Fourier theory and wavelet analysis

Øyvind Ryan and Knut Mørken

January 11, 2013

Contents

Preface

These notes deal with the topics Fourier analysis, signal processing, and wavelet theory, and applications related to these. A central idea in the notes is to explain the connections

theory – numerical methods – applications

In other words, we want to explain the theoretical foundations for the selected topics, but also to go from there to numerical methods and, and finally motivate these by their applications in diverse fields.

When it comes to the theoretical foundations, the common denominator between the presented topics is first of all linear algebra. We build on a basic background in linear algebra, and state and prove several results in linear algebra. Together with a book from an elementary course in linear algebra, the notes should be self-contained as it depends on only a minimum of source prerequisites. Recommended sources in this respect are given. In this respect the notes fulfill a major gap when compared to much existing literature, much of which are not self-contained. Only theoretical parts which are needed for the applications we present are included.

When it comes to the applications, they have been carefully chosen applications of linear algebra, like compression of sound and images. The applications explain how the presented theory can be adapted in order for it to be used in practice. This also includes interfaces towards modern programming languages, and how the applications can use modern computer architectures efficiently. These notes go longer than existing textbooks in fulfilling and combining the three purposes mentioned above. Many books on linear algebra sneak in words like "applied" or "applications" in their title. The main contents in most of these books may still be theory, and particular applications where the presented theory is used are perhaps only mentioned superficially, without digging deep enough to explain how these applications use the presented theory. For these notes applications is meant in a wider sense: we also unveil how we can turn the theory into practical implementations of the applications. By "Practical implementation" we do not mean a "full implementation", which typically involves many other components, unrelated or only weakly related to the theory we concentrate on.

One goal of these notes is to make the reader, in addition to understanding the theory, know enough about how the theory so that he can be operational enough to implement or reconstruct parts of the applications he learns about. Since many textbooks do not present the theory in such a way that this is possible, some further developmentments in the theory had to be made in order to meet this. These developments are not seen in new results, but rather in simplifications of existing results so that they are presentable in our setting. International standards which are presented, and attempted to make the reader operational on are the JPEG- and JPEG2000-standards for images, and the MP3-standard for compression of sound.

The notes are split into two parts, one on Fourier analysis, and one on wavelets. They have strong connections with the field of *signal processing*. Although these notes have many applications to sound and images in common with signal processing textbooks, they also differ from such textbooks in other respects. First of all, these notes are expressed in a language which is compatible with linear algebra textbooks. Such compatibility is not always the case in signal processing textbooks, which is unfortunate since the language of linear algebra is very powerful and natural to use. Also, signal processing literature requires the reader to be familiar with signal processing nomenclature. This requirement has been minimized in these notes by stating things in terms of linear algebra whenever possible. Also, we include a translation guide, in order to make a student of signal processing find his way in linear algebra concepts.

Notation

We will follow linear algebra notation as you know it from classical linear algebra textbooks. In particular, vectors will be in boldface ($\boldsymbol{x}$, $\boldsymbol{y}$, etc.), while matrices will be in uppercase (A, B, etc.). The zero vector, or the zero matrix, is denoted by $\mathbf{0}$. All vectors stated will be assumed to be column vectors. A row vector will always be written as $\boldsymbol{x}^T$, where $\boldsymbol{x}$ is a (column) vector.

Acknowledgment

The notes were written for the course MAT-INF2360, Applications of Linear Algebra, at UiO. The authors would like to thank each other for establishing the notes. The authors would also like to thank Andreas Våvang Solbrå for his valuable contributions to the notes.

Knut Mørken, and Øyvind Ryan
Oslo, January 2013.

Part I

Fourier analysis and applications to sound processing

Chapter 1

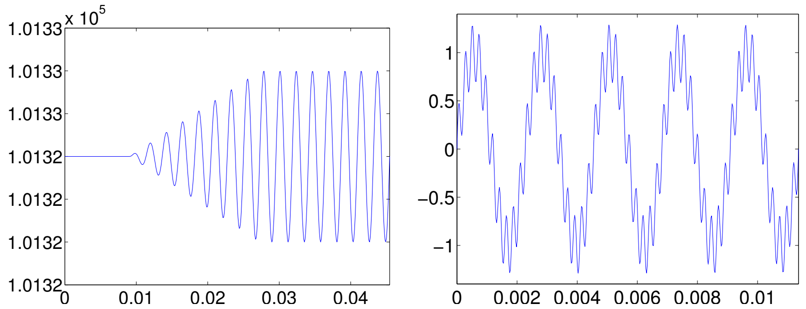
Sound and Fourier series

A major part of the information we receive and perceive every day is in the form of audio. Most sounds are transferred directly from the source to our ears, like when we have a face to face conversation with someone or listen to the sounds in a forest or a street. However, a considerable part of the sounds are generated by loudspeakers in various kinds of audio machines like cell phones, digital audio players, home cinemas, radios, television sets and so on. The sounds produced by these machines are either generated from information stored inside, or electromagnetic waves are picked up by an antenna, processed, and then converted to sound. It is this kind of sound we are going to study in this chapter. The sound that is stored inside the machines or picked up by the antennas is usually represented as *digital sound*. This has certain limitations, but at the same time makes it very easy to manipulate and process the sound on a computer.

What we perceive as sound corresponds to the physical phenomenon of slight variations in air pressure near our ears. Larger variations mean louder sounds, while faster variations correspond to sounds with a higher pitch. The air pressure varies continuously with time, but at a given point in time it has a precise value. This means that sound can be considered to be a mathematical function.

Observation 1.1. A sound can be represented by a mathematical function, with time as the free variable. When a function represents a sound, it is often referred to as a *continuous sound*.

In the following we will briefly discuss the basic properties of sound: first the significance of the size of the variations, and then how many variations there are per second, the *frequency* of the sound. We also consider the important fact that any reasonable sound may be considered to be built from very simple basis sounds. Since a sound may be viewed as a function, the mathematical equivalent of this is that any decent function may be constructed from very simple basis functions. Fourier-analysis is the theoretical study of this, and in the last part of this chapter we establish the framework for this study, and analyze this on



(a) A sound shown in terms of air pressure (b) A sound shown in terms of the difference from the ambient air pressure

Figure 1.1: Two examples of audio signals.

some examples for sound.

1.1 Loudness: Sound pressure and decibels

An example of a simple sound is shown in Figure 1.1(a) where the oscillations in air pressure are plotted agains time. We observe that the initial air pressure has the value 101 325 (we will shortly return to what unit is used here), and then the pressure starts to vary more and more until it oscillates regularly between the values 101 323 and 101 327. In the area where the air pressure is constant, no sound will be heard, but as the variations increase in size, the sound becomes louder and louder until about time $t = 0.6$ where the size of the oscillations becomes constant. The following summarises some basic facts about air pressure.

Fact 1.2 (Air pressure)**.** Air pressure is measured by the SI-unit Pa (Pascal) which is equivalent to N/m^2 (force / area). In other words, 1 Pa corresponds to the force exerted on an area of 1 m^2 by the air column above this area. The normal air pressure at sea level is 101 325 Pa.

Fact 1.2 explains the values on the vertical axis in Figure 1.1(a): The sound was recorded at the normal air pressure of 101 325 Pa. Once the sound started, the pressure started to vary both below and above this value, and after a short transient phase the pressure varied steadily between 101 324 Pa and 101 326 Pa, which corresponds to variations of size 1 Pa about the fixed value. Everyday sounds typically correspond to variations in air pressure of about 0.00002–2 Pa, while a jet engine may cause variations as large as 200 Pa. Short exposure to variations of about 20 Pa may in fact lead to hearing damage. The volcanic eruption at Krakatoa, Indonesia, in 1883, produced a sound wave with variations as large as almost 100 000 Pa, and the explosion could be heard 5000 km away.

When discussing sound, one is usually only interested in the variations in air pressure, so the ambient air pressure is subtracted from the measurement. This corresponds to subtracting 101 325 from the values on the vertical axis in Figure 1.1(a). In Figure 1.1(b) the subtraction has been performed for another sound, and we see that the sound has a slow, cos-like, variation in air pressure, with some smaller and faster variations imposed on this. This combination of several kinds of systematic oscillations in air pressure is typical for general sounds. The size of the oscillations is directly related to the loudness of the sound. We have seen that for audible sounds the variations may range from 0.00002 Pa all the way up to 100 000 Pa. This is such a wide range that it is common to measure the loudness of a sound on a logarithmic scale. Often air pressure is normalized so that it lies between -1 and 1: The value 0 then represents the ambient air pressure, while -1 and 1 represent the lowest and highest representable air pressure, respectively. The following fact box summarises the previous discussion of what a sound is, and introduces the logarithmic decibel scale.

Fact 1.3 (Sound pressure and decibels)**.** The physical origin of sound is variations in air pressure near the ear. The *sound pressure* of a sound is obtained by subtracting the average air pressure over a suitable time interval from the measured air pressure within the time interval. A square of this difference is then averaged over time, and the sound pressure is the square root of this average.

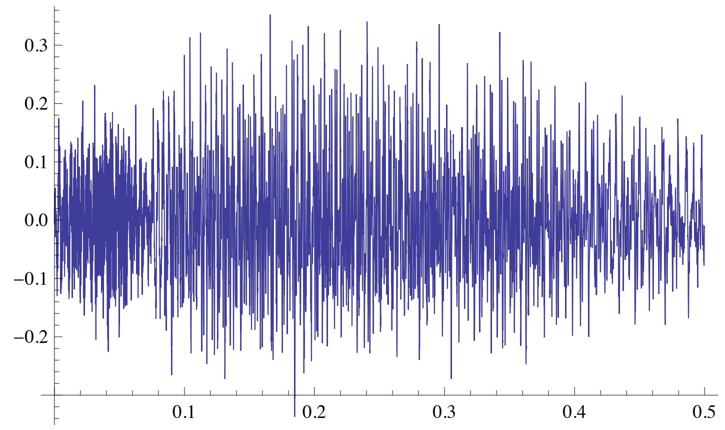
It is common to relate a given sound pressure to the smallest sound pressure that can be perceived, as a level on a decibel scale,

$$L_p = 10\log_{10}\left(\frac{p^2}{p_{\text{ref}}^2}\right) = 20\log_{10}\left(\frac{p}{p_{\text{ref}}}\right).$$

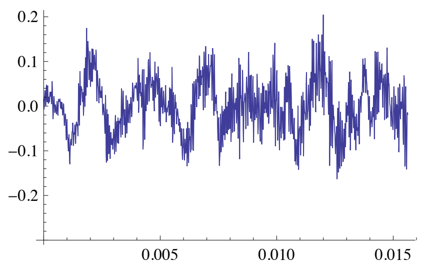
Here p is the measured sound pressure while p_{ref} is the sound pressure of a just perceivable sound, usually considered to be 0.00002 Pa.

The square of the sound pressure appears in the definition of L_p since this represents the *power* of the sound which is relevant for what we perceive as loudness.

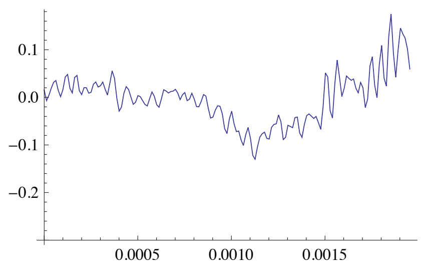
The sounds in Figure 1.1 are synthetic in that they were constructed from mathematical formulas (see Exercises 2.2.1 and 2.2.2). The sounds in Figure 1.2 on the other hand show the variation in air pressure when there is no mathematical formula involved, such as is the case for a song. In (a) there are so many oscillations that it is impossible to see the details, but if we zoom in as in (c) we can see that there is a continuous function behind all the ink. It is important to realise that in reality the air pressure varies more than this, even over the short time period in (c). However, the measuring equipment was not able to pick up those variations, and it is also doubtful whether we would be able to perceive such rapid variations.



(a) 0.5 seconds of the song



(b) the first 0.015 seconds



(c) the first 0.002 seconds

Figure 1.2: Variations in air pressure during parts of a song.

Exercises for Section 1.1

1. Compute the loudness of the Krakatoa explosion on the decibel scale, assuming that the variation in air pressure peaked at 100 000 Pa.

1.2 The pitch of a sound

Besides the size of the variations in air pressure, a sound has another important characteristic, namely the frequency (speed) of the variations. For most sounds the frequency of the variations varies with time, but if we are to perceive variations in air pressure as sound, they must fall within a certain range.

Fact 1.4. For a human with good hearing to perceive variations in air pressure as sound, the number of variations per second must be in the range 20–20 000.

To make these concepts more precise, we first recall what it means for a function to be periodic.

Definition 1.5. A real function f is said to be periodic with period τ if

$$f(t+\tau) = f(t)$$

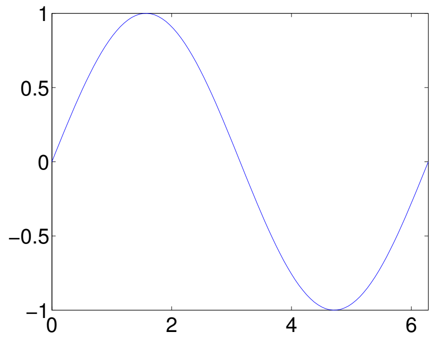
for all real numbers t.

Note that all the values of a periodic function f with period τ are known if $f(t)$ is known for all t in the interval $[0, \tau)$. The prototypes of periodic functions are the trigonometric ones, and particularly $\sin t$ and $\cos t$ are of interest to us. Since $\sin(t + 2\pi) = \sin t$, we see that the period of $\sin t$ is 2π and the same is true for $\cos t$.

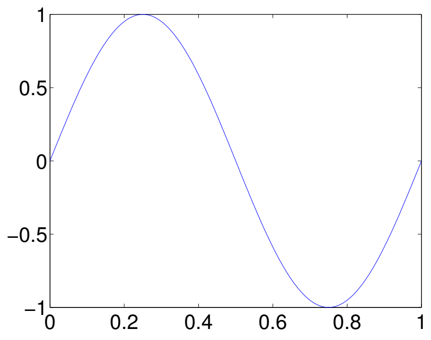
There is a simple way to change the period of a periodic function, namely by multiplying the argument by a constant.

Observation 1.6 (Frequency)**.** If ν is an integer, the function $f(t) = \sin(2\pi\nu t)$ is periodic with period $\tau = 1/\nu$. When t varies in the interval $[0, 1]$, this function covers a total of ν periods. This is expressed by saying that f has *frequency* ν.

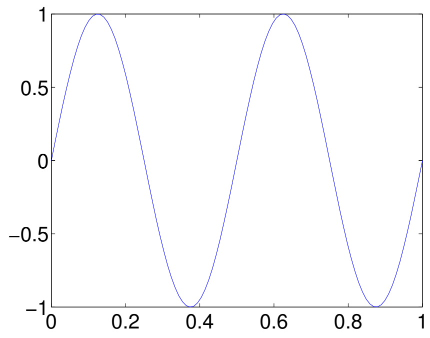
Figure 1.3 illustrates observation 1.6. The function in (a) is the plain $\sin t$ which covers one period when t varies in the interval $[0, 2\pi]$. By multiplying the argument by 2π, the period is squeezed into the interval $[0, 1]$ so the function $\sin(2\pi t)$ has frequency $\nu = 1$. Then, by also multiplying the argument by 2, we push two whole periods into the interval $[0, 1]$, so the function $\sin(2\pi 2t)$ has frequency $\nu = 2$. In (d) the argument has been multiplied by 5 — hence the frequency is 5 and there are five whole periods in the interval $[0, 1]$. Note that any function on the form $\sin(2\pi\nu t + a)$ has frequency ν, regardless of the value of a.



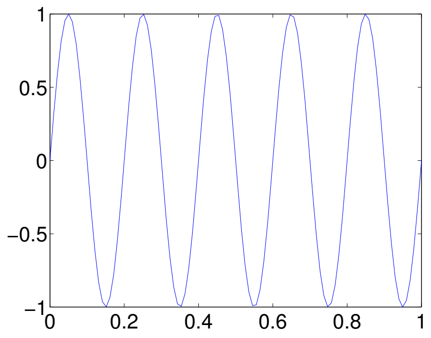
(a) $\sin t$



(b) $\sin(2\pi t)$



(c) $\sin(2\pi 2t)$



(d) $\sin(2\pi 5t)$

Figure 1.3: Versions of sin with different frequencies.

Since sound can be modelled by functions, it is reasonable to say that a sound with frequency ν is a trigonometric function with frequency ν.

Definition 1.7. The function $\sin(2\pi\nu t)$ represents what we will call a pure tone with frequency ν. Frequency is measured in Hz (Herz) which is the same as s^{-1} (the time t is measured in seconds).

A pure tone with frequency 440 Hz sounds like this, and a pure tone with frequency 1500 Hz sounds like this. In Section 2.1 we will explain how we generated these sounds so that they could be played on a computer.

Any sound may be considered to be a function. In the next section we will explain why any reasonable function may be written as a sum of simple sin- and cos- functions with integer frequencies. When this is translated into properties of sound, we obtain an important principle.

Observation 1.8 (Decomposition of sound into pure tones)**.** Any sound f is a sum of pure tones at different frequencies. The amount of each frequency required to form f is the frequency content of f. Any sound can be reconstructed from its frequency content.

The most basic consequence of observation 1.8 is that it gives us an understanding of how any sound can be built from the simple building blocks of pure tones. This also means that we can store a sound f by storing its frequency content, as an alternative to storing f itself. This also gives us a possibility for lossy compression of digital sound: It turns out that, in a typical audio signal, most information is found in the lower frequencies, and some frequencies will be almost completely absent. This can be exploited for compression if we change the frequencies with small contribution a little bit and set them to 0, and then store the signal by only storing the nonzero part of the frequency content. When the sound is to be played back, we first convert the adjusted values to the adjusted frequency content back to a normal function representation with an inverse mapping.

Fact 1.9 (Basic idea behind audio compression)**.** Suppose an audio signal f is given. To compress f, perform the following steps:

1. Rewrite the signal f in a new format where frequency information becomes accessible.
2. Remove those frequencies that only contribute marginally to human perception of the sound.
3. Store the resulting sound by coding the adjusted frequency content with some lossless coding method.

This lossy compression strategy is essentially what is used in practice by commercial audio formats. The difference is that commercial software does everything in a more sophisticated way and thereby gets better compression rates. We will return to this in later chapters.

We will see later that Observation 1.8 also is the basis for many operations on sound. The same observation also makes it possible to explain more precisely what it means that we only perceive sounds with a frequency in the range 20–20000 Hz:

Fact 1.10. Humans can only perceive variations in air pressure as sound if the Fourier series of the sound signal contains at least one sufficiently large term with frequency in the range 20–20 000 Hz.

With appropriate software it is easy to generate a sound from a mathematical function; we can 'play' the function. If we play a function like $\sin(2\pi 440t)$, we hear a pleasant sound with a very distinct pitch, as expected. There are, however, many other ways in which a function can oscillate regularly. The function in Figure 1.1(b) for example, definitely oscillates 2 times every second, but it does not have frequency 2 Hz since it is not a pure tone. This sound is also not that pleasant to listen to. We will consider two more important examples of this, which are very different from smooth, trigonometric functions.

Example 1.11. We define the *square wave* of period T as the function which repeats with period T, and is 1 on the first half of each period, and -1 on the second half. This means that we can define it as the function

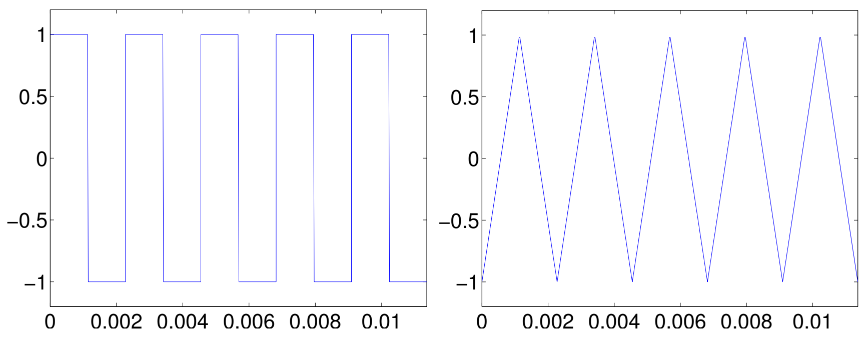
$$f_s(t) = \begin{cases} 1, & \text{if } 0 \le t < T/2; \\ -1, & \text{if } T/2 \le t < T. \end{cases} \tag{1.1}$$

In Figure 1.4(a) we have plotted the square wave when $T = 1/440$. This period is chosen so that it corresponds to the pure tone we already have listened to, and you can listen to this square wave here. In Exercise 2.1.4 you will learn how to generate this sound. We hear a sound with the same pitch as $\sin(2\pi 440t)$, but note that the square wave is less pleasant to listen to: There seems to be some sharp corners in the sound, translating into a rather shrieking, piercing sound. We will later explain this by the fact that the square wave can be viewed as a sum of many frequencies, and that all the different frequencies pollute the sound so that it is not pleasant to listen to.

♣

Example 1.12. We define the *triangle wave* of period T as the function which repeats with period T, and increases linearly from -1 to 1 on the first half of each period, and decreases linearly from 1 to -1 on the second half of each period. This means that we can define it as the function

$$f_t(t) = \begin{cases} 4t/T - 1, & \text{if } 0 \le t < T/2; \\ 3 - 4t/T, & \text{if } T/2 \le t < T. \end{cases} \tag{1.2}$$



(a) The first five periods of the square wave (b) The first five periods of the triangle wave

Figure 1.4: The square wave and the triangle wave, two functions with regular oscillations, but which are not simple, trigonometric functions.

In Figure 1.4(b) we have plotted the triangle wave when $T = 1/440$. Again, this same choice of period gives us an audible sound, and you can listen to the triangle wave here. Again you will note that the triangle wave has the same pitch as $\sin(2\pi 440 t)$, and is less pleasant to listen to than this pure tone. However, one can argue that it is somewhat more pleasant to listen to than a square wave. This will also be explained in terms of pollution with other frequencies later. ♣

In Section 1.3 we will begin to peek behind the curtains as to why these waves sound so different, even though we recognize them as having the exact same pitch.

Exercises for Section 1.2

1. Consider a sum of two pure tones, $f(t) = A_1 \sin 2(\pi\nu_1 t) + A_2 \sin 2(\pi\nu_2 t)$. For which values of A_1, A_2, ν_1, ν_2 is f periodic? What is the period of f when it is periodic?

1.3 Fourier series: Basic concepts

In Section 1.2 we identified audio signals with functions and discussed informally the idea of decomposing a sound into basis sounds (pure sounds) to make its frequency content available. In this chapter we will make this kind of decomposition more precise by discussing how a given function can be expressed in terms of the basic trigonometric functions. This is similar to Taylor series where functions are approximated by combinations of polynomials. But it is also different from Taylor series because we use trigonometric series rather than power series, and the approximations are computed in a very different way. The theory of approximation of functions with trigonometric functions is generally refered to as *Fourier analysis*. This is a central tool in practical fields like image- and signal processing, but it is also an important field of research within pure mathematics.

In the start of this chapter we had no constraints on the function f. Although Fourier analysis can be performed for very general functions, it turns out that it takes its simplest form when we assume that the function is periodic. Periodic functions are fully known when we know their values on a period $[0, T]$. In this case we will see that we can carry out the Fourier analysis in finite dimensional vector spaces of functions. This makes linear algebra a very useful tool in Fourier analysis: Many of the tools from your linear algebra course will be useful, in a situation that at first may seem far from matrices and vectors.

The basic idea of Fourier series is to approximate a given function by a combination of simple cos and sin functions. This means that we have to address at least three questions:

1. How general do we allow the given function to be?
2. What exactly are the combinations of cos and sin that we use for the approximations?
3. How do we determine the approximation?

Each of these questions will be answered in this section. Since we restrict to periodic functions, we will without much loss of generality assume that the functions are defined on $[0, T]$, where T is some positive number. Mostly we will also assume that f is continuous, but the theory can also be extended to functions which are only Riemann-integrable, and more precisely, to square integrable functions.

Definition 1.13 (Continuous and square-integrable functions)**.** The set of continuous, real functions defined on an interval $[0, T]$ is denoted $C[0, T]$.

A real function f defined on $[0, T]$ is said to be square integrable if f^2 is Riemann-integrable, i.e., if the Riemann integral of f^2 on $[0, T]$ exists,

$$\int_0^T f(t)^2 \, dt < \infty.$$

The set of all square integrable functions on $[0, T]$ is denoted $L^2[0, T]$.

The sets of continuous and square-integrable functions can be equippped with an inner-product, a generalisation of the so-called dot-product for vectors.

Theorem 1.14. Both $L^2[0, T]$ and $C[0, T]$ are vector spaces. Moreover, if the two functions f and g lie in $L^2[0, T]$ (or in $C[0, T]$), then the product fg is also in $L^2[0,T]$ (or in $C[0,T]$). Moreover, both spaces are inner product spaces[1], with inner product[2] defined by

$$\langle f, g\rangle = \frac{1}{T}\int_0^T f(t)g(t)\,dt, \tag{1.3}$$

and associated norm

$$\|f\| = \sqrt{\frac{1}{T}\int_0^T f(t)^2 dt}. \tag{1.4}$$

The mysterious factor $1/T$ is included so that the constant function $f(t) = 1$ has norm 1, i.e., its role is as a normalizing factor.

Definition 1.13 and Theorem 1.14 answer the first question above, namely how general we allow our functions to be. Theorem 1.14 also gives an indication of how we are going to determine approximations—we are going to use inner products. We recall from linear algebra that the projection of a function f onto a subspace W with respect to an inner product $\langle\cdot,\cdot\rangle$ is the function $g \in W$ which minimizes $\|f - g\|$, also called *the error* in the approximation[3]. This projection is therefore also called a best approximation of f from W and is characterised by the fact that the function $f - g$, also called the *error function*, should be orthogonal to the subspace W, i.e. we should have

$$\langle f - g, h\rangle = 0, \quad \text{for all } h \in W.$$

More precisely, if $\boldsymbol{\phi} = \{\phi_i\}_{i=1}^m$ is an orthogonal basis for W, then the best approximation g is given by

$$g = \sum_{i=1}^m \frac{\langle f, \phi_i\rangle}{\langle \phi_i, \phi_i\rangle}\phi_i. \tag{1.5}$$

The error $\|f - g\|$ is often referred to as the *least square error*.

We have now answered the second of our primary questions. What is left is a description of the subspace W of trigonometric functions. This space is spanned by the pure tones we discussed in Section 1.2.

Definition 1.15 (Fourier series)**.** Let $V_{N,T}$ be the subspace of $C[0,T]$ spanned by the set of functions given by

$$\begin{aligned}\mathcal{D}_{N,T} = \{&1, \cos(2\pi t/T), \cos(2\pi 2t/T), \cdots, \cos(2\pi Nt/T),\\ &\sin(2\pi t/T), \sin(2\pi 2t/T), \cdots, \sin(2\pi Nt/T)\}.\end{aligned} \tag{1.6}$$

The space $V_{N,T}$ is called the N'th order Fourier space. The Nth-order Fourier series approximation of f, denoted f_N, is defined as the best approximation of f from $V_{N,T}$ with respect to the inner product defined by (1.3).

[3]See Section 6.3 in [8] for a review of projections and least squares approximations.

The space $V_{N,T}$ can be thought of as the space spanned by the pure tones of frequencies $1/T$, $2/T$, $\dots$, N/T, and the Fourier series can be thought of as linear combination of all these pure tones. From our discussion in Section 1.2, we should expect that if N is sufficiently large, $V_{N,T}$ can be used to approximate most sounds in real life. The approximation f_N of a sound f from a space $V_{N,T}$ can also serve as a compressed version if many of the coefficients can be set to 0 without the error becoming too big.

Note that all the functions in the set $\mathcal{D}_{N,T}$ are periodic with period T, but most have an even shorter period. More precisely, $\cos(2\pi nt/T)$ has period T/n, and frequency n/T. In general, the term *fundamental frequency* is used to denote the lowest frequency of a given periodic function.

Definition 1.15 characterises the Fourier series. The next lemma gives precise expressions for the coefficients.

Theorem 1.16. The set $\mathcal{D}_{N,T}$ is an orthogonal basis for $V_{N,T}$. In particular, the dimension of $V_{N,T}$ is $2N+1$, and if f is a function in $L^2[0,T]$, we denote by $a_0,\dots,a_N$ and $b_1,\dots,b_N$ the coordinates of f_N in the basis $\mathcal{D}_{N,T}$, i.e.

$$f_N(t) = a_0 + \sum_{n=1}^{N} \left(a_n \cos(2\pi nt/T) + b_n \sin(2\pi nt/T)\right). \tag{1.7}$$

The $a_0,\dots,a_N$ and $b_1,\dots,b_N$ are called the (real) Fourier coefficients of f, and they are given by

$$a_0 = \langle f, 1\rangle = \frac{1}{T}\int_0^T f(t)\,dt, \tag{1.8}$$

$$a_n = 2\langle f, \cos(2\pi nt/T)\rangle = \frac{2}{T}\int_0^T f(t)\cos(2\pi nt/T)\,dt \quad \text{for } n \geq 1, \tag{1.9}$$

$$b_n = 2\langle f, \sin(2\pi nt/T)\rangle = \frac{2}{T}\int_0^T f(t)\sin(2\pi nt/T)\,dt \quad \text{for } n \geq 1. \tag{1.10}$$

Proof. To prove orthogonality, assume first that $m \neq n$. We compute the inner product

$$\begin{aligned}
&\langle \cos(2\pi mt/T), \cos(2\pi nt/T)\rangle \\
&= \frac{1}{T}\int_0^T \cos(2\pi mt/T)\cos(2\pi nt/T)dt \\
&= \frac{1}{2T}\int_0^T (\cos(2\pi mt/T + 2\pi nt/T) + \cos(2\pi mt/T - 2\pi nt/T)) \\
&= \frac{1}{2T}\left[\frac{T}{2\pi(m+n)}\sin(2\pi(m+n)t/T) + \frac{T}{2\pi(m-n)}\sin(2\pi(m-n)t/T)\right]_0^T \\
&= 0.
\end{aligned}$$

Here we have added the two identities $\cos(x \pm y) = \cos x \cos y \mp \sin x \sin y$ together to obtain an expression for $\cos(2\pi mt/T)\cos(2\pi nt/T)dt$ in terms of $\cos(2\pi mt/T + 2\pi nt/T)$ and $\cos(2\pi mt/T - 2\pi nt/T)$. By testing all other combinations of sin and cos also, we obtain the orthogonality of all functions in $\mathcal{D}_{N,T}$ in the same way.

We find the expressions for the Fourier coefficients from the general formula (1.5). We first need to compute the following inner products of the basis functions,

$$\begin{aligned}
\langle \cos(2\pi mt/T), \cos(2\pi mt/T)\rangle &= \frac{1}{2}\\
\langle \sin(2\pi mt/T), \sin(2\pi mt/T)\rangle &= \frac{1}{2}\\
\langle 1, 1\rangle &= 1,
\end{aligned}$$

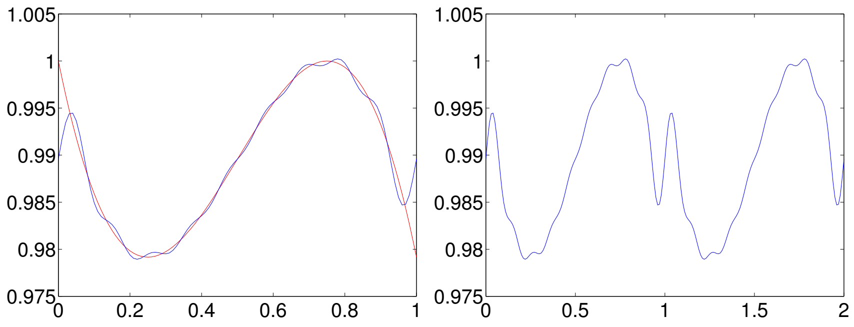
which are easily derived in the same way as above. The orthogonal decomposition theorem (1.5) now gives

$$\begin{aligned}
f_N(t) &= \frac{\langle f, 1\rangle}{\langle 1, 1\rangle}1 + + \sum_{n=1}^{N} \frac{\langle f, \cos(2\pi nt/T)\rangle}{\langle \cos(2\pi nt/T), \cos(2\pi nt/T)\rangle} \cos(2\pi nt/T)\\
&+ \sum_{n=1}^{N} \frac{\langle f, \sin(2\pi nt/T)\rangle}{\langle \sin(2\pi nt/T), \sin(2\pi nt/T)\rangle} \sin(2\pi nt/T)\\
&= \frac{\frac{1}{T}\int_0^T f(t)dt}{1} + \sum_{n=1}^{N} \frac{\frac{1}{T}\int_0^T f(t)\cos(2\pi nt/T)dt}{\frac{1}{2}} \cos(2\pi nt/T)\\
&+ \sum_{n=1}^{N} \frac{\frac{1}{T}\int_0^T f(t)\sin(2\pi nt/T)dt}{\frac{1}{2}} \sin(2\pi nt/T)\\
&= \frac{1}{T}\int_0^T f(t)dt + \sum_{n=1}^{N} \left(\frac{2}{T}\int_0^T f(t)\cos(2\pi nt/T)dt\right) \cos(2\pi nt/T)\\
&+ \sum_{n=1}^{N} \left(\frac{2}{T}\int_0^T f(t)\sin(2\pi nt/T)dt\right) \sin(2\pi nt/T).
\end{aligned}$$

The relations (1.8)- (1.10) now follow by comparison with (1.7). □

Since f is a function in time, and the a_n, b_n represent contributions from different frequencies, the Fourier series can be thought of as a change of coordinates, from what we vaguely can call the *time domain*, to what we can call the *frequency domain* (or *Fourier domain*). We will call the basis $\mathcal{D}_{N,T}$ the *N'th order Fourier basis* for $V_{N,T}$. We note that $\mathcal{D}_{N,T}$ is not an orthonormal basis; it is only orthogonal.

In the signal processing literature, Equation (1.7) is known as *the synthesis equation*, since the original function f is synthesized as a sum of trigonometric functions. Similarly, equations (1.8)- (1.10) are called *analysis equations*.



(a) The function and its Fourier series (b) The Fourier series on a larger interval

Figure 1.5: The cubic polynomial $f(x) = -\frac{1}{3}x^3 + \frac{1}{2}x^2 - \frac{3}{16}x + 1$ on the interval $[0, 1]$, together with its Fourier series approximation from $V_{9,1}$.

A major topic in harmonic analysis is to state conditions on f which guarantees the convergence of its Fourier series. We will not discuss this in detail here, since it turns out that, by choosing N large enough, any reasonable periodic function can be approximated arbitrarily well by its Nth-order Fourier series approximation. More precisely, we have the following result for the convergence of the Fourier series, stated without proof.

Theorem 1.17 (Convergence of Fourier series)**.** Suppose that f is periodic with period T, and that

1. f has a finite set of discontinuities in each period.
2. f contains a finite set of maxima and minima in each period.
3. $\int_0^T |f(t)|dt < \infty$.

Then we have that $\lim_{N\to\infty} f_N(t) = f(t)$ for all t, except at those points t where f is not continuous.

The conditions in Theorem 1.17 are called the *Dirichlet conditions* for the convergence of the Fourier series. They are just one example of conditions that ensure the convergence of the Fourier series. There also exist much more general conditions that secure convergence. These can require deep mathematical theory in order to prove, depending on the generality.

An illustration of Theorem 1.17 is shown in Figure 1.5 where the cubic polynomial $f(x) = -\frac{1}{3}x^3 + \frac{1}{2}x^2 - \frac{3}{16}x + 1$ is approximated by a 9th order Fourier series. The trigonometric approximation is periodic with period 1 so the approximation becomes poor at the ends of the interval since the cubic polynomial is not periodic. The approximation is plotted on a larger interval in Figure 1.5(b), where its periodicity is clearly visible.

Exercises for Section 1.3

1. Find a function f which is Riemann-integrable on $[0,T]$, and so that $\int_0^T f(t)^2 dt$ is infinite.

2. Given the two Fourier spaces V_{N_1,T_1}, V_{N_2,T_2}. Find necessary and sufficient conditions in order for $V_{N_1,T_1} \subset V_{N_2,T_2}$.

1.4 Examples of Fourier series

Let us compute the Fourier series of some interesting functions.

Example 1.18. Let us compute the Fourier coefficients of the square wave, as defined by Equation (1.1) in Example 1.11. If we first use Equation (1.8) we obtain

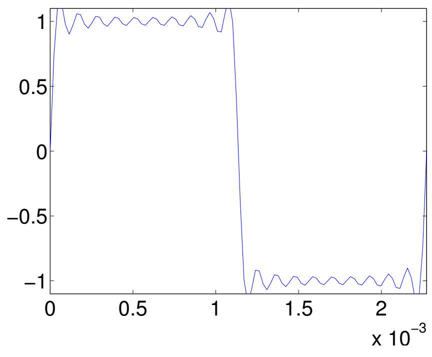
$$a_0 = \frac{1}{T}\int_0^T f_s(t)dt = \frac{1}{T}\int_0^{T/2} dt - \frac{1}{T}\int_{T/2}^T dt = 0.$$

Using Equation (1.9) we get

$$\begin{aligned}
a_n &= \frac{2}{T}\int_0^T f_s(t)\cos(2\pi nt/T)dt \\
&= \frac{2}{T}\int_0^{T/2} \cos(2\pi nt/T)dt - \frac{2}{T}\int_{T/2}^T \cos(2\pi nt/T)dt \\
&= \frac{2}{T}\left[\frac{T}{2\pi n}\sin(2\pi nt/T)\right]_0^{T/2} - \frac{2}{T}\left[\frac{T}{2\pi n}\sin(2\pi nt/T)\right]_{T/2}^T \\
&= \frac{2}{T}\frac{T}{2\pi n}((\sin(n\pi) - \sin 0) - (\sin(2n\pi) - \sin(n\pi)) = 0.
\end{aligned}$$

Finally, using Equation (1.10) we obtain

$$\begin{aligned}
b_n &= \frac{2}{T}\int_0^T f_s(t)\sin(2\pi nt/T)dt \\
&= \frac{2}{T}\int_0^{T/2} \sin(2\pi nt/T)dt - \frac{2}{T}\int_{T/2}^T \sin(2\pi nt/T)dt \\
&= \frac{2}{T}\left[-\frac{T}{2\pi n}\cos(2\pi nt/T)\right]_0^{T/2} + \frac{2}{T}\left[\frac{T}{2\pi n}\cos(2\pi nt/T)\right]_{T/2}^T \\
&= \frac{2}{T}\frac{T}{2\pi n}((-\cos(n\pi) + \cos 0) + (\cos(2n\pi) - \cos(n\pi))) \\
&= \frac{2(1-\cos(n\pi)}{n\pi} \\
&= \begin{cases} 0, & \text{if } n \text{ is even;} \\ 4/(n\pi), & \text{if } n \text{ is odd.} \end{cases}
\end{aligned}$$


(a) The Fourier series for $N = 20$

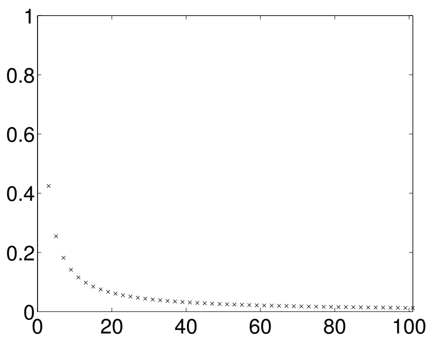

(b) Values of the first 100 Fourier coefficients b_n

Figure 1.6: The Fourier series of the square wave of Example 1.18

In other words, only the b_n-coefficients with n odd in the Fourier series are nonzero. The Fourier series of the square wave is thus

$$\frac{4}{\pi}\sin(2\pi t/T)+\frac{4}{3\pi}\sin(2\pi 3t/T)+\frac{4}{5\pi}\sin(2\pi 5t/T)+\frac{4}{7\pi}\sin(2\pi 7t/T)+\cdots . \quad (1.11)$$

With $N = 20$, there are 10 trigonometric terms in this sum. The corresponding Fourier series can be plotted on the same interval with the following code.

```
t=0:(1/fs):3;
y=zeros(1,length(t));
for n=1:2:19
  y = y + (4/(n*pi))*sin(2*pi*n*t/T);
end
plot(t,y)
```

In Figure 1.6(a) we have plotted the Fourier series of the square wave when $T = 1/440$, and when $N = 20$. In Figure 1.6(b) we have also plotted the values of the first 100 Fourier coefficients b_n, to see that they actually converge to zero. This is clearly necessary in order for the Fourier series to converge.

Even though f oscillates regularly between -1 and 1 with period T, the discontinuities mean that it is far from the simple $\sin(2\pi t/T)$ which corresponds to a pure tone of frequency $1/T$. From Figure 1.6(b) we see that the dominant coefficient in the Fourier series is b_1, which tells us how much there is of the pure tone $\sin(2\pi t/T)$ in the square wave. This is not surprising since the square wave oscillates T times every second as well, but the additional nonzero coefficients pollute the pure sound. As we include more and more of these coefficients, we gradually approach the square wave, as shown for $N = 20$.

There is a connection between how fast the Fourier coefficients go to zero, and how we perceive the sound. A pure sine sound has only one nonzero coefficient, while the square wave Fourier coefficients decrease as $1/n$, making the sound less pleasant. This explains what we heard when we listened to the sound in Example 1.11. Also, it explains why we heard the same pitch as the pure tone, since the first frequency in the Fourier series has the same frequency as the pure tone we listened to, and since this had the highest value.

Let us listen to the Fourier series approximations of the square wave. For $N = 1$ and with $T = 1/440$ as above, it sounds like this. This sounds exactly like the pure sound with frequency 440Hz, as noted above. For $N = 5$ the Fourier series approximation sounds like this, and for $N = 9$ it sounds like this. Indeed, these sounds are more like the square wave itself, and as we increase N we can hear how the introduction of more frequencies gradually pollutes the sound more and more. In Exercise 2.1.5 you will be asked to write a program which verifies this. ♣

Example 1.19. Let us also compute the Fourier coefficients of the triangle wave, as defined by Equation (1.2) in Example 1.12. We now have

$$a_0 = \frac{1}{T}\int_0^{T/2} \frac{4}{T}\left(t - \frac{T}{4}\right) dt + \frac{1}{T}\int_{T/2}^{T} \frac{4}{T}\left(\frac{3T}{4} - t\right) dt.$$

Instead of computing this directly, it is quicker to see geometrically that the graph of f_t has as much area above as below the x-axis, so that this integral must be zero. Similarly, since f_t is symmetric about the midpoint $T/2$, and $\sin(2\pi nt/T)$ is antisymmetric about $T/2$, we have that $f_t(t)\sin(2\pi nt/T)$ also is antisymmetric about $T/2$, so that

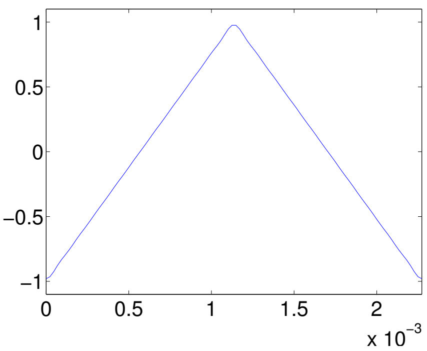
$$\int_0^{T/2} f_t(t)\sin(2\pi nt/T)dt = -\int_{T/2}^{T} f_t(t)\sin(2\pi nt/T)dt.$$

This means that, for $n \geq 1$,

$$b_n = \frac{2}{T}\int_0^{T/2} f_t(t)\sin(2\pi nt/T)dt + \frac{2}{T}\int_{T/2}^{T} f_t(t)\sin(2\pi nt/T)dt = 0.$$

For the final coefficients, since both f and $\cos(2\pi nt/T)$ are symmetric about $T/2$, we get for $n \geq 1$,

$$\begin{aligned}
a_n &= \frac{2}{T}\int_0^{T/2} f_t(t)\cos(2\pi nt/T)dt + \frac{2}{T}\int_{T/2}^{T} f_t(t)\cos(2\pi nt/T)dt \\
&= \frac{4}{T}\int_0^{T/2} f_t(t)\cos(2\pi nt/T)dt = \frac{4}{T}\int_0^{T/2} \frac{4}{T}\left(t - \frac{T}{4}\right)\cos(2\pi nt/T)dt \\
&= \frac{16}{T^2}\int_0^{T/2} t\cos(2\pi nt/T)dt - \frac{4}{T}\int_0^{T/2} \cos(2\pi nt/T)dt \\
&= \frac{4}{n^2\pi^2}(\cos(n\pi) - 1) \\
&= \begin{cases} 0, & \text{if } n \text{ is even;} \\ -8/(n^2\pi^2), & \text{if } n \text{ is odd.} \end{cases}
\end{aligned}$$


(a) The Fourier series for $N = 20$

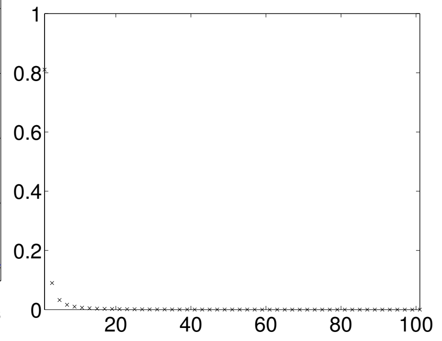

(b) Values of the first 100 Fourier coefficients a_n

Figure 1.7: The Fourier series of the triangle wave of Example 1.19

where we have dropped the final tedious calculations (use integration by parts). From this it is clear that the Fourier series of the triangle wave is

$$-\frac{8}{\pi^2}\cos(2\pi t/T)-\frac{8}{3^2\pi^2}\cos(2\pi 3t/T)-\frac{8}{5^2\pi^2}\cos(2\pi 5t/T)-\frac{8}{7^2\pi^2}\cos(2\pi 7t/T)+\cdots. \tag{1.12}$$

In Figure 1.7 we have repeated the plots used for the square wave, for the triangle wave. As before, we have used $T = 1/440$. The figure clearly shows that the Fourier series coefficients decay much faster.

Let us also listen to different Fourier series approximations of the triangle wave. For $N = 1$ and with $T = 1/440$ as above, it sounds like this. Again, this sounds exactly like the pure sound with frequency 440Hz. For $N = 5$ the Fourier series approximation sounds like this, and for $N = 9$ it sounds like this. Again these sounds are more like the triangle wave itself, and as we increase N we can hear that the introduction of more frequencies pollutes the sound. However, since the triangle wave Fourier coefficients decrease as $1/n^2$ instead of $1/n$ as for the square wave, the sound is, although unpleasant due to pollution by many frequencies, not as unpleasant as the square wave. Also, it converges faster to the triangle wave itself, as also can be heard. In Exercise 2.1.5 you will be asked to write a program which verifies this. ♣

There is an important lesson to be learnt from the previous examples: Even if the signal is nice and periodic, it may not have a nice representation in terms of trigonometric functions. Thus, trigonometric functions may not be the best bases to use for expressing other functions. Unfortunately, many more such cases can be found, as the next example shows.

Example 1.20. Let us consider a periodic function which is 1 on $[0, T_0]$, but 0 is on $[T_0, T]$. This is a signal with short duration when T_0 is small compared to

T. We compute that $y_0 = T_0/T$, and

$$a_n = \frac{2}{T}\int_0^{T_0} \cos(2\pi nt/T)dt = \frac{1}{\pi n}\left[\sin(2\pi nt/T)\right]_0^{T_0} = \frac{\sin(2\pi nT_0/T)}{\pi n}$$

for $n \geq 1$. Similar computations hold for b_n. We see that $|a_n|$ is of the order $1/(\pi n)$, and that infinitely many n contribute, This function may be thought of as a simple building block, corresponding to a small time segment. However, we see that it is not a simple building block in terms of trigonometric functions. This time segment building block may be useful for restricting a function to smaller time segments, and later on we will see that it still can be useful. ♣

1.4.1 Fourier series for symmetric and antisymmetric functions

In Example 1.18 we saw that the Fourier coefficients b_n vanished, resulting in a sine-series for the Fourier series of the square wave. Similarly, in Example 1.19 we saw that a_n vanished, resulting in a cosine-series for the triangle wave. This is not a coincident, and is captured by the following result, since the square wave was defined so that it was antisymmetric about 0, and the triangle wave so that it was symmetric about 0.

Theorem 1.21 (Symmetry and antisymmetry)**.** If f is antisymmetric about 0 (that is, if $f(-t) = -f(t)$ for all t), then $a_n = 0$, so the Fourier series is actually a sine-series. If f is symmetric about 0 (which means that $f(-t) = f(t)$ for all t), then $b_n = 0$, so the Fourier series is actually a cosine-series.

Proof. Note first that we can write

$$a_n = \frac{2}{T}\int_{-T/2}^{T/2} f(t)\cos(2\pi nt/T)dt \qquad b_n = \frac{2}{T}\int_{-T/2}^{T/2} f(t)\sin(2\pi nt/T)dt,$$

i.e. we can change the integration bounds from $[0, T]$ to $[-T/2, T/2]$. This follows from the fact that all $f(t)$, $\cos(2\pi nt/T)$ and $\sin(2\pi nt/T)$ are periodic with period T.

Suppose first that f is symmetric. We obtain

$$\begin{aligned}
b_n &= \frac{2}{T}\int_{-T/2}^{T/2} f(t)\sin(2\pi nt/T)dt \\
&= \frac{2}{T}\int_{-T/2}^{0} f(t)\sin(2\pi nt/T)dt + \frac{2}{T}\int_{0}^{T/2} f(t)\sin(2\pi nt/T)dt \\
&= \frac{2}{T}\int_{-T/2}^{0} f(t)\sin(2\pi nt/T)dt - \frac{2}{T}\int_{0}^{-T/2} f(-t)\sin(-2\pi nt/T)dt \\
&= \frac{2}{T}\int_{-T/2}^{0} f(t)\sin(2\pi nt/T)dt - \frac{2}{T}\int_{-T/2}^{0} f(t)\sin(2\pi nt/T)dt = 0.
\end{aligned}$$

where we have made the substitution $u = -t$, and used that sin is antisymmetric. The case when f is antisymmetric can be proved in the same way, and is left as an exercise. $\square$

In fact, the connection between symmetric and antisymmetric functions, and sine- and cosine series can be made even stronger by observing the following:

1. Any cosine series $a_0 + \sum_{n=1}^N a_n \cos(2\pi nt/T)$ is a symmetric function.
2. Any sine series $\sum_{n=1}^N b_n \sin(2\pi nt/T)$ is an antisymmetric function.
3. Any periodic function can be written as a sum of a symmetric- and an antisymmetric function by writing
$$f(t) = \frac{f(t)+f(-t)}{2} + \frac{f(t)-f(-t)}{2}. \tag{1.13}$$
4. If $f_N(t) = a_0 + \sum_{n=1}^N (a_n \cos(2\pi nt/T) + b_n \sin(2\pi nt/T))$, then
$$\begin{aligned}\frac{f_N(t)+f_N(-t)}{2} &= a_0 + \sum_{n=1}^N a_n \cos(2\pi nt/T)\\ \frac{f_N(t)-f_N(-t)}{2} &= \sum_{n=1}^N b_n \sin(2\pi nt/T).\end{aligned}$$

Exercises for Section 1.4

1. Prove the second part of Theorem 1.21, i.e. show that if f is antisymmetric about 0 (i.e. $f(-t) = -f(t)$ for all t), then $a_n = 0$, i.e. the Fourier series is actually a sine-series.

2. Find the Fourier series coefficients of the periodic functions with period T defined by being $f(t) = t$, $f(t) = t^2$, and $f(t) = t^3$, on $[0, T]$.

3. Write down difference equations for finding the Fourier coefficients of $f(t) = t^{k+1}$ from those of $f(t) = t^k$, and write a program which uses this recursion. Use the program to verify what you computed in Exercise 2.

4. Use the previous exercise to find the Fourier series for $f(x) = -\frac{1}{3}x^3 + \frac{1}{2}x^2 - \frac{3}{16}x + 1$ on the interval $[0, 1]$. Plot the 9th order Fourier series for this function. You should obtain the plots from Figure 1.5.

1.5 Complex Fourier series

In Section 1.3 we saw how a function can be expanded in a series of sines and cosines. These functions are related to the complex exponential function via Eulers formula
$$e^{ix} = \cos x + i \sin x$$

where i is the imaginary unit with the property that $i^2 = -1$. Because the algebraic properties of the exponential function are much simpler than those of cos and sin, it is often an advantage to work with complex numbers, even though the given setting is real numbers. This is definitely the case in Fourier analysis. More precisely, we will make the substitutions

$$\cos(2\pi nt/T) = \frac{1}{2}\left(e^{2\pi int/T} + e^{-2\pi int/T}\right) \tag{1.14}$$

$$\sin(2\pi nt/T) = \frac{1}{2i}\left(e^{2\pi int/T} - e^{-2\pi int/T}\right) \tag{1.15}$$

in Definition 1.15. From these identities it is clear that the set of complex exponential functions $e^{2\pi int/T}$ also is a basis of periodic functions (with the same period) for $V_{N,T}$. We may therefore reformulate Definition 1.15 as follows:

Definition 1.22 (Complex Fourier basis)**.** We define the set of functions

$$\mathcal{F}_{N,T} = \{e^{-2\pi iNt/T}, e^{-2\pi i(N-1)t/T}, \cdots, e^{-2\pi it/T}, \tag{1.16}$$

$$1, e^{2\pi it/T}, \cdots, e^{2\pi i(N-1)t/T}, e^{2\pi iNt/T}\}, \tag{1.17}$$

and call this the order N complex Fourier basis for $V_{N,T}$.

The function $e^{2\pi int/T}$ is also called a pure tone with frequency n/T, just as sines and cosines are. We would like to show that these functions also are orthogonal. To show this, we need to say more on the inner product we have defined by Equation (1.3). A weakness with this definition is that we have assumed real functions f and g, so that this can not be used for the complex exponential functions $e^{2\pi int/T}$. For general complex functions we will extend the definition of the inner product as follows:

$$\langle f, g\rangle = \frac{1}{T}\int_0^T f\bar{g}\,dt. \tag{1.18}$$

The associated norm now becomes

$$\|f\| = \sqrt{\frac{1}{T}\int_0^T |f(t)|^2 dt}. \tag{1.19}$$

The motivation behind Equation 1.18, where we have conjugated the second function, lies in the definition of an *inner product for vector spaces over complex numbers*. From before we are used to vector spaces over real numbers, but vector spaces over complex numbers are defined through the same set of axioms as for real vector spaces, only replacing real numbers with complex numbers. For complex vector spaces, the axioms defining an inner product are the same as for real vector spaces, except for that the axiom

$$\langle f, g\rangle = \langle g, f\rangle \tag{1.20}$$

is replaced with the axiom

$$\langle f, g\rangle = \overline{\langle g, f\rangle}, \tag{1.21}$$

i.e. a conjugation occurs when we switch the order of the functions. This new axiom can be used to prove the property $\langle f, cg\rangle = \bar{c}\langle f, g\rangle$, which is a somewhat different property from what we know for real inner product spaces. This follows by writing

$$\langle f, cg\rangle = \overline{\langle cg, f\rangle} = \overline{c\langle g, f\rangle} = \bar{c}\overline{\langle g, f\rangle} = \bar{c}\langle f, g\rangle.$$

Clearly the inner product given by (1.18) satisfies Axiom (1.21). With this definition it is quite easy to see that the functions $e^{2\pi int/T}$ are orthonormal. Using the orthogonal decomposition theorem we can therefore write

$$\begin{aligned} f_N(t) &= \sum_{n=-N}^{N} \frac{\langle f, e^{2\pi int/T}\rangle}{\langle e^{2\pi int/T}, e^{2\pi int/T}\rangle} e^{2\pi int/T} = \sum_{n=-N}^{N} \langle f, e^{2\pi int/T}\rangle e^{2\pi int/T} \\ &= \sum_{n=-N}^{N} \left(\frac{1}{T}\int_0^T f(t)e^{-2\pi int/T}dt\right) e^{2\pi int/T}. \end{aligned}$$

We summarize this in the following theorem, which is a version of Theorem 1.16 which uses the complex Fourier basis:

Theorem 1.23. We denote by $y_{-N}, \ldots, y_0, \ldots, y_N$ the coordinates of f_N in the basis $\mathcal{F}_{N,T}$, i.e.

$$f_N(t) = \sum_{n=-N}^{N} y_n e^{2\pi int/T}. \tag{1.22}$$

The y_n are called the complex Fourier coefficients of f, and they are given by.

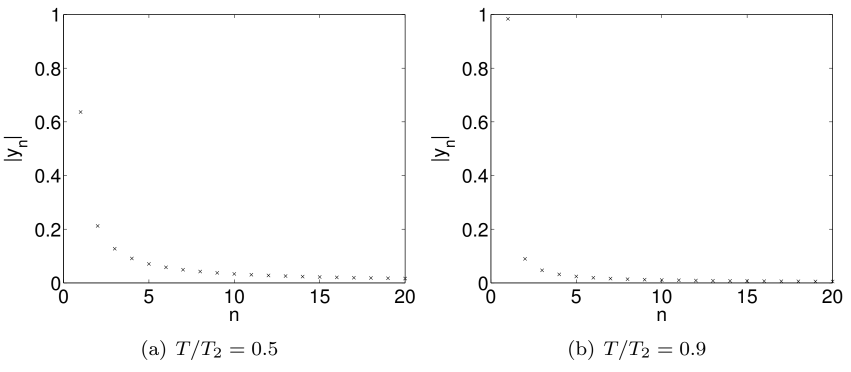
$$y_n = \langle f, e^{2\pi int/T}\rangle = \frac{1}{T}\int_0^T f(t)e^{-2\pi int/T}dt. \tag{1.23}$$

Let us consider some examples where we compute complex Fourier series.

Example 1.24. Let us consider the pure sound $f(t) = e^{2\pi it/T_2}$ with period T_2, but let us consider it only on the interval $[0, T]$ instead, where $T < T_2$. Note that this f is not periodic, since we only consider the part $[0, T]$ of the period $[0, T_2]$. The Fourier coefficients are

$$\begin{aligned} y_n &= \frac{1}{T}\int_0^T e^{2\pi it/T_2}e^{-2\pi int/T}dt = \frac{1}{2\pi iT(1/T_2 - n/T)}\left[e^{2\pi it(1/T_2 - n/T)}\right]_0^T \\ &= \frac{1}{2\pi i(T/T_2 - n)}\left(e^{2\pi iT/T_2} - 1\right). \end{aligned}$$

Here it is only the term $1/(T/T_2 - n)$ which depends on n, so that y_n can only be large when n is close T/T_2. In Figure 1.8 we have plotted $|y_n|$ for two different



(a) $T/T_2 = 0.5$ (b) $T/T_2 = 0.9$

Figure 1.8: Plot of $|y_n|$ when $f(t) = e^{2\pi it/T_2}$, and $T_2 > T$

combinations of T, T_2. In both examples it is seen that many Fourier coefficients contribute, but this is more visible when $T/T_2 = 0.5$. When $T/T_2 = 0.9$, most conribution is seen to be in the y_1-coefficient. This sounds reasonable, since f then is closest to the pure tone $f(t) = e^{2\pi it/T}$ of frequency $1/T$ (which in turn has $y_1 = 1$ and all other $y_n = 0$). ♣

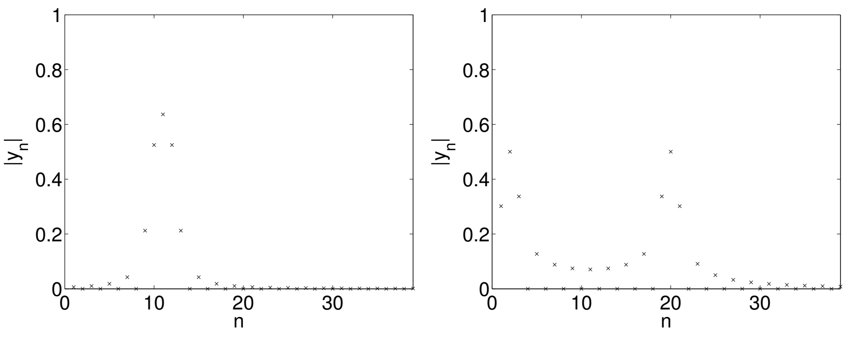
Apart from computing complex Fourier series, there is an important lesson to be learnt from the previous example: In order for a periodic function to be approximated by other periodic functions, their period must somehow match. Let us consider another example as well.

Example 1.25. What often is the case is that a sound changes in content over time. Assume that it is equal to a pure tone of frequency n_1/T on $[0, T/2)$, and equal to a pure tone of frequency n_2/T on $[T/2, T)$, i.e.

$$f(t) = \begin{cases} e^{2\pi i n_1 t/T} & \text{on } [0, T_2] \\ e^{2\pi i n_2 t/T} & \text{on}[T_2, T) \end{cases}.$$

When $n \neq n_1, n_2$ we have that

$$\begin{aligned} y_n &= \frac{1}{T}\left(\int_0^{T/2} e^{2\pi i n_1 t/T} e^{-2\pi i n t/T} dt + \int_{T/2}^{T} e^{2\pi i n_2 t/T} e^{-2\pi i n t/T} dt\right) \\ &= \frac{1}{T}\left(\left[\frac{T}{2\pi i(n_1 - n)} e^{2\pi i(n_1-n)t/T}\right]_0^{T/2} + \left[\frac{T}{2\pi i(n_2 - n)} e^{2\pi i(n_2-n)t/T}\right]_{T/2}^{T}\right) \\ &= \frac{e^{\pi i(n_1-n)} - 1}{2\pi i(n_1 - n)} + \frac{1 - e^{\pi i(n_2-n)}}{2\pi i(n_2 - n)}. \end{aligned}$$



(a) $n_1 = 10$, $n_2 = 12$ (b) $n_1 = 2$, $n_2 = 20$

Figure 1.9: Plot of $|y_n|$ when we have two different pure tones at the different parts of a period.

Let us restrict to the case when n_1 and n_2 are both even. We see that

$$y_n = \begin{cases} \frac{1}{2} + \frac{1}{\pi i(n_2-n_1)} & n = n_1, n_2 \\ 0 & n \text{ even } , n \neq n_1, n_2 \\ \frac{n_1-n_2}{\pi i(n_1-n)(n_2-n)} & n \text{ odd} \end{cases}$$

Here we have computed the cases $n = n_1$ and $n = n_2$ as above. In Figure 1.9 we have plotted $|y_n|$ for two different combinations of n_1, n_2. We see from the figure that, when n_1, n_2 are close, the Fourier coefficients are close to those of a pure tone with $n \approx n_1, n_2$, but that also other frequencies contribute. When n_1, n_2 are further apart, we see that the Fourier coefficients are like the sum of the two base frequencies, but that other frequencies contribute also here. ♣

There is an important lesson to be learnt from this as well: We should be aware of changes in a sound over time, and it may not be smart to use a frequency representation over a large interval when we know that there are simpler frequency representations on the smaller intervals.

If we reorder the real and complex Fourier bases so that the two functions $\{\cos(2\pi nt/T), \sin(2\pi nt/T)\}$ and $\{e^{2\pi int/T}, e^{-2\pi int/T}\}$ have the same index in the bases, equations (1.14)-(1.15) give us that the change of coordinates matrix[4] from $\mathcal{D}_{N,T}$ to $\mathcal{F}_{N,T}$, denoted $P_{\mathcal{F}_{N,T}\leftarrow\mathcal{D}_{N,T}}$, is represented by repeating the matrix

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

along the diagonal (with an additional 1 for the constant function 1). In other words, since a_n, b_n are coefficients relative to the real basis and y_n, y_{-n} the

[4]See Section 4.7 in [8], to review the mathematics behind change of coordinates.

corresponding coefficients relative to the complex basis, we have for $n > 0$,

$$\begin{pmatrix} y_n \\ y_{-n} \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & 1/i \\ 1 & -1/i \end{pmatrix} \begin{pmatrix} a_n \\ b_n \end{pmatrix}.$$

This can be summarized by the following theorem:

Theorem 1.26 (Change of coefficients between real and complex Fourier bases)**.** The complex Fourier coefficients y_n and the real Fourier coefficients a_n, b_n of a function f are related by

$$\begin{aligned} y_0 &= a_0, \\ y_n &= \frac{1}{2}(a_n - ib_n), \\ y_{-n} &= \frac{1}{2}(a_n + ib_n), \end{aligned}$$

for $n = 1, \ldots, N$.

Combining with Theorem 1.21, Theorem 1.26 can help us state properties of complex Fourier coefficients for symmetric- and antisymmetric functions. We look into this in Exercise 7.

Due to the somewhat nicer formulas for the complex Fourier coefficients when compared to the real Fourier coefficients, we will write most Fourier series in complex form in the following.

Exercises for Section 1.5

1. Show that the complex functions $e^{2\pi int/T}$ are orthonormal.

2. Repeat Exercise 1.3.2, computing the complex Fourier series instead of the real Fourier series.

3. In this exercise we will find a connection with certain Fourier series and the rows in Pascal's triangle.

a. Show that both $\cos^n t$ and $\sin^n t$ are in $V_{N,T}$,

b. Write down the N'th order complex Fourier series for $f_1(t) = \cos t$, $f_2(t) = \cos^2 t$, og $f_3(t) = \cos^3 t$.

c. In b. you should be able to see a connection between the Fourier coefficients and the three first rows in Pascal's triangle. Formulate and prove a general relationship between row n in Pascal's triangle and the Fourier coefficients of $f_n(t) = \cos^n t$.

4. Compute the complex Fourier coefficients of the square wave using Equation 1.23, i.e. repeat the calculations from Example 1.18 for the complex case. Use Theorem 1.26 to verify your result.

5. Repeat Exercise 4 for the triangle wave.

6. Use Equation (1.23) to compute the complex Fourier coefficients of the periodic functions with period T defined by, respectively, $f(t) = t$, $f(t) = t^2$, and $f(t) = t^3$, on $[0, T]$. Use Theorem 1.26 to verify your calculations from Exercise 1.4.2.

7. In this exercise we will prove a version of Theorem 1.21 for complex Fourier coefficients.

a. If f is symmetric about 0, show that y_n is real, and that $y_{-n} = y_n$.

b. If f is antisymmetric about 0, show that the y_n are purely imaginary, $y_0 = 0$, and that $y_{-n} = -y_n$.

c. Show that $\sum_{n=-N}^{N} y_n e^{2\pi i n t/T}$ is symmetric when $y_{-n} = y_n$ for all n, and rewrite it as a cosine-series.

d. Show that $\sum_{n=-N}^{N} y_n e^{2\pi i n t/T}$ is antisymmetric when $y_0 = 0$ and $y_{-n} = -y_n$ for all n, and rewrite it as a sine-series.

1.6 Rate of convergence for Fourier series

We have earlier mentioned criteria which guarantee that the Fourier series converges. Another important topic is the rate of convergence, given that it actually converges. If the series converges quickly, we may only need a few terms in the Fourier series to obtain a reasonable approximation. We have already seen examples which illustrate different convergence rates: The square wave seemed to have very slow convergence rate near the discontinuities, while the triangle wave did not seem to have the same problem.

Before discussing results concerning convergence rates we consider a simple lemma which will turn out to be useful.

Lemma 1.27. Assume that f is differentiable. Then $(f_N)'(t) = (f')_N(t)$. In other words, the derivative of the Fourier series equals the Fourier series of the derivative.

Proof. We first compute

$$\begin{aligned}
\langle f, e^{2\pi int/T}\rangle &= \frac{1}{T}\int_0^T f(t)e^{-2\pi int/T}dt \\
&= \frac{1}{T}\left(\left[-\frac{T}{2\pi in}f(t)e^{-2\pi int/T}\right]_0^T + \frac{T}{2\pi in}\int_0^T f'(t)e^{-2\pi int/T}dt\right) \\
&= \frac{T}{2\pi in}\frac{1}{T}\int_0^T f'(t)e^{-2\pi int/T}dt = \frac{T}{2\pi in}\langle f', e^{2\pi int/T}\rangle.
\end{aligned}$$

where we used integration by parts, and that $-\frac{T}{2\pi in}f(t)e^{-2\pi int/T}$ are periodic with period T. It follows that $\langle f, e^{2\pi int/T}\rangle = \frac{T}{2\pi in}\langle f', e^{2\pi int/T}\rangle$. From this we get that

$$\begin{aligned}
(f_N)'(t) &= \left(\sum_{n=-N}^{N}\langle f, e^{2\pi int/T}\rangle e^{2\pi int/T}\right)' = \frac{2\pi in}{T}\sum_{n=-N}^{N}\langle f, e^{2\pi int/T}\rangle e^{2\pi int/T} \\
&= \sum_{n=-N}^{N}\langle f', e^{2\pi int/T}\rangle e^{2\pi int/T} = (f')_N(t).
\end{aligned}$$

where we substituted the connection between the inner products we just found. □

Example 1.28. The connection between the Fourier series of the function and its derivative can be used to simplify the computation of Fourier series for new functions. Let us see how we can use this to compute the Fourier series of the triangle wave, which was quite a tedious job in Example 1.19. However, the relationship $f_t'(t) = \frac{4}{T}f_s(t)$ is straightforward to see from the plots of the square wave f_s and the triangle wave f_t. From this relationship and from Equation (1.11) for the Fourier series of the square wave it follows that

$$((f_t)')_N(t) = \frac{4}{T}\left(\frac{4}{\pi}\sin(2\pi t/T) + \frac{4}{3\pi}\sin(2\pi 3t/T) + \frac{4}{5\pi}\sin(2\pi 5t/T) + \cdots\right).$$

If we integrate this we obtain

$$(f_t)_N(t) = -\frac{8}{\pi^2}\left(\cos(2\pi t/T) + \frac{1}{3^2}\cos(2\pi 3t/T) + \frac{1}{5^2}\cos(2\pi 5t/T) + \cdots\right) + C.$$

What remains is to find the integration constant C. This is simplest found if we set $t = T/4$, since then all cosine terms are 0. Clearly then $C = 0$, and we arrive at the same expression as in Equation (1.12) for the Fourier series of the triangle wave. This approach clearly had less computations involved. There is a minor point here which we have not addressed: the triangle wave is not differentiable at two points, as required by Lemma 1.27. It is, however, not too difficult to see that this result still holds in cases where we have a finite number of nondifferentiable points only. ♣

We get the following corollary to Lemma 1.27:

Corollary 1.29. If the complex Fourier coefficients of f are y_n and f is differentiable, then the Fourier coefficients of $f'(t)$ are $\frac{2\pi i n}{T} y_n$.

If we turn this around, we note that the Fourier coefficients of $f(t)$ are $T/(2\pi i n)$ times those of $f'(t)$. If f is s times differentiable, we can repeat this argument to show that the Fourier coefficients of $f(t)$ are $\big(T/(2\pi i n)\big)^s$ times those of $f^{(s)}(t)$. In other words, the Fourier coefficients of a function which is many times differentiable decay to zero very fast.

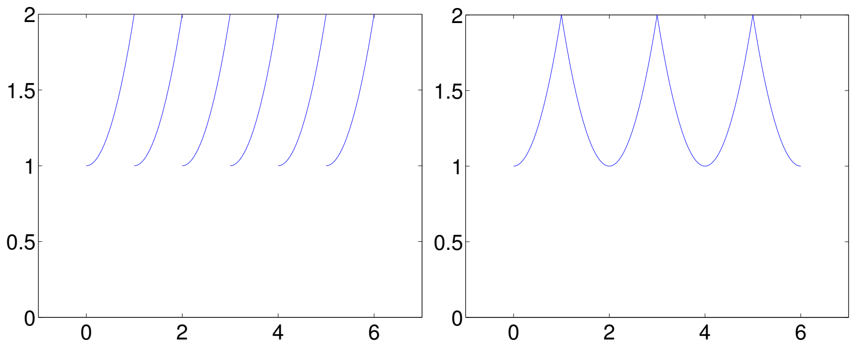
Observation 1.30. The Fourier series converges quickly when the function is many times differentiable.

An illustration is found in examples 1.18 and 1.19, where we saw that the Fourier series coefficients for the triangle wave converged more quickly to zero than those of the square wave. This is explained by the fact that the square wave is discontinuous, while the triangle wave is continuous with a discontinuous first derivative. Also, the functions considered in examples 1.24 and 1.25 are not continuous, which partially explain why we there saw contributions from many frequencies.

The requirement of continuity in order to obtain quickly converging Fourier series may seem like a small problem. However, often the function is not defined on the whole real line: it is often only defined on the interval $[0, T)$. If we extend this to a periodic function on the whole real line, by repeating one period as shown in Figure 1.10(a), there is no reason why the new function should be continuous at the boundaries $0, T, 2T$ etc., even though the function we started with may be continuous on $[0, T)$. This would require that $f(0) = \lim_{t\to T} f(t)$. If this does not hold, the function may not be well approximated with trigonometric functions, due to a slowly convergence Fourier series. We can therefore ask ourselves the following question:

Idea 1.31. Assume that f is continuous on $[0, T)$. Can we construct another periodic function which agrees with f on $[0, T]$, and which is both continuous and periodic (maybe with period different from T)?

If this is possible the Fourier series of the new function could produce better approximations for f. It turns out that the following extension strategy does the job:



(a) Perodic extension of f (b) Symmetric extension of f

Figure 1.10: Two different extensions of f to a periodic function on the whole real line

Definition 1.32 (Symmetric extension of a function)**.** Let f be a function defined on $[0, T]$. By the symmetric extension of f, denoted $\breve{f}$, we mean the function defined on $[0, 2T]$ by

$$\breve{f}(t) = \begin{cases} f(t), & \text{if } 0 \leq t \leq T; \\ f(2T - t), & \text{if } T < t \leq 2T. \end{cases}$$

Clearly the following holds:

Theorem 1.33. If f is continuous on $[0, T]$, then $\breve{f}$ is continuous on $[0, 2T]$, and $\breve{f}(0) = \breve{f}(2T)$. If we extend $\breve{f}$ to a periodic function on the whole real line (which we also will denote by $\breve{f}$), this function is continuous, agrees with f on $[0, T)$, and is a symmetric function.

This also means that the Fourier series of $\breve{f}$ is a cosine series, so that it is determined by the cosine-coefficients a_n. The symmetric extension of f is shown in Figure 1.10(b). $\breve{f}$ is symmetric since, for $0 \leq t \leq T$,

$$\breve{f}(-t) = \breve{f}(2T - t) = f(2T - (2T - t)) = f(t) = \breve{f}(t).$$

In summary, we now have two possibilities for approximating a function f defined only on $[0, T)$, where the latter addresses a shortcoming of the first:

1. By the Fourier series of f
2. By the Fourier series of $\breve{f}$ restricted to $[0, T)$ (which actually is a cosine-series)

Example 1.34. Let f be the function with period T defined by $f(t) = 2t/T - 1$ for $0 \leq t < T$. In each period the function increases linearly from -1 to 1. Because f is discontinuous at the boundaries, we would except the Fourier series to converge slowly. Since the function is antisymmetric, the coefficients a_n are zero, and we compute b_n as

$$\begin{aligned} b_n &= \frac{2}{T}\int_0^T \frac{2}{T}\left(t - \frac{T}{2}\right)\sin(2\pi nt/T)dt = \frac{4}{T^2}\int_0^T \left(t - \frac{T}{2}\right)\sin(2\pi nt/T)dt \\ &= \frac{4}{T^2}\int_0^T t\sin(2\pi nt/T)dt - \frac{2}{T}\int_0^T \sin(2\pi nt/T)dt \\ &= -\frac{2}{\pi n}, \end{aligned}$$

so that the Fourier series is

$$-\frac{2}{\pi}\sin(2\pi t/T) - \frac{2}{2\pi}\sin(2\pi 2t/T) - \frac{2}{3\pi}\sin(2\pi 3t/T) - \frac{2}{4\pi}\sin(2\pi 4t/T) - \cdots,$$

which indeed converges slowly to 0. Let us now instead consider the symmetric extension of f. Clearly this is the triangle wave with period $2T$, and the Fourier series of this was

$$\begin{aligned} &-\frac{8}{\pi^2}\cos(2\pi t/(2T)) - \frac{8}{3^2\pi^2}\cos(2\pi 3t/(2T)) - \frac{8}{5^2\pi^2}\cos(2\pi 5t/(2T)) \\ &-\frac{8}{7^2\pi^2}\cos(2\pi 7t/(2T)) + \cdots. \end{aligned}$$

Comparing the two series, we see that the coefficient at frequency n/T in the first series has value $-2/(n\pi)$, while in the second series it has value

$$-\frac{8}{(2n)^2\pi^2} = -\frac{2}{n^2\pi^2}.$$

The second series clearly converges faster than the first.

If we use $T = 1/880$, the symmetric extension will be the triangle wave of Example 1.19. Its Fourier series for $N = 10$ is shown in Figure 1.7(b) and the Fourier series for $N = 20$ is shown in Figure 1.11. The value $N = 10$ is used since this corresponds to the same frequencies as the previous figure for $N = 20$. It is clear from the plot that the Fourier series for f itself is not a very good approximation. However, we cannot differentiate between the Fourier series and the function itself for the triangle wave. ♣

Exercises for Section 1.6

1. Show that the complex Fourier coefficients y_n of f, and the cosine-coefficients a_n of $\breve{f}$ are related by $a_{2n} = y_n + y_{-n}$. This result is not enough to obtain the entire Fourier series of $\breve{f}$, but at least it gives us half of it.

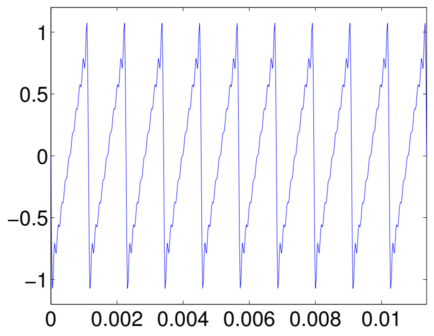


Figure 1.11: The Fourier series for $N = 10$ for the function in Example 1.34

1.7 Some properties of Fourier series

We continue by establishing some important properties of Fourier series, in particular the Fourier coefficients for some important functions. In these lists, we will use the notation $f \to y_n$ to indicate that y_n is the n'th Fourier coefficient of $f(t)$.

Theorem 1.35 (Fourier series pairs)**.** The functions 1, $e^{2\pi int/T}$, and $\chi_{-a,a}$ have the Fourier coefficients

$$\begin{aligned} 1 &\to \boldsymbol{e}_0 = (1, 0, 0, 0\ldots,) \\ e^{2\pi int/T} &\to \boldsymbol{e}_n = (0, 0, \ldots, 1, 0, 0, \ldots) \\ \chi_{-a,a} &\to \frac{\sin(2\pi na/T)}{\pi n}. \end{aligned}$$

The 1 in $\boldsymbol{e}_n$ is at position n and the function $\chi_{-a,a}$ is the characteristic function of the interval $[-a, a]$, defined by

$$\chi_{-a,a}(t) = \begin{cases} 1, & \text{if } t \in [-a, a]; \\ 0, & \text{otherwise.} \end{cases}$$

The first two pairs are easily verified, so the proofs are omitted. The case for $\chi_{-a,a}$ is very similar to the square wave, but easier to prove, and therefore also omitted.

Theorem 1.36 (Fourier series properties)**.** The mapping $f \to y_n$ is linear: if $f \to x_n$, $g \to y_n$, then

$$af + bg \to ax_n + by_n$$

For all n. Moreover, if f is real and periodic with period T, the following properties hold:

1. $y_n = \overline{y_{-n}}$ for all n.
2. If $g(t) = f(-t)$ and $f \to y_n$, then $g \to \overline{y_n}$. In particular,
   (a) if $f(t) = f(-t)$ (i.e. f is symmetric), then all y_n are real, so that b_n are zero and the Fourier series is a cosine series.
   (b) if $f(t) = -f(-t)$ (i.e. f is antisymmetric), then all y_n are purely imaginary, so that the a_n are zero and the Fourier series is a sine series.
3. If $g(t) = f(t-d)$ (i.e. g is the function f delayed by d) and $f \to y_n$, then $g \to e^{-2\pi ind/T} y_n$.
4. If $g(t) = e^{2\pi idt/T} f(t)$ with d an integer, and $f \to y_n$, then $g \to y_{n-d}$.
5. Let d be a number. If $f \to y_n$, then $f(d+t) = f(d-t)$ for all t if and only if the argument of y_n is $-2\pi nd/T$ for all n.

The last property looks a bit mysterious. We will not have use for this property before the next chapter.

Proof. The proof of linearity is left to the reader. Property 1 follows immediately by writing

$$\begin{aligned} y_n &= \frac{1}{T}\int_0^T f(t)e^{-2\pi int/T}dt = \overline{\frac{1}{T}\int_0^T f(t)e^{2\pi int/T}dt} \\ &= \overline{\frac{1}{T}\int_0^T f(t)e^{-2\pi i(-n)t/T}dt} = \overline{y_{-n}}. \end{aligned}$$

Also, if $g(t) = f(-t)$, we have that

$$\begin{aligned} \frac{1}{T}\int_0^T g(t)e^{-2\pi int/T}dt &= \frac{1}{T}\int_0^T f(-t)e^{-2\pi int/T}dt = -\frac{1}{T}\int_0^{-T} f(t)e^{2\pi int/T}dt \\ &= \frac{1}{T}\int_0^T f(t)e^{2\pi int/T}dt = \overline{y_n}. \end{aligned}$$

Property 2 follows from this, since the remaining statements here were established in Theorems 1.21, 1.26, and Exercise 1.5.7. To prove property 3, we observe that the Fourier coefficients of $g(t) = f(t-d)$ are

$$\begin{aligned}\frac{1}{T}\int_0^T g(t)e^{-2\pi int/T}dt &= \frac{1}{T}\int_0^T f(t-d)e^{-2\pi int/T}dt\\ &= \frac{1}{T}\int_0^T f(t)e^{-2\pi in(t+d)/T}dt\\ &= e^{-2\pi ind/T}\frac{1}{T}\int_0^T f(t)e^{-2\pi int/T}dt = e^{-2\pi ind/T}y_n.\end{aligned}$$

For property 4 we observe that the Fourier coefficients of $g(t) = e^{2\pi idt/T}f(t)$ are

$$\begin{aligned}\frac{1}{T}\int_0^T g(t)e^{-2\pi int/T}dt &= \frac{1}{T}\int_0^T e^{2\pi idt/T}f(t)e^{-2\pi int/T}dt\\ &= \frac{1}{T}\int_0^T f(t)e^{-2\pi i(n-d)t/T}dt = y_{n-d}.\end{aligned}$$

If $f(d+t) = f(d-t)$ for all t, we define the function $g(t) = f(t+d)$ which is symmetric about 0, so that it has real Fourier coefficients. But then the Fourier coefficients of $f(t) = g(t-d)$ are $e^{-2\pi ind/T}$ times the (real) Fourier coefficients of g by property 3. It follows that y_n, the Fourier coefficients of f, has argument $-2\pi nd/T$. The proof in the other direction follows by noting that any function where the Fourier coefficients are real must be symmetric about 0, once the Fourier series is known to converge. This proves property 5. □

From this theorem we see that there exist several cases of duality between the function and its Fourier series:

1. Delaying a function corresponds to multiplying the Fourier coefficients with a complex exponential. Vice versa, multiplying a function with a complex exponential corresponds to delaying the Fourier coefficients.

2. Symmetry/antisymmetry for a function corresponds to the Fourier coefficients being real/purely imaginary. Vice versa, a function which is real has Fourier coefficients which are conjugate symmetric.

Actually, these dualities would have become even stronger if we had considered Fourier series of complex functions instead of real functions. We will not go into this.

Exercises for Section 1.7

1. Define the function f with period T on $[-T/2, T/2]$ by

$$f(t) = \begin{cases} 1, & \text{if } -T/4 \le t < T/4;\\ -1, & \text{if } |T/4| \le t < |T/2|.\end{cases}$$

f is just the square wave, shifted with $T/4$. Compute the Fourier coefficients of f directly, and use Property 3 in Theorem 1.36 to verify your result.

2. Find a function f which has the complex Fourier series

$$\sum_{n \text{ odd}} \frac{4}{\pi(n+4)} e^{2\pi i n t/T}.$$

Hint: Attempt to use one of the properties in Theorem 1.36 on the Fourier series of the square wave.

1.8 Operations on sound: filters

It is easy to see how we can use Fourier coefficients to analyse or improve sound: Noise in a sound often corresponds to the presence of some high frequencies with large coefficients, and by removing these, we remove the noise. For example, we could set all the coefficients except the first one to zero. This would change the unpleasant square wave to the pure tone $\sin(2\pi 440 t)$, which we started our experiments with. Doing so is an example of an important operation on sound called *filtering*:

Definition 1.37 (Analog filters)**.** An operation on sound is called an *analog filter* if it preserves the different frequencies in the sound. In other words, s is an analog filter if, for any sound $f_1 = \sum_f c(f)e^{2\pi i f t}$, the output of $s(f_1)$ of s can be written

$$s(f_1) = s\left(\sum_f c(f)e^{2\pi i f t}\right) = \sum_f c(f)\lambda_s(f)e^{2\pi i f t},$$

where $\lambda_s(f)$ is a function describing how s treats the different frequencies. $\lambda_s(f)$ uniquely determines s, and is also called the frequency response of s.

The following is clear:

Theorem 1.38. The following hold for an analog filter s:

1. When f_1 is periodic with period T, $f_2 = s(f_1)$ is also periodic with period T.
2. When $f_2 = s(f_1)$ we have that $(f_2)_N = s((f_1)_N)$, i.e. s maps the N'th order Fourier series to oneanother.
3. Any pure tone is an eigenvector of s.

Example 1.39. To see how an analog filter may look, consider the operation which sends the function $f_1(t)$ to the function $f_2(t) = \int_{-\infty}^{\infty} g(s)f_1(t-s)ds$. Clearly the integral may not exist, but if we assume that g is nonzero only on $[a,b]$ the integral becomes $\int_a^b g(s)f_1(t-s)ds$, which exists if f_1 and g are bounded. We compute

$$s(e^{2\pi ift}) = \int_a^b g(s)e^{2\pi if(t-s)}ds = \int_a^b g(s)e^{-2\pi ifs}dse^{2\pi ift} = \lambda_s(f)e^{2\pi ift},$$

where $\lambda_s(f) = \int_a^b g(s)e^{-2\pi ifs}ds$. Since s also is linear, this shows that s is a filter, and we have also expressed its frequency response as an integral. Since the function g is arbitrary, this strategy leads to a wide class of analog filters. We may ask the question of whether the general analog filter always has this form. We will not go further into this, although one can find partially affirmative answers to this question. ♣

We also need to say something about the connection between filters and symmetric functions. We saw that the symmetric extension of a function took the form of a cosine-series, and that this converged faster to the symmetric extension than the Fourier series did to the function. If a filter preserves cosine-series it will also preserve symmetric extensions, and therefore also map fast-converging Fourier series to fast-converging Fourier series. The following result will be useful in this respect:

Theorem 1.40. If the frequency response of a filter satisfies $\lambda_s(f) = \lambda_s(-f)$ for all frequencies f, then the filter preserves cosine series and sine series.

Proof. We have that

$$\begin{aligned} s(\cos(2\pi nt/T)) &= s\left(\frac{1}{2}(e^{2\pi int/T} + e^{-2\pi int/T})\right) \\ &= \frac{1}{2}\lambda_s(n/T)e^{2\pi int/T} + \frac{1}{2}\lambda_s(-n/T)e^{-2\pi int/T} \\ &= \lambda_s(n/T)\left(\frac{1}{2}(e^{2\pi int/T} + e^{-2\pi int/T})\right) = \lambda_s(n/T)\cos(2\pi nt/T). \end{aligned}$$

This means that s preserves cosine-series. A similar computation holds for sine-series holds as well. □

An analog filter where $\lambda_s(f) = \lambda_s(-f)$ is also called a *symmetric filter*. As an example, consider the analog filter $s(f_1) = \int_{-a}^a g(s)f_1(t-s)ds$ where g is symmetric around 0 and supported on $[-a,a]$. s is a symmetric filter since

$$\lambda_s(f) = \int_{-a}^a g(s)e^{-2\pi ifs}ds = \int_{-a}^a g(s)e^{2\pi ifs}ds = \lambda_s(-f).$$

Filters are much used in practice, but the way we have defined them here makes them not very useful for computation. We will handle the problem of making filters suitable for computation in Chapter 3.

1.9 The MP3 standard

Digital audio first became commonly available when the CD was introduced in the early 1980s. As the storage capacity and processing speeds of computers increased, it became possible to transfer audio files to computers and both play and manipulate the data, in ways such as in the previous section. However, audio was represented by a large amount of data and an obvious challenge was how to reduce the storage requirements. Lossless coding techniques like Huffman and Lempel-Ziv coding were known and with these kinds of techniques the file size could be reduced to about half of that required by the CD format. However, by allowing the data to be altered a little bit it turned out that it was possible to reduce the file size down to about ten percent of the CD format, without much loss in quality. The MP3 audio format takes advantage of this.

MP3, or more precisely *MPEG-1 Audio Layer 3*, is part of an audio-visual standard called MPEG. MPEG has evolved over the years, from MPEG-1 to MPEG-2, and then to MPEG-4. The data on a DVD disc can be stored with either MPEG-1 or MPEG-2, while the data on a bluray-disc can be stored with either MPEG-2 or MPEG-4. MP3 was developed by Philips, CCETT (Centre commun d'études de télévision et télécommunications), IRT (Institut für Rundfunktechnik) and Fraunhofer Society, and became an international standard in 1991. Virtually all audio software and music players support this format. MP3 is just a sound format. It leaves a substantial amount of freedom in the encoder, so that different encoders can exploit properties of sound in various ways, in order to alter the sound in removing inaudible components therein. As a consequence there are many different MP3 encoders available, of varying quality. In particular, an encoder which works well for higher bit rates (high quality sound) may not work so well for lower bit rates.

With MP3, the sound is split into *frequency bands*, each band corresponding to a particular frequency range. In the simplest model, 32 frequency bands are used. A frequency analysis of the sound, based on what is called a *psycho-acoustic model*, is the basis for further transformation of these bands. The psycho-acoustic model computes the significance of each band for the human perception of the sound. When we hear a sound, there is a mechanical stimulation of the ear drum, and the amount of stimulus is directly related to the size of the sample values of the digital sound. The movement of the ear drum is then converted to electric impulses that travel to the brain where they are perceived as sound. The perception process uses a transformation of the sound so that a steady oscillation in air pressure is perceived as a sound with a fixed frequency. In this process certain kinds of perturbations of the sound are hardly noticed by the brain, and this is exploited in lossy audio compression.

More precisely, when the psycho-acoustic model is applied to the frequency content resulting from our frequency analysis, *scale factors* and *masking thresholds* are assigned for each band. The computed masking thresholds have to do with a phenomenon called *masking*. A simple example of this is that a loud sound will make a simultaneous low sound inaudible. For compression this means that if certain frequencies of a signal are very prominent, most of the other frequencies can be removed, even when they are quite large. If the sounds are below the masking threshold, it is simply ommited by the encoder, since the model says that the sound should be inaudible.

Masking effects are just one example of what is called psycho-acoustic effects, and all such effects can be taken into account in a psycho-acoustic model. Another obvious such effect regards computing the scale factors: the human auditory system can only perceive frequencies in the range 20 Hz – 20 000 Hz. An obvious way to do compression is therefore to remove frequencies outside this range, although there are indications that these frequencies may influence the listening experience inaudibly. The computed scaling factors tell the encoder about the precision to be used for each frequency band: If the model decides that one band is very important for our perception of the sound, it assigns a big scale factor to it, so that more effort is put into encoding it by the encoder (i.e. it uses more bits to encode this band).

Using appropriate scale factors and masking thresholds provide compression, since bits used to encode the sound are spent on parts important for our perception. Developing a useful psycho-acoustic model requires detailed knowledge of human perception of sound. Different MP3 encoders use different such models, so they may produce very different results, worse or better.

The information remaining after frequency analysis and using a psycho-acoustic model is coded efficiently with (a variant of) Huffman coding. MP3 supports bit rates from 32 to 320 kb/s and the sampling rates 32, 44.1, and 48 kHz. The format also supports variable bit rates (the bit rate varies in different parts of the file). An MP3 encoder also stores metadata about the sound, such as the title of the audio piece, album and artist name and other relevant data.

MP3 too has evolved in the same way as MPEG, from MP1 to MP2, and to MP3, each one more sophisticated than the other, providing better compression. MP3 is not the latest development of audio coding in the MPEG family: AAC (Advanced Audio Coding) is presented as the successor of MP3 by its principal developer, Fraunhofer Society, and can achieve better quality than MP3 at the same bit rate, particularly for bit rates below 192 kb/s. AAC became well known in April 2003 when Apple introduced this format (at 128 kb/s) as the standard format for their iTunes Music Store and iPod music players. AAC is also supported by many other music players, including the most popular mobile phones.

The technologies behind AAC and MP3 are very similar. AAC supports more sample rates (from 8 kHz to 96 kHz) and up to 48 channels. AAC uses the same transformation as MP3, but AAC processes 1 024 samples at a time. AAC also uses much more sophisticated processing of frequencies above 16 kHz and has a number of other enhancements over MP3. AAC, as MP3, uses Huffman coding for efficient coding of the transformed values. Tests seem quite conclusive that AAC is better than MP3 for low bit rates (typically below 192 kb/s), but for higher rates it is not so easy to differentiate between the two formats. As for MP3 (and the other formats mentioned here), the quality of an AAC file depends crucially on the quality of the encoding program.

There are a number of variants of AAC, in particular AAC Low Delay (AAC-

LD). This format was designed for use in two-way communication over a network, for example the internet. For this kind of application, the encoding (and decoding) must be fast to avoid delays (a delay of at most 20 ms can be tolerated).

1.10 Summary

We discussed the basic question of what is sound is, and concluded that sound could be modeled as a sum of frequency components. If the function was periodic we could define its Fourier series, which can be thought of as an approximation scheme for periodic functions using finite-dimensional spaces of trigonometric functions. We established the basic properties of Fourier series, and some duality relationships between the function and its Fourier series. We have also computed the Fourier series of the square wave and the triangle wave, and we saw that we could speed up the convergence of the Fourier series by instead considering the symmetric extension of the function.

We also discussed the MP3 standard for compression of sound, and its relation to a psychoacoutic model which describes how the human auditory system perceives sound. There exist a wide variety of documents on this standard. In [9], an overview is given, which, although written in a signal processing friendly language and representing most relevant theory such as for the psychoacoutic model, does not dig into all the details.

Chapter 2

Digital sound and Discrete Fourier analysis

In Chapter 1 we saw how a periodic function can be decomposed into a linear combination of sines and cosines, or equivalently, a linear combination of complex exponential functions. This kind of decomposition is, however, not very convenient from a computational point of view. First of all, the coefficients are given by integrals that in most cases cannot be evaluated exactly, so some kind of numerical integration technique needs to be applied. Secondly, functions are defined for all time instances. On computers and various kinds of media players, however, the sound is *digital*, meaning that it is represented by a large number of function values, and not by a function defined for all time instances.

In this chapter our starting point is simply a vector which represents the sound values (rather than the function $f(t)$), and as before we would like to decompose this in terms of linear combinations of vectors built from complex exponentials. As before it turns out that this is simplest when we assume that the values in the vector repeat periodically. Then a vector of finite dimension can be used to represent all sound values, and operations and the computation of (discrete) Fourier series simply amounts to multiplying the vector by a matrix, and there are efficient algorithms for doing this. It turns out that these algorithms can also be used for computing good approximations to the Fourier series in Chapter 1.

2.1 Digital sound

We start by defining what a digital sound is and by establishing some notation and terminology.

Definition 2.1 (Digital sound)**.** A digital sound is a sequence $\boldsymbol{x} = \{x_i\}_{i=0}^{N-1}$ that corresponds to measurements of the air pressure of a sound f, recorded at a fixed rate of f_s (the sampling frequency or sampling rate) measurements per second, i.e.,

$$x_k = f(k/f_s), \quad \text{for } k = 0, 1; \ldots, N.$$

The measurements are often referred to as samples. The time between successive measurements is called the sampling period and is usually denoted T_s. The length of the vector is usually assumed to be N, and it is indexed from 0 to $N-1$. If the sound is in stereo there will be two arrays $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, one for each channel. Measuring the sound is also referred to as sampling the sound, or analog to digital (AD) conversion.

Note that this indexing convention for vectors is not standard in mathematics. Note also that Matlab indexes vectors from 1, so algorithms given here must be adjusted accordingly.

In most cases, a digital sound is sampled from an analog (continuous) audio signal. This is usually done with a technique called Pulse Code Modulation (PCM). The audio signal is sampled at regular intervals and the sampled values stored in a suitable number format. Both the sampling frequency, and the accuracy and number format used for storing the samples, may vary for different kinds of audio, and both influence the quality of the resulting sound. For simplicity the quality is often measured by the number of bits per second, i.e., the product of the sampling rate and the number of bits (binary digits) used to store each sample. This is also referred to as the *bit rate*. For the computer to be able to play a digital sound, samples must be stored in a file or in memory on a computer. To do this efficiently, digital sound formats are used. A couple of them are described in the examples below.

Example 2.2. In the classical CD-format the audio signal is sampled 44 100 times per second and the samples stored as 16-bit integers. This works well for music with a reasonably uniform dynamic range, but is problematic when the range varies. Suppose for example that a piece of music has a very loud passage. In this passage the samples will typically make use of almost the full range of integer values, from $-2^{15}-1$ to 2^{15}. When the music enters a more quiet passage the sample values will necessarily become much smaller and perhaps only vary in the range -1000 to 1000, say. Since $2^{10} = 1024$ this means that in the quiet passage the music would only be represented with 10-bit samples. This problem can be avoided by using a floating-point format instead, but very few audio formats appear to do this.

The bit rate for CD-quality stereo sound is $44100 \times 2 \times 16$ bits/s = 1411.2 kb/s. This quality measure is particularly popular for lossy audio formats where the uncompressed audio usually is the same (CD-quality). However, it should be remembered that even two audio files in the same file format and with the same bit rate may be of very different quality because the encoding programs may be of different quality. ♣

Example 2.3. For telephony it is common to sample the sound 8000 times per second and represent each sample value as a 13-bit integer. These integers are then converted to a kind of 8-bit floating-point format with a 4-bit significand. Telephony therefore generates a bit rate of 64 000 bits per second, i.e. 64 kb/s. ♣

Newer formats with higher quality are available. Music is distributed in various formats on DVDs (DVD-video, DVD-audio, Super Audio CD) with sampling rates up to 192 000 and up to 24 bits per sample. These formats also support surround sound (up to seven channels in contrast to the two stereo channels on a CD). In the following we will assume all sound to be digital. Later we will return to how we reconstruct audible sound from digital sound.

2.2 Simple operations on digital sound

Simple operations and computations with digital sound can be done in any programming environment. Let us take a look at how these. From Definition 2.1, digital sound is just an array of sample values $\boldsymbol{x} = (x_i)_{i=0}^{N-1}$, together with the sample rate f_s. Performing operations on the sound therefore amounts to doing the appropriate computations with the sample values and the sample rate. The most basic operation we can perform on a sound is simply playing it.

2.2.1 Playing a sound

You may already have listened to pure tones, square waves and triangle waves in the last section. The corresponding sound files were generated in a way we will describe shortly, placed in a directory available on the internet, and linked to from these notes. A program on your computer was able to play these files when you clicked on them. Let us take a closer look at the different steps here. You will need these steps in Exercise 3, where you will be asked to implement a Matlab-function which plays a pure sound with a given frequency on your computer.

First we need to know how we can obtain the samples of a pure tone. The following code does this for a pure tone with frequency f, over a period of 3 seconds, and with sampling rate f_s.

```
t=0:(1/fs):3;
sd=sin(2*pi*f*t);
```

Matlab code will be displayed in this way throughout these notes. In order to listen to these sound samples, we need to take a look at the functions built into Matlab for playing sound. We have the two functions

```
playblocking(playerobj)
playblocking(playerobj,[start stop])
```

These simply play the audio segment encapsulated by the object `playerobj`. `playblocking` means that the method playing the sound will block until it has finished playing. We will have use for this functionality later on, since we may play sounds in successive order. With the first function the entire audio segment is played. With the second function the playback starts at sample `start`, and ends at sample `stop`. These functions are just software interfaces to the sound card in your computer. It basically sends the array of sound samples and sample rate to the sound card, which uses some method for reconstructing the sound to an analog sound signal. This analog signal is then sent to the loudspeakers and we hear the sound.

Fact 2.4. The basic command in a programming environment that handles sound takes as input an array of sound samples $\boldsymbol{x}$ and a sample rate s, and plays the corresponding sound through the computer's loudspeakers.

The mysterious `playerobj` object above can be obtained from the sound samples (represented by a vector `S`) and the sampling rate (`fs`) by the function:

```
playerobj=audioplayer(S,fs)
```

The sound samples can have different data types. We will always assume that they are of type `double`. Matlab requires that they have values between -1 and 1 (i.e. these represent the range of numbers which can be played through the sound card of the computer). Also, `S` can actually be a matrix: Each column in the matrix represents a sound channel. Sounds we generate on our own from a mathematical function (as for the pure tone above) will typically have only one channel, so that `S` has only one column. If `S` originates from a stereo sound file, it will have two columns.

You can create `S` on your own, and set the sampling rate to whatever value you like. However, we can also fill in the sound samples from a sound file. To do this from a file in the `wav`-format named `filename`, simply write

```
[S,fs]=wavread(filename)
```

The `wav`-format format was developed by Microsoft and IBM, and is one of the most common file formats for CD-quality audio. It uses a 32-bit integer to specify the file size at the beginning of the file, which means that a WAV-file cannot be larger than 4 GB. In addition to filling in the sound samples in the vector `S`, this function also returns the sampling rate `fs` used in the file. The function

```
wavwrite(S,fs,filename)
```

can similarly be used to write the data stored in the vector `S` to the `wav`-file by the name `filename`. In the following we will both fill in the vector `S` on our own by using values from mathematical functions, as well as from a file. As an example of the first, we can listen to and write to a file the pure tone of frequency 440Hz, which we listened to in Section 1.2, with the help of the following code:

```
antsec=3;
```

```
fs=40000;
t=linspace(0,antsec,fs*antsec);
S=sin(2*pi*440*t);
playerobj=audioplayer(S,fs);
playblocking(playerobj);
wavwrite(S,fs,'puretone440.wav');
```

The code creates a pure tone which lasts for three seconds (if you want the tone to last longer, you can change the value of the variable `antsec`). We also tell the computer that there are 40000 samples per second. This value is not coincidental, and we will return to this. In fact, the sound file for the pure tone embedded into this document was created in this way! In the same way we can listen to the square wave with the help of the following code:

```
antsec=3;
fs=44100;
samplesperperiod=round(fs/440);
oneperiod=[ones(1,round(samplesperperiod/2)) ...
           -ones(1,round(samplesperperiod/2))];
allsamples=zeros(1,antsec*440*length(oneperiod));
for k=1:(antsec*440)
  allsamples(((k-1)*length(oneperiod)+1):k*length(oneperiod))=oneperiod;
end
playerobj=audioplayer(allsamples,fs);
playblocking(playerobj);
```

The code creates 440 copies of the square wave per second by first computing the number of samples needed for one period when it is known that we should have a total of 40000 samples per second, and then constructing the samples needed for one period. In the same fashion we can listen to the triangle wave simply by replacing the code for generating the samples for one period with the following:

```
oneperiod=[linspace(-1,1,round(samplesperperiod/2)) ...
           linspace(1,-1,round(samplesperperiod/2))];
```

Instead of using the formula for the triangle wave, directly, we have used the function `linspace`.

As an example of how to fill in the sound samples from a file, the code

```
[S fs] = wavread('castanets.wav');
```

reads the file castanets.wav, and stores the sound samples in the matrix `S`. In this case there are two sound channels, so there are two columns in `S`. To work with sound from only one channel, we extract the second channel as follows:

```
x=S(:,2);
```

`wavread` returns sound samples with floating point precision. If we have made any changes to the sound samples, we need to secure that they are between -1 and 1 before we play them. If the sound samples are stored in `x`, this can be achieved as follows:

```
x = x / max(abs(x));
```

`x` can now be played just as the signals we constructed from mathematical formulas above.

It may be that some other environment than Matlab gives you the `play` functionality on your computer. Even if no environment on your computer supports such `play`-functionality at all, you may still be able to play the result of your computations if there is support for saving the sound in some standard format like mp3. The resulting file can then be played by the standard audio player on your computer.

Example 2.5 (Changing the sample rate)**.** We can easily play back a sound with a different sample rate than the standard one. If we in the code above instead wrote `fs=80000`, the sound card will assume that the time distance between neighbouring samples is half the time distance in the original. The result is that the sound takes half as long, and the frequency of all tones is doubled. For voices the result is a characteristic Donald Duck-like sound.

Conversely, the sound can be played with half the sample rate by setting `fs=20000`. Then the length of the sound is doubled and all frequencies are halved. This results in low pitch, roaring voices.

Fact 2.6. A digital sound can be played back with a double or half sample rate by replacing

```
playerobj=audioplayer(S,fs);
```

with

```
playerobj=audioplayer(S,2*fs);
```

and

```
playerobj=audioplayer(S,fs/2);
```

respectively.

The sample file `castanets.wav` played at double sampling rate sounds like this, while it sounds like this when it is played with half the sampling rate. ♣

Example 2.7 (Playing the sound backwards)**.** At times a popular game has been to play music backwards to try and find secret messages. In the old days of analog music on vinyl this was not so easy, but with digital sound it is quite simple; we just need to reverse the samples. To do this we just loop through the array and put the last samples first.

Fact 2.8. Let $\boldsymbol{x} = (x_i)_{i=0}^{N-1}$ be the samples of a digital sound. Then the samples $\boldsymbol{y} = (y_i)_{i=0}^{N-1}$ of the reverse sound are given by

$$y_i = x_{N-i-1}, \text{ for } i = 0, 1, \ldots N-1.$$

When we reverse the sound samples with Matlab, we have to reverse the elements in both sound channels. This can be performed as follows

```
sz=size(S,1);
newS=[S(sz:(-1):1,1) S(sz:(-1):1,2)];
```

Performing this on our sample file you generate a sound which sounds like this. ♣

Example 2.9 (Adding noise)**.** To remove noise from recorded sound can be very challenging, but adding noise is simple. There are many kinds of noise, but one kind is easily obtained by adding random numbers to the samples of a sound.

Fact 2.10. Let $\boldsymbol{x}$ be the samples of a digital sound of length N. A new sound $\boldsymbol{y}$ with noise added can be obtained by adding a random number to each sample,

```
y=x+c*(2*rand(1,N)-1);
```

where `rand` is a Matlab function that returns random numbers in the interval $[0,1]$, and c is a constant (usually smaller than 1) that dampens the noise. The effect of writing `(2*rand(1,N)-1)` is that random numbers between -1 and 1 are returned instead of random numbers between 0 and 1.

Adding noise in this way will produce a general hissing noise similar to the noise you hear on the radio when the reception is bad. As before you should add noise to both channels. Note alse that the sound samples may be outside $[-1,1]$ after adding noise, so that you should scale the samples before writing them to file. The factor c is important, if it is too large, the noise will simply drown the signal $\boldsymbol{y}$: `castanets.wav` with noise added with $c = 0.4$ sounds like this, while with $c = 0.1$ it sounds like this. ♣

In addition to the operations listed above, the most important operations on digital sound are *digital filters*. These are given a separate treatment in Chapter 3.

Exercises for Section 2.2

1. Define the following sound signal

$$f(t) = \begin{cases} 0 & 0 \le t \le 4/440 \\ 2\frac{440t-4}{8}\sin(2\pi 440t) & 4/440 \le t \le 12/440 \\ 2\sin(2\pi 440t) & 12/440 \le t \le 20/440 \end{cases}$$

This corresponds to the sound plotted in Figure 1.1(a), where the sound is unaudible in the beginning, and increases linearly in loudness over time with a given frequency until maximum loudness is avchieved. Write a Matlab program which generates this sound, and listen to it.

2. Find two constant a and b so that the function $f(t) = a\sin(2\pi 440t) + b\sin(2\pi 4400t)$ resembles the plot from Figure 1.1(b) as closely as possible. Generate the samples of this sound, and listen to it with Matlab.

3. Let us write some code so that we can experiment with different pure sounds

a. Write a function

```
function playpuresound(f)
```

which generates the samples over a period of 3 seconds for a pure tone with frequency f, with sampling frequency $f_s = 2.5f$ (we will explain this value later).

b. Use the function `playpuresound` to listen to pure sounds of frequency 440Hz and 1500Hz, and verify that they are the same as the sounds you already have listened to in this section.

c. How high frequencies are you able to hear with the function `playpuresound`? How low frequencies are you able to hear?

4. Write functions

```
function playsquare(T)
function playtriangle(T)
```

which plays the square wave of Example 1.11 and the triangle wave of Example 1.12, respectively, where T is given by the parameter. In your code, let the samples of the waves be taken at a frequency of 40000 samples per second. Verify that you generate the same sounds as you played in these examples when you set $T = \frac{1}{440}$.

5. Let us write programs so that we can listen to the Fourier approximations of the square wave and the triangle wave.

a. Write functions

```
function playsquaretrunk(T,N)
function playtriangletrunk(T,N)
```

which plays the order N Fourier approximation of the square wave and the triangle wave, respectively, for three seconds. Verify that you can generate the sounds you played in examples 1.18 and 1.19.

b. For these Fourier approximations, how high must you choose N for them to be indistuingishable from the square/triangle waves themselves? Also describe how the characteristics of the sound changes when n increases.

6. In this exercise we will experiment as in the first examples of this section.

a. Write a function

```
function playdifferentfs()
```

which plays the sound samples of `castanets.wav` with the same sample rate as the original file, then with twice the sample rate, and then half the sample rate. You should start with reading the file into a matrix (as explained in this section). Are the sounds the same as those you heard in Example 2.5?

b. Write a function

```
function playreverse()
```

which plays the sound samples of `castanets.wav` backwards. Is the sound the same as the one you heard in Example 2.7?

c. Write the new sound samples from b. to a new `wav`-file, as described above, and listen to it with your favourite mediaplayer.

7. In this exercise, we will experiment with adding noise to a signal.

a. Write a function

```
function playnoise(c)
```

which plays the sound samples of `castanets.wav` with noise added for the damping constant c as described above. Your code should add noise to both channels of the sound, and scale the sound samples so that they are between -1 and 1.

b. With your program, generate the two sounds played in Example 2.9, and verify that they are the same as those you heard.

c. Listen to the sound samples with noise added for different values of c. For which range of c is the noise audible?

2.3 Discrete Fourier analysis: Basic concepts

In this section we will parallel the developments we did for Fourier series, assuming instead that vectors (rather than functions) are involved. As with Fourier series we will assume that the vector is periodic. This means that we can represent it with the samples from only the first period. In the following we will only work with these samples, but we will remined ourselves from time to time that the samples actually come from a periodic vector. At the outset our vectors will have real components, but since we use complex exponentials we must be able to work with complex vectors also. We therefore first need to define the standard inner product and norm for complex vectors.

Definition 2.11. For complex vectors of length N the Euclidean inner product is given by

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \sum_{k=0}^{N-1} x_k \overline{y_k}. \tag{2.1}$$

The associated norm is

$$\|\boldsymbol{x}\| = \sqrt{\sum_{k=0}^{N-1} |x_k|^2}. \tag{2.2}$$

In the previous chapter we saw that, using a Fourier series, a function with period T could be approximated by linear combinations of the functions (the pure tones) $\{e^{2\pi int/T}\}_{n=0}^{N}$. This can be generalised to vectors (digital sounds), but then the pure tones must of course also be vectors.

Definition 2.12 (Discrete Fourier analysis)**.** In Discrete Fourier analysis, a vector $\boldsymbol{x} = (x_0, \ldots, x_{N-1})$ is represented as a linear combination of the N vectors

$$\boldsymbol{\phi}_n = \frac{1}{\sqrt{N}} \left(1, e^{2\pi in/N}, e^{2\pi i2n/N}, \ldots, e^{2\pi ikn/N}, \ldots, e^{2\pi in(N-1)/N}\right).$$

These vectors are called the normalised complex exponentials, or the pure digital tones of order N. The whole collection $\mathcal{F}_N = \{\boldsymbol{\phi}_n\}_{n=0}^{N-1}$ is called the N-point Fourier basis.

The following lemma shows that the vectors in the Fourier basis are orthonormal, so they do indeed form a basis.

Lemma 2.13. The normalised complex exponentials $\{\boldsymbol{\phi_n}\}_{n=0}^{N-1}$ of order N form an orthonormal basis in $\mathbb{R}^N$.

Proof. Let n_1 and n_2 be two distinct integers in the range $[0, N-1]$. The inner product of $\boldsymbol{\phi}_{n_1}$ and $\boldsymbol{\phi}_{n_2}$ is then given by

$$\begin{aligned}
\langle \boldsymbol{\phi}_{n_1}, \boldsymbol{\phi}_{n_2} \rangle &= \frac{1}{N}\langle e^{2\pi i n_1 k/N}, e^{2\pi i n_2 k/N} \rangle \\
&= \frac{1}{N}\sum_{k=0}^{N-1} e^{2\pi i n_1 k/N} e^{-2\pi i n_2 k/N} \\
&= \frac{1}{N}\sum_{k=0}^{N-1} e^{2\pi i (n_1 - n_2)k/N} \\
&= \frac{1}{N}\frac{1 - e^{2\pi i (n_1 - n_2)}}{1 - e^{2\pi i (n_1 - n_2)/N}} \\
&= 0.
\end{aligned}$$

In particular, this orthogonality means that the the complex exponentials form a basis. Clearly also $\langle \boldsymbol{\phi}_n, \boldsymbol{\phi}_n \rangle = 1$, so that the N-point Fourier basis is in fact an orthonormal basis. □

Note that the normalising factor $\frac{1}{\sqrt{N}}$ was not present for pure tones in the previous chapter. Also, the normalising factor $\frac{1}{T}$ from the last chapter is not part of the definition of the inner product in this chapter. These are small differences which have to do with slightly different notation for functions and vectors, and which will not cause confusion in what follows.

2.4 The Discrete Fourier Transform

The focus in Discrete Fourier analysis is to change coordinates from the standard basis to the Fourier basis, performing some operations on this "Fourier representation", and then change coordinates back to the standard basis. Such operations are of crucial importance, and in this section we study some of their basic properties. We start with the following definition.

Definition 2.14 (Discrete Fourier Transform)**.** The change of coordinates from the standard basis of $\mathbb{R}^N$ to the Fourier basis $\mathcal{F}_N$ is called the discrete Fourier transform (or DFT). The $N \times N$ matrix F_N that represents this change of basis is called the (N-point) Fourier 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 Fourier basis are called the DFT coefficients of $\boldsymbol{x}$ (in other words $\boldsymbol{y} = F_N\boldsymbol{x}$). The DFT of $\boldsymbol{x}$ is sometimes written as $\hat{\boldsymbol{x}}$.

We will normally write $\boldsymbol{x}$ for the given vector in $\mathbb{R}^N$, and $\boldsymbol{y}$ for the DFT of this vector. In applied fields, the Fourier basis vectors are also called *synthesis vectors*, since they can be used used to "synthesize" the vector $\boldsymbol{x}$, with weights provided by the DFT coefficients $\boldsymbol{y} = (y_n)_{n=0}^{N-1}$. To be more precise, we have that the change of coordinates performed by the DFT can be written as

$$\boldsymbol{x} = y_0\boldsymbol{\phi}_0 + y_1\boldsymbol{\phi}_1 + \cdots + y_{N-1}\boldsymbol{\phi}_{N-1} = \begin{pmatrix}\boldsymbol{\phi}_0 & \boldsymbol{\phi}_1 & \cdots & \boldsymbol{\phi}_{N-1}\end{pmatrix}\boldsymbol{y} = F_N^{-1}\boldsymbol{y}, \quad (2.3)$$

where we have used the inverse of the defining relation $\boldsymbol{y} = F_N\boldsymbol{x}$, and that the $\boldsymbol{\phi}_n$ are the columns in F_N^{-1} (this follows from the fact that F_N^{-1} is the change of coordinates matrix from the Fourier basis to the standard basis, and the Fourier basis vectors are clearly the columns in this matrix). Equation (2.3) is also called the synthesis equation.

Example 2.15. Let $\boldsymbol{x}$ be the vector of length N defined by $x_k = \cos(2\pi 5k/N)$, and $\boldsymbol{y}$ the vector of length N defined by $y_k = \sin(2\pi 7k/N)$. Let us see how we can compute $F_N(2\boldsymbol{x} + 3\boldsymbol{y})$. By the definition of the DFT as a change of coordinates, $F_N(\phi_n) = \boldsymbol{e}_n$. We therefore get

$$\begin{aligned}
F_N(2\boldsymbol{x}+3\boldsymbol{y}) &= F_N(2\cos(2\pi 5\cdot/N) + 3\sin(2\pi 7\cdot/N)) \\
&= F_N(2\frac{1}{2}(e^{2\pi i 5\cdot/N} + e^{-2\pi i 5\cdot/N}) + 3\frac{1}{2i}(e^{2\pi i 7\cdot/N} - e^{-2\pi i 7\cdot/N})) \\
&= F_N(\sqrt{N}\phi_5 + \sqrt{N}\phi_{N-5} - \frac{3i}{2}\sqrt{N}(\phi_7 - \phi_{N-7})) \\
&= \sqrt{N}(F_N(\phi_5) + F_N(\phi_{N-5}) - \frac{3i}{2}F_N\phi_7 + \frac{3i}{2}F_N\phi_{N-7}) \\
&= \sqrt{N}\boldsymbol{e}_5 + \sqrt{N}\boldsymbol{e}_{N-5} - \frac{3i}{2}\sqrt{N}\boldsymbol{e}_7 + \frac{3i}{2}\sqrt{N}\boldsymbol{e}_{N-7}.
\end{aligned}$$

♣

Let us also find the matrix F_N itself. From Lemma 2.13 we know that the columns of F_N^{-1} are orthonormal. If the matrix was real, it would have been called orthogonal, and the inverse matrix could have been obtained by transposing. F_N^{-1} is complex, however, and it is easy to see that the conjugation present in the definition of the inner product (2.1), implies that the inverse of F_N can be obtained if we also conjugate, in addition to transpose, i.e. $(F_N)^{-1} = (\overline{F_N})^T$. We call $(\overline{A})^T$ the conjugate transpose of A, and write A^H for it. We thus have that $(F_N)^{-1} = (F_N)^H$. Matrices which fulfill this are called unitary, which thus is the parallel to orthogonal matrices in the world of complex matrices.

Theorem 2.16. The Fourier matrix F_N is the unitary $N \times N$-matrix with entries given by

$$(F_N)_{nk} = \frac{1}{\sqrt{N}}e^{-2\pi i nk/N},$$

for $0 \le n, k \le N-1$.

Note that in the signal processing literature, it is not common to include the normalizing factor $1/\sqrt{N}$ in the definition of the DFT. From our more mathematical point of view this is useful since it makes the Fourier matrix unitary.

Let us now consider the change of coordinate from the Fourier basis back to the standard basis. This operation is also very common, so it also deserves its own definition.

Definition 2.17 (IDFT)**.** If $\boldsymbol{y} \in \mathbb{R}^N$ the vector $\boldsymbol{x} = (F_N)^H \boldsymbol{y}$ is referred to as the inverse discrete Fourier transform or (IDFT) of $\boldsymbol{y}$.

That $\boldsymbol{y}$ is the DFT of $\boldsymbol{x}$ and $\boldsymbol{x}$ is the IDFT of $\boldsymbol{y}$ can also be expressed in component form

$$x_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} y_n e^{2\pi i nk/N}, \tag{2.4}$$

$$y_n = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} x_k e^{-2\pi i nk/N}. \tag{2.5}$$

In applied fields such as signal processing, it is more common to state the DFT and IDFT in these component forms, rather than in the matrix forms $\boldsymbol{x} = (F_N)^H \boldsymbol{y}$ and $\boldsymbol{y} = F_N \boldsymbol{y}$.

Let us now see how these formulas work out in practice by considering some examples.

Example 2.18 (DFT on a square wave)**.** Let us attempt to apply the DFT to a signal $\boldsymbol{x}$ which is 1 on indices close to 0, and 0 elsewhere. Assume that

$$x_{-L} = \ldots = x_{-1} = x_0 = x_1 = \ldots = x_L = 1,$$

while all other values are 0. This is similar to a square wave, with some modifications: First of all we assume symmetry around 0, while the square wave of Example 1.11 assumes antisymmetry around 0. Secondly the values of the square wave are now 0 and 1, contrary to -1 and 1 before. Finally, we have a different proportion of where the two values are assumed. Nevertheless, we will also refer to the current digital sound as a square wave.

Since indices with the DFT are between 0 an $N-1$, and since $\boldsymbol{x}$ is assumed to have period N, the indices $[-L, L]$ where our signal is 1 translates to the indices $[0, L]$ and $[N-L, N-1]$ (i.e., it is 1 on the first and last parts of the vector). Elsewhere our signal is zero. Since $\sum_{k=N-L}^{N-1} e^{-2\pi i nk/N} = \sum_{k=-L}^{-1} e^{-2\pi i nk/N}$

(since $e^{-2\pi i n k/N}$ is periodic with period N), the DFT of $\boldsymbol{x}$ is

$$\begin{aligned}
y_n &= \frac{1}{\sqrt{N}}\sum_{k=0}^{L} e^{-2\pi i n k/N} + \frac{1}{\sqrt{N}}\sum_{k=N-L}^{N-1} e^{-2\pi i n k/N} \\
&= \frac{1}{\sqrt{N}}\sum_{k=0}^{L} e^{-2\pi i n k/N} + \frac{1}{\sqrt{N}}\sum_{k=-L}^{-1} e^{-2\pi i n k/N} \\
&= \frac{1}{\sqrt{N}}\sum_{k=-L}^{L} e^{-2\pi i n k/N} \\
&= \frac{1}{\sqrt{N}} e^{2\pi i n L/N} \frac{1-e^{-2\pi i n(2L+1)/N}}{1-e^{-2\pi i n/N}} \\
&= \frac{1}{\sqrt{N}} e^{2\pi i n L/N} e^{-\pi i n(2L+1)/N} e^{\pi i n/N} \frac{e^{\pi i n(2L+1)/N}-e^{-\pi i n(2L+1)/N}}{e^{\pi i n/N}-e^{-\pi i n/N}} \\
&= \frac{1}{\sqrt{N}} \frac{\sin(\pi n(2L+1)/N)}{\sin(\pi n/N)}.
\end{aligned}$$

This computation does in fact also give us the IDFT of the same vector, since the IDFT just requires a change of sign in all the exponents. From this example we see that, in order to represent $\boldsymbol{x}$ in terms of frequency components, all components are actually needed. The situation would have been easier if only a few frequencies were needed. ♣

Example 2.19. In most cases it is difficult to compute a DFT by hand, due to the entries $e^{-2\pi i n k/N}$ in the matrices, which typically can not be represented exactly. The DFT is therefore usually calculated on a computer only. However, in the case $N = 4$ the calculations are quite simple. In this case the Fourier matrix takes the form

$$F_4 = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix}.$$

We now can compute the DFT of a vector like $(1, 2, 3, 4)^T$ simply as

$$F_4\begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 1+2+3+4 \\ 1-2i-3+4i \\ 1-2+3-4 \\ 1+2i-3-4i \end{pmatrix} = \begin{pmatrix} 5 \\ -1+i \\ -1 \\ -1-i \end{pmatrix}.$$

♣

Example 2.20 (Direct implementation of the DFT)**.** Matlab supports complex arithmetic, so the DFT can be implemented very simply and directly by the code

```
function y=DFTImpl(x)
  N=length(x);
  FN=zeros(N);
  for n=1:N
    FN(n,:)=exp(-2*pi*1i*(n-1)*(0:(N-1))/N)/sqrt(N);
  end
  y=FN*x;
```

Note that n has been replaced by $n-1$ in this code since n runs from 1 to N (array indices must start at 1 in Matlab).

A direct implementation of the IDFT, which we could call `IDFTImpl` can be done similarly. Multiplying a full $N \times N$ matrix by a vector requires roughly N^2 arithmetic operations. The DFT algorithm above will therefore take a long time when N becomes moderately large, particularly in Matlab. It turns out that a much more efficient algorithm exists for computing the DFT, which we will study at the end of this chapter. Matlab also has a built-in implementation of the DFT which uses such an efficient algorithm. ♣

The DFT has properties which are very similar to those of Fourier series, as they were listed in Theorem 1.36. The following theorem sums this up:

Theorem 2.21 (DFT properties)**.** Let $\boldsymbol{x}$ be a real vector of length N. The DFT has the following properties:

1. $(\widehat{\boldsymbol{x}})_{N-n} = \overline{(\widehat{\boldsymbol{x}})_n}$ for $0 \leq n \leq N-1$.
2. If $\boldsymbol{z}$ is the vector with the components of $\boldsymbol{x}$ reversed so that $z_k = x_{N-k}$ for $0 \leq k \leq N-1$, then $\widehat{\boldsymbol{z}} = \overline{\widehat{\boldsymbol{x}}}$. In particular,
   (a) if $x_k = x_{N-k}$ for all n (so $\boldsymbol{x}$ is symmetric), then $\widehat{\boldsymbol{x}}$ is a real vector.
   (b) if $x_k = -x_{N-k}$ for all k (so $\boldsymbol{x}$ is antisymmetric), then $\widehat{\boldsymbol{x}}$ is a purely imaginary vector.
3. If d is an integer and $\boldsymbol{z}$ is the vector with components $z_k = x_{k-d}$ (the vector $\boldsymbol{x}$ with its elements delayed by d), then $(\widehat{\boldsymbol{z}})_n = e^{-2\pi i d n/N}(\widehat{\boldsymbol{x}})_n$.
4. If d is an integer and $\boldsymbol{z}$ is the vector with components $z_k = e^{2\pi i d k/N} x_k$, then $(\widehat{\boldsymbol{z}})_n = (\widehat{\boldsymbol{x}})_{n-d}$.

Proof. The methods used in the proof are very similar to those used in the proof of Theorem 1.36. From the definition of the DFT we have

$$\begin{aligned}(\widehat{\boldsymbol{x}})_{N-n} &= \frac{1}{\sqrt{N}}\sum_{k=0}^{N-1} e^{-2\pi i k(N-n)/N} x_k = \frac{1}{\sqrt{N}}\sum_{k=0}^{N-1} e^{2\pi i k n/N} x_k \\ &= \overline{\frac{1}{\sqrt{N}}\sum_{k=0}^{N-1} e^{-2\pi i k n/N} x_k} = \overline{(\widehat{\boldsymbol{x}})_n}\end{aligned}$$

which proves property 1. To prove property 2, we write

$$\begin{aligned}(\widehat{\boldsymbol{z}})_n &= \frac{1}{\sqrt{N}}\sum_{k=0}^{N-1} z_k e^{-2\pi i k n/N} = \frac{1}{\sqrt{N}}\sum_{k=0}^{N-1} x_{N-k} e^{-2\pi i k n/N} \\ &= \frac{1}{\sqrt{N}}\sum_{u=1}^{N} x_u e^{-2\pi i (N-u) n/N} = \frac{1}{\sqrt{N}}\sum_{u=0}^{N-1} x_u e^{2\pi i u n/N} \\ &= \overline{\frac{1}{\sqrt{N}}\sum_{u=0}^{N-1} x_u e^{-2\pi i u n/N}} = \overline{(\widehat{\boldsymbol{x}})_n}.\end{aligned}$$

If $\boldsymbol{x}$ is symmetric it follows that $\boldsymbol{z} = \boldsymbol{x}$, so that $(\widehat{\boldsymbol{x}})_n = \overline{(\widehat{\boldsymbol{x}})_n}$. Therefore $\boldsymbol{x}$ must be real. The case of antisymmetry follows similarly.

To prove property 3 we observe that

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

For the proof of property 4 we note that the DFT of $\boldsymbol{z}$ is

$$(\widehat{\boldsymbol{z}})_n = \frac{1}{\sqrt{N}}\sum_{k=0}^{N-1} e^{2\pi i d k/N} x_n e^{-2\pi i k n/N} = \frac{1}{\sqrt{N}}\sum_{k=0}^{N-1} x_n e^{-2\pi i (n-d) k/N} = (\widehat{\boldsymbol{x}})_{n-d}.$$

This completes the proof. □

For real sequences, Property 1 says that we need to store only about one half of the DFT coefficients, since the remaining coefficients can be obtained by conjugation. In particular, when N is even, we only need to store $y_0, y_1, \ldots, y_{N/2}$, since the other coefficients can be obtained by conjugating these. The theorem generalizes the properties from Theorem 1.36, except for the last property where the signal had a point of symmetry. We will delay the generalization of this property to later.

Exercises for Section 2.4

1. Compute the 4 point DFT of the vector $(2,3,4,5)^T$.

2. As in Example 2.19, state the exact cartesian form of the Fourier matrix for the cases $N=6$, $N=8$, and $N=12$.

3. We have a real vector $\boldsymbol{x}$ with length N, and define the vector $\boldsymbol{z}$ by delaying all elements in $\boldsymbol{x}$ with 5 cyclically, i.e. $z_5 = x_0$, $z_6 = x_1$,...,$z_N = x_{N-5}$, and $z_0 = x_{N-4}$,...,$z_4 = x_{N-1}$. If $|(F_N\boldsymbol{x})_n| = 2$, what is then $|(F_N\boldsymbol{z})_n|$? Justify the answer.

4. (Exam UIO V2012) Given a real vector $\boldsymbol{x}$ of length 8 where $(F_8(\boldsymbol{x}))_2 = 2-i$, what is $(F_8(\boldsymbol{x}))_6$?

5. Let $\boldsymbol{x}$ be the vector with entries $x_k = c^k$. Show that the DFT of $\boldsymbol{x}$ is given by the vector with components

$$y_n = \frac{1}{\sqrt{N}}\frac{1-c^N}{1-ce^{-2\pi in/N}}$$

for $n = 0, \ldots, N-1$.

6. If $\boldsymbol{x}$ is complex, Write the DFT in terms of the DFT on real sequences. Hint: Split into real and imaginary parts, and use linearity of the DFT.

7. As in Example 2.20, write a function

```
function x=IDFTImpl(y)
```

which computes the IDFT.

8. Let $\boldsymbol{x}_1, \boldsymbol{x}_2$ be real vectors, and set $\boldsymbol{x} = \boldsymbol{x}_1 + i\boldsymbol{x}_2$. Use Theorem 2.21 to show that

$$\begin{aligned}(F_N(\boldsymbol{x}_1))_k &= \frac{1}{2}\left((F_N(\boldsymbol{x}))_k + \overline{(F_N(\boldsymbol{x}))_{N-k}}\right)\\ (F_N(\boldsymbol{x}_2))_k &= \frac{1}{2i}\left((F_N(\boldsymbol{x}))_k - \overline{(F_N(\boldsymbol{x}))_{N-k}}\right)\end{aligned}$$

This shows that, sometimes, one DFT can be used to compute two different DFT's.

2.5 Connection between the DFT and Fourier series

So far we have focused on the DFT as a tool to rewrite a vector in terms of the Fourier basis vectors. In practice, the given vector $\boldsymbol{x}$ will often be sampled from some real data given by a function $f(t)$. We may then compare the frequency content of the vector $\boldsymbol{x}$ and the frequency content of f, and ask how these are related: What is the relationship between the Fourier coefficients of f and the DFT-coefficients of $\boldsymbol{x}$?

In order to study this, assume for simplicity that f lies in a Fourier space $V_{M,T}$ for some M. This means that f equals its Fourier approximation f_M,

$$f(t) = f_M(t) = \sum_{n=-M}^{M} z_n e^{2\pi i n t/T}, \tag{2.6}$$

where

$$z_n = \frac{1}{T}\int_0^T f(t)e^{-2\pi i n t/T}\,dt.$$

We here have changed our notation for the Fourier coefficients form z_n to y_n, in order not to confuse them with the DFT coefficients. We recall that in order to represent the frequency n/T fully, we need the corresponding exponentials with both positive and negative arguments, i.e., both $e^{2\pi i n t/T}$ and $e^{-2\pi i n t/T}$.

Fact 2.22. Suppose f is given by its Fourier series (2.6). Then the total frequency content for the frequency n/T is given by the two coefficients z_n and z_{-n}.

The following connection between the Fourier coefficients of f and the DFT of the samples of f also states how the DFT can be used to compute a Fourier series.

Proposition 2.23 (Relation between Fourier coefficients and DFT coefficients)**.** Let

$$f(t) = \sum_{n=-M}^{M} z_n e^{2\pi i n t/T},$$

be a function in $V_{M,T}$, and let $N = 2M+1$. Suppose that $\boldsymbol{x}$ are N uniform samples from f over one period, i.e.

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

and let $\boldsymbol{y}$ be the DFT of $\boldsymbol{x}$. Then $\boldsymbol{z} = (y_{M+1}, \ldots, y_{2M}, y_0, \ldots, y_M)/\sqrt{N}$. In particular, the total contribution to f from frequency n/T, where n is an integer in the range $0 \le n \le M$, is given by y_n and y_{N-n}.

Proof. The vector $\boldsymbol{x}$ can be expressed in terms of its DFT $\boldsymbol{y}$ as

$$x_k = \frac{1}{\sqrt{N}}\sum_{n=0}^{N-1} y_n e^{2\pi i n k/N}. \tag{2.7}$$

If we evaluate f at the sample points we have

$$f(kT/N) = \sum_{n=-M}^{M} z_n e^{2\pi ink/N}, \tag{2.8}$$

and a comparison now gives

$$\sum_{n=-M}^{M} z_n e^{2\pi ink/N} = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} y_n e^{2\pi ink/N} \quad \text{for } k = 0, 1, \ldots, N-1.$$

Exploiting the fact that both $\boldsymbol{y}$ and the complex exponentials are periodic with period N, we can rewrite this as

$$\sum_{n=-M}^{M} z_n e^{2\pi ink/N} = \frac{1}{\sqrt{N}} \sum_{n=-M}^{-1} y_{n+2M+1} e^{2\pi ink/N} + \frac{1}{\sqrt{N}} \sum_{n=0}^{M} y_n e^{2\pi ink/N}.$$

This system of N equations can be written on the form

$$G\boldsymbol{z} = G(y_{M+1}, \ldots, y_{2M}, y_0, \ldots, y_M)/\sqrt{N},$$

where G is the $N \times N$-matrix with entries $\frac{1}{\sqrt{N}} e^{2\pi ink/N}$, $-M \leq n < M$. It is straightforward to show that the matrix G is invertible (we can argue in the same way as we did when we showed that the Fourier basis was orthonormal), so that $\boldsymbol{z} = (y_{M+1}, \ldots, y_{2M}, y_0, \ldots, y_M)/\sqrt{N}$. This proves the result. $\square$

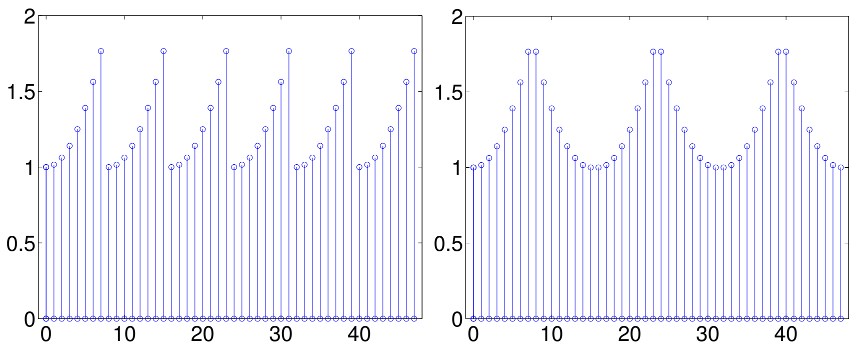
The proof above bases itself on taking $N = 2M + 1$ samples. However, if we take more samples (including the old samples), say $N = 2K + 1$ samples with $K > M$, it is clear that we will get the Fourier coefficients in $V_{K,T}$ (since $V_{M,T} \subset V_{K,T}$), where the new coefficients are 0.

In section 1.6 we mentioned the role of the Fourier series approximation f_N as a best approximation to f. We usually can't compute f_N exactly, however, since this requires us to compute the Fourier integrals. What we instead could do is to take the samples $\boldsymbol{x}$. If N is high, f_N is a good approximation to f, so that $\boldsymbol{x}$ is a good approximation to the samples of f_N. By continuity of the DFT it follows that, with $\boldsymbol{y} = F_N\boldsymbol{x}$, $(y_{M+1}, \ldots, y_{2M}, y_0, \ldots, y_M)/\sqrt{N}$ is a good approximation to the Fourier coefficients $\boldsymbol{z}$, so that $\tilde{f}(t) = \sum_{n=0}^{N-1} y_n e^{2\pi int/T}$ is a good approximation to f_N, and therefore also to f. We have illustrated this in Figure 2.1. The new function $\tilde{f}$ has the same values as f in the sample points, but note that it may not be equal to f_N. The interpolating function $\tilde{f}$ is thus another kind of approximation to f, and very useful, since it can be computed without evaluating the Fourier integrals, contrary to the Fourier series:

Idea 2.24. The function $\tilde{f}$ resulting from sampling, taking the DFT, and reconstructing, as shown in Figure 2.1, also gives an approximation to f. $\tilde{f}$ is a worse approximation to f than f_N (since we already know that f_N is a best approximation), but it is much more useful since it avoids evaluation of the Fourier integrals.

$$\begin{tikzcd} f \arrow[r] \arrow[d, "s"] & \tilde{f} \\ \boldsymbol{x} \arrow[r, "F_N"] & \boldsymbol{Y} \arrow[u, "r"'] \end{tikzcd}$$

Figure 2.1: How we can interpolate f from $V_{M,T}$ with help of the DFT. The vertical arrows represent sampling (s) and reconstruction (r), reconstruction means that we compute $\sum_n y_n e^{2\pi i n t/T}$ for values of t.


(a) Perodic extension of $\boldsymbol{x}$

(b) Symmetric extension of $\boldsymbol{x}$

Figure 2.2: The two different extensions of $\boldsymbol{x}$ to a periodic vector.

In Section 1.6 we also mentioned that a function often is defined on an interval $[0, T)$, not as a periodic function on the whole real line. The same is the case for digital sound. The extension strategies for a vector parallel those for functions from Figure 1.10, and are shown in Figure 2.2. The vector simply consists of 8 samples from the function from Figure 1.10. The symmetric extension has the original vector $\boldsymbol{x}$ as its first half, and a copy of $\boldsymbol{x}$ in reverse order as its second half:

Definition 2.25 (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

$$\breve{\boldsymbol{x}}_k = \begin{cases} x_k & 0 \le k < N \\ x_{2N-1-k} & N \le k < 2N-1 \end{cases} \tag{2.9}$$

This is not the only way to construct a symmetric extension, as we will return to later.

Another approximation strategy could be to first form the symmetric extension $\breve{f}$, and then $\tilde{\breve{f}}$ through sampling, the DFT, and reconstruction, as above.

$$\begin{tikzcd}
f \arrow[r, "?"] \arrow[d] & \tilde{f} \\
\breve{f} \arrow[d, "s"] & \tilde{\breve{f}} \arrow[u] \\
\boldsymbol{x} \arrow[r, "F_{2N}"] & \boldsymbol{Y} \arrow[u, "r"']
\end{tikzcd}$$

Figure 2.3: How we alternatively can approximate a function using the symmetric extension $\breve{f}$ instead.

Since $\breve{f}_N$ is a better approximation to f on $[0, T]$ than f_N (as previously argued), the sample points $\boldsymbol{x}$ of $\breve{f}$ are even closer to the sample points of $\breve{f}_N$. Again, by continuity of F_N, the DFT $\boldsymbol{y}$ of $\boldsymbol{x}$ gives an even better approximation $\sum_n y_n e^{2\pi int/T}$ to $\breve{f}_N(t) = \sum_n z_n e^{2\pi int/T}$. Then $\sum_n y_n e^{2\pi int/T}$ is also a better approximation to $\breve{f}$ (and thus f) than what we have obtained before, since $\breve{f}_N$ is a better approximation to $\breve{f}$ than f_N is to f. We have illustrated the new approximation scheme in Figure 2.3, where the old strategy from Figure 2.1 also is shown in the lower two lines. One may believe that there also exists an operation similar to the DFT for this scheme, indicated with a ? in the figure. This operation could hopefully be made N-dimensional: Although we take $2N$ samples from $\breve{f}$, the last part is equal to the first part, so that N dimensions are in play. If we succeed in finding such an operation, it would be even more important than the DFT for approximation of functions. We will return to how this operation can be constructed in Chapter 4. We already now can use the DFT to describe such an operation (since we only need to look at the cosine-series), but we would also ensure that the operation is orthogonal.

2.6 Using the DFT to adjust frequencies in sound

Before we continue we need a remark on what we should interpret as high and low frequency contributions, when we have applied a DFT. The low "frequency contribution" in f is the contribution from

$$e^{-2\pi iLt/T}, \ldots, e^{-2\pi it/T}, 1, e^{2\pi it/T}, \ldots, e^{2\pi iLt/T}$$

in f, i.e. $\sum_{n=-L}^{L} z_n e^{2\pi int/T}$. This means that low frequencies correspond to indices n so that $-L \leq n \leq L$. However, since DFT coefficients have indices between 0 and $N-1$, low frequencies correspond to indices n in $[0, L] \cup [N-L, N-1]$. If we make the same argument for high frequencies, we see that they correspond to DFT indices near $N/2$:

Observation 2.26 (DFT indices for high and low frequencies)**.** When $\boldsymbol{y}$ is the DFT of $\boldsymbol{x}$, the low frequencies in $\boldsymbol{x}$ correspond to the indices in $\boldsymbol{y}$ near 0 and N. The high frequencies in $\boldsymbol{x}$ correspond to the indices in $\boldsymbol{y}$ near $N/2$.

We will use this observation in the following example, when we use the DFT to distinguish between high and low frequencies in a sound.

Example 2.27 (Using the DFT to adjust frequencies in sound)**.** Since the DFT coefficients represent the contribution in a sound at given frequencies, we can listen to the different frequencies of a sound by adjusting the DFT coefficients. Let us first see how we can listen to the lower frequencies only. As explained, these correspond to DFT-indices n in $[0, L] \cup [N-L, N-1]$. In Matlab these have indices from 1 to $L+1$, and from $N-L+1$ to N. The remaining frequencies, i.e. the higher frequencies which we want to eliminate, thus have Matlab-indices between $L+2$ and $N-L$. We can now perform a DFT, eliminate these high frequencies, and perform an inverse DFT, to recover the sound signal where these frequencies have been eliminated. With the help of the DFT implementation from Example 2.20, all this can be achieved with the following code:

```
y=DFTImpl(x);
y((L+2):(N-L))=zeros(N-(2*L+1),1);
newx=IDFTImpl(y);
```

To test this in practice, we also need to obtain the actual sound samples. If we use our sample file `castanets.wav`, you will see that the code runs very slowly. In fact it seems to never complete. The reason is that `DFTImpl` attempts to construct a matrix F_N with as many rows and columns as there are sound samples in the file, and there are just too many samples, so that F_N grows too big, and matrix multiplication with it gets too time-consuming. We will shortly see much better strategies for applying the DFT to a sound file, but for now we will simply attempt instead to split the sound file into smaller blocks, each of size $N = 32$, and perform the code above on each block. This is less time-consuming, since big matrices are avoided. You will be spared the details for actually splitting the sound file into blocks: you can find the function `playDFTlower(L)` which performs this splitting, sets the relevant frequency components to 0, and plays the resulting sound samples. If you try this for $L = 7$ (i.e. we keep only 15 of the DFT coefficients) the result sounds like this. You can hear the disturbance in the sound, but we have not lost that much even if more than half the DFT coefficients are dropped. If we instead try $L = 3$ the result will sound like this. The quality is much poorer now. However we can still recognize the song, and this suggests that most of the frequency information is contained in the lower frequencies.

Similarly we can listen to high frequencies by including only DFT coefficients with index close to $\frac{N}{2}$. The function `playDFThigher(L)` sets all DFT coefficients to zero, except for those with indices $\frac{N}{2} - L, \ldots, \frac{N}{2}, \ldots, \frac{N}{2} + L$. Let us verify that there is less information in the higher frequencies by trying the same values for L as above for this function. For $L = 7$ (i.e. we keep only the middle 15 DFT coefficients) the result sounds like this, for $L = 3$ the result sounds like this. Both sounds are quite unrecognizable, confirming that most information is contained in the lower frequencies. ♣

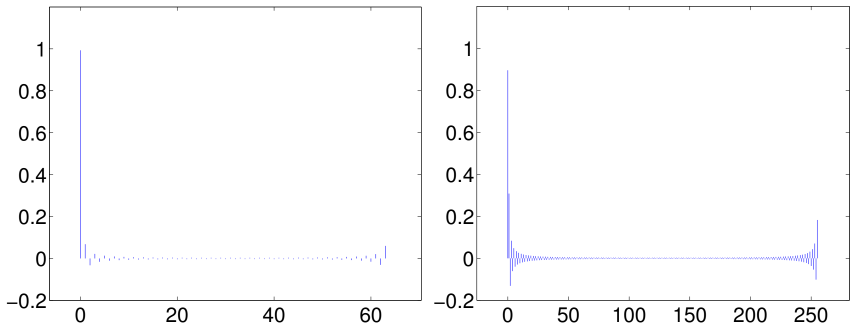
Note that there may be a problem in the previous example: when we restrict to the values in a given block, we actually look at a different signal. The new signal repeats the values in the block in periods, while the old signal consists of one much bigger block. What are the differences in the frequency representations of the two signals?

Assume that the entire sound has length M. The frequency representation of this is computed as an M-point DFT (the signal is actually repeated with period M), and we write the sound samples as a sum of frequencies: $x_k = \sum_{n=0}^{M-1} y_n e^{2\pi ikn/M}$. Let us consider the effect of restricting to a block for each of the contributing pure tones $\frac{1}{\sqrt{N}} e^{2\pi ikn_0/M}$, $0 \leq n_0 \leq M-1$ (where we have scaled with $\frac{1}{\sqrt{N}}$). When we restrict this to a block of size N, we get the signal $\{e^{2\pi ikn_0/M}\}_{k=0}^{N-1}$. Depending on n_0, this may not be a Fourier basis vector! Its N-point DFT gives us its frequency representation, and the absolute value of this is

$$\begin{aligned}
|y_n| &= \left| \frac{1}{N} \sum_{k=0}^{N-1} e^{2\pi ikn_0/M} e^{-2\pi ikn/N} \right| = \left| \frac{1}{N} \sum_{k=0}^{N-1} e^{2\pi ik(n_0/M - n/N)} \right| \\
&= \left| \frac{1}{N} \frac{1 - e^{2\pi iN(n_0/M - n/N)}}{1 - e^{2\pi i(n_0/M - n/N)}} \right| = \frac{1}{N} \left| \frac{\sin(\pi N(n_0/M - n/N))}{\sin(\pi(n_0/M - n/N))} \right|. \qquad (2.10)
\end{aligned}$$

If $n_0 = M/N$, this gives $y_{n_0} = 1$, and $y_n = 0$ when $n \neq n_0$. Thus, splitting the signal into blocks gives another pure tone when $n_0 = M/N$. When n_0 is different from M/N the situation is different. Let us set $M = 1000$, $n_0 = 1$, and experiment with different values of N. Figure 2.4 shows the y_n values for different values of N. We see that the frequency representation is now very different, and that many frequencies contribute. The explanation is that the pure tone is not a pure tone when $N = 64$ and $N = 256$, since at this scale such frequencies are too high to be represented exactly. The closest pure tone in frequency is $n = 0$, and we see that this has the biggest contribution, but other frequencies also contribute. The other frequencies contribute much more when $N = 256$, as can be seen from the peak in the closest frequency $n = 0$. In conclusion, when we split into blocks, the frequency representation may change in an undesirable way. This is a common problem in signal processing theory, that one in practice needs to restrict to smaller segments of samples, but that this restriction may have undesired effects.

Another problem when we restrict to a shorter periodic signal is that we may obtain discontinuities at the boundaries between the new periods, even if there were no discontinuities in the original signal. And, as we know from the square wave, discontinuities introduce undesired frequencies. We have already mentioned that symmetric extensions may be used to remedy this.



(a) N=64 (b) N=256

Figure 2.4: The frequency representation obtained when restricting to a block of size N of the signal.

There are two other interesting facets to Theorem 2.23, besides connecting the DFT and the Fourier series. These are covered in the two next sections.

2.7 The DFT and interpolation

Theorem 2.23 enables us to find (unique) trigonometric functions which interpolate (pass through) a set of data points. We have in elementary calculus courses seen how to determine a polynomial of degree $N-1$ that interpolates a set of N data points — such polynomials are called interpolating polynomials. The following result tells how we can find an interpolating trigonometric function using the DFT.

Corollary 2.28 (Interpolation with the Fourier basis)**.** Let f be a function defined on the interval $[0,T]$, and let $\boldsymbol{x}$ be the sampled sequence given by

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

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

$$g(t) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} y_n e^{2\pi i n t/T}$$

which satisfies the conditions

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

and its coefficients are determined by the DFT $\boldsymbol{y}$ of $\boldsymbol{x}$.

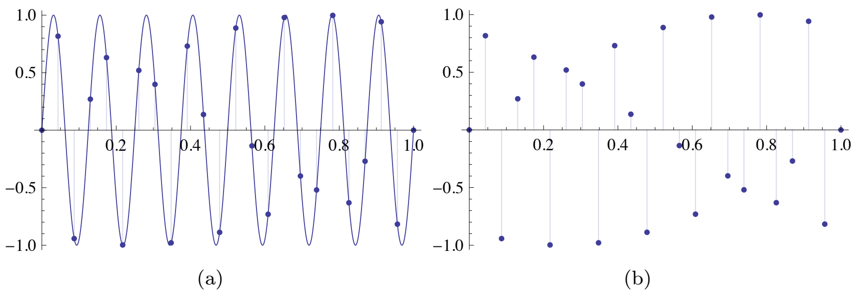


Figure 2.5: An example of sampling. Figure (a) shows how the samples are picked from an underlying continuous time function. Figure (b) shows what the samples look like on their own.

The proof for this follows by inserting $t = 0$, $t = T/N$, $t = 2T/N$, $\ldots$, $t = (N-1)T/N$ in the equation $f(t) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} y_n e^{2\pi int/T}$ to arrive at the equations

$$f(kT/N) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} y_n e^{2\pi ink/N} \qquad\qquad 0 \le k \le N-1.$$

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

2.8 Reconstruction of a function from its samples. The sampling theorem.

The second interesting facet to Theorem 2.23 has to do with when exact reconstruction of a function from its samples is possible. An example of sampling a function is illustrated in Figure 2.5. From Figure 2.5(b) it is clear that some information is lost when we discard everything but the sample values. There may however be an exception to this, if we assume that the function satisfies some property. Assume for instance that f is equal to a finite Fourier series. This means that f can be written on the form (2.6), so that the highest frequency in the signal is bounded by M/T. Our analysis prior to Theorem 2.23 states that all such functions can be reconstructed exactly from their samples, as long as the number of samples is $N \ge 2M+1$, taken uniformly over a period. Moreover, the DFT is central in the reconstruction formula. Dividing by T we get $\frac{N}{T} \ge \frac{2M+1}{T}$, which states that the sampling frequency ($f_s = N/T$ is the number of samples per second) should be bigger than two times the highest frequency (M/T). In Figure 2.6 we try to get some intuition on this by considering some pure tones. In (a) we consider one period of $\sin 2\pi t$, and see that we need at least two sample points in $[0,1]$: one point would clearly be too little.

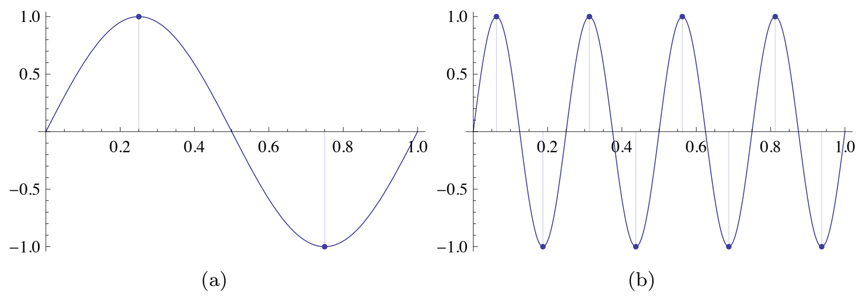


Figure 2.6: Sampling the function $\sin 2\pi t$ with two points, and the function $\sin 2\pi 4t$ with eight points.

This translates directly into having at least eight sample points in (b) where the function is $\sin 2\pi 4t$, which has four periods in the interval $[0,1]$.

Let us restate the reconstruction of f, so that it only uses the samples. The reconstruction formula was

$$f(t) = \frac{1}{\sqrt{N}} \sum_{n=-M}^{M} z_n e^{2\pi int/T},$$

where the z_n can be found from the equations

$$f(kT_s) = \frac{1}{\sqrt{N}} \sum_{n=-M}^{M} z_n e^{2\pi ink/N} \qquad\qquad -M \le k \le M,$$

where we have substituted $N = T/T_s$ (deduced from $T = NT_s$ with T_s being the sampling period). From this equation we see that

$$z_n = \frac{1}{\sqrt{N}} \sum_{k=-M}^{M} f(kT_s) e^{-2\pi ink/N},$$

and inserting this in the reconstruction formula we get

$$\begin{aligned}
f(t) &= \frac{1}{N} \sum_{n=-M}^{M} \sum_{k=-M}^{M} f(kT_s) e^{-2\pi ink/N} e^{2\pi int/T} \\
&= \sum_{k=-M}^{M} \frac{1}{N} \left(\sum_{n=-M}^{M} f(kT_s) e^{2\pi in(t/T - k/N)} \right) \\
&= \sum_{k=-M}^{M} \frac{1}{N} e^{-2\pi iM(t/T-k/N)} \frac{1 - e^{2\pi iN(t/T-k/N)}}{1 - e^{2\pi i(t/T-k/N)}} f(kT_s) \\
&= \sum_{k=-M}^{M} \frac{1}{N} \frac{\sin(\pi(t-kT_s)/T_s)}{\sin(\pi(t-kT_s)/T)} f(kT_s)
\end{aligned}$$

Let us summarize our findings as follows:

Theorem 2.29 (Sampling theorem and the ideal interpolation formula for periodic functions)**.** Let f be a periodic function with period T, and assume that f has no frequencies higher than νHz. Then f can be reconstructed exactly from its samples $f(-MT_s), \ldots, f(MT_s)$ (where T_s is the sampling period, $N = \frac{T}{T_s}$ is the number of samples per period, and $M = 2N + 1$) when the sampling rate $f_s = \frac{1}{T_s}$ is bigger than 2ν. Moreover, the reconstruction can be performed through the formula

$$f(t) = \sum_{k=-M}^{M} f(kT_s) \frac{1}{N} \frac{\sin(\pi(t - kT_s)/T_s)}{\sin(\pi(t - kT_s)/T)}. \qquad (2.11)$$

Formula (2.12) is also called the ideal interpolation formula for periodic functions. Such formulas, where one reconstructs a function based on a weighted sum of the sample values, are more generally called *interpolation formulas*. The function $\frac{1}{N}\frac{\sin(\pi(t-kT_s)/T_s)}{\sin(\pi(t-kT_s)/T)}$ is also called an interpolation kernel. Note that f itself may not be equal to a finite Fourier series, and reconstruction is in general not possible then. The ideal interpolation formula can in such cases still be used, but the result we obtain may be different from $f(t)$.

It turns out that the interpolation formula above can be rewritten without the dependence on T and N, i.e. so that the interpolation formula is valid for all numbers of samples. This formula is what is usually listed in the literature:

Theorem 2.30 (Sampling theorem and the ideal interpolation formula for periodic functions, general version)**.** Assume that f is periodic with period T, and has no frequencies higher than νHz. Then f can be reconstructed exactly from its samples $\ldots, f(-2T_s), f(-T_s), f(0), f(T_s), f(2T_s), \ldots$ when T is a multiple of T_s, and when the sampling rate is bigger than 2ν. Moreover, the reconstruction can be performed through the formula

$$f(t) = \sum_{k=-M}^{M} f(kT_s) \frac{\sin(\pi(t - kT_s)/T_s)}{\pi(t - kT_s)/T_s}. \qquad (2.12)$$

Proof. Note first that f can also be viewed as a function with period sT for any integer $s > 1$. Writing sT for T, and sN for N in the previous interpolation formula, we get

$$\begin{aligned}
f(t) &= \sum_{k=-sM}^{sM} f(kT_s)\frac{1}{sN}\frac{\sin(\pi(t-kT_s)/T_s)}{\sin(\pi(t-kT_s)/(sT))}\\
&= \sum_{k=-rM}^{rM} f(kT_s)\frac{1}{sN}\frac{\sin(\pi(t-kT_s)/T_s)}{\sin(\pi(t-kT_s)/(sT))}\\
&+ \sum_{rM<|k|\leq sM} f(kT_s)\frac{1}{sN}\frac{\sin(\pi(t-kT_s)/T_s)}{\sin(\pi(t-kT_s)/(sT))},
\end{aligned}$$

where we have split the summation further (r is a number smaller than s). Note that $\lim_{s\to\infty} sN\sin(\pi(t-kT_s)/(sT)) = \pi(t-kT_s)/T_s$. Thus if we let $s\to\infty$ while keeping r fixed, the first sum above converges to

$$\sum_{k=-rM}^{rM} f(kT_s)\frac{\sin(\pi(t-kT_s)/T_s)}{\pi(t-kT_s)/T_s}.$$

The second sum can be written as

$$\sum_{k=-M}^{M} f(kT_s)\sum_{r<|u|\leq s}\frac{1}{sN}\frac{\sin(\pi(t-(k+uM)T_s)/T_s)}{\sin(\pi(t-(k+uM)T_s)/(sT))}.$$

The numerator here turns the inner sum into an alternating series, and the denominator is increasing when r is chosen big enough at the start. This means that the sum converges, and is bounded by the first term

$$\begin{aligned}
&\frac{1}{sN\sin(\pi(t-(k+rM)T_s)/(sT))}\\
&\leq \frac{2}{sN\pi(t-(k+rM)T_s)/(sT)} = \frac{2T_s}{\pi(t-(k+rM)T_s)}.
\end{aligned}$$

For a given t, this can be made a small as we like, if we choose r big enough. □

In the literature one actually shows that this formula is valid also for functions which are not periodic.

2.9 The Fast Fourier Transform (FFT)

The main application of the DFT is as a tool to compute frequency information in large datasets. It is therefore important that these operations can be performed by efficient algorithms. Straightforward implementation from the definition is not efficient if the data sets are large. However, it turns out that the underlying matrices may be factored in a way that leads to much more efficient algorithms, and this is the topic of the present section, where we discuss the most widely used implementation of the DFT, usually referred to as the Fast

Fourier Transform (FFT). For simplicity, we will assume that N, the length of the vector that is to be transformed by the DFT, is a power of 2. In this case it is relatively easy to simplify the DFT algorithm via a factorisation of the Fourier matrix. The foundation is provided by a simple reordering of the DFT.

Theorem 2.31 (FFT algorithm)**.** Let $\boldsymbol{y} = F_N\boldsymbol{x}$ be the N-point DFT of $\boldsymbol{x}$ with N an even number. For any integer n in the interval $[0, N/2-1]$ the DFT $\boldsymbol{y}$ of $\boldsymbol{x}$ is then given by

$$y_n = \frac{1}{\sqrt{2}}\left((F_{N/2}\boldsymbol{x}^{(e)})_n + e^{-2\pi in/N}(F_{N/2}\boldsymbol{x}^{(o)})_n\right), \tag{2.13}$$

$$y_{N/2+n} = \frac{1}{\sqrt{2}}\left((F_{N/2}\boldsymbol{x}^{(e)})_n - e^{-2\pi in/N}(F_{N/2}\boldsymbol{x}^{(o)})_n\right), \tag{2.14}$$

where $\boldsymbol{x}^{(e)}, \boldsymbol{x}^{(o)}$ are the sequences of length $N/2$ consisting of the even and odd samples of $\boldsymbol{x}$, respectively. In other words,

$$\begin{aligned}(\boldsymbol{x}^{(e)})_k &= x_{2k} \text{ for } 0 \le k \le N/2-1,\\ (\boldsymbol{x}^{(o)})_k &= x_{2k+1} \text{ for } 0 \le k \le N/2-1.\end{aligned}$$

Put differently, the formulas (2.13)–(2.14) reduces the computation of an N-point DFT to two $N/2$-point DFT's. It turns out that this can speed up computations considerably, but let us first check that these formulas are correct.

Proof. Suppose first that $0 \le n \le N/2-1$. We start by splitting the sum in the expression for the DFT into even and odd indices,

$$\begin{aligned}
y_n &= \frac{1}{\sqrt{N}}\sum_{k=0}^{N-1} x_k e^{-2\pi ink/N}\\
&= \frac{1}{\sqrt{N}}\sum_{k=0}^{N/2-1} x_{2k}e^{-2\pi in2k/N} + \frac{1}{\sqrt{N}}\sum_{k=0}^{N/2-1} x_{2k+1}e^{-2\pi in(2k+1)/N}\\
&= \frac{1}{\sqrt{2}}\frac{1}{\sqrt{N/2}}\sum_{k=0}^{N/2-1} x_{2k}e^{-2\pi ink/(N/2)}\\
&\qquad\qquad + e^{-2\pi in/N}\frac{1}{\sqrt{2}}\frac{1}{\sqrt{N/2}}\sum_{k=0}^{N/2-1} x_{2k+1}e^{-2\pi ink/(N/2)}\\
&= \frac{1}{\sqrt{2}}\left(F_{N/2}\boldsymbol{x}^{(e)}\right)_n + \frac{1}{\sqrt{2}}e^{-2\pi in/N}\left(F_{N/2}\boldsymbol{x}^{(o)}\right)_n,
\end{aligned}$$

where we have substituted $\boldsymbol{x}^{(e)}$ and $\boldsymbol{x}^{(o)}$ as in the text of the theorem, and recognized the $N/2$-point DFT in two places. For the second half of the DFT coefficients, i.e. $\{y_{N/2+n}\}_{0\leq n\leq N/2-1}$, we similarly have

$$\begin{aligned}
y_{N/2+n} &= \frac{1}{\sqrt{N}}\sum_{k=0}^{N-1} x_k e^{-2\pi i(N/2+n)k/N} = \frac{1}{\sqrt{N}}\sum_{k=0}^{N-1} x_k e^{-\pi i k} e^{-2\pi i nk/N} \\
&= \frac{1}{\sqrt{N}}\sum_{k=0}^{N/2-1} x_{2k}e^{-2\pi i n 2k/N} - \frac{1}{\sqrt{N}}\sum_{k=0}^{N/2-1} x_{2k+1}e^{-2\pi i n(2k+1)/N} \\
&= \frac{1}{\sqrt{2}}\frac{1}{\sqrt{N/2}}\sum_{k=0}^{N/2-1} x_{2k}e^{-2\pi i nk/(N/2)} \\
&\qquad - e^{-2\pi i n/N}\frac{1}{\sqrt{2}}\frac{1}{\sqrt{N/2}}\sum_{k=0}^{N/2-1} x_{2k+1}e^{-2\pi i nk/(N/2)} \\
&= \frac{1}{\sqrt{2}}\left(F_{N/2}\boldsymbol{x}^{(e)}\right)_n - \frac{1}{\sqrt{2}}e^{-2\pi i n/N}\left(F_{N/2}\boldsymbol{x}^{(o)}\right)_n .
\end{aligned}$$

This concludes the proof. $\square$

It turns out that Theorem 2.31 can be interpreted as a matrix factorization. For this we need to define the concept of a block matrix.

Definition 2.32. Let $m_0, \ldots, m_{r-1}$ and $n_0, \ldots, n_{s-1}$ be integers, and let $A^{(i,j)}$ be an $m_i \times n_j$-matrix for $i = 0, \ldots, r-1$ and $j = 0, \ldots, s-1$. The notation

$$A = \left(\begin{array}{c|c|c|c}
A^{(0,0)} & A^{(0,1)} & \cdots & A^{(0,s-1)} \\ \hline
A^{(1,0)} & A^{(1,1)} & \cdots & A^{(1,s-1)} \\ \hline
\vdots & \vdots & \vdots & \vdots \\ \hline
A^{(r-1,0)} & A^{(r-1,1)} & \cdots & A^{(r-1,s-1)}
\end{array}\right)$$

denotes the $(m_0 + m_1 + \ldots + m_{r-1}) \times (n_0 + n_1 + \ldots + n_{s-1})$-matrix where the matrix entries occur as in the $A^{(i,j)}$ matrices, in the way they are ordered, and with solid lines indicating borders between the blocks. When A is written in this way it is referred to as a block matrix.

We will express the Fourier matrix in factored form involving block matrices. The following observation is just a formal way to split a vector into its even and odd components.

Observation 2.33. Define the permutation matrix P_N by

$$\begin{aligned}
(P_N)_{i,2i} &= 1, && \text{for } 0 \leq i \leq N/2-1; \\
(P_N)_{i,2i-N+1} &= 1, && \text{for } N/2 \leq i < N; \\
(P_N)_{i,j} &= 0, && \text{for all other } i \text{ and } j;
\end{aligned}$$

and let $\boldsymbol{x}$ be a column vector. The mapping $\boldsymbol{x} \to P\boldsymbol{x}$ permutes the components of $\boldsymbol{x}$ so that the even components are placed first and the odd components last,

$$P_N \boldsymbol{x} = \begin{pmatrix} \boldsymbol{x}^{(e)} \\ \boldsymbol{x}^{(o)} \end{pmatrix},$$

with $\boldsymbol{x}^{(e)}$, $\boldsymbol{x}^{(o)}$ defined as in Theorem 2.31.

Let $D_{N/2}$ be the $(N/2) \times (N/2)$-diagonal matrix with entries $(D_{N/2})_{n,n} = e^{-2\pi i n/N}$ for $n = 0, 1, \ldots, N/2 - 1$. It is clear from Equation (2.13) that the first half of $\boldsymbol{y}$ is then given by obtained as

$$\frac{1}{\sqrt{2}} \left(\begin{array}{c|c} F_{N/2} & D_{N/2} F_{N/2} \end{array} \right) P_N \boldsymbol{x},$$

and from Equation (2.14) that the second half of $\boldsymbol{y}$ can be obtained as

$$\frac{1}{\sqrt{2}} \left(\begin{array}{c|c} F_{N/2} & -D_{N/2} F_{N/2} \end{array} \right) P_N \boldsymbol{x}.$$

From these two formulas we can derive the promised factorisation of the Fourier matrix.

Theorem 2.34 (DFT matrix factorization)**.** The Fourier matrix may be factored as

$$F_N = \frac{1}{\sqrt{2}} \left(\begin{array}{c|c} F_{N/2} & D_{N/2} F_{N/2} \\ \hline F_{N/2} & -D_{N/2} F_{N/2} \end{array} \right) P_N. \tag{2.15}$$

This factorization in terms of block matrices is commonly referred to as the *FFT factorization* of the Fourier matrix. In implementations, this factorization is typically repeated, so that $F_{N/2}$ is replaced with a factorization in terms of $F_{N/4}$, this again with a factorization in terms of $F_{N/8}$, and so on.

The input vector $\boldsymbol{x}$ to the FFT algorithm is mostly assumed to be real. In this case, the second half of the FFT factorization can be simplified, since we have shown that the second half of the Fourier coefficients can be obtained by symmetry from the first half. In addition we need the formula

$$y_{N/2} = \frac{1}{\sqrt{N}} \sum_{n=0}^{N/2-1} \left((\boldsymbol{x}^{(e)})_n - (\boldsymbol{x}^{(o)})_n \right)$$

to obtain coefficient $\frac{N}{2}$, since this is the only coefficient which can't be obtained from $y_0, y_1, \ldots, y_{N/2-1}$ by symmetry.

In an implementation based on formula (2.15), we would first compute $P_N \boldsymbol{x}$, which corresponds to splitting $\boldsymbol{x}$ into the even-indexed and odd-indexed samples. The two leftmost blocks in the block matrix in (2.15) correspond to applying the $\frac{N}{2}$-point DFT to the even samples. The two rightmost blocks correspond to applying the $N/2$-point DFT to the odd samples, and multiplying the result with $D_{N/2}$. The results from these transforms are finally added together. By repeating the splitting we will eventually come to the case where $N = 1$. Then F_1 is just the scalar 1, so the DFT is the trivial assignment $y_0 = x_0$. The FFT can therefore be implemented with the following Matlab code:

```
function y = FFTImpl(x)
N = length(x);
if N == 1
    y = x(1);
else
    xe = x(1:2:(N-1));
    xo = x(2:2:N);
    ye = FFTImpl(xe);
    yo = FFTImpl(xo);
    D=exp(-2*pi*1j*(0:N/2-1)'/N);
    y = [ ye + yo.*D; ye - yo.*D]/sqrt(2);
end
```

Note that this function is recursive; it calls itself. If this is your first encounter with a recursive program, it is worth running through the code manually for a given value of N, such as $N = 4$.

2.9.1 The Inverse Fast Fourier Transform (IFFT)

The IDFT is very similar to the DFT, and it is straightforward to prove the following analog to Theorem 2.31 and (2.15).

Theorem 2.35 (IDFT matrix factorization)**.** The inverse of the Fourier matrix can be factored as

$$(F_N)^H = \frac{1}{\sqrt{2}} \left(\begin{array}{c|c} (F_{N/2})^H & E_{N/2}(F_{N/2})^H \\ (F_{N/2})^H & -E_{N/2}(F_{N/2})^H \end{array} \right) P_N, \tag{2.16}$$

where $E_{N/2}$ is the $(N/2) \times (N/2)$-diagonal matrix with entries given by $(E_{N/2})_{n,n} = e^{2\pi in/N}$, for $n = 0, 1, \ldots, N/2 - 1$.

We note that the only difference between the factored forms of F_N and F_N^H is the positive exponent in $e^{2\pi in/N}$. With this in mind it is straightforward to modify `FFTImpl.m` so that it performs the inverse DFT.

Matlab has built-in functions for computing the DFT and the IDFT using the FFT algorithm. These functions are called `fft` and `ifft`. Note, however, that these functions do not use the normalizing factor $1/\sqrt{N}$ that we have adopted here. The Matlab help pages give a short description of these algorithms. Note in particular that Matlab makes no assumption about the length of the vector. Matlab may however check if the length of the vector is 2^r, and in those cases a variant of the algorithm discussed here is used. In general, fast algorithms exist when the vector length N can be factored as a product of small integers.

Many audio and image formats make use of the FFT. To get optimal speed these algorithms typically split the signals into blocks of length 2^r with r some integer in the range 5–10 and utilise a suitable variant of the algorithms discussed above.

2.9.2 Reduction in the number of multiplications with the FFT

Before we continue we also need to explain why the FFT and IFFT factorizations lead to more efficient implementations than the direct DFT and IDFT implementations. We first need some terminology for how we count the number of operations of a given type in an algorithm. In particular we are interested in the limiting behaviour when N becomes large, which is the motivation for the following definition.

Definition 2.36 (Order of an algorithm)**.** Let R_N be the number of operations of a given type (such as multiplication or addition) in an algorithm, where N describes the dimension of the data in the algorithm (such as the size of the matrix or length of the vector), and let f be a positive function. The algorithm is said to be of order N, also written $O(f(N))$, if the number of operations grows as $f(N)$ for large N, or more precisely, if

$$\lim_{N\to\infty} \frac{R_N}{f(N)} = 1.$$

We will also use this notation for functions, and say that a real function g is $O(f(\boldsymbol{x}))$ if $\lim g(\boldsymbol{x})/f(\boldsymbol{x}) = 0$ where the limit mostly will be taken as $\boldsymbol{x} \to \boldsymbol{0}$ (this means that $g(\boldsymbol{x})$ is much smaller than $f(\boldsymbol{x})$ when $\boldsymbol{x}$ approaches the limit).

Let us see how we can use this terminology to describe the complexity of the FFT algorithm. Let M_N be the number of multiplications needed by the N-point FFT as defined by Theorem 2.31. It is clear from the algorithm that

$$M_N = 2M_{N/2} + N/2. \tag{2.17}$$

The factor 2 corresponds to the two matrix multiplications, while the term $N/2$ denotes the multiplications in the exponent of the exponentials that make up the matrix $D_{N/2}$ (or $E_{N/2}$). The exponent $2\pi i/N$ may be computed once and for all outside the loops, and has therefore not been counted. Also, we have not counted the multiplications with `1/sqrt(2)`. The reason is that, in most implementations, this factor is absorbed in the definition of the DFT itself.

Note that all multiplications performed by the FFT are complex. It is normal to count the number of real multiplications instead, since any multiplication of two complex numbers can be performed as four multiplications of real numbers (and two additions), by writing the number in terms of its real and imaginary part, and myltiplying them together. Therefore, if we instead define M_N to be the number of real multiplications required by the FFT, we obtain the alternative recurrence relation

$$M_N = 2M_{N/2} + 2N. \tag{2.18}$$

In Exercise 2 you will be asked to derive the solution of this equation and show that the number of real multiplications required by this algorithm is $O(2N \log_2 N)$. In contrast, the direct implementation of the DFT requires N^2 complex multiplications, and thus $4N^2$ real multiplications. The exact same numbers are found for the IFFT.

Theorem 2.37 (Number of operations in the FFT and IFFT algorithms)**.** The N-point FFT and IFFT algorithms both require $O(2N \log_2 N)$ real multiplications. In comparison, the number of real multiplications required by direct implementations of the N-point DFT and IDFT is $4N^2$.

In other words, the FFT and IFFT significantly reduce the number of multiplications, and one can show in a similar way that the number of additions required by the algorithm is also roughly $O(N \log_2 N)$. This partially explains the efficiency of the FFT algorithm. Another reason is that since the FFT splits the calculation of the DFT into computing two DFT's of half the size, the FFT is well suited for parallel computing: the two smaller FFT's can be performed independently of one another, for instance in two different computing cores on the same computer.

2.9.3 Applications of the FFT

The FFT has been stated as one the ten most important inventions of the 20'th century. With its invention, the Discrete Fourier Transform was within computational reach in many fields. Real time processing is one important apllication, such as processing of sound, image, and video.

Compression of sound is one of the many important applications of the FFT. The MP3 standard uses it to find the different frequency components in sound, and to macth this information with a psychoachoustic model, in order to find how the different data should be compressed. We will later see that a set of filters is applied to the sound data. It turns out that the MP3 standard applies the FFT to the sound data directly, not to the filtered data. The output of the FFT and the psychoachoustic model is then applied to the filtered data, to decide how this data should be compressed.

Exercises for Section 2.9

1. (Trial exam UIO Spring 2012)

a. Compute the DFT of the vectors $\boldsymbol{x}_1 = (1,3,5,7)$, and of $\boldsymbol{x}_2 = (2,4,6,8)$ (i.e. compute $F_4\boldsymbol{x}_1$ and $F_4\boldsymbol{x}_2$).

b. Explain how you can compute the DFT of the vector $(1,2,3,4,5,6,7,8)$ based on the computation from a. (you don't need to perform the actual computation). What are the benefits of this approach, and what is the algorithm called?

2. In this exercise we will compute the number of real multiplications needed by the FFT algorithm given in the text. The starting point will be the difference equation (2.18) for the number of real multiplications for an N-point FFT.

a. Explain why $x_r = M_{2^r}$ is the solution to the difference equation $x_{r+1} - 2x_r = 4 \cdot 2^r$.

b. Show that the general solution to the difference equation is $x_r = 2r2^r + C2^r$.

c. Explain why $M_N = O(2N\log_2 N)$ (you do not need to write down the initial conditions for the difference equation in order to find the particular solution).

3. When we wrote down the difference equation $M_N = 2M_{N/2} + 2N$ for the number of multiplications in the FFT algorithm, you could argue that some multiplications were not counted. Which multiplications in the FFT algorithm were not counted when writng down this difference equation? Do you have a suggestion to why these multiplications were not counted?

4. Write down a difference equation for computing the number of real additions required by the FFT algorithm.

5. It is of course not always the case that the number of points in a DFT is $N = 2^n$. In this exercise we will see how we can attack the more general case.

a. Assume that N can be divided by 3, and consider the following splitting, which follows in the same way as the splitting used in the deduction of the FFT-algorithm:

$$
\begin{aligned}
y_n &= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} x_k e^{-2\pi i n k/N} \\
&= \frac{1}{\sqrt{N}} \sum_{k=0}^{N/3-1} x_{3k} e^{-2\pi i n 3k/N} + \frac{1}{\sqrt{N}} \sum_{k=0}^{N/3-1} x_{3k+1} e^{-2\pi i n (3k+1)/N} \\
&+ \frac{1}{\sqrt{N}} \sum_{k=0}^{N/3-1} x_{3k+2} e^{-2\pi i n (3k+2)/N}
\end{aligned}
$$

Find a formula which computes $y_0, y_1, \ldots, y_{N/3-1}$ by performing 3 DFT's of size $N/3$.

b. Find similar formulas for computing $y_{N/3}, y_{N/3+1}, \ldots, y_{2N/3-1}$, and $y_{2N/3}, y_{2N/3}+1, \ldots, y_{N-1}$. State a similar factorization of the DFT matrix as in Theorem 2.34, but this time where the matrix has 3×3 blocks.

c. Assume that $N = 3^n$, and that you implement the FFT using the formulas you have deduced in a. and b.. How many multiplications does this algorithm require?

d. Sketch a general procedure for speeding up the computation of the DFT, which uses the factorization of N into a product of prime numbers.

2.10 Summary

We defined digital sound, and demonstrated how we could perform simple operations on digital sound such as adding noise, playing at different rates e.t.c.. Digital sound could be obtained by sampling the sounds from the previous chapter. We considered the analog of Fourier series for digital sound, which is called the Discrete Fourier Transform, and looked at its properties and its relation to Fourier series. We also saw that the sampling theorem guaranteed that there is no loss in considering the samples of a function, as long as the sampling rate is high enough compared to the highest frequency in the sound. We also obtained an implementation of the DFT, called the FFT, which is more efficient in terms of the number of arithmetic operations than a direct implementation of the DFT. The FFT has been cited as one of the ten most important algorithms of the 20'th century [3]. The original paper [4] by Cooley and Tukey dates back to 1965.

Chapter 3

Operations on digital sound: digital filters

In Section 1.8 we defined analog filters as operations on sound which preserved different frequencies. Such operations are important since they can change the frequency content in many ways. Analog filters can not be used 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 analog filters computable. It turns out that what we will define as 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*. We will demonstrate that it smooths the variations in the sound, and this is where it gets its name from. We will start this chapter by 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 analog filters, and show that the formal definition is equivalent to such operations.

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 $\boldsymbol{x} = (x_0, x_1, \dots, x_{N-1})$ to the vector $\boldsymbol{z} = (z_0, z_1, \dots, z_{N-1})$, i.e., we have $\boldsymbol{z} = S\boldsymbol{x}$. To find S, for $1 \le n \le N-2$ 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. While Toeplitz matrices here show up in the context of filters, they will also show up later in the context of wavelets.

Equation (3.1) leads us to the more general expression

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

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 $t_0 = 1/2$. By following the same argument as above, the following is clear:

Proposition 3.2. 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.

Example 3.3. Let us apply Proposition 3.2 to the operation defined by formula (3.1):

1. for $k = 0$ Equation 3.4 gives $s_0 = t_0 = 1/2$.
2. For $k = 1$ Equation 3.4 gives $s_1 = t_1 = 1/4$.
3. 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). ♣

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):

Example 3.4. 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

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

♣

Exercises for Section 3.1

1. 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 $N = 8$.

2. 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.

3. Assume that S is a circulant Toeplitz matrix so that only

$$S_{0,0}, \ldots, S_{0,F} \text{ and } S_{0,N-E}, \ldots, S_{0,N-1}$$

are nonzero on the first row, where E, F are given numbers. When implementing this filter on a computer we need to make sure that the vector indices lie in $[0, N-1]$. Show that $z_n = (S\boldsymbol{x})_n$ can be split into the following different formulas, depending on n, to achieve this:

a. $0 \leq n < E$:

$$z_n = \sum_{k=0}^{n-1} S_{0,N+k-n}x_k + \sum_{k=n}^{F+n} S_{0,k-n}x_k + \sum_{k=N-1-E+n}^{N-1} S_{0,k-n}x_k. \quad (3.5)$$

b. $E \leq n < N - F$:

$$z_n = \sum_{k=n-E}^{n+F} S_{0,k-n}x_k. \quad (3.6)$$

c. $N - F \leq n < N$:

$$z_n = \sum_{k=0}^{n-(N-F)} S_{0,k-n} x_k + \sum_{k=n-E}^{n-1} S_{0,N+k-n} x_k + \sum_{k=n}^{N-1} S_{0,k-n} x_k. \tag{3.7}$$

From these three formulas we can write down a full implementation of the filter. This implementation is often more useful than writing down the entire matrix S, since we save computation when many of the matrix entries are zero.

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 2.27, 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 3.5 (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 it maps any Fourier vector in $\mathbb{R}^N$ to a multiple of itself. In other words, for any integer n in the range $0 \leq n \leq N-1$ there exists a value $\lambda_{S,n}$ so that

$$S(\phi_n) = \lambda_{S,n} \phi_n, \tag{3.8}$$

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.

We will identify the linear transformation S with its matrix relative to the standard basis. 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

$$S = F_N^H D F_N \tag{3.9}$$

whenever S is a digital filter, where D has the frequency response (i.e. the eigenvalues) on the diagonal[1]. 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

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

[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]).

Since $D_1D_2 = D_2D_1$ for any diagonal matrices, we get the following corollary:

Corollary 3.6. 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, $S_1S_2 = S_2S_1$.

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*.

The formal definition of digital filters resembles that of analog filters, the difference being that the Fourier basis is now discrete. From this one may think that one can construct digital filters from analog filters. The following result clarifies this:

Theorem 3.7. Let s be an analog filter with frequency response $\lambda_s(f)$, and assume that $f_1 \in V_{M,T}$ and that $s(f_1) = f_2$. Let

$$\begin{aligned}\boldsymbol{x} &= (f_1(0 \cdot T/N), f_1(1 \cdot T/N), \ldots, f_1((N-1)T/N))\\ \boldsymbol{z} &= (f_2(0 \cdot T/N), f_2(1 \cdot T/N), \ldots, f_2((N-1)T/N))\end{aligned}$$

be vectors of $N = 2M + 1$ uniform samples from the input and the output. 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 frequency response $\lambda_{S,n} = \lambda_s(n/T)$.

Proof. Assume that $s(f_1) = f_2$. When $N = 2M + 1$ we have that $f_1 \in V_{M,T}$ is uniquely determined from the N sample values in $\boldsymbol{x}$. This means that $s(f_1)$ also is uniquely determined from $\boldsymbol{x}$, so that the sample values in $\boldsymbol{z}$ also are uniquely determined from $\boldsymbol{x}$. It follows that the operation $S : \boldsymbol{x} \to \boldsymbol{z}$ is 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 $\lambda_{S,n} = \lambda_s(n/T)$. $\square$

It is interesting that the digital frequency response above is obtained by sampling the analog frequency response. In this way we also see that it is easy to realize any digital filter as the restriction of an analog filter: any analog filter s will do where the frequency response has the values $\lambda_{S,n}$ at the points n/T. In the theorem it is essential that $f_1 \in V_{M,T}$. There are many functions with the same samples, but where the samples of the output from the analog filter are different. When we restrict to $V_{M,T}$, however, the output samples are always determined from the input samples.

The previous theorem explains how digital filters can occur in practice. In the real world, a signal is modeled as a continuous signal $f_1(t)$, and an operation

$$\begin{tikzcd} f_1 \arrow[r, "s"] \arrow[d, "s"] & f_2 \\ \boldsymbol{x} \arrow[r, "S"] & \boldsymbol{z} \arrow[u, "r"'] \end{tikzcd}$$

Figure 3.1: The connections between sampling, analog and digital filters provided by Theorem 3.7. The vertical arrows here represent sampling (s) and reconstruction (r).

on signals as an analog filter s. We can't compute the entire output $s(f_1)$ of the analog filter, but we can obtain the vector $\boldsymbol{x}$ by sampling f_1, and then compute the output of a corresponding digital filter S. The output f_2 can finally be approximated by finding a function in $V_{M,T}$ interpolating this output, using the interpolation formula from the sampling theorem. The role of a digital filter as an approximation to an analog filter is illustrated in Figure 3.1. This clearly also shows the algorithm for approximating the analog filter in terms of sampling, a digital filter, and reconstruction/interpolation. If the input to s is in $V_{M,T}$, then clearly there is no error in using the digital filter instead of the analog filter. If there is a bound on the highest frequency in f_1, then f_1 lies in some $V_{M,T}$ (as long as M is chosen big enough), so that the output can be computed exactly. What happens when there is no bound on the highest frequency? We know that $s((f_1)_N) = (f_2)_N$. Since $(f_1)_N$ is a good approximation to f_1, the samples $\boldsymbol{x}$ of f are close to the samples of f_N. By continuity of the digital filter, the output $\boldsymbol{z}$ of the filter will also be close to the samples of $(f_2)_N$ (since $\boldsymbol{z} = S\boldsymbol{x}$ and $(f_2)_N = s((f_1)_N)$). The interpolation formula from the sampling theorem then gives a good approximation to $(f_2)_N$, which is again a good approximation to f_2. By construction then, the digital filter is a better approximation when N is high. It is also a better approximation in cases where the Fourier series $(f_1)_N$ is a very good approximation. We can summarize this as follows:

Idea 3.8 (Approximating an analog filter)**.** An analog filter s can be approximated by applying the digital filter S with frequency response $\lambda_{S,n} = \lambda_s(n/T)$, to samples of the input. When we restrict s to $V_{M,T}$, it is the same as the digital filter. When we increase the size N of the filter, the approximation becomes better. If there is a bound on the highest frequency in the sound, there exists an N so that the output can be computed exactly with the digital filter for that size.

There are several other equivalent characterizations of a digital filter as well. The next characterization helps us prove that the operations we started this chapter with actually are filters.

Theorem 3.9. A linear transformation S is a digital filter if and only if it is a circulant Toeplitz matrix.

Proof. That S is a filter is equivalent to the fact that $S = (F_N)^H DF_N$ for some diagonal matrix D. We observe that the entry at position (k, l) in S is given by

$$S_{k,l} = \frac{1}{N}\sum_{n=0}^{N-1} e^{2\pi ikn/N}\lambda_{S,n}e^{-2\pi inl/N} = \frac{1}{N}\sum_{n=0}^{N-1} e^{2\pi i(k-l)n/N}\lambda_{S,n}.$$

Another entry on the same diagonal (shifted s rows and s columns) is

$$\begin{aligned} S_{(k+s) \bmod N,(l+s) \bmod N} &= \frac{1}{N}\sum_{n=0}^{N-1} e^{2\pi i((k+s) \bmod N-(l+s) \bmod N)n/N}\lambda_{S,n} \\ &= \frac{1}{N}\sum_{n=0}^{N-1} e^{2\pi i(k-l)n/N}\lambda_{S,n} = S_{k,l}, \end{aligned}$$

which proves that S is a circulant Toeplitz matrix. Conversely, if S is a circulant Toeplitz matrix, it is an exercise to prove that $e^{2\pi ikn/N}$ are eigenvectors, so that S is a digital filter. □

In particular, 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, so that we do not need to form the characteristic polynomial and find its roots:

Theorem 3.10. 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

$$\boldsymbol{\lambda}_S = \sqrt{N}F_N\boldsymbol{s}. \tag{3.10}$$

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

$$\boldsymbol{s} = \frac{1}{\sqrt{N}}(F_N)^H\boldsymbol{\lambda}_S. \tag{3.11}$$

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

$$F_N\boldsymbol{s} = F_N S\begin{pmatrix}1\\0\\\vdots\\0\end{pmatrix} = F_N F_N^H DF_N\begin{pmatrix}1\\0\\\vdots\\0\end{pmatrix} = DF_N\begin{pmatrix}1\\0\\\vdots\\0\end{pmatrix} = \frac{1}{\sqrt{N}}D\begin{pmatrix}1\\\vdots\\1\end{pmatrix},$$

where we have used the fact that the first column in F_N has all entries equal to $1/\sqrt{N}$. But the diagonal matrix D has all the eigenvalues of S on its diagonal, and hence the last expression is the vector of eigenvalues $\boldsymbol{\lambda}_S$, which proves (3.10). Equation (3.11) follows directly by applying the inverse DFT to (3.10). $\square$

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 (3.10) states that the frequency response can be written as

$$\lambda_{S,n} = \sum_{k=0}^{N-1} s_k e^{-2\pi i n k/N}, \quad \text{for } n = 0, 1, \ldots, N-1, \tag{3.12}$$

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

Example 3.11. 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. ♣

Example 3.12. 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

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

♣

Equations (3.9), (3.10), and (3.11) are important relations between the matrix- and frequency representations of a filter. We see that the DFT is a crucial ingredient in these relations. A consequence is that, once you recognize a matrix as circulant Toeplitz, you do not need to make the tedious calculation of eigenvectors and eigenvalues which you are used to. Let us illustrate this with an example.

Example 3.13. Let us compute the eigenvalues and eigenvectors of the 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 learnt 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 $e^{2\pi ink/N} = e^{\pi ink} = (-1)^{nk}$. 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 Matlab. Note that Matlab 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). Matlab may here for instance switch the signs of the eigenvectors. We have no control over what Matlab actually chooses to do, since it uses some underlying numerical algorithm for computing eigenvectors which we can't influence. ♣

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 3.10 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).

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, as the following example shows.

Example 3.14. 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

$$\begin{aligned}\sqrt{N}\phi_5 &= \left(e^{2\pi i5\cdot 0/N}, e^{2\pi i5\cdot 1/N}, \ldots, e^{2\pi i5\cdot (N-1)/N}\right)\\ \sqrt{N}\phi_{N-5} &= \left(e^{-2\pi i5\cdot 0/N}, e^{-2\pi i5\cdot 1/N}, \ldots, e^{-2\pi i5\cdot (N-1)/N}\right),\end{aligned}$$

Since $e^{2\pi i5k/N} + e^{-2\pi i5k/N} = 2\cos(2\pi 5k/N)$, 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

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

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

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

♣

Exercises for Section 3.2

1. Let S be a circulant Toeplitz matrix. Show that the Fourier vectors $e^{2\pi i k n/N}$ are eigenvectors of S. This completes the proof of the converse part of Theorem 3.9.

2. Let S be a digital filter. Show that S is symmetric if and only if the frequency response satisfies $s_k = s_{N-k}$ for all k.

3. 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 Matlab 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.

4. 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.?

5. 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 3.15. The function $\lambda_S(\omega)$ defined on $[0, 2\pi)$ by

$$\lambda_S(\omega) = \sum_k t_k e^{-ik\omega}, \tag{3.13}$$

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 reponse lies on the curve λ_S.

Proof. For any N we have that

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

where we have used Equation (3.4). □

Both $\lambda_S(\omega)$ and $\lambda_{S,n}$ will be referred to as frequency responses in the following. To distinguish the two, while $\lambda_{S,n}$ is called the vector frequency response of S, $\lambda_S(\omega))$ is called the *continuous frequency response* of S. ω is called *angular frequency.*

The difference in the definition of the continuous- and the vector frequency response lies in that one uses the filter coefficients t_k, while the other uses the impulse response s_k. While these contain the same values, they are ordered differently. Had we used the impulse response to define the continuous frequency response, we would have needed to compute $\sum_{k=0}^{N-1} s_k e^{-\pi i\omega}$, which does 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 2.26 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 3.16. 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

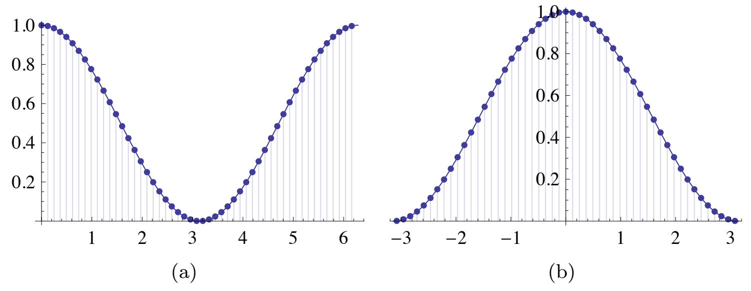


Figure 3.2: The (absolute value of the) frequency response of the moving average filter of Formula (3.1) from the beginning of this chapter.

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 3.17. 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 3.18. 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 $\lambda_{S,n} = \lambda_S(2\pi n/N) = \lambda_S(2\pi f T/N) = \lambda_S(2\pi f f_s)$.

Example 3.19. In Example 3.12 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 3.12. Figure 3.2 shows plots of this frequency response, plotted on the intervals $[0, 2\pi)$ and $[-\pi, \pi)$. Both the continuous frequency response and the vector frequency response for $N = 51$ are shown. Figure (b) shows clearly how the high frequencies are softened by the filter. ♣

Since the frequency response is essentially a DFT, it inherits several properties from Theorem 2.21. We will mostly use the continuous frequency response to express these properties.

Theorem 3.20. We have that

1. The continuous frequency response satisfies $\lambda_S(-\omega) = \overline{\lambda_S(\omega)}$.
2. If S is a digital filter, S^T is also a digital filter. Morever, if the frequency response of S is $\lambda_S(\omega)$, then the frequency response of S^T is $\overline{\lambda_S(\omega)}$.
3. 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.
4. 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 $\lambda_{S^{-1},n} = 1/\lambda_{S,n}$.
5. 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 2.21. Transposing a matrix corresponds to reversing the first colum of the matrix and thus also the filter coefficients. Due to this Property 2. also follows from Theorem 2.21. If $S = (F_N)^H DF_N$, and all $\lambda_{S,n} \neq 0$, we have that $S^{-1} = (F_N)^H D^{-1}F_N$, where D^{-1} is a digonal 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:

$$S_1S_2 = (F_N)^H D_1F_N(F_N)^H D_2F_N = (F_N)^H D_1D_2F_N.$$

The frequency response of S_1S_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 3.20 applies also for the vector frequency response.

Example 3.21. 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 3.20 to compute the filter coefficients and the matrix of the filter $S = S_1S_2$. We first notice that, since both frequency responses are real, all S_1, S_2, and $S = S_1S_2$ are symmetric. We rewrite the frequency responses as

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

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

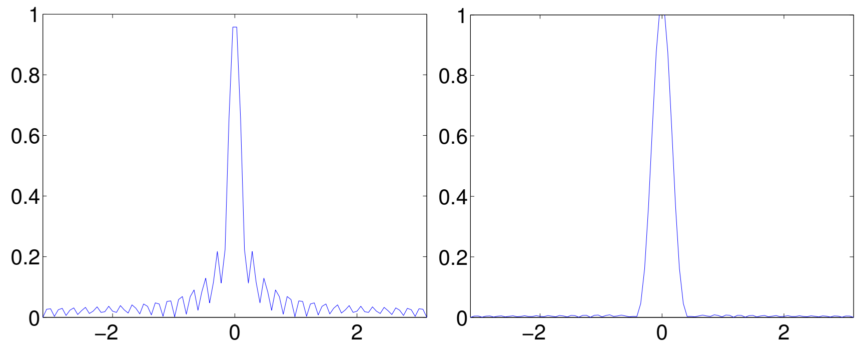
In this section we will take a look at a very important, and perhaps surprising, application of the continuous frequency response. Let us return to the computations from Example 2.27. There we saw that, when we restricted to a block of the signal, this affected the frequency representation. If we substitute with the angular frequencies $\omega = 2\pi n/N$ and $\omega_0 = 2\pi n_0/M$ in Equation (2.10), 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 restrcited 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 constribution 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 3.22. Let $(x_0, \ldots, x_M)$ 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 $(w_0x_0, \ldots, w_{N-1}x_{N-1})$ of length N (i.e. where the samples are attenuated by the window samples, and where samples have been discarded) has a frequency representation as close to $\boldsymbol{x}$ as possible. The vector $\boldsymbol{w}$ is called a window of length N, and the new sound is called the windowed signal.



(a) Rectangular window (b) Hamming window

Figure 3.3: The frequency responses of the windows we have considered for restricting to a block of the 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

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

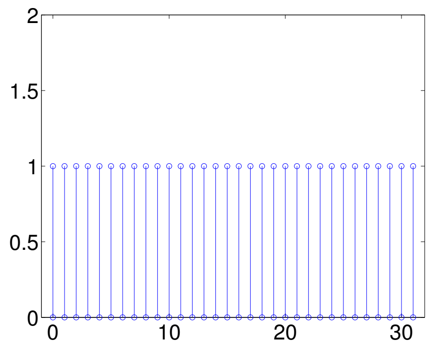
This is the frequency response of $\frac{1}{N}\boldsymbol{w}$. In order to limit the pollution from other frequencies, we thus need to construct a window with a frequency response with smaller values than that of the rectangular window away from 0. Let us summarize our findings as follows:

Observation 3.23. 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.

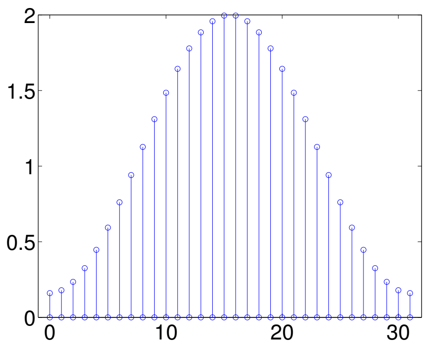
We will not go into techniques for how such frequency responses can be constrcuted, but only consider one example different from the rectangular window. We define the Hamming window by

$$w_n = 2(0.54 - 0.46\cos(2\pi n/(N-1))). \tag{3.14}$$

The frequency responses of the rectangular window and the Hamming window are compared in Figure 3.3 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 3.4. It is seen that the Hamming filter attenuates more and more as we get close to the boundaries.



(a) Rectangular window



(b) Hamming window

Figure 3.4: The window coefficients we have considered for restricting to a block of the signal.

Exercises for Section 3.3

1. 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 3.1.1. Compute and plot (the magnitude of) $\lambda_S(\omega)$.

2. 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.

3. 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 Assembling the filter matrix and compact notation

Let us return to how we first defined a filter in Equation (3.3):

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

As mentioned, the range of k may not be specified. In some applications in signal processing there may in fact be infinitely many nonzero t_k. However, when $\boldsymbol{x}$ is assumed to have period N, we may as well assume that the k's range over an interval of length N (else many of the t_k's can be added together to simplify the formula). Also, any such interval can be chosen. It is common to choose the interval so that it is centered around 0 as much as possible. For this, we can choose for instance $[\lfloor N/2 \rfloor - N + 1, \lfloor N/2 \rfloor]$. With this choice we can write Equation (3.3) as

$$z_n = \sum_{k=\lfloor N/2 \rfloor - N+1}^{\lfloor N/2 \rfloor} t_k x_{n-k}. \tag{3.15}$$

The index range in this sum is typically even smaller, since often much less than N of the t_k are nonzero (For Equation (3.1), there were only three nonzero t_k). In such cases one often uses a more compact notation for the filter:

Definition 3.24 (Compact notation for filters)**.** Let k_{min}, k_{max} be the smallest and biggest index of a filter coefficient in Equation (3.15) so that $t_k \neq 0$ (if no such values exist, let $k_{\text{min}} = k_{\text{max}} = 0$), i.e.

$$z_n = \sum_{k=k_{min}}^{k_{max}} t_k x_{n-k}. \tag{3.16}$$

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 3.25. 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 3.4, we would write $S = \{1, \underline{2}, 3\}$. ♣

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):

Definition 3.26 (Convolution of vectors)**.** By the *convolution* of two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ we mean the vector $\boldsymbol{x} * \boldsymbol{y}$ defined by

$$(\boldsymbol{x} * \boldsymbol{y})_n = \sum_k x_k y_{n-k}. \tag{3.17}$$

If both $\boldsymbol{x}$ and $\boldsymbol{y}$ have infinitely many nonzero entries, the sum is an infinite one, and may diverge.

The case where both vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ have a finite number of nonzero elements deserves extra attention. Assume that $x_0, \ldots, x_{N-1}$ and $y_0, \ldots, y_{M-1}$ are the only nonzero elements. If $\boldsymbol{z} = \boldsymbol{x} * \boldsymbol{y}$, it is clear from Equation (3.17) that only the elements $z_0, \ldots, z_{M+N-2}$ can be nonzero. The convolution operation is therefore fully represented by the finite-dimensional operation from $\mathbb{R}^N \times \mathbb{R}^M \to \mathbb{R}^{M+N-1}$ defined by

$$(x_0, \ldots, x_{N-1}) \times (y_0, \ldots, y_{M-1}) \to (z_0, \ldots, z_{M+N-2}). \tag{3.18}$$

Matlab has the built-in function `conv` for performing this operation. The `conv` function thus considers the convolution in terms of the (finite) nonzero parts of the vectors, without keeping track of the start and end indices. Exercise 7 explains how one may keep track of these indices. Put differently, Matlab's `conv`-operation pads our original values with zeros in both directions, instead of considering a periodic input vector. Due to its simplicity it is used much in practice, but it is not exactly the same as applying a digital filter. The following result states that the difference between convolution and filtering is only at the start and end of the vector.

Theorem 3.27. Let $S = \{t_{-E}, \ldots, \underline{t_0}, \ldots, t_F\}$ where $E, F \geq 0$, and let $\boldsymbol{x} = (x_0, \ldots, x_{N-1})$. We have that $\boldsymbol{t}*\boldsymbol{x}$ equals $S\boldsymbol{x}$ at the indices from F to $N-E-1$.

Proof. We have that $(S\boldsymbol{x})_n = t_{-E}x_{n+E} + \ldots + t_F x_{n-F}$. Here the indices of $\boldsymbol{x}$ lie between 0 and $N-1$ if and only if $n + E \leq N-1$ and $n - F \geq 0$, which happen if $F \leq n \leq N-E-1$. We therefore do not access the added zeros outside $[0, N-1]$, so that the result is equal. □

We also have a very important connection between convolution and polynomials

Proposition 3.28. 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.

Clearly also, this polynomial is the frequency response, when we insert $e^{i\omega}$ for the variable. 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. Exercise 3 investigates this further. For large N we risk running into out of memory situations if we had to form the entire matrix.

Exercises for Section 3.4

1. 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?

2. 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 1?

3. 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.

4. 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 filter coefficients of this matrix, and use the compact notation $\{t_{k_{min}}, \ldots, t_{-1}, \underline{t_0}, t_1, \ldots, t_{k_{max}}\}$. Compute and plot (the magnitude) of $\lambda_S(\omega)$.

5. 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?

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

7. In this exercise we will find out how to keep to track of the length and the start and end indices when we convolve two sequences.

a. Let $\boldsymbol{x}$ be zero outside $x_a, \ldots, x_{a+N-1}$, and $\boldsymbol{y}$ be zero outside $y_b, \ldots, y_{b+M-1}$. Show that $\boldsymbol{z} = \boldsymbol{x} * \boldsymbol{y}$ is zero outside $z_{a+b}, \ldots, z_{a+b+M+N-2}$. Explain why this means that $l(\boldsymbol{x} * \boldsymbol{y}) = l(\boldsymbol{x}) + l(\boldsymbol{y})$ for general vectors.

b. Find expressions for the start- and end indices k_{min}, k_{max} for $\boldsymbol{x} * \boldsymbol{y}$, in terms of those of $\boldsymbol{x}$ and $\boldsymbol{y}$.

8. Implement a Matlab function

```
function z=convimpl(x,y)
```

which from input vectors of dimension N and M returns an output vector of dimension $N + M - 1$, as dictated by Equation (3.18). Test your function together with the built-in Matlab `conv`-function to verify that the give the same result.

9. Show that if $S = \{\underline{t_0}, \ldots, t_F\}$ and $\boldsymbol{x} \in \mathbb{R}^N$, then $S \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{0}_F \end{pmatrix} = \boldsymbol{t} * \boldsymbol{x}$. Thus if we add zeros in a vector, filtering and convolution are the same.

3.5 Some examples of filters

We have now established the basic theory of filters, and it is time to study some specific examples.

Example 3.29 (Time delay filters)**.** The simplest possible type of Toeplitz matrix is one where there is only one nonzero diagonal. Let us define the Toeplitz matrix E_d as the one which has first column equal to $\boldsymbol{e}_d$. In the notation above, and when $d > 0$, this filter can also be written as $S = \{\underline{0}, \ldots, 1\}$ where the 1 occurs at position d. We observe that

$$(E_d\boldsymbol{x})_n = \sum_{k=0}^{N-1} (E_d)_{n,k}\, x_k = \sum_{k=0}^{N-1} (E_d)_{(n-k) \bmod N, 0}\, x_k = x_{n-d},$$

since only when $(n - k) \bmod N = d$ do we have a contribution in the sum. It is thus clear that multiplication with E_d delays a vector by d samples. For this reason E_d is also called a *time delay filter*. The frequency response of the time delay filter is clearly the function $\lambda_S(\omega) = e^{-id\omega}$, which has magnitude 1. This filter therefore does not change the magnitude of the different frequencies. ♣

Example 3.30 (Adding echo)**.** An echo is a copy of the sound that is delayed and softer than the original sound. The sample that comes t seconds before sample i has index $i - tf_s$ where f_s is the sampling rate. This also makes sense even if t is not an integer so we can use this to produce delays that are less than one second. The one complication with this is that the number tf_s may not be an integer. We can get round this by rounding it to the nearest integer. This corresponds to adjusting the echo slightly. The following holds:

Fact 3.31. Let $(\boldsymbol{x}, s)$ be a digital sound. Then the sound $\boldsymbol{y}$ with samples given by

```
y=[x zeros(1,d)];
y((d+1):N)=x((d+1):N)-c*x(1:(N-d));
```

will include an echo of the original sound. Here `d=round(ms)` is the integer closest to tf_s, and c is a constant which is usually smaller than 1.

This is an example of a filtering operation where each output element is constructed from two input elements. As in the case of noise it is important to dampen the part that is added to the original sound, otherwise the echo will be too loud. Note also that the formula that creates the echo is not used at the beginning of the signal, since it is not audible until after d samples. Also, the echo is unaudible if d is too small. 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 $\lambda_S(\omega) = 1 + ce^{-id\omega}$. This frequency response is not real, which means that the filter is not symmetric. In Figure 3.5 we have plotted the magnitude of this frequency response with $c = 0.1$ and $d = 10$. We see that the response varies between 0.9 and 1.1, so that the deviation from 1 is controlled by the damping factor c. Also, we see that the oscillation in the frequency response, as visible in the plot, is controlled by the delay d. ♣

Let us now take a look at some filters which adjust the bass and treble in sound. The fact that the filters are useful for these purposes will be clear when we plot the frequency response.

Example 3.32 (Reducing the treble with moving average filters)**.** The treble in a sound is generated by the fast oscillations (high frequencies) in the signal. If we want to reduce the treble we have to adjust the sample values in a way that reduces those fast oscillations. A general way of reducing variations in a sequence of numbers is to replace one number by the average of itself and its neighbours, and this is easily done with a digital sound signal. If we let the new sound signal be $\boldsymbol{z} = (z_i)_{i=0}^{N-1}$ we can compute it as

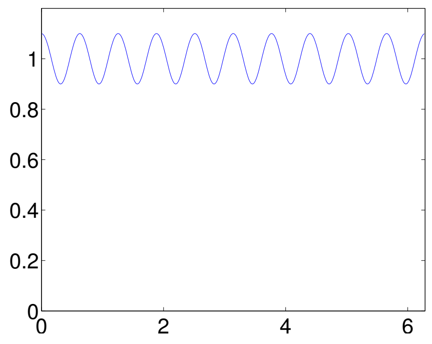


Figure 3.5: The frequency response of a filter which adds an echo with damping factor $c = 0.1$ and delay $d = 10$.

```
z(1)=(x(N)+x(1)+x(2))/3;
for t=2:(N-1)
  z(t)=(x(t-1)+x(t)+x(t+1))/3;
end
z(N)=(x(N-1)+x(N)+x(1))/3;
```

In Example 3.30 we did not take into account that the signal is assumed periodic. Above this has been taken into account through the addition of the first and last lines (which correspond to the circulating part of the matrix). This filter is also called a *moving average filter*, and it can be written in compact form as

$$S = \left\{ \frac{1}{3}, \underline{\frac{1}{3}}, \frac{1}{3} \right\}.$$

If we set $N = 4$, the corresponding circulant Toeplitz matrix for the filter is

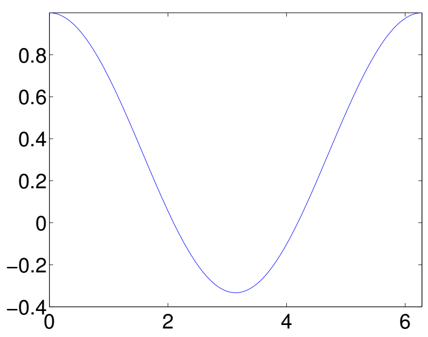
$$S = \frac{1}{3} \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

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. In Example 2.18 we computed that the DFT of the vector $\{1, \cdots, \underline{1}, \cdots, 1\}$ was

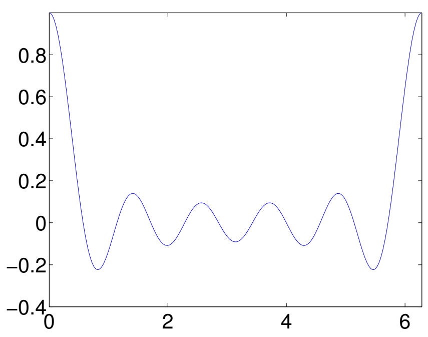
$$\boldsymbol{x} = \frac{1}{\sqrt{N}} \frac{\sin(\pi n(2L + 1)/N)}{\sin(\pi n/N)}.$$

Due to this the frequency response of S is

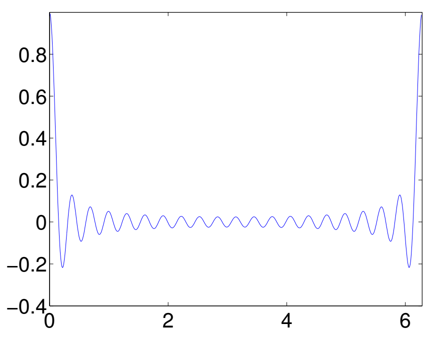
$$\lambda_{S,n} = \frac{1}{2L + 1} \frac{\sin(\pi n(2L + 1)/N)}{\sin(\pi n/N)},$$



(a) L=1



(b) L=5



(c) L=20

Figure 3.6: The frequency response of moving average filters of different length.

and thus

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

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 it is clear that this peak gets narrower and narrower as L increases, i.e. we use more and more samples in the averaging process. This appeals to our intuition that this kind of filters smooth the sound by keeping only lower frequencies. In Figure 3.6 we have plotted the frequency response for moving average filters with $L = 1$, $L = 5$, and $L = 20$. We see, unfortunately, that the frequency response is far from a filter which keeps some frequencies unaltered, while annihilating others (this is a desirable property which is refered to as being a *bandpass filter*): 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. ♣

In the previous example we mentioned filters which favour certain frequencies of interest, while annihilating the others. This is a desirable property for filters, so let us give names to such filters:

Definition 3.33. A filter S is called

1. a *lowpass 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 annhilates high frequencies),
2. a *highpass 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 annhilates low frequencies),
3. 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 lowpass and highpass filters, it may be that the frequency responses are still quite far from what is commonly refered to as *ideal lowpass or highpass filters*, where the frequency response only assumes the values 0 and 1 near 0 and π. The next example considers an ideal lowpass filter.

Example 3.34 (Ideal lowpass filters)**.** By definition, the ideal lowpass 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 2.27 we implemented the ideal lowpass 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 2.18 we computed the DFT of this vector, and it followed from Theorem 2.21 that the IDFT of this vector equals its DFT. This means that we can find the filter coefficients by using Equation (3.11): Since the IDFT was $\frac{1}{\sqrt{N}} \frac{\sin(\pi k(2L+1)/N)}{\sin(\pi k/N)}$, the filter coefficients are

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

This means that the filter coefficients lie as N points uniformly spaced on the curve $\frac{1}{N} \frac{\sin(\omega(2L+1)/2)}{\sin(\omega/2)}$ between 0 and π. 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. ♣

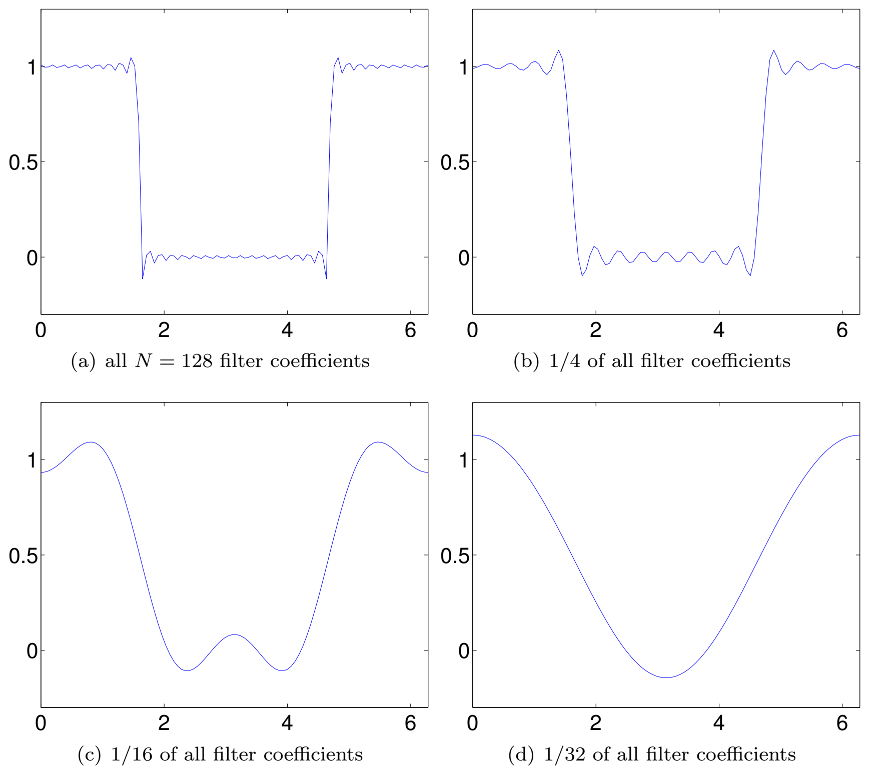
The two previous examples show an important duality between vectors which are 1 on some elements and 0 on others (also called window vectors), and the vector $\frac{1}{N}\frac{\sin(\pi k(2L+1)/N)}{\sin(\pi k/N)}$ (also called a sinc): filters of the one type correspond to frequency responses of the other 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 lowpass 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 3.35. In order to decrease the computational complexity for the ideal lowpass filter in Example 3.34, one can for instance include only the first filter coefficients, i.e. $\{\frac{1}{N}\frac{\sin(\pi k(2L+1)/N)}{\sin(\pi k/N)}\}_{k=-N_0}^{N_0}$, ignoring the last ones. Hopefully this gives us a filter where the frequency reponse is not that different from the ideal lowpass filter. In Figure 3.7 we show the corresponding frequency responses. In the figure we have set $N = 128$, $L = 32$, so that the filter removes all frequencies $\omega > \pi/2$. N_0 has been chosen so that the given percentage of all coefficients are included. Clearly the figure 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 filter. In particular, we see that the new frequency response oscillates wildly near the discontinuity of the ideal lowpass filter. Such oscillations are called *Gibbs oscillations.* ♣

Example 3.36 (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 lowpass filter, there were some undesired effects in the frequency response of the resulting filter. Are there other and better approximations to the ideal lowpass 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



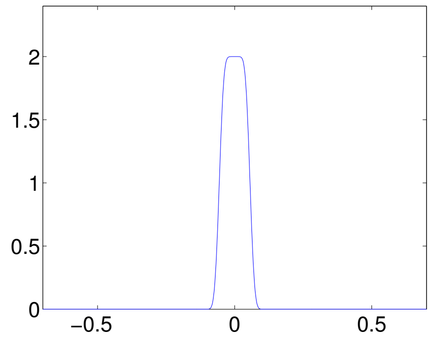
(a) all $N = 128$ filter coefficients

(b) 1/4 of all filter coefficients

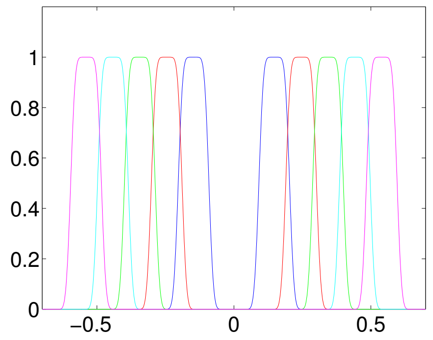
(c) 1/16 of all filter coefficients

(d) 1/32 of all filter coefficients

Figure 3.7: The frequency response which results by omitting the last filter coefficients for the ideal lowpass filter.



(a) Frequency response of the prototype filter used in the MP3 standard



(b) Frequency responses of 5 of the other filters used in the MP3 standard. They are shifted copies of that of the prototype

Figure 3.8: Frequency responses of some filters used in the MP3 standard.

have been constructed. These filters are more advanced than the ones we have seen upto 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 Figure 3.8(a), the "prototype filter" which is used in the MP3 standard is shown. We see that this is very close to an ideal lowpass filter. Moverover, 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 2.21, 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 Figure 3.8(b). 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 lowpass 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. ♣

Example 3.37 (Reducing the treble II)**.** When reducing the treble it is reasonable to let the middle sample x_i count more than the neighbours in the average, so an alternative is to compute the average by instead writing

```
  z(t)=(x(t-1)+2*x(t)+x(t+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 3.32 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 3.20 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 $0 \le r \le k$). 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 we take a k'th power with k large, λ_{S^k} is more square-like near 0, i.e. it becomes more and more like a bandpass filter near 0. In Figure 3.9 we have plotted the magnitude of the frequence response when $k = 5$, and when $k = 30$. This behaviour near 0 is not so easy to see from the figure. Note that we have zoomed in on the frequency response to the area where it actually decreases to 0.

If we pick coefficients from row 4 of Pascals triangle instead, we would write

```
for t=3:(N-2)
  y(t)=(x(t-2)+4*x(t-1)+6*x(t)+4*x(t+1)+x(t+2))/16;
end
```

Here we have dropped the first and last part, which have special expressions due to the circulant structure of the matrix. Picking coefficients from a row in Pascal's triangle works better the longer the filter is:

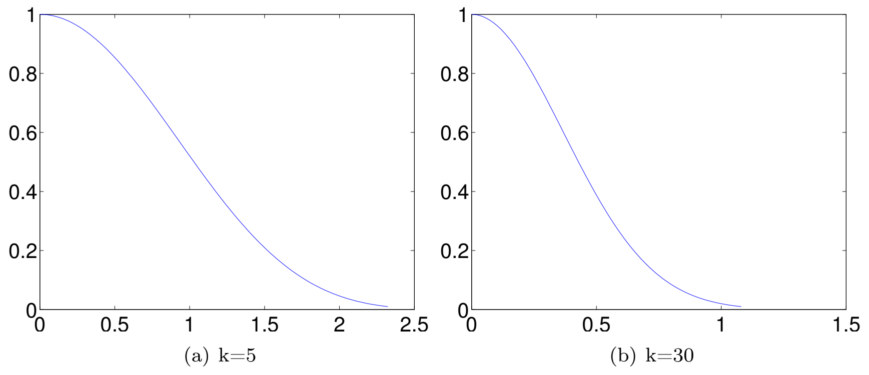


Figure 3.9: The frequency response of filters corresponding to a moving average filter convolved with itself k times.

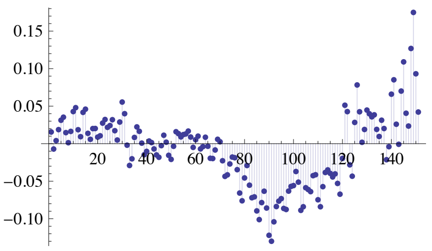
Observation 3.38. Let $\boldsymbol{x}$ be the samples of a digital sound, and let $\{c_i\}_{i=1}^{2k+1}$ be the numbers in row $2k$ of Pascal's triangle. Then the sound with samples $\boldsymbol{y}$ given by

```
y=zeros(length(x));
for t=(k+1):(N-k)
  for j=1:(2*k+1)
    y(t)=y(t)+c(j)*x(t+k+1-j))/2^k;
  end
end
```

has reduced treble compared with the sound given by the samples $\boldsymbol{x}$.

An example of the result of smoothing is shown in Figure 3.10. (a) shows a real sound sampled at CD-quality (44 100 samples per second). (b) shows the result of applying the averaging process by using row 4 of Pascals triangle. We see that the oscillations have been reduced, and if we play the sound it has considerably less treble. In Exercise 4 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. ♣

Another common option in an audio system is reducing the bass. This corresponds to reducing the low frequencies in the sound, or equivalently, the slow variations in the sample values. It turns out that this can be accomplished by simply changing the sign of the coefficients used for reducing the treble. Let us explain why this is the case. Let S be a filter with filter coefficients s_k, and let us consider the filter T with filter coefficient $(-1)^k s_k$. The frequency response


(a) The original sound signal

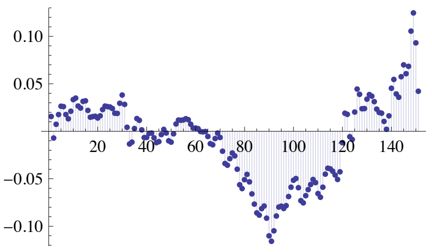

(b) The result of applying the filter from row 4 of Pascal's triangle

Figure 3.10: Reducing the treble.

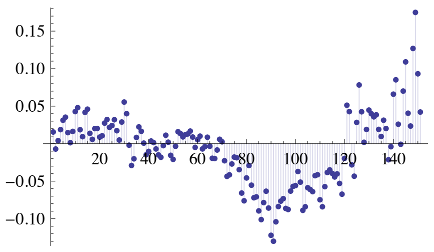
of T is

$$\begin{aligned}\lambda_T(\omega) &= \sum_k (-1)^k s_k e^{-i\omega k} = \sum_k (e^{-i\pi})^k s_k e^{-i\omega k} \\ &= \sum_k e^{-i\pi k} s_k e^{-i\omega k} = \sum_k s_k e^{-i(\omega+\pi)k} = \lambda_S(\omega+\pi).\end{aligned}$$

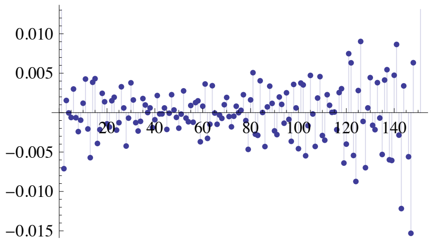
where we have set $-1 = e^{-i\pi}$ (note that this is nothing but Property 4. in Theorem 2.21, with $d = N/2$). Now, for a lowpass filter S, $\lambda_S(\omega)$ has large values when ω is close to 0 (the low frequencies), and values near 0 when ω is close to π (the high frequencies). For a highpass filter T, $\lambda_T(\omega)$ 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). When T is obtained by adding an alternating sign to the filter coefficicents of S, The relation $\lambda_T(\omega) = \lambda_S(\omega+\pi)$ thus says that T is a highpass filter when S is a lowpass filter, and vice versa:

Observation 3.39. Assume that T is obtained by adding an alternating sign to the filter coefficicents of S. If S is a lowpass filter, then T is a highpass filter. If S is a highpass filter, then T is a lowpass filter.

The following example elaborates further on this.



(a) The original sound



(b) The result of applying the filter deduced from row 4 in Pascals triangle

Figure 3.11: Reducing the bass.

Example 3.40 (Reducing the bass)**.** Consider the bass-reducing filter deduced from the fourth row in Pascals triangle:

```
y(t)=(x(t-2)-4*x(t-1)+6*x(t)-4*x(t+1)+x(t+2))/16;
```

An example of this filter applied to a sound is shown in Figure 3.11. The original sound is shown in (a) and the result in (b). We observe that the samples in (b) oscillate much more than the samples in (a). If we play the sound in (b), it is quite obvious that the bass has disappeared almost completely.

Observation 3.41. Let $\boldsymbol{x}$ be the samples of a digital sound, and let $\{c_i\}_{i=1}^{2k+1}$ be the numbers in row $2k$ of Pascal's triangle. Then the sound with samples $\boldsymbol{y}$ given by

```
y=zeros(length(x));
for t=(k+1):(N-k)
  for j=1:(2*k+1)
    y(t)=x(t)+(-1)^(k+1-j)*c(j)*x(t+j-k-1))/2^k;
  end
end
```

has reduced bass compared to the sound given by the samples $\boldsymbol{y}$.

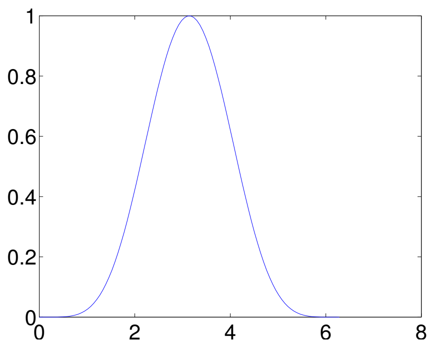


Figure 3.12: The frequency response of the bass reducing filter, which corresponds to row 5 of Pascal's triangle.

In Exercise 4 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 $k = 2$.

The frequency response we obtain from using row 5 of Pascal's triangle is shown in Figure 3.12. 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 3.37 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. ♣

Exercises for Section 3.5

1. 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 2.21, i.e. by using a property for the Discrete Fourier Transform.

2. In this exercise, we will experiment with adding echo to a signal.

a. Write a function

```
function playwithecho(c,d)
```

which plays the sound samples of `castanets.wav` with echo added for damping constant c and delay d as described in Example 3.30.

b. Generate the sound from Example 3.30, and verify that it is the same as the one you heard there.

c. Listen to the sound samples for different values of d and c. For which range of d is the echo distinguisible from the sound itself? How low can you choose c in order to still hear the echo?

3. 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?

4. In this exercise, we will experiment with increasing and reducing the treble and bass in a signal as in examples 3.37 and 3.40.

a. Write functions

```
function reducetreble(k)
function reducebass(k)
```

which reduces bass and treble in the ways described above for the sound from the file `castanets.wav`, and plays the result, when row number $2k$ in Pascal' triangle is used to construct the filters. Look into the Matlab function `conv` to help you to find the values in Pascal's triangle.

b. Generate the sounds you heard in examples 3.37 and 3.40, and verify that they are the same.

c. In your code, it will not be necessary to scale the values after reducing the treble, i.e. the values are already between -1 and 1. Explain why this is the case.

d. 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?

5. Consider again Example 3.34. 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 highpass filter).

6. In this exercise we will investigate how we can combine lowpass and highpass filters to produce other filters

a. Assume that S_1 and $S2$ are lowpass filters. What kind of filter is S_1S_2? What if both S_1 and S_2 are highpass filters?

b. Assume that one of S_1, S_2 is a highpass filter, and that the other is a lowpass filter. What kind of filter S_1S_2 in this case?

7. 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 lowpass filter, or a highpass 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 3.20 to compute the frequency response of S_1S_2 first.

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

8. 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 $T_2 = E_{d_2}TE_{d_1} = E_4TE_2$ 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.

9. (Exam UIO V2012)

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

```
[S,fs]=wavread('castanets.wav');
S=S(:,1);
N=length(S);
newS=zeros(N,1);
for k=2:(N-1)
  newS(k)=2*S(k+1) + 4*S(k) + 2*S(k-1);
end
newS(1)=2*S(2) + 4*S(1) + 2*S(N);
newS(N)=2*S(1) + 4*S(N) + 2*S(N-1);
newS=newS/max(abs(newS));
playerobj=audioplayer(newS,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 lowpass- or a highpass 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 lowpass- or a highpass filter?

3.6 Time-invariance of filters

As we have seen, filters can be characterized by their eigenvectors, and also as circulant Toeplitz matrices. To make the picture more complete, we will in this section state a third, important characterization of filters. This characterization is stated in terms of the following concept:

Definition 3.42 (Time-invariance)**.** A linear transformation from $\mathbb{R}^N$ to $\mathbb{R}^N$ is said to be time-invariant if, for any d, the output of the *delayed* input vector $\boldsymbol{z}$ defined by $z_n = x_{n-d}$ is the delayed output vector $\boldsymbol{w}$ defined by $w_n = y_{n-d}$.

We have the following result:

Theorem 3.43. A linear transformation S is a digital filter if and only if it is time-invariant.

Proof. Let $\boldsymbol{y} = S\boldsymbol{x}$, and $\boldsymbol{z}, \boldsymbol{w}$ as defined above. We have that

$$\begin{aligned} w_n = (S\boldsymbol{x})_{n-d} &= \sum_{k=0}^{N-1} S_{n-d,k} x_k \\ &= \sum_{k=0}^{N-1} S_{n,k+d} x_k = \sum_{k=0}^{N-1} S_{n,k} x_{k-d} \\ &= \sum_{k=0}^{N-1} S_{n,k} z_k = (S\boldsymbol{z})_n \end{aligned}$$

This proves that $S\boldsymbol{z} = \boldsymbol{w}$, so that S is time-invariant. □

By Example 3.29, delaying a vector with d elements corresponds to applying the filter E_d. Clearly, that S is time-invariant is the same as $SE_d = E_dS$ for any d. That all filters are time-invariant follows thus also immediately from the fact that all filters commute.

Due to Theorem 3.43, digital filters are also called *LTI filters* (LTI stands for Linear, Time-Invariant). By combining the definition of a digital filter with theorems 3.10 and 3.43 we get the following theorem which summarizes our findings.

Theorem 3.44 (Characterizations of digital filters)**.** The following are equivalent characterizations of a digital filter:

1. $S = (F_N)^H D F_N$ for a diagonal matrix D, i.e. the Fourier basis is a basis of eigenvectors for S.
2. S is a circulant Toeplitz matrix.
3. S is linear and time-invariant.

Exercises for Section 3.6

1. In Example 2.7 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.

3.7 More general filters

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

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

The important point here was that we had a limited number of nonzero t_k, and this enabled us to compute the filter on a computer. In practice, however, there exist many filtering operations with an infinite number of filter coefficients. The ideal lowpass filter from Example 3.34 is 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 3.45 (More general filters (1))**.** Let $\boldsymbol{x}$ be the input to a filter, and let T be a 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 coefficicents, 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 3.46 (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 coefficicents, 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 3.47. 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 3.48. Consider again the moving average filter S from Example 3.32:

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

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.

Exercises for Section 3.7

1. 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 Matlab to compute the output of the vector $\boldsymbol{x} = (1, \ldots, 10)$. In order to do this you should write down two 10×10-circulant Toeplitz matrices, and use these in Matlab.

3.8 Implementation of filters

As we saw in Example 3.48, 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.

3.8.1 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},...,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.

3.8.2 Factoring a filter into several filters

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 3.49. Let S be a filter with real coefficients. Assume that

$$\lambda_S(\omega) = Ke^{ik\omega}(e^{i\omega} - a_1)\ldots(e^{i\omega} - a_m)(e^{2i\omega} + b_1 e^{i\omega} + c_1)\ldots(e^{2i\omega} + b_n e^{i\omega} + c_n). \tag{3.19}$$

Then we can write $S = KE_k A_1 \ldots A_m B_1 \ldots B_n$, where $A_i = \{1, \underline{-a_i}\}$ and $B_i = \{1, b_i, \underline{c_i}\}$.

Note that in Equation 3.19 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), the number of of arithmetic operations is usually not reduced. However, consider Example 3.37, 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 3.37 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.

Exercises for Section 3.8

1. Write a function

```
function filterdftimpl(t,k0)
```

which takes the filter coefficients $\boldsymbol{t}$ and the value k_0 from Section 3.8.1 as input, computes the optimal M and implements the filter as described in that section.

2. 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.

3.9 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 [12]. 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 [6].

Chapter 4

Symmetric filters and the DCT

Recall that, in Figure 2.3, we listed an unknown operation, which should be a "DFT specialized to symmetric vectors". In this chapter we will construct this operation. It turns out that the final ingredient we need in order to do this is to specialize our analysis from the previous chapter to filters which are symmetric. We start with this in the first section.

4.1 Symmetric filters and symmetric vectors

We will start this section by seeing how we can find a better approximation to an analog filter s than the one we defined in Section 3.2. Assume that we apply s to a general function f_1, and denote as before its symmetric extension by $\breve{f}_1$. Let also $f_2 = s(f_1)$ be the output of the filter, which we would like to obtain a better approximation to. We start with the following observation.

Observation 4.1. Due to continuity of s, and since $(\breve{f}_1)_N$ is a better approximation to $\breve{f}_1$, compared to what $(f_1)_N$ is to f_1, $s(\breve{f}_1)_N)$ is a better approximation to $s(\breve{f}_1)$, compared to what $s((f_1)_N)$ is to $s(f_1) = f_2$.

Since $s(\breve{f}_1)$ agrees with $s(f_1)$ except near the boundaries, $s((\breve{f}_1)_N)$ is a better approximation to f_2 than what $s((f_1)_N)$ 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_1, $\breve{\boldsymbol{x}}$ the samples of $\breve{f}_1$. Turning around the fact that $(\breve{f}_1)_N$ is a better approximation to $\breve{f}_1$, compared to what $(f_1)_N$ is to f_1, the following is clear.

Observation 4.2. The samples $\breve{\boldsymbol{x}}$ are a better approximation to the samples of $(\breve{f}_1)_N$, than the samples $\boldsymbol{x}$ are to the samples of $(f_1)_N$.

$$\begin{tikzcd}
f_1 \arrow[r, "s"] \arrow[d] & f_2 \arrow[d] \\
\breve{f}_1 \arrow[d, "s"] & \breve{f}_2 \arrow[d, shift left, "s"] \\
\breve{\boldsymbol{x}} \arrow[d] & \breve{\boldsymbol{z}} \arrow[u, shift left, "r"] \arrow[d] \\
(\breve{\boldsymbol{x}}_0, \breve{\boldsymbol{x}}_1, \ldots, \breve{\boldsymbol{x}}_{N-1}) \arrow[r, "S_r"] & (\breve{\boldsymbol{z}}_0, \breve{\boldsymbol{z}}_1, \ldots, \breve{\boldsymbol{z}}_{N-1})
\end{tikzcd}$$

Figure 4.1: The connections between the new mapping S_r, sampling, and reconstruction.

Now, let $\boldsymbol{z} = S\boldsymbol{x}$, and $\breve{\boldsymbol{z}} = S\breve{\boldsymbol{x}}$. The following is also clear.

Observation 4.3. Due to continuity of the digital filter S, it follows from the preceding observation that $\breve{\boldsymbol{z}}$ is a better approximation to $S(\text{samples of } (\breve{f}_1)_N) = \text{samples of } s((\breve{f}_1)_N)$, than $\boldsymbol{z}$ is to $S(\text{samples of } (f_1)_N) = \text{samples of } s((f_1)_N)$.

Since we previously noted that $s((\breve{f}_1)_N)$ is a better approximation to the output f_2, we conclude that $\breve{\boldsymbol{z}}$ is a better approximation that $\boldsymbol{z}$ to the samples of the output of the filter:

Observation 4.4. A better way to approximate the output of a filter is to compute $S\breve{\boldsymbol{x}}$, where S is the corresponding digital filter, and where $\boldsymbol{x}$ are the samples of the input f_1 to the filter.

If S furthermore preserves symmetric extensions, we can view its restriction to symmetric extensions as a mapping S_r from $\mathbb{R}^N$ to $\mathbb{R}^N$ (since any symmetric extension is uniquely characterized from the first half of the elements), and we can thus approximate the analog filter s with the mapping S_r. We have summarized these remarks in Figure 4.1, which is simply a specialization of Figure 3.1 to symmetric functions. The mapping S_r will turn out not to be a digital filter, and we will later characterize these mappings.

Let us therefore try to characterize filters which preserve symmetric extensions. It will turn out that this is where symmetric filters come into the picture. We will solve this in more generality by also considering symmetric vectors:

Definition 4.5 (Symmetric vector)**.** We say that a 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 $\boldsymbol{x}$

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 $1/2$. Clearly, the symmetry point in symmetric extensions as defined in Definition 2.25 is $d = N - \frac{1}{2}$. To characterize symmetric vectors, it turns out to be useful first to characterize how the DFT of a symmetric vector must look like. We will state two results for this. The first result is the simplest, and considers integer symmetry points:

Theorem 4.6 (Symmetric vectors with integer symmetry points)**.** Let d be an integer. The following are equivalent

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

Proof. Assume first that $d = 0$. It follows in this case from property 2(a) of Theorem 2.21 that $(\widehat{\boldsymbol{x}})_n$ is a real vector. Combining this with property 1 of Theorem 2.21 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 2.21it follows that, when d is an integer, $\boldsymbol{x}$ is real and symmetric about d if and only if $(\widehat{\boldsymbol{x}})_n = y_n e^{-2\pi i d n/N}$, where y_n is real and symmetric about 0. This completes the proof. □

Theorem 4.7 (Symmetric vectors with non-integer symmetry points)**.** Let d be an odd multiple of $1/2$. The following are equivalent

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

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

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

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

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

This shows that $y_{N-n} = -y_n$, and this completes one way of the proof.

The other way, we can write

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

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

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

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

$$\begin{aligned}
&2y_n\cos(2\pi((d+s)-d)n/N) = 2y_n\cos(2\pi sn/N)\\
&2y_n\cos(2\pi((d-s)-d)n/N) = 2y_n\cos(-2\pi sn/N) = 2y_n\cos(2\pi sn/N).
\end{aligned}$$

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

Once we have this characterization of symmetric vectors, it is straightforward to characterize filters preserving them, by using a well-known property of the frequency response:

Theorem 4.8. Let S be a filter. The following are equivalent

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

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

1. 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.
2. 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

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

for some d_2, y_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 4.6 and 4.7. 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. $\square$

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. We will therefore write $2N$ for N in the following. In the literature, we encounter mostly symmetric vectors where there is symmetry around $N-1$ or $N-1/2$. From Theorem 4.8 it follows that symmetric filters preserve such symmetric vectors. For these filters we have the following important result:

Theorem 4.9 (Characterization of filters which preserve vectors symmetric about $N - 1/2$)**.** Assume that S is a symmetric filter, and let $\boldsymbol{y} = S\boldsymbol{x}$, where $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^{2N}$. The restriction of S to vectors symmetric about $N-1/2$ is uniquely characterized by the mapping $S_r : \mathbb{R}^N \to \mathbb{R}^N$ which sends $(x_0, \ldots, x_{N-1})$ to $(y_0, \ldots, y_{N-1})$. Moreover, if we write $S = \begin{pmatrix} S_1 & S_2 \\ S_3 & S_4 \end{pmatrix}$, we have that $S_r = S_1 + (S_2)^f$, 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

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

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

$$\begin{aligned} & S_1 \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix} + S_2 \begin{pmatrix} x_{N-1} \\ \vdots \\ x_0 \end{pmatrix} \\ &= S_1 \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix} + (S_2)^f \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix} = \begin{pmatrix} S_1 + (S_2)^f \end{pmatrix} \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix}. \end{aligned}$$

This shows that the mapping $S_r : (x_0, \ldots, x_{N-1}) \to (y_0, \ldots, y_{N-1})$ is well-defined, and that $S_r = S_1 + (S_2)^f$. $\square$

S_2 contains the circulant part of the matrix, and forming $(S_2)^f$ means that the circulant parts switch corners.

Example 4.10. Consider the averaging filter $\boldsymbol{g} = \{\frac{1}{4}, \underline{\frac{1}{2}}, \frac{1}{4}\}$. Let us write down the matrix S_r for the case when $N = 4$. First we obtain the matrix S as

$$\left(\begin{array}{cccc|cccc} \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} \end{array}\right)$$

where we have drawn the boundaries between the blocks S_1, S_2, S_3, S_4. From this we see that

$$S_1 = \begin{pmatrix} \frac{1}{2} & \frac{1}{4} & 0 & 0 \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 \\ 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ 0 & 0 & \frac{1}{4} & \frac{1}{2} \end{pmatrix} \quad S_2 = \begin{pmatrix} 0 & 0 & 0 & \frac{1}{4} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{4} & 0 & 0 & 0 \end{pmatrix} \quad (S_2)^f = \begin{pmatrix} \frac{1}{4} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{4} \end{pmatrix}.$$

From this we get

$$S_r = S_1 + (S_2)^f = \begin{pmatrix} \frac{3}{4} & \frac{1}{4} & 0 & 0 \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 \\ 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ 0 & 0 & \frac{1}{4} & \frac{3}{4} \end{pmatrix}.$$

♣

Note that S_r is not a circulant matrix. Therefore, its eigenvectors are not pure tones. We will return to this. The properties we just have proved essentially say that filters which preserve symmetric vectors have a frequency response where the argument of $\lambda_{S,n}$ ($\lambda_S(\omega)$) is linear in n (ω). Since ω is also called the phase, such filters are often called *linear phase filters*:

Definition 4.11 (Linear phase)**.** We say that a digital filter S has *linear phase* if there exists some d so that $S_{d+n,0} = S_{d-n,0}$ for all n.

Moreover, the case $d = 0$, 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.

4.1.1 Implementations of symmetric filters

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

$$\begin{aligned}(S\boldsymbol{x})_n &= \sum_{k=0}^{N-1} s_k x_{(n-k) \bmod N} \\ &= s_0 x_n + \sum_{k=1}^{(N-1)/2} s_k x_{(n-k) \bmod N} + \sum_{k=(N+1)/2}^{N-1} s_k x_{(n-k) \bmod N} \\ &= s_0 x_n + \sum_{k=1}^{(N-1)/2} s_k x_{(n-k) \bmod N} + \sum_{k=1}^{(N-1)/2} s_k x_{(N-(n-k)) \bmod N} \\ &= s_0 x_n + \sum_{k=1}^{(N-1)/2} s_k (x_{(n-k) \bmod N} + x_{(k-n) \bmod N}). \qquad (4.1)\end{aligned}$$

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 4.12. For symmetric filters one can find implementations with a reduced number of multiplications.

Similarly to as in Section 3.8.2, 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 4.13. 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

$$\begin{aligned}\lambda_S(\omega) =& K(a_1 e^{i\omega} + 1 + a_1 e^{-i\omega}) \ldots (a_m e^{i\omega} + 1 + a_m e^{-i\omega}) \\ & \times (b_1 e^{2i\omega} + c_1 e^{i\omega} + 1 + c_1 e^{-i\omega} + b_1 e^{-2i\omega}) \ldots (b_n e^{2i\omega} + c_n e^{i\omega} + 1 + c_n e^{-i\omega} + b_n e^{-2i\omega}).\end{aligned}$$

We can write $S = KA_1 \ldots A_m B_1 \ldots B_n$, 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.

Exercises for Section 4.1

1. Recall that in Exercise 3.1.3 we wrote down formulas for the output of a filter. Using the results of this section these formulas can be be written in a way which reduces the number of arithmetic operations. Assume that $S = t_{-E}, \ldots, \underline{t_0}, \ldots, t_E$ is a symmetric filter. Use Equation (4.1) to show that $z_n = (S\boldsymbol{x})_n$ in this case can be split into the following different formulas, depending on n:

a. $0 \leq n < E$:

$$z_n = t_0 x_n + \sum_{k=1}^{n} t_k (x_{n+k} + x_{n-k}) + \sum_{k=n+1}^{E} t_k (x_{n+k} + x_{n-k+N}). \quad (4.2)$$

b. $E \leq n < N - E$:

$$z_n = t_0 x_n + \sum_{k=1}^{E} t_k (x_{n+k} + x_{n-k}). \quad (4.3)$$

c. $N - E \leq n < N$:

$$z_n = t_0 x_n + \sum_{k=1}^{N-1-n} t_k(x_{n+k} + x_{n-k}) + \sum_{k=N-1-n+1}^{E} t_k(x_{n+k-N} + x_{n-k}). \tag{4.4}$$

2. Assume that $S = t_{-E}, \ldots, \underline{t_0}, \ldots, t_E$ is a symmetric filter, and let $\boldsymbol{s} = (t_0, \ldots, t_E)\}$. Write a function

```
function z=filterS(s,x)
```

which takes the vector $\boldsymbol{s}$ as input, and returns $\boldsymbol{z} = S\boldsymbol{x}$ using the formulas from Exercise 1.

3. Repeat Exercise 2.2.4 by reimplementing the functions `reducetreble` and `reducebass` using the function `filterS` from the previous exercise. The resulting sound files should sound the same, since the only difference is that we have modified the way we handle the beginning and end portion of the sound samples.

4. In Example 3.36, we mentioned that the filters used in the MP3-standard were constructed from a lowpass prototype filter by multiplying the filter coefficients with a complex exponential. The prototytype filter turns out to be a symmetric filter, however, and then the new filters will not be symmetric after we have multiplied by a complex exponential. Show that, when H is a symmetric lowpass filter (i.e. 0 is the center frequency), the filter $H_n = H\cos\left(2\pi\frac{n(k+1/2)\pi}{2N}\right)$ is also a filter with a symmetric frequency response, with the two center frequencies $\pm\pi n/N$. (this formula for constructing teh new filters is what actually is used in the MP3-standard, rather than the multiplication with complex exponentials).

4.2 Construction of the DCT

In Section 4.1 we characterized filters which preserved symmetric vectors and the symmetry point as the symmetric ones. We mentioned that such filters applied to symmetric vectors are determined by looking at the first half of the input and the output, so that they can be viewed as an operation from $\mathbb{R}^N$ to $\mathbb{R}^N$. We also remarked that the matrix S_r is not a circulant matrix. Therefore, its eigenvectors are not pure tones. Let us find the eigenvectors of S_r. This will enable us define the DCT, which plays the same role for symmetric filters/vectors as the DFT did for filters in diagonalizing them.

Theorem 4.14. The set of all $\boldsymbol{x}$ symmetric around $N - 1/2$ is a vector space of dimension N, and we have that

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

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 have that $(\widehat{\boldsymbol{x}})_n = y_n e^{-2\pi i(N-1/2)n/(2N)}$, with $y_{2N-n} = -y_n$. It is thus clear 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

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

This also means that

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

is an orthonormal basis. □

We immediately get the following result:

Theorem 4.15.

$$\left\{ \frac{1}{\sqrt{2N}} \cos\left(2\pi \frac{0}{2N} \left(k + \frac{1}{2} \right) \right), \left\{ \frac{1}{\sqrt{N}} \cos\left(2\pi \frac{n}{2N} \left(k + \frac{1}{2} \right) \right) \right\}_{n=1}^{N-1} \right\} \tag{4.5}$$

is an orthonormal basis for the set of vectors symmetric around $N - 1/2$ in $\mathbb{R}^{2N}$.

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. □

Finally, we need to establish that the given symmetric vectors are eigenvectors for S, and that the restriction to the first half of the elements are eigenvectors for S_r.

Corollary 4.16 (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 us first show that the basis we have obtained actually are eigenvectors for S. We have that

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

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 $\lambda_{S,2N-n}$. If S preserves symmetric extensions, we know that $\lambda_{S,n} = \lambda_{S,2N-n}$, so that this can be written as

$$\lambda_{S,n}\frac{1}{2}\left(e^{2\pi i(n/(2N))(k+1/2)} + e^{-2\pi i(n/(2N))(k+1/2)}\right) = \lambda_{S,n}\cos\left(2\pi\frac{n}{2N}\left(k+\frac{1}{2}\right)\right).$$

so that the vectors actually are eigenvectors. Finally, it is clear that the first half of the vectors must be eigenvectors of S_r, since when $\boldsymbol{y} = S\boldsymbol{x} = \lambda_{S,n}\boldsymbol{x}$, we also have that

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

Moreover, since

$$\langle x, y\rangle = 2\langle(x_0, x_1, \ldots, x_{N-1}), (y_0, y_1, \ldots, y_{N-1})\rangle$$

whenever $\boldsymbol{x}$ and $\boldsymbol{y}$ are symmetric around $N-1/2$, we only need to multiply the vectors we have by $\sqrt{2}$ to obtain an orthonormal basis of eigenvectors for S_r. This completes the proof. $\square$

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

Definition 4.17 (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 4.18 (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 D_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, $\boldsymbol{y} = D_N \boldsymbol{x}$).

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, D_N^{-1} is the matrix where the $\boldsymbol{d}_n$ are columns. But since these vectors are real and orthonormal, D_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 4.19. D_N is the orthogonal matrix where the rows are $\boldsymbol{d}_n$. Moreover, for any digital filter S which preserves symmetric extensions, $(D_N)^T$ diagonalizes S_r, i.e. $S_r = D_N^T D D_N$ where D is a diagonal matrix.

Let us also make the following definition:

Definition 4.20 (IDCT)**.** We will call $\boldsymbol{x} = (D_N)^T \boldsymbol{y}$ the inverse DCT or (IDCT) of $\boldsymbol{x}$.

Example 4.21. As with Example 2.19, 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 2.19 does not give the same type of nice, exact values, so let us instead consider the case $N = 2$. We have that

$$D_4 = \begin{pmatrix} \frac{1}{\sqrt{2}}\cos(0) & \frac{1}{\sqrt{2}}\cos(0) \\ \cos\left(\frac{\pi}{2}\left(0+\frac{1}{2}\right)\right) & \cos\left(\frac{\pi}{2}\left(1+\frac{1}{2}\right)\right) \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$$

The DCT of the same vector as in Example 2.19 can now be computed as:

$$D_2 \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} \frac{3}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix}.$$

♣

Example 4.22. A direct implementation of the DCT could be made as follows:

```
function y=DCTImpl(x)
  N=length(x);
  DN=zeros(N);
  DN(1,:)=ones(1,N)/sqrt(N);
  for n=1:N
    DN(n,:)=cos(2*pi*((n-1)/(2*N))*((0:N-1)+1/2))*sqrt(2/N);
  end
  y=DN*x;
```

In the next chapter we will see that one also can make a much more efficient implementation of the DCT than this. ♣

Similarly to Theorem 2.28 for 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 4.23 (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 $\boldsymbol{y} = D_N\boldsymbol{x}$.

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. $\square$

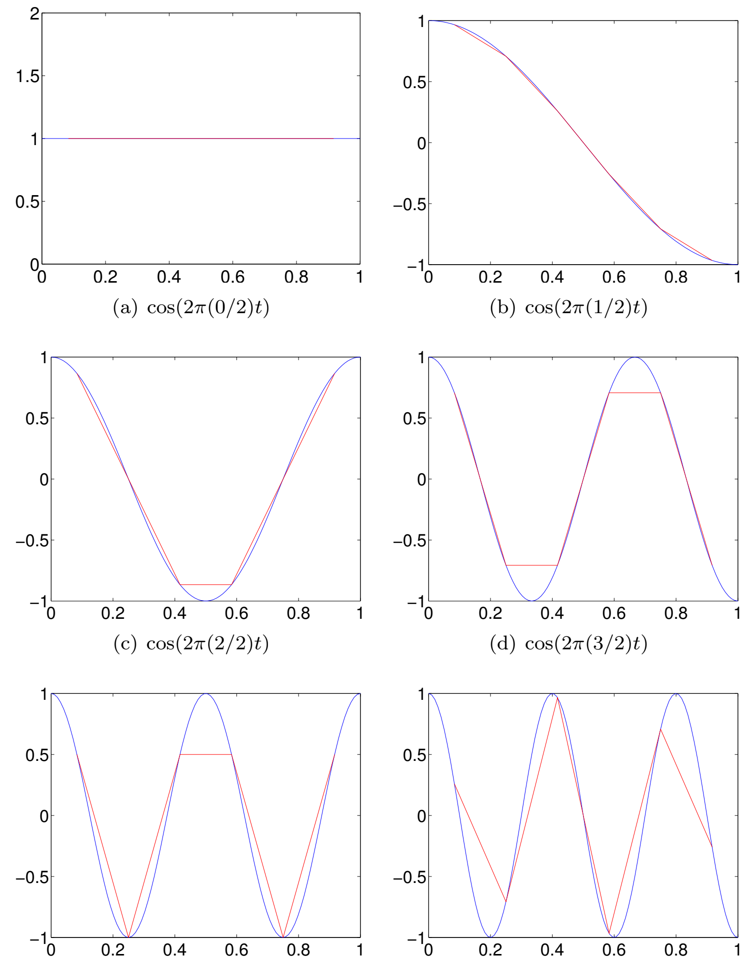
From this result, and from our construction, it follows that the DCT can be used as the unknown operation in Figure 2.3. There is, however, a slight difference to how we applied the DFT, due to the subtle change in the sample points, from kT/N for the DFT, to $(2k+1)T/(2N)$ for the DCT. The sample points 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 4.23 for $T = 1$, as well as the sample points used in that theorem. The sample points in (a) correspond to the first column in the DCT matrix, the sample points in (b) 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 $\boldsymbol{y} = D_N\boldsymbol{x}$ and $\boldsymbol{x} = (D_N)^T\boldsymbol{y}$ 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 $\boldsymbol{y} = D_N\boldsymbol{x}$, we can analyse 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.

Exercises for Section 4.2

1. 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.


(a) $\cos(2\pi(0/2)t)$ (b) $\cos(2\pi(1/2)t)$

(c) $\cos(2\pi(2/2)t)$ (d) $\cos(2\pi(3/2)t)$

(e) $\cos(2\pi(4/2)t)$ (f) $\cos(2\pi(5/2)t)$

Figure 4.2: The 6 different sinusoids used in DCT for $N = 6$, i.e. $\cos(2\pi(n/2)t)$, $0 \le n < 6$. The plots also show piecewise linear functions (in red) between the sample points $\frac{2k+1}{2N}$ $0 \le k < 6$, since only the values at these points are used in Theorem 4.23.

b. Use Matlab 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?

4.3 Use of DCT in lossy compression of sound

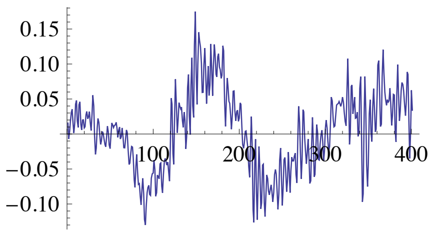
The DCT is particularly popular for processing sound data before they are compressed with lossless techniques such as Huffman coding or arithmetic coding. The reason is, as mentioned, that the DCT provides a better approximation from a low-dimensional space than the DFT does. The computation of scale-factors in MP3 is based on applying a variant of the DCT (called the Modified Discrete Cosine Transform, MDCT) to groups of 576 (in special circumstances 192) samples. One does here not actually apply the DCT directly. Rather one applies a much more complex transformation, which can be implemented in parts by using DCT in a clever way.

We have mentioned that we could achieve compression by setting the Fourier coefficients which are small to zero. Translated to the DCT, we should set the DCT coefficients which are small to zero, and we apply the inverse DCT in order to reconstruct the signal in order to play it again. Let us test compression based on this idea. The plots in figure 4.3 illustrate the principle. A signal is shown in (a) and its DCT in (b). In (d) all values of the DCT with absolute value smaller than 0.02 have been set to zero. The signal can then be reconstructed with the inverse DCT; the result of this is shown in (c). The two signals in (a) and (c) visually look almost the same even though the signal in (c) can be represented with less than 25 % of the information present in (a).

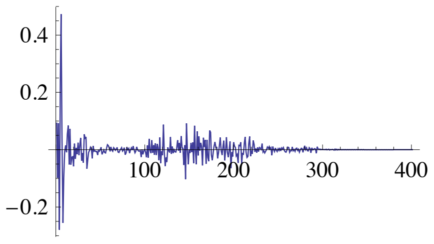
We test this compression strategy on a data set that consists of 300 001 points. We compute the DCT and set all values smaller than a suitable tolerance to 0. With a tolerance of 0.04, a total of 142 541 values are set to zero. When we then reconstruct the sound with the inverse DCT, we obtain a signal that differs at most 0.019 from the original signal. To verify that the new file is not too different from the old file, we can take the read sound samples from `castanets.wav`, run the following function for different `eps`

```
function A=skipsmallvals(eps,A)
  B=dct(A);
  B=(B>=eps).*B;
  A=idct(B);
```

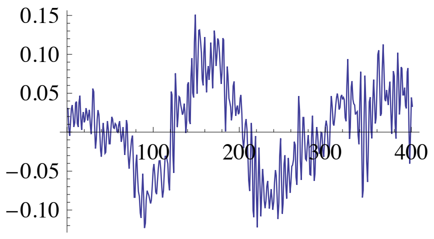
and play the new samples. We have here used Matlab's functions for computing the DCT and IDCT, called `dct`, and `idct`. These functions are defined in Matlab exactly as they are here, contrary to the case for the FFT (where a different normalizing factor was used). Finally we can store the signal by storing a `gzip`'ed version of the DCT-values (as 32-bit floating-point numbers) of the



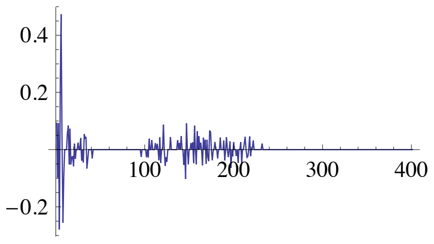
(a)



(b)



(c)



(d)

Figure 4.3: The signal in (a) are the sound samples from a small part of a song. The plot in (b) shows the DCT of the signal. In (d), all values of the DCT that are smaller than 0.02 in absolute value have been set to 0, a total of 309 values. In (c) the signal has been reconstructed from these perturbed values of the DCT. The values have been connected by straight lines to make it easier to interpret the plots.

perturbed signal. This gives a file with 622 551 bytes, which is 88 % of the `gzip`'ed version of the original data.

The choice of the DCT in the MP3 standard has also much to do with that the DCT, just as the DFT, has a very efficient implementation, as we will see next.

Exercises for Section 4.3

1. In Section 4.3 we implemented the function `skipsmallvals`, which ran a DCT on the entire vector. Explain why there is less computation involved in splitting the vector into many parts and performing a DCT for each part. Change the code accordingly. In light of Example 2.27, what are the disadvantages with this strategy?

2. As in Example 4.21, state the exact cartesian form of the DCT matrix for the case $N = 3$.

4.4 Efficient implementations of the DCT

When we defined the DCT in the preceding section, we converted to the frequency domain, so that the DFT is involved. This enables us to use efficient implementations of the DFT, such as the one we considered in Section 2.9. However, the way we defined the DCT, there is a penalty in that we need to compute a DFT of twice the length (the length doubles when we instead consider the symmetric extension). We are also forced to use complex arithmetic (note that any complex multiplication corresponds to 4 real multiplications, and that any complex addition corresponds to 2 real additions). Is there a way to get around these penalties, so that we can get an implementation of the DCT which is more efficient, and uses less additions and multiplications than the one you made in Exercise 1? The following theorem states an expression of the DCT which achieves this. This expression is, together with a similar result for the DFT in the next section, much used in practical implementations:

Theorem 4.24 (DCT algorithm)**.** Let $\boldsymbol{y} = D_N\boldsymbol{x}$ be the N-point DCT of the vector $\boldsymbol{x}$. Then we have that

$$y_n = c_{n,N}\left(\cos\left(\pi\frac{n}{2N}\right)\Re((F_N\boldsymbol{x}^{(1)})_n) + \sin\left(\pi\frac{n}{2N}\right)\Im((F_N\boldsymbol{x}^{(1)})_n)\right), \tag{4.7}$$

where $c_{0,N} = 1$ and $c_{n,N} = \sqrt{2}$ for $n \geq 1$, and where $\boldsymbol{x}^{(1)} \in \mathbb{R}^N$ is defined by

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

Proof. The N-point DCT of $\boldsymbol{x}$ is

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

Splitting this sum into two sums, where the indices are even and odd, we get

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

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

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

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

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

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

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

where we have recognized the N-point DFT, and where $c_{n,N} = \sqrt{N} d_{n,N}$. Inserting the values for $d_{n,N}$, we see that $c_{0,N} = 1$ and $c_{n,N} = \sqrt{2}$ for $n \geq 1$, which agrees with the definition of $c_{n,N}$ in the theorem. 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 $Q_{n,n} = c_{n,N}e^{-\pi in/(2N)}$, the result can also be written on the more compact form

$$\boldsymbol{y} = D_N\boldsymbol{x} = \Re\left(Q_N F_N \boldsymbol{x}^{(1)}\right).$$

We will, however, not use this form, since there is complex arithmetic involved, contrary to (4.7). Let us see how we can use (4.7) to implement the DCT, once we already have implemented the DFT in terms of the function `FFTImpl` as in Section 2.9:

```
function y = DCTImpl(x)
N = length(x);
if N == 1
    y = x;
else
    x1 = [x(1:2:(N-1)); x(N:(-2):2)];
    y = FFTImpl(x1);
    rp = real(y);
    ip = imag(y);
    y = cos(pi*((0:(N-1))')/(2*N)).*rp + sin(pi*((0:(N-1))')/(2*N)).*ip;
    y(2:N) = sqrt(2)*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 (4.7).

4.4.1 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 4.24, for $n \geq 1$

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

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

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

Adding we get

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

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

Theorem 4.25 (IDCT algorithm)**.** Let $\boldsymbol{x} = (D_N)^T\boldsymbol{y}$ be the IDCT of $\boldsymbol{y}$. and let $\boldsymbol{z}$ be the vector with component 0 being $\frac{1}{c_{0,N}}y_0$, and the remaining components being $\frac{1}{c_{n,N}}e^{\pi in/(2N)}(y_n - iy_{N-n})$. Then we have that

$$\boldsymbol{x}^{(1)} = (F_N)^H\boldsymbol{z},$$

where $\boldsymbol{x}^{(1)}$ is defined as in Theorem 4.24.

The implementation of IDCT can thus go as follows:

```
function x = IDCTImpl(y)
N = length(y);
if N == 1
    x = y(1);
else
  Q=exp(pi*1i*((0:(N-1))')/(2*N));
  Q(2:N)=Q(2:N)/sqrt(2);
  yrev=y(N:(-1):2);
  toapply=[ y(1); Q(2:N).*(y(2:N)-1i*yrev) ];
  x1=IFFTImpl(toapply);
  x=zeros(N,1);
  x(1:2:(N-1))=x1(1:(N/2));
  x(2:2:N)=x1(N:(-1):(N/2+1));
end
```

4.4.2 Reduction in the number of multiplications with the DCT

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 4.26 (Number of multiplications required by the DCT and IDCT algorithms)**.** Both the N-point DCT and IDCT factorizations given by Theorem 4.24 and Theorem 4.25 require $O(2(N+1)\log_2 N)$ real multiplications. In comparison, the number of real multiplications required by a direct implementation of the N-point DCT and IDCT is N^2.

Proof. By Theorem 2.37, the number of multiplications required by the FFT is $O(2N\log_2 N)$. By Theorem 4.24, two additional multiplications are needed for each index, giving additionally $2N$ multiplications in total, so that we end up with $O(2(N+1)\log_2 N)$ real multiplications. For the IDCT, note first that the vector $\boldsymbol{z} = \frac{1}{c_{n,N}}e^{\pi in/(2N)}(y_n - iy_{N-n})$ seen in Theorem 4.25 should require $4N$ real multiplications to compute. But since the IDFT of $\boldsymbol{z}$ is real, $\boldsymbol{z}$ must have conjugate symmetry between the first half and the second half of the coefficients, so that we only need to perform $2N$ multiplications. Since the IFFT takes an additional $O(2N\log_2 N)$ real multiplications, we end up with a total of $O(2N + 2N\log_2 N) = O(2(N+1)\log_2 N)$ real multiplications also here. It is clear that the direct implementation of the DCT and IDCT needs N^2 multiplications, since only real arithmetic is involved. $\square$

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. In the next section we will show an example on how this can be achieved, with the help of extra work and some tedious math. Some more notes on computational complexity are in order. For instance, 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. Another point has to do with multiplication of $\frac{1}{\sqrt{N}}$. As long as $N = 2^{2r}$, multiplication with N need not be considered as a multiplication, since it can be implemented using a bitshift.

4.4.3 *An efficient joint implementation of the DCT and the FFT

We will now present a more advanced FFT algorithm, which will turn out to decrease the number of multiplications and additions even further. It also has the advantage that it avoids complex number arithmetic altogether (contrary to Theorem 2.31), and that it factors the computation into smaller FFTs and DCTs so that we can also use our previous DCT implementation. This implementation of the DCT and the DFT is what is mostly used in practice. For simplicity we will drop this presentation for the inverse transforms, and concentrate only on the DFT and the DCT.

Theorem 4.27 (Revised FFT algorithm)**.** Let $\boldsymbol{y} = F_N\boldsymbol{x}$ be the N-point DFT of the real vector $\boldsymbol{x}$. Then we have that

$$\Re(y_n) = \begin{cases} \frac{1}{\sqrt{2}}\Re((F_{N/2}\boldsymbol{x}^{(e)})_n) + (ED_{N/4}\boldsymbol{z})_n & 0 \le n \le N/4-1 \\ \frac{1}{\sqrt{2}}\Re((F_{N/2}\boldsymbol{x}^{(e)})_n) & n = N/4 \\ \frac{1}{\sqrt{2}}\Re((F_{N/2}\boldsymbol{x}^{(e)})_n) - (ED_{N/4}\boldsymbol{z})_{N/2-n} & N/4+1 \le n \le N/2-1 \end{cases} \tag{4.8}$$

$$\Im(y_n) = \begin{cases} \frac{1}{\sqrt{2}}\Im((F_{N/2}\boldsymbol{x}^{(e)})_n) & q = 0 \\ \frac{1}{\sqrt{2}}\Im((F_{N/2}\boldsymbol{x}^{(e)})_n) + (ED_{N/4}\boldsymbol{w})_{N/4-n} & 1 \le n \le N/4-1 \\ \frac{1}{\sqrt{2}}\Im((F_{N/2}\boldsymbol{x}^{(e)})_n) + (ED_{N/4}\boldsymbol{w})_{n-N/4} & N/4 \le n \le N/2-1 \end{cases} \tag{4.9}$$

where $\boldsymbol{x}^{(e)}$ is as defined in Theorem 2.31, where $\boldsymbol{z}, \boldsymbol{w} \in \mathbb{R}^{N/4}$ defined by

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

and where E is a diagonal matrix with diagonal entries $E_{0,0} = \frac{1}{2}$ and $E_{n,n} = \frac{1}{2\sqrt{2}}$ for $n \ge 1$.

Proof. Taking real and imaginary parts in (2.13) we obtain

$$\begin{aligned} \Re(y_n) &= \frac{1}{\sqrt{2}}\Re((F_{N/2}\boldsymbol{x}^{(e)})_n + \frac{1}{\sqrt{2}}\Re((D_{N/2}F_{N/2}\boldsymbol{x}^{(o)})_n) \\ \Im(y_n) &= \frac{1}{\sqrt{2}}\Im((F_{N/2}\boldsymbol{x}^{(e)})_n + \frac{1}{\sqrt{2}}\Im((D_{N/2}F_{N/2}\boldsymbol{x}^{(o)})_n). \end{aligned}$$

These equations explain the first parts on the right hand side in (4.8) and (4.9).

Furthermore, for $0 \leq n \leq N/4-1$ we can write

$$\begin{aligned}
&\Re((D_{N/2}F_{N/2}\boldsymbol{x}^{(o)})_n) \\
&= \frac{1}{\sqrt{N/2}}\Re(e^{-2\pi i n/N}\sum_{k=0}^{N/2-1}(\boldsymbol{x}^{(o)})_k e^{-2\pi i n k/(N/2)}) \\
&= \frac{1}{\sqrt{N/2}}\Re(\sum_{k=0}^{N/2-1}(\boldsymbol{x}^{(o)})_k e^{-2\pi i n(k+\frac{1}{2})/(N/2)}) \\
&= \frac{1}{\sqrt{N/2}}\sum_{k=0}^{N/2-1}(\boldsymbol{x}^{(o)})_k \cos\left(2\pi\frac{n(k+\frac{1}{2})}{N/2}\right) \\
&= \frac{1}{\sqrt{N/2}}\sum_{k=0}^{N/4-1}(\boldsymbol{x}^{(o)})_k \cos\left(2\pi\frac{n(k+\frac{1}{2})}{N/2}\right) \\
&\quad + \frac{1}{\sqrt{N/2}}\sum_{k=0}^{N/4-1}(\boldsymbol{x}^{(o)})_{N/2-1-k} \cos\left(2\pi\frac{n(N/2-1-k+\frac{1}{2})}{N/2}\right) \\
&= \frac{1}{\sqrt{2}}\frac{1}{\sqrt{N/4}}\sum_{k=0}^{N/4-1}((\boldsymbol{x}^{(o)})_k + (\boldsymbol{x}^{(o)})_{N/2-1-k}) \cos\left(2\pi\frac{n\left(k+\frac{1}{2}\right)}{N/2}\right) \\
&= (E_0 D_{N/4}\boldsymbol{z})_n,
\end{aligned}$$

where we have used that cos is periodic with period 2π and symmetric, where z is the vector defined in the text of the theorem, where we have recognized the DCT matrix, and where E_0 is a diagonal matrix with diagonal entries $(E_0)_{0,0} = \frac{1}{\sqrt{2}}$ and $(E_0)_{n,n} = \frac{1}{2}$ for $n \geq 1$ (E_0 absorbs the factor $\frac{1}{\sqrt{N/2}}$, and the factor $d_{n,N}$ from the DCT). By absorbing the additional factor $\frac{1}{\sqrt{2}}$, we get a matrix E as stated in the theorem. For $N/4+1 \leq n \leq N/2-1$, everything above but the last statement is valid. We can now use that

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

to arrive at $-(E_0 D_{N/4}\boldsymbol{z})_{N/2-n}$ instead. For the case $n = \frac{N}{4}$ all the cosine entries are zero, and this completes (4.8). For the imaginary part, we obtain as above

$$\begin{aligned}
&\Im((D_{N/2}F_{N/2}\boldsymbol{x}^{(o)})_n) \\
&= \frac{1}{\sqrt{N/2}}\sum_{k=0}^{N/4-1}((\boldsymbol{x}^{(o)})_{N/2-1-k} - (\boldsymbol{x}^{(o)})_k)\sin\left(2\pi\frac{n(k+\frac{1}{2})}{N/2}\right) \\
&= \frac{1}{\sqrt{N/2}}\sum_{k=0}^{N/4-1}((\boldsymbol{x}^{(o)})_{N/2-1-k} - (\boldsymbol{x}^{(o)})_k)(-1)^k\cos\left(2\pi\frac{(N/4-n)(k+\frac{1}{2})}{N/2}\right).
\end{aligned}$$

where we have used that sin is periodic with period 2π and anti-symmetric, that

$$\begin{aligned}\sin\left(2\pi\frac{n(k+\frac{1}{2})}{N/2}\right) &= \cos\left(\frac{\pi}{2}-2\pi\frac{n(k+\frac{1}{2})}{N/2}\right)\\ &= \cos\left(2\pi\frac{(N/4-n)(k+\frac{1}{2})}{N/2}-k\pi\right)\\ &= (-1)^k\cos\left(2\pi\frac{(N/4-n)(k+\frac{1}{2})}{N/2}\right),\end{aligned}$$

When $n=0$ this is 0 since all the cosines entries are zero. When $1\le n\le N/4$ this is $(E_0D_{N/4}\boldsymbol{w})_{N/4-n}$, where $\boldsymbol{w}$ is the vector defined as in the text of the theorem. For $N/4\le n\le N/2-1$ we arrive instead at $(E_0D_{N/4}\boldsymbol{z})_{n-N/4}$, similarly to as above. This also proves (4.9), and the proof is done. □

As for Theorem 2.31, this theorem says nothing about the coefficients y_n for $n>\frac{N}{2}$. These are obtained in the same way as before through symmetry. The theorem also says nothing about $y_{N/2}$. This can be obtained with the same formula as in Theorem 2.31.

It is more difficult to obtain a matrix interpretation for Theorem 4.27, so we will only sketch an algorithm which implements it. The following code implements the recursive formulas for $\Re F_N$ and $\Im F_N$ in the theorem:

```
function y = FFTImpl2(x)
N = length(x);
if N == 1
    y = x;
elseif N==2
    y = 1/sqrt(2)*[x(1) + x(2); x(1) - x(2)];
else
    xe = x(1:2:(N-1));
    xo = x(2:2:N);
    yx = FFTImpl2(xe);

    z = x(N:(-2):(N/2+2))+x(2:2:(N/2));
    dctz = DCTImpl(z);
    dctz(1)=dctz(1)/2;
    dctz(2:length(dctz)) = dctz(2:length(dctz))/(2*sqrt(2));

    w = (-1).^((0:(N/4-1))').*(x(N:-2:(N/2+2))-x(2:2:(N/2)));
    dctw = DCTImpl(w);
    dctw(1)=dctw(1)/2;
    dctw(2:length(dctw)) = dctw(2:length(dctw))/(2*sqrt(2));

    y = yx/sqrt(2);
    y(1:(N/4))=y(1:(N/4))+dctz;
    if (N>4)
```

```
      y((N/4+2):(N/2))=y((N/4+2):(N/2))-dctz((N/4):(-1):2);
      y(2:(N/4))=y(2:(N/4))+1j*dctw((N/4):(-1):2);
    end
    y((N/4+1):(N/2))=y((N/4+1):(N/2))+1j*dctw;
    y = [y; ...
         sum(xe-xo)/sqrt(N); ...
         conj(y((N/2):(-1):2))];
end
```

In addition, we need to change the code for `DCTImpl` so that it calls `FFTImpl2` instead of `FFTImpl`. The following can now be shown:

Theorem 4.28 (Number of multiplications required by the revised FFT algorithm)**.** Let M_N be the number of real multiplications required by the revised algorithm of Theorem 4.27. Then we have that $M_N = O(\frac{2}{3}N\log_2 N)$.

This is a big reduction from the $O(2N\log_2 N)$ required by the FFT algorithm from Theorem 2.31. We will not prove Theorem 4.28. Instead we will go through the steps in a proof in Exercise 3. The revised FFT has yet a bigger advantage that the FFT when it comes to parallel computing: It splits the computation into, not two FFT computations, but three computations (one of which is an FFT, the other two DCT's). This makes it even easier to make use of many cores on computers which have support for this.

Exercises for Section 4.4

1. Write a function

```
function samples=DCTImpl(x)
```

which returns the DCT of the column vector $\boldsymbol{x} \in \mathbb{R}^{2N}$ as a column vector. The function should use the FFT-implementation from the previous section, and the factorization $C = E^{-1}AFB$ from above. The function should not construct the matrices A, B, E explicitly.

2. Explain why, if `FFTImpl` needs M_N multiplications A_N additions, then the number of multiplications and additions required by `DCTImpl` are $M_N + 2N$ and $A_N + N$, respectively.

3. In this exercise we will compute the number of real multiplications needed by the revised N-point FFT algorithm of Theorem 4.27, denoted M_N.

a. Explain from the algorithm of Theorem 4.27 that

$$M_N = 2(M_{N/4} + N/2) + M_{N/2} = N + M_{N/2} + 2M_{N/4}. \qquad (4.10)$$

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

$$x_{r+2} - x_{r+1} - 2x_r = 4 \times 2^r.$$

c. Show that the general solution to the difference equation is

$$x_r = \frac{2}{3} r 2^r + C 2^r + D(-1)^r.$$

d. Explain why $M_N = O(\frac{2}{3} N \log_2 N)$ (you do not need to write down the initial conditions for the difference equation in order to find the particular solution to it).

4.5 Summary

We started this chapter by noting that an analog filter could be better approximated with a digital filter if we instead considered the symmetric extension. If the filter also was symmetric, we showed that it preserved these symmetric extensions, and we obtained an N-dimensional mapping representing this process. We found the eigenvectors and eigenvalues for this mapping, and this was what lead us to the definition of the DCT. We also showed how to obtain an efficient implementation of the DCT, which could reuse the FFT implementation. We also took a look at the role of the DCT in the MP3 standard. The standard document for MP3 [7] 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.