,-96,\underline{280},-96,20\} \end{pmatrix} = \begin{pmatrix} \frac{1}{4} & \frac{1}{128}\{20,\underline{-116},-116,20\} \\ \mathbf{0} & 4 \end{pmatrix}$$

Since

$$\begin{pmatrix} I & -\frac{1}{512}\{20,\underline{-116},-116,20\} \\ \mathbf{0} & I \end{pmatrix}\begin{pmatrix} \frac{1}{4} & \frac{1}{128}\{20,\underline{-116},-116,20\} \\ \mathbf{0} & 4 \end{pmatrix} = \begin{pmatrix} \frac{1}{4} & \mathbf{0} \\ \mathbf{0} & 4 \end{pmatrix},$$

it follows that

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

b. Implement kernels `DWTKernelN14N24` and `IDWTKernelN14N24` for this wavelet.
Solution: Again we note that we have an even lifting step on the form given in Equation (8.1), and can therefore again use the function `liftingstepeven2symm` in the kernel transformations. Based on what we have computed, the code can look as follows.

```
def DWTKernelN14N24( x, symm, dual):
    """
    Apply the DWT kernel transformation where both the wavelet and
    the dual wavelet have four vanishing moments

    x: The vector which we apply this kernel transformation to
    symm: Whether to apply symmetric extension to the input
    dual: Whether to apply the wavelet kernel or dual wavelet kernel.
    """
    if dual:
        liftingstepoddsymm(1/4., x, symm)
        liftingstepevensymm(1, x, symm)
        liftingstepodd2symm(-29/128., 5/128., x, symm)
        x[0::2] /= 4
```

```
            x[1::2] *= 4
        else:
            liftingstepevensymm(-1/4., x, symm)
            liftingstepoddsymm(-1, x, symm)
            liftingstepeven2symm(29/128., -5/128., x, symm)
            x[0::2] *= 4
            x[1::2] /= 4
```

```
def IDWTKernelN14N24( x, symm, dual):
    """
    Apply the IDWT kernel transformation where both the wavelet and
    the dual wavelet have four vanishing moments

    x: The vector which we apply this kernel transformation to
    symm: Whether to apply symmetric extension to the input
    dual: Whether to apply the wavelet kernel or dual wavelet kernel.
    """
    if dual:
        x[0::2] *= 4
        x[1::2] /= 4
        liftingstepodd2symm(29/128., -5/128., x, symm)
        liftingstepevensymm(-1, x, symm)
        liftingstepoddsymm(-1/4., x, symm)
    else:
        x[0::2] /= 4
        x[1::2] *= 4
        liftingstepeven2symm(-29/128., 5/128., x, symm)
        liftingstepoddsymm(1, x, symm)
        liftingstepevensymm(1/4., x, symm)
```

c. Listen to the low-resolution approximations and detail components in sound for this wavelet.
Solution: The following code can be used for listening to the low-resolution approximations for a given value of m.

```
# Play lowres approx for the piecewise quadratic wavelet
playDWT(m, DWTKernelN14N24, IDWTKernelN14N24, True)
```

d. Use the function `plotwaveletfunctions` from Exercise 1 in Section 7.1 to plot all scaling functions and mother wavelets for this wavelet.
Solution: The code can look as follows.

```
# Plot wavelet functions for the piecewise quadratic wavelet
plotwaveletfunctions(IDWTKernelN14N24, -4, 4)
```

e. We can also implement the kernels of the wavelet of this exercise using what you did in Exercise 12 in Section 6.1 (the filter coefficients are stated at the beginning of this exercise). Use the code form Exercise 12 in Section 6.1 to test that the implementation is correct (test on a randomly generated vector,

and use an assert statement to check whether the error is below a certain tolerance). This test is useful, since it also can be used to test the functions `liftingstepeven2symm` and `liftingstepodd2symm`, and that your lifting factorization is correct.
Solution:

```
H0 = array([-5, 20, -1, -96, 70, 280, 70, -96, -1, 20, -5])/128.
H1 = array([1, -4, 6, -4, 1])/16.
G0 = array([1, 4, 6, 4, 1])/16.
G1 = array([5, 20, 1, -96, -70, 280, -70, -96, 1, 20, 5])/128.
f   =lambda x, symm, dual: DWTKernelFilters(H0,H1,G0,G1,x,symm,dual)
invf=lambda x, symm, dual: IDWTKernelFilters(H0,H1,G0,G1,x,symm,dual)
x = random.random(32)

y1 = x.copy()
y2 = x.copy()
DWTKernelN14N24(y1, 0, 0)
f(y2, 0, 0)
diff = abs(y1-y2).max()
assert diff < 1E-13, 'bug, diff=%s' % diff

y1 = x.copy()
y2 = x.copy()
IDWTKernelN14N24(y1, 0, 0)
invf(y2, 0, 0)
diff = abs(y1-y2).max()
assert diff < 1E-13, 'bug, diff=%s' % diff
```

8.3

1. Run the forward and then the reverse transform from Exercise 2 in Section 6.3 on the vector $(1,2,3,\ldots,8192)$. Verify that there seems to be a delay on 481 elements, as promised by Therorem 8.24. Do you get the exact same result back?
Solution: The following code can be used:

```
# Listen to sound after forward and reverse mp3 filter bank transform
x = arange(1,8193)
mp3reversefbt(mp3forwardfbt(x))
plot(x)
```

There are some small errors from the original vector in the resulting vector, when one compensates for the delay of 481 elements.

2. Use your computer to verify the symmetries we have stated for the symmetries in the prototype filters, i.e. that

$$C_i = \begin{cases} -C_{512-i} & i \neq 64,128,\ldots,448 \\ C_{512-i} & i = 64,128,\ldots,448. \end{cases}$$

Explain also that this implies that $h_i = h_{512-i}$ for $i = 1,\ldots,511$. In other words, the prototype filter has symmetry around $(511+1)/2 = 256$, so that it has linear phase.

3. We mentioned that we could use the lifting factorization to construct filters on the form stated in Equation (8.20), so that the matrices on the form given by Equation (8.24), i.e.

$$\begin{pmatrix} V^{(32-i)} & V^{(i)} \\ -V^{(64-i)} & V^{(32+i)} \end{pmatrix},$$

are invertible. Let us see what kind of lifting steps produce such matrices.

> **a.** Show that the lifting steps $\begin{pmatrix} I & \lambda E_2 \\ \mathbf{0} & I \end{pmatrix}$ and $\begin{pmatrix} I & \mathbf{0} \\ \lambda I & I \end{pmatrix}$ applied in alternating order to a matrix on the form given by Equation (8.24), where the filters are on the from given by Equation (8.20), again produces matrices and filters on these forms. This explains how we can parametrize a larger number of such matrices with the help of lifting steps.It also explain why the inverse matrix is on the form stated in Equation (8.24) with filters on the same form, since the inverse lifting steps are of the same type.
>
> **b.** Explain that 16 numbers $\{\lambda_i\}_{i=1}^{16}$ are needed (together with what we start with on the diagonal in the lifting construction), in order to construct filters so that the prototype filter has 512 coefficients. Since there are 15 submatrices, this gives 240 optimization variables.

Lifting gives the following strategy for finding a corresponding synthesis prototype filter which gives perfect reconstruction: First compute matrices V, W which are inverses of oneanother using lifting (using the lifting steps of this exercise ensures that all filters will be on the form stated in Equation (8.20)), and write

$$\begin{aligned} VW &= \begin{pmatrix} V^{(1)} & V^{(2)} \\ -V^{(3)} & V^{(4)} \end{pmatrix} \begin{pmatrix} W^{(1)} & -W^{(3)} \\ W^{(2)} & W^{(4)} \end{pmatrix} = \begin{pmatrix} V^{(1)} & V^{(2)} \\ -V^{(3)} & V^{(4)} \end{pmatrix} \begin{pmatrix} (W^{(1)})^T & (W^{(2)})^T \\ -(W^{(3)})^T & (W^{(4)})^T \end{pmatrix}^T \\ &= \begin{pmatrix} V^{(1)} & V^{(2)} \\ -V^{(3)} & V^{(4)} \end{pmatrix} \begin{pmatrix} E_{15}(W^{(1)})^T & E_{15}(W^{(2)})^T \\ -E_{15}(W^{(3)})^T & E_{15}(W^{(4)})^T \end{pmatrix}^T \begin{pmatrix} E_{15} & \mathbf{0} \\ \mathbf{0} & E_{15} \end{pmatrix} = I. \end{aligned}$$

Now, the matrices $U^{(i)} = E_{15}(W^{(i)})^T$ are on the form stated in Equation (8.20), and we have that

$$\begin{pmatrix} V^{(1)} & V^{(2)} \\ -V^{(3)} & V^{(4)} \end{pmatrix} \begin{pmatrix} U^{(1)} & U^{(2)} \\ -U^{(3)} & U^{(4)} \end{pmatrix} = \begin{pmatrix} E_{-15} & \mathbf{0} \\ \mathbf{0} & E_{-15} \end{pmatrix}$$

We can now conclude from Theorem 8.23 that if we define the synthesis prototype filter as therein, and set $c = 1, d = -15$, we have that $GH = 16E_{481-32\cdot 15} = 16E_1$.

Chapter 9

9.1

9.2

1. Black and white images can be generated from greyscale images (with values between 0 and 255) by replacing each pixel value with the one of 0 and 255 which is closest. Use this strategy to generate the black and white image shown in Figure 9.2(a).
Solution: The following code can be used:

```
# Exercise 9.2.1
Z = 255*((X[:, :, 0] + X[:, : ,1] + X[:, :, 2])/3. >= 128)
```

2. Generate the right image in Figure 9.9 on your own by writing code which uses the function `contrastadjust`.
Solution: The following code can be used:

```
# Exercise 9.2.2
contrastadjust(X, 0.01)
```

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

a. Write a function `contrastadjust0` which instead uses the function from Equation (9.1) to increase the contrast. n should be a parameter to the function.
Solution: The code could look as follows:

```
def contrastadjust0(X,n):
    """
    Assumes that the values are in [0,255]
    """
    X /= 255.
```

Figure 9.1: Secret message.

```
X -= 1/2.
X *= n
arctan(X, X)
X /= (2*arctan(n/2.))
X += 1/2.0
X *= 255 # Maps the values back to [0,255]
```

b. Generate the left image in Figure 9.9 on your own by using your code from Exercise 2, and instead calling the function `contrastadjust0`.
Solution: The following code can be used:

```
# Exercise 9.2.3
contrastadjust0(X, 10)
```

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 $n \to \infty$.

b. The color image `secret.jpg`, shown in Figure 9.1, 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.
Solution: The secret message is revealed in Figure 9.2.

Figure 9.2: Secret message revealed!

9.3

1. Generate the right images in Figure 9.10 by writing code which calls the function `tensor_impl` with appropriate filters.
Solution: The following code can be used:

```
# Exercise 9.3.1
excerpt = sqrt(X[:,:,0]**2 + X[:,:,1]**2 + X[:,:,2]**2)
mapto01(excerpt)
excerpt *= 255


def shortmolecule(x):
    convkernel(x, array([1., 2., 1.])/4.)

Z1 = zeros((171, 171))
Z1[:, :] = excerpt[169:340, 169:340]
tensor_impl(Z1, shortmolecule, shortmolecule)

def longmolecule(x):
    convkernel(x, array([1., 6., 15., 20., 15., 6., 1.])/64.)

Z2 = zeros((171, 171))
Z2[:, :] = excerpt[169:340, 169:340];
tensor_impl(Z2, longmolecule, longmolecule)
```

2. Generate the right image in Figure 9.12 by writing code in the same way. Also generate the images in figures 9.13, 9.14, and 9.15.
Solution: The following code can be used:

```
# Exercise 9.3.2
excerpt = sqrt(X[:,:,0]**2 + X[:,:,1]**2 + X[:,:,2]**2)
res1 = zeros((171, 171))
res2 = zeros((171, 171))
resxx = zeros((171, 171))
resxy = zeros((171, 171))
resyy = zeros((171, 171))
```

```
res1[:, :] = excerpt[169:340, 169:340]
res2[:, :] = excerpt[169:340, 169:340]
resxx[:, :] = excerpt[169:340, 169:340]
resxy[:, :] = excerpt[169:340, 169:340]
resyy[:, :] = excerpt[169:340, 169:340]

def diffxmolecule(x):
    y = convkernel(x, array([-1., 0., 1.])/2.)

def diffymolecule(x):
    convkernel(x, array([1., 0., -1.])/2.)

def donothing(x):
    a = 2


tensor_impl(res1, donothing, diffymolecule)
tensor_impl(res2, diffxmolecule, donothing)

tensor_impl(resxx, donothing, diffymolecule)
tensor_impl(resxx, donothing, diffymolecule)

tensor_impl(resxy, donothing, diffymolecule)
tensor_impl(resxy, diffxmolecule, donothing)

tensor_impl(resyy, diffxmolecule, donothing)
tensor_impl(resyy, diffxmolecule, donothing)

mapto01(res1); res1 *= 255; Z1 = res1.copy()
mapto01(res2); res2 *= 255
mapto01(resxx); resxx *= 255
mapto01(resxy); resxy *= 255
mapto01(resyy); resyy *= 255

contrastadjust0(Z1, 50)    # Figure 9.10c
Z2 = sqrt(res1**2 + res2**2)      # Figure 9.11a
contrastadjust(res1, 0.01)   # Figure 9.12a
contrastadjust(res2, 0.01)   # Figure 9.12b
contrastadjust0(resxx, 100)  # Figure 9.13a
contrastadjust0(resxy, 100)  # Figure 9.13b
contrastadjust0(resyy, 100)  # Figure 9.13c
```

3. Let the filter S be defined by $S = \{\underline{-1}, 1\}$.

a. Let X be a matrix which represents the pixel values in an image. What can you say about how the new images $(S \otimes I)X$ og $(I \otimes S)X$ look? What are the interpretations of these operations?

b. Write down the $4 \otimes 4$-matrix $X = (1,1,1,1) \otimes (0,0,1,1)$. Compute $(S \otimes I)X$ by applying the filters to the corresponding rows/columns of X as we have learnt, and interpret the result. Do the same for $(I \otimes S)X$.

4. Let S be the moving average filter of length $2L+1$, i.e. $T = \frac{1}{L}\{\underbrace{1, \cdots, 1, \underline{1}, 1, \cdots, 1}_{2L+1 \text{ times}}\}$.

What is the computational molecule of $S \otimes S$?

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 $F(\boldsymbol{x},\alpha\boldsymbol{y}_1+\beta\boldsymbol{y}_2) = \alpha F(\boldsymbol{x},\boldsymbol{y}_1)+\beta F(\boldsymbol{x},\boldsymbol{y}_2)$.
Solution: We have that

$$\begin{aligned} F(\alpha\boldsymbol{x}_1+\beta\boldsymbol{x}_2,\boldsymbol{y}) &= (\alpha\boldsymbol{x}_1+\beta\boldsymbol{x}_2)\otimes\boldsymbol{y} = (\alpha\boldsymbol{x}_1+\beta\boldsymbol{x}_2)\boldsymbol{y}^T \\ &= \alpha\boldsymbol{x}_1\boldsymbol{y}^T+\beta\boldsymbol{x}_2\boldsymbol{y}^T = \alpha(\boldsymbol{x}_1\otimes\boldsymbol{y})+\beta(\boldsymbol{x}\otimes\boldsymbol{y}) \\ &= \alpha F(\boldsymbol{x}_1,\boldsymbol{y})+\beta F(\boldsymbol{x}_1,\boldsymbol{y}). \end{aligned}$$

The second statement follows similarly.

6. Attempt to find matrices $S_1 : \mathbb{R}^M \to \mathbb{R}^M$ and $S_2 : \mathbb{R}^N \to \mathbb{R}^N$ so that the following mappings from $L_{M,N}(\mathbb{R})$ to $L_{M,N}(\mathbb{R})$ can be written on the form $X \to S_1X(S_2)^T = (S_1\otimes S_2)X$. In all the cases, it may be that no such S_1, S_2 can be found. If this is the case, prove it.

a. The mapping which reverses the order of the rows in a matrix.
Solution: Multiplicaton with the matrix

$$S = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 1 & 0 & 0 & \cdots & 0 & 0 & 0 \end{pmatrix}$$

reverses the elements in a vector. This means that

$$((S\otimes I)(\boldsymbol{x}\otimes\boldsymbol{y}))_{i,j} = ((S\boldsymbol{x})\otimes\boldsymbol{y})_{i,j} = (S\boldsymbol{x})_i y_j = x_{M-1-i}y_j = (\boldsymbol{x}\otimes\boldsymbol{y})_{M-1-i,j}.$$

This means that also $((S\otimes I)X)_{i,j} = X_{M-1-i,j}$ for all X, so that $S\otimes I$ reverses rows, and thus is a solution to a..

b. The mapping which reverses the order of the columns in a matrix.
Solution: Similarly one shows that $I\otimes S$ reverses columns, and is thus a solution to b..

c. The mapping which transposes a matrix.
Solution: It turns out that it is impossible to find S_1 and S_2 so that transposing a matrix X corresponds to computing $(S_1\otimes S_2)X$. To see why, S_1 and S_2 would need to fulfill

$$(S_1\otimes S_2)(\boldsymbol{e}_i\otimes\boldsymbol{e}_j) = (S_1\boldsymbol{e}_i)\otimes(S_2\boldsymbol{e}_j) = \boldsymbol{e}_j\otimes\boldsymbol{e}_i,$$

since $\boldsymbol{e}_j\otimes\boldsymbol{e}_i$ is the transpose of $\boldsymbol{e}_i\otimes\boldsymbol{e}_j$. This would require that $S_1\boldsymbol{e}_i = \boldsymbol{e}_j$ for all i,j, which is impossible.

7. Let the filter S be defined by $S = \{1,\underline{2},1\}$.

a. Write down the computational molecule of $S \otimes S$.
Solution: The computational molecule of $S \otimes S$ is

$$\text{rev}(1,\underline{2},1) \otimes \text{rev}(1,\underline{2},1) = (1,\underline{2},1) \otimes (1,\underline{2},1) = \begin{pmatrix} 1 \\ \underline{2} \\ 1 \end{pmatrix} \begin{pmatrix} 1 & \underline{2} & 1 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}.$$

b. Let us define $\boldsymbol{x} = (1,2,3)$, $\boldsymbol{y} = (3,2,1)$, $\boldsymbol{z} = (2,2,2)$, and $\boldsymbol{w} = (1,4,2)$. Compute the matrix $A = \boldsymbol{x} \otimes \boldsymbol{y} + \boldsymbol{z} \otimes \boldsymbol{w}$.
Solution: We get that

$$\begin{aligned} A &= \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \begin{pmatrix} 3 & 2 & 1 \end{pmatrix} + \begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix} \begin{pmatrix} 1 & 4 & 2 \end{pmatrix} \\ &= \begin{pmatrix} 3 & 2 & 1 \\ 6 & 4 & 2 \\ 9 & 6 & 3 \end{pmatrix} + \begin{pmatrix} 2 & 8 & 4 \\ 2 & 8 & 4 \\ 2 & 8 & 4 \end{pmatrix} = \begin{pmatrix} 5 & 10 & 5 \\ 8 & 12 & 6 \\ 11 & 14 & 7 \end{pmatrix}. \end{aligned}$$

c. Compute $(S \otimes S)A$ by applying the filter S to every row and column in the matrix the way we have learnt. If the matrix A was more generally an image, what can you say about how the new image will look?
Solution: We need to compute $(S \otimes S)A = SAS^T$, which corresponds to first applying S to every column in the image, and then applying S to every row in the resulting image. If we apply S to every column in the image we first get the matrix $SA = \begin{pmatrix} 29 & 46 & 23 \\ 32 & 48 & 24 \\ 35 & 50 & 25 \end{pmatrix}$. If we apply the filter to the rows here we get $SAS^T = \begin{pmatrix} 127 & 144 & 121 \\ 136 & 152 & 128 \\ 145 & 160 & 135 \end{pmatrix}$. Since the filter which is applied is a lowpass filter, the new image should look a bit more smooth than the original image.

8. Let $S = \frac{1}{4}\{1,\underline{2},1\}$ be a filter.

a. What is the effect of applying the tensor products $S \otimes I$, $I \otimes S$, and $S \otimes S$ on an image represented by the matrix X?
Solution: Note forst that the filter is a smoothing filter (a lowpass filter). We know that $S \otimes I$ corresponds to applying S to the columns of the matrix, so that we get the result by applying the smoothing filter to the columns of the matrix. The result of this is that horizontal edges are smoothed. Similarly, the tensor product $I \otimes S$ corresponds to applying S to the rows of the matrix, so that vertical edges are smoothed. Finally, $S \otimes S$ corresponds to applying S first to the columns of the matrix, then to the rows. The result is that both horizontal and vertical edges are smoothed. You could also have computed the computational molecules for $S \otimes I$, $I \otimes S$, and $S \otimes S$, by taking the tensor product of the filter coefficients $\frac{1}{4}\{1,\underline{2},1\}$ with itself. From these molecules it is also clear that they either work on the columns, the rows, or on both rows and columns.

b. Compute $(S \otimes S)(\boldsymbol{x} \otimes \boldsymbol{y})$, where $\boldsymbol{x} = (4,8,8,4)$, $\boldsymbol{y} = (8,4,8,4)$ (i.e. both $\boldsymbol{x}$ and $\boldsymbol{y}$ are column vectors).
Solution: A 4×4 circulant Toeplitz matrix for S is

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

From this we can quickly compute that

$$S\boldsymbol{x} = \frac{1}{4}\begin{pmatrix} 2 & 1 & 0 & 1 \\ 1 & 2 & 1 & 0 \\ 0 & 1 & 2 & 1 \\ 1 & 0 & 1 & 2 \end{pmatrix}\begin{pmatrix} 4 \\ 8 \\ 8 \\ 4 \end{pmatrix} = \begin{pmatrix} 2+2+1 \\ 4+2+1 \\ 4+2+1 \\ \text{ï¿¡}2+2+1 \end{pmatrix} = \begin{pmatrix} 5 \\ 7 \\ 7 \\ 5 \end{pmatrix}$$

$$S\boldsymbol{y} = \frac{1}{4}\begin{pmatrix} 2 & 1 & 0 & 1 \\ 1 & 2 & 1 & 0 \\ 0 & 1 & 2 & 1 \\ 1 & 0 & 1 & 2 \end{pmatrix}\begin{pmatrix} 8 \\ 4 \\ 8 \\ 4 \end{pmatrix} = \begin{pmatrix} 4+1+1 \\ 2+2+2 \\ 4+1+1 \\ 2+2+2 \end{pmatrix} = \begin{pmatrix} 6 \\ 6 \\ 6 \\ 6 \end{pmatrix}.$$

From this it is clear that

$$(S \otimes S)(\boldsymbol{x} \otimes \boldsymbol{y}) = (S\boldsymbol{x})(S\boldsymbol{y})^T = \begin{pmatrix} 5 \\ 7 \\ 7 \\ 5 \end{pmatrix}\begin{pmatrix} 6 & 6 & 6 & 6 \end{pmatrix} = \begin{pmatrix} 30 & 30 & 30 & 30 \\ 42 & 42 & 42 & 42 \\ 42 & 42 & 42 & 42 \\ 30 & 30 & 30 & 30 \end{pmatrix}.$$

9. Suppose that we have an image given by the $M \times N$-matrix `X`, and consider the following code:

```
for n in range(N):
    X[0, n] = 0.25*X[N-1, n] + 0.5*X[0, n] + 0.25*X[1, n]
    X[1:(N-1), n] = 0.25*X[0:(N-2), n] + 0.5*X[1:(N-1), n] \
    + 0.25*X[2:N, n]
    X[N-1, n] = 0.25*X[N-2, n] + 0.5*X[N-1, n] + 0.25*X[0, n]
for m in range(m):
    X[m, 0] = 0.25*X[m, M-1] + 0.5*X[m, 0] + 0.25*X[m, 1]
    X[m, 1:(M-1)] = 0.25*X[m, 0:(M-2)] + 0.5*X[m, 1:(M-1)] \
    + 0.25*X[m, 2:M]
    X[m, M-1] = 0.25*X[m, M-2] + 0.5*X[m, M-1] + 0.25*X[m, 0]
```

Which tensor product is applied to the image? Comment what the code does, in particular the first and third line in the inner `for`-loop. What effect does the code have on the image?
Solution: In the code the filter $S = \{1/4, \underline{1/2}, 1/4\}$ is applied to the clumns and the rows in the image. We have learnt that this corresponds to applying the tensor product $S \otimes S$ to the image. `k=1` in the outer `for`-loop corresponds to applying S on the columns, `k=2` corresponds to applying S on the rows. The first and last lines in the inner `for`-loop are necessary since we apply S to the periodic extension of the image. Since S is a smoothing filter, the effect will be that the image is smoothed vertically and horizontally.

10. Let $\boldsymbol{v}_A$ be an eigenvector of A with eigenvalue λ_A, and $\boldsymbol{v}_B$ an eigenvector of B with eigenvalue λ_B. Show that $\boldsymbol{v}_A \otimes \boldsymbol{v}_B$ is an eigenvector of $A \otimes B$ with eigenvalue $\lambda_A \lambda_B$. Explain from this why $\|A \otimes B\| = \|A\|\|B\|$, where $\|\cdot\|$ denotes the operator norm of a matrix.

11. The *Kronecker tensor product* of two matrices A and B, written $A \otimes^k B$, is defined as

$$A \otimes^k B = \begin{pmatrix} a_{11}B & a_{12}B & \cdots & a_{1M}B \\ a_{21}B & a_{22}B & \cdots & a_{2M}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{p1}B & a_{p2}B & \cdots & a_{pM}B \end{pmatrix},$$

where the entries of A are a_{ij}. The tensor product of a $p \times M$-matrix, and a $q \times N$-matrix is thus a $(pq) \times (MN)$-matrix. Note that this tensor product in particular gives meaning for vectors: if $\boldsymbol{x} \in \mathbb{R}^M$, $\boldsymbol{y} \in \mathbb{R}^N$ are column vectors, then $\boldsymbol{x} \otimes^k \boldsymbol{y} \in \mathbb{R}^{MN}$ is also a column vector. In this exercise we will investigate how the Kronecker tensor product is related to tensor products as we have defined them in this section.

a. Explain that, if $\boldsymbol{x} \in \mathbb{R}^M$, $\boldsymbol{y} \in \mathbb{R}^N$ are column vectors, then $\boldsymbol{x} \otimes^k \boldsymbol{y}$ is the column vector where the rows of $\boldsymbol{x} \otimes \boldsymbol{y}$ have first been stacked into one large row vector, and this vector transposed. The linear extension of the operation defined by

$$\boldsymbol{x} \otimes \boldsymbol{y} \in \mathbb{R}^{M,N} \to \boldsymbol{x} \otimes^k \boldsymbol{y} \in \mathbb{R}^{MN}$$

thus stacks the rows of the input matrix into one large row vector, and transposes the result.

b. Show that $(A \otimes^k B)(\boldsymbol{x} \otimes^k \boldsymbol{y}) = (A\boldsymbol{x}) \otimes^k (B\boldsymbol{y})$. We can thus use any of the defined tensor products $\otimes$, $\otimes_k$ to produce the same result, i.e. we have the following commutative diagram,

$$\begin{array}{ccc} \boldsymbol{x} \otimes \boldsymbol{y} & \xrightarrow{A \otimes B} & (A\boldsymbol{x}) \otimes (B\boldsymbol{y}) \\ \downarrow & & \downarrow \\ \boldsymbol{x} \otimes^k \boldsymbol{y} & \xrightarrow{A \otimes^k B} & (A\boldsymbol{x}) \otimes^k (B\boldsymbol{y}), \end{array}$$

where the vertical arrows represent stacking the rows in the matrix, and transposing, and the horizontal arrows represent the two tensor product linear transformations we have defined. In particular, we can compute the tensor product in terms of vectors, or in terms of matrices, and it is clear that the Kronecker tensor product gives the matrix of tensor product operations.

Solution: We have that

$$
\begin{aligned}
&(A \otimes^k B)(\boldsymbol{x} \otimes^k \boldsymbol{y}) \\
&= \begin{pmatrix} a_{11}B & a_{12}B & \cdots & a_{1M}B \\ a_{21}B & a_{22}B & \cdots & a_{2M}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{p1}B & a_{p2}B & \cdots & a_{pM}B \end{pmatrix} \begin{pmatrix} x_1\boldsymbol{y} \\ x_2\boldsymbol{y} \\ \vdots \\ x_M\boldsymbol{y} \end{pmatrix} = \begin{pmatrix} (a_{11}x_1 + \ldots + a_{1M}x_m)B\boldsymbol{y} \\ (a_{21}x_1 + \ldots + a_{2M}x_m)B\boldsymbol{y} \\ \vdots \\ (a_{p1}x_1 + \ldots + a_{pM}x_m)B\boldsymbol{y} \end{pmatrix} \\
&= \begin{pmatrix} (A\boldsymbol{x})_1 B\boldsymbol{y} \\ (A\boldsymbol{x})_2 B\boldsymbol{y} \\ \vdots \\ (A\boldsymbol{x})_p B\boldsymbol{y} \end{pmatrix} = (A\boldsymbol{x}) \otimes_k (B\boldsymbol{y}).
\end{aligned}
$$

c. Using the Euclidean inner product on $L(M,N) = \mathbb{R}^{MN}$, i.e.

$$\langle X, Y \rangle = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} X_{i,j} \overline{Y_{i,j}}.$$

and the correspondence in a. we can define the inner product of $\boldsymbol{x}_1 \otimes \boldsymbol{y}_1$ and $\boldsymbol{x}_2 \otimes \boldsymbol{y}_2$ by

$$\langle \boldsymbol{x}_1 \otimes \boldsymbol{y}_1, \boldsymbol{x}_2 \otimes \boldsymbol{y}_2 \rangle = \langle \boldsymbol{x}_1 \otimes^k \boldsymbol{y}_1, \boldsymbol{x}_2 \otimes^k \boldsymbol{y}_2 \rangle.$$

Show that

$$\langle \boldsymbol{x}_1 \otimes \boldsymbol{y}_1, \boldsymbol{x}_2 \otimes \boldsymbol{y}_2 \rangle = \langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle \langle \boldsymbol{y}_1, \boldsymbol{y}_2 \rangle.$$

Clearly this extends linearly to an inner product on $L_{M,N}$.
Solution: We have that

$$
\begin{aligned}
\langle \boldsymbol{x}_1 \otimes \boldsymbol{y}_1, \boldsymbol{x}_2 \otimes \boldsymbol{y}_2 \rangle &= \left\langle \begin{pmatrix} (\boldsymbol{x}_1)_0 \boldsymbol{y}_1 \\ \vdots \\ (\boldsymbol{x}_1)_{M-1} \boldsymbol{y}_1 \end{pmatrix}, \begin{pmatrix} (\boldsymbol{x}_2)_0 \boldsymbol{y}_2 \\ \vdots \\ (\boldsymbol{x}_2)_{M-1} \boldsymbol{y}_2 \end{pmatrix} \right\rangle = \sum_{i=0}^{M-1} (\boldsymbol{x}_1)_i (\boldsymbol{x}_2)_i \langle \boldsymbol{y}_1, \boldsymbol{y}_2 \rangle \\
&= \langle \boldsymbol{y}_1, \boldsymbol{y}_2 \rangle \sum_{i=0}^{M-1} (\boldsymbol{x}_1)_i (\boldsymbol{x}_2)_i = \langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle \langle \boldsymbol{y}_1, \boldsymbol{y}_2 \rangle.
\end{aligned}
$$

d. Show that the FFT factorization can be written as

$$\begin{pmatrix} F_{N/2} & D_{N/2}F_{N/2} \\ F_{N/2} & -D_{N/2}F_{N/2} \end{pmatrix} = \begin{pmatrix} I_{N/2} & D_{N/2} \\ I_{N/2} & -D_{N/2} \end{pmatrix} (I_2 \otimes_k F_{N/2}).$$

Also rewrite the sparse matrix factorization for the FFT from Equation (2.19) in terms of tensor products.

9.4

1. Implement a function `tensor_impl` which takes a matrix `X`, and functions `S1` and `S2` as parameters, and applies `S1` to the columns of `X`, and `S2` to the rows of `X`.

Explain how you can use this function to implement FFT2, IFFT2, DCT2, and IDCT2.
Solution: The following code can be used:

```
def tensor_impl(X, S1, S2):
    M, N = shape(X)[0:2]
    for n in range(N):
        S1(X[:,n])
    for m in range(M):
        S2(X[m, :])
```

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

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

a. Explain this code line by line.

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

3. Suppose that we have given an image by the matrix `X`. Consider the following code:

```
threshold = 30
[M, N] = shape(X)[0:2]
for n in range(N):
    FFTImpl(X[:, n], FFTKernelStandard)
for m in range(M):
    FFTImpl(X[m, :], FFTKernelStandard)

X = X.*(abs(X) >= threshold)

for n in range(N):
    FFTImpl(X[:, n], FFTKernelStandard, 0)
for m in range(M):
    FFTImpl((X[m, :], FFTKernelStandard, 0)
```

Comment what the code does. Comment in particular on the meaning of the parameter `threshold`, and what effect this has on the image.

Solution: In the first part of the code one makes a change of coordinates with the DFT. More precisely, this is a change of coordinates on a tensor product, as we have defined it. In the last part the change of coordinates is performed the opposite way. Both these change of coordinates is performed is performed the way we have described them, first on the rows in the matrix, then on the columns. The parameter `threshold` is used to neglect DFT-coefficients which are below a certain value. We have seen that this can give various visual artefacts in the image, even though the main contents of the image still may be visible. If we increase `threshold`, these artefacts will be more dominating since we then neglect many DFT-coefficients.

Chapter 10

10.1

10.2

10.3

1. Implement functions `DWT2Impl` and `IDWT2Impl` which perform the m-level DWT2 and the IDWT2, respectively, on an image. The functions should take the same input as `DWTImpl` and `IDWTImpl`, with the input vector replaced with a two-dimensional object. The functions should at each stage call `DWTImpl` and `IDWTImpl` with $m = 1$, and each call to these functions should alter the appropriate upper left submatrix in the coordinate matrix. If the image has several colour components, the functions should be applied to each colour component. There are three colour components in the test image 'lena.png'.
Solution: The following code can be used:

```
def DWT2Impl(X, nres, f, symm=True, dual=False):
    """
    Compute a 2-dimensional DWT. The one-dimensional DWT is applied
    to each row and column in X at each stage. X may have a third
    axis, as is the case for images with more than one color
    component. The DWT2 is then applied to each color component.

    X: A 2-dimensional object for which we apply the 2-dim DWT
    nres: The number of stages
    f: Wavelet kernel to apply. See DWTImpl for documentation
    symm: Whether to apply symmetric extension to the input
    dual: Whether to apply the wavelet kernel or dual wavelet kernel.
    """
    M, N = shape(X)[0:2]
    for res in range(nres):
        for n in range(0,N,2**res):
            f(X[0::2**res, n], symm, dual)
        for m in range(0,M,2**res):
            f(X[m, 0::2**res], symm, dual)
    reorganize_coefficients2(X, nres, True)
```

```
def IDWT2Impl(X, nres, f, symm=True, dual=False):
    """
    Compute a 2-dimensional IDWT. The one-dimensional IDWT is applied
    to each row and column in X at each stage. X may have a third
    axis, as is the case for images with more than one color
    component. The IDWT2 is then applied to each color component.

    X: A 2-dimensional object for which we apply the 2-dim IDWT
    nres: The number of stages
    f: Wavelet kernel to apply. See IDWTImpl for documentation
    symm: Whether to apply symmetric extension to the input
    dual: Whether to apply the wavelet kernel or dual wavelet kernel.
    """
    reorganize_coefficients2(X, nres, False)
    M, N = shape(X)[0:2]
    for res in range(nres - 1, -1, -1):
        for n in range(0, N, 2**res):
            f(X[0::2**res, n], symm, dual)
        for m in range(0, M, 2**res):
            f(X[m, 0::2**res], symm, dual)
```

2. Assume that we have an image represented by the $M \times N$-matrix `X`, and consider the following code:

```
for n in range(N):
    c = (X[0:M:2, n] + X[0:M:2, n])/sqrt(2)
    w = (X[0:M:2, n] - X[0:M:2, n])/sqrt(2)
    X[:, n] = vstack([c, w])

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

a. Comment what the code does, and explain what you will see if you display `X` as an image after the code has run.
Solution: The code runs a DWT over one level, and the Haar wavelet is used. Inside the `for`-loops the DWT is applied to every row and column in the image. `k=1` i the `for`-loop corresponds to applying the DWT to the columns, `k=2` corresponds to applying the DWT to the rows. In the upper left corner we will see a low-resolution version of the image. In the other three corners you will see different tyoes of detail: In the upper right corner you will see detail which corresponds to quick vertical changes, in the lower left corner you will see detail which corresponds to quick horizontal changes, and in the lower right corner you will see points where quick changes both vertically and horizontally occur simultaneously.

b. The code above has an inverse transformation, which reproduce the original image from the transformed values which we obtained. Assume that you zero out the values in the lower left and the upper right corner of the matrix

X after the code above has run, and that you then reproduce the image by applying this inverse transformation. What changes can you then expect in the image?
Solution: By zeroing out the two corners you remove detail which correpond to quick horizontal and vertical changes. But since we keep the lower right corner, we keep detail which corresponds to simultaneous changes vertically and horizontally. The result after the inverse transformation is that most edges have been smoothed, but we see no smoothing effect in points where quick changes occur both horizontally and vertically. In Example 10.14, this corresponds to that we emphasize the gridpoints in the chess pattern, mut that we smooth out the horizontal and vertical edges in the chess pattern.

3. In this exercise we will use the filters $G_0 = \{\underline{1}, 1\}$, $G_1 = \{1, \underline{-1}\}$.

a. Let X be a matrix which represents the pixel values in an image. Define $\boldsymbol{x} = (1,0,1,0)$ and $\boldsymbol{y} = (0,1,0,1)$. Compute $(G_0 \otimes G_0)(\boldsymbol{x} \otimes \boldsymbol{y})$.

b. For a general image X, describe how the images $(G_0 \otimes G_0)X$, $(G_0 \otimes G_1)X$, $(G_1 \otimes G_0)X$, and $(G_1 \otimes G_1)X$ may look.

c. Assume that we run the following code on an image represented by the matrix X:

```
M, N = shape(X)
for n in range(N):
    c = X[0:M:2, n] + X[1:M:2, n]
    w = X[0:M:2, n] - X[1:M:2, n]
    X[:, n] = vstack([c,w])

for m in range(M):
    c = X[m, 0:N:2] + X[m, 1:N:2]
    w = X[m, 0:N:2] - X[m, 1:N:2]
    X[m, :] = hstack([c,w])
```

Comment the code. Describe what will be shown in the upper left corner of X after the code has run. Do the same for the lower left corner of the matrix. What is the connection with the images $(G_0 \otimes G_0)X$, $(G_0 \otimes G_1)X$, $(G_1 \otimes G_0)X$, and $(G_1 \otimes G_1)X$?

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

a. Write a function showDWT,
Solution: The following code achieves the task:

```
def showDWT(m, f, invf, lowres = True):
    """
    Show an image after removing either the detail or the lowres part

    m: The number of resolutions
    f: The DWT kernel
    invf: The IDWT kernel
```

```
        lowres: If true, set the detail to 0 and show the lowres part.
                If false, set the lowres part to 0 and show the detail.
    """
    img = imread('lena.png')
    M, N = shape(img)[0:2]
    DWT2Impl(img, m, f)
    if lowres:
        tokeep = img[0:(M/(2**m)), 0:(N/(2**m))]
        img=zeros_like(img)
        img[0:(M/(2**m)),0:(N/(2**m))] = tokeep
    else:
        img[0:(M/2**m), 0:(N/2**m)] = 0
    IDWT2Impl(img, m, invf)
    mapto01(img)
    img *= 255
    imshow(img.astype(uint8))
```

which takes m, DWT kernel, and IDWT kernel as input, and

1. reads the image file `lena.png`,
2. performs an m-level DWT2 to the image samples using the function `DWT2Impl`, with DWT kernel `f`
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 `IDWT2Impl`, with IDWT kernel `invf`,
5. displays the resulting image.

b. Run the function `showDWT` for different values of m for the Haar wavelet. 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 `showDWT` lie in $[0,255]$?
Solution: There is no reason to believe that image samples returned by the function lie in $[0,255]$. You can check this by printing the maximum value in the returned array on screen inside this method.

5. This exercise parallels the previous exercise, but we instead keep the detail components in the image, and throw away the low-resolution approximation.

a. When you perform the same experiment as in the previous image for the detail components, what kind of image do you see? Can you recognize the original image in what you see?

b. In the code in Example 10.17, set `lowres` to false in the call to `showDWT`. Describe the images you see for different m for the different wavelets. Try to explain why the images seem to get clearer when you increase m.
Solution: After the replacements we get the following code.

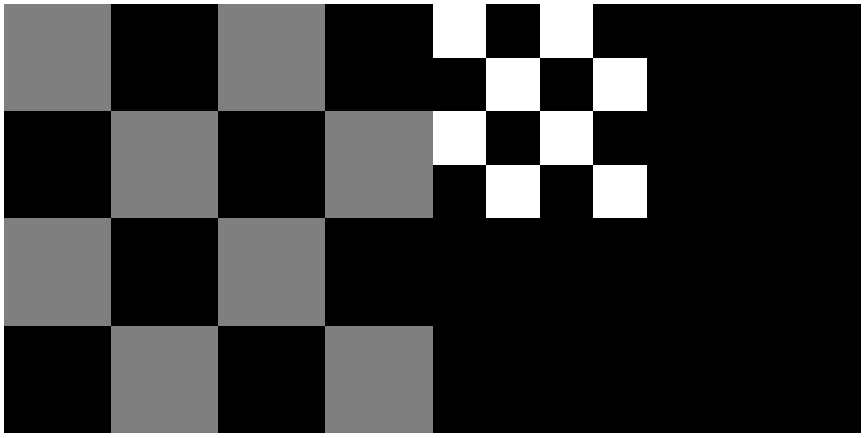

Figure 10.1: A simple image before and after one level of the DWT2. The Haar wavelet was used.

```
# Show detail components for the Haar wavelet
showDWT(m, DWTKernelHaar, IDWTKernelHaar, False)
```

```
# Show detail components for the Spline 5/3 wavelet
showDWT(m, DWTKernel53, IDWTKernel53, False)
```

```
# Show detail components for the CDF 9/7 wavelet
showDWT(m, DWTKernel97, IDWTKernel97, False)
```

6. In Figure 10.1 we have applied the DWT2 with the Haar wavelet to an image very similar to the one you see in Figure 10.6. You see here, however, that there seems to be no detail components, which is very different from Figure 10.6, even though the images are very similar. Attempt to explain what causes this to happen.
Hint: Compare with Exercise 8 in Section 5.3.
Solution: In Figure 10.6, the borders in the chess pattern was chosen so that they occur at odd numbers. This means that the image can not be represented exactly in $V_{m-1} \otimes V_{m-1}$, so that there is detail present in the image at all the borders in the chess pattern. In Figure 10.1, the borders in the chess pattern was chosen so that they occur at even numbers. This means that the image can be represented exactly in $V_{m-1} \otimes V_{m-1}$, so that there is no detail components present in the image.

7. Run the function from Exercise 4 with the Spline 5/3 wavelet and the CDF 9/7 wavelets instead. Look at the result using for different m, using the code from Example 10.16. Can you see any difference from the Haar wavelet? If so, which wavelet gives the best image quality?

10.4

1. Write code which generates the images shown in figures 10.18, 10.19, and 10.20. Use the functions `DWT2Impl` and `IDWT2Impl` with the CDF 9/7 wavelet kernel functions as input.