Chapter 3

Operations on digital sound: digital filters

In Section 1.5 we defined filters as operations on continuous sound which preserved different frequencies. Such operations are important since they can change the frequency content in many ways. They are difficult to use computationally, however, since they are defined for all instances in time. As when we defined the DFT to make Fourier series computable, we would like to define *digital filters*, in order to make filters computable. It will turn out that such digital filters can be computed by the following procedure:

$$z_n = \frac{1}{4}(x_{n-1} + 2x_n + x_{n+1}), \quad \text{for } n = 0, 1, \ldots, N-1. \tag{3.1}$$

Here $\boldsymbol{x}$ denotes the *input vector*, and $\boldsymbol{z}$ the *output vector*. In other words, the output of a digital filter is constructed by combining several input elements linearly. The concrete filter defined by Equation (3.1) is called a *smoothing filter*, as we will demonstrate that it smooths the variations in the sound. We will start this chapter by looking at matrix representations for operations as given by Equation (3.1). Then we will formally define digital filters in terms of preservation of frequencies as we did for the filters in Chapter 1, and show that this is equivalent to operations on the form (3.1).

3.1 Matrix representations of filters

Let us consider Equation (3.1) in some more detail to get more intuition about filters. As before we assume that the input vector is periodic with period N, so that $x_{n+N} = x_n$. Our first observation is that the output vector $\boldsymbol{z}$ is also periodic with period N since

$$z_{n+N} = \frac{1}{4}(x_{n+N-1} + 2x_{n+N} + x_{n+N+1}) = \frac{1}{4}(x_{n-1} + 2x_n + x_{n+1}) = z_n.$$

The filter is also clearly a linear transformation and may therefore be represented by an $N \times N$ matrix S that maps the vector $\boldsymbol{x} = (x_0, x_1, \ldots, x_{N-1})$ to the vector $\boldsymbol{z} = (z_0, z_1, \ldots, z_{N-1})$, i.e., we have $\boldsymbol{z} = S\boldsymbol{x}$. To find S, for $1 \leq n \leq N-2$ it is clear from Equation (3.1) that row n has the value $1/4$ in column $n-1$, the value $1/2$ in column n, and the value $1/4$ in column $n+1$. For row 0 we must be a bit more careful, since the index -1 is outside the legal range of the indices. This is where the periodicity helps us out so that

$$z_0 = \frac{1}{4}(x_{-1} + 2x_0 + x_1) = \frac{1}{4}(x_{N-1} + 2x_0 + x_1) = \frac{1}{4}(2x_0 + x_1 + x_{N-1}).$$

From this we see that row 0 has the value $1/4$ in columns 1 and $N-1$, and the value $1/2$ in column 0. In exactly the same way we can show that row $N-1$ has the entry $1/4$ in columns 0 and $N-2$, and the entry $1/2$ in column $N-1$. In summary, the matrix of the smoothing filter is given by

$$S = \frac{1}{4}\begin{pmatrix} 2 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \\ 1 & 2 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 2 & 1 \\ 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 2 \end{pmatrix}. \tag{3.2}$$

A matrix on this form is called a Toeplitz matrix. The general definition is as follows and may seem complicated, but is in fact quite straightforward:

Definition 3.1. *Toeplitz matrices.*

An $N \times N$-matrix S is called a Toeplitz matrix if its elements are constant along each diagonal. More formally, $S_{k,l} = S_{k+s,l+s}$ for all nonnegative integers k, l, and s such that both $k+s$ and $l+s$ lie in the interval $[0, N-1]$. A Toeplitz matrix is said to be circulant if in addition

$$S_{(k+s) \bmod N,(l+s) \bmod N} = S_{k,l}$$

for all integers k, l in the interval $[0, N-1]$, and all s (Here mod denotes the remainder modulo N).

Toeplitz matrices are very popular in the literature and have many applications. A Toeplitz matrix is constant along each diagonal, while the additional property of being circulant means that each row and column of the matrix 'wraps over' at the edges. Clearly the matrix given by Equation (3.2) satisfies Definition 3.1 and is a circulant Toeplitz matrix. A Toeplitz matrix is uniquely identified by the values on its nonzero diagonals, and a circulant Toeplitz matrix is uniquely identified by the values on the main diagonal, and on the diagonals above (or under) it. Toeplitz matrices show up here in the context of filters, but they will also show up later in the context of wavelets.

Equation (3.1) leads us to the more general expression

$$z_n = \sum_k t_k x_{n-k}. \tag{3.3}$$

If $\boldsymbol{t}$ has infinitely many nonzero entries, the sum is an infinite one, and may diverge. We will, however, mostly assume that $\boldsymbol{t}$ has a finite number of nonzero entries. This general expression opens up for defining many types of operations. The values t_k will be called *filter coefficients*. The range of k is not specified, but is typically an interval around 0, since z_n usually is calculated by combining x_k's with indices close to n. Both positive and negative indices are allowed. As an example, for formula (3.1) k ranges over $-1, 0$, and 1, and we have that $t_{-1} = t_1 = 1/4$, and $t_0 = 1/2$. Since Equation (3.3) needs to be computed for each n, if only $t_0, \ldots, t_{kmax}$ are nonzero, we need to go through the following `for`-loop to compute $z_{kmax}, \ldots, z_{N-1}$:

```
z = zeros(1, N);
for n = kmax:(N-1)
    for k = 0:kmax
        z(n + 1) = z(n + 1) + t(k + 1)*x(n - k + 1);
    end
end
```

It is clearly possible to vectorize the inner loop here, since it takes the form of a dot product. Another possible way to vectorize is to first change the order of summation, and then vectorize as follows

```
z = zeros(1, N);
for k = 0:kmax
    z((kmax+1):N) = z((kmax+1):N) + t(k + 1)*x((kmax-k+1):(N-k));
end
```

Depending on how vectorization is supported, this code will in general execute faster, and is to prefer. The drawback, however, is that a filter often is applied in real time, with the output computed only when enough input is available, with the input becoming available continuously. This second approach then clearly fails, since it computes nothing before all input is available. In the exercise we will compare the computation times for the two approaches above, and compare them with a built-in function which computes the same.

Note that above we did not consider the first entries in $\boldsymbol{z}$, since this is where the circulation occurs. Taken this into account, the first filter we considered in this chapter can be implemented in vectorized form simply as

```
z(1)       = x(2)/4 + x(1)/2 + x(N)/4;
z(2:(N-1)) = x(3:N)/4 + x(2:(N-1))/2 + x(1:(N-2))/4;
z(N)       = x(1))/4 + x(N)/2 + x(N-1)/4;
```

In the following we will avoid such implementations, since `for`-loops can be very slow. We will see that an efficient built-in function exists for computing this, and use this instead.

By following the same argument as above, the following is clear:

Proposition 3.2. *Filters as matrices.*

Any operation defined by Equation (3.3) is a linear transformation which transforms a vector of period N to another of period N. It may therefore be represented by an $N \times N$ matrix S that maps the vector $\boldsymbol{x} = (x_0, x_1, \ldots, x_{N-1})$ to the vector $\boldsymbol{z} = (z_0, z_1, \ldots, z_{N-1})$, i.e., we have $\boldsymbol{z} = S\boldsymbol{x}$. Moreover, the matrix S is a circulant Toeplitz matrix, and the first column $\boldsymbol{s}$ of this matrix is given by

$$s_k = \begin{cases} t_k, & \text{if } 0 \leq k < N/2; \\ t_{k-N} & \text{if } N/2 \leq k \leq N-1. \end{cases} \tag{3.4}$$

In other words, the first column of S can be obtained by placing the coefficients in (3.3) with positive indices at the beginning of $\boldsymbol{s}$, and the coefficients with negative indices at the end of $\boldsymbol{s}$.

This proposition will be useful for us, since it explains how to pass from the form (3.3), which is most common in practice, to the matrix form S. Since the filter coefficients t_k uniquely define any $N \times N$-circulant Toeplitz matrix, we will establish the following shorthand notation for the filter matrix for a given set of filter coefficients. We will use this notation only when we have a finite set of nonzero filter coefficients (note however that many interesting filters in signal processing have infinitely many nonzero filter coefficients, see Section 3.5) Note also that we always choose N so large that the placement of the filter coefficients in the first column, as dictated by Proposition 3.2, never collide (as happens when N is smaller than the number of filter coefficients).

Definition 3.3. *Compact notation for filters.*

Let $k_{\min}$, $k_{\max}$ be the smallest and biggest index of a filter coefficient in Equation (3.3) so that $t_k \neq 0$ (if no such values exist, let $k_{\min} = k_{\max} = 0$), i.e.

$$z_n = \sum_{k=k_{min}}^{k_{max}} t_k x_{n-k}. \tag{3.5}$$

We will use the following compact notation for S:

$$S = \{t_{k_{min}}, \ldots, t_{-1}, \underline{t_0}, t_1, \ldots, t_{k_{max}}\}.$$

In other words, the entry with index 0 has been underlined, and only the nonzero t_k's are listed. k_{max} and k_{min} are also called the start and end indices of S. By the length of S, denoted $l(S)$, we mean the number $k_{max} - k_{min}$.

One seldom writes out the matrix of a filter, but rather uses this compact notation.

Example 3.1: Finding the matrix elements from the filter coefficients

Let us apply Proposition 3.2 to the operation defined by formula (3.1):

- for $k = 0$ Equation (3.4) gives $s_0 = t_0 = 1/2$.
- For $k = 1$ Equation (3.4) gives $s_1 = t_1 = 1/4$.
- For $k = N - 1$ Equation (3.4) gives $s_{N-1} = t_{-1} = 1/4$.

For all k different from 0, 1, and $N - 1$, we have that $s_k = 0$. Clearly this gives the matrix in Equation (3.2).

Example 3.2: Finding the filter coefficients from the matrix

Proposition 3.2 is also useful when we have a circulant Toeplitz matrix S, and we want to find filter coefficients t_k so that $\boldsymbol{z} = S\boldsymbol{x}$ can be written on the form (3.3). Consider the matrix

$$S = \begin{pmatrix} 2 & 1 & 0 & 3 \\ 3 & 2 & 1 & 0 \\ 0 & 3 & 2 & 1 \\ 1 & 0 & 3 & 2 \end{pmatrix}.$$

This is a circulant Toeplitz matrix with $N = 4$, and we see that $s_0 = 2$, $s_1 = 3$, $s_2 = 0$, and $s_3 = 1$. The first equation in (3.4) gives that $t_0 = s_0 = 2$, and $t_1 = s_1 = 3$. The second equation in (3.4) gives that $t_{-2} = s_2 = 0$, and $t_{-1} = s_3 = 1$. By including only the t_k which are nonzero, the operation can be written as

$$z_n = t_{-1}x_{n-(-1)} + t_0x_n + t_1x_{n-1} + t_2x_{n-2} = x_{n+1} + 2x_0 + 3x_{n-1}.$$

Example 3.3: Writing down compact filter notation

Using the compact notation for a filter, we would write $S = \{1/4, \underline{1/2}, 1/4\}$ for the filter given by formula (3.1)). For the filter

$$z_n = x_{n+1} + 2x_0 + 3x_{n-1}$$

from Example 3.2, we would write $S = \{1, \underline{2}, 3\}$.

3.1.1 Convolution

Applying a filter to a vector $\boldsymbol{x}$ is also called taking the *convolution* of the two vectors $\boldsymbol{t}$ and $\boldsymbol{x}$. Convolution is usually defined without the assumption that the input vector is periodic, and without any assumption on the vector lengths (i.e. they may be sequences of inifinite length). The case where both vectors $\boldsymbol{t}$ and $\boldsymbol{x}$ have a finite number of nonzero elements dererves extra attention. Assume that $t_0, \ldots, t_{M-1}$ and $x_0, \ldots, x_{N-1}$ are the only nonzero elements in $\boldsymbol{t}$ and $\boldsymbol{x}$ (i.e. we can view them as vectors in $\mathbb{R}^M$ and $\mathbb{R}^N$, respectively). It is clear from the expression $z_n = \sum t_k x_{n-k}$ that only $z_0, \ldots, z_{M+N-2}$ can be nonzero. This motivates the following definition.

Definition 3.4. *Convolution of vectors.*

By the *convolution* of two vectors $\boldsymbol{t} \in \mathbb{R}^M$ and $\boldsymbol{x} \in \mathbb{R}^N$ we mean the vector $\boldsymbol{t} * \boldsymbol{x} \in \mathbb{R}^{M+N-1}$ defined by

$$(\boldsymbol{t} * \boldsymbol{x})_n = \sum_k t_k x_{n-k}, \tag{3.6}$$

where we only sum over k so that $0 \leq k < M$, $0 \leq n - k < N$.

Note that convolution in the literature usually assumes infinite vectors. Matlab has the built-in function `conv` for computing $\boldsymbol{t} * \boldsymbol{x}$. As we shall see in the exercises this function is highly optimized, and is therefore much used in practice. Since convolution is not exactly the same as our definition of a filter (since we assume that a vector is repeated periodically), it would be a good idea to express our definition of filters in terms of convolution. This can be achieved with the next proposition, which is formulated for the case with equally many filter coefficients with negative and positive indices. The result is thus directly applicable for symmetric filters, which is the type of filters we will mostly concentrate on. It is a simple exercise to generalize the result to other filters, however.

Proposition 3.5. *Using convolution to compute filters.*

Assume that S is a filter on the form

$$S = \{t_{-L}, \ldots, \underline{t_0}, \ldots, t_L\}.$$

If $\boldsymbol{x} \in \mathbb{R}^N$, then $S\boldsymbol{x}$ can be computed as follows:

- Form the vector $\tilde{\boldsymbol{x}} = (x_{N-L}, \cdots, x_{N-1}, x_0, \cdots, x_{N-1}, x_0, \cdots, x_{L-1}) \in \mathbb{R}^{N+2L}$.
- Use the `conv` function to compute $\tilde{\boldsymbol{z}} = \boldsymbol{t} * \tilde{\boldsymbol{x}} \in \mathbb{R}^{M+N+2L-1}$.
- We have that $S\boldsymbol{x} = (\tilde{z}_{2L}, \ldots, \tilde{z}_{M+N-2})$.

We will consider an implementation of this result using the `conv` function in the exercises.

Proof. When $\boldsymbol{x} \in \mathbb{R}^N$, the operation $\boldsymbol{x} \to \boldsymbol{t} * \boldsymbol{x}$ can be represented by an $(M + N - 1) \times N$ matrix. It is easy to see that this matrix has element $(i + s, i)$ equal to t_s, for $0 \leq i < M$, $0 \leq s < N$. In the left part of Figure 3.1 such a matrix is shown for $M = 5$. The (constant) nonzero diagonals are shown as diagonal lines.

Now, form the vector $\tilde{\boldsymbol{x}} \in \mathbb{R}^{N+2L}$ as in the text of the theorem. Convolving $(t_{-L}, \ldots, t_L)$ with vectors in $\mathbb{R}^{N+2L}$ can similarly be represented by an $(M+N+2L-1) \times (N+2L)$-matrix. The rows from $2L$ up to and including $M + N - 2$ in this matrix (we have marked these with horizontal lines above) make up a new matrix $\tilde{S}$, shown in the right part of Figure 3.1 ($\tilde{S}$ is an $N \times (N + 2L)$ matrix).

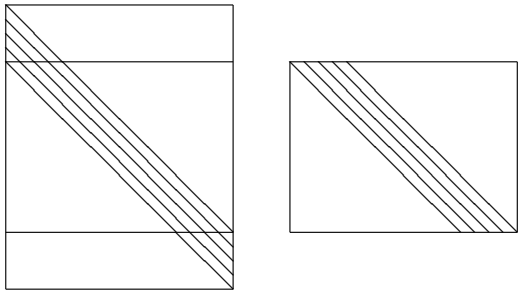

Figure 3.1: Matrix for the operation $\boldsymbol{x} \to \boldsymbol{t} * \boldsymbol{x}$ (left), as well as this matrix with the first and last $2L$ rows dropped (right).

We need to show that $S\boldsymbol{x} = \tilde{S}\tilde{\boldsymbol{x}}$. We have that $\tilde{S}\tilde{\boldsymbol{x}}$ equals the matrix shown in the left part of Figure 3.2 multiplied with $(x_{N-L}, \ldots, x_{N-1}, x_0, \ldots, x_{N-1}, x_0, \ldots, x_{L-1})$ (we inserted extra vertical lines in the matrix where circulation occurs), which equals the matrix shown in the right part of Figure 3.2 multiplied with $(x_0, \ldots, x_{N-1})$. We see that this is $S\boldsymbol{x}$, and the proof is complete.

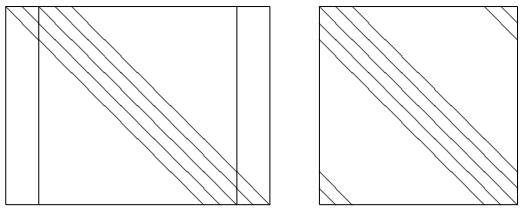

Figure 3.2: The matrix we multiply with $(x_{N-L}, \ldots, x_{N-1}, x_0, \ldots, x_{N-1}, x_0, \ldots, x_{L-1})$ (left), and the matrix we multiply with $(x_0, \ldots, x_{N-1})$ (right).

□

There is also a very nice connection between convolution and polynomials:

Proposition 3.6. *Convolution and polynomials.*
Assume that $p(x) = a_N x^N + a_{N-1}x_{N-1} + \ldots, a_1 x + a_0$ and $q(x) = b_M x^M + b_{M-1}x_{M-1} + \ldots, b_1 x + b_0$ are polynomials of degree N and M respectively. Then the coefficients of the polynomial pq can be obtained by computing `conv(a,b)`.

We can thus interpret a filter as a polynomial. In this setting, clearly the length $l(S)$ of the filter can be interpreted as the degree of the polynomial. If $\boldsymbol{t} \in \mathbb{R}^M$ and $\boldsymbol{x} \in \mathbb{R}^N$, then they can be associated with polynomials of degree $M-1$ and $N-1$, respectively. Also, their convolution, which is in $\mathbb{R}^{M+N-1}$, can be associated with a polynomial of degree $M+N-2$, which is the sum of the degrees of the individual polynomials. Of course we can make the same addition

of degrees when we multiply polynomials. Clearly the polynomial associated with $\boldsymbol{t}$ is the frequency response, when we insert $x = e^{-i\omega}$. Also, applying two filters in succession is equivalent to applying the convolution of the filters, so that two filtering operations can be combined to one.

Since the number of nonzero filter coefficients is typically much less than N (the period of the input vector), the matrix S have many entries which are zero. Multiplication with such matrices requires less additions and multiplications than for other matrices: If S has k nonzero filter coefficients, S has Nk nonzero entries, so that kN multiplications and $(k-1)N$ additions are needed to compute $S\boldsymbol{x}$. This is much less than the N^2 multiplications and $(N-1)N$ additions needed in the general case. Perhaps more important is that we need not form the entire matrix, we can perform the matrix multiplication directly in a loop. For large N we risk running into out of memory situations if we had to form the entire matrix.

Exercise 3.4: Finding the filter coefficients and the matrix

Assume that the filter S is defined by the formula

$$z_n = \frac{1}{4}x_{n+1} + \frac{1}{4}x_n + \frac{1}{4}x_{n-1} + \frac{1}{4}x_{n-2}.$$

Write down the filter coefficients t_k, and the matrix for S when $N = 8$.

Exercise 3.5: Finding the filter coefficients from the matrix

Given the circulant Toeplitz matrix

$$S = \begin{pmatrix} 1 & 2 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 0 & 1 & 2 \\ 2 & 0 & 0 & 1 \end{pmatrix},$$

write down the filter coefficients t_k.

Exercise 3.6: Convolution and polynomials

Compute the convolution of $\{\underline{1}, 2, 1\}$ with itself. interpret the result in terms of two polynomials.

Exercise 3.7: Implementation of convolution

Implement code which computes $\boldsymbol{t} * \boldsymbol{x}$ in the two ways described after Equation (3.3), i.e. as a double for loop, and as a simple for loop in `k`, with `n` vectorized. As your $\boldsymbol{t}$, take k randomly generated numbers. Compare execution times for these two methods and the `conv` function, for different values of k. Present the result as a plot where k runs along the x-axis, and execution times run along the y-axis. Your result will depend on how Matlab performs vectorization.

Exercise 3.8: Filters with a different number of coefficients with positive and negative indices

Assume that $S = \{t_{-E}, \ldots, \underline{t_0}, \ldots, t_F\}$. Formulate a generalization of Proposition 3.5 for such filters, i.e. to filters where there may be a different number of filter coefficients with positive and negative indices. You should only need to make some small changes to the proof of Proposition 3.5 to achieve this.

Exercise 3.9: Implementing filtering with convolution

Implement a function `filterS` which uses Proposition 3.5 and the `conv` function $S\boldsymbol{x}$ when $S = \{t_{-L}, \ldots, \underline{t_0}, \ldots, t_L\}$ The function should take the vectors $(t_{-L}, \ldots, t_0, \ldots, t_L)$ and $\boldsymbol{x}$ as input.

3.2 Formal definition of filters and the vector frequency response

Let us now define digital filters formally, and establish their relationship to Toeplitz matrices. We have seen that a sound can be decomposed into different frequency components, and we would like to define filters as operations which adjust these frequency components in a predictable way. One such example is provided in Example 2.16, where we simply set some of the frequency components to 0. The natural starting point is to require for a filter that the output of a pure tone is a pure tone with the same frequency.

Definition 3.7. *Digital filters and vector frequency response.*

A linear transformation $S : \mathbb{R}^N \mapsto \mathbb{R}^N$ is a said to be a digital filter, or simply a filter, if, for any integer n in the range $0 \leq n \leq N-1$ there exists a value $\lambda_{S,n}$ so that

$$S(\phi_n) = \lambda_{S,n}\phi_n, \tag{3.7}$$

i.e., the N Fourier vectors are the eigenvectors of S. The vector of (eigen)values $\boldsymbol{\lambda}_S = (\lambda_{S,n})_{n=0}^{N-1}$ is often referred to as the *(vector) frequency response* of S.

Since the Fourier basis vectors are orthogonal vectors, S is clearly orthogonally diagonalizable. Since also the Fourier basis vectors are the columns in $(F_N)^H$, we have that

$$S = F_N^H D F_N \tag{3.8}$$

whenever S is a digital filter, where D has the frequency response (i.e. the eigenvalues) on the diagonal [1]. We could also use DFT_N to diagonalize filters, but it is customary to use an orthogonal matrix (i.e. F_N) when the matrix is

[1] Recall that the orthogonal diagonalization of S takes the form $S = PDP^T$, where P contains as columns an orthonormal set of eigenvectors, and D is diagonal with the eigenvectors listed on the diagonal (see Section 7.1 in [8]).

orthogonally diagonalizable. In particular, if S_1 and S_2 are digital filters, we can write $S_1 = F_N^H D_1 F_N$ and $S_2 = F_N^H D_2 F_N$, so that

$$S_1 S_2 = F_N^H D_1 F_N F_N^H D_2 F_N = F_N^H D_1 D_2 F_N.$$

Since $D_1 D_2 = D_2 D_1$ for any diagonal matrices, we get the following corollary:

Corollary 3.8. *The product of two filters is a filter.*

The product of two digital filters is again a digital filter. Moreover, all digital filters commute, i.e. if S_1 and S_2 are digital filters, $S_1 S_2 = S_2 S_1$.

Clearly also $S_1 + S_2$ is a filter when S_1 and S_2 are. The set of all filters is thus a vector space, which also is closed under multiplication. Such a space is called an *algebra*. Since all filters commute, this algebra is also called a *commutative algebra*.

3.2.1 Time delay

Time delay with d elements is defined by $E_d(\boldsymbol{x} = \boldsymbol{z}$, with $z_k = x_{k-d}$. From this it follows that the vector with components $e^{2\pi ikn/N}$ is sent to the vector with components $e^{2\pi i(k-d)n/N} = e^{-2\pi dn/N} e^{2\pi ikn/N}$. This means that ϕ_n is an eigenvector, so that time delay with d elements is a digital filter. Since $e^{-2\pi dn/N}$ is the corresponding eigenvalue for ϕ_n, this also gives us the frequency response. Since all $|\lambda_{S,n}| = 1$, time-delay does not change the amplitude of frequencies in sounds.

The next result states three equivalent characterizations of a digital filter. The first one is simply the definition in terms of having the Fourier basis as eigenvectors. The second is that the matrix is circulant Toeplitz, i.e. that the operations we started this chapter with actually are filters. The third characterization is in terms of a new concept which we now define.

Definition 3.9. *Time-invariance.*

Assume that S is a linear transformation from $\mathbb{R}^N$ to $\mathbb{R}^N$. Let $\boldsymbol{x}$ be input to S, and $\boldsymbol{y} = S\boldsymbol{x}$ the corresponding output. Let also $\boldsymbol{z}$, $\boldsymbol{w}$ be delays of $\boldsymbol{x}$, $\boldsymbol{y}$ with d elements (i.e. $\boldsymbol{z} = E_d \boldsymbol{x}$, $\boldsymbol{w} = E_d \boldsymbol{y}$). S is said to be *time-invariant* if, for any d and $\boldsymbol{x}$, $S\boldsymbol{z} = \boldsymbol{w}$ (i.e. S sends the delayed input vector to the delayed output vector, equivalently $SE_d = E_d S$).

Clearly time delay is time-invariant, since $E_{d_1} E_{d_2} = E_{d_2} E_{d_1} = E_{d_1 + d_2}$ for any d_1 and d_2.

Theorem 3.10. *Characterizations of digital filters.*

The following are equivalent characterizations of a digital filter:

- $S = (F_N)^H D F_N$ for a diagonal matrix D, i.e. the Fourier basis is a basis of eigenvectors for S.
- S is a circulant Toeplitz matrix.

- S is linear and time-invariant.

Proof. If S is a filter, then $SE_d = E_dS$ for all d since all filters commute, so that S is time-invariant. This proves 1. $\to$ 3..

Assume that S is time-invariant. Note that $E_d\boldsymbol{e}_0 = \boldsymbol{e}_d$, and since $SE_d\boldsymbol{e}_0 = E_dS\boldsymbol{e}_0$ we have that $S\boldsymbol{e}_d = E_d\boldsymbol{s}$, where $\boldsymbol{s}$ is the first column of S. This also says that column d of S can be obtained by delaying the first column of S with d elements. But then d is a circulant Toeplitz matrix. This proves 3. $\to$ 2..

Finally, any circulant Toeplitz matrix can be written on the form $\sum_{d=0}^{N-1} s_dE_d$ (by splitting the matrix into a sum of its diagonals). Since all E_d are filters, it is clear that any circulant Toeplitz matrix is a filter. This proves 2. $\to$ 1.. □

Due to this result, filters are also called *LTI filters*, LTI standing for Linear, Time-Invariant. Also, operations defined by (3.3) are digital filters, when restricted to vectors with period N. The following results enables us to compute the eigenvalues/frequency response easily through the DFT, so that we do not need to form the characteristic polynomial and find its roots:

Theorem 3.11. *Connection between frequency response and the matrix.*

Any digital filter is uniquely characterized by the values in the first column of its matrix. Moreover, if $\boldsymbol{s}$ is the first column in S, the frequency response of S is given by

$$\boldsymbol{\lambda}_S = \mathrm{DFT}_N\boldsymbol{s}. \tag{3.9}$$

Conversely, if we know the frequency response $\boldsymbol{\lambda}_S$, the first column $\boldsymbol{s}$ of S is given by

$$\boldsymbol{s} = \mathrm{IDFT}_N\boldsymbol{\lambda}_S. \tag{3.10}$$

Proof. If we replace S by $(F_N)^H DF_N$ we find that

$$\mathrm{DFT}_N\boldsymbol{s} = \sqrt{N}F_N\boldsymbol{s} = \sqrt{N}F_NS\begin{pmatrix}1\\0\\\vdots\\0\end{pmatrix} = \sqrt{N}F_NF_N^HDF_N\begin{pmatrix}1\\0\\\vdots\\0\end{pmatrix}$$
$$= \sqrt{N}DF_N\begin{pmatrix}1\\0\\\vdots\\0\end{pmatrix} = D\begin{pmatrix}1\\\vdots\\1\end{pmatrix} = \boldsymbol{\lambda}_S,$$

where we have used that the first column in F_N has all entries equal to $1/\sqrt{N}$, and that the diagonal matrix D has all the eigenvalues of S on its diagonal, so that the last expression is the vector of eigenvalues $\boldsymbol{\lambda}_S$. This proves (3.9). Equation (3.10) follows directly by applying the inverse DFT to (3.9). □

The first column $\boldsymbol{s}$, which thus characterizes the filter, is also called the *impulse response*. This name stems from the fact that we can write $\boldsymbol{s} = S\boldsymbol{e}_0$, i.e. the vector $\boldsymbol{s}$ is the output (often called response) to the vector $\boldsymbol{e}_0$ (often called an impulse). Equation (3.9) states that the frequency response can be written as

$$\lambda_{S,n} = \sum_{k=0}^{N-1} s_k e^{-2\pi i n k/N}, \quad \text{for } n = 0, 1, \ldots, N-1, \tag{3.11}$$

where s_k are the components of $\boldsymbol{s}$.

The identity matrix is a digital filter since $I = (F_N)^H I F_N$. Since $\boldsymbol{e}_0$ is the first column, it has impulse response $\boldsymbol{s} = \boldsymbol{e}_0$. Its frequency response has 1 in all components and therefore preserves all frequencies, as expected.

In signal processing, the frequency content of a vector (i.e., its DFT) is also referred to as its spectrum. This may be somewhat confusing from a linear algebra perspective, because in this context the term spectrum is used to denote the eigenvalues of a matrix. But because of Theorem 3.11 this is not so confusing after all if we interpret the spectrum of a vector (in signal processing terms) as the spectrum of the corresponding digital filter (in linear algebra terms).

Example 3.10: Frequency response of a simple filter

When only few of the coefficients s_k are nonzero, it is possible to obtain nice expressions for the frequency response. To see this, let us compute the frequency response of the filter defined from formula (3.1). We saw that the first column of the corresponding Toeplitz matrix satisfied $s_0 = 1/2$, and $s_{N-1} = s_1 = 1/4$. The frequency response is thus

$$\begin{aligned}\lambda_{S,n} &= \frac{1}{2}e^0 + \frac{1}{4}e^{-2\pi i n/N} + \frac{1}{4}e^{-2\pi i n(N-1)/N} \\ &= \frac{1}{2}e^0 + \frac{1}{4}e^{-2\pi i n/N} + \frac{1}{4}e^{2\pi i n/N} = \frac{1}{2} + \frac{1}{2}\cos(2\pi n/N).\end{aligned}$$

Example 3.11: Matrix form

We have seen that the DFT can be used to spare us the tedious calculation of eigenvectors and eigenvalues we are used to, at least for circulant Toeplitz matrices. Let us compare the two approaches for a simple matrix.

$$S = \begin{pmatrix} 4 & 1 \\ 1 & 4 \end{pmatrix}.$$

It is straightforward to compute the eigenvalues and eigenvectors of this matrix the way you learned in your first course in linear algebra. However, this matrix is also a circulant Toeplitz matrix, so that we can use the results in this section to compute the eigenvalues and eigenvectors. Since here $N = 2$, we have that $e^{2\pi i n k/N} = e^{\pi i n k} = (-1)^{nk}$. This means that the Fourier basis vectors are

$(1,1)/\sqrt{2}$ and $(1,-1)/\sqrt{2}$, which also are the eigenvectors of S. The eigenvalues are the frequency response of S, which can be obtained as

$$\sqrt{N}F_N\boldsymbol{s} = \sqrt{2}\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}4\\1\end{pmatrix} = \begin{pmatrix}5\\3\end{pmatrix}$$

The eigenvalues are thus 3 and 5. You could have obtained the same result with your computer. Note that the computer may not return the eigenvectors exactly as the Fourier basis vectors, since the eigenvectors are not unique (the multiple of an eigenvector is also an eigenvector). The computer may for instance switch the signs of the eigenvectors. We have no control over what the computer chooses to do, since some underlying numerical algorithm for computing eigenvectors is used, which we can't influence.

Example 3.12: Computing the output of a filter

Certain vectors are easy to express in terms of the Fourier basis. This enables us to compute the output of such vectors from a digital filter easily.

Let us consider the filter S defined by $z_n = \frac{1}{6}(x_{n+2} + 4x_{n+1} + 6x_n + 4x_{n-1} + x_{n-2})$, and see how we can compute $S\boldsymbol{x}$ when

$$\boldsymbol{x} = (\cos(2\pi 5\cdot 0/N), \cos(2\pi 5\cdot 1/N), \ldots, \cos(2\pi 5\cdot (N-1)/N))\,,$$

where N is the length of the vector. We note first that

$$\begin{aligned}\sqrt{N}\phi_5 &= \left(e^{2\pi i 5\cdot 0/N}, e^{2\pi i 5\cdot 1/N}, \ldots, e^{2\pi i 5\cdot (N-1)/N}\right)\\ \sqrt{N}\phi_{N-5} &= \left(e^{-2\pi i 5\cdot 0/N}, e^{-2\pi i 5\cdot 1/N}, \ldots, e^{-2\pi i 5\cdot (N-1)/N}\right),\end{aligned}$$

Since $e^{2\pi i 5k/N} + e^{-2\pi i 5k/N} = 2\cos(2\pi 5k/N)$, we get by adding the two vectors that $\boldsymbol{x} = \frac{1}{2}\sqrt{N}(\phi_5 + \phi_{N-5})$. Since the ϕ_n are eigenvectors, we have expressed $\boldsymbol{x}$ as a sum of eigenvectors. The corresponding eigenvalues are given by the vector frequency response, so let us compute this. If $N = 8$, computing $S\boldsymbol{x}$ means to multiply with the 8×8 circulant Toeplitz matrix

$$\frac{1}{6}\begin{pmatrix}6 & 4 & 1 & 0 & 0 & 0 & 1 & 4\\ 4 & 6 & 4 & 1 & 0 & 0 & 0 & 1\\ 1 & 4 & 6 & 4 & 1 & 0 & 0 & 0\\ 0 & 1 & 4 & 6 & 4 & 1 & 0 & 0\\ 0 & 0 & 1 & 4 & 6 & 4 & 1 & 0\\ 0 & 0 & 0 & 1 & 4 & 6 & 4 & 1\\ 1 & 0 & 0 & 0 & 1 & 4 & 6 & 4\\ 4 & 1 & 0 & 0 & 0 & 1 & 4 & 6\end{pmatrix}$$

We now see that

$$\begin{aligned}\lambda_{S,n} &= \frac{1}{6}(6 + 4e^{-2\pi in/N} + e^{-2\pi i2n/N} + e^{-2\pi i(N-2)n/N} + 4e^{-2\pi i(N-1)n/N}) \\ &= \frac{1}{6}(6 + 4e^{2\pi in/N} + 4e^{-2\pi in/N} + e^{2\pi i2n/N} + e^{-2\pi i2n/N}) \\ &= 1 + \frac{4}{3}\cos(2\pi n/N) + \frac{1}{3}\cos(4\pi n/N).\end{aligned}$$

The two values of this we need are

$$\begin{aligned}\lambda_{S,5} &= 1 + \frac{4}{3}\cos(2\pi 5/N) + \frac{1}{3}\cos(4\pi 5/N) \\ \lambda_{S,N-5} &= 1 + \frac{4}{3}\cos(2\pi(N-5)/N) + \frac{1}{3}\cos(4\pi(N-5)/N) \\ &= 1 + \frac{4}{3}\cos(2\pi 5/N) + \frac{1}{3}\cos(4\pi 5/N).\end{aligned}$$

Since these are equal, $\boldsymbol{x}$ is a sum of eigenvectors with equal eigenvalues. This means that $\boldsymbol{x}$ itself also is an eigenvector, with the same eigenvalue, so that

$$S\boldsymbol{x} = \left(1 + \frac{4}{3}\cos(2\pi 5/N) + \frac{1}{3}\cos(4\pi 5/N)\right)\boldsymbol{x}.$$

3.2.2 Using digital filters to approximate filters

The formal definition of digital filters resembles that of filters from Chapter 1, the difference being that the Fourier basis now is discrete. Let us try to connect the two. In doing so, let us differ them by calling the ones from Chapter 1 simply filters (without *digital* in front). We have the following result.

Theorem 3.12. *Connection with the frequency response.*

Let s be a filter with frequency response $\lambda_s(f)$, and assume that $f \in V_{M,T}$ (so that also $s(f) \in V_{M,T}$). Let

$$\begin{aligned}\boldsymbol{x} &= (f(0 \cdot T/N), f(1 \cdot T/N), \ldots, f((N-1)T/N)) \\ \boldsymbol{z} &= (s(f)(0 \cdot T/N), s(f)(1 \cdot T/N), \ldots, s(f)((N-1)T/N))\end{aligned}$$

be vectors of $N = 2M + 1$ uniform samples from f and $s(f)$. Then the operation $S : \boldsymbol{x} \to \boldsymbol{z}$ (i.e. the operation which sends the samples of the input to the samples of the output) is well-defined on $\mathbb{R}^N$, and is an $N \times N$-digital filter with vector frequency response $\lambda_{S,n} = \lambda_s(n/T)$.

Proof. With $N = 2M + 1$ we know that $f \in V_{M,T}$ is uniquely determined from $\boldsymbol{x}$. This means that $s(f)$ also is uniquely determined from $\boldsymbol{x}$, so that $\boldsymbol{z}$ also is uniquely determined from $\boldsymbol{x}$. The operation $S : \boldsymbol{x} \to \boldsymbol{z}$ is therefore well-defined on $\mathbb{R}^N$.

Clearly also $s(e^{2\pi int/T}) = \lambda_s(n/T)e^{2\pi int/T}$. Since the samples of $e^{2\pi int/T}$ is the vector $e^{2\pi ikn/N}$, and the samples of $\lambda_s(n/T)e^{2\pi int/T}$ is $\lambda_s(n/T)e^{2\pi ikn/N}$, the vector $e^{2\pi ikn/N}$ is an eigenvector of S with eigenvalue $\lambda_s(n/T)$. Clearly then S is a digital filter with frequency response $\lambda_{S,n} = \lambda_s(n/T)$. □

It is interesting that the vector frequency response above is obtained by sampling the frequency response. In this way we also see that it is easy to realize any digital filter as the restriction of a filter: an filter s where the frequency response has the values $\lambda_{S,n}$ at the points n/T will do. In the theorem it is essential that $f \in V_{M,T}$. There are many functions with the same samples, but where the samples of the output from the filter are different. When we restrict to $V_{M,T}$, however, the output samples are always determined from the input samples.

Theorem 3.12 explains how digital filters can occur in practice. In the real world, a signal is modeled as a continuous function $f(t)$, and an operation on signals as a filter s. We can't compute the entire output $s(f)$ of the filter, but it is possible to apply the digital filter from Theorem 3.12 to the samples $\boldsymbol{x}$ of f. In general $f(t)$ may not lie in $V_{M,T}$, but we can denote by $\tilde{f}$ the unique function in $V_{M,T}$ with the same samples as f (as in Section 2.2). By definition, $S\boldsymbol{x}$ are the samples of $s(\tilde{f}) \in V_{M,T}$. $s(\tilde{f})$ can finally be found from these samples by using the procedure from Figure 2.4 for finding $s(\tilde{f})$. This procedure for finding $s(\tilde{f})$ is illustrated in Figure 3.3.

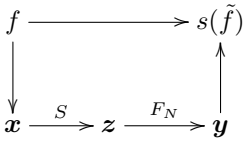

Figure 3.3: The connections between filters and digital filters, sampling and interpolation, provided by Theorem 3.12. The left vertical arrow represents sampling, the right vertical arrow represents interpolation.

Clearly, $s(\tilde{f})$ is an approximation to $s(f)$, since $\tilde{f}$ is an approximation to f, and since s is continuous. Let us summarize this as follows:

Idea 3.13. *Approximating a filter.*
A filter s can be approximated through sampling, a digital filter, the DFT, and interpolation, as illustrated in Figure 3.3. S is the digital filter with frequency response $\lambda_{S,n} = \lambda_s(n/T)$. When $f \in V_{M,T}$, this approximation equals $s(f)$. When we increase the number of sample points/the size of the filter, the approximation becomes better. If there is a bound on the highest frequency in f, there exists an N so that when sampling of that size, the approximation equals $s(f)$.

Let us comment on why the last statements here are true. That the approximation equals $s(f)$ when $f \in V_{M,T}$ is obvious, since both f and $s(f) \in V_{M,T}$

are determined from their samples then. If there is a bound on the highest frequency in f, then f lies in $V_{M,T}$ for large enough M, so that we recover $s(f)$ as our approximation using $N = 2M + 1$. Finally, what happens when there is no bound on the highest frequency? We know that $s(f_N) = (s(f))_N$. Since f_N is a good approximation to f, the samples $\boldsymbol{x}$ of f are close to the samples of f_N. By continuity of the digital filter, $\boldsymbol{z} = S\boldsymbol{x}$ will also be close to the samples of $(s(f))_N = s(f_N)$, so that (also by continuity) interpolating with $\boldsymbol{z}$ gives a good approximation to $(s(f))_N$, which is again a good approximation to $s(f)$). From this it follows that the digital filter is a better approximation when N is high.

Exercise 3.13: Time reversal is not a filter

In Example 1.2 we looked at time reversal as an operation on digital sound. In $\mathbb{R}^N$ this can be defined as the linear mapping which sends the vector $\boldsymbol{e}_k$ to $\boldsymbol{e}_{N-1-k}$ for all $0 \leq k \leq N-1$.

a) Write down the matrix for the time reversal linear mapping, and explain from this why time reversal is not a digital filter.

b) Prove directly that time reversal is not a time-invariant operation.

Exercise 3.14: When is a filter symmetric?

Let S be a digital filter. Show that S is symmetric if and only if the frequency response satisfies $\lambda_{S,n} = \lambda_{S,N-n}$ for all n.

Exercise 3.15: Eigenvectors and eigenvalues

Consider the matrix

$$S = \begin{pmatrix} 4 & 1 & 3 & 1 \\ 1 & 4 & 1 & 3 \\ 3 & 1 & 4 & 1 \\ 1 & 3 & 1 & 4 \end{pmatrix}.$$

a) Compute the eigenvalues and eigenvectors of S using the results of this section. You should only need to perform one DFT in order to achieve this.

b) Verify the result from a) by computing the eigenvectors and eigenvalues the way you taught in your first course in linear algebra. This should be a much more tedious task.

c) Use a computer to compute the eigenvectors and eigenvalues of S also. For some reason some of the eigenvectors seem to be different from the Fourier basis vectors, which you would expect from the theory in this section. Try to find an explanation for this.

Exercise 3.16: Composing filters

Assume that S_1 and S_2 are two circulant Toeplitz matrices.

a) How can you express the eigenvalues of $S_1 + S_2$ in terms of the eigenvalues of S_1 and S_2?

b) How can you express the eigenvalues of $S_1 S_2$ in terms of the eigenvalues of S_1 and S_2?

c) If A and B are general matrices, can you find a formula which expresses the eigenvalues of $A + B$ and AB in terms of those of A and B? If not, can you find a counterexample to what you found in a) and b)?

Exercise 3.17: Keeping every second component

Consider the linear mapping S which keeps every second component in $\mathbb{R}^N$, i.e. $S(\boldsymbol{e}_{2k}) = \boldsymbol{e}_{2k}$, and $S(\boldsymbol{e}_{2k-1}) = \mathbf{0}$. Is S a digital filter?

3.3 The continuous frequency response and properties

If we make the substitution $\omega = 2\pi n/N$ in the formula for $\lambda_{S,n}$, we may interpret the frequency response as the values on a continuous function on $[0, 2\pi)$.

Theorem 3.14. *Connection between vector- and continuous frequency response.*
The function $\lambda_S(\omega)$ defined on $[0, 2\pi)$ by

$$\lambda_S(\omega) = \sum_k t_k e^{-ik\omega}, \tag{3.12}$$

where t_k are the filter coefficients of S, satisfies

$$\lambda_{S,n} = \lambda_S(2\pi n/N) \text{ for } n = 0,\, 1,\, \ldots,\, N-1$$

for any N. In other words, regardless of N, the vector frequency response lies on the curve λ_S. $\lambda_S(\omega))$ is called the *continuous frequency response* of S, and ω is called *angular frequency*.

The difference between the vector- and continuous frequency response lies in that one uses the filter coefficients t_k, while the other uses the impulse response s_k. These contain the same values, but they are ordered differently. The result shows that, at the points $2\pi n/N$, they are equal.

Proof. For any N we have that

$$\begin{aligned}
\lambda_{S,n} &= \sum_{k=0}^{N-1} s_k e^{-2\pi ink/N} = \sum_{0\le k<N/2} s_k e^{-2\pi ink/N} + \sum_{N/2\le k\le N-1} s_k e^{-2\pi ink/N} \\
&= \sum_{0\le k<N/2} t_k e^{-2\pi ink/N} + \sum_{N/2\le k\le N-1} t_{k-N} e^{-2\pi ink/N} \\
&= \sum_{0\le k<N/2} t_k e^{-2\pi ink/N} + \sum_{-N/2\le k\le -1} t_k e^{-2\pi in(k+N)/N} \\
&= \sum_{0\le k<N/2} t_k e^{-2\pi ink/N} + \sum_{-N/2\le k\le -1} t_k e^{-2\pi ink/N} \\
&= \sum_{-N/2\le k<N/2} t_k e^{-2\pi ink/N} = \lambda_S(2\pi n/N).
\end{aligned}$$

where we have used Equation (3.4). □

If `t` is the set of filter coefficients of S, we can combine Theorem 3.14 with Equation (3.9), and use the fact that time delay does not affect the absolute value of the DFT (Theorem 2.7), in order to plot the frequency response of S as follows:

```
omega = 2*pi*(0:(N-1))/N;
s = [t; zeros(N - length(t), 1)];
plot(omega, abs(fft(s)));
```

With this procedure we avoid computing the frequency response by hand. We will have use for this several places later.

Note that $\sum_{k=0}^{N-1} s_k e^{-\pi i\omega}$ typically will not converge when $N \to \infty$ (although it gives the right values at all points $\omega = 2\pi n/N$ for all N)! The filter coefficients avoid this convergence problem, however, since we assume that only t_k with $|k|$ small are nonzero. In other words, filter coefficients are used in the definition of the continuous frequency response so that we can find a continuous curve where we can find the vector frequency response values for all N.

The frequency response contains the important characteristics of a filter, since it says how it behaves for the different frequencies. When analyzing a filter, we therefore often plot the frequency response. Often we plot only the absolute value (or the magnitude) of the frequency response, since this is what explains how each frequency is amplified or attenuated. Since λ_S is clearly periodic with period 2π, we may restrict angular frequency to the interval $[0, 2\pi)$. The conclusion in Observation 2.10 was that the low frequencies in a vector correspond to DFT indices close to 0 and $N-1$, and high frequencies correspond to DFT indices close to $N/2$. This observation is easily translated to a statement about angular frequencies:

Observation 3.15. *Plotting the frequency response.*

When plotting the frequency response on $[0, 2\pi)$, angular frequencies near 0 and 2π correspond to low frequencies, angular frequencies near π correspond to high frequencies

λ_S may also be viewed as a function defined on the interval $[-\pi, \pi)$. Plotting on $[-\pi, \pi]$ is often done in practice, since it makes clearer what corresponds to lower frequencies, and what corresponds to higher frequencies:

Observation 3.16. *Higher and lower frequencies.*
When plotting the frequency response on $[-\pi, \pi)$, angular frequencies near 0 correspond to low frequencies, angular frequencies near $\pm\pi$ correspond to high frequencies.

The following holds:

Theorem 3.17. *Connection between analog and digital filters.*
Assume that s is an analog filter, and that we sample a periodic function at rate f_s over one period, and denote the corresponding digital filter by S. The analog and digital frequency responses are related by $\lambda_s(f) = \lambda_S(2\pi f f_s)$.

To see this, note first that S has frequency response $\lambda_{S,n} = \lambda_s(n/T) = \lambda_s(f)$, where $f = n/T$. We then rewrite $\lambda_{S,n} = \lambda_S(2\pi n/N) = \lambda_S(2\pi fT/N) = \lambda_S(2\pi f f_s)$.
Since the frequency response is essentially a DFT, it inherits several properties from Theorem 2.7. We will mostly use the continuous frequency response to express these properties.

Theorem 3.18. *Properties of the frequency response.*
We have that

- The continuous frequency response satisfies $\lambda_S(-\omega) = \overline{\lambda_S(\omega)}$.
- If S is a digital filter, S^T is also a digital filter. Moreover, if the frequency response of S is $\lambda_S(\omega)$, then the frequency response of S^T is $\overline{\lambda_S(\omega)}$.
- If S is symmetric, λ_S is real. Also, if S is antisymmetric (the element on the opposite side of the diagonal is the same, but with opposite sign), λ_S is purely imaginary.
- A digital filter S is an invertible if and only if $\lambda_{S,n} \neq 0$ for all n. In that case S^{-1} is also a digital filter, and $\lambda_{S^{-1},n} = 1/\lambda_{S,n}$.
- If S_1 and S_2 are digital filters, then S_1S_2 also is a digital filter, and $\lambda_{S_1S_2}(\omega) = \lambda_{S_1}(\omega)\lambda_{S_2}(\omega)$.

Proof. Property 1. and 3. follow directly from Theorem 2.7. Transposing a matrix corresponds to reversing the first column of the matrix and thus also the filter coefficients. Due to this Property 2. also follows from Theorem 2.7. If $S = (F_N)^H D F_N$, and all $\lambda_{S,n} \neq 0$, we have that $S^{-1} = (F_N)^H D^{-1} F_N$, where

D^{-1} is a diagonal matrix with the values $1/\lambda_{S,n}$ on the diagonal. Clearly then S^{-1} is also a digital filter, and its frequency response is $\lambda_{S^{-1},n} = 1/\lambda_{S,n}$, which proves 4. The last property follows in the same was as we showed that filters commute:

$$S_1 S_2 = (F_N)^H D_1 F_N (F_N)^H D_2 F_N = (F_N)^H D_1 D_2 F_N.$$

The frequency response of $S_1 S_2$ is thus obtained by multiplying the frequency responses of S_1 and S_2. □

In particular the frequency response may not be real, although this was the case in the first example of this section. Theorem 3.18 applies also for the vector frequency response. Since the vector frequency response are the eigenvalues of the filter, the last property above says that, for filters, multiplication of matrices corresponds to multiplication of eigenvalues. Clearly this is an important property which is shared with all other matrices which have the same eigenvectors.

Example 3.18: Plotting a simple frequency response

In Example 3.10 we computed the vector frequency response of the filter defined in formula (3.1). The filter coefficients are here $t_{-1} = 1/4$, $t_0 = 1/2$, and $t_1 = 1/4$. The continuous frequency response is thus

$$\lambda_S(\omega) = \frac{1}{4}e^{i\omega} + \frac{1}{2} + \frac{1}{4}e^{-i\omega} = \frac{1}{2} + \frac{1}{2}\cos\omega.$$

Clearly this matches the computation from Example 3.10. Figure 3.4 shows plots of this frequency response, plotted on the intervals $[0, 2\pi)$ and $[-\pi, \pi)$.

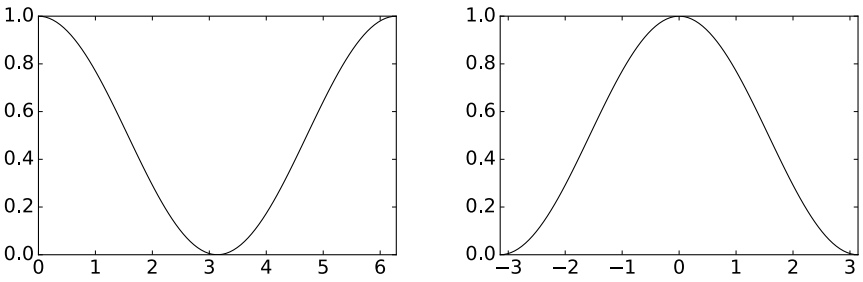

Figure 3.4: $|\lambda_S(\omega)|$ of the moving average filter of Formula (3.1), plotted over $[0, 2\pi]$ and $[-\pi, \pi]$.

Both the continuous frequency response and the vector frequency response for $N = 51$ are shown. The right part shows clearly how the high frequencies are softened by the filter.

Example 3.19: Computing a composite filter

Assume that the filters S_1 and S_2 have the frequency responses $\lambda_{S_1}(\omega) = \cos(2\omega)$, $\lambda_{S_2}(\omega) = 1 + 3\cos\omega$. Let us see how we can use Theorem 3.18 to compute the filter coefficients and the matrix of the filter $S = S_1 S_2$. We first notice that, since both frequency responses are real, all S_1, S_2, and $S = S_1 S_2$ are symmetric. We rewrite the frequency responses as

$$\lambda_{S_1}(\omega) = \frac{1}{2}(e^{2i\omega} + e^{-2i\omega}) = \frac{1}{2}e^{2i\omega} + \frac{1}{2}e^{-2i\omega}$$
$$\lambda_{S_2}(\omega) = 1 + \frac{3}{2}(e^{i\omega} + e^{-i\omega}) = \frac{3}{2}e^{i\omega} + 1 + \frac{3}{2}e^{-i\omega}.$$

We now get that

$$\begin{aligned}\lambda_{S_1S_2}(\omega) = \lambda_{S_1}(\omega)\lambda_{S_2}(\omega) &= \left(\frac{1}{2}e^{2i\omega} + \frac{1}{2}e^{-2i\omega}\right)\left(\frac{3}{2}e^{i\omega} + 1 + \frac{3}{2}e^{-i\omega}\right) \\ &= \frac{3}{4}e^{3i\omega} + \frac{1}{2}e^{2i\omega} + \frac{3}{4}e^{i\omega} + \frac{3}{4}e^{-i\omega} + \frac{1}{2}e^{-2i\omega} + \frac{3}{4}e^{-3i\omega}\end{aligned}$$

From this expression we see that the filter coefficients of S are $t_{\pm 1} = 3/4$, $t_{\pm 2} = 1/2$, $t_{\pm 3} = 3/4$. All other filter coefficients are 0. Using Theorem 3.2, we get that $s_1 = 3/4$, $s_2 = 1/2$, and $s_3 = 3/4$, while $s_{N-1} = 3/4$, $s_{N-2} = 1/2$, and $s_{N-3} = 3/4$ (all other s_k are 0). This gives us the matrix representation of S.

3.3.1 Windowing operations

In this section we will take a look at a very important, and perhaps surprising, application of the continuous frequency response. Let us return to the computations from Example 2.16. There we saw that, when we restricted to a block of the signal, this affected the frequency representation. If we substitute with the angular frequencies $\omega = 2\pi n/N$ and $\omega_0 = 2\pi n_0/M$ in Equation (2.12), we get

$$y_n = \frac{1}{N}\sum_{k=0}^{N-1} e^{ik\omega_0}e^{-ik\omega} = \frac{1}{N}\sum_{k=0}^{N-1} e^{-ik(\omega-\omega_0)}$$

(here y_n were the DFT components of the sound after we had restricted to a block). This expression states that, when we restrict to a block of length N in the signal by discarding the other samples, a pure tone of angular frequency ω_0 suddenly gets a frequency contribution at angular frequency ω also, and the contribution is given by this formula. The expression is seen to be the same as the frequency response of the filter $\frac{1}{N}\{\underline{1}, 1, \ldots, 1\}$ (where 1 is repeated N times), evaluated at $\omega - \omega_0$. This filter is nothing but a (delayed) moving average filter. The frequency response of a moving average filter thus governs how the different frequencies pollute when we limit ourselves to a block of the signal. Since this

frequency response has its peak at 0, angular frequencies ω close to ω_0 have biggest values, so that the pollution is mostly from frequencies close to ω_0. But unfortunately, other frequencies also pollute.

One can also ask the question if there are better ways to restrict to a block of size N of the signal. We formulate the following idea.

Idea 3.19. *Windows.*

Let $\boldsymbol{x} = (x_0, \ldots, x_M)$ be a sound of length M. We would like to find values $\boldsymbol{w} = \{\underline{w_0}, \ldots, w_{N-1}\}$ so that the new sound $(w_0 x_0, \ldots, w_{N-1} x_{N-1})$ of length $N < M$ has a frequency representation similar to that of $\boldsymbol{x}$. $\boldsymbol{w}$ is called a *window* of length N, and the new sound is called the *windowed signal*.

Above we encountered the window $\boldsymbol{w} = \{\underline{1}, 1, \ldots, 1\}$. This is called the rectangular window. To see how we can find a good window, note first that the DFT values in the windowed signal of length N is

$$y_n = \frac{1}{N} \sum_{k=0}^{N-1} w_k e^{ik\omega_0} e^{-ik\omega} = \frac{1}{N} \sum_{k=0}^{N-1} w_k e^{-ik(\omega-\omega_0)}.$$

This is the frequency response of $\frac{1}{N}\boldsymbol{w}$. In order to limit the pollution from other frequencies, we thus need to construct a window with a frequency response with smaller values than that of the rectangular window away from 0. Let us summarize our findings as follows:

Observation 3.20. *Constructing a window.*

Assume that we would like to construct a window of length N. It is desirable that the frequency response of the window has small values away from zero.

We will not go into techniques for how such frequency responses can be constructed, but only consider one example different from the rectangular window. We define the Hamming window by

$$w_n = 2(0.54 - 0.46\cos(2\pi n/(N-1))). \tag{3.13}$$

The frequency responses of the rectangular window and the Hamming window are compared in Figure 3.5 for $N = 32$.

We see that the Hamming window has much smaller values away from 0, so that it is better suited as a window. However, the width of the "main lobe" (i.e. the main structure at the center), seems to be bigger. The window coefficients themselves are shown in Figure 3.6. It is seen that the frequency response of the Hamming window attenuates more and more as we get close to the boundaries.

Many other windows are used in the literature. The concrete window from Exercise 3.29 is for instance used in the MP3 standard. It is applied to the sound, and after this an FFT is applied to the windowed sound in order to make a frequency analysis of that part of the sound. The effect of the window is that there is smaller loss in the frequency representation of the sound when we restrict to a block of sound samples. This is a very important part of the psychoacoustic model used in the MP3 encoder, since it has to make compression decisions based on the frequency information in the sound.

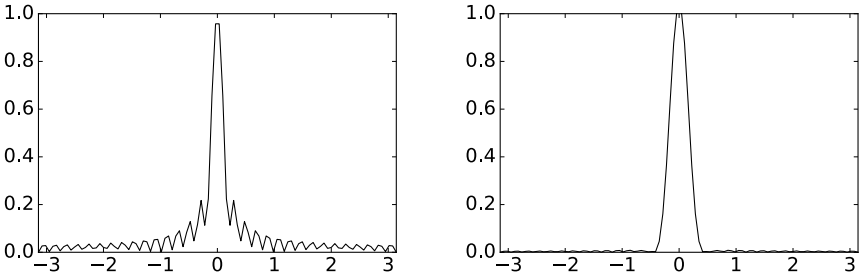

Figure 3.5: The frequency responses of the rectangular and Hamming windows, which we considered for restricting to a block of the signal.

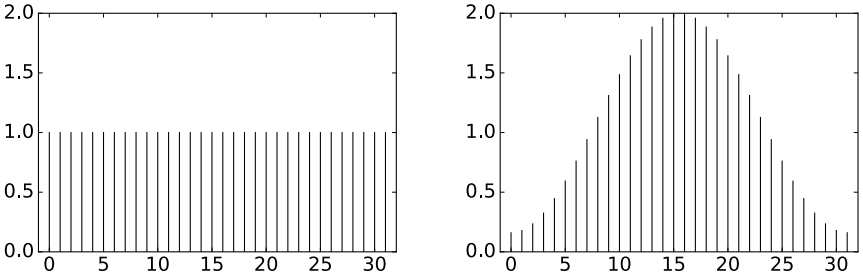

Figure 3.6: The coefficients of the rectangular and Hamming windows, which we considered for restricting to a block of the signal.

Exercise 3.20: Plotting a simple frequency response

Let again S be the filter defined by the equation

$$z_n = \frac{1}{4}x_{n+1} + \frac{1}{4}x_n + \frac{1}{4}x_{n-1} + \frac{1}{4}x_{n-2},$$

as in Exercise 3.4. Compute and plot (the magnitude of) $\lambda_S(\omega)$.

Exercise 3.21: Low-pass and high-pass filters

A filter S is defined by the equation

$$z_n = \frac{1}{3}(x_n + 3x_{n-1} + 3x_{n-2} + x_{n-3}).$$

a) Compute and plot the (magnitude of the continuous) frequency response of the filter, i.e. $|\lambda_S(\omega)|$. Is the filter a low-pass filter or a high-pass filter?

b) Find an expression for the vector frequency response $\lambda_{S,2}$. What is $S\boldsymbol{x}$ when $\boldsymbol{x}$ is the vector of length N with components $e^{2\pi i 2k/N}$?

Exercise 3.22: Circulant matrices

A filter S_1 is defined by the equation

$$z_n = \frac{1}{16}(x_{n+2} + 4x_{n+1} + 6x_n + 4x_{n-1} + x_{n-2}).$$

a) Write down an 8×8 circulant Toeplitz matrix which corresponds to applying S_1 on a periodic signal with period $N = 8$.

b) Compute and plot (the continuous) frequency response of the filter. Is the filter a low-pass filter or a high-pass filter?

c) Another filter S_2 has (continuous) frequency response $\lambda_{S_2}(\omega) = (e^{i\omega} + 2 + e^{-i\omega})/4$. Write down the filter coefficients for the filter S_1S_2.

Exercise 3.23: Composite filters

Assume that the filters S_1 and S_2 have the frequency responses $\lambda_{S_1}(\omega) = 2 + 4\cos(\omega)$, $\lambda_{S_2}(\omega) = 3\sin(2\omega)$.

a) Compute and plot the frequency response of the filter S_1S_2.

b) Write down the filter coefficients t_k and the impulse response $\boldsymbol{s}$ for the filter S_1S_2.

Exercise 3.24: Maximum and minimum

Compute and plot the continuous frequency response of the filter $S = \{1/4, \underline{1/2}, 1/4\}$. Where does the frequency response achieve its maximum and minimum value, and what are these values?

Exercise 3.25: Plotting a simple frequency response

Plot the continuous frequency response of the filter $T = \{1/4, \underline{-1/2}, 1/4\}$. Where does the frequency response achieve its maximum and minimum value, and what are these values? Can you write down a connection between this frequency response and that from Exercise 3.24?

Exercise 3.26: Continuous- and vector frequency responses

Define the filter S by $S = \{1, 2, \underline{3}, 4, 5, 6\}$. Write down the matrix for S when $N = 8$. Plot (the magnitude of) $\lambda_S(\omega)$, and indicate the values $\lambda_{S,n}$ for $N = 8$ in this plot.

Exercise 3.27: Starting with circulant matrices

Given the circulant Toeplitz matrix

$$S = \frac{1}{5}\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 1 & 1 & \cdots & 0 \\ 0 & 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 1 \\ 1 & 1 & 0 & \cdots & 1 \\ 1 & 1 & 1 & \cdots & 1 \end{pmatrix}$$

Write down the compact notation for this filter. Compute and plot (the magnitude) of $\lambda_S(\omega)$.

Exercise 3.28: When the filter coefficients are powers

Assume that $S = \{\underline{1}, c, c^2, \ldots, c^k\}$. Compute and plot $\lambda_S(\omega)$ when $k = 4$ and $k = 8$. How does the choice of k influence the frequency response? How does the choice of c influence the frequency response?

Exercise 3.29: The Hanning window

The Hanning window is defined by $w_n = 1-\cos(2\pi n/(N-1))$. Compute and plot the window coefficients and the continuous frequency response of this window for $N = 32$, and compare with the window coefficients and the frequency responses for the rectangular- and the Hamming window.

3.4 Some examples of filters

We have now established the basic theory of filters, and it is time to study some specific examples. Some of the filters have the desirable property that they favor certain frequencies, while annihilating others. Such filters have their own names.

Definition 3.21. *Lowpass and high-pass filters.*
A filter S is called

- a *low-pass filter* if $\lambda_S(\omega)$ is large when ω is close to 0, and $\lambda_S(\omega) \approx 0$ when ω is close to π (i.e. S keeps low frequencies and annihilates high frequencies),
- a *high-pass filter* if $\lambda_S(\omega)$ is large when ω is close to π, and $\lambda_S(\omega) \approx 0$ when ω is close to 0 (i.e. S keeps high frequencies and annihilates low frequencies),
- a *bandpass filter* if $\lambda_S(\omega)$ is large within some interval $[a, b] \subset [0, 2\pi]$, and $\lambda_S(\omega) \approx 0$ outside this interval.

This definition should be considered rather vague when it comes to what we mean by "ω close to $0, \pi$", and "$\lambda_S(\omega)$ is large": in practice, when we talk about low-pass and high-pass filters, it may be that the frequency responses are still quite far from what is commonly refered to as *ideal low-pass or high-pass filters*, where the frequency response only assumes the values 0 and 1 near 0 and π.

One common application of low-pass filters is to reduce treble in sound, which is a common option in audio systems. The treble in a sound is generated by the fast oscillations (high frequencies) in the signal. Another option in audio systems is to reduce the bass. This corresponds to reducing the low frequencies in the sound, and high-pass filters are suitable for this purpose. It turns out that there is a simple way to jump between low-pass and high-pass filters:

Observation 3.22. *Passing between low-pass- and high-pass filters.*
Assume that S_2 is obtained by adding an alternating sign to the filter coefficients of S_1. If S_1 is a low-pass filter, then S_2 is a high-pass filter, and vice versa.

To see why this is the case, let S_1 be a filter with filter coefficients t_k, and let us consider the filter S_2 with filter coefficient $(-1)^k t_k$. The frequency response of S_2 is

$$\begin{aligned}\lambda_{S_2}(\omega) &= \sum_k (-1)^k t_k e^{-i\omega k} = \sum_k (e^{-i\pi})^k t_k e^{-i\omega k} \\ &= \sum_k e^{-i\pi k} t_k e^{-i\omega k} = \sum_k t_k e^{-i(\omega+\pi)k} = \lambda_{S_1}(\omega+\pi).\end{aligned}$$

where we have set $e^{-i\pi} = -1$ (note that this is nothing but Property 4. in Theorem 2.7, with $d = N/2$). Now, for a low-pass filter S_1, $\lambda_{S_1}(\omega)$ has large values when ω is close to 0 (the low frequencies), and values near 0 when ω is close to π (the high frequencies). For a high-pass filter S_2, $\lambda_{S_2}(\omega)$ has values near 0 when ω is close to 0 (the low frequencies), and large values when ω is close to π (the high frequencies). Therefore, the relation $\lambda_{S_2}(\omega) = \lambda_{S_1}(\omega+\pi)$ says that S_1 is low-pass when S_2 is high-pass, and vice versa.

Example 3.30: Adding echo

An echo is a copy of the sound that is delayed and softer than the original sound. If $\boldsymbol{x}$ is the sound, the sound $\boldsymbol{z}$ with samples given by

```
[N, nchannels] = size(x);
z = zeros(N, nchannels);
z(1:d,:) = x(1:d,:); % No echo at the beginning of the signal
z((d+1):N,:) = x((d+1):N,:)+c*x(1:(N-d),:); % Add echo
z = z/max(max(abs(z))); % Scale so that sound values are within [-1,1].
```

will include an echo of the original sound. This is an example of a filtering operation where each output element is constructed from two input elements.

`d` is an integer which represents the delay in samples. If you need a delay in t seconds, simply multiply this with the sample rate to obtain the delay d in samples (d needs to be rounded to the nearest integer). c is a constant called the damping factor. Since an echo is usually weaker than the original sound, we usually have that $c < 1$. If we choose $c > 1$, the echo will dominate in the sound.

You can listen to the sample file with echo added with $d = 10000$ and $c = 0.5$ here.

Using our compact filter notation, the filter which adds echo can be written as

$$S = \{\underline{1}, 0, \ldots, 0, c\},$$

where the damping factor c appears after the delay d. The frequency response of this is $\lambda_S(\omega) = 1 + ce^{-id\omega}$, which is not real, so that the filter is not symmetric.

In Figure 3.7 we have plotted the magnitude of this frequency response with $c = 0.1$ and $d = 10$.

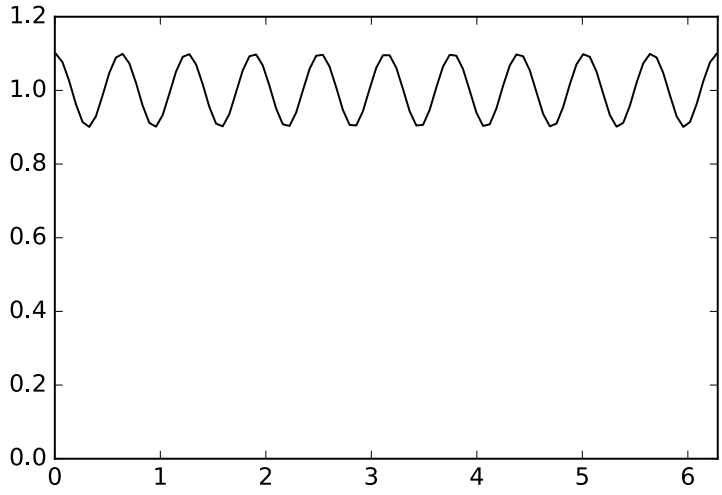

Figure 3.7: The frequency response of a filter which adds an echo with damping factor $c = 0.1$ and delay $d = 10$.

We see that the response varies between 0.9 and 1.1. The deviation from 1 is controlled by the damping factor c, and the oscillation is controlled by the delay d.

Example 3.31: Reducing the treble with moving average filters

Let us now take a look at filters which adjust the treble in sound. The fact that the filters are useful for these purposes will be clear when we plot the frequency response. A general way of reducing variations in a sequence of numbers is to replace one number by the average of itself and its neighbors, and this is easily done with a digital sound signal. If $\boldsymbol{z} = (z_i)_{i=0}^{N-1}$ is the sound signal produced by taking the average of three successive samples, we have that

$$z_n = \frac{1}{3}(x_{n+1} + x_n + x_{n-1}),$$

i.e. $S = \{1/3, \underline{1/3}, 1/3\}$. This filter is also called a *moving average filter* (with three elements), and it can be written in compact form as. If we set $N = 4$, the corresponding circulant Toeplitz matrix for the filter is

$$S = \frac{1}{3}\begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

The frequency response is

$$\lambda_S(\omega) = (e^{i\omega} + 1 + e^{-i\omega})/3 = (1 + 2\cos(\omega))/3.$$

More generally we can construct the moving average filter of $2L + 1$ elements, which is $S = \{1, \cdots, \underline{1}, \cdots, 1\}/(2L + 1)$, where there is symmetry around 0. Clearly then the first column of S is $\boldsymbol{s} = (\underbrace{1, \ldots, 1}_{L+1 \text{ times}}, 0, \ldots, 0, \underbrace{1, \ldots, 1}_{L \text{ times}})/(2L+1)$. In Example 2.2 we computed that the DFT of the vector $\boldsymbol{x} = (\underbrace{1, \ldots, 1}_{L+1 \text{ times}}, 0, \ldots, 0, \underbrace{1, \ldots, 1}_{L \text{ times}})$ had components

$$y_n = \frac{\sin(\pi n(2L + 1)/N)}{\sin(\pi n/N)}.$$

Since $\boldsymbol{s} = \boldsymbol{x}/(2L + 1)$ and $\lambda_S = \text{DFT}_N \boldsymbol{s}$, the frequency response of S is

$$\lambda_{S,n} = \frac{1}{2L + 1}\frac{\sin(\pi n(2L + 1)/N)}{\sin(\pi n/N)},$$

so that

$$\lambda_S(\omega) = \frac{1}{2L + 1}\frac{\sin((2L + 1)\omega/2)}{\sin(\omega/2)}.$$

We clearly have

$$0 \leq \frac{1}{2L + 1}\frac{\sin((2L + 1)\omega/2)}{\sin(\omega/2)} \leq 1,$$

and this frequency response approaches 1 as $\omega \to 0$. The frequency response thus peaks at 0, and this peak gets narrower and narrower as L increases, i.e. as

we use more and more samples in the averaging process. This filter thus "keeps" only the lowest frequencies. When it comes to the highest frequencies it is seen that the frequency response is small for $\omega \approx \pi$. In fact it is straightforward to see that $|\lambda_S(\pi)| = 1/(2L+1)$. In Figure 3.8 we have plotted the frequency response for moving average filters with $L = 1$, $L = 5$, and $L = 20$.

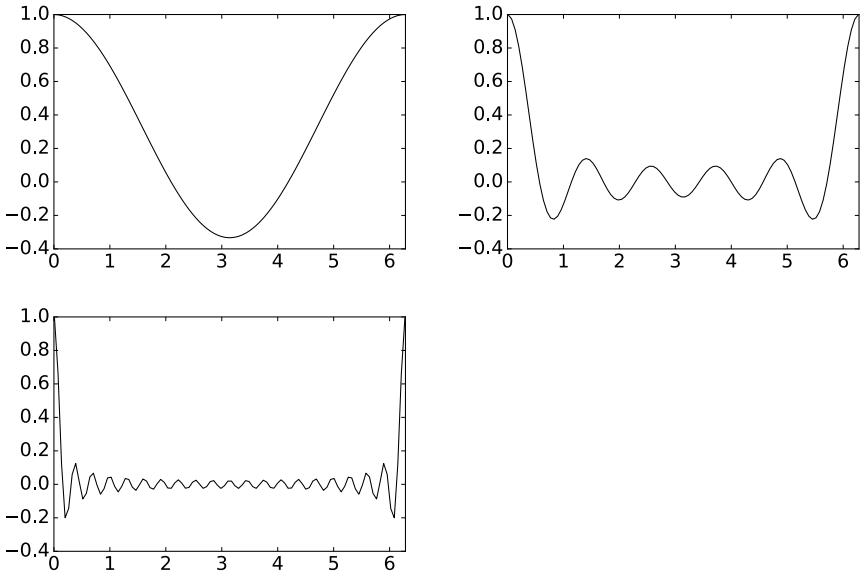

Figure 3.8: The frequency response of moving average filters with $L = 1$, $L = 5$, and $L = 20$.

Unfortunately, the frequency response is far from a filter which keeps some frequencies unaltered, while annihilating others: Although the filter distinguishes between high and low frequencies, it slightly changes the small frequencies. Moreover, the higher frequencies are not annihilated, even when we increase L to high values.

Example 3.32: Ideal low-pass filters

By definition, the ideal low-pass filter keeps frequencies near 0 unchanged, and completely removes frequencies near π. We now have the theory in place in order to find the filter coefficients for such a filter: In Example 2.16 we implemented the ideal low-pass filter with the help of the DFT. Mathematically you can see that this code is equivalent to computing $(F_N)^H D F_N$ where D is the diagonal matrix with the entries $0, \ldots, L$ and $N-L, \ldots, N-1$ being 1, the rest being 0. Clearly this is a digital filter, with frequency response as stated. If the filter should keep the angular frequencies $|\omega| \leq \omega_c$ only, where ω_c describes the highest frequency we should keep, we should choose L so that $\omega_c = 2\pi L/N$. Again, in Example 2.2

we computed the DFT of this vector, and it followed from Theorem 2.7 that the IDFT of this vector equals its DFT. This means that we can find the filter coefficients by using Equation (3.10), i.e. we take an IDFT. We then get the filter coefficients

$$\frac{1}{N}\frac{\sin(\pi k(2L+1)/N)}{\sin(\pi k/N)}.$$

This means that the filter coefficients lie as N points uniformly spaced on the curve $\frac{1}{N}\frac{\sin(\pi t(2L+1)/2)}{\sin(\pi t/2)}$ between 0 and 1. This curve has been encountered many other places in these notes. The filter which keeps only the frequency $\omega_c = 0$ has all filter coefficients being $\frac{1}{N}$ (set $L = 1$), and when we include all frequencies (set $L = N$) we get the filter where $x_0 = 1$ and all other filter coefficients are 0. When we are between these two cases, we get a filter where s_0 is the biggest coefficient, while the others decrease towards 0 along the curve we have computed. The bigger L and N are, the quicker they decrease to zero. All filter coefficients are usually nonzero for this filter, since this curve is zero only at certain points. This is unfortunate, since it means that the filter is time-consuming to compute.

The two previous examples show an important duality between vectors which are 1 on some elements and 0 on others (also called window vectors), and the vector $\frac{1}{N}\frac{\sin(\pi k(2L+1)/N)}{\sin(\pi k/N)}$ (also called a sinc): filters of the one type correspond to frequency responses of the other type, and vice versa. The examples also show that, in some cases only the filter coefficients are known, while in other cases only the frequency response is known. In any case we can deduce the one from the other, and both cases are important.

Filters are much more efficient when there are few nonzero filter coefficients. In this respect the second example displays a problem: in order to create filters with particularly nice properties (such as being an ideal low-pass filter), one may need to sacrifice computational complexity by increasing the number of nonzero filter coefficients. The trade-off between computational complexity and desirable filter properties is a very important issue in *filter design theory*.

Example 3.33: Dropping filter coefficients

In order to decrease the computational complexity for the ideal low-pass filter in Example 3.32, one can for instance include only the first filter coefficients, i.e.

$$\left\{\frac{1}{N}\frac{\sin(\pi k(2L+1)/N)}{\sin(\pi k/N)}\right\}_{k=-N_0}^{N_0}.$$

Hopefully this gives us a filter where the frequency response is not that different from the ideal low-pass filter. Let us set $N = 128$, $L = 32$, so that the filter removes all frequencies $\omega > \pi/2$. In Figure 3.9 we show the corresponding frequency responses. N_0 has been chosen so that the given percentage of all coefficients are included.

This shows that we should be careful when we omit filter coefficients: if we drop too many, the frequency response is far away from that of an ideal bandpass

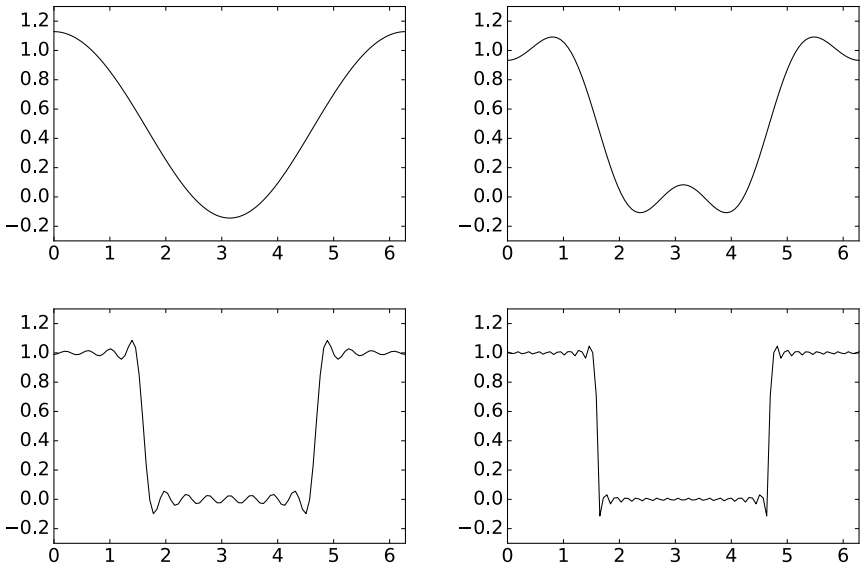

Figure 3.9: The frequency response which results by including the first 1/32, the first 1/16, the first 1/4, and and all of the filter coefficients for the ideal low-pass filter.

filter. In particular, we see that the new frequency response oscillates wildly near the discontinuity of the ideal low-pass filter. Such oscillations are called *Gibbs oscillations*.

Example 3.34: Filters and the MP3 standard

We mentioned previously that the MP3 standard splits the sound into frequency bands. This splitting is actually performed by particular filters, which we will consider now. In the example above, we saw that when we dropped the last filter coefficients in the ideal low-pass filter, there were some undesired effects in the frequency response of the resulting filter. Are there other and better approximations to the ideal low-pass filter which uses the same number of filter coefficients? This question is important, since the ear is sensitive to certain frequencies, and we would like to extract these frequencies for special processing, using as low computational complexity as possible. In the MP3-standard, such filters have been constructed. These filters are more advanced than the ones we have seen up to now. They have as many as 512 filter coefficients! We will not go into the details on how these filters are constructed, but only show how their frequency responses look.

In the left plot in Figure 3.10, the "prototype filter" used in the MP3 standard is shown. We see that this is very close to an ideal low-pass filter. Moreover,

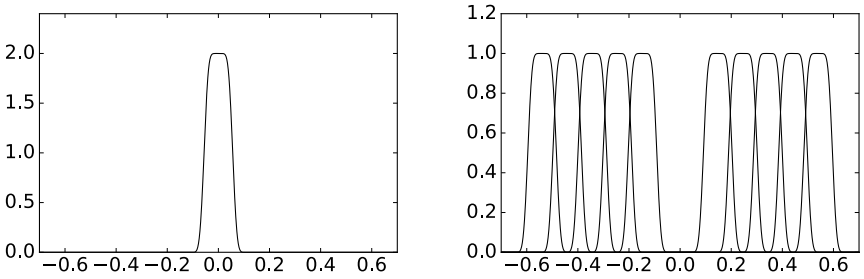

Figure 3.10: Frequency responses of some filters used in the MP3 standard. The prototype filter is shown left. The other frequency responses at right are simply shifted copies of this.

many of the undesirable effect from the previous example have been eliminated: The oscillations near the discontinuities are much smaller, and the values are lower away from 0. Using Property 4 in Theorem 2.7, it is straightforward to construct filters with similar frequency responses, but centered around different frequencies: We simply need to multiply the filter coefficients with a complex exponential, in order to obtain a filter where the frequency response has been shifted to the left or right. In the MP3 standard, this observation is used to construct 32 filters, each having a frequency response which is a shifted copy of that of the prototype filter, so that all filters together cover the entire frequency range. 5 of these frequency responses are shown in the right plot in Figure 3.10. To understand the effects of the different filters, let us apply them to our sample sound. If you apply all filters in the MP3 standard in successive order with the most low-pass filters first, the result will sound like this. You should interpret the result as low frequencies first, followed by the high frequencies. π corresponds to the frequency 22.05KHz (i.e. the highest representable frequency equals half the sampling rate on 44.1KHz. The different filters are concentrated on 1/32 of these frequencies each, so that the angular frequencies you here are $[\pi/64, 3\pi/64]$, $[3\pi/64, 5\pi/64]$, $[5\pi/64, 7\pi/64]$, and so on, in that order.

In Section 3.3.1 we mentioned that the psycoacoustic model of the MP3 standard applied a window the the sound data, followed by an FFT to that data. This is actually performed in parallel on the same sound data. Applying two different operations in parallel to the sound data may seem strange. In the MP3 standard [6] (p. 109) this is explained by "the lack of spectral selectivity obtained at low frequencies" by the filters above. In other words, the FFT can give more precise frequency information than the filters can. This more precise information is then used to compute psychoacoustic information such as masking thresholds, and this information is applied to the output of the filters.

Example 3.35: Reducing the treble using Pascals triangle

When reducing the treble it is reasonable to let the middle sample x_i count more than the neighbors in the average, so an alternative is to compute the average by instead writing

$$z_n = (x_{n-1} + 2x_n + x_{n+1})/4$$

The coefficients $1, 2, 1$ here have been taken from row 2 in Pascal's triangle. It turns out that this is a good choice of coefficients. Also if we take averages of more numbers it will turn out that higher rows of Pascals triangle are good choices. Let us take a look at why this is the case. Let S be the moving average filter of two elements, i.e.

$$(S\boldsymbol{x})_n = \frac{1}{2}(x_{n-1} + x_n).$$

In Example 3.31 we had an odd number of filter coefficients. Here we have only two. We see that the frequency response in this case is

$$\lambda_S(\omega) = \frac{1}{2}(1 + e^{-i\omega}) = e^{-i\omega/2}\cos(\omega/2).$$

The frequency response is complex now, since the filter is not symmetric in this case. Let us now apply this filter k times, and denote by S_k the resulting filter. Theorem 3.18 gives us that the frequency response of S_k is

$$\lambda_{S^k}(\omega) = \frac{1}{2^k}(1 + e^{-i\omega})^k = e^{-ik\omega/2}\cos^k(\omega/2),$$

which is a polynomial in $e^{-i\omega}$ with the coefficients taken from Pascal's triangle (remember that the values in Pascals triangle are the coefficients of x in the expression $(1 + x)^k$, i.e. the binomial coefficients $\binom{k}{r}$ for $0 \leq r \leq k$). At least, this partially explains how filters with coefficients taken from Pascal's triangle appear. The reason why these are more desirable than moving average filters, and are used much for smoothing abrupt changes in images and in sound, is the following: Since $(1+e^{-i\omega})^k$ is a factor in $\lambda_{S^k}(\omega)$, it has a zero of multiplicity of k at $\omega = \pi$. In other words, when k is large, λ_{S^k} has a zero of high multiplicity at $\omega = \pi$, and this implies that the frequency response is very flat for $\omega \approx \pi$ when k increases, i.e. the filter is good at removing the highest frequencies. This can be seen in Figure 3.11, where we have plotted the magnitude of the frequency response when $k = 5$, and when $k = 30$. Clearly the latter frequency response is much flatter for $\omega \approx \pi$. On the other side, it is easy to show that the moving average filters of Example 3.31 had a zero of multiplicity one at $\omega = \pi$, regardless of L. Clearly, the corresponding frequency responses, shown in Figure 3.8, were not as flat for $\omega \approx \pi$, when compared to the ones in Figure 3.11.

While using S^k gives a desirable behaviour for $\omega \approx \pi$, we see that the behaviour is not so desirable for small frequencies $\omega \approx 0$: Only frequencies very close to 0 are kept unaltered. It should be possible to produce better low-pass

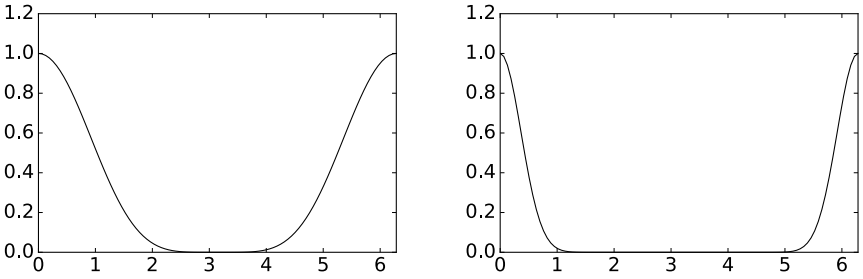

Figure 3.11: The frequency response of filters corresponding to iterating the moving average filter $\{\underline{1/2}, 1/2\}$ $k = 5$ and $k = 30$ times (i.e. using row k in Pascal's triangle).

filters than this also, and the frequency responses we plotted for the filters used in the MP3 standard gives an indication to this.

Let us now see how to implement the filters S^k. Since convolution corresponds to multiplication of polynomials, we can obtain their filter coefficients with the following code

```
t = [1];
for kval=1:k
    t = conv(t,[1/2 1/2]);
end
```

Note that S^k has $k+1$ filter coefficients, and that S^k corresponds to the filter coefficients of a symmetric filter when k is even. Having computed `t`, we can simply compute the convolution of the input `x` and `t`. In using `conv` we disregard the circularity of S, and we introduce a time delay. These issues will, however, not be audible when we listen to the output. An example of the result of smoothing is shown in Figure 3.12.

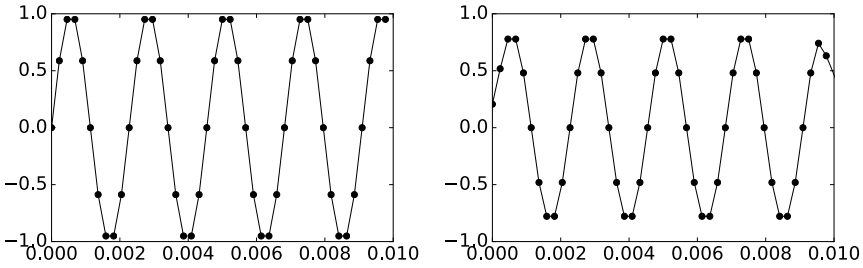

Figure 3.12: Reducing the treble. The original sound signal is shown left, the result after filtering using row 4 in Pascal's triangle is shown right.

The left plot shows the samples of the pure sound with frequency 440Hz (with sampling frequency $f_s = 4400$Hz). The right plot shows the result of applying the

averaging process by using row 4 of Pascals triangle. We see that the oscillations have been reduced. In Exercise 3.39 you will be asked to implement reducing the treble in our sample audio file. If you do this you should hear that the sound gets softer when you increase k: For $k = 32$ the sound will be like this, for $k = 256$ it will be like this.

Example 3.36: Reducing the bass using Pascals triangle

Due to Observation 3.22 and Example 3.35, we can create bass-reducing filters by adding an alternating sign to rows in Pascals triangle. Consider the bass-reducing filter deduced from the fourth row in Pascals triangle:

$$z_n = \frac{1}{16}(x_{n-2} - 4x_{n-1} + 6x_n - 4x_{n+1} + x_{n+2})$$

Let us apply this filter to the sound in Figure 3.12. The result is shown in Figure 3.13.

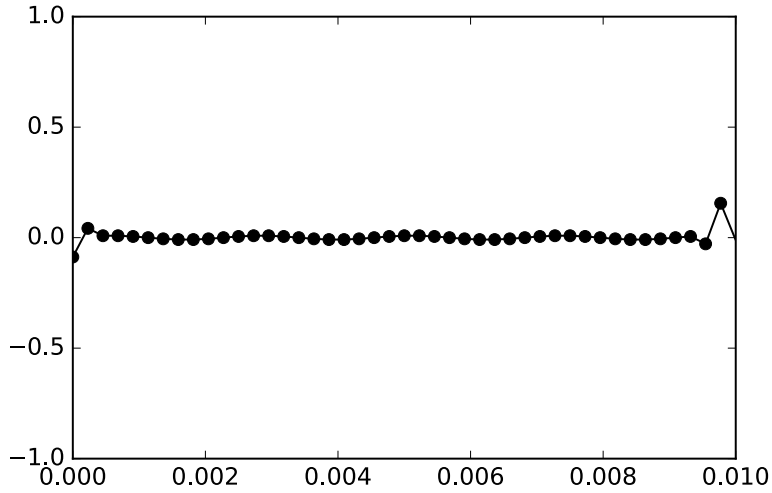

Figure 3.13: The result of applying the bass-reducing filter deduced from row 4 in Pascals triangle to the pure sound in the left plot of Figure 3.12.

We observe that the samples oscillate much more than the samples of the original sound. In Exercise 3.39 you will be asked to implement reducing the bass in our sample audio file. The new sound will be difficult to hear for large k, and we will explain why later. For $k = 1$ the sound will be like this, for $k = 2$ it will be like this. Even if the sound is quite low, you can hear that more of the bass has disappeared for $k = 2$.

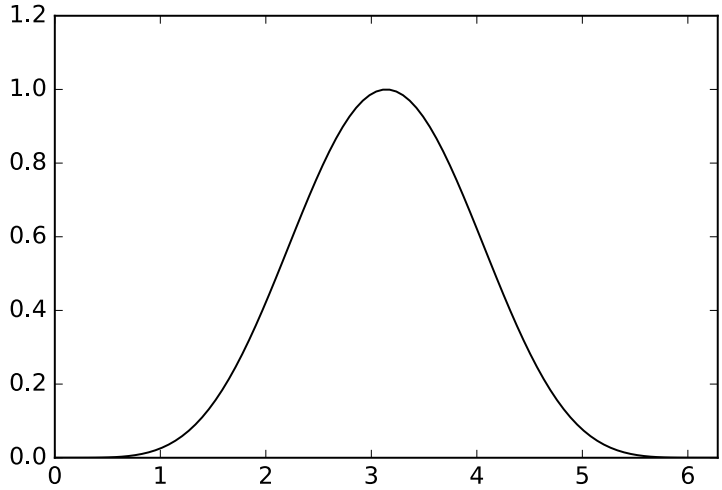

Figure 3.14: The frequency response of the bass reducing filter, which corresponds to row 5 of Pascal's triangle.

The frequency response we obtain from using row 5 of Pascal's triangle is shown in Figure 3.14. It is just the frequency response of the corresponding treble-reducing filter shifted with π. The alternating sign can also be achieved if we write the frequency response $\frac{1}{2^k}(1+e^{-i\omega})^k$ from Example 3.35 as $\frac{1}{2^k}(1-e^{-i\omega})^k$, which corresponds to applying the filter $S(\boldsymbol{x}) = \frac{1}{2}(-x_{n-1} + x_n)$ k times.

Exercise 3.37: Composing time delay filters

Let E_{d_1} and E_{d_2} be two time delay filters. Show that $E_{d_1}E_{d_2} = E_{d_1+d_2}$ (i.e. that the composition of two time delays again is a time delay) in two different ways:

a) Give a direct argument which uses no computations.

b) By using Property 3 in Theorem 2.7, i.e. by using a property for the Discrete Fourier Transform.

Exercise 3.38: Adding echo filters

Consider the two filters $S_1 = \{\underline{1}, 0, \ldots, 0, c\}$ and $S_2 = \{\underline{1}, 0, \ldots, 0, -c\}$. Both of these can be interpreted as filters which add an echo. Show that $\frac{1}{2}(S_1 + S_2) = I$. What is the interpretation of this relation in terms of echos?

Exercise 3.39: Reducing bass and treble

Write code where you reduce the treble and the bass as described in examples 3.35 and 3.36, generate the sounds you heard in these examples, and verify that they are the same. In your code, it will be necessary to scale the values after reducing the bass, but not the values after reducing the bass. Explain why this is the case. How high must k be in order for you to hear difference from the actual sound? How high can you choose k and still recognize the sound at all? If you solved Exercise 3.9, you can also use the function `filterS` to perform the filtering, rather than sing the `conv` function (the latter disregards circularity).

Exercise 3.40: Constructing a high-pass filter

Consider again Example 3.32. Find an expression for a filter so that only frequencies so that $|\omega - \pi| < \omega_c$ are kept, i.e. the filter should only keep angular frequencies close to π (i.e. here we construct a high-pass filter).

Exercise 3.41: Combining low-pass and high-pass filters

In this exercise we will investigate how we can combine low-pass and high-pass filters to produce other filters

a) Assume that S_1 and $S2$ are low-pass filters. What kind of filter is S_1S_2? What if both S_1 and S_2 are high-pass filters?

b) Assume that one of S_1, S_2 is a high-pass filter, and that the other is a low-pass filter. What kind of filter S_1S_2 in this case?

Exercise 3.42: Composing filters

A filter S_1 has the frequency response $\frac{1}{2}(1 + \cos\omega)$, and another filter has the frequency response $\frac{1}{2}(1 + \cos(2\omega))$.

a) Is S_1S_2 a low-pass filter, or a high-pass filter?

b) What does the filter S_1S_2 do with angular frequencies close to $\omega = \pi/2$.

c) Find the filter coefficients of S_1S_2.

Hint. Use Theorem 3.18 to compute the frequency response of S_1S_2 first.

d) Write down the matrix of the filter S_1S_2 for $N = 8$.

Exercise 3.43: Composing filters

An operation describing some transfer of data in a system is defined as the composition of the following three filters:

- First a time delay filter with delay $d_1 = 2$, due to internal transfer of data in the system,

- then the treble-reducing filter $T = \{1/4, \underline{1/2}, 1/4\}$,
- finally a time delay filter with delay $d_2 = 4$ due to internal transfer of the filtered data.

We denote by $T_2 = E_{d_2} T E_{d_1} = E_4 T E_2$ the operation which applies these filters in succession.

a) Explain why T_2 also is a digital filter. What is (the magnitude of) the frequency response of E_{d_1}? What is the connection between (the magnitude of) the frequency response of T and T_2?

b) Show that $T_2 = \{\underline{0}, 0, 0, 0, 0, 1/4, 1/2, 1/4\}$.

Hint. Use the expressions $(E_{d_1}\boldsymbol{x})_n = x_{n-d_1}$, $(T\boldsymbol{x})_n = \frac{1}{4}x_{n+1} + \frac{1}{2}x_n + \frac{1}{4}x_{n-1}$, $(E_{d_2}\boldsymbol{x})_n = x_{n-d_2}$, and compute first $(E_{d_1}\boldsymbol{x})_n$, then $(TE_{d_1}\boldsymbol{x})_n$, and finally $(T_2\boldsymbol{x})_n = (E_{d_2}TE_{d_1}\boldsymbol{x})_n$. From the last expression you should be able to read out the filter coefficients.

c) Assume that $N = 8$. Write down the 8×8-circulant Toeplitz matrix for the filter T_2.

Exercise 3.44: Filters in the MP3 standard

In Example 3.34, we mentioned that the filters used in the MP3-standard were constructed from a low-pass prototype filter by multiplying the filter coefficients with a complex exponential. Clearly this means that the new frequency response is a shift of the old one. The disadvantage is, however, that the new filter coefficients are complex. It is possible to address this problem as follows. Assume that t_k are the filter coefficients of a filter S_1, and that S_2 is the filter with filter coefficients $\cos(2\pi kn/N)t_k$, where $n \in \mathbb{N}$. Show that

$$\lambda_{S_2}(\omega) = \frac{1}{2}(\lambda_{S_1}(\omega - 2\pi n/N) + \lambda_{S_1}(\omega + 2\pi n/N)).$$

In other words, when we multiply (modulate) the filter coefficients with a cosine, the new frequency response can be obtained by shifting the old frequency response with $2\pi n/N$ in both directions, and taking the average of the two.

Exercise 3.45: Explain code

a) Explain what the code below does, line by line.

```
[x, fs] = audioread('sounds/castanets.wav');
[N, nchannels] = size(x);
z = zeros(N, nchannels);
for n=2:(N-1)
    z(n,:) = 2*x(n+1,:) + 4*x(n,:) + 2*x(n-1,:);
end
z(1,:) = 2*x(2,:) + 4*x(1,:) + 2*x(N,:);
z(N,:) = 2*x(1,:) + 4*x(N,:) + 2*x(N-1,:);
```

```
z = z/max(abs(z));
playerobj=audioplayer(z, fs);
playblocking(playerobj)
```

Comment in particular on what happens in the three lines directly after the `for`-loop, and why we do this. What kind of changes in the sound do you expect to hear?

b) Write down the compact filter notation for the filter which is used in the code, and write down a 5×5 circulant Toeplitz matrix which corresponds to this filter. Plot the (continuous) frequency response. Is the filter a low-pass- or a high-pass filter?

c) Another filter is given by the circulant Toeplitz matrix

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

Express a connection between the frequency responses of this filter and the filter from b). Is the new filter a low-pass- or a high-pass filter?

3.5 More general filters

The starting point for defining filters at the beginning of this chapter was equations on the form

$$z_n = \sum_k t_k x_{n-k}.$$

For most filters we have looked at, we had a limited number of nonzero t_k, and this enabled us to compute them on a computer using a finite number of additions and multiplications. Filters which have a finite number of nonzero filter coefficients are also called *FIR-filters* (FIR is short for Finite Impulse Response. Recall that the impulse response of a filter can be found from the filter coefficients). However, there exist many useful filters which are not FIR filters, i.e. where the sum above is infinite. The ideal lowpass filter from Example 3.32 was one example. It turns out that many such cases can be made computable if we change our procedure slightly. The old procedure for computing a filter is to compute $\boldsymbol{z} = S\boldsymbol{x}$. Consider the following alternative:

Idea 3.23. *More general filters (1).*
Let $\boldsymbol{x} \in \mathbb{R}^N$, and T an $N \times N$ filter. By solving the system $T\boldsymbol{z} = \boldsymbol{x}$ for $\boldsymbol{z}$ we get another filter, which we denote by S.

Of course T must then be the inverse of S (which also is a filter), but the point is that the inverse of a filter may have a finite number of filter coefficients, even

if the filter itself does not. In such cases this new procedure is more attractive that the old one, since the equation system can be solved with few arithmetic operations when T has few filter coefficients.

It turns out that there also are highly computable filters where neither the filter nor its inverse have a finite number of filter coefficients. Consider the following idea:

Idea 3.24. *More general filters (2).*

Let $\boldsymbol{x}$ be the input to a filter, and let U and V be filters. By solving the system $U\boldsymbol{z} = V\boldsymbol{x}$ for $\boldsymbol{z}$ we get another filter, which we denote by S. The filter S can be implemented in two steps: first we compute the right hand side $\boldsymbol{y} = V\boldsymbol{x}$, and then we solve the equation $U\boldsymbol{z} = \boldsymbol{y}$.

If both U and V are invertible we have that the filter is $S = U^{-1}V$, and this is invertible with inverse $S^{-1} = V^{-1}U$. The point is that, when U and V have a finite number of filter coefficients, both S and its inverse will typically have an infinite number of filter coefficients. The filters from this idea are thus more general than the ones from the previous idea, and the new idea makes a wider class of filters implementable using row reduction of sparse matrices. Computing a filter by solving $U\boldsymbol{z} = V\boldsymbol{x}$ may also give meaning when the matrices U and V are singular: The matrix system can have a solution even if U is singular. Therefore we should be careful in using the form $T = U^{-1}V$.

We have the following result concerning the frequency responses:

Theorem 3.25. *Frequency response of IIR filters.*

Assume that S is the filter defined from the equation $U\boldsymbol{z} = V\boldsymbol{x}$. Then we have that $\lambda_S(\omega) = \frac{\lambda_V(\omega)}{\lambda_U(\omega)}$ whenever $\lambda_U(\omega) \neq 0$.

Proof. Set $\boldsymbol{x} = \phi_n$. We have that $U\boldsymbol{z} = \lambda_{U,n}\lambda_{S,n}\phi_n$, and $V\boldsymbol{x} = \lambda_{V,n}\phi_n$. If the expressions are equal we must have that $\lambda_{U,n}\lambda_{S,n} = \lambda_{V,n}$, so that $\lambda_{S,n} = \frac{\lambda_{V,n}}{\lambda_{U,n}}$ for all n. By the definition of the continuous frequency response this means that $\lambda_S(\omega) = \frac{\lambda_V(\omega)}{\lambda_U(\omega)}$ whenever $\lambda_U(\omega) \neq 0$. □

The following example clarifies the points made above, and how one may construct U and V from S. The example also shows that, in addition to making some filters with infinitely many filter coefficients computable, the procedure $U\boldsymbol{z} = V\boldsymbol{x}$ for computing a filter can also reduce the complexity in some filters where we already have a finite number of filter coefficients.

Example 3.46: Moving average filter

Consider again the moving average filter S from Example 3.31:

$$z_n = \frac{1}{2L+1}(x_{n+L} + \cdots + x_n + \cdots + x_{n-L}).$$

If we implemented this directly, $2L$ additions would be needed for each n, so that we would need a total of $2NL$ additions. However, we can also write

$$\begin{aligned} z_{n+1} &= \frac{1}{2L+1}(x_{n+1+L} + \cdots + x_{n+1} + \cdots + x_{n+1-L}) \\ &= \frac{1}{2L+1}(x_{n+L} + \cdots + x_n + \cdots + x_{n-L}) + \frac{1}{2L+1}(x_{n+1+L} - x_{n-L}) \\ &= z_n + \frac{1}{2L+1}(x_{n+1+L} - x_{n-L}). \end{aligned}$$

This means that we can also compute the output from the formula

$$z_{n+1} - z_n = \frac{1}{2L+1}(x_{n+1+L} - x_{n-L}),$$

which can be written on the form $U\boldsymbol{z} = V\boldsymbol{x}$ with $U = \{1, \underline{-1}\}$ and $V = \frac{1}{2L+1}\{1, 0, \ldots, 0, -1\}$ where the 1 is placed at index $-L-1$ and the -1 is placed at index L. We now perform only $2N$ additions in computing the right hand side, and solving the equation system requires only $2(N-1)$ additions. The total number of additions is thus $2N + 2(N-1) = 4N - 2$, which is much less than the previous $2LN$ when L is large.

A perhaps easier way to find U and V is to consider the frequency response of the moving average filter, which is

$$\begin{aligned} \frac{1}{2L+1}(e^{-Li\omega} + \ldots + e^{Li\omega}) &= \frac{1}{2L+1}e^{-Li\omega}\frac{1 - e^{(2L+1)i\omega}}{1 - e^{i\omega}} \\ &= \frac{\frac{1}{2L+1}\left(-e^{(L+1)i\omega} + e^{-Li\omega}\right)}{1 - e^{i\omega}}, \end{aligned}$$

where we have used the formula for the sum of a geometric series. From here we easily see the frequency responses of U and V from the numerator and the denominator.

Filters with an infinite number of filter coefficients are also called *IIR filters* (IIR stands for *Infinite Impulse Response*). Thus, we have seen that some IIR filters may still have efficient implementations.

Exercise 3.47: A concrete IIR filter

A filter is defined by demanding that $z_{n+2} - z_{n+1} + z_n = x_{n+1} - x_n$.

a) Compute and plot the frequency response of the filter.

b) Use a computer to compute the output when the input vector is $\boldsymbol{x} = (1, 2, \ldots, 10)$. In order to do this you should write down two 10×10-circulant Toeplitz matrices.

3.6 Implementation of filters

As we saw in Example 3.46, a filter with many filter coefficients could be factored into the application of two simpler filters, and this could be used as a basis for an efficient implementation. There are also several other possible efficient implementations of filters. In this section we will consider two such techniques. The first technique considers how we can use the DFT to speed up the computation of filters. The second technique considers how we can factorize a filter into a product of simpler filters.

3.6.1 Implementation of filters using the DFT

If there are k filter coefficients, a direct implementation of a filter would require kN multiplications. Since filters are diagonalized by the DFT, one can also compute the filter as the product $S = F_N^H D F_N$. This would instead require $O(N \log_2 N)$ complex multiplications when we use the FFT algorithm, which may be a higher number of multiplications. We will however see that, by slightly changing our algorithm, we may end up with a DFT-based implementation of the filter which requires fewer multiplications.

The idea is to split the computation of the filter into smaller parts. Assume that we compute M elements of the filter at a time. If the nonzero filter coefficients of S are $t_{-k_0},\ldots,t_{k-k_0-1}$, we have that

$$(S\boldsymbol{x})_t = \sum_r t_r x_{s-r} = t_{-k_0}x_{t+k_0} + .. + t_{k-k_0-1}x_{t-(k-k_0-1)}.$$

From this it is clear that $(S\boldsymbol{x})_t$ only depends on $x_{t-(k-k_0-1)},\ldots,x_{t+k_0}$. This means that, if we restrict the computation of S to $x_{t-(k-k_0-1)},\ldots,x_{t+M-1+k_0}$, the outputs $x_t,\ldots,x_{t+M-1}$ will be the same as without this restriction. This means that we can compute the output M elements at a time, at each step multiplying with a circulant Toeplitz matrix of size $(M+k-1)\times(M+k-1)$. If we choose M so that $M+k-1 = 2^r$, we can use the FFT and IFFT algorithms to compute $S = F_N^H D F_N$, and we require $O(r2^r)$ multiplications for every block of length M. The total number of multiplications is $\frac{Nr2^r}{M} = \frac{Nr2^r}{2^r-k+1}$. If $k = 128$, you can check on your calculator that the smallest value is for $r = 10$ with value $11.4158 \times N$. Since the direct implementation gives kN multiplications, this clearly gives a benefit for the new approach, it gives a 90% decrease in the number of multiplications.

3.6.2 Factoring a filter

In practice, filters are often applied in hardware, and applied in real-time scenarios where performance is a major issue. The most CPU-intensive tasks in such applications often have few memory locations available. These tasks are thus not compatible with filters with many filter coefficients, since for each output sample we then need access to many input samples and filter coefficients. A strategy which addresses this is to factorize the filter into the product of several smaller

filters, and then applying each filter in turn. Since the frequency response of the product of filters equals the product of the frequency responses, we get the following idea:

Idea 3.26. *Factorizing a filter.*
Let S be a filter with real coefficients. Assume that

$$\lambda_S(\omega) = Ke^{ik\omega}(e^{i\omega} - a_1)\dots(e^{i\omega} - a_m)(e^{2i\omega} + b_1e^{i\omega} + c_1)\dots(e^{2i\omega} + b_ne^{i\omega} + c_n). \tag{3.14}$$

Then we can write $S = KE_kA_1\dots A_mB_1\dots B_n$, where $A_i = \{1, \underline{-a_i}\}$ and $B_i = \{1, b_i, \underline{c_i}\}$.

Note that in Equation (3.14) a_i correspond to the real roots of the frequency response, while b_i, c_i are obtained by pairing the complex conjugate roots. Clearly the frequency responses of A_i, B_i equal the factors in the frequency response of S, which in any case can be factored into the product of filters with 2 and 3 filter coefficients, followed by a time-delay.

Note that, even though this procedure factorizes a filter into smaller parts (which is attractive for hardware implementations since smaller filters require fewer locations in memory), the number of of arithmetic operations is usually not reduced. However, consider Example 3.35, where we factorized the treble-reducing filters into a product of moving average filters of length 2 (all roots in the previous idea are real, and equal). Each application of a moving average filter of length 2 does not really require any multiplications, since multiplication with $\frac{1}{2}$ corresponds to a bitshift. Therefore, the factorization of Example 3.35 removes the need for doing any multiplications at all, while keeping the number of additions the same. There are computational savings in this case, due to the special filter structure here.

Exercise 3.48: Implementing the factorization

Write a function `filterdftimpl`, which takes the filter coefficients $\boldsymbol{t}$ and the value k_0 from this section, computes the optimal M, and implements the filter as here.

Exercise 3.49: Factoring concrete filter

Factor the filter $S = \{\underline{1}, 5, 10, 6\}$ into a product of two filters, one with two filter coefficients, and one with three filter coefficients.

3.7 Summary

We defined digital filters, which do the same job for digital sound as analog filters do for (continuous) sound. Digital filters turned out to be linear transformations diagonalized by the DFT. We proved several other equivalent characterizations

of digital filters as well, such as being time-invariant, and having a matrix which is circulant and Toeplitz. Just as for continuous sound, digital filters are characterized by their frequency response, which explains how the filter treats the different frequencies. We also went through several important examples of filters, some of which corresponded to meaningful operations on sound, such as adjustmest of bass and treble, and adding echo. We also explained that there exist filters with useful implementations which have an infinite number of filter coefficients, and we considered techniques for implementing filters efficiently. Most of the topics covered on that can also be found in [11]. We also took a look at the role of filters in the MP3 standard for compression of sound.

In signal processing literature, the assumption that vectors are periodic is often not present, and filters are thus not defined as finite-dimensional operations. With matrix notation they would then be viewed as infinite matrices which have the Toeplitz structure (i.e. constant values on the diagonals), but with no circulation. The circulation in the matrices, as well as the restriction to finite vectors, come from the assumption of a periodic vector. There are, however, also some books which view filters as circulant Toeplits matrices as we have done, such as [4].

What you should have learned in this chapter.

- How to write down the circulant Toeplitz matrix from a digital filter expression, and vice versa.
- How to find the first column of this matrix ($\boldsymbol{s}$) from the filter coefficients ($\boldsymbol{t}$), and vice versa.
- The compact filter notation for filters with a finite number of filter coefficients.
- The definition of convolution, its connection with filters, and the `conv`-function for computing convolution.
- Connection between applying a filter and multiplying polynomials.
- The formal definition of a digital filter in terms of having the Fourier vectors as eigenvectors.
- The definition of the vector frequency response in terms of the corresponding eigenvalues.
- The definition of time-invariance and the three equivalent characterizations of a filter.
- For filters, eigenvalues can be computed by taking the DFT of the first column $\boldsymbol{s}$, and there is no need to compute eigenvectors explicitly.
- How to apply a digital filter to a sum of sines or cosines, by splitting these into a sum of eigenvectors.

- The definition of the continuous frequency response in terms of the filter coefficients $\boldsymbol{t}$.
- Connection with the vector frequency response.
- Properties of the continuous frequency response, in particular that the product of two frequency responses equals the frequency response of the product.
- How to compute the frequency response of the product of two filters,.
- How to find the filter coefficients when the continuous frequency response is known.
- Simple examples of filters, such as time delay filters and filters which add echo.
- Low-pass and high-pass filters and their frequency responses, and their interpretation as treble- and bass-reducing filters. Moving average filters, and filters arising from rows in Pascal's triangle, as examples of such filters.
- How to pass between low-pass and high-pass filters by adding an alternating sign to the filter coefficients.

Chapter 4

Symmetric filters and the DCT

In Chapter 1 we approximated a signal of finite duration with trigonometric functions. Since these are all periodic, there are some undesirable effects near the boundaries of the signal (at least when the values at the boundaries are different), and this resulted in a slowly converging Fourier series. This was addressed by instead considering the symmetric extension of the function, for which we obtained a more precise Fourier representation, as fewer Fourier basis vectors were needed in order to get a precise approximation.

This chapter is dedicated to addressing these thoughts for vectors. We will start by defining symmetric extensions of vectors, similarly to how we defined these for functions. Just as the Fourier series of a symmetric function was a cosine series, we will see that the symmetric extension can be viewed as a cosine vector. This gives rise to a different change of coordinates than the DFT, which we will call the DCT, which enables us to express a symmetric vector as a sum of cosine-vectors (instead of the non-symmetric complex exponentials). Since a cosine also can be associated with a given frequency, the DCT is otherwise similar to the DFT, in that it extracts the frequency information in the vector. The advantage is that the DCT can give more precise frequency information than the DFT, since it avoids the discontinuity problem of the Fourier series. This makes the DCT very practical for applications, and we will explain some of these applications. We will also show that the DCT has a a very efficient implementation, comparable with the FFT.

In this chapter we will also see that the DCT has a very similar role as the DFT when it comes to filters: just as the DFT diagonalized filters, we will see that symmetric filters can be diagonalized by the DCT, when we apply the filter to the symmetric extension of the input. We will actually show that the filters which preserve our symmetric extensions are exactly the symmetric filters.

4.1 Symmetric vectors and the DCT

As in Chapter 1, vectors can also be extended in a symmetric manner, besides the simple periodic extension procedure from Figure 2.1. In Figure 4.1 we have shown such an extension of a vector $\boldsymbol{x}$. It has $\boldsymbol{x}$ as its first half, and a copy of $\boldsymbol{x}$ in reverse order as its second half.

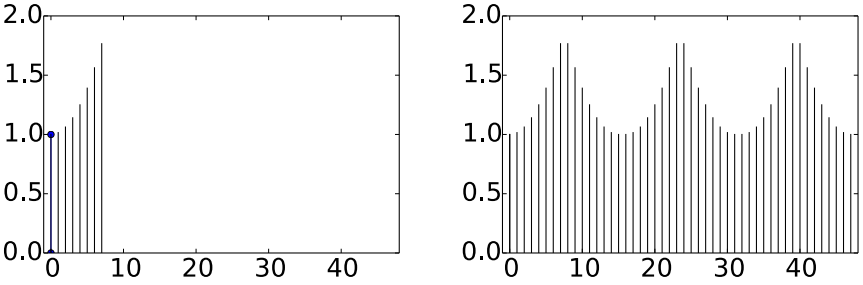

Figure 4.1: A vector and its symmetric extension.

We will call this the symmetric extension of $\boldsymbol{x}$:

Definition 4.1. *Symmetric extension of a vector.*

By the *symmetric extension* of $\boldsymbol{x} \in \mathbb{R}^N$, we mean the symmetric vector $\breve{\boldsymbol{x}} \in \mathbb{R}^{2N}$ defined by

$$\breve{\boldsymbol{x}}_k = \begin{cases} x_k & 0 \leq k < N \\ x_{2N-1-k} & N \leq k < 2N-1 \end{cases} \tag{4.1}$$

Clearly, the symmetric extension is symmetric around $N - 1/2$. This is not the only way to construct a symmetric extension, as we will return to later. As shown in Figure 4.1, but not included in Definition 4.1, we also repeat $\breve{\boldsymbol{x}} \in \mathbb{R}^{2N}$ in order to obtain a periodic vector. Creating a symmetric extension is thus a two-step process:

- First, "mirror" the vector to obtain a vector in $\mathbb{R}^{2N}$,
- repeat this periodically to obtain a periodic vector.

The result from the first step lies in an N-dimensional subspace of all vectors in $\mathbb{R}^{2N}$, which we will call *the space of symmetric vectors*. To account for the fact that a periodic vector can have a different symmetry point than $N - 1/2$, let us make the following general definition:

Definition 4.2. *Symmetric vector.*

We say that a periodic vector $\boldsymbol{x}$ is *symmetric* if there exists a number d so that $x_{d+k} = x_{d-k}$ for all k so that $d + k$ and $d - k$ are integers. d is called the *symmetry point* of $\boldsymbol{x}$

Due to the inherent periodicity of $\boldsymbol{x}$, it is clear that N must be an even number for symmetric vectors to exist at all. d can take any value, and it may not be an integer: It can also be an odd multiple of $1/2$, because then both $d+k$ and $d-k$ are integers when k also is an odd multiple of $1/2$. The symmetry point in symmetric extensions as defined in Definition 4.1 was $d = N - 1/2$. This is very common in the literature, and this is why we concentrate on this in this chapter. Later we will also consider symmetry around $N-1$, as this also is much used.

We would like to find a basis for the N-dimensional space of symmetric vectors, and we would like this basis to be similar to the Fourier basis. Since the Fourier basis corresponds to the standard basis in the frequency domain, we are lead to studying the DFT of a symmetric vector. If the symmetry point is an integer, it is straightforward to prove the following:

Theorem 4.3. *Symmetric vectors with integer symmetry points.*
Let d be an integer. The following are equivalent

- $\boldsymbol{x}$ is real and symmetric with d as symmetry point.
- $\left(\widehat{\boldsymbol{x}}\right)_n = z_n e^{-2\pi i d n/N}$ where z_n are real numbers so that $z_n = z_{N-n}$.

Proof. Assume first that $d = 0$. It follows in this case from property 2a) of Theorem 2.7 that $\left(\widehat{\boldsymbol{x}}\right)_n$ is a real vector. Combining this with property 1 of Theorem 2.7 we see that $\widehat{\boldsymbol{x}}$, just as $\boldsymbol{x}$, also must be a real vector symmetric about 0. Since the DFT is one-to-one, it follows that $\boldsymbol{x}$ is real and symmetric about 0 if and only if $\widehat{\boldsymbol{x}}$ is. From property 3 of Theorem 2.7it follows that, when d is an integer, $\boldsymbol{x}$ is real and symmetric about d if and only if $\left(\widehat{\boldsymbol{x}}\right)_n = z_n e^{-2\pi i d n/N}$, where z_n is real and symmetric about 0. This completes the proof. □

Symmetric extensions were here defined by having the non-integer symmetry point $N - 1/2$, however. For these we prove the following, which is slightly more difficult.

Theorem 4.4. *Symmetric vectors with non-integer symmetry points.*
Let d be an odd multiple of $1/2$. The following are equivalent

- $\boldsymbol{x}$ is real and symmetric with d as symmetry point.
- $\left(\widehat{\boldsymbol{x}}\right)_n = z_n e^{-2\pi i d n/N}$ where z_n are real numbers so that $z_{N-n} = -z_n$.

Proof. When $\boldsymbol{x}$ is as stated we can write

$$\begin{aligned}
(\widehat{\boldsymbol{x}})_n &= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} x_k e^{-2\pi i k n/N} \\
&= \frac{1}{\sqrt{N}} \left(\sum_{s\geq 0} x_{d+s} e^{-2\pi i (d+s) n/N} + \sum_{s\geq 0} x_{d-s} e^{-2\pi i (d-s) n/N} \right) \\
&= \frac{1}{\sqrt{N}} \sum_{s\geq 0} x_{d+s} \left(e^{-2\pi i (d+s) n/N} + e^{-2\pi i (d-s) n/N} \right) \\
&= \frac{1}{\sqrt{N}} e^{-2\pi i d n/N} \sum_{s\geq 0} x_{d+s} \left(e^{-2\pi i s n/N} + e^{2\pi i s n/N} \right) \\
&= \frac{1}{\sqrt{N}} e^{-2\pi i d n/N} \sum_{s\geq 0} 2x_{d+s} \cos(2\pi s n/N).
\end{aligned}$$

Here s runs through odd multiples of $1/2$. Since $z_n = \frac{1}{\sqrt{N}} \sum_{s\geq 0} 2x_{d+s} \cos(2\pi s n/N)$ is a real number, we can write the result as $z_n e^{-2\pi i d n/N}$. Substituting $N - n$ for n, we get

$$\begin{aligned}
(\widehat{\boldsymbol{x}})_{N-n} &= \frac{1}{\sqrt{N}} e^{-2\pi i d (N-n)/N} \sum_{s\geq 0} 2x_{d+s} \cos(2\pi s (N-n)/N) \\
&= \frac{1}{\sqrt{N}} e^{-2\pi i d (N-n)/N} \sum_{s\geq 0} 2x_{d+s} \cos(-2\pi s n/N + 2\pi s) \\
&= -\frac{1}{\sqrt{N}} e^{-2\pi i d (N-n)/N} \sum_{s\geq 0} 2x_{d+s} \cos(2\pi s n/N) = -z_n e^{-2\pi i d (N-n)/N}.
\end{aligned}$$

This shows that $z_{N-n} = -z_n$, and this completes one way of the proof. The other way, we can write

$$x_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} (\widehat{\boldsymbol{x}})_n \, e^{2\pi i k n/N}$$

if $(\widehat{\boldsymbol{x}})_n = z_n e^{-2\pi i d n/N}$ and $(\widehat{\boldsymbol{x}})_{N-n} = -z_n e^{-2\pi i d (N-n)/N}$, the sum of the n'th term and the $N - n$'th term in the sum is

$$\begin{aligned}
&z_n e^{-2\pi i d n/N} e^{2\pi i k n/N} - z_n e^{2-\pi i d (N-n)/N} e^{2\pi i k (N-n)/N} \\
&= z_n (e^{2\pi i (k-d) n/N} - e^{-2\pi i d + 2\pi i d n/N - 2\pi i k n/N}) \\
&= z_n (e^{2\pi i (k-d) n/N} + e^{2\pi i (d-k) n/N}) = 2 z_n \cos(2\pi (k-d) n/N).
\end{aligned}$$

This is real, so that all x_k are real. If we set $k = d + s$, $k = d - s$ here we get

$$2z_n \cos(2\pi((d+s)-d)n/N) = 2z_n \cos(2\pi sn/N)$$
$$2z_n \cos(2\pi((d-s)-d)n/N) = 2z_n \cos(-2\pi sn/N) = 2z_n \cos(2\pi sn/N).$$

By adding terms together and comparing we must have that $x_{d+s} = x_{d-s}$, and the proof is done. □

Now, let us specialize to symmetric extensions as defined in Definition 4.1, i.e. where $d = N - 1/2$. The following result gives us an orthonormal basis for the symmetric extensions, which are very simple in the frequency domain:

Theorem 4.5. *Orthonormal basis for symmetric vectors.*
The set of all $\boldsymbol{x}$ symmetric around $N - 1/2$ is a vector space of dimension N, and we have that

$$\left\{ \boldsymbol{e}_0, \left\{ \frac{1}{\sqrt{2}} \left(e^{\pi in/(2N)} \boldsymbol{e}_n + e^{-\pi in/(2N)} \boldsymbol{e}_{2N-n} \right) \right\}_{n=1}^{N-1} \right\}$$

is an orthonormal basis for $\widehat{\boldsymbol{x}}$ where $\boldsymbol{x}$ is symmetric around $N - 1/2$.

Proof. For a vector $\boldsymbol{x}$ symmetric about $d = N - 1/2$ we know that

$$(\widehat{\boldsymbol{x}})_n = z_n e^{-2\pi i(N-1/2)n/(2N)},$$

and the only requirement on the vector $\boldsymbol{z}$ is the antisymmetry condition $z_{2N-n} = -z_n$. The vectors $\boldsymbol{z}_i = \frac{1}{\sqrt{2}}(\boldsymbol{e}_i - \boldsymbol{e}_{2N-i})$, $1 \le i \le N-1$, together with the vector $\boldsymbol{z}_0 = \boldsymbol{e}_0$, are clearly orthonormal and satisifes the antisymmetry condition. From these we obtain that

$$\left\{ \boldsymbol{e}_0, \left\{ \frac{1}{\sqrt{2}} \left(e^{-2\pi i(N-1/2)n/(2N)} \boldsymbol{e}_n - e^{-2\pi i(N-1/2)(2N-n)/(2N)} \boldsymbol{e}_{2N-n} \right) \right\}_{n=1}^{N-1} \right\}$$

is an orthonormal basis for the $\widehat{\boldsymbol{x}}$ with $\boldsymbol{x}$ symmetric. We can write

$$\begin{aligned} &\frac{1}{\sqrt{2}} \left(e^{-2\pi i(N-1/2)n/(2N)} \boldsymbol{e}_n - e^{-2\pi i(N-1/2)(2N-n)/(2N)} \boldsymbol{e}_{2N-n} \right) \\ &= \frac{1}{\sqrt{2}} \left(e^{-\pi in} e^{\pi in/(2N)} \boldsymbol{e}_n + e^{\pi in} e^{-\pi in/(2N)} \boldsymbol{e}_{2N-n} \right) \\ &= \frac{1}{\sqrt{2}} e^{\pi in} \left(e^{\pi in/(2N)} \boldsymbol{e}_n + e^{-\pi in/(2N)} \boldsymbol{e}_{2N-n} \right). \end{aligned}$$

This also means that

$$\left\{ \boldsymbol{e}_0, \left\{ \frac{1}{\sqrt{2}} \left(e^{\pi in/(2N)} \boldsymbol{e}_n + e^{-\pi in/(2N)} \boldsymbol{e}_{2N-n} \right) \right\}_{n=1}^{N-1} \right\}$$

is an orthonormal basis. □

We immediately get the following result:

Theorem 4.6. *Orthonormal basis for symmetric vectors.*
We have that

$$\left\{ \frac{1}{\sqrt{2N}} \cos\left(2\pi \frac{0}{2N} \left(k + \frac{1}{2} \right) \right), \left\{ \frac{1}{\sqrt{N}} \cos\left(2\pi \frac{n}{2N} \left(k + \frac{1}{2} \right) \right) \right\}_{n=1}^{N-1} \right\} \quad (4.2)$$

is an orthonormal basis for the set of vectors symmetric around $N - 1/2$ in $\mathbb{R}^{2N}$. Moreover, the n'th vector in this basis has frequency contribution only from the indices n and $2N - n$.

Proof. Since the IDFT is unitary, the IDFT applied to the vectors above gives an orthonormal basis for the set of symmetric extensions. We get that

$$(F_{2N})^H(\boldsymbol{e}_0) = \left(\frac{1}{\sqrt{2N}}, \frac{1}{\sqrt{2N}}, \ldots, \frac{1}{\sqrt{2N}} \right) = \frac{1}{\sqrt{2N}} \cos\left(2\pi \frac{0}{2N} \left(k + \frac{1}{2} \right) \right).$$

We also get that

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

Since F_{2N} is unitary, and thus preserves the scalar product, the given vectors are orthonormal. □

We need to address one final thing before we can define the DCT: The vector $\boldsymbol{x}$ we start with is in $\mathbb{R}^N$, but the vectors above are in $\mathbb{R}^{2N}$. We would like to have orthonormal vectors in $\mathbb{R}^N$, so that we can use them to decompose $\boldsymbol{x}$. It is possible to show with a direct argument that, when we restrict the vectors above to the first N elements, they are still orthogonal. We will, however, apply a more instructive argument to show this, which gives us some intuition into the connection with symmetric filters. We start with the following result, which shows that a filter preserves symmetric vectors if and only if the filter is symmetric.

Theorem 4.7. *Criteria for preserving symmetric vectors.*
Let S be a filter. The following are equivalent

- S preserves symmetric vectors (i.e. $S\boldsymbol{x}$ is a symmetric vector whenever $\boldsymbol{x}$ is).
- The set of filter coefficients of S is a symmetric vector.

Also, when S preserves symmetric vectors, the following hold:

- The vector of filter coefficients has an integer symmetry point if and only if the input and output have the same type (integer or non-integer) of symmetry point.
- The input and output have the same symmetry point if and only if the filter is symmetric.

Proof. Assume that the filter S maps a symmetric vector with symmetry at d_1 to another symmetric vector. Let $\boldsymbol{x}$ be the symmetric vector so that $(\widehat{\boldsymbol{x}})_n = e^{-2\pi i d_1 n/N}$ for $n < N/2$. Since the output is a symmetric vector, we must have that

$$\lambda_{S,n} e^{-2\pi i d_1 n/N} = z_n e^{-2\pi i d_2 n/N}$$

for some d_2, z_n and for $n < N/2$. But this means that $\lambda_{S,n} = y_n e^{-2\pi i (d_2 - d_1) n/N}$. Similar reasoning applies for $n > N/2$, so that $\lambda_{S,n}$ clearly equals $\widehat{s}$ for some symmetric vector $\boldsymbol{s}$ from Theorems 4.3 and 4.4. This vector equals (up to multiplication with $\sqrt{N}$) the filter coefficients of S, which therefore is a symmetric. Moreover, it is clear that the filter coefficients have an integer symmetry point if and only if the input and output vector either both have an integer symmetry point, or both a non-integer symmetry point. □

Since the filter coefficients of a filter which preserves symmetric vectors also is a symmetric vector, this means that its frequency response takes the form $\lambda_{S,n} = z_n e^{-2\pi i d n/N}$, where $\boldsymbol{z}$ is a real vector. This means that the phase (argument) of the freqency response is $-2\pi dn/N$ or $\pi - 2\pi dn/N$, depending on the sign of z_n. In other words, the phase is linear in n. Filters which preserve symmetric vectors are therefore also called *linear phase filters*

.

Note also that the case $d = 0$ or $d = N - 1/2$ corresponds to symmetric filters. An example of linear phase filters which are not symmetric are smoothing filters where the coefficients are taken from odd rows in Pascal's triangle.

When S is symmetric, it preserves symmetric extensions, so that it makes sense to restrict S to symmetric vectors. We therefore make the following definition.

Definition 4.8. *Symmetric restriction.*

Assume that $S : \mathbb{R}^{2N} \to \mathbb{R}^{2N}$ is a symmetric filter. We define $S_r : \mathbb{R}^N \to \mathbb{R}^N$ as the mapping which sends $\boldsymbol{x} \in \mathbb{R}^N$ to the first N components of the vector $S\check{\boldsymbol{x}}$. S_r is also called the symmetric restriction of S.

S_r is clearly linear, and the restriction of S to vectors symmetric about $N - 1/2$ is characterized by S_r. We continue with the following result:

Theorem 4.9. *Expression for S_r.*
Assume that $S : \mathbb{R}^{2N} \to \mathbb{R}^{2N}$ is a symmetric filter, and that

$$S = \begin{pmatrix} S_1 & S_2 \\ S_3 & S_4 \end{pmatrix}.$$

Then S_r is symmetric, and $S_r = S_1 + (S_2)^f$, where $(S_2)^f$ is the matrix S_2 with the columns reversed.

Proof. With S as in the text of the theorem, we compute

$$S_r\boldsymbol{x} = \begin{pmatrix} S_1 & S_2 \end{pmatrix} \left(\begin{array}{c} x_0 \\ \vdots \\ x_{N-1} \\ \hline x_{N-1} \\ \vdots \\ x_0 \end{array}\right) = S_1 \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix} + S_2 \begin{pmatrix} x_{N-1} \\ \vdots \\ x_0 \end{pmatrix}$$
$$= S_1 \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix} + (S_2)^f \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix} = (S_1 + (S_2)^f)\boldsymbol{x},$$

so that $S_r = S_1 + (S_2)^f$. Since S is symmetric, S_1 is also symmetric. $(S_2)^f$ is also symmetric, since it is constant on anti-diagonals. It follows then that S is also symmetric. This completes the proof. □

Note that S_r is not a digital filter, since its matrix is not circulant. In particular, its eigenvectors are not pure tones. In the block matrix factorization of S, S_2 contains the circulant part of the matrix, and forming $(S_2)^f$ means that the circulant parts switch corners. With the help of Theorem 4.9 we can finally establish the orthogonality of the cosine-vectors in $\mathbb{R}^N$.

Corollary 4.10. *Basis of eigenvectors for S_r.*
Let S be a symmetric filter, and let S_r be the mapping defined in Theorem 4.9. Define

$$d_{n,N} = \begin{cases} \sqrt{\frac{1}{N}} & , n = 0 \\ \sqrt{\frac{2}{N}} & , 1 \leq n < N \end{cases}$$

and $\boldsymbol{d}_n = d_{n,N} \cos\left(2\pi \frac{n}{2N}\left(k + \frac{1}{2}\right)\right)$ for $0 \leq n \leq N - 1$, then $\{\boldsymbol{d}_0, \boldsymbol{d}_1, \ldots, \boldsymbol{d}_{N-1}\}$ is an orthonormal basis of eigenvectors for S_r.

Proof. Let S be a symmetric filter of length $2N$. We know then that $\lambda_{S,n} = \lambda_{S,2N-n}$, so that

$$\begin{aligned}
&S\left(\cos\left(2\pi\frac{n}{2N}\left(k+\frac{1}{2}\right)\right)\right)\\
&= S\left(\frac{1}{2}\left(e^{2\pi i(n/(2N))(k+1/2)} + e^{-2\pi i(n/(2N))(k+1/2)}\right)\right)\\
&= \frac{1}{2}\left(e^{\pi in/(2N)}S\left(e^{2\pi ink/(2N)}\right) + e^{-\pi in/(2N)}S\left(e^{-2\pi ink/(2N)}\right)\right)\\
&= \frac{1}{2}\left(e^{\pi in/(2N)}\lambda_{S,n}e^{2\pi ink/(2N)} + e^{-\pi in/(2N)}\lambda_{S,2N-n}e^{-2\pi ink/(2N)}\right)\\
&= \frac{1}{2}\left(\lambda_{S,n}e^{2\pi i(n/(2N))(k+1/2)} + \lambda_{S,2N-n}e^{-2\pi i(n/(2N))(k+1/2)}\right)\\
&= \lambda_{S,n}\frac{1}{2}\left(e^{2\pi i(n/(2N))(k+1/2)} + e^{-2\pi i(n/(2N))(k+1/2)}\right)\\
&= \lambda_{S,n}\cos\left(2\pi\frac{n}{2N}\left(k+\frac{1}{2}\right)\right),
\end{aligned}$$

where we have used that $e^{2\pi ink/(2N)}$ is an eigenvector of S with eigenvalue $\lambda_{S,n}$, and $e^{-2\pi ink/(2N)} = e^{2\pi i(2N-n)k/(2N)}$ is an eigenvector of S with eigenvalue $\lambda_{S,2N-n}$. This shows that the vectors are eigenvectors for symmetric filters of length $2N$. It is also clear that the first half of the vectors must be eigenvectors for S_r with the same eigenvalue, since when $\boldsymbol{y} = S\boldsymbol{x} = \lambda_{S,n}\boldsymbol{x}$, we also have that

$$(y_0, y_1, \ldots, y_{N-1}) = S_r(x_0, x_1, \ldots, x_{N-1}) = \lambda_{S,n}(x_0, x_1, \ldots, x_{N-1}).$$

To see why these vectors are orthogonal, choose at the outset a symmetric filter where $\{\lambda_{S,n}\}_{n=0}^{N-1}$ are distinct. Then the cosine-vectors of length N are also eigenvectors with distinct eigenvalues, and they must be orthogonal since S_r is symmetric. Moreover, since

$$
\begin{aligned}
&\sum_{k=0}^{2N-1} \cos^2\left(2\pi\frac{n}{2N}\left(k+\frac{1}{2}\right)\right)\\
&= \sum_{k=0}^{N-1} \cos^2\left(2\pi\frac{n}{2N}\left(k+\frac{1}{2}\right)\right) + \sum_{k=N}^{2N-1} \cos^2\left(2\pi\frac{n}{2N}\left(k+\frac{1}{2}\right)\right)\\
&= \sum_{k=0}^{N-1} \cos^2\left(2\pi\frac{n}{2N}\left(k+\frac{1}{2}\right)\right) + \sum_{k=0}^{N-1} \cos^2\left(2\pi\frac{n}{2N}\left(k+N+\frac{1}{2}\right)\right)\\
&= \sum_{k=0}^{N-1} \cos^2\left(2\pi\frac{n}{2N}\left(k+\frac{1}{2}\right)\right) + (-1)^{2n}\sum_{k=0}^{N-1} \cos^2\left(2\pi\frac{n}{2N}\left(k+\frac{1}{2}\right)\right)\\
&= 2\sum_{k=0}^{N-1} \cos^2\left(2\pi\frac{n}{2N}\left(k+\frac{1}{2}\right)\right),
\end{aligned}
$$

where we used that $\cos(x+n\pi) = (-1)^n \cos x$. This means that

$$
\left\|\left\{\cos\left(2\pi\frac{n}{2N}\left(k+\frac{1}{2}\right)\right)\right\}_{k=0}^{2N-1}\right\| = \sqrt{2}\left\|\left\{\cos\left(2\pi\frac{n}{2N}\left(k+\frac{1}{2}\right)\right)\right\}_{k=0}^{N-1}\right\|.
$$

Thus, in order to make the vectors orthonormal when we consider the first N elements instead of all $2N$ elements, we need to multiply with $\sqrt{2}$. This gives us the vectors $\boldsymbol{d}_n$ as defined in the text of the theorem. This completes the proof. □

We now clearly see the analogy between symmetric functions and vectors: while the first can be written as a sum of cosine-functions, the second can be written as a sum of cosine-vectors. The orthogonal basis we have found is given its own name:

Definition 4.11. *DCT basis.*
We denote by $\mathcal{D}_N$ the orthogonal basis $\{\boldsymbol{d}_0, \boldsymbol{d}_1, \ldots, \boldsymbol{d}_{N-1}\}$. We also call $\mathcal{D}_N$ the N-point *DCT* basis.

Using the DCT basis instead of the Fourier basis we can make the following definitions, which parallel those for the DFT:

Definition 4.12. *Discrete Cosine Transform.*
The change of coordinates from the standard basis of $\mathbb{R}^N$ to the DCT basis $\mathcal{D}_N$ is called the *discrete cosine transform* (or DCT). The $N \times N$ matrix DCT_N that represents this change of basis is called the (N-point) DCT matrix. If $\boldsymbol{x}$ is a vector in R^N, its coordinates $\boldsymbol{y} = (y_0, y_1, \ldots, y_{N-1})$ relative to the DCT basis are called the DCT coefficients of $\boldsymbol{x}$ (in other words, $\boldsymbol{y} = \mathrm{DCT}_N\boldsymbol{x}$).

Note that we can also write

$$\mathrm{DCT}_N = \sqrt{\frac{2}{N}} \begin{pmatrix} 1/\sqrt{2} & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} \left(\cos\left(2\pi \tfrac{n}{2N}(k+1/2)\right)\right). \tag{4.3}$$

Since this matrix is orthogonal, it is immediate that

$$\left(\cos\left(2\pi \tfrac{n}{2N}(k+1/2)\right)\right)^{-1} = \frac{2}{N}\left(\cos\left(2\pi \tfrac{n+1/2}{2N}k\right)\right) \begin{pmatrix} 1/2 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} \tag{4.4}$$

$$\left(\cos\left(2\pi \tfrac{n+1/2}{2N}k\right)\right)^{-1} = \frac{2}{N} \begin{pmatrix} 1/2 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} \left(\cos\left(2\pi \tfrac{n}{2N}(k+1/2)\right)\right). \tag{4.5}$$

In other words, not only can DCT_N be directly expressed in terms of a cosine-matrix, but our developments helped us to express the inverse of a cosine matrix in terms of other cosine-matrices. In the literature different types of cosine-matrices have been useful:

I Cosine-matrices with entries $\cos(2\pi nk/(2(N-1)))$.

II Cosine-matrices with entries $\cos(2\pi n(k+1/2)/(2N))$.

III Cosine-matrices with entries $\cos(2\pi(n+1/2)k/(2N))$.

IV Cosine-matrices with entries $\cos(2\pi(n+1/2)(k+1/2)/(2N))$.

We will call these type-I, type-II, type-III, and type-IV cosine-matrices, respectively. What we did above handles the case of type-II cosine-matrices. It will turn out that not all of these cosine-matrices are orthogonal, but that we in all cases, as we did above for type-II cosine matrices, can express the inverse of a cosine-matrix of one type in terms of a cosine-matrix of another type, and that any cosine-matrix is easily expressed in terms of an orthogonal matrix. These orthogonal matrices will be called $\mathrm{DCT}_N^{(I)}$, $\mathrm{DCT}_N^{(II)}$, $\mathrm{DCT}_N^{(III)}$, and $\mathrm{DCT}_N^{(IV)}$, respectively, and they are all called DCT-matrices. The DCT_N we constructed abobe is thus $\mathrm{DCT}_N^{(II)}$. The type-II DCT matrix is the most commonly used, and the type is therefore often dropped when refering to these. We will consider the other cases of cosine-matrices at different places in this book: In the next chapter we will run into type-I cosine matrices, in connection with a different extension strategy used for wavelets. Type-IV cosine-matrices will be encountered in exercises 4.5 and 4.6 at the end of this section.

As with the Fourier basis vectors, the DCT basis vectors are called synthesis vectors, since we can write

$$\boldsymbol{x} = y_0\boldsymbol{d}_0 + y_1\boldsymbol{d}_1 + \cdots + y_{N-1}\boldsymbol{d}_{N-1} \tag{4.6}$$

in the same way as for the DFT. Following the same reasoning as for the DFT, DCT_N^{-1} is the matrix where the $\boldsymbol{d}_n$ are columns. But since these vectors are real and orthonormal, DCT_N must be the matrix where the $\boldsymbol{d}_n$ are rows. Moreover, since Theorem 4.9 also states that the same vectors are eigenvectors for filters which preserve symmetric extensions, we can state the following:

Theorem 4.13. *The DCT is orthogonal.*

DCT_N is the orthogonal matrix where the rows are $\boldsymbol{d}_n$. Moreover, for any digital filter S which preserves symmetric extensions, $(\mathrm{DCT}_N)^T$ diagonalizes S_r, i.e. $S_r = \mathrm{DCT}_N^T D \mathrm{DCT}_N$ where D is a diagonal matrix.

Let us also make the following definition:

Definition 4.14. *IDCT.*

We will call $\boldsymbol{x} = (\mathrm{DCT}_N)^T\boldsymbol{y}$ the inverse DCT or (IDCT) of $\boldsymbol{x}$.

Example 4.1: Computing lower order DCTs

As with Example 2.3, exact expressions for the DCT can be written down just for a few specific cases. It turns out that the case $N = 4$ as considered in Example 2.3 does not give the same type of nice, exact values, so let us instead consider the case $N = 2$. We have that

$$\mathrm{DCT}_4 = \begin{pmatrix} \frac{1}{\sqrt{2}}\cos(0) & \frac{1}{\sqrt{2}}\cos(0) \\ \cos\left(\frac{\pi}{2}\left(0+\frac{1}{2}\right)\right) & \cos\left(\frac{\pi}{2}\left(1+\frac{1}{2}\right)\right) \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$$

The DCT of the same vector as in Example 2.3 can now be computed as:

$$\mathrm{DCT}_2\begin{pmatrix}1\\2\end{pmatrix} = \begin{pmatrix} \frac{3}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix}.$$

Matlab's functions for computing the DCT and IDCT are called `dct`, and `idct`, respectively. These are defined exactly as they are here, contrary to the case for the FFT (where a different normalizing factor was used).

With these functions we can repeat examples 2.16- 2.18, by simply replacing the calls to `DFTImpl` with calls to the DCT counterparts. You may not here much improvements in these simple experiments, but in theory the DCT should be able to approximate sound better.

Similarly to the DFT, one can think of the DCT as a least squares approximation and the unique representation of a function having the same sample values, but this time in terms of sinusoids instead of complex exponentials:

Theorem 4.15. *Interpolation with the DCT basis.*

Let f be a function defined on the interval $[0, T]$, and let $\boldsymbol{x}$ be the sampled vector given by

$$x_k = f((2k+1)T/(2N)) \quad \text{for } k = 0,\, 1,\, \ldots,\, N-1.$$

There is exactly one linear combination $g(t)$ on the form

$$\sum_{n=0}^{N-1} y_n d_{n,N} \cos(2\pi(n/2)t/T)$$

which satisfies the conditions

$$g((2k+1)T/(2N)) = f((2k+1)T/(2N)), \quad k = 0,\, 1,\, \ldots,\, N-1,$$

and its coefficients are determined by $\boldsymbol{y} = \mathrm{DCT}_N \boldsymbol{x}$.

Proof. This follows by inserting $t = (2k+1)T/(2N)$ in the equation

$$g(t) = \sum_{n=0}^{N-1} y_n d_{n,N} \cos(2\pi(n/2)t/T)$$

to arrive at the equations

$$f(kT/N) = \sum_{n=0}^{N-1} y_n d_{n,N} \cos\left(2\pi \frac{n}{2N}\left(k + \frac{1}{2}\right)\right) \qquad 0 \le k \le N-1.$$

This gives us an equation system for finding the y_n with the invertible DCT matrix as coefficient matrix, and the result follows. □

Due to this there is a slight difference to how we applied the DFT, due to the subtle change in the sample points, from kT/N for the DFT, to $(2k+1)T/(2N)$ for the DCT. The sample points for the DCT are thus the midpoints on the intervals in a uniform partition of $[0, T]$ into N intervals, while they for the DFT are the start points on the intervals. Also, the frequencies are divided by 2. In Figure 4.2 we have plotted the sinusoids of Theorem **??** for $T = 1$, as well as the sample points used in that theorem.

The sample points in the upper left plot correspond to the first column in the DCT matrix, the sample points in the upper right plot to the second column of the DCT matrix, and so on (up to normalization with $d_{n,N}$). As n increases, the functions oscillate more and more. As an example, y_5 says how much content of maximum oscillation there is. In other words, the DCT of an audio signal shows the proportion of the different frequencies in the signal, and the two formulas $\boldsymbol{y} = \mathrm{DCT}_N \boldsymbol{x}$ and $\boldsymbol{x} = (\mathrm{DCT}_N)^T \boldsymbol{y}$ allow us to switch back and forth between the time domain representation and the frequency domain representation of the sound. In other words, once we have computed $\boldsymbol{y} = \mathrm{DCT}_N \boldsymbol{x}$, we can analyse

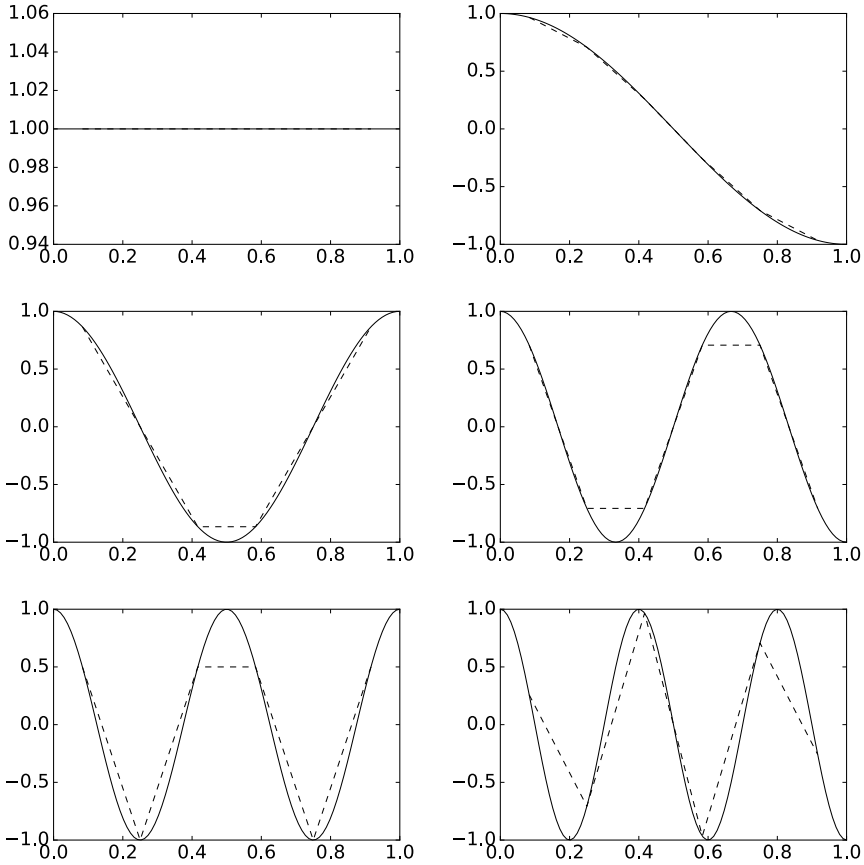

Figure 4.2: The 6 different sinusoids used in DCT for $N = 6$, i.e. $\cos(2\pi(n/2)t)$, $0 \leq n < 6$. The plots also show piecewise linear functions (in red) between the sample points $\frac{2k+1}{2N}$ $0 \leq k < 6$, since only the values at these points are used in Theorem **??**.

the frequency content of $\boldsymbol{x}$. If we want to reduce the bass we can decrease the $\boldsymbol{y}$-values with small indices and if we want to increase the treble we can increase the $\boldsymbol{y}$-values with large indices.

Exercise 4.2: Computing eigenvalues

Consider the matrix

$$S = \frac{1}{3}\begin{pmatrix} 2 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 2 \end{pmatrix}$$

a) Compute the eigenvalues and eigenvectors of S using the results of this section. You should only need to perform one DFT or one DCT in order to achieve this.

b) Use a computer to compute the eigenvectors and eigenvalues of S also. What are the differences from what you found in a)?

c) Find a filter T so that $S = T_r$. What kind of filter is T?

Exercise 4.3: Writing down lower order S_r

Consider the averaging filter $S = \{\frac{1}{4}, \underline{\frac{1}{2}}, \frac{1}{4}\}$. Write down the matrix S_r for the case when $N = 4$.

Exercise 4.4: Writing down lower order DCTs

As in Example 4.1, state the exact cartesian form of the DCT matrix for the case $N = 3$.

Exercise 4.5: DCT-IV

Show that the vectors $\left\{\cos\left(2\pi\frac{n+\frac{1}{2}}{2N}\left(k+\frac{1}{2}\right)\right)\right\}_{n=0}^{N-1}$ in $\mathbb{R}^N$ are orthogonal, with lengths $\sqrt{N/2}$. This means that the matrix with entries $\sqrt{\frac{2}{N}}\cos\left(2\pi\frac{n+\frac{1}{2}}{2N}\left(k+\frac{1}{2}\right)\right)$ is orthogonal. Since this matrix also is symmetric, it is its own inverse. This is the DCT-IV, which we denote by $\mathrm{DCT}_N^{(\mathrm{IV})}$. Although we will not consider this, the DCT-IV also has an efficient implementation.

Hint. Compare with the orthogonal vectors $\boldsymbol{d}_n$, used in the DCT.

Exercise 4.6: MDCT

The MDCT is defined as the $N\times(2N)$-matrix M with elements $M_{n,k} = \cos(2\pi(n+1/2)(k+1/2+N/2)/(2N))$. This exercise will take you through the details of the transformation which corresponds to multiplication with this matrix. The MDCT is very useful, and is also used in the MP3 standard and in more recent standards.

a) Show that

$$M = \sqrt{\frac{N}{2}} \mathrm{DCT}_N^{(\mathrm{IV})} \begin{pmatrix} \mathbf{0} & A \\ B & \mathbf{0} \end{pmatrix}$$

where A and B are the $(N/2) \times N$-matrices

$$A = \begin{pmatrix} \cdots & \cdots & 0 & -1 & -1 & 0 & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & -1 & \cdots & \cdots & \cdots & \cdots & -1 & 0 \\ -1 & 0 & \cdots & \cdots & \cdots & \cdots & 0 & -1 \end{pmatrix} = \begin{pmatrix} -I_{N/2}^f & -I_{N/2} \end{pmatrix}$$

$$B = \begin{pmatrix} 1 & 0 & \cdots & \cdots & \cdots & \cdots & 0 & -1 \\ 0 & 1 & \cdots & \cdots & \cdots & \cdots & -1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \cdots & 0 & 1 & -1 & 0 & \cdots & \cdots \end{pmatrix} = \begin{pmatrix} I_{N/2} & -I_{N/2}^f \end{pmatrix}.$$

Due to this expression, any algorihtm for the DCT-IV can be used to compute the MDCT.

b) The MDCT is not invertible, since it is not a square matrix. We will show here that it still can be used in connection with invertible transformations. We first define the IMDCT as the matrix M^T/N. Transposing the matrix expression we obtained in a) gives

$$\frac{1}{\sqrt{2N}} \begin{pmatrix} \mathbf{0} & B^T \\ A^T & \mathbf{0} \end{pmatrix} \mathrm{DCT}_N^{(\mathrm{IV})}$$

for the IMDCT, which thus also has an efficient implementation. Show that if

$$\boldsymbol{x}_0 = (x_0, \ldots, x_{N-1}) \qquad \boldsymbol{x}_1 = (x_N, \ldots, x_{2N-1}) \qquad \boldsymbol{x}_2 = (x_{2N}, \ldots, x_{3N-1})$$

and

$$\boldsymbol{y_{0,1}} = M \begin{pmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \end{pmatrix} \qquad\qquad \boldsymbol{y_{1,2}} = M \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{pmatrix}$$

(i.e. we compute two MDCT's where half of the data overlap), then

$$\boldsymbol{x}_1 = \{\mathrm{IMDCT}(\boldsymbol{y_{0,1}})\}_{k=N}^{2N-1} + \{\mathrm{IMDCT}(\boldsymbol{y_{1,2}})\}_{k=0}^{N-1}.$$

Even though the MDCT itself is not invertible, the input can still be recovered from overlapping MDCT's.

4.2 Improvements using the DCT for interpolation

Recall that, in Section 3.2.2, we explained how to approximate an analog filter from the samples. It turns out that, when an analog filter is symmetric, we can use symmetric extensions to create a better approximation from the samples.

Assume that s is an analog filter, and that we apply it to a general function f. Denote as before the symmetric extension of f by $\breve{f}$. We start with the following observation, which follows from the continuity of s.

Observation 4.16. *Using symmetric extensions for approximations.*
Since $(\breve{f})_N$ is a better approximation to $\breve{f}$, compared to what f_N is to f, $s((\breve{f})_N)$ is a better approximation to $s(\breve{f})$, compared to what $s(f_N)$ is to $s(f)$.

Since $s(\breve{f})$ agrees with $s(f)$ except near the boundaries, we can thus conclude that $s((\breve{f})_N)$ is a better approximation to $s(f)$ than what $s(f_N)$ is.

We have seen that the restriction of s to $V_{M,T}$ is equivalent to an $N \times N$ digital filter S, where $N = 2M + 1$. Let $\boldsymbol{x}$ be the samples of f, $\breve{\boldsymbol{x}}$ the samples of $\breve{f}$. Turning around the fact that $(\breve{f})_N$ is a better approximation to $\breve{f}$, compared to what f_N is to f, the following is clear.

Observation 4.17. *Using symmetric extensions for approximations.*
The samples $\breve{\boldsymbol{x}}$ are a better approximation to the samples of $(\breve{f})_N$, than the samples $\boldsymbol{x}$ are to the samples of f_N.

Now, let $\boldsymbol{z} = S\boldsymbol{x}$, and $\breve{\boldsymbol{z}} = S\breve{\boldsymbol{x}}$. The following is also clear from the preceding observation, due to continuity of the digital filter S.

Observation 4.18. *Using symmetric extensions for approximations.*
$\breve{\boldsymbol{z}}$ is a better approximation to S(samples of $(\breve{f})_N$) = samples of $s((\breve{f})_N)$, than $\boldsymbol{z}$ is to S(samples of f_N) = samples of $s(f_N)$.

Since by Observation 4.16 $s((\breve{f})_N)$ is a better approximation to the output $s(f)$, we conclude that $\breve{\boldsymbol{z}}$ is a better approximation than $\boldsymbol{z}$ to the samples of the output of the filter.

Observation 4.19. *Using symmetric extensions for approximations.*
$S\breve{\boldsymbol{x}}$ is a better approximation to the samples of $s(f)$ than $S\boldsymbol{x}$ is ($\boldsymbol{x}$ are the samples of f).

Now, let us also bring in the assumption that s is symmetric. Then the corresponding digital filter S is also symmetric, and we know then that we can view its restriction to symmetric extensions in $\mathbb{R}^{2N}$ in terms of the mapping $S_r : \mathbb{R}^N \to \mathbb{R}^N$. We can thus specialize Figure 3.3 to symmetric filters by adding the step of creating the symmetric extension, and replacing S with S_r. We have summarized these remarks in Figure 4.3. The DCT appears here, since we have used Theorem **??** to interpolate with the DCT basis, instead of the Fourier basis. Note that this also requires that the sampling is performed as required in that

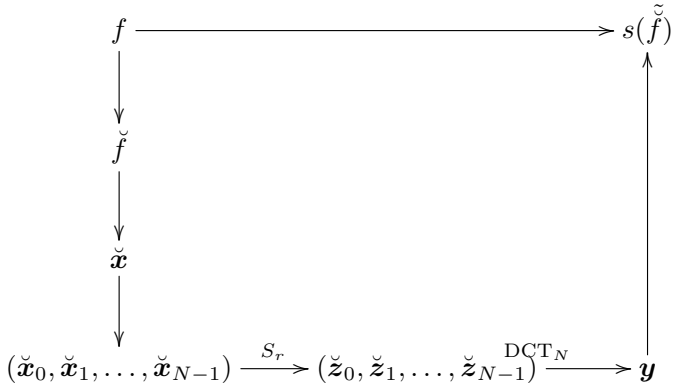

Figure 4.3: The connections between the new mapping S_r, sampling, and interpolation. The right vertical arrow represents interpolation with the DCT, i.e. that we compute $\sum_{n=0}^{N-1} y_n d_{n,N} \cos(2\pi(n/2)t/T)$ for values of t.

theorem, i.e. the samples are the midpoints on all intervals. This new sampling procedure is not indicated in Figure 4.3.

Figure 4.3 can be further simplified to that shown in Figure 4.4.

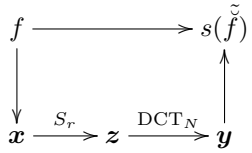

Figure 4.4: Simplification of Figure 4.3. The left vertical arrow represents sampling as dictated by the DCT.

Note that the assumption that s is symmetric only helped us to implement the approximation $s(\breve{\breve{f}})$ more efficiently, since S_r has N points and S has $2N$ points. $s(\breve{\breve{f}})$ can in any way be used as an approximation, even if s is not symmetric, but the mapping does not preserve symmetry.

As mentioned in Section 3.2, interpolation of a function from its samples can be seen as a special case. This can thus be illustrated as in Figure 4.5.

Note that the approximation lies in $V_{2M,2T}$ (i.e. it is in a higher order Fourier space), but the point is that the same number of samples is used.

4.2.1 Implementations of symmetric filters

Symmetric filters are also important for applications since they can be implemented efficiently. To see this, we can write

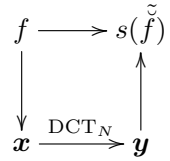

Figure 4.5: How we can approximate a function from its samples with the DCT.

$$
\begin{aligned}
(S\boldsymbol{x})_n &= \sum_{k=0}^{N-1} s_k x_{(n-k) \bmod N} \\
&= s_0 x_n + \sum_{k=1}^{(N-1)/2} s_k x_{(n-k) \bmod N} + \sum_{k=(N+1)/2}^{N-1} s_k x_{(n-k) \bmod N} \\
&= s_0 x_n + \sum_{k=1}^{(N-1)/2} s_k x_{(n-k) \bmod N} + \sum_{k=1}^{(N-1)/2} s_k x_{(n-(N-k)) \bmod N} \\
&= s_0 x_n + \sum_{k=1}^{(N-1)/2} s_k (x_{(n-k) \bmod N} + x_{(n+k) \bmod N}). \qquad (4.7)
\end{aligned}
$$

If we compare the first and last expressions here, we need the same number of summations, but the number of multiplications needed in the latter expression has been halved.

Observation 4.20. *Reducing arithmetic operations for symmetric filters.*

Assume that a symmetric filter has $2s+1$ filter coefficients. The filter applied to a vector of length N can then be implemented using $(s+1)N$ multiplications and $2sN$ additions. This gives a reduced number of arithmetic operations when compared to a filter with the same number of coefficients which is not symmetric, where a direct implementations requires $(2s+1)N$ multiplications and $2sN$ additions.

Similarly to as in Section 3.6.2, a symmetric filter can be factored into a product of symmetric filters. To see how, note first that a real polynomial is symmetric if and only if $1/a$ is a root whenever a is. If we pair together the factors for the roots $a, 1/a$ when a is real we get a component in the frequency response of degree 2. If we pair the factors for the roots $a, 1/a, \overline{a}, \overline{1/a}$ when a is complex, we get a component in the frequency response of degree 4. We thus get the following idea:

Idea 4.21. *Factorizing symmetric filters.*

Let S be a symmetric filter with real coefficients. There exist constants K, $a_1, \ldots, a_m$, $b_1, c_1, \ldots, b_n, c_n$ so that

$$\begin{aligned}\lambda_S(\omega) =& K(a_1e^{i\omega} + 1 + a_1e^{-i\omega})\dots(a_me^{i\omega} + 1 + a_me^{-i\omega}) \\ &\times (b_1e^{2i\omega} + c_1e^{i\omega} + 1 + c_1e^{-i\omega} + b_1e^{-2i\omega})\dots \\ &\times (b_ne^{2i\omega} + c_ne^{i\omega} + 1 + c_ne^{-i\omega} + b_ne^{-2i\omega}).\end{aligned}$$

We can write $S = KA_1 \dots A_m B_1 \dots B_n$, where $A_i = \{a_i, \underline{1}, a_i\}$ and $B_i = \{b_i, c_i, \underline{1}, c_i, b_i\}$.

In any case we see that the component filters have 3 and 5 filter coefficients.

Exercise 4.7: Component expressions for a symmetric filter

Assume that $S = t_{-L}, \dots, \underline{t_0}, \dots, t_L$ is a symmetric filter. Use Equation (4.7) to show that $z_n = (S\boldsymbol{x})_n$ in this case can be split into the following different formulas, depending on n:

a) $0 \le n < L$:

$$z_n = t_0x_n + \sum_{k=1}^{n} t_k(x_{n+k} + x_{n-k}) + \sum_{k=n+1}^{L} t_k(x_{n+k} + x_{n-k+N}). \tag{4.8}$$

b) $L \le n < N - L$:

$$z_n = t_0x_n + \sum_{k=1}^{L} t_k(x_{n+k} + x_{n-k}). \tag{4.9}$$

c) $N - L \le n < N$:

$$z_n = t_0x_n + \sum_{k=1}^{N-1-n} t_k(x_{n+k} + x_{n-k}) + \sum_{k=N-1-n+1}^{L} t_k(x_{n+k-N} + x_{n-k}). \tag{4.10}$$

The `conv` function may not pick up this reduction in the number of multiplications, since it does not assume that the filter is symmetric. We will still use the `conv` function in implementations, however, due to its heavy optimization.

4.3 Efficient implementations of the DCT

When we defined the DCT in the preceding section, we considered symmetric vectors of twice the length, and viewed these in the frequency domain. In order to have a fast algorithm for the DCT, which are comparable to the FFT algorithms we developed in Section 2.3, we need to address the fact that vectors of twice the length seem to be involved. The following theorem addresses this. This result is much used in practical implementations of DCT, and can also be used

for practical implementation of the DFT as we will see in Exercise 4.9. Note that the result, and the following results in this section, are stated in terms of the cosine matrix C_N (where the entries are $(C_N)_{n,k} = \cos\left(2\pi \frac{n}{2N}\left(k+\frac{1}{2}\right)\right)$, rather than the DCT_N matrix (which uses the additional scaling factor $d_{n,N}$ for the rows). The reason is that C_N appears to me most practical for stating algorithms. When computing the DCT, we simply need to scale with the $d_{n,N}$ at the end, after using the statements below.

Theorem 4.22. *DCT algorithm.*
Let $\boldsymbol{y} = C_N \boldsymbol{x}$. Then we have that

$$y_n = \left(\cos\left(\pi \frac{n}{2N}\right) \Re((\mathrm{DFT}_N \boldsymbol{x}^{(1)})_n) + \sin\left(\pi \frac{n}{2N}\right) \Im((\mathrm{DFT}_N \boldsymbol{x}^{(1)})_n)\right), \quad (4.11)$$

where $\boldsymbol{x}^{(1)} \in \mathbb{R}^N$ is defined by

$$(\boldsymbol{x}^{(1)})_k = x_{2k} \text{ for } 0 \le k \le N/2 - 1$$
$$(\boldsymbol{x}^{(1)})_{N-k-1} = x_{2k+1} \text{ for } 0 \le k \le N/2 - 1,$$

Proof. Using the definition of C_N, ans splitting the computation of $\boldsymbol{y} = C_N \boldsymbol{x}$ into two sums, corresponding to the even and odd indices as follows:

$$\begin{aligned} y_n &= \sum_{k=0}^{N-1} x_k \cos\left(2\pi \frac{n}{2N}\left(k+\frac{1}{2}\right)\right) \\ &= \sum_{k=0}^{N/2-1} x_{2k} \cos\left(2\pi \frac{n}{2N}\left(2k+\frac{1}{2}\right)\right) + \sum_{k=0}^{N/2-1} x_{2k+1} \cos\left(2\pi \frac{n}{2N}\left(2k+1+\frac{1}{2}\right)\right). \end{aligned}$$

If we reverse the indices in the second sum, this sum becomes

$$\sum_{k=0}^{N/2-1} x_{N-2k-1} \cos\left(2\pi \frac{n}{2N}\left(N-2k-1+\frac{1}{2}\right)\right).$$

If we then also shift the indices with $N/2$ in this sum, we get

$$\begin{aligned} &\sum_{k=N/2}^{N-1} x_{2N-2k-1} \cos\left(2\pi \frac{n}{2N}\left(2N-2k-1+\frac{1}{2}\right)\right) \\ &= \sum_{k=N/2}^{N-1} x_{2N-2k-1} \cos\left(2\pi \frac{n}{2N}\left(2k+\frac{1}{2}\right)\right), \end{aligned}$$

where we used that cos is symmetric and periodic with period 2π. We see that we now have the same cos-terms in the two sums. If we thus define the vector $\boldsymbol{x}^{(1)}$ as in the text of the theorem, we see that we can write

$$
\begin{aligned}
y_n &= \sum_{k=0}^{N-1} (\boldsymbol{x}^{(1)})_k \cos\left(2\pi \frac{n}{2N}\left(2k+\frac{1}{2}\right)\right) \\
&= \Re\left(\sum_{k=0}^{N-1} (\boldsymbol{x}^{(1)})_k e^{-2\pi i n\left(2k+\frac{1}{2}\right)/(2N)}\right) \\
&= \Re\left(e^{-\pi i n/(2N)} \sum_{k=0}^{N-1} (\boldsymbol{x}^{(1)})_k e^{-2\pi i n k/N}\right) \\
&= \Re\left(e^{-\pi i n/(2N)} (\mathrm{DFT}_N \boldsymbol{x}^{(1)})_n\right) \\
&= \left(\cos\left(\pi \frac{n}{2N}\right) \Re((\mathrm{DFT}_N \boldsymbol{x}^{(1)})_n) + \sin\left(\pi \frac{n}{2N}\right) \Im((\mathrm{DFT}_N \boldsymbol{x}^{(1)})_n)\right),
\end{aligned}
$$

where we have recognized the N-point DFT. This completes the proof. □

With the result above we have avoided computing a DFT of double size. If we in the proof above define the $N \times N$-diagonal matrix Q_N by $Q_{n,n} = e^{-\pi i n/(2N)}$, the result can also be written on the more compact form

$$
\boldsymbol{y} = C_N \boldsymbol{x} = \Re\left(Q_N \mathrm{DFT}_N \boldsymbol{x}^{(1)}\right).
$$

We will, however, not use this form, since there is complex arithmetic involved, contrary to Equation(4.11). Code which uses Equation (4.11) to compute the DCT, using the function `FFTImpl` from Section 2.3, can look as follows:

```
function y = DCTImpl(x)
    N = length(x);
    if N == 1
        y = x;
    else
        x1 = [x(1:2:N, :); x(N:(-2):1, :)];
        y = FFTImpl(x1, @FFTKernelStandard);
        cosvec = cos(pi*((0:(N-1))')/(2*N));
        sinvec = sin(pi*((0:(N-1))')/(2*N));
        for s2 = 1:size(x, 2)
            y(:, s2) = cosvec.*real(y(:, s2)) + sinvec.*imag(y(:, s2));
        end
        y(1, :) = sqrt(1/N)*y(1, :);
        y(2:N, :) = sqrt(2/N)*y(2:N, :);
    end
```

In the code, the vector $\boldsymbol{x}^{(1)}$ is created first by rearranging the components, and it is sent as input to `FFTImpl`. After this we take real parts and imaginary parts, and multiply with the cos- and sin-terms in Equation (4.11).

4.3.1 Efficient implementations of the IDCT

As with the FFT, it is straightforward to modify the DCT implementation so that it returns the IDCT. To see how we can do this, write from Theorem 4.22, for $n \geq 1$

$$\begin{aligned} y_n &= \left(\cos\left(\pi\frac{n}{2N}\right)\Re((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n) + \sin\left(\pi\frac{n}{2N}\right)\Im((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n)\right) \\ y_{N-n} &= \left(\cos\left(\pi\frac{N-n}{2N}\right)\Re((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_{N-n}) + \sin\left(\pi\frac{N-n}{2N}\right)\Im((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_{N-n})\right) \\ &= \left(\sin\left(\pi\frac{n}{2N}\right)\Re((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n) - \cos\left(\pi\frac{n}{2N}\right)\Im((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n)\right), \end{aligned} \tag{4.12}$$

where we have used the symmetry of DFT_N for real signals. These two equations enable us to determine $\Re((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n)$ and $\Im((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n)$ from y_n and y_{N-n}. We get

$$\begin{aligned} \cos\left(\pi\frac{n}{2N}\right)y_n + \sin\left(\pi\frac{n}{2N}\right)y_{N-n} &= \Re((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n) \\ \sin\left(\pi\frac{n}{2N}\right)y_n - \cos\left(\pi\frac{n}{2N}\right)y_{N-n} &= \Im((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n). \end{aligned}$$

Adding we get

$$\begin{aligned} (\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n =& \cos\left(\pi\frac{n}{2N}\right)y_n + \sin\left(\pi\frac{n}{2N}\right)y_{N-n} + i(\sin\left(\pi\frac{n}{2N}\right)y_n - \cos\left(\pi\frac{n}{2N}\right)y_{N-n}) \\ =& (\cos\left(\pi\frac{n}{2N}\right) + i\sin\left(\pi\frac{n}{2N}\right))(y_n - iy_{N-n}) = e^{\pi in/(2N)}(y_n - iy_{N-n}). \end{aligned}$$

This means that $(\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n = e^{\pi in/(2N)}(y_n + iy_{N-n}) = (y_n + iy_{N-n})/Q_{n,n}$ for $n \geq 1$. Since $\Im((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_0) = 0$ we have that $(\mathrm{DFT}_N\boldsymbol{x}^{(1)})_0 = \frac{1}{d_{0,N}}y_0 = y_0/Q_{0,0}$. This means that $\boldsymbol{x}^{(1)}$ can be recovered by taking the IDFT of the vector with component 0 being $y_0/Q_{0,0}$, and the remaining components being $(y_n - iy_{N-n})/Q_{n,n}$:

Theorem 4.23. *IDCT algorithm.*

Let $\boldsymbol{x} = (C_N)^{-1}\boldsymbol{y}$. and let $\boldsymbol{z}$ be the vector with component 0 being $y_0/Q_{0,0}$, and the remaining components being $(y_n - iy_{N-n})/Q_{n,n}$. Then we have that

$$\boldsymbol{x}^{(1)} = \mathrm{IDFT}_N\boldsymbol{z},$$

where $\boldsymbol{x}^{(1)}$ is defined as in Theorem 4.22.

The implementation of IDCT can thus go as follows:

```
function x = IDCTImpl(y)
    N = size(y, 1);
    if N == 1
        x = y;
    else
        y(1, :) = y(1, :)/sqrt(1/N);
        y(2:N, :) = y(2:N, :)/sqrt(2/N);
        Q = exp(-pi*1i*((0:(N-1))')/(2*N));
```

```
        y1 = zeros(size(y)); y1(1, :) = y(1, :)/Q(1);
        for s2 = 1:size(y, 2)
            y1(2:N, s2) = (y(2:N, s2)-1i*y(N:(-1):2, s2))./Q(2:N);
        end
        y1 = FFTImpl(y1, @FFTKernelStandard, 0);
        x = zeros(size(y));
        x(1:2:N, :) = real(y1(1:(N/2), :));
        x(2:2:N, :) = real(y1(N:(-1):(N/2+1), :));
    end
```

4.3.2 Reduction in the number of arithmetic operations

Let us also state a result which confirms that the DCT and IDCT implementations we have described give the same type of reductions in the number multiplications as the FFT and IFFT:

Theorem 4.24. *Number of multiplications required by the DCT and IDCT algorithms.*

The DCT and the IDCT can be implemented so that they use any FFT and IFFT algorithms. Their operation counts then have the same order as these. In particular, when the standard FFT algorithms of Section 2.3 are used, i.e. their operation counts are $O(5N \log_2 /2)$. In comparison, the operation count for a direct implementation of the N-point DCT/IDCT is $2N^2$.

Note that we divide the previous operation counts by 2 since the DCT applies an FFT to real input only, and the operation count for the FFT can be halved when we adapt to real data, see Exercise 2.27.

Proof. By Theorem 2.20, the number of multiplications required by the standard FFT algorithm from Section 2.3 adapted to real data is $O(N \log_2 N)$, while the number of additions is $O(3N \log_2 N/2)$. By Theorem 4.22, two additional multiplications and one addition are required for each index (so that we have $2N$ extra real multiplications and N extra real additions in total), but this does not affect the operation count, since $O(N \log_2 N + 2N) = O(N \log_2 N)$. Since the operation counts for the IFFT is the same as for the FFT, we only need to count the additional multiplications needed in forming the vector $\boldsymbol{z} = (y_n - iy_{N-n})/Q_{n,n}$. Clearly, this also does not affect the order of the algorithm. □

Since the DCT and IDCT can be implemented using the FFT and IFFT, it has the same advantages as the FFT when it comes to parallel computing. Much literature is devoted to reducing the number of multiplications in the DFT and the DCT even further than what we have done (see [7] for one of the most recent developments). Another note on computational complexity is in order: we have not counted the operations sin and cos in the DCT. The reason is that these values can be precomputed, since we take the sine and cosine of a specific set of values for each DCT or DFT of a given size. This is contrary to to multiplication and addition, since these include the input values, which are only known at runtime. We have, however, not written down that we use precomputed arrays for sine and cosine in our algorithms: This is an issue to include in more optimized algorithms.

Exercise 4.8: Trick for reducing the number of multiplications with the DCT

In this exercise we will take a look at a small trick which reduces the number of additional multiplications we need for DCT algorithm from Theorem 4.22. This exercise does not reduce the order of the DCT algorithms, but we will see in Exercise 4.9 how the result can be used to achieve this.

a) Assume that $\boldsymbol{x}$ is a real signal. Equation (4.12), which said that

$$\begin{aligned} y_n &= \cos\left(\pi\frac{n}{2N}\right)\Re((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n) + \sin\left(\pi\frac{n}{2N}\right)\Im((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n) \\ y_{N-n} &= \sin\left(\pi\frac{n}{2N}\right)\Re((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n) - \cos\left(\pi\frac{n}{2N}\right)\Im((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n) \end{aligned}$$

for the n'th and $N-n$'th coefficient of the DCT. This can also be rewritten as

$$\begin{aligned} y_n &= \left(\Re((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n) + \Im((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n)\right)\cos\left(\pi\frac{n}{2N}\right) \\ &\quad - \Im((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n)(\cos\left(\pi\frac{n}{2N}\right) - \sin\left(\pi\frac{n}{2N}\right)) \\ y_{N-n} &= -\left(\Re((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n) + \Im((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n)\right)\cos\left(\pi\frac{n}{2N}\right) \\ &\quad + \Re((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n)(\sin\left(\pi\frac{n}{2N}\right) + \cos\left(\pi\frac{n}{2N}\right)). \end{aligned}$$

Explain that the first two equations require 4 multiplications to compute y_n and y_{N-n}, and that the last two equations require 3 multiplications to compute y_n and y_{N-n}.

b) Explain why the trick in a) reduces the number of additional multiplications in a DCT, from $2N$ to $3N/2$.

c) Explain why the trick in a) can be used to reduce the number of additional multiplications in an IDCT with the same number.

Hint. match the expression $e^{\pi in/(2N)}(y_n - iy_{N-n})$ you encountered in the IDCT with the rewriting you did in b).

d) Show that the penalty of the trick we here have used to reduce the number of multiplications, is an increase in the number of additional additions from N to $3N/2$. Why can this trick still be useful?

Exercise 4.9: An efficient joint implementation of the DCT and the FFT

In this exercise we will explain another joint implementation of the DFT and the DCT, which has the benefit of a low multiplication count, at the expense of a higher addition count. It also has the benefit that it is specialized to real

vectors, with a very structured implementation (this is not always the case for the quickest FFT implementations. Not surprisingly, one often sacrifices clarity of code when one pursues higher computational speed). a) of this exercise can be skipped, as it is difficult and quite technical. For further details of the algorithm the reader is refered to [14].

a) Let $\boldsymbol{y} = \mathrm{DFT}_N \boldsymbol{x}$ be the N-point DFT of the real vector $\boldsymbol{x}$. Show that

$$\Re(y_n) = \begin{cases} \Re((\mathrm{DFT}_{N/2}\boldsymbol{x}^{(e)})_n) + (C_{N/4}\boldsymbol{z})_n & 0 \leq n \leq N/4-1 \\ \Re((\mathrm{DFT}_{N/2}\boldsymbol{x}^{(e)})_n) & n = N/4 \\ \Re((\mathrm{DFT}_{N/2}\boldsymbol{x}^{(e)})_n) - (C_{N/4}\boldsymbol{z})_{N/2-n} & N/4+1 \leq n \leq N/2-1 \end{cases} \tag{4.13}$$

$$\Im(y_n) = \begin{cases} \Im((\mathrm{DFT}_{N/2}\boldsymbol{x}^{(e)})_n) & n = 0 \\ \Im((\mathrm{DFT}_{N/2}\boldsymbol{x}^{(e)})_n) + (C_{N/4}\boldsymbol{w})_{N/4-n} & 1 \leq n \leq N/4-1 \\ \Im((\mathrm{DFT}_{N/2}\boldsymbol{x}^{(e)})_n) + (C_{N/4}\boldsymbol{w})_{n-N/4} & N/4 \leq n \leq N/2-1 \end{cases} \tag{4.14}$$

where $\boldsymbol{x}^{(e)}$ is as defined in Theorem 2.15, where $\boldsymbol{z}, \boldsymbol{w} \in \mathbb{R}^{N/4}$ defined by

$$\begin{aligned} z_k &= x_{2k+1} + x_{N-2k-1} && 0 \leq k \leq N/4-1, \\ w_k &= (-1)^k (x_{N-2k-1} - x_{2k+1}) && 0 \leq k \leq N/4-1, \end{aligned}$$

Explain from this how you can make an algorithm which reduces an FFT of length N to an FFT of length $N/2$ (on $\boldsymbol{x}^{(e)}$), and two DCT's of length $N/4$ (on $\boldsymbol{z}$ and $\boldsymbol{w}$). We will call this algorithm the revised FFT algorithm.

a) says nothing about the coefficients y_n for $n > \frac{N}{2}$. These are obtained in the same way as before through symmetry. a) also says nothing about $y_{N/2}$. This can be obtained with the same formula as in Theorem 2.15.

Let us now compute the number of arithmetic operations our revised algorithm needs. Denote by the number of real multiplications needed by the revised N-point FFT algorithm

b) Explain from the algorithm in a) that

$$M_N = 2(M_{N/4} + 3N/8) + M_{N/2} \quad A_N = 2(A_{N/4} + 3N/8) + A_{N/2} + 3N/2 \tag{4.15}$$

Hint. $3N/8$ should come from the extra additions/multiplications (see Exercise 4.8) you need to compute when you run the algorithm from Theorem 4.22 for $C_{N/4}$. Note also that the equations in a) require no extra multiplications, but that there are xix equations involved, each needing $N/4$ additions, so that we need $6N/4 = 3N/2$ extra additions.

c) Explain why $x_r = M_{2^r}$ is the solution to the difference equation

$$x_{r+2} - x_{r+1} - 2x_r = 3 \times 2^r,$$

and that $x_r = A_{2^r}$ is the solution to

$$x_{r+2} - x_{r+1} - 2x_r = 9 \times 2^r.$$

and show that the general solution to these are $x_r = \frac{1}{2}r2^r + C2^r + D(-1)^r$ for multiplications, and $x_r = \frac{3}{2}r2^r + C2^r + D(-1)^r$ for additions.

d) Explain why, regardless of initial conditions to the difference equations, $M_N = O\left(\frac{1}{2}N\log_2 N\right)$ and $A_N = O\left(\frac{3}{2}N\log_2 N\right)$ both for the revised FFT and the revised DCT. The total number of operations is thus $O(2N\log_2 N)$, i.e. half the operation count of the split-radix algorithm. The orders of these algorithms are thus the same, since we here have adapted to read data.

e) Explain that, if you had not employed the trick from Exercise 4.8, we would instead have obtained $M_N = O\left(\frac{2}{3}\log_2 N\right)$, and $A_N = O\left(\frac{4}{3}\log_2 N\right)$, which equal the orders for the number of multiplications/additions for the split-radix algorithm. In particular, the order of the operation count remains the same, but the trick from Exercise 4.8 turned a bigger percentage of the arithmetic operations into additions.

The algorithm we here have developed thus is constructed from the beginning to apply for real data only. Another advantage of the new algorithm is that it can be used to compute both the DCT and the DFT.

Exercise 4.10: Implementation of the IFFT/IDCT

We did not write down corresponding algorithms for the revised IFFT and IDCT algorithms. We will consider this in this exercise.

a) Using equations (4.13)-(4.14), show that

$$\begin{aligned}\Re(y_n) - \Re(y_{N/2-n}) &= 2(C_{N/4}\boldsymbol{z})_n \\ \Im(y_n) + \Im(y_{N/2-n}) &= 2(C_{N/4}\boldsymbol{w})_{N/4-n}\end{aligned}$$

for $1 \leq n \leq N/4 - 1$. Explain how one can compute $\boldsymbol{z}$ and $\boldsymbol{w}$ from this using two IDCT's of length $N/4$.

b) Using equations (4.13)-(4.14), show that

$$\begin{aligned}\Re(y_n) + \Re(y_{N/2-n}) &= \Re((\mathrm{DFT}_{N/2}\boldsymbol{x}^{(e)})_n) \\ \Im(y_n) - \Im(y_{N/2-n}) &= \Im((\mathrm{DFT}_{N/2}\boldsymbol{x}^{(e)})_n),\end{aligned}$$

and explain how one can compute $\boldsymbol{x}^{(e)}$ from this using an IFFT of length $N/2$.

4.4 Summary

We started this chapter by extending a previous result which had to do with that the Fourier series of a symmetric function converged quicker. To build on this we first needed to define symmetric extensions of vectors and symmetric vectors, before we classified symmetric extensions in the frequency domain. From this we could find a nice, orthonormal basis for the symmetric extensions, which lead us to the definition of the DCT. We also saw a connection with symmetric filters: These are exactly the filters which preserve symmetric extensions, and we could characterize symmetric filters restricted to symmetric extension as an N-dimensional mapping. We also showed that it is smart to replace the DFT with the DCT when we work with filters which are known to be symmetric. Among other things, this lead to a better way of approximating analog filters, and better interpolation of functions.

We also showed how to obtain an efficient implementation of the DCT, which could reuse the FFT implementation. The DCT has an important role in the MP3 standard. As we have explained, the MP3 standard applies several filters to the sound, in order to split it into bands concentrating on different frequency ranges. Later we will look closer at how these filters can be implemented and constructed. The implementation can use transforms similar to the MDCT, as explained in Exercise 4.6. The MDCT is also used in the more advanced version of the MP3 standard (layer III). Here it is applied to the filtered data to obtain a higher spectral resolution of the sound. The MDCT is applied to groups of 576 (in special circumstances 192) samples. The MP3 standard document [6] does not dig into the theory for this, only representing what is needed in order to make an implementation. It is somewhat difficult to read this document, since it is written in quite a different language, familiar mainly to those working with international standards.

The different type of cosine-matrices can all be associated with some extension strategy for the signal. [10] contains a review of these.

The DCT is particularly popular for processing sound data before they are compressed with lossless techniques such as Huffman coding or arithmetic coding. The reason is, as mentioned, that the DCT provides a better approximation from a low-dimensional space than the DFT does, and that it has a very efficient implementation. Libraries exist which goes into lengths to provide efficient implementation of the FFT and the DCT. FFTW, short for *Fastest Fourier Transform in the West* [5], is perhaps the best known of these.

Signal processing literature often does not motivate digital filters in explaining where they come from, and where the input to the filters come from. Using analog filters to motivate this, and to argue for improvements in using the DCT and symmeric extensions, is not that common. Much literature simply says that the property of linear phase is good, without elaborating on this further.