In addition, we need to change the code for `DCTImpl` so that it calls `FFTImpl2` instead of `FFTImpl`. Even though the code for `DCTImpl` is unchanged, it too has a revised algorithm, since it calls a function which we have changed.

Section 5.2

1 . 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 $\{\phi_{0,0},\phi_{0,1},\dots,\phi_{0,N-1}\}$. 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 $(f(0), f(1),\dots.f(N-1))$ are the coordinates of f.

2 . 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 .a. From lemma 5.11 it follows that

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

This means that

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

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

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

3 .b. From lemma **??** it follows that

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

This means that

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

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 .a. 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 $\text{proj}_{V_0}(\psi) = 0$.

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

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

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

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

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.

4 .c. Follows in the same way as (b).

5 . 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

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

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 . We have that

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

Now, note that

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

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 . The orthogonal decomposition theorem gives that

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

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

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

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.

Section 5.3

1 . 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

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

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.

2 . 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.

3 . 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.

4 . 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.

5 . 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.

6 . 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.

9 . The following code achieves this:

```
[S,fs]=wavread('castanets.wav');
newx=DWTHaarImpl(S(1:2^17,1),2);
plot(0:(2^17-1),newx(1:2^17,1))
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.

1 0.a. The following code achieves the task

```
function playDWTlower(m)
  [S fs]=wavread('../castanets.wav');
  newx=DWTHaarImpl(S(1:2^17,1),m);
  len=length(newx);
  newx((len/2^m+1):length(newx))=zeros(length(newx)-len/2^m,1);
  newx=IDWTHaarImpl(newx,m);
  playerobj=audioplayer(newx,fs);
  playblocking(playerobj);
```

1 0.b. For $m = 2$ we clearly hear a degradation in the sound. For $m = 4$ and above most of the sound is unrecognizable.

1 0.c. 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 . The following code can be used

```
function playDWTlowerdifference(m)
  [S fs]=wavread('../castanets.wav');
  newx=DWTHaarImpl(S(1:2^17,1),m);
  len=length(newx);
  newx(1:(len/2^m))=zeros(len/2^m,1);
  newx=IDWTHaarImpl(newx,m);
  playerobj=audioplayer(newx,fs);
  playblocking(playerobj);
```

13 . Note first that, similarly to the computation in Exercise 5.2.7, 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.

14 . 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.

15 . 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:

```
DWTHaarImpl(repmat([1; -1],512,1),10)
```

Section 5.4

1 . 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 $[f]_{\boldsymbol{\phi}_0}=(f(0),f(1),\dots,f(N-1))$.

3 . We have that

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

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 . 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.

Section 5.5

1 .a. 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 (5.34)). 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.

1 .b. 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 $\frac{1}{4} - \beta$.

1 .c. 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 .a. 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.

```
A=zeros(2);
b=zeros(2,1);
for k=0:1
    A(k+1,:) = [quad(@(t)t.^k.*(1-abs(t)),-1,1)...
                quad(@(t)t.^k.*(1-abs(t-1)),0,2)];
    b(k+1)=quad(@(t)t.^k.*(1-2*abs(t-1/2)),0,1);
end
A\b
```

2 .b. Similarly to a., Equation (5.42) 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 $\alpha, \beta, \gamma, \delta$.

```
A=zeros(4);
b=zeros(4,1);
for k=0:3
    A(k+1,:) = [quad(@(t)t.^k.*(1-abs(t)),-1,1)...
                quad(@(t)t.^k.*(1-abs(t-1)),0,2)...
                quad(@(t)t.^k.*(1-abs(t+1)),-2,0)...
                quad(@(t)t.^k.*(1-abs(t-2)),1,3)];
    b(k+1)=quad(@(t)t.^k.*(1-2*abs(t-1/2)),0,1);
end
coeffs=A\b
```

2 .c. The function $\hat{\psi}$ now is supported on $[-2,3]$, and can be plotted as follows:

```
t=linspace(-2,3,100);
plot(t, (t>=0).*(t<=1).*(1-2*abs(t-0.5)) ...
        -coeffs(1)*(t>=-1).*(t<=1).*(1-abs(t))...
        -coeffs(2)*(t>=0).*(t<=2).*(1-abs(t-1))...
        -coeffs(3)*(t>=-2).*(t<=0).*(1-abs(t+1))...
        -coeffs(4)*(t>=1).*(t<=3).*(1-abs(t-2)))
```

2 .e. 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.

Section 5.6

1 . We write

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

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

$$\begin{aligned} & S_1 \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix} + S_2 \begin{pmatrix} x_{N-2} \\ \vdots \\ x_1 \end{pmatrix} \\ &= S_1 \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix} + (S_2)^f \begin{pmatrix} x_1 \\ \vdots \\ x_{N-2} \end{pmatrix} = S_1 \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix} + \begin{pmatrix} 0 & (S_2)^f & 0 \end{pmatrix} \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix} \\ &= \left(S_1 + \begin{pmatrix} 0 & (S_2)^f & 0 \end{pmatrix} \right) \begin{pmatrix} cx_0 \\ \vdots \\ x_{N/2-1} \end{pmatrix}. \end{aligned}$$

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

Section 6.1

2 . 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 . The Haar wavelet is a counterexample.

Section 6.2

3 .a. 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 $z_n = z_{2N-2-n}$.

3 .b. 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.

3 .c. 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

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

These coincide with the vectors listed in the exercise.

3 .d. Since S is symmetric, it preserves vectors which are symmetric around $N-1$. In the frequency domain, applying S to a vector listed in (6.35) corresponds to multiplying the vectors listed in 6.34 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.

4 .a. 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.

6 .a. `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.

6 .b. This corresponds to reconstructing a low-resolution approximation of the sound.

Section 6.3

1 .a. 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, and we have that `h0`=(1/5, 1/5, 1/5, 1/5, 1/5), and `h1`=(−1/3, 1/3, −1/3).

1 .b. 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, and we have that `g0`=(1/4, 1/2, 1/4), and `g1`=(1/16, −1/4, 3/8, −1/4, 1/16).

2 .a. We have that $H_0 = \{1/16, 1/4, \underline{3/8}, 1/4, 1/16\}$, and $H_1 = \{-1/4, \underline{1/2}, -1/4\}$. 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 $P_{\mathcal{C}_1 \leftarrow \boldsymbol{\phi}_1}$ 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.

2 .b. We have that $G_0 = \frac{1}{3}\{1, \underline{1}, 1\}$, and $G_1 = \frac{1}{5}\{1, -1, \underline{1}, -1, 1\}$. 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.

3 . The following code can be used:

```
function playDWTfilterslower(m,h0,h1,g0,g1)
  [S fs]=wavread('../castanets.wav');
  newx=DWTImpl(h0,h1,S(1:2^17,1),m);
  len=length(newx);
  newx((len/2^m+1):length(newx))=zeros(length(newx)-len/2^m,1);
  newx=IDWTImpl(g0,g1,newx,m);
  playerobj=audioplayer(newx,fs);
  playblocking(playerobj);
```

4 .a. The following code can be used:

```
function playDWTfilterslowerdifference(m,h0,h1,g0,g1)
  [S fs]=wavread('../castanets.wav');
  newx=DWTImpl(h0,h1,S(1:2^17,1),m);
  len=length(newx);
  newx(1:(len/2^m))=zeros(len/2^m,1);
  newx=IDWTImpl(g0,g1,newx,m);
  playerobj=audioplayer(newx,fs);
  playblocking(playerobj);
```

4 .b. After the replacements in the function `playDWTall`, we get code which looks like this

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

5 .a. The code which can be used looks like this:

```
newx=IDWTImpl([1/2 1 1/2]/sqrt(2),[1]/sqrt(2),...
              [-coeffs(1); -coeffs(2); -coeffs(4); 0; 0; 0; 0; ...
               -coeffs(3); 1; 0; 0; 0; 0; 0; 0; 0],1);
```

5 .b. The code which can be used looks like this:

```
g1=[newx((length(newx)-2):length(newx))' newx(1:6)'] % compact filter notation
g0=[1/2 1 1/2]/sqrt(2);
omega=linspace(0,2*pi,100);
plot(omega,g1(5)+g1(6)*2*cos(omega)+g1(7)*2*cos(2*omega)...
           +g1(8)*2*cos(3*omega)+g1(9)*2*cos(4*omega))
```

5 .c. The code which can be used looks like this:

```
alpha=1/(g0(2)*g1(5)+2*g0(3)*g1(6));
h0=alpha*(-1).^(1:(length(g1))).*g1;
h1=alpha*(-1).^(0:(length(g0)-1)).*g0;
```

5 .d. The code which can be used looks like this:

```
for m=1:4
    m
    playDWTfilterslower(m,h0,h1,g0,g1);
    playDWTfilterslowerdifference(m,h0,h1,g0,g1);
end
```

Section 6.4

2 .a. The code can look as follows:

```
function z=mp3forwardfbt(x)
  N=length(x);
  z=zeros(N,1);
  C = mp3ctable(); % The analysis window;
  x = flipud(x);
  x = [x; zeros(512-32,1)];
  % The 32x64 matrix M
  M=cos((2*((0:31)')+1)*((0:63)-16)*pi/64);

  i=[0:63]';i=repmat(i,1,8);
  j=[0:7];j=repmat(j,64,1);
  I_Z=i+64*j+1;

  for n=1:(N/32);
    X = x((length(x)-512-(n-1)*32+1):(length(x)-(n-1)*32));
    Z=C.*X;
    Y=sum(Z(I_Z),2);
    z((1+(n-1)*32):(n*32))=M*Y;
  end;
```

2 .b. The code can look as follows:

```
function x=mp3reversefbt(z)
  Ns=length(z)/32;
  x=zeros(32*Ns,1);
  D = mp3dtable();          % The reconstruction window.
  V=zeros(1024,1);
  % The 64x32 matrix N
  N=cos((16+((0:63)'))*(2*(0:31)+1)*pi/64);

  i1=0:7;i1=repmat(i1,32,1);
  j1=(0:31)';j1=repmat(j1,1,8);
  inds1=64*i1+j1;inds1=inds1(:);
  inds2=128*i1+j1;inds2=inds2(:);

  i2=(0:15)';i2=repmat(i2,1,32);
  j2=0:31;j2=repmat(j2,16,1);

  U=zeros(512,1);
  for n=1:Ns;
    V(65:1024)=V(1:(1024-64));
    V(1:64)=N*z((1+(n-1)*32):(n*32));
    U(inds1+1)=V(inds2+1);
    U(inds1+32+1)=V(inds2+96+1);

    W=U.*D;
    x((1+(n-1)*32):(n*32))=sum(W(j2+32*i2+1),1);
  end
```

Section 7.2

Section 7.3

Section 7.4

Section 7.5

Section 8.1

1 . 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 $S^{(i+r) \bmod M,(j+r) \bmod M}S_{i,j}$.

Section 8.3

2 . The code can look like this:

```
function xnew=liftingstepapplyA(lambda,x)
  N=length(x);
  x(1)=x(1)+2*lambda*x(2); % With symmetric extension
  for k=3:2:(N-1)
    x(k)=x(k)+lambda*(x(k-1)+x(k+1)); % This saves one multiplication
  end
  if mod(N,2)==1
      x(N)=x(N)+2*lambda*x(N-1);
  end
  xnew=x;
```

```
function xnew=liftingstepapplyB(lambda,x)
  N=length(x);
  for k=2:2:(N-1)
    x(k)=x(k)+lambda*(x(k-1)+x(k+1)); % This saves one multiplication
  end
  if mod(N,2)==0
      x(N)=x(N)+2*lambda*x(N-1);
  end
  xnew=x;
```

In this code we have not made the assumption that N is even, for generality.

3 . The code can look like this:

```
function xnew=DWTImpl53(x,m)
lambda1=-1;
lambda2=0.125;
alpha=0.5;
beta=2;

len=length(x);
for mres=1:m
  x(1:2:len)=x(1:2:len)/alpha;
  x(2:2:len)=x(2:2:len)/beta;
  x(1:len)=liftingstepapplyB(-lambda2,x(1:len));
  x(1:len)=liftingstepapplyA(-lambda1,x(1:len));

  % Reorganize the coefficients
  l=x(1:2:len);
  h=x(2:2:len);
  x(1:len)=[l h];
  len=ceil(len/2);
end
xnew=x;
```

```
function x=IDWTImpl53(xnew,m)
  lambda1=-1;
  lambda2=0.125;
  alpha=0.5;
  beta=2;

  lenall=zeros(m+1,1);
  lenall(1)=length(xnew);
  for mres=1:m
      lenall(mres+1)=ceil(lenall(mres)/2);
  end
  % lenall is now the lengths of the (lowpass,highpass) sections of xnew

  for mres=m:(-1):1
    % Reorganize the coefficients first
    l=xnew(1:lenall(mres+1));
    h=xnew((lenall(mres+1)+1):lenall(mres));

    xnew(1:2:lenall(mres))=l;
    xnew(2:2:lenall(mres))=h;

    xnew(1:lenall(mres))=liftingstepapplyA(lambda1,xnew(1:lenall(mres)));
    xnew(1:lenall(mres))=liftingstepapplyB(lambda2,xnew(1:lenall(mres)));
    xnew(1:2:lenall(mres))=xnew(1:2:lenall(mres))*alpha;
    xnew(2:2:lenall(mres))=xnew(2:2:lenall(mres))*beta;
  end
  x=xnew;
```

Here again we have not made the assumption that N is a power of 2.

4 . The code can look like this:

```
function xnew=DWTImpl97(x,m)

lambda1=-0.586134342059950;
lambda2=-0.668067171029734;
lambda3=0.070018009414994;
lambda4=1.200171016244178;
alpha=0.869864451624777;
beta=1.149604398860250;

len=length(x);
for mres=1:m
  x(1:2:len)=x(1:2:len)/alpha;
  x(2:2:len)=x(2:2:len)/beta;
  x(1:len)=liftingstepapplyB(-lambda4,x(1:len));
  x(1:len)=liftingstepapplyA(-lambda3,x(1:len));
  x(1:len)=liftingstepapplyB(-lambda2,x(1:len));
  x(1:len)=liftingstepapplyA(-lambda1,x(1:len));

  % Reorganize the coefficients
  l=x(1:2:len);
  h=x(2:2:len);
  x(1:len)=[l h];
  len=ceil(len/2);
end
xnew=x;
```

```
function x=IDWTImpl97(xnew,m)
  lambda1=-0.586134342059950;
  lambda2=-0.668067171029734;
  lambda3=0.070018009414994;
  lambda4=1.200171016244178;
  alpha=0.869864451624777;
  beta=1.149604398860250;

  lenall=zeros(m+1,1);
  lenall(1)=length(xnew);
  for mres=1:m
```

```
    lenall(mres+1)=ceil(lenall(mres)/2);
  end
  % lenall is now the lengths of the (lowpass,highpass) sections of xnew

  for mres=m:(-1):1
    % Reorganize the coefficients first
    l=xnew(1:lenall(mres+1));
    h=xnew((lenall(mres+1)+1):lenall(mres));
    xnew(1:2:lenall(mres))=l;
    xnew(2:2:lenall(mres))=h;

    xnew(1:lenall(mres))=liftingstepapplyA(lambda1,xnew(1:lenall(mres)));
    xnew(1:lenall(mres))=liftingstepapplyB(lambda2,xnew(1:lenall(mres)));
    xnew(1:lenall(mres))=liftingstepapplyA(lambda3,xnew(1:lenall(mres)));
    xnew(1:lenall(mres))=liftingstepapplyB(lambda4,xnew(1:lenall(mres)));
    xnew(1:2:lenall(mres))=xnew(1:2:lenall(mres))*alpha;
    xnew(2:2:lenall(mres))=xnew(2:2:lenall(mres))*beta;
  end
  x=xnew;
```

Section 8.5

1 . The following code can be used:

```
x=(1:8192)';
xnew=mp3reversefbt(mp3forwardfbt(x));
plot(xnew)
```

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

Section 9.2

1 . The following code can be used:

```
255*((X(:,:,1)+X(:,:,2)+X(:,:,3))/3>=128
```

2 . The following code can be used:

```
imwrite(uint8(contrastadjust(X,0.01)),'contrast4.png','png');
```

3 .a. The code could look as follows:

```
function Z=contrastadjust0(X,n)
  Z = X/255; % Maps the pixel values to [0,1]
  Z = atan(n*(Z-1/2))/(2*atan(n/2)) + 1/2;
  Z = Z*255; % Maps the values back to [0,255]
```

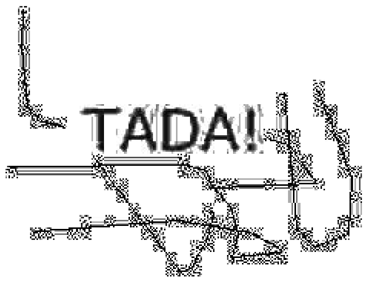

Figure C.1: Secret message revealed!

3 .b. The following code can be used:

```
imwrite(uint8(contrastadjust0(X,10)),'contrast3.png','png');
```

4 .b. The secret message is revealed in Figure C.1.

Section 9.3

1 . The following code can be used:

```
excerpt=255*mapto01(sqrt(X(:,:,1).^2+X(:,:,2).^2+X(:,:,3).^2));
excerpt=excerpt(170:340,170:340);
imwrite(uint8(excerpt),'ggl2part.png','png');
imwrite(uint8(filterimage(excerpt,[1 2 1]/4,[1 2 1]/4)),'smooth1.png','png');
newmolecule=conv([1 3 3 1]/8,[1 3 3 1]/8);
imwrite(uint8(filterimage(excerpt,newmolecule,newmolecule)),'smooth2.png','png');
```

2 . The following code can be used:

```
excerpt=255*mapto01(sqrt(X(:,:,1).^2+X(:,:,2).^2+X(:,:,3).^2));
excerpt=excerpt(170:340,170:340);

res=filterimage(excerpt,[1],[1 0 -1]/2);
imwrite(uint8(contrastadjust0(255*mapto01(res),50)),'px31.png','png');

res1=res;
res2=filterimage(excerpt,[-1 0 1]/2,[1]);
grad=sqrt(res1.^2+res2.^2);
imwrite(uint8(grad),'grad1.png','png');
imwrite(uint8(255*mapto01(grad)),'grad2.png','png');
imwrite(uint8(contrastadjust0(255*mapto01(grad),50)),'grad.png','png');
imwrite(uint8(contrastadjust(255*mapto01(res1),0.01)),'px3.png','png');
imwrite(uint8(contrastadjust(255*mapto01(res2),0.01)),'py.png','png');

resxx=filterimage(res1,[1],[1 0 -1]/2);
resxy=filterimage(res1,[-1 0 1]/2,[1]);
resyy=filterimage(res2,[-1 0 1]/2,[1]);
imwrite(uint8(contrastadjust0(255*mapto01(resxx),100)),'dxx.png','png')
imwrite(uint8(contrastadjust0(255*mapto01(resxy),100)),'dxy.png','png')
imwrite(uint8(contrastadjust0(255*mapto01(resyy),100)),'dyy.png','png')
```

5 . 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 .a. 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..

6 .b. Similarly one shows that $I\otimes S$ reverses columns, and is thus a solution to b..

6 .c. 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 .a. The computational molecule of $S\otimes S$ is

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

7 .b. We get that

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

7 .c. 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 $SA = \begin{pmatrix}29 & 46 & 23\\ \text{Ê}32 & 48 & 24\\ \text{Ê}35 & 50 & 25\end{pmatrix}$. If we apply the filter to the rows here we get $SAS^T = \begin{pmatrix}127 & 144 & 121\\ \text{Ê}136 & 152 & 128\\ \text{Ê}145 & 160 & 135\end{pmatrix}$. Since the filter which is applied is a lowpass filter, the new image should look a bit more smooth than the original image.

8 .a. 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.

8 .b. 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

$$\begin{aligned} S\boldsymbol{x} &= \frac{1}{4}\begin{pmatrix}2 & 1 & 0 & 1\\ 1 & 2 & 1 & 0\\ 0 & 1 & 2 & 1\\ 1 & 0 & 1 & 2\end{pmatrix}\begin{pmatrix}4\\8\\8\\4\end{pmatrix} = \begin{pmatrix}2+2+1\\ 4+2+1\\ 4+2+1\\ \text{Ê}2+2+1\end{pmatrix} = \begin{pmatrix}5\\7\\7\\5\end{pmatrix} \\ S\boldsymbol{y} &= \frac{1}{4}\begin{pmatrix}2 & 1 & 0 & 1\\ 1 & 2 & 1 & 0\\ 0 & 1 & 2 & 1\\ 1 & 0 & 1 & 2\end{pmatrix}\begin{pmatrix}8\\4\\8\\4\end{pmatrix} = \begin{pmatrix}4+1+1\\ 2+2+2\\ 4+1+1\\ 2+2+2\end{pmatrix} = \begin{pmatrix}6\\6\\6\\6\end{pmatrix}. \end{aligned}$$

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 . 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.

1 1.b. We have that

$$\begin{aligned}&(A\otimes^k B)(\boldsymbol{x}\otimes^k\boldsymbol{y})\\ &= \begin{pmatrix} a_{11}B & a_{12}B & \cdots & a_{1M}B\\ a_{21}B & a_{22}B & \cdots & a_{2M}B\\ \vdots & \vdots & \ddots & \vdots\\ a_{p1}B & a_{p2}B & \cdots & a_{pM}B\end{pmatrix}\begin{pmatrix} x_1\boldsymbol{y}\\ x_2\boldsymbol{y}\\ \vdots\\ x_M\boldsymbol{y}\end{pmatrix} = \begin{pmatrix}(a_{11}x_1+\ldots+a_{1M}x_m)B\boldsymbol{y}\\ (a_{21}x_1+\ldots+a_{2M}x_m)B\boldsymbol{y}\\ \vdots\\ (a_{p1}x_1+\ldots+a_{pM}x_m)B\boldsymbol{y}\end{pmatrix}\\ &= \begin{pmatrix}(A\boldsymbol{x})_1B\boldsymbol{y}\\ (A\boldsymbol{x})_2B\boldsymbol{y}\\ \vdots\\ (A\boldsymbol{x})_pB\boldsymbol{y}\end{pmatrix} = (A\boldsymbol{x})\otimes_k(B\boldsymbol{y}).\end{aligned}$$

1 1.c. We have that

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

Section 9.4

1 . The following code can be used:

```
function newX=FFT2Impl(X)
  for k=1:2
      for s=1:size(X,2)
          X(:,s)=FFTImpl(X(:,s));
      end
      X=X';
  end
  newX=X;
```

```
function newX=IFFT2Impl(X)
  for k=1:2
      for s=1:size(X,2)
          X(:,s)=IFFTImpl(X(:,s));
      end
      X=X';
  end
  newX=X;
```

```
function newX=DCT2Impl(X)
  for k=1:2
      for s=1:size(X,2)
          X(:,s)=DCTImpl(X(:,s));
      end
      X=X';
  end
  newX=X;
```

```
function newX=IDCT2Impl(X)
  for k=1:2
      for s=1:size(X,2)
          X(:,s)=IDCTImpl(X(:,s));
      end
      X=X';
  end
  newX=X;
```

2 . The following code can be used:

```
function newX=transform2jpeg(X)
  numblocksx=size(X,1)/8;
  numblocksy=size(X,2)/8;
  for bx=0:(numblocksx-1)
    for by=0:(numblocksy-1)
      X((1+8*bx):8*(bx+1),(1+8*by):8*(by+1))=...
          DCT2Impl(X((1+8*bx):8*(bx+1),(1+8*by):8*(by+1)));
    end
  end
  newX=X;
```

```
function X=transform2invjpeg(newX)
  numblocksx=size(newX,1)/8;
  numblocksy=size(newX,2)/8;
  for bx=0:(numblocksx-1)
```

```
    for by=0:(numblocksy-1)
      newX((1+8*bx):8*(bx+1),(1+8*by):8*(by+1))=...
           IDCT2Impl(newX((1+8*bx):8*(bx+1),(1+8*by):8*(by+1)));
    end
  end
  X=newX;
```

3 . 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.

Section 10.3

1 .a. 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.

1 .b. 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 grid-points in the chess pattern, mut that we smooth out the horizontal and vertical edges in the chess pattern.

3 . The following code can be used:

```
function Xnew=DWT2HaarImpl(X,m)
  for mres=1:m
    l1=size(X,1)/2^(mres-1);
    l2=size(X,2)/2^(mres-1);
    for s=1:l2
      X(1:l1,s)=DWTHaarImpl(X(1:l1,s),1);
    end
    X=X';

    for s=1:l1
      X(1:l2,s)=DWTHaarImpl(X(1:l2,s),1);
    end
    X=X';
  end
  Xnew=X;
```

```
function X=IDWT2HaarImpl(Xnew,m)
  for mres=m:(-1):1
    l1=size(Xnew,1)/2^(mres-1);
    l2=size(Xnew,2)/2^(mres-1);
    for s=1:l2
      Xnew(1:l1,s)=IDWTHaarImpl(Xnew(1:l1,s),1);
    end
    Xnew=Xnew';

    for s=1:l1
      Xnew(1:l2,s)=IDWTHaarImpl(Xnew(1:l2,s),1);
    end
    Xnew=Xnew';
  end
  X=Xnew;
```

4 .a. The following code achieves the task:

```
function showDWTlowerHaar(m)
  X=double(imread('lena.png','png'));
  for k=1:3
      X(:,:,k)=DWT2HaarImpl(X(:,:,k),m);
  end
  [l1,l2,l3]=size(X);
  tokeep=X(1:(l1/(2^m)),1:(l2/(2^m)),:);
  X=zeros(size(X));
  X(1:(l1/(2^m)),1:(l2/(2^m)),:)=tokeep;
  for k=1:3
      X(:,:,k)=IDWT2HaarImpl(X(:,:,k),m);
  end
  imshow(uint8(255*mapto01(X)));
```

4 .c. 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 . The following code can be used

```
function showDWTlowerdifferenceHaar(m)
  img=double(imread('lena.png','png'));
  for k=1:3
      img(:,:,k)=DWT2HaarImpl(img(:,:,k),m);
  end
  [l1,l2,l3]=size(img);
  img(1:(l1/2^m),1:(l2/2^m),:)=zeros(l1/2^m,l1/2^m,3);
  for k=1:3
      img(:,:,k)=IDWT2HaarImpl(img(:,:,k),m);
  end
  imshow(uint8(255*mapto01(img)));
```

6 . 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 10.17, 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 .a. The following code can be used:

```
function Xnew=DWT2Impl53(X,m)
  [l1,l2]=size(X);
  for mres=1:m
    for s=1:l2
      X(1:l1,s)=DWTImpl53(X(1:l1,s),1);
    end
    X=X';

    for s=1:l1
      X(1:l2,s)=DWTImpl53(X(1:l2,s),1);
    end
    X=X';
    l1=ceil(l1/2);
    l2=ceil(l2/2);
  end
  Xnew=X;
```

```
function X=IDWT2Impl53(Xnew,m)
  lenall1=zeros(m+1,1);
  lenall2=zeros(m+1,1);
  lenall1(1)=size(Xnew,1);
  lenall2(1)=size(Xnew,2);
  for mres=1:m
```

```
      lenall1(mres+1)=ceil(lenall1(mres)/2);
      lenall2(mres+1)=ceil(lenall2(mres)/2);
  end
  % lenall is now the lengths of the (lowpass,highpass) sections of xnew

  for mres=m:(-1):1
    for s=1:lenall2(mres)
      Xnew(1:lenall1(mres),s)=IDWTImpl53(Xnew(1:lenall1(mres),s),1);
    end
    Xnew=Xnew';

    for s=1:lenall1(mres)
      Xnew(1:lenall2(mres),s)=IDWTImpl53(Xnew(1:lenall2(mres),s),1);
    end
    Xnew=Xnew';
  end
  X=Xnew;
```

Here we have not made the assumption that the image dimensions are powers of 2.

7 .b. The following code can be used:

```
function Xnew=DWT2Impl97(X,m)
  [l1,l2]=size(X);
  for mres=1:m
    for s=1:l2
      X(1:l1,s)=DWTImpl97(X(1:l1,s),1);
    end
    X=X';

    for s=1:l1
      X(1:l2,s)=DWTImpl97(X(1:l2,s),1);
    end
    X=X';
    l1=ceil(l1/2);
    l2=ceil(l2/2);
  end
  Xnew=X;
```

```
function X=IDWT2Impl97(Xnew,m)
  lenall1=zeros(m+1,1);
  lenall2=zeros(m+1,1);
  lenall1(1)=size(Xnew,1);
  lenall2(1)=size(Xnew,2);
  for mres=1:m
      lenall1(mres+1)=ceil(lenall1(mres)/2);
      lenall2(mres+1)=ceil(lenall2(mres)/2);
  end
  % lenall is now the lengths of the (lowpass,highpass) sections of xnew

  for mres=m:(-1):1
    for s=1:lenall2(mres)
      Xnew(1:lenall1(mres),s)=IDWTImpl97(Xnew(1:lenall1(mres),s),1);
    end
    Xnew=Xnew';

    for s=1:lenall1(mres)
      Xnew(1:lenall2(mres),s)=IDWTImpl97(Xnew(1:lenall2(mres),s),1);
    end
    Xnew=Xnew';
  end
  X=Xnew;
```

8 . The following code can be used:

```
function showDWTlower53(m)
  img=double(imread('lena.png','png'));
  for k=1:3
      img(:,:,k)=DWT2Impl53(img(:,:,k),m);
  end
  [l1,l2,l3]=size(img);
  tokeep=img(1:(l1/(2^m)),1:(l2/(2^m)),:);
  img=zeros(size(img));
  img(1:(l1/(2^m)),1:(l2/(2^m)),:)=tokeep;
  for k=1:3
      img(:,:,k)=IDWT2Impl53(img(:,:,k),m);
  end
  imshow(uint8(255*mapto01(img)));
```

```
function showDWTlower97(m)
  img=double(imread('lena.png','png'));
  for k=1:3
      img(:,:,k)=DWT2Impl97(img(:,:,k),m);
  end
  [l1,l2,l3]=size(img);
  tokeep=img(1:(l1/(2^m)),1:(l2/(2^m)),:);
  img=zeros(size(img));
  img(1:(l1/(2^m)),1:(l2/(2^m)),:)=tokeep;
  for k=1:3
      img(:,:,k)=IDWT2Impl97(img(:,:,k),m);
  end
  imshow(uint8(255*mapto01(img)));
```

9 .a. The following code can be used:

```
function showDWTlowerdifference53(m)
  img=double(imread('lena.png','png'));
  for k=1:3
      img(:,:,k)=DWT2Impl53(img(:,:,k),m);
  end
  [l1,l2,l3]=size(img);
  img(1:(l1/2^m),1:(l2/2^m),:)=zeros(l1/2^m,l1/2^m,3);
  for k=1:3
      img(:,:,k)=IDWT2Impl53(img(:,:,k),m);
  end
  imshow(uint8(255*mapto01(img)));
```

```
function showDWTlowerdifference97(m)
  img=double(imread('lena.png','png'));
  for k=1:3
      img(:,:,k)=DWT2Impl97(img(:,:,k),m);
  end
  [l1,l2,l3]=size(img);
  img(1:(l1/2^m),1:(l2/2^m),:)=zeros(l1/2^m,l1/2^m,3);
  for k=1:3
      img(:,:,k)=IDWT2Impl97(img(:,:,k),m);
  end
  imshow(uint8(255*mapto01(img)));
```

9 .b. After the replacements in the function `showDWTall`, we get code which looks like this

```
function showDWTalldifference(m)
  disp('Haar wavelet');
  showDWTlowerdifferenceHaar(m);
  disp('5/3 wavelet');
  showDWTlowerdifference53(m);
  disp('9/7 wavelet');
  showDWTlowerdifference97(m);
```

Section 10.4

Bibliography

[1] *ISO/IEC FDIS 15444-1. JPEG2000 Part 1 final draft international standard*, 2000.

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

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

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

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

[6] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex fourier series. *Math. Comput.*, 19:297Ð301, 1965.

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

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

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

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

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

[12] G. B. Folland. *Real analysis. Modern techniques and their applications*. John Wiley and sons, 1984.