

Fourier theory, wavelet analysis and nonlinear
optimization
Lecture notes for the course MAT-INF2360

Øyvind Ryan, Geir Dahl, and Knut Mørken

March 28, 2012

Contents

I	Fourier analysis and applications to sound processing	8
1	Sound	9
1.1	Loudness: Sound pressure and decibels	10
1.2	The pitch of a sound	13
1.3	Digital sound	17
1.4	Simple operations on digital sound	19
1.4.1	Playing a sound	19
1.4.2	Filtering operations	23
1.5	Compression of sound and the MP3 standard	30
1.6	Summary	32
2	Fourier analysis for periodic functions: Fourier series	33
2.1	Basic concepts	33
2.1.1	Fourier series for symmetric and antisymmetric functions	42
2.2	Complex Fourier series	45
2.3	Rate of convergence for Fourier series	48
2.4	Some properties of Fourier series	51
2.5	Summary	54
3	Fourier analysis for vectors	55
3.1	Basic ideas	55
3.2	The Discrete Fourier Transform	57
3.2.1	Connection between the DFT and Fourier series	62
3.2.2	Interpolation with the DFT	65
3.2.3	Sampling and reconstruction with the DFT	66
3.3	Operations on vectors: filters	69
3.3.1	Formal definition of filters and frequency response	72
3.3.2	Some properties of the frequency response	75
3.3.3	Assembling the filter matrix and compact notation	79
3.3.4	Some examples of filters	80
3.3.5	Time-invariance of filters	88
3.3.6	Linear phase filters	89
3.3.7	Perfect reconstruction systems	90

3.4	Symmetric digital filters and the DCT	95
3.4.1	Other types of symmetric extensions	102
3.4.2	Use of DCT in lossy compression of sound	104
3.5	Summary	107
4	Implementation of the DFT and the DCT	108
4.1	The Fast Fourier Transform (FFT)	108
4.1.1	The Inverse Fast Fourier Transform (IFFT)	112
4.1.2	Reduction in the number of multiplications with the FFT	112
4.2	Efficient implementations of the DCT	115
4.2.1	Efficient implementations of the IDCT	118
4.2.2	Reduction in the number of multiplications with the DCT	119
4.2.3	*An efficient joint implementation of the DCT and the FFT	120
4.3	Summary	124
II	Wavelets and applications to image processing	125
5	Wavelets	126
5.1	Why wavelets?	127
5.2	Wavelets constructed from piecewise constant functions	128
5.2.1	Resolution spaces	128
5.2.2	Function approximation property	132
5.2.3	Detail spaces and wavelets	133
5.3	Multiresolution analysis for piecewise constant functions	142
5.3.1	Extraction of details at higher resolutions	142
5.3.2	Matrix factorization of the DWT	149
5.3.3	Summary	151
5.4	Wavelets constructed from piecewise linear functions	154
5.4.1	Multiresolution analysis	155
5.4.2	Detail spaces and wavelets	159
5.5	Alternative wavelets for piecewise linear functions	166
5.6	Wavelets and filters	174
5.6.1	Frequency response for the Haar Wavelet	177
5.6.2	Frequency responses for wavelets of piecewise linear func- tions	177
5.6.3	Filter-based algorithm for the DWT and the IDWT	180
5.7	Summary	186
6	Digital images	187
6.1	What is an image?	187
6.1.1	Light	187
6.1.2	Digital output media	188
6.1.3	Digital input media	189
6.1.4	Definition of digital image	189
6.1.5	Images as surfaces	193

6.2	Operations on images	194
6.2.1	Images and MATLAB	194
6.2.2	Comparing the first derivatives	207
6.2.3	Second-order derivatives	207
6.3	Adaptations to image processing	209
6.3.1	Lossless coding	210
6.3.2	Quantization	210
6.3.3	Preprocessing	210
6.3.4	Tiles, blocks, and error resilience	210
6.3.5	Metadata	211
6.4	Summary	211
7	Definition and properties of tensor products	212
7.1	The tensor product of vectors	213
7.2	Change of bases in tensor products	220
7.3	Summary	225
8	Tensor products in a wavelet setting	226
8.1	Adopting the tensor product terminology to wavelets	228
8.2	Summary	246
III	Nonlinear optimization	247
9	The basics and applications	248
9.1	The basic concepts	249
9.2	Some applications	250
9.2.1	Portfolio optimization	251
9.2.2	Fitting a model	252
9.2.3	Maximum likelihood	253
9.2.4	Optimal control problems	254
9.2.5	Linear optimization	255
9.3	Multivariate calculus and linear algebra	256
10	A crash course in convexity	261
10.1	Convex sets	261
10.2	Convex functions	262
10.3	Properties of convex functions	264
11	Nonlinear equations	268
11.1	Equations and fixed points	268
11.2	Newton's method	270
12	Unconstrained optimization	276
12.1	Optimality conditions	276
12.2	Methods	279

13 Constrained optimization - theory	288
13.1 Equality constraints and the Lagrangian	288
13.2 Inequality constraints and KKT	293
13.3 Convex optimization	299
14 Constrained optimization - methods	305
14.1 Equality constraints	305
14.2 Inequality constraints	306
Mathematics index	317
Index for MATLAB commands	320

Preface

These lecture notes have been written for the course MAT-INF2360, Applications of Linear Algebra, at UiO. The different parts of this course deals with topics such as Fourier analysis, wavelet theory, and nonlinear optimization. A central idea in the course 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.

These notes are split into three parts, one on Fourier analysis, one on wavelets, and one on nonlinear optimization. The first two parts have strong connections with the field of *signal processing*, which also is taught in other courses at UiO, such as INF3470. 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 that already known from linear algebra courses, such as MAT1120. 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 make the necessary connections in order to make a student of signal processing person to feel at home.

The third part on nonlinear optimization also has its mathematical foundations in linear algebra, but multivariate calculus as taught in MAT1110 is also an important ingredient.

Notation

We will follow multivariate calculus and linear algebra notation as you know it from MAT1110 and MAT1120. In particular, vectors will be in boldface (\mathbf{x} , \mathbf{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 \mathbf{x}^T , where \mathbf{x} is a (column) vector. Vector-valued functions will be in uppercase boldface (\mathbf{F} , \mathbf{G} , etc.). Functions are written using both uppercase and lowercase. Uppercase is often used for the component functions of a vector-valued function.

Acknowledgment

The parts on Fourier analysis and wavelet theory have been written by Øyvind Ryan, with many useful suggestions and help from Knut Mørken and Geir Dahl. The part in nonlinear optimization has been written by Geir Dahl, with many useful suggestions and help from Øyvind Ryan. The three authors would all like to thank each other for establishing these notes. The authors would also like to

thank Andreas Våvang Solbrå for his valuable contributions to the notes.

Geir Dahl, Knut Mørken, and Øyvind Ryan
Oslo, January 2012.

Part I

Fourier analysis and applications to sound processing

Chapter 1

Sound

A major part of the information we receive and perceive every day is in the form of audio. Most of these 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 signal*.

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 next chapters we are going to study this from a practical and computational

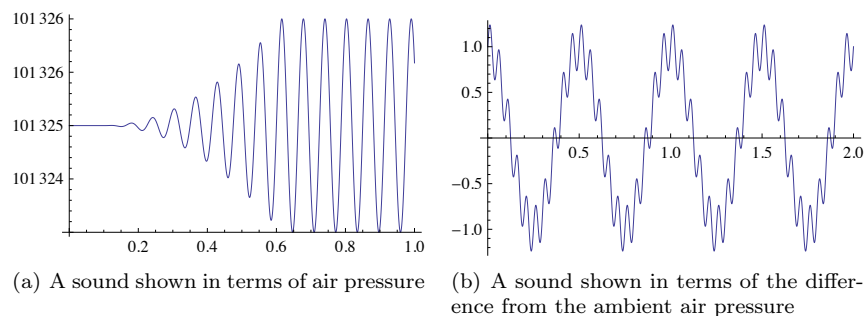


Figure 1.1: Two examples of audio signals.

perspective. Towards the end of this chapter we also consider the basics of digital audio, and illustrate its power by performing some simple operations on digital sounds.

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 against 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.

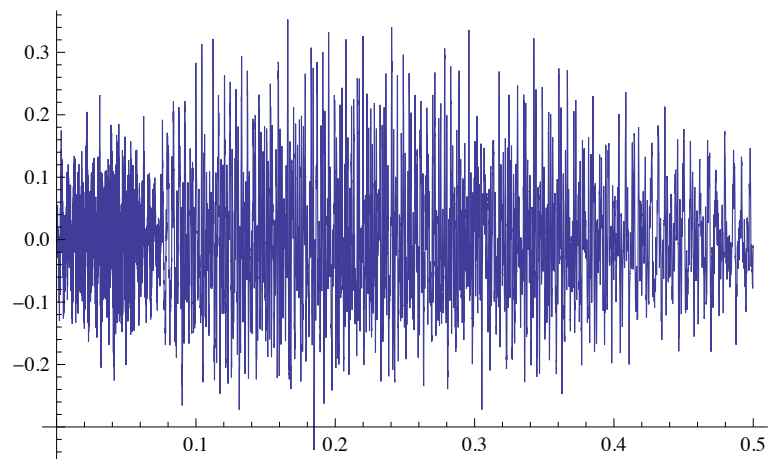
It is common to relate a given sound pressure to the smallest sound pressure that can be perceived, as a level on a decibel scale,

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

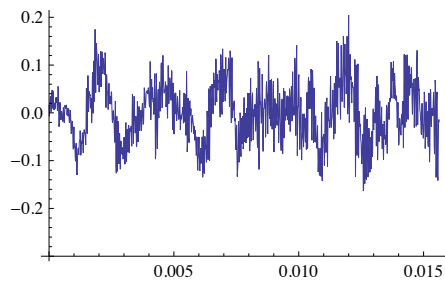
Here p is the measured sound pressure while p_{ref} is the sound pressure of a just perceivable sound, usually considered to be 0.00002 Pa.

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.

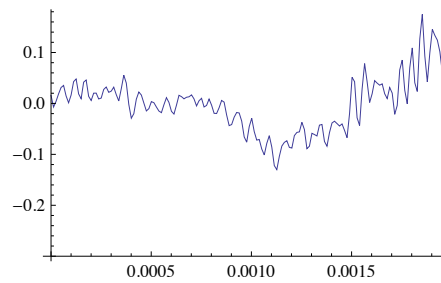
The sounds in Figure 1.1 are synthetic in that they were constructed from mathematical formulas (see Exercises 1.4.2 and 1.4.3). The sounds in Figure 1.2 on the other hand show the variation in air pressure when there is no mathematical formula involved, such as is the case for a song. In (a) there are 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.

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 .

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 .

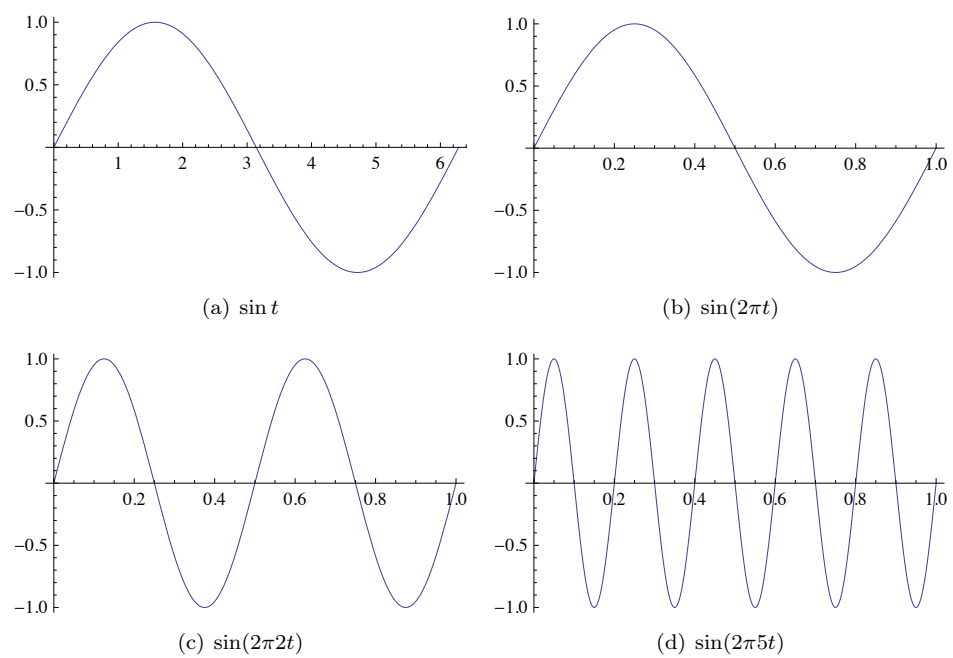


Figure 1.3: Versions of sin with different frequencies.

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.

Any sound may be considered to be a function. In the next chapter we are going to see that 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 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 there will be most information 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 sounds. 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

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

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 5 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

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

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 (in Exercise 5 you will learn how to generate this sound). Again you will note that the triangle wave has the same pitch as $\sin(2\pi 440t)$,

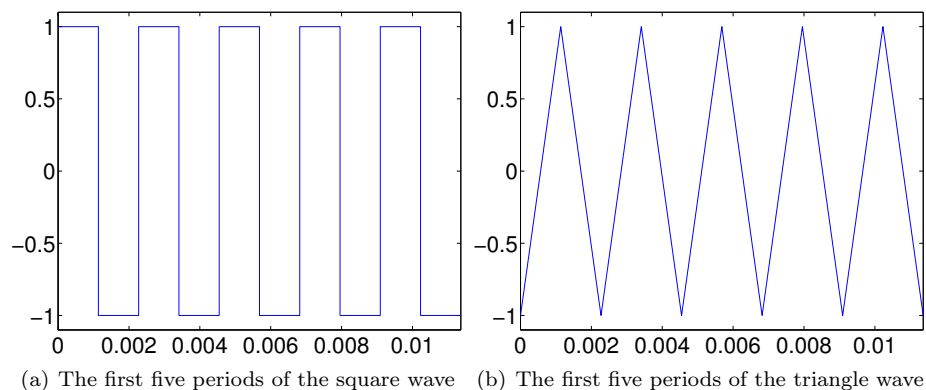


Figure 1.4: The square wave and the triangle wave, two functions with regular oscillations, but which are not simple, trigonometric functions.

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 2.1 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.

1.3 Digital sound

In the previous section we considered some basic properties of sound, but it was all in terms of functions defined for all time instances in some interval. On computers and various kinds of media players the sound is usually *digital* which means that the sound is represented by a large number of function values, and not by a function defined for all times in some interval.

Definition 1.13 (Digital sound). A digital sound is a sequence $\mathbf{x} = \{x_i\}_{i=0}^N$ 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.,

$$x_i = f(i/f_s), \quad \text{for } i = 0, 1; \dots, N.$$

The measurements are often referred to as samples. The time between successive measurements is called the sampling period and is usually denoted T_s . If the sound is in stereo there will be two arrays \mathbf{x}_1 and \mathbf{x}_2 , one for each channel. Measuring the sound is also referred to as sampling the sound, or analog to digital (AD) conversion.

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.

In Exercise 4 you will be asked to implement a Matlab-function which plays a pure sound with a given frequency on your computer. For this you will need to know that for a pure tone with frequency f , you can obtain its samples over a period of 3 seconds with sampling rate f_s from the code

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

Here the code is in MATLAB. MATLAB code will be displayed in this way throughout these notes.

Example 1.14. 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 1.15. 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.

1.4 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 this can be done. From Definition 1.13, digital sound is just an array of sample values $\mathbf{x} = (x_i)_{i=0}^{N-1}$, 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, and if we are working with sound we need a mechanism for doing this.

1.4.1 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. We will now describe how to use Matlab to play the same sounds. There we have the two functions

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

These simply play an audio segment encapsulated by the object `playerobj` (we will shortly see how we can construct such an object from given audio samples and sampling rate). `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 1.16. The basic command in a programming environment that handles sound takes as input an array of sound samples \mathbf{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 from a mathematical function on our own will typically have one 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 considered above 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 1.17 (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 1.18. 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 1.19 (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 1.20. Let $\mathbf{x} = (x_i)_{i=0}^{N-1}$ be the samples of a digital sound. Then the samples $\mathbf{y} = (y_i)_{i=0}^{N-1}$ of the reverse sound are given by

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

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 1.21 (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 1.22. Let \mathbf{x} be the samples of a digital sound of length N . A new sound \mathbf{y} with noise added can be obtained by adding a random number to each sample,

$$\mathbf{y} = \mathbf{x} + c * (2 * \text{rand}(1, N) - 1);$$

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 $(2 * \text{rand}(1, N) - 1)$ 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 also 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 \mathbf{y} : `castanets.wav` with noise added with $c = 0.4$ sounds like this, while with $c = 0.1$ it sounds like this.

1.4.2 Filtering operations

Later on we will focus on particular operations on sound, where the output is constructed by combining several input elements in a particular way. We say that we *filter* the sound, and we call such operations *filtering operations*, or simply *filters*. Filters are important since they can change the frequency content in a signal in many ways. We will defer the precise definition of filters to Section 3.3, where we also will give the filters listed below a closer mathematical analysis.

Example 1.23 (Adding echo). An echo is a copy of the sound that is delayed and softer than the original sound. We observe that the sample that comes m seconds before sample i has index $i - ms$ where s is the sample rate. This also makes sense even if m 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 ms may not be an integer. We can get round this by rounding ms to the nearest integer which corresponds to adjusting the echo slightly.

Fact 1.24. Let (\mathbf{x}, s) be a digital sound. Then the sound \mathbf{y} with samples given by

$$\mathbf{y} = \mathbf{x}((d+1):N) - c * \mathbf{x}(1:(N-d));$$

will include an echo of the original sound. Here $d = \text{round}(ms)$ is the integer closest to ms , 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 does not work at the beginning of the signal, and that the echo is inaudible if d is too small. You can listen to the sample file with echo added with $d = 10000$ and $c = 0.5$ here.

Example 1.25 (Reducing the treble). 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 $\mathbf{y} = (y_i)_{i=0}^{N-1}$ we can compute it as

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

This is another example of a filtering operation, but this time three input elements are needed in order to produce an output element. Note that the vector $\{1/3, 1/3, 1/3\}$ uniquely describe how the input elements should be combined to produce the output. The elements in this vector are also referred to as the *filter coefficients*. Since this filter is based on forming averages it is also called a *moving average filter*.

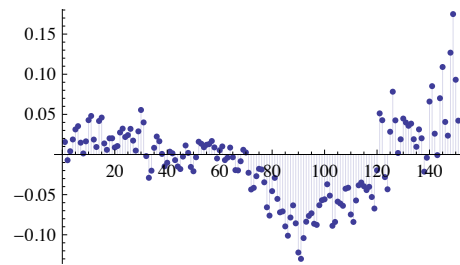
It is reasonable to let the middle sample x_i count more than the neighbours in the average, so an alternative is to compute the average by instead writing

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

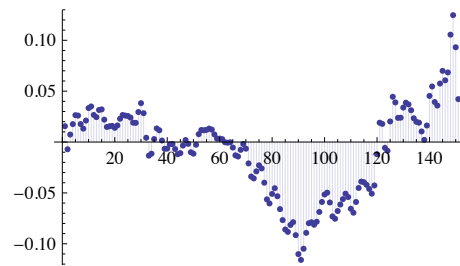
The coefficients 1, 2, 1 here have been taken from row 2 in Pascal's triangle. It will turn out that this is a good choice of coefficients. We have not developed the tools needed to analyse the quality of filters yet, so this will be discussed later. We can also take averages of more numbers, where it will also turn out that row k of Pascal's triangle also is a very good choice. The values in Pascal's triangle can be computed as the coefficients of x in the expression $(1 + x)^k$, which also equal the binomial coefficients $\binom{k}{r}$ for $0 \leq r \leq k$. As an example, if we pick coefficients from row 4 of Pascal's triangle instead, we would write

```
y(1)=x(1); y(2)=x(2);
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
y(N-1)=x(N-1); y(N)=x(N);
```

It will turn out that picking coefficients from a row in Pascal's triangle works better the longer the filter is:



(a) The original sound signal



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

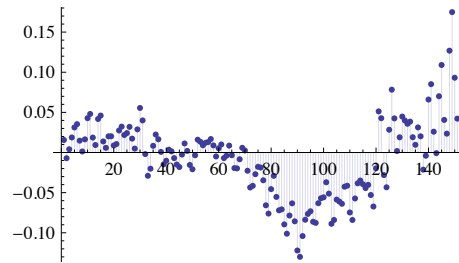
Figure 1.5: Reducing the treble.

Observation 1.26. Let \mathbf{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 \mathbf{y} given by

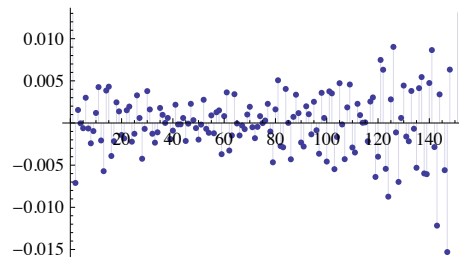
```
y=zeros(length(x));
y(1:k)=x(1:k);
for t=(k+1):(N-k)
    for j=1:(2*k+1)
        y(t)=y(t)+c(j)*x(t+k+1-j))/2^k;
    end
end
y((N-k+1):N)=x((N-k+1):N);
```

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

An example of the result of averaging is shown in Figure 1.5. (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 9 you will be asked to implement reducing the treble in the file `castanets.wav`. 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 $k = 256$



(a) The original sound signal



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

Figure 1.6: Reducing the bass.

it will be like this.

Example 1.27 (Reducing the bass). 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. We can for instance change the filter described for the fourth row in Pascals triangle to

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

An example is shown in Figure 1.6. The original signal 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 1.28. Let \mathbf{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 \mathbf{y} given by

```
y=zeros(length(x));
y(1:k)=x(1:k);
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
y((N-k+1):N)=x((N-k+1):N);

```

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

In Exercise 9 you will be asked to implement reducing the bass in the file `castanets.wav`. 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 $k = 2$.

There are also other operations we would like to perform for digital sound. For instance, it would be nice to adjust a specific set of frequencies only, so that we end up with a sound where unpleasant components of the sound have been removed. Later on we will establish mathematics which will enable us to construct filters which have the properties that they work in desirable ways on any frequencies. It will also turn out that the filters listed above can be given a frequency interpretation: When the input sound has a given frequency, the output sound has the same frequency.

Exercises for Section 1.4

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

Ex. 2 — Define the following sound signal

$$f(t) = \begin{cases} 0 & 0 \leq t \leq 0.2 \\ \frac{t-0.2}{0.4} \sin(2\pi 440t) & 0.2 \leq t \leq 0.6 \\ \sin(2\pi 440t) & 0.6 \leq t \leq 1 \end{cases}$$

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

Ex. 3 — 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.

Ex. 4 — 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?

Ex. 5 — 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 $T = \frac{1}{440}$.

Ex. 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 1.17?

- 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 1.19?

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

Ex. 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 1.21, 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?

Ex. 8 — 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 1.23.

- b. Generate the sound from Example 1.23, 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 distinguishable from the sound itself? How low can you choose c in order to still hear the echo?

Ex. 9 — In this exercise, we will experiment with increasing and reducing the treble and bass in a signal as in examples 1.25 and 1.27.

- a. Write functions

```
function reducetreb(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's 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 1.25 and 1.27, 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?

1.5 Compression of sound and 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 and does not specify the details of how the encoding should be done. 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 samples are transformed using methods we will go through in the next section. A frequency analysis of the sound is the basis for this transformation. Based on this frequency analysis, the sound is split into frequency bands, each band corresponding to a particular frequency range. With MP3, 32 frequency bands are used. Based on the frequency analysis, the encoder uses what is called a *psycho-acoustic model* to compute 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 thresh-*

olds are assigned for each band. The computed masking thresholds have to do with a phenomenon called *masking effects*. 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 omitted by the encoder, since the model says that the sound should be inaudible.

Masking effect is just one example of what is called a psycho-acoustic effect. 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 that 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).

1.6 Summary

We discussed the basic question of what is sound is, and concluded that sound could be modeled as a sum of frequency components. We discussed meaningful operations of sound, such as adjust the bass and treble, adding echo, or adding noise. We also gave an introduction to the MP3 standard for compression of sound.

Chapter 2

Fourier analysis for periodic functions: Fourier series

In Chapter 1 we identified audio signals with functions and discussed informally the idea of decomposing a sound into basis sounds to make its frequency content available. In this chapter we will make this kind of decomposition 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 polynomials are different from polynomials, and the approximations are computed in a very different way. The theory of approximation of functions with trigonometric functions is generally referred to as *Fourier analysis*. This is a central tool in practical fields like image and signal processing, but it also an important field of research within pure mathematics. We will only discuss Fourier analysis for functions defined on a finite interval and for finite sequences (vectors), but Fourier analysis may also be applied to functions defined on the whole real line and to infinite sequences.

Perhaps a bit surprising, linear algebra is a very useful tool in Fourier analysis. This is because the sets of functions involved are vector spaces, both of finite and infinite dimension. Therefore 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.

2.1 Basic concepts

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.

We have already indicated that the functions we consider are defined on an interval, and without much loss of generality we assume this interval to be $[0, T]$, where T is some positive number. Note that any function f defined on $[0, T]$ gives rise to a related function defined on the whole real line, by simply gluing together copies of f . The result is a periodic function with period T that agrees with f on $[0, T]$.

We have to make some more restrictions. Mostly we will assume that f is continuous, but the theory can also be extended to functions which are only Riemann-integrable, more precisely, that the square of the function is integrable.

Definition 2.1 (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 equipped with an inner-product, a generalisation of the so-called dot-product for vectors.

Theorem 2.2. 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 spaces¹, with inner product² defined by

$$\langle f, g \rangle = \frac{1}{T} \int_0^T f(t)g(t) dt, \quad (2.1)$$

and associated norm

$$\|f\| = \sqrt{\frac{1}{T} \int_0^T f(t)^2 dt}. \quad (2.2)$$

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 2.1 and Theorem 2.2 answer the first question above, namely how general do we allow our functions to be. Theorem 2.2 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 $\|f - g\|$, which we recognise as the error³. This projection is therefore also called a best approximation of f from W and is characterised by the fact that the error should be orthogonal to the subspace W , i.e., we should have

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

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

$$g = \sum_{i=1}^m \frac{\langle f, \phi_i \rangle}{\langle \phi_i, \phi_i \rangle} \phi_i. \quad (2.3)$$

The error $\|f - g\|$ in the approximation 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 Chapter 1.

Definition 2.3 (Fourier series). Let $V_{N,T}$ be the subspace of $C[0, T]$ spanned by the set of functions given by

$$\begin{aligned} \mathcal{D}_{N,T} = \{ & 1, \cos(2\pi t/T), \cos(2\pi 2t/T), \dots, \cos(2\pi Nt/T), \\ & \sin(2\pi t/T), \sin(2\pi 2t/T), \dots, \sin(2\pi Nt/T) \}. \end{aligned} \quad (2.4)$$

The space $V_{N,T}$ is called the N 'th order Fourier space. The N th-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 (2.1).

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 Chapter 1, we see that if N is sufficiently large, we get a space which 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 2.3 characterises the Fourier series. The next lemma gives precise expressions for the coefficients.

³See Section 6.3 in [7] for a review of projections and least squares approximations.

Theorem 2.4. 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.

$$f_N(t) = a_0 + \sum_{n=1}^N (a_n \cos(2\pi nt/T) + b_n \sin(2\pi nt/T)). \quad (2.5)$$

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

$$a_0 = \langle f, 1 \rangle = \frac{1}{T} \int_0^T f(t) dt, \quad (2.6)$$

$$a_n = 2 \langle f, \cos(2\pi nt/T) \rangle = \frac{2}{T} \int_0^T f(t) \cos(2\pi nt/T) dt \quad \text{for } n \geq 1, \quad (2.7)$$

$$b_n = 2 \langle f, \sin(2\pi nt/T) \rangle = \frac{2}{T} \int_0^T f(t) \sin(2\pi nt/T) dt \quad \text{for } n \geq 1. \quad (2.8)$$

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 (2.3). 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 (2.3) now gives

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

The relations (2.6)- (2.8) now follow by comparison with (2.5). \square

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 (2.5) is known as the *synthesis equation*, since the original function f is synthesized as a sum of trigonometric functions. Similarly, equations (2.6)- (2.8) are called *analysis equations*.

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 N th-order Fourier series approximation. More precisely, we have the following result for the convergence of the Fourier series, stated without proof.

Theorem 2.5 (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. $\int_0^T |f(t)| dt < \infty$.

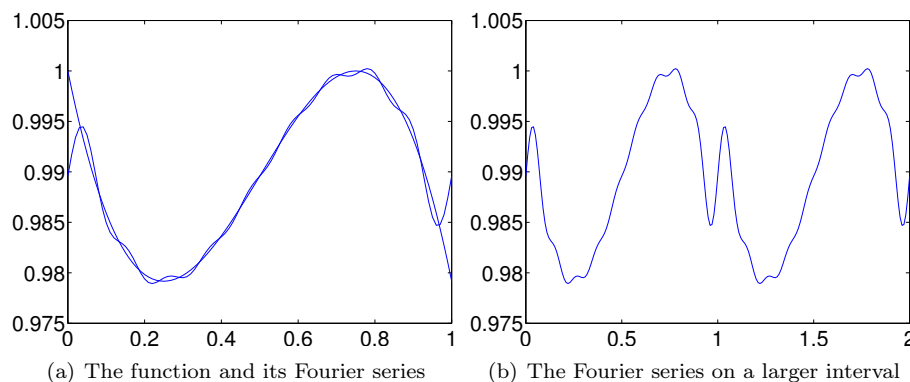


Figure 2.1: The cubic polynomial $f(x) = -\frac{1}{3}x^3 + \frac{1}{2}x^2 - \frac{3}{16}x + 1$ on the interval $[0, 1]$, together with its Fourier series approximation from $V_{9,1}$.

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

The conditions in Theorem 2.5 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, depending on the generality.

An illustration of Theorem 2.5 is shown in Figure 2.1 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 2.1(b), where its periodicity is clearly visible.

Example 2.6. Let us compute the Fourier coefficients of the square wave, as defined by (1.1) in Example 1.11. If we first use (2.6) we obtain

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

Using (2.7) we get

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

Finally, using (2.8) we obtain

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

In other words, only the b_n -coefficients with n odd in the Fourier series are nonzero. From this it is clear that the Fourier series is

$$\frac{4}{\pi} \sin(2\pi t/T) + \frac{4}{3\pi} \sin(2\pi 3t/T) + \frac{4}{5\pi} \sin(2\pi 5t/T) + \frac{4}{7\pi} \sin(2\pi 7t/T) + \dots$$

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 2.2(a) we have plotted the Fourier series of the square wave when $T = 1/440$, and when $N = 20$. In Figure 2.2(b) we have also plotted the values

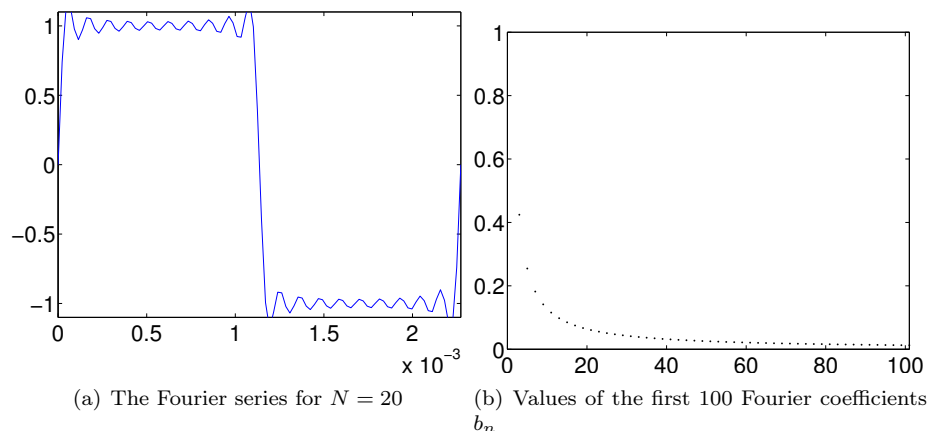


Figure 2.2: The Fourier series of the square wave of Example 2.6

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 2.2(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.

The Fourier series approximations of the square wave can be played with the `play` function, just as the square wave itself. 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 introduction of more frequencies gradually pollutes the sound more and more. In Exercise 7 you will be asked to write a program which verifies this.

Example 2.7. Let us also compute the Fourier coefficients of the triangle wave,

as defined by (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 has as much area above as below the x -axis, so that this integral must be zero. Similarly, since f is symmetric about the midpoint $T/2$, and $\sin(2\pi nt/T)$ is antisymmetric about $T/2$, we have that $f(t) \sin(2\pi nt/T)$ also is antisymmetric about $T/2$, so that

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

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

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

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

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

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

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

In Figure 2.3 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.

We can play different Fourier series approximations of the triangle wave, just as those for the square 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

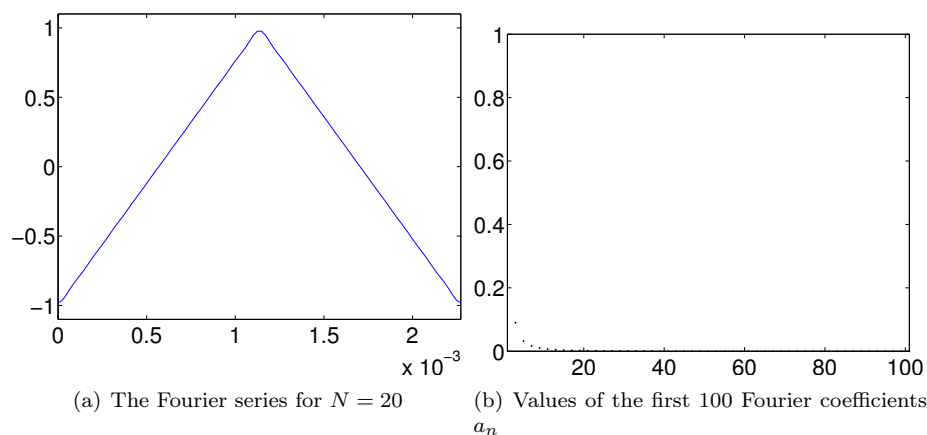


Figure 2.3: The Fourier series of the triangle wave of Example 2.7

sounds like this. Again these sounds are more like the triangle wave itself, and as we increase N we can hear that 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 7 you will be asked to write a program which verifies this.

From the previous examples we understand how we can use the Fourier coefficients to analyse or improve the 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 $\sin 2\pi 440t$, which we started our experiments with.

2.1.1 Fourier series for symmetric and antisymmetric functions

In Example 2.6 we saw that the Fourier coefficients b_n vanished, resulting in a sine-series for the Fourier series. Similarly, in Example 2.7 we saw that a_n vanished, resulting in a cosine-series. 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 2.8 (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

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

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 antisymmetric function by writing

$$f(t) = \frac{f(t) + f(-t)}{2} + \frac{f(t) - f(-t)}{2}. \quad (2.9)$$

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 2.1

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

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

Ex. 3 — Prove the second part of Theorem 2.8, i.e. show that if f is anti-symmetric 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.

Ex. 4 — 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 $[0, T]$.

Ex. 5 — 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 4.

Ex. 6 — 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 2.1.

Ex. 7 — 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 2.6 and 2.7.

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

2.2 Complex Fourier series

In Section 2.1 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 the 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 would like to make the substitutions

$$\cos(2\pi nt/T) = \frac{1}{2} (e^{2\pi int/T} + e^{-2\pi int/T}) \quad (2.10)$$

$$\sin(2\pi nt/T) = \frac{1}{2i} (e^{2\pi int/T} - e^{-2\pi int/T}) \quad (2.11)$$

in Definition 2.3. 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 2.3 as follows:

Definition 2.9 (Complex Fourier basis). We define the set of functions

$$\mathcal{F}_{N,T} = \{e^{-2\pi iNt/T}, e^{-2\pi i(N-1)t/T}, \dots, e^{-2\pi it/T}, \quad (2.12)$$

$$1, e^{2\pi it/T}, \dots, e^{2\pi i(N-1)t/T}, e^{2\pi iNt/T}\}, \quad (2.13)$$

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 for sines and cosines. 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 (2.1). 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:

$$\langle f, g \rangle = \frac{1}{T} \int_0^T f \bar{g} dt. \quad (2.14)$$

The associated norm now becomes

$$\|f\| = \sqrt{\frac{1}{T} \int_0^T |f(t)|^2 dt}. \quad (2.15)$$

The motivation behind Equation 2.14, 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

$$\langle f, g \rangle = \langle g, f \rangle \quad (2.16)$$

is replaced with the axiom

$$\langle f, g \rangle = \overline{\langle g, f \rangle}, \quad (2.17)$$

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 property 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 2.14 satisfies Axiom 2.17. 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 2.4 which uses the complex Fourier basis:

Theorem 2.10. We denote by $y_{-N}, \dots, y_0, \dots, y_N$ the coordinates of f_N in the basis $\mathcal{F}_{N,T}$, i.e.

$$f_N(t) = \sum_{n=-N}^N y_n e^{2\pi int/T}. \quad (2.18)$$

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

$$y_n = \langle f, e^{2\pi int/T} \rangle = \frac{1}{T} \int_0^T f(t) e^{-2\pi int/T} dt. \quad (2.19)$$

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 (2.10)-(2.11) give us that the change of basis matrix⁴ from

⁴See Section 4.7 in [7], to review the mathematics behind change of basis.

$\mathcal{D}_{N,T}$ 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 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 2.11 (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 $n = 1, \dots, N$.

Combining with Theorem 2.8, Theorem 2.11 can help us state properties of complex Fourier coefficients for symmetric- and antisymmetric functions. We look into this in Exercise 8.

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 2.2

Ex. 1 — Show that the complex functions $e^{2\pi i n t/T}$ are orthonormal.

Ex. 2 — Repeat Exercise 2.1.4, computing the complex Fourier series instead of the real Fourier series.

Ex. 3 — Show that both $\cos^n t$ and $\sin^n t$ are in $V_{N,T}$, and find an expression for their complex Fourier coefficients.

Ex. 4 — Consider a sum of two complex exponentials. When is their sum also periodic? What is the fundamental period of the sum if the sum also is periodic?

Ex. 5 — Compute the complex Fourier coefficients of the square wave using Equation 2.19, i.e. repeat the calculations from Example 2.6 for the complex case. Use Theorem 2.11 to verify your result.

Ex. 6 — Repeat Exercise 5 for the triangle wave.

Ex. 7 — Use Equation 2.19 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 2.11 to verify your calculations from Exercise 4.

Ex. 8 — In this exercise we will prove a version of Theorem 2.8 for complex Fourier coefficients.

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

2.3 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 of the Fourier series, given that it converges. If the series converges quickly, we may only need a few terms in the Fourier series to obtain a reasonable approximation, meaning that good Fourier series approximations can be computed quickly. We have already seen examples which illustrate convergence rates that appear to be different: 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 2.12. If the complex Fourier coefficients of f are y_n and f is differentiable, then the Fourier coefficients of $f'(t)$ are $\frac{2\pi i n}{T} y_n$.

Proof. The Fourier coefficients of $f'(t)$ are

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

where the second equation was obtained from integration by parts. \square

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 $(T/(2\pi i n))^s$ 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 2.13. The Fourier series converges quickly when the function is many times differentiable.

An illustration is found in examples 2.6 and 2.7, 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 triangle wave is continuous with a discontinuous first derivative.

Very often, the slow convergence of a Fourier series is due to some discontinuity of (a derivative of) the function at a given point. In this case a strategy to speed up the convergence of the Fourier series could be to create an extension of the function which is continuous, if possible, and use the Fourier series of this new function instead. With the help of the following definition, we will show that this strategy works, at least in cases where there is only one single point of discontinuity (for simplicity we have assumed that the discontinuity is at 0).

Definition 2.14 (Symmetric extension of a function). Let f be a function defined on $[0, T]$. The symmetric extension of f denotes the function \check{f} defined on $[0, 2T]$ by

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

Clearly $\check{f}(0) = \check{f}(2T)$, so when f is continuous, it can be periodically extended to a continuous function with period $2T$, contrary to the function f we started with. Also, \check{f} keeps the characteristics of f , since they are equal on $[0, T]$. Also, \check{f} is clearly a symmetric function, so that it can be expressed as a cosine-series. The Fourier coefficients of the two functions are related.

Theorem 2.15. The complex Fourier coefficients y_n of f , and the cosine-coefficients a_n of \check{f} are related by $a_{2n} = y_n + y_{-n}$.

Proof. The $2n$ th complex Fourier coefficient of \check{f} is

$$\begin{aligned} & \frac{1}{2T} \int_0^{2T} \check{f}(t) e^{-2\pi i 2nt/(2T)} dt \\ &= \frac{1}{2T} \int_0^T f(t) e^{-2\pi i nt/T} dt + \frac{1}{2T} \int_T^{2T} f(2T-t) e^{-2\pi i nt/T} dt. \end{aligned}$$

Substituting $u = 2T - t$ in the second integral we see that this is

$$\begin{aligned} &= \frac{1}{2T} \int_0^T f(t) e^{-2\pi i nt/T} dt - \frac{1}{2T} \int_T^0 f(u) e^{2\pi i nu/T} du \\ &= \frac{1}{2T} \int_0^T f(t) e^{-2\pi i nt/T} dt + \frac{1}{2T} \int_0^T f(t) e^{2\pi i nt/T} dt \\ &= \frac{1}{2} y_n + \frac{1}{2} y_{-n}. \end{aligned}$$

Therefore we have $a_{2n} = y_n + y_{-n}$. □

This result is not enough to obtain the entire Fourier series of \check{f} , but at least it gives us half of it.

Example 2.16. 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 0 to 1. Because f is discontinuous at the boundaries between the periods, we would expect 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 symmetrization of f . Clearly this is the triangle wave with period $2T$, and the Fourier series of this is

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

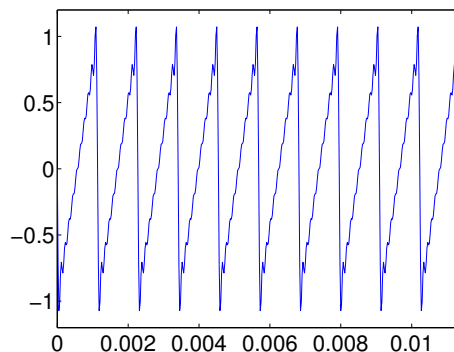


Figure 2.4: The Fourier series for $N = 10$ for the function in Example 2.16

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. Also, we could have obtained half of the second set of coefficients from the first, by using Theorem 2.15.

If we use $T = 1/880$, the symmetrization will be the square wave of Example 2.7. Its Fourier series for $N = 10$ is shown in Figure 2.3(b) and the Fourier series for $N = 20$ is shown in Figure 2.4. 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 of f is not a very good approximation. However, we cannot differentiate between the Fourier series and the function itself for the triangle wave.

2.4 Some properties of Fourier series

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

Theorem 2.17 (Fourier series pairs). The functions 1, $e^{2\pi int/T}$, and $\chi_{-a,a}$

have the Fourier coefficients

$$\begin{aligned} 1 &\rightarrow \mathbf{e}_0 = (1, 0, 0, 0, \dots) \\ e^{2\pi i n t/T} &\rightarrow \mathbf{e}_k = (0, 0, \dots, 1, 0, 0, \dots) \\ \chi_{-a,a} &\rightarrow \frac{\sin(2\pi n a/T)}{\pi n}. \end{aligned}$$

The 1 in \mathbf{e}_k 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 2.18 (Fourier series properties). The mapping $f \rightarrow y_n$ is linear: if $f \rightarrow x_n$, $g \rightarrow y_n$, then

$$af + bg \rightarrow ax_n + by_n$$

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 $f \rightarrow y_n$, then $g \rightarrow \overline{y_n}$. 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 $f \rightarrow y_n$, then $g \rightarrow e^{-2\pi i n d/T} y_n$.
4. If $g(t) = e^{2\pi i d t/T} f(t)$ with d an integer, and $f \rightarrow y_n$, then $g \rightarrow y_{n-d}$.
5. Let d be a number. If $f \rightarrow y_n$, then $f(d + t) = f(d - t)$ for all t if and only if the argument of y_n is $-2\pi n d/T$ 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

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

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

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

Property 2 follows from this, since the remaining statements here were established in Theorems 2.8, 2.11, and Exercise 2.2.8. To prove property 3, we observe that the Fourier coefficients of $g(t) = f(t - d)$ are

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

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

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

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 $e^{-2\pi i n d / T}$ times the (real) Fourier coefficients of g by property 3. It follows that y_n , the Fourier coefficients of f , has argument $-2\pi n d / T$. 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. \square

From this theorem we see that there exist several cases of duality between Fourier coefficients, and the function itself:

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.

Note that these dualities become even more explicit if we consider Fourier series of complex functions, and not just real functions.

Exercises for Section 2.4

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

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

f is just the square wave, shifted with $T/4$. Compute the Fourier coefficients of f directly, and use 3. in Theorem 2.18 to verify your result.

Ex. 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 2.18 on the Fourier series of the square wave.

2.5 Summary

In this chapter we have defined and studied Fourier series, which is an approximation scheme for periodic functions using trigonometric functions. We have 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 investigated a technique for speeding up the convergence of the Fourier series.

Chapter 3

Fourier analysis for vectors

In Chapter 2 we saw how a function defined on an interval can be decomposed into a linear combination of sines and cosines, or equivalently, a linear combination of complex exponential functions. However, this kind of decomposition is not very convenient from a computational point of view. The coefficients are given by integrals that in most cases cannot be evaluated exactly, so some kind of numerical integration technique would have to be applied.

In this chapter our starting point is simply a vector of finite dimension. Our aim is then to decompose this vector in terms of linear combinations of vectors built from complex exponentials. This simply amounts to multiplying the original 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 continuous Fourier series in Chapter 2.

Recall from Chapter 1 that a digital sound is simply a sequence of numbers, in other words, a vector. An algorithm for decomposing a vector into combinations of complex exponentials therefore corresponds to an algorithm for decomposing a digital sound into a combination of pure tones.

3.1 Basic ideas

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

Fact 3.1. A digital sound is a finite sequence (or equivalently a vector) \mathbf{x} of numbers, together with a number (usually an integer) f_s , the sample rate, which denotes the number of measurements of the sound per second. The length of the vector is usually assumed to be N , and it is indexed from 0 to $N - 1$. Sample k is denoted by x_k , i.e.,

$$\mathbf{x} = (x_k)_{k=0}^{N-1}.$$

Note that this indexing convention for vectors is not standard in mathematics and is different from what we have used before. Note in particular that MATLAB indexes vectors from 1, so algorithms given here must be adjusted appropriately.

We also need the standard inner product and norm for complex vectors. At the outset our vectors will have real components, but we are going to perform Fourier analysis with complex exponentials which will often result in complex vectors.

Definition 3.2. For complex vectors of length N the Euclidean inner product is given by

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{k=0}^{N-1} x_k \overline{y_k}. \quad (3.1)$$

The associated norm is

$$\|\mathbf{x}\| = \sqrt{\sum_{k=0}^{N-1} |x_k|^2}. \quad (3.2)$$

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) $\{e^{2\pi i n t/T}\}_{n=0}^N$. This can be generalised to vectors (digital sounds), but then the pure tones must of course also be vectors.

Definition 3.3 (Fourier analysis for vectors). In Fourier analysis of vectors, a vector $\mathbf{x} = (x_0, \dots, x_{N-1})$ is represented as a linear combination of the N vectors

$$\phi_n = \frac{1}{\sqrt{N}} \left(1, e^{2\pi i n/N}, e^{2\pi i 2n/N}, \dots, e^{2\pi i kn/N}, \dots, e^{2\pi i n(N-1)/N} \right).$$

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

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

Lemma 3.4. The normalised complex exponentials $\{\phi_n\}_{n=0}^{N-1}$ 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 ϕ_{n_1} and ϕ_{n_2} is then given by

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

In particular, this orthogonality means that the complex exponentials form a basis. And since we also have $\langle\phi_n, \phi_n\rangle = 1$ it is in fact an orthonormal basis. \square

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.

3.2 The Discrete Fourier Transform

Fourier analysis for finite vectors is focused around mapping a given vector from the standard basis to the Fourier basis, performing some operations on the Fourier representation, and then changing the result back to the standard basis. The Fourier matrix, which represents this change of basis, is therefore of crucial importance, and in this section we study some of its basic properties. We start by defining the Fourier matrix.

Definition 3.5 (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 \mathbf{x} is a vector in \mathbb{R}^N , its coordinates $\mathbf{y} = (y_0, y_1, \dots, y_{N-1})$ relative to the Fourier basis are called the Fourier coefficients of \mathbf{x} , in other words $\mathbf{y} = F_N \mathbf{x}$. The DFT of \mathbf{x} is sometimes denoted by $\hat{\mathbf{x}}$.

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

that the change of coordinates performed by the DFT can be written as

$$\mathbf{x} = y_0\phi_0 + y_1\phi_1 + \cdots + y_{N-1}\phi_{N-1} = (\phi_0 \ \phi_1 \ \cdots \ \phi_{N-1}) \mathbf{y} = F_N^{-1} \mathbf{y}, \quad (3.3)$$

where we have used the inverse of the defining relation $\mathbf{y} = F_N \mathbf{x}$, and that the ϕ_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 (3.3) is also called the synthesis equation.

Let us also find the matrix F_N itself. From Lemma 3.4 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 be 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 (3.1) translates into that the inverse of a complex matrix with orthonormal columns is given by the matrix where the entries are both transposed and conjugated. Let us denote the conjugated transpose of T by T^H , and say that a complex matrix is unitary when $T^{-1} = T^H$. From our discussion it is clear that F_N^{-1} is a unitary matrix, i.e. its inverse, F_N , is its conjugate transpose. Moreover since F_N^{-1} is symmetric, its inverse is in fact just its conjugate,

$$F_N = \overline{F_N^{-1}}.$$

Theorem 3.6. 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 $0 \leq n, k \leq N-1$.

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.

In practical applications of Fourier analysis one typically applies the DFT, performs some operations on the coefficients, and then maps the result back using the inverse Fourier matrix. This inverse transformation is so common that it deserves a name of its own.

Definition 3.7 (IDFT). If $\mathbf{y} \in \mathbb{R}^N$ the vector $\mathbf{x} = (F_N)^H \mathbf{y}$ is referred to as the inverse discrete Fourier transform or (IDFT) of \mathbf{y} .

That \mathbf{y} is the DFT of \mathbf{x} and \mathbf{x} is the IDFT of \mathbf{y} can also be expressed in

component form

$$x_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} y_n e^{2\pi i n k / N}, \quad (3.4)$$

$$y_n = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} x_k e^{-2\pi i n k / N}. \quad (3.5)$$

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 $\mathbf{x} = (F_N)^H \mathbf{y}$ and $\mathbf{y} = F_N \mathbf{x}$.

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

Example 3.8 (DFT on a square wave). Let us attempt to apply the DFT to a signal \mathbf{x} which is 1 on indices close to 0, and 0 elsewhere. Assume that

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

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 and $N-1$, and since \mathbf{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 $\sum_{k=N-L}^{N-1} e^{-2\pi i n k / N} = \sum_{k=-L}^{-1} e^{-2\pi i n k / N}$ (since $e^{-2\pi i n k / N}$ is periodic with period N), the DFT of \mathbf{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 \mathbf{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 3.9. In most cases it is difficult to compute a DFT by hand, due to the entries $e^{-2\pi ink/N}$ 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 3.10 (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 if N is a power of 2, there is a much more efficient algorithm for computing the DFT which we will study in a later 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 2.18. The following theorem sums this up:

Theorem 3.11 (DFT properties). Let \mathbf{x} be a real vector of length N . The DFT has the following properties:

1. $(\hat{\mathbf{x}})_{N-n} = \overline{(\hat{\mathbf{x}})_n}$ for $0 \leq n \leq N-1$.
2. If \mathbf{z} is the vector with the components of \mathbf{x} reversed so that $z_k = x_{N-k}$ for $0 \leq k \leq N-1$, then $\hat{\mathbf{z}} = \overline{\hat{\mathbf{x}}}$. In particular,
 - (a) if $x_k = x_{N-k}$ for all n (so \mathbf{x} is symmetric), then $\hat{\mathbf{x}}$ is a real vector.
 - (b) if $x_k = -x_{N-k}$ for all k (so \mathbf{x} is antisymmetric), then $\hat{\mathbf{x}}$ is a purely imaginary vector.
3. If d is an integer and \mathbf{z} is the vector with components $z_k = x_{k-d}$ (the vector \mathbf{x} with its elements delayed by d), then $(\hat{\mathbf{z}})_n = e^{-2\pi i d n / N} (\hat{\mathbf{x}})_n$.
4. If d is an integer and \mathbf{z} is the vector with components $z_k = e^{2\pi i d k / N} x_k$, then $(\hat{\mathbf{z}})_n = (\hat{\mathbf{x}})_{n-d}$.
5. Let d be a multiple of $1/2$. Then the following are equivalent:
 - (a) $x_{d+k} = x_{d-k}$ for all k so that $d+k$ and $d-k$ are integers (in other words \mathbf{x} is symmetric about d).
 - (b) The argument of $(\hat{\mathbf{x}})_n$ is $-2\pi d n / N$ for all n .

Proof. The methods used in the proof are very similar to those used in the proof of Theorem 2.18. From the definition of the DFT we have

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

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

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

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

To prove property 3 we observe that

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

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

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

Finally, to prove property 5, we note that if d is an integer, the vector \mathbf{z} where \mathbf{x} is delayed by $-d$ samples satisfies the relation $(\hat{\mathbf{z}})_n = e^{2\pi i d n / N} (\hat{\mathbf{x}})_n$ because of property 3. Since \mathbf{z} satisfies $z_n = z_{N-n}$, we have by property 2 that $(\hat{\mathbf{z}})_n$ is real, and it follows that the argument of $(\hat{\mathbf{x}})_n$ is $-2\pi d n / N$. It is straightforward to convince oneself that property 5 also holds when d is not an integer also (i.e., a multiple of $1/2$). \square

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 $y_0, y_1, \dots, y_{N/2}$, since the other coefficients can be obtained by conjugating these.

3.2.1 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 digital, pure tones. In practice, the given vector \mathbf{x} will often be sampled from some real data given by a function $f(t)$. We may then talk about the frequency content of the vector \mathbf{x} and the frequency content of f and ask ourselves how these are related. More precisely, what is the relationship between the Fourier coefficients of f and the DFT of \mathbf{x} ?

In order to study this, assume for simplicity that f is a sum of finitely many frequencies. This means that there exists an M so that f is equal to its Fourier approximation f_M ,

$$f(t) = f_M(t) = \sum_{n=-M}^M z_n e^{2\pi i n t / T}, \quad (3.6)$$

where z_n is given by

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

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 3.12. Suppose f is given by its Fourier series (3.6). Then the total frequency content for the frequency n/T is given by the two coefficients z_n and z_{-n} .

Suppose that the vector \mathbf{x} contains values sampled uniformly from f at N points,

$$x_k = f(kT/N), \quad \text{for } k = 0, 1, \dots, N-1. \quad (3.7)$$

The vector \mathbf{x} can be expressed in terms of its DFT \mathbf{y} as

$$x_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} y_n e^{2\pi i n k / N}. \quad (3.8)$$

If we evaluate f at the sample points we have

$$f(kT/N) = \sum_{n=-M}^M z_n e^{2\pi i n k / N}, \quad (3.9)$$

and a comparison now gives

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

Exploiting the fact that both \mathbf{y} and the complex exponentials are periodic with period N , and assuming that we take N samples with N odd, we can rewrite this as

$$\sum_{n=-M}^M z_n e^{2\pi i n k / N} = \frac{1}{\sqrt{N}} \sum_{n=-(N-1)/2}^{(N-1)/2} y_n e^{2\pi i n k / N}.$$

This is a matrix relation on the form $G\mathbf{z} = H\mathbf{y}/\sqrt{N}$, where

1. G is the $N \times (2M+1)$ -matrix with entries $\frac{1}{\sqrt{N}} e^{2\pi i n k / N}$,
2. H is the $N \times N$ -matrix with entries $\frac{1}{\sqrt{N}} e^{2\pi i n k / N}$.

In Exercise 6 you will be asked to show that $G^H G = I_{2M+1}$, and that $G^H H = \begin{pmatrix} I & \mathbf{0} \end{pmatrix}$, when $N \geq 2M+1$. Thus, if we choose the number of sample points N so that $N \geq 2M+1$, multiplying with G^H on both sides in $G\mathbf{z} = H\mathbf{y}/\sqrt{N}$ gives us that

$$\mathbf{z} = \begin{pmatrix} I & \mathbf{0} \end{pmatrix} \left(\frac{1}{\sqrt{N}} \mathbf{y} \right),$$

i.e. \mathbf{z} consists of the first $2M+1$ elements in \mathbf{y}/\sqrt{N} . Setting $N = 2M+1$ we can summarize this.

Proposition 3.13 (Relation between Fourier coefficients and DFT coefficients). Let f be a Fourier series

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

on the interval $[0, T]$ and let $N = 2M + 1$ be an odd integer. Suppose that \mathbf{x} is sampled from f by

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

and let \mathbf{y} be the DFT of \mathbf{x} . Then $\mathbf{z} = \mathbf{y}/\sqrt{N}$, and the total contribution to f from frequency n/T , where n is an integer in the range $0 \leq n \leq M$, is given by y_n and y_{N-n} .

We also 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

$$e^{-2\pi i L t / T}, \dots, e^{-2\pi i t / T}, 1, e^{2\pi i t / T}, \dots, e^{2\pi i L t / T}$$

in f , i.e. $\sum_{n=-L}^L z_n e^{2\pi i n t / T}$. 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 3.14 (DFT indices for high and low frequencies). When \mathbf{y} is the DFT of \mathbf{x} , the low frequencies in \mathbf{x} correspond to the indices in \mathbf{y} near 0 and N . The high frequencies in \mathbf{x} correspond to the indices in \mathbf{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 3.15 (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 high frequencies by setting the corresponding frequencies to zero, and perform an inverse DFT to recover the sound signal with these frequencies eliminated. With the help of the DFT implementation from Example 3.10, 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. It turns out that 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 $\frac{N}{2} - L, \dots, \frac{N}{2}, \dots, \frac{N}{2} + L$. 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: for each block we compute the frequency representation of the values in that block. But the frequency representation may be different when we take all the samples into consideration. In other words, when we split into blocks, we can't expect that we exactly eliminate all the frequencies in question. 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 in terms of the frequencies in the output.

3.2.2 Interpolation with the DFT

There are two other interesting facets to Theorem 3.13, besides connecting the DFT and the Fourier series: The first has to do with interpolation: The theorem 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 3.16 (Interpolation with the Fourier basis). Let f be a function defined on the interval $[0, T]$, and let \mathbf{x} be the sampled sequence given by

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

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

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

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

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

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

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

3.2.3 Sampling and reconstruction with the DFT

The second interesting facet to Theorem 3.13 has to do with when reconstruction of a function from its sample values is possible. An example of sampling a function is illustrated in Figure 3.1. From Figure 3.1(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 that f is equal to a finite Fourier series. This means that f can be written on the form (3.6), so that the highest frequency in the signal is bounded by M/T . Such functions also have their own name:

Definition 3.17 (Band-limited functions). A function f is said to be *band-limited* if there exists a number ν so that f does not contain frequencies higher than ν .

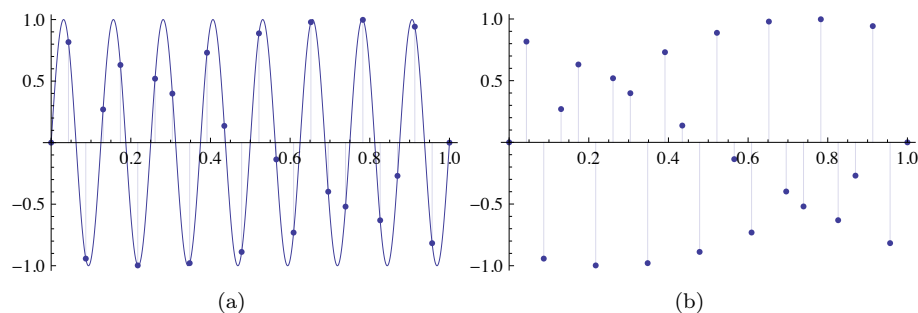


Figure 3.1: An example of sampling. Figure (a) shows how the samples are picked from underlying continuous time function. Figure (b) shows what the samples look like on their own.

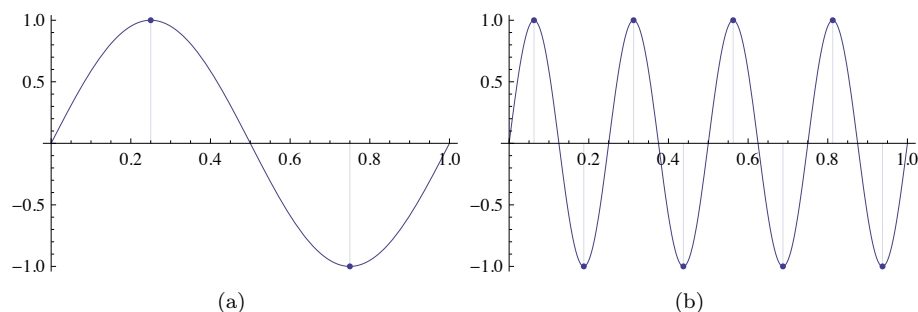


Figure 3.2: Sampling the function $\sin 2\pi t$ with two points, and the function $\sin 2\pi 4t$ with eight points.

Our analysis prior to Theorem 3.13 states that all periodic, band-limited functions can be reconstructed exactly from their samples, using the DFT, as long as the number of samples is $N \geq 2M + 1$, taken uniformly over a period. Moreover, the DFT is central in the reconstruction formula. We say that we reconstruct f from its samples. Dividing by T we get $\frac{N}{T} \geq \frac{2M+1}{T}$, 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 3.2 we try to get some intuition on this by considering some pure tones. In Figure (a) we consider one period of $\sin 2\pi t$, and see that we need at least two sample points in $[0, 1]$, since one point would clearly be too little. This translates directly into having at least eight sample points in Figure (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 without the DFT. The reconstruction

formula was

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

If we here substitute $\mathbf{y} = F_N \mathbf{x}$ we get that this equals

$$\begin{aligned} & \frac{1}{N} \sum_{n=-M}^M \sum_{k=0}^{N-1} x_k e^{-2\pi i n k / N} e^{2\pi i n t / T} \\ &= \sum_{k=0}^{N-1} \frac{1}{N} \left(\sum_{n=-M}^M x_k e^{2\pi i n (t/T - k/N)} \right) \\ &= \sum_{k=0}^{N-1} \frac{1}{N} e^{-2\pi i M (t/T - k/N)} \frac{1 - e^{2\pi i (2M+1)(t/T - k/N)}}{1 - e^{2\pi i (t/T - k/N)}} x_k \\ &= \sum_{k=0}^{N-1} \frac{1}{N} \frac{\sin(\pi(t - kT_s)/T_s)}{\sin(\pi(t - kT_s)/T)} f(kT_s), \end{aligned}$$

where we have substituted $N = T/T_s$ (deduced from $T = NT_s$ with T_s being the sampling period). Let us summarize our findings as follows:

Theorem 3.18 (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 ν Hz. Then f can be reconstructed exactly from its samples $f(0), \dots, f((N-1)T_s)$ (where T_s is the sampling period and $N = \frac{T}{T_s}$ is the number of samples per period) when the sampling rate $F_s = \frac{1}{T_s}$ is bigger than 2ν . Moreover, the reconstruction can be performed through the formula

$$f(t) = \sum_{k=0}^{N-1} f(kT_s) \frac{1}{N} \frac{\sin(\pi(t - kT_s)/T_s)}{\sin(\pi(t - kT_s)/T)}. \quad (3.10)$$

Formula (3.10) 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*. We will return to other interpolation formulas later, which have different properties.

Note that f itself may not be equal to a finite Fourier series, and reconstruction is in general not possible then. Interpolation as performed in Section 3.2.2 is still possible, however, but the $g(t)$ we obtain from Corollary 3.16 may be different from $f(t)$.

Exercises for Section 3.2

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

Ex. 2 — As in Example 3.9, state the exact cartesian form of the Fourier matrix for the cases $N = 6$, $N = 8$, and $N = 12$.

Ex. 3 — Let \mathbf{x} be the vector with entries $x_k = c^k$. Show that the DFT of \mathbf{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 $n = 0, \dots, N - 1$.

Ex. 4 — If \mathbf{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.

Ex. 5 — As in Example 3.10, write a function

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

which computes the IDFT.

Ex. 6 — Let G be the $N \times (2M + 1)$ -matrix with entries $\frac{1}{\sqrt{N}}e^{2\pi ink/N}$, and H the $N \times N$ -matrix with entries $\frac{1}{\sqrt{N}}e^{2\pi ink/N}$. Show that $G^H G = I_{2M+1}$ and that $G^H H = \begin{pmatrix} I & \mathbf{0} \end{pmatrix}$ when $N \geq 2M + 1$. Write also down an expression for $G^H G$ when $N < 2M + 1$, to show that it is in general different from the identity matrix.

3.3 Operations on vectors: filters

In Chapter 1 we defined some operations on digital sounds, which we loosely referred to as filters. One example was the averaging filter

$$z_n = \frac{1}{4}(x_{n-1} + 2x_n + x_{n+1}), \quad \text{for } n = 0, 1, \dots, N - 1 \quad (3.11)$$

of Example 1.25 where \mathbf{x} denotes the input vector and \mathbf{z} the output vector. Before we state the formal definition of filters, let us consider Equation (3.11) 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 \mathbf{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 $\mathbf{x} = (x_0, x_1, \dots, x_{N-1})$ to the vector $\mathbf{z} = (z_0, z_1, \dots, z_{N-1})$, i.e., we have $\mathbf{z} = S\mathbf{x}$. To find S we note that for $1 \leq n \leq N-2$ it is clear from Equation (3.11) 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 averaging filter is given by

$$S = \frac{1}{4} \begin{pmatrix} 2 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \\ 1 & 2 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 2 & 1 \\ 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 2 \end{pmatrix}. \quad (3.12)$$

A matrix on this form is called a Toeplitz matrix. Such matrices are very popular in the literature and have many applications. The general definition may seem complicated, but is in fact quite straightforward:

Definition 3.19 (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).

As the definition says, 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. It is quite easy to check that the matrix S given by Equation (3.12) satisfies Definition 3.19 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 $N/2$ diagonals above or on the main diagonal, and the $N/2$ diagonals below the main diagonal. We will encounter Toeplitz matrices also in other contexts in these notes.

In Chapter 1, the operations we loosely referred to as filters, such as formula (3.11), could all be written on the form

$$z_n = \sum_k t_k x_{n-k}. \quad (3.13)$$

Many other operations are also defined in this way. 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 (3.11) 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 3.20. Any operation defined by Equation (3.13) 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 $\mathbf{x} = (x_0, x_1, \dots, x_{N-1})$ to the vector $\mathbf{z} = (z_0, z_1, \dots, z_{N-1})$, i.e., we have $\mathbf{z} = S\mathbf{x}$. Moreover, the matrix S is a circulant Toeplitz matrix, and the first column \mathbf{s} of this matrix is given by

$$s_k = \begin{cases} t_k, & \text{if } 0 \leq k < N/2; \\ t_{k-N}, & \text{if } N/2 \leq k \leq N-1. \end{cases} \quad (3.14)$$

In other words, the first column of S can be obtained by placing the coefficients in (3.13) with positive indices at the beginning of \mathbf{s} , and the coefficients with negative indices at the end of \mathbf{s} .

This proposition will be useful for us, since it explains how to pass from the form (3.13), which is most common in practice, to the matrix form S .

Example 3.21. Let us apply Proposition 3.20 on the operation defined by formula (3.11):

1. for $k = 0$ Equation 3.14 gives $s_0 = t_0 = 1/2$.
2. For $k = 1$ Equation 3.14 gives $s_1 = t_1 = 1/4$.
3. For $k = N - 1$ Equation 3.14 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 (3.12).

Proposition 3.20 is also useful when we have a circulant Toeplitz matrix S , and we want to find filter coefficients t_k so that $\mathbf{z} = S\mathbf{x}$ can be written as in Equation (3.13):

Example 3.22. 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 (3.14) gives that $t_0 = s_0 = 2$, and $t_1 = s_1 = 3$. The second equation in (3.14) 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

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

3.3.1 Formal definition of filters and frequency response

Let us now define 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 3.15, 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 3.23 (Digital filters and frequency response). A linear transformation $S : \mathbb{R}^N \mapsto \mathbb{R}^N$ is 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

$$S(\phi_n) = \lambda_{S,n}\phi_n, \quad (3.15)$$

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

$$S = F_N^H D F_N \quad (3.16)$$

whenever S is a digital filter, where D has the frequency response (i.e. the eigenvalues) on the diagonal¹. In particular, if S_1 and S_2 are digital filters, we

¹Recall 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 eigenvalues listed on the diagonal (see Section 7.1 in [7]).

can write $S_1 = F_N^H D_1 F_N$ and $S_2 = F_N^H D_2 F_N$, so that

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

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

Corollary 3.24. All digital filters commute, i.e. if S_1 and S_2 are digital filters, $S_1 S_2 = S_2 S_1$.

There are several equivalent characterizations of a digital filter. The first one was stated above in terms of the definition through eigenvectors and eigenvalues. The next characterization helps us prove that the operations from Chapter 1 actually are filters.

Theorem 3.25. 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 $S = (F_N)^H D F_N$ for some diagonal matrix D . We observe that the entry at position (k, l) in S is given by

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

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. \square

In particular, operations defined by (3.13) 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 3.26. Any digital filter is uniquely characterized by the values in the first column of its matrix. Moreover, if \mathbf{s} is the first column in S , the frequency response of S is given by

$$\boldsymbol{\lambda}_S = \sqrt{N} F_N \mathbf{s}. \quad (3.17)$$

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

$$\mathbf{s} = \frac{1}{\sqrt{N}} (F_N)^H \boldsymbol{\lambda}_S. \quad (3.18)$$

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

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

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 (3.17). Equation (3.18) follows directly by applying the inverse DFT to (3.17). \square

Since the first column \mathbf{s} characterizes the filter S uniquely, one often refers to S by the vector \mathbf{s} . The first column \mathbf{s} is also called the *impulse response*. This name stems from the fact that we can write $\mathbf{s} = S \mathbf{e}_0$, i.e., the vector \mathbf{s} is the output (often called response) to the vector \mathbf{e}_0 (often called an impulse).

Example 3.27. The identity matrix is a digital filter since $I = (F_N)^H I F_N$. Since $\mathbf{e}_0 = S \mathbf{e}_0$, it has impulse response $\mathbf{s} = \mathbf{e}_0$. Its frequency response has 1 in all components and therefore preserves all frequencies, as expected.

Equations (3.16), (3.17), and (3.18) 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 3.28. 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 also use the results in this section to compute the eigenvalues and eigenvectors. Since here $N = 2$, we have that $e^{2\pi i n k / N} = e^{\pi i n k} = (-1)^{n k}$. 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

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

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 still an eigenvector). In this case Matlab may 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 3.26 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).

3.3.2 Some properties of the frequency response

Equation (3.17) states that the frequency response can be written as

$$\lambda_{S,n} = \sum_{k=0}^{N-1} s_k e^{-2\pi i n k / N}, \quad \text{for } n = 0, 1, \dots, N-1, \quad (3.19)$$

where s_k are the components of the impulse response \mathbf{s} .

Example 3.29. 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 (3.11). 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

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

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

$$\lambda_S(\omega) = \sum_k t_k e^{-ik\omega}, \quad (3.20)$$

where t_k are the filter coefficients of S , satisfies

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

for any N . In other words, regardless of N , the frequency response lies on the curve λ_S .

Proof. For any N we have that

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

where we have used Equation (3.14). \square

Both $\lambda_S(\omega)$ and $\lambda_{S,n}$ will be referred to as frequency responses in the following. When there is a need to distinguish the two we will call $\lambda_{S,n}$ the *vector frequency response*, and $\lambda_S(\omega)$ the *continuous frequency response*. ω is also 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 stored 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 $N \rightarrow \infty$ (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 3.14 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 3.31. 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

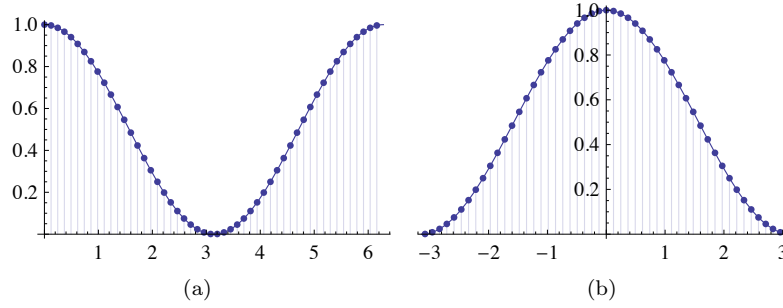


Figure 3.3: The (absolute value of the) frequency response of the smoothing filter in Example 1.25 which we discussed at the beginning of this section.

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 3.32. 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.

Example 3.33. In Example 3.29 we computed the vector frequency response of the filter defined in formula (3.11). 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 3.29. Figure 3.3 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 3.11.

Theorem 3.34. The frequency response has the properties:

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. Moreover, if the frequency response of S is $\lambda_S(\omega)$, then the frequency response of S^T is $\lambda_S(\omega)$.

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. If S_1 and S_2 are digital filters, then $S_1 S_2$ also is a digital filter, and $\lambda_{S_1 S_2}(\omega) = \lambda_{S_1}(\omega) \lambda_{S_2}(\omega)$.

Proof. Property 1. and 3. follow directly from Theorem 3.11. Transposing a matrix corresponds to reversing the first column of the matrix and thus also the filter coefficients. Due to this Property 2. also follows from Theorem 3.11. The last property follows in the same way as we showed that filters commute:

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

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

In particular the frequency response may not be real, although this was the case in the first example of this section. Theorem 3.34 applies both for the vector- and continuous frequency response. Also, clearly $S_1 + S_2$ 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*.

Example 3.35. 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 3.34 to compute the filter coefficients and the matrix of the filter $S = S_1 S_2$. We first notice that, since both frequency responses are real, all S_1 , S_2 , and $S = S_1 S_2$ are symmetric. We rewrite the frequency responses as

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

We now get that

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

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 3.20, 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 .

3.3.3 Assembling the filter matrix and compact notation

Let us return to how we first defined a filter in Equation (3.13):

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

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 \mathbf{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 (3.13) as

$$z_n = \sum_{k=\lfloor N/2 \rfloor - N + 1}^{\lfloor N/2 \rfloor} t_k x_{n-k}. \quad (3.21)$$

The index range in this sum is typically even smaller, since often much less than N of the t_k are nonzero (For Equation (3.11), there were only three nonzero t_k). In such cases one often uses a more compact notation for the filter:

Definition 3.36 (Compact notation for filters). Let $k_{\min} \leq 0$, $k_{\max} \geq 0$ be the smallest and biggest index of a filter coefficient in Equation (3.21) so that $t_k \neq 0$ (if no such values exist, let $k_{\min} = 0$, or $k_{\max} = 0$), i.e.

$$z_n = \sum_{k=k_{\min}}^{k_{\max}} t_k x_{n-k}. \quad (3.22)$$

We will use the following compact notation for S :

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

In other words, the entry with index 0 has been underlined, and only the nonzero t_k 's are listed. By the length of S , denoted $l(S)$, we mean the number $k_{\max} - k_{\min} + 1$.

One seldom writes out the matrix of a filter, but rather uses this compact notation. Note that the length of S can also be written as the number of nonzero filter coefficients minus 1. $l(S)$ thus follows the same convention as the degree of a polynomial: It is 0 if the polynomial is constant (i.e. one nonzero filter coefficient).

Example 3.37. Using the compact notation for a filter, we would write $S = \{1/4, 1/2, 1/4\}$ for the filter given by formula (3.11). For the filter

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

from Example 3.22, we would write $S = \{1, \underline{2}, 3\}$.

Equation (3.13) is also called the convolution of the two vectors \mathbf{t} and \mathbf{x} . Convolution is usually defined without the assumption that the vectors are periodic, and without any assumption on their lengths (i.e. they may be sequences of infinite length):

Definition 3.38 (Convolution of vectors). By the *convolution* of two vectors \mathbf{x} and \mathbf{y} we mean the vector $\mathbf{x} * \mathbf{y}$ defined by

$$(\mathbf{x} * \mathbf{y})_n = \sum_k x_k y_{n-k}. \quad (3.23)$$

In other words, applying a filter S corresponds to convolving the filter coefficients of S with the input. If both \mathbf{x} and \mathbf{y} have infinitely many nonzero entries, the sum is an infinite one, which may diverge. For the filters we look at, we always have a finite number of nonzero entries t_k , so we never have this convergence problem since the sum is a finite one. MATLAB has the built-in function `conv` for convolving two vectors of finite length. This function does not indicate which indices the elements of the returned vector belongs to, however. Exercise 11 explains how one may keep track of these indices.

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 $S\mathbf{x}$. 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 10 investigates this further. For large N we risk running into out of memory situations if we had to form the entire matrix.

3.3.4 Some examples of filters

We have now established the basic theory of filters, so it is time to study some specific examples. Many of the filters below were introduced in Section 1.4.

Example 3.39 (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 \mathbf{e}_d . In the notation above, and when $d > 0$, this filter can also be written as $S = \{\underline{0}, \dots, 1\}$ where the 1 occurs at position d . We observe that

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

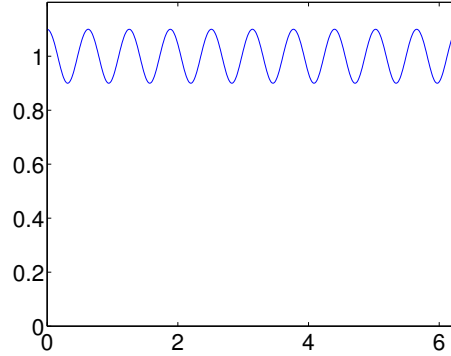


Figure 3.4: The frequency response of a filter which adds an echo with damping factor $c = 0.1$ and delay $d = 10$.

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, in a circular way. 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 3.40 (Adding echo). In Example 1.23 we encountered a filter which could be used for adding echo to sound. Using our compact filter notation this can be written as

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

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 3.4 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 adding exho changes frequencies according to the damping factor c . 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 .

Previously we have claimed that some operations, such as averaging the samples, can be used for adjusting the bass and the treble of a sound. Theorem 3.25 supports this, since the averaging operations we have defined correspond to circulant Toeplitz matrices, which are filters which adjust the frequencies as dictated by the frequency response. Below we will analyze the frequency response of the corresponding filters, to verify that it works as we have claimed for the frequencies corresponding to bass and treble in sound.

Example 3.41 (Reducing the treble). In Example 1.25 we encountered the moving average filter

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

This could be used for reducing the treble in a sound. 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, if the filter is $\mathbf{s} = (1, \dots, 1, \dots, 1)/(2L+1)$, where there is symmetry around 0, we recognize this as $\mathbf{x}/(2L+1)$, where \mathbf{x} is a vector of ones and zeros, as defined in Example 3.8. From that example we recall that

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

so that the frequency response of S is

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

and

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

We clearly have

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

so this frequency response approaches 1 as $\omega \rightarrow 0^+$. 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 smooths the sound by keeping only lower frequencies. In Figure 3.5 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 referred 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 a filter which kept some frequencies unaltered, and annihilated others. This is a desirable property for filters, so let us give names to such filters:

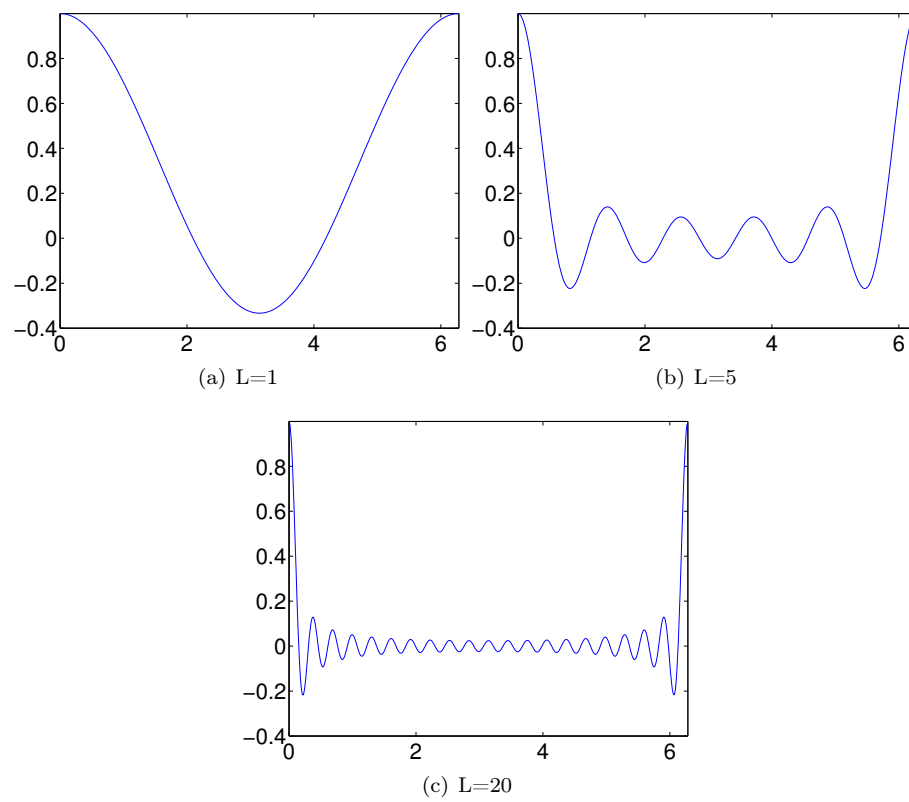


Figure 3.5: The frequency response of moving average filters of different length.

Definition 3.42. A filter S is called

1. a *lowpass filter* if $\lambda_S(\omega) \approx 1$ when ω is close to 0, and $\lambda_S(\omega) \approx 0$ when ω is close to π (i.e. S keeps low frequencies and annihilates high frequencies),
2. a *highpass filter* if $\lambda_S(\omega) \approx 1$ when ω is close to π , and $\lambda_S(\omega) \approx 0$ when ω is close to 0 (i.e. S keeps high frequencies and annihilates low frequencies),
3. a *bandpass filter* if $\lambda_S(\omega) \approx 1$ 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, π ”, and “ $\lambda_S(\omega) \approx 0, 1$ ”: 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 referred 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 3.43 (Ideal lowpass filters). By definition, the ideal lowpass filter keeps frequencies near 0, and removes frequencies near π . In Chapter 1 we mentioned that we were not able to find the filter coefficients for such a filter. We now have the theory in place in order to achieve this: In Example 3.15 we implemented the ideal lowpass filter with the help of the DFT. Mathematically, the code was equivalent to computing $(F_N)^H D F_N$ where D is the diagonal matrix with the entries $0, \dots, L$ and $N - L, \dots, N - 1$ 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 $\omega_c = 2\pi L/N$. In Example 3.8 we computed the DFT of this vector, and it followed from Theorem 3.11 that the IDFT of this vector equals its DFT. This means that we can find the filter coefficients by using Equation (3.18): 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 typically 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 3.44. In order to decrease the computational complexity for the ideal lowpass filter in Example 3.43, 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 response is not that different from the ideal lowpass filter. In Figure 3.6 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.

Example 3.45 (Reducing the treble II). Let S be the moving average filter of two elements, i.e.

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

In Example 3.41 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 3.34 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. At least, this partially explains how filters with coefficients taken from Pascal's triangle appear, as in Example 1.25. These filters are more desirable than the

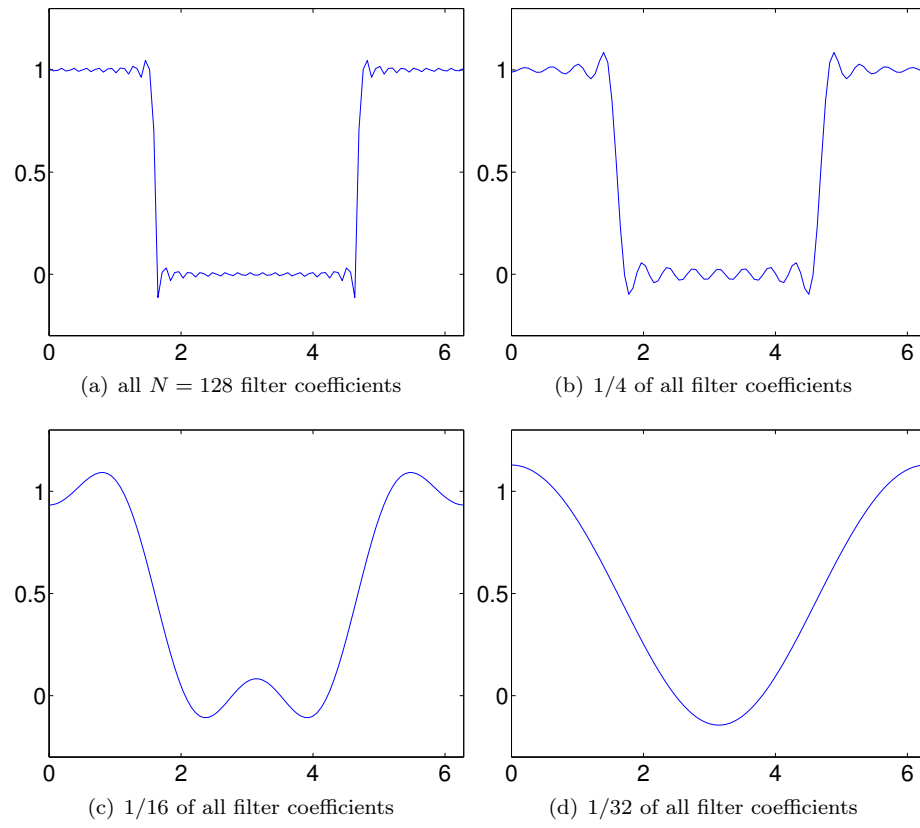


Figure 3.6: The frequency response which results by omitting the last filter coefficients for the ideal lowpass filter.

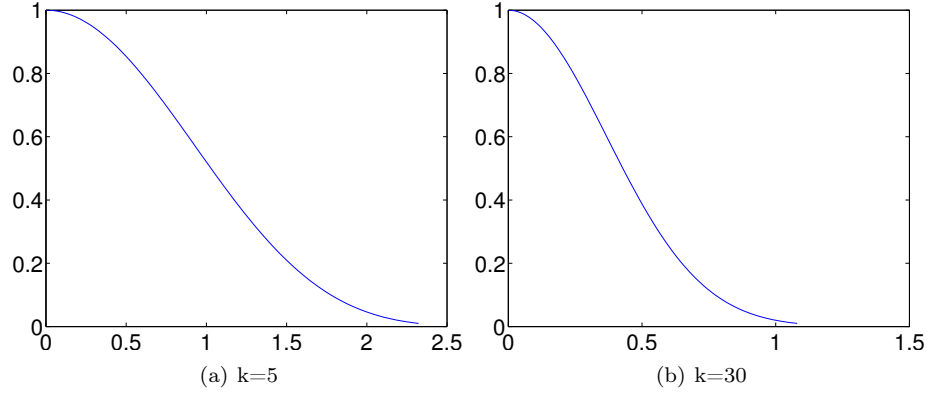


Figure 3.7: The frequency response of filters corresponding to a moving average filter convolved with itself k times.

moving average filters, and are used for smoothing abrupt changes in images and in sound. The reason is that, 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 3.7 we have plotted the magnitude of the frequency 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.

In Example 1.27 we claimed that we could obtain a bass-reducing filter by using alternating signs on the filter coefficients in a treble-reducing filter. 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 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 3.11, with $d = N/2$). Now, for a lowpass filter S , $\lambda_S(\omega)$ has values near 1 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 values near 1 when ω is close to π (the high frequencies). When T is obtained by adding an alternating sign to the filter coefficients 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:

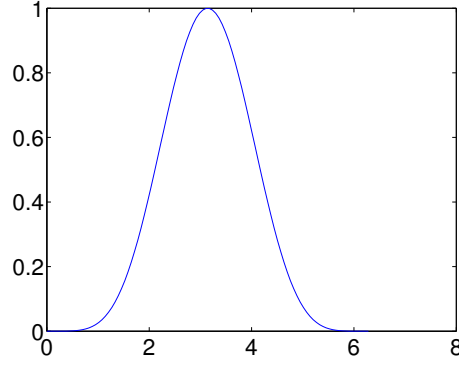


Figure 3.8: The frequency response of the bass reducing filter, which corresponds to row 5 of Pascal's triangle.

Observation 3.46. Assume that T is obtained by adding an alternating sign to the filter coefficients 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 explains why this is the case.

Example 3.47 (Reducing the bass). In Example 1.27 we constructed filters where the rows in Pascal's triangle appeared, but with alternating sign. The frequency response of this when using row 5 of Pascal's triangle is shown in Figure 3.8. 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 3.45 as $\frac{1}{2^k}(1 - e^{-i\omega})^k$, which corresponds to applying the filter $S(\mathbf{x}) = \frac{1}{2}(-x_{n-1} + x_n)$ k times.

3.3.5 Time-invariance of filters

The third characterization of digital filters we will prove is stated in terms of the following concept:

Definition 3.48 (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 \mathbf{z} defined by $z_n = x_{(n-d) \bmod N}$ is the delayed output vector \mathbf{w} defined by $w_n = y_{(n-d) \bmod N}$.

We have the following result:

Theorem 3.49. A linear transformation S is a digital filter if and only if it is time-invariant.

Proof. Let $\mathbf{y} = S\mathbf{x}$, and \mathbf{z}, \mathbf{w} as defined above. We have that

$$\begin{aligned} w_n &= (S\mathbf{x})_{n-d} = \sum_{k=0}^{N-1} S_{n-d,k} x_k \\ &= \sum_{k=0}^{N-1} S_{n,k+d} x_k = \sum_{k=0}^{N-1} S_{n,k} x_{k-d} \\ &= \sum_{k=0}^{N-1} S_{n,k} z_k = (S\mathbf{z})_n \end{aligned}$$

This proves that $S\mathbf{z} = \mathbf{w}$, so that S is time-invariant. \square

By Example 3.39, delaying a vector with d elements corresponds to multiplication with the filter E_d . That S is time-invariant could thus also have been defined by demanding that $SE_d = E_dS$ for any d . That all filters are time invariant follows also immediately from the fact that all filters commute.

Due to Theorem 3.49, digital filters are also called LTI filters (LTI stands for Linear, Time-Invariant). By combining the definition of a digital filter with theorems 3.26, and 3.49, we get the following:

Theorem 3.50 (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.

3.3.6 Linear phase filters

Some filters are particularly important for applications:

Definition 3.51 (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 .

From Theorem 3.11 4. it follows that the argument of the frequency response at n for S is $-2\pi dn/N$. Moreover, the frequency response is real if $d = 0$, and this also corresponds to that the matrix is symmetric. One reason that linear phase filters are important for applications is that they can be more efficiently

implemented than general filters. As an example, if S is symmetric around 0, we can write

$$\begin{aligned}
(S\mathbf{x})_n &= \sum_{k=0}^{N-1} s_k x_{n-k} = \sum_{k=0}^{N/2-1} s_k x_{n-k} + \sum_{k=N/2}^{N-1} s_k x_{n-k} \\
&= \sum_{k=0}^{N/2-1} s_k x_{n-k} + \sum_{k=0}^{N/2-1} s_{k+N/2} x_{n-k-N/2} \\
&= \sum_{k=0}^{N/2-1} s_k x_{n-k} + \sum_{k=0}^{N/2-1} s_{N/2-k} x_{n-k-N/2} \\
&= \sum_{k=0}^{N/2-1} s_k x_{n-k} + \sum_{k=0}^{N/2-1} s_k x_{n+k} = \sum_{k=0}^{N/2-1} s_k (x_{n-k} + x_{n+k})
\end{aligned}$$

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. The same point can also be made about the factorization into a composition of many moving average filters of length 2 in Example 3.45. This also corresponds to a linear phase filter. Each application of a moving average filter of length 2 does not really require any multiplications, since multiplication with $\frac{1}{2}$ really corresponds to a bitshift. Therefore, the factorization of Example 3.45 removes the need for doing any multiplications at all, while keeping the number of additions the same. There is a huge computational saving in this. We will see another desirable property of linear phase filters in the next section, and we will also return to these filters later.

3.3.7 Perfect reconstruction systems

The following is easily proved, and left as exercises:

Theorem 3.52. The following hold:

1. The set of (circulant) Toeplitz matrices form a vector space.
2. If G_1 and G_2 are (circulant) Toeplitz matrices, then $G_1 G_2$ is also a (circulant) Toeplitz matrix, and $l(G_1 G_2) = l(G_1) + l(G_2)$.
3. $l(G) = 0$ if and only if G has only one nonzero diagonal.

An immediate corollary of this is in terms of what is called *perfect reconstruction systems*:

Definition 3.53 (Perfect reconstruction system). By a perfect reconstruction system we mean a pair of $N \times N$ -matrices (G_1, G_2) so that $G_2 G_1 = I$. For a vector \mathbf{x} we refer to $\mathbf{z} = G_1 \mathbf{x}$ as the transformed vector. For a vector $\mathbf{\bar{z}}$ we refer to $\mathbf{\bar{x}} = G_2 \mathbf{\bar{z}}$ as the reconstructed vector.

The terms perfect reconstruction, transformation, and reconstruction come from signal processing, where one thinks of G_1 as a transform, and G_2 as another transform which reconstructs the input to the first transform from its output. In practice, we are interested in finding perfect reconstruction systems where the transformed $G_1 \mathbf{x}$ is so that it is more suitable for further processing, such as compression, or playback in an audio system. One example is the DFT: We have already proved that $(F_N, (F_N)^H)$ is a perfect reconstruction system for ant N . One problem with this system is that the Fourier matrix is not sparse. Although efficient algorithms exist for the DFT, one may find systems which are quicker to compute in the transform and reconstruction steps. We are therefore in practice interested in establishing perfect reconstruction systems, where the involved matrices have particular forms. Digital filters is one such form, since these are quick to compute when there are few nonzero filter coefficients. Unfortunately, related to this we have the following corollary to Theorem 3.52:

Corollary 3.54. let G_1 and G_2 be circulant Toeplitz matrices so that (G_1, G_2) is a perfect reconstruction system. Then there exist a scalar α and an integer d so that $G_1 = \alpha E_d$ and $G_2 = \alpha^{-1} E_{-d}$, i.e. both matrices have only one nonzero diagonal, with the values being inverse of oneanother, and the diagonals being symmetric about the main diagonal.

In short, this states that there do not exist perfect reconstruction systems involving nontrivial digital filters. This sounds very bad, since filters, as we will see, represent some of the nicest operations which can be implemented. Note that, however, it may still be possible to construct such (G_1, G_2) so that $G_1 G_2$ is “close to” I . Such systems can be called “reconstruction systems”, and may be very important in settings where some loss in the transformation process is acceptable. We will not consider such systems here.

In a search for other perfect reconstruction systems than those given by the DFT (and DCT in the next section), we thus have to look for other matrices than those given by digital filters. In Section ?? we will see that it is possible to find such systems for the matrices we define in the next chapter, which are a natural generalization of digital filters.

Exercises for Section 3.3

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

Ex. 2 — Plot the frequency response of the filter $T = \{1/4, -1/2, 1/4\}$. 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?

Ex. 3 — Consider the two filters $S_1 = \{1, 0, \dots, 0, c\}$ and $S_2 = \{1, 0, \dots, 0, -c\}$. 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?

Ex. 4 — In Example 1.19 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 e_k to e_{N-1-k} for all $0 \leq k \leq N-1$.

- Write down the matrix for the time reversal linear mapping, and explain from this why time reversal is not a digital filter.
- Prove directly that time reversal is not a time-invariant operation.

Ex. 5 — Consider the linear mapping S which keeps every second component in \mathbb{R}^N , i.e. $S(e_{2k}) = e_{2k}$, and $S(e_{2k-1}) = \mathbf{0}$. Is S a digital filter?

Ex. 6 — A filter S_1 has the frequency response $\frac{1}{2}(1 + \cos \omega)$, and another filter has the frequency response $\frac{1}{2}(1 + \cos(2\omega))$.

- Is $S_1 S_2$ a lowpass filter, or a highpass filter?
- What does the filter $S_1 S_2$ do with angular frequencies close to $\omega = \pi/2$.
- Find the filter coefficients of $S_1 S_2$.
Hint: Use Theorem 3.34 to compute the frequency response of $S_1 S_2$ first.
- Write down the matrix of the filter $S_1 S_2$ for $N = 8$.

Ex. 7 — 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

- Give a direct argument which uses no computations.
- By using Property 3 in Theorem 3.11, i.e. by using a property for the Discrete Fourier Transform.

Ex. 8 — 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 .

Ex. 9 — Consider again Example 3.43. 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).

Ex. 10 — Assume that S is a circulant Toeplitz matrix so that only

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

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 $y_n = (S\mathbf{x})_n$ can be split into the following different formulas, depending on n , to achieve this:

a. $0 \leq n < E$:

$$y_n = \sum_{k=0}^{n-1} S_{0,N+k-n}x_k + \sum_{k=n}^{F+n} S_{0,k-n}x_k + \sum_{k=N-1-E+n}^{N-1} S_{0,k-n}x_k. \quad (3.24)$$

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

$$y_n = \sum_{k=n-E}^{n+F} S_{0,k-n}x_k. \quad (3.25)$$

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

$$y_n = \sum_{k=0}^{n-(N-F)} S_{0,k-n}x_k + \sum_{k=n-E}^{n-1} S_{0,N+k-n}x_k + \sum_{k=n}^{N-1} S_{0,k-n}x_k. \quad (3.26)$$

These three cases give us the 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.

Ex. 11 — In this exercise we will find out how to keep track of the length and the start and end indices when we convolve two sequences

- Let \mathbf{g} and \mathbf{h} be two sequences with finitely many nonzero elements. Show that $\mathbf{g}*\mathbf{h}$ also has finitely many nonzero elements, and show that $l(\mathbf{g}*\mathbf{h}) = l(\mathbf{g}) + l(\mathbf{h})$.
- Find expressions for the values k_{min}, k_{max} for the filter $\mathbf{g}*\mathbf{h}$, in terms of those for the filters \mathbf{g} and \mathbf{h} .

Ex. 12 — Write a function

```
function [gconvh gconvhmin]=filterimpl(g,gmin,h,hmin)
```

which performs the convolution of two sequences, but also keeps track of the index of the smallest nonzero coefficient in the sequences.

Ex. 13 — Consider the matrix

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

- 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.
- 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.
- 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.

Ex. 14 — 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.

Ex. 15 — 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 matrix for S when $N = 8$. Compute and plot (the magnitude of) $\lambda_S(\omega)$.

Ex. 16 — 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 $\{t_{k_{min}}, \dots, t_{-1}, \underline{t_0}, t_1, \dots, t_{k_{max}}\}$. Compute and plot (the magnitude) of $\lambda_S(\omega)$.

Ex. 17 — Assume that $S = \{1, c, c^2, \dots, c^k\}$. 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?

Ex. 18 — Assume that S_1 and S_2 are two circulant Toeplitz matrices.

- How can you express the eigenvalues of $S_1 + S_2$ in terms of the eigenvalues of S_1 and S_2 ?
- How can you express the eigenvalues of $S_1 S_2$ in terms of the eigenvalues of S_1 and S_2 ?
- 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.?

Ex. 19 — In this exercise we will investigate how we can combine lowpass and highpass filters to produce other filters

- Assume that S_1 and S_2 are lowpass filters. What kind of filter is $S_1 S_2$? What if both S_1 and S_2 are highpass filters?
- Assume that one of S_1, S_2 is a highpass filter, and that the other is a lowpass filter. What kind of filter $S_1 S_2$ in this case?

3.4 Symmetric digital filters and the DCT

We have mentioned that most periodic functions can be approximated well by Fourier series on the form $\sum_n y_n e^{2\pi i n t/T}$. The convergence speed of this Fourier series may be slow however, and we mentioned that it depends on the regularity of the function. In particular, the convergence speed is higher when the function is continuous. For $f \in C[0, T]$ where $f(0) \neq f(T)$, we do not get a continuous function if we repeat the function in periods. However, we demonstrated that we could create a symmetric extension g , defined on $[0, 2T]$, so that $g(0) = g(2T)$. The Fourier series of g actually took the form of a cosine-series, and we saw that this series converged faster to g , than the Fourier series of f did to f .

In this section we will specialize this argument to vectors: by defining the symmetric extension of a vector, we will attempt to find a better approximation than we could with the Fourier basis vectors. This approach is useful for digital filters also, if the filters preserve these symmetric extensions. Let us summarize with the following idea:

Idea 3.55 (Increasing the convergence speed of the DFT). Assume that we have a function f , and that we take the samples $x_k = f(kT/N)$. From $\mathbf{x} \in \mathbb{R}^N$ we would like to create an extension $\check{\mathbf{x}}$ so that the first and last values are equal. For such an extension, the Fourier basis vectors can give a very good approximation to f .

As our candidate for the extension of \mathbf{x} , we will consider the following:

Definition 3.56 (Symmetric extension of a vector). By the symmetric extension of $\mathbf{x} \in \mathbb{R}^N$, we mean $\check{\mathbf{x}} \in \mathbb{R}^{2N}$ defined by

$$\check{\mathbf{x}}_k = \begin{cases} x_k & 0 \leq k < N \\ x_{2N-1-k} & N \leq k < 2N-1 \end{cases} \quad (3.27)$$

We say that a vector in \mathbb{R}^{2N} is symmetric if it can be written as the symmetric extension of a vector in \mathbb{R}^N .

The symmetric extension $\check{\mathbf{x}}$ thus has the original vector \mathbf{x} as its first half, and a copy of \mathbf{x} in reverse order as its second half. Clearly, the first and last values of $\check{\mathbf{x}}$ are equal. In other words, a vector in \mathbb{R}^{2N} is a symmetric extension of a vector in \mathbb{R}^N if and only if it is symmetric about $N - \frac{1}{2}$. Clearly also the set of symmetric extensions is a vector space. Our idea in terms of filters is the following:

Idea 3.57 (Increasing the precision of a digital filter). If a filter maps a symmetric extension of one vector to a symmetric extension of another vector then it is a good approximation of an analog version in terms of Fourier series.

We will therefore be interested in finding filters which preserves symmetric extensions. We will show the following, which characterize such filters:

Theorem 3.58 (Characterization of filters which preserve symmetric extensions). A digital filter S of size $2N$ preserves symmetric extensions if and only if S is symmetric. Moreover, S is uniquely characterized by its restriction to \mathbb{R}^N , denoted by S_r , which is given by $S_1 + (S_2)^f$, where $S = \begin{pmatrix} S_1 & S_2 \\ S_3 & S_4 \end{pmatrix}$, and where $(S_2)^f$ is the matrix S_2 with the columns reversed. Moreover, if we define

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

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

Proof. Let $\mathbf{z} = \hat{\mathbf{x}}$. Since $\check{\mathbf{x}}$ is symmetric about $N - \frac{1}{2}$, by Theorem 3.11 it follows that the argument of z_n is $-2\pi(N - \frac{1}{2})n/(2N)$. Since z_{2N-n} is the conjugate of z_n by the same theorem, it follows that $z_{2N-n} = e^{4\pi i(N - \frac{1}{2})n/(2N)} z_n = e^{-2\pi i n/(2N)} z_n$. It follows that a vector in \mathbb{R}^{2N} is a symmetric extension if and only if its DFT is in the span of the vectors

$$\{\mathbf{e}_0, \{\mathbf{e}_n + e^{-2\pi i n/(2N)} \mathbf{e}_{2N-n}\}_{n=1}^{N-1}\}.$$

These vectors are clearly orthogonal. Their span can also be written as the span of the vectors

$$\left\{ \mathbf{e}_0, \left\{ \frac{1}{\sqrt{2}} \left(e^{\pi i n/(2N)} \mathbf{e}_n + e^{-\pi i n/(2N)} \mathbf{e}_{2N-n} \right) \right\}_{n=1}^{N-1} \right\}, \quad (3.28)$$

where the last vectors have been first multiplied with $e^{\pi i n/(2N)}$, and then normalized so that they have norm 1. Equation (3.28) now gives us an orthonormal basis for the DFT's of all symmetric extensions. Let us map these vectors back with the IDFT. We get first that

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

We also get that

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

Since F_{2N} is unitary, and thus preserves the scalar product, this means that

$$\left\{ \frac{1}{\sqrt{2N}} \cos \left(2\pi \frac{0}{2N} \left(k + \frac{1}{2} \right) \right), \left\{ \frac{1}{\sqrt{N}} \cos \left(2\pi \frac{n}{2N} \left(k + \frac{1}{2} \right) \right) \right\}_{n=1}^{N-1} \right\} \quad (3.29)$$

is an orthonormal basis for the set of symmetric extensions in \mathbb{R}^{2N} . We have

that

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

where we have used that $e^{2\pi i n k/(2N)}$ is an eigenvector of S with eigenvalue $\lambda_{S,n}$, and $e^{-2\pi i n k/(2N)} = e^{2\pi i(2N-n)k/(2N)}$ is an eigenvector of S with eigenvalue $\lambda_{S,2N-n}$. If S preserves symmetric extensions, we see that we must have that $\lambda_{S,n} = \lambda_{S,2N-n}$, and also that the vectors listed in Equation (3.29) must be eigenvectors for S . This is reflected in that the entries in D in the diagonalization $S = (F_{2N})^H D F_{2N}$ are symmetric about the midpoint on the diagonal. From Exercise 3.3.8 we know that this occurs if and only if S is symmetric, which proves the first part of the theorem.

Since S preserves symmetric extensions it is clearly characterized by its restriction to the first N elements. With $S = \begin{pmatrix} S_1 & S_2 \\ S_3 & S_4 \end{pmatrix}$, we compute this restriction as

$$\begin{pmatrix} y_0 \\ \vdots \\ y_{N-1} \end{pmatrix} = \begin{pmatrix} S_1 & S_2 \end{pmatrix} \begin{pmatrix} x_0 \\ \vdots \\ \frac{x_{N-1}}{x_N} \\ \vdots \\ x_{2N-1} \end{pmatrix} = \begin{pmatrix} S_1 + (S_2)^f \end{pmatrix} \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix}.$$

S_2 contains the circulant part of the matrix, and forming $(S_2)^f$ means that the circulant parts switch corners. This shows that S is uniquely characterized by the matrix S_r as defined in the text of the theorem. Finally, since (3.29) are eigenvectors of S , the vectors in \mathbb{R}^N restricted to their first N elements are eigenvectors for S_r . Since the scalar product of two symmetric extensions is the double of the scalar product of the first half of the vectors, we have that these vectors must also be orthogonal, and that

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

is an orthonormal basis of eigenvectors for S_r . We see that we now can define $d_{n,N}$ and the vectors \mathbf{d}_n as in the text of the theorem, and this completes the proof. \square

From the proof we 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:

Corollary 3.59.

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

form an orthonormal basis for the set of all symmetric vectors in \mathbb{R}^{2N} .

Note also that S_r is not a circulant matrix. Therefore, its eigenvectors are not pure tones. An example should clarify this:

Example 3.60. Consider the averaging filter $\mathbf{g} = \{\frac{1}{4}, \frac{1}{2}, \frac{1}{4}\}$. 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

$$S_r = S_1 + (S_2)^f = \begin{pmatrix} \frac{3}{4} & \frac{1}{4} & 0 & 0 \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 \\ 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ 0 & 0 & \frac{1}{4} & \frac{3}{4} \end{pmatrix}.$$

The orthogonal basis we have found is given its own name:

Definition 3.61 (DCT basis). We denote by \mathcal{D}_N the orthogonal basis $\{\mathbf{d}_0, \mathbf{d}_1, \dots, \mathbf{d}_{N-1}\}$. 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 3.62 (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 \mathbf{x} is a vector in \mathbb{R}^N , its coordinates $\mathbf{y} = (y_0, y_1, \dots, y_{N-1})$ relative to the DCT basis are called the DCT coefficients of \mathbf{x} (in other words, $\mathbf{y} = D_N \mathbf{x}$).

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

$$\mathbf{x} = y_0 \mathbf{d}_0 + y_1 \mathbf{d}_1 + \dots + y_{N-1} \mathbf{d}_{N-1} \quad (3.30)$$

in the same way as for the DFT. Following the same reasoning as for the DFT, D_N^{-1} is the matrix where the \mathbf{d}_n are columns. But since these vectors are real and orthonormal, D_N must be the matrix where the \mathbf{d}_n are rows. Moreover, since Theorem 3.58 also states that the same vectors are eigenvectors for filters which preserve symmetric extensions, we can state the following:

Theorem 3.63. D_N is the orthogonal matrix where the rows are \mathbf{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 3.64 (IDCT). We will call $\mathbf{x} = (D_N)^T \mathbf{y}$ the inverse DCT or (IDCT) of \mathbf{x} .

Example 3.65. As with Example 3.9, 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 3.9 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 3.9 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 3.66. 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.

With the DCT one constructs a vector twice as long. One might think due to this that one actually use matrices twice the size. This is, however, avoided since D_N has the same dimensions as F_N , and we will see shortly that the same algorithms as for the DFT also can be used for the DCT. By construction we also see that it is easy to express the N -point DCT in terms of the $2N$ -point DFT. Let us write down this connection:

1. Write $e_0 = \sqrt{2}$ and $e_n = e^{\pi i n / (2N)}$ for $n \neq 0$, and define E as the diagonal matrix with the values e_0, e_1, \dots on the diagonal.
2. Let B be the $2N \times N$ -matrix which is nonzero only when $i = j$ or $i + j = 2N - 1$, and 1 in all these places.
3. Let also A be the $N \times 2N$ -matrix with 1 on the diagonal.

We can now write

$$D_N = E^{-1} A F_{2N} B. \quad (3.31)$$

A here extracts the first rows of the matrix, E^{-1} eliminates the complex coefficients, while B adds columns symmetrically. This factorization enables us to use the efficient FFT-implementation, since the matrices A, B, E all are sparse. We will, however, find an even more efficient implementation of the DCT, which will avoid computing a DFT of twice the size as here.

Similarly to Theorem 3.16 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 3.67 (Interpolation with the DCT basis). Let f be a function defined on the interval $[0, T]$, and let \mathbf{x} be the sampled vector given by

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

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

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

and its coefficients are determined by $\mathbf{y} = D_N \mathbf{x}$.

The proof for 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

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

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

Note the subtle change in the sample points of these cosine functions, 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 3.9 we have plotted the sinusoids of Theorem 3.67 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 $\mathbf{y} = D_N \mathbf{x}$ and $\mathbf{x} = (D_N)^T \mathbf{y}$ 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 $\mathbf{y} = D_N \mathbf{x}$, we can analyse the frequency content of \mathbf{x} . If we want to reduce the bass we can decrease the \mathbf{y} -values with small indices and if we want to increase the treble we can increase the \mathbf{y} -values with large indices.

3.4.1 Other types of symmetric extensions

Note that our definition of symmetric extension duplicates the values x_0 and x_{N-1} : both are repeated when creating the symmetric extension. This is in fact unnecessary when we are creating a longer vector which has equal first and last values, and is primarily motivated from existing efficient implementations for the DCT when all vector lengths are powers of 2. When an efficient DCT implementation is not important, we can change the definition of the symmetric extension as follows (it is this type of symmetric extension we will use later):

Definition 3.68 (Symmetric extension of a vector). By the symmetric extension of $\mathbf{x} \in \mathbb{R}^N$, we mean $\check{\mathbf{x}} \in \mathbb{R}^{2N-2}$ defined by

$$\check{\mathbf{x}}_k = \begin{cases} x_k & 0 \leq k < N \\ x_{2N-2-k} & N \leq k < 2N-2 \end{cases} \quad (3.32)$$

In other words, a vector in \mathbb{R}^{2N} is a symmetric extension of a vector in \mathbb{R}^N if and only if it is symmetric about $N-1$. Theorem 3.58 now instead takes the following form:

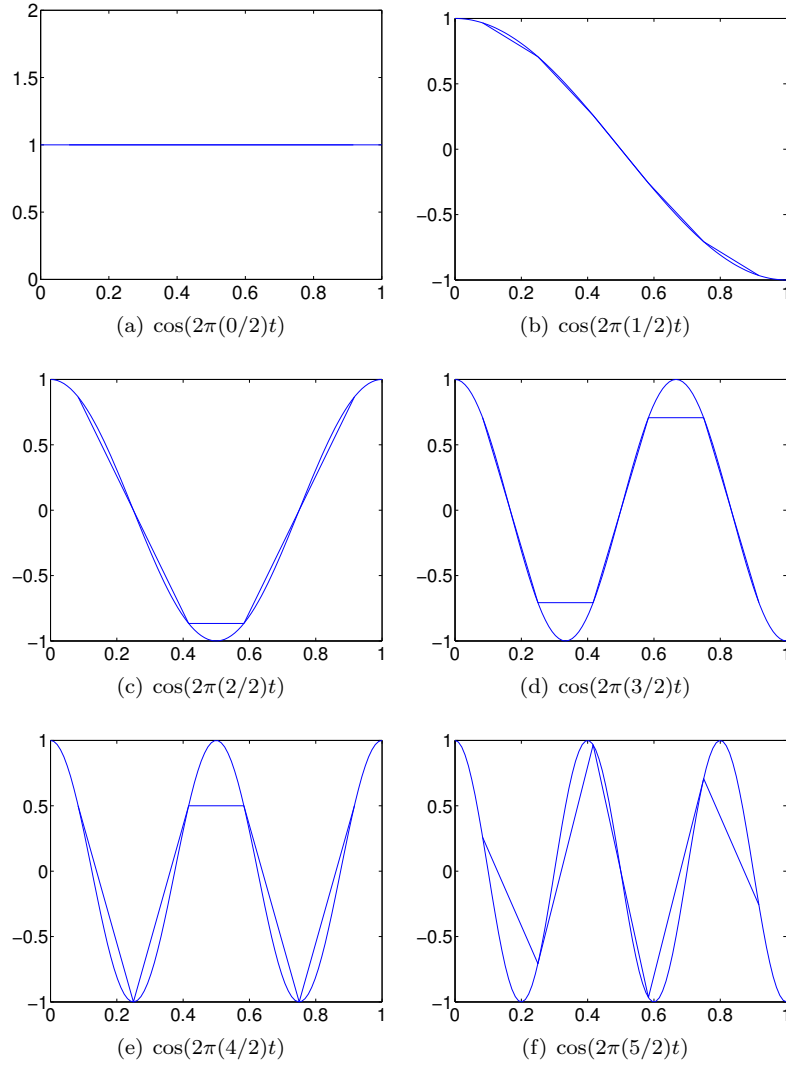


Figure 3.9: The 6 different sinusoids used in DCT for $N = 6$, i.e. $\cos(2\pi(n/2)t)$, $0 \leq n < 6$. The plots also show piecewise linear functions between the the sample points $\frac{2k+1}{2N}$ $0 \leq k < 6$, since only the values at these points are used in Theorem 3.67.

Theorem 3.69 (Characterization of filters which preserve symmetric extensions). A real, circulant $(2N - 2) \times (2N - 2)$ -Toeplitz matrix preserves symmetric extensions if and only if it is symmetric. For such S , S is uniquely characterized by its restriction to \mathbb{R}^N , denoted by S_r , which is given by $T_1 + \begin{pmatrix} 0 & (T_2)^f & 0 \end{pmatrix}$, where $T = \begin{pmatrix} T_1 & T_2 \\ T_3 & T_4 \end{pmatrix}$, where T_1 is $N \times N$, T_2 is $N \times (N - 2)$. Moreover, an orthogonal basis of eigenvectors for S_r are $\{\cos(2\pi \frac{n}{2N} n)\}_{n=0}^{N-1}$.

Proof. Let $z = \hat{x}$. Since \check{x} is symmetric about 0, by Theorem 3.11 it follows that $z_n = z_{2(N-1)-n}$, so that the DFT of a symmetric extension (as now defined) is in the span of the vectors

$$\{e_0, \{e_n + e_{2(N-1)-n}\}_{n=1}^{N-1}\}.$$

It follows as before that

$$\cos\left(2\pi \frac{n}{2(N-1)} n\right)$$

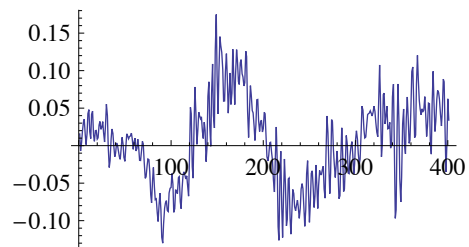
is a basis of eigenvectors. The same type of symmetry about the midpoint on the diagonal follows as before, which as before is equivalent to symmetry of the matrix. \square

It is not customary to write down an orthonormal basis for the eigenvectors in this case, since we don't have the same type of efficient DCT implementation due to the lack of powers of 2.

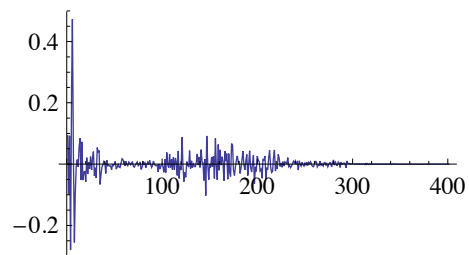
3.4.2 Use of DCT in lossy compression of sound

The DCT is particularly popular for processing the sound before compression. 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 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.

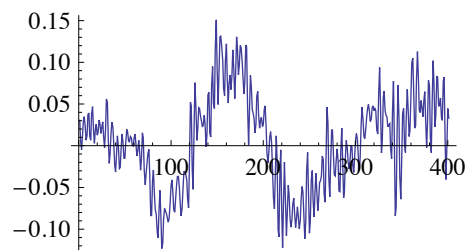
We mentioned previously that we could achieve compression by setting the Fourier coefficients which are small to zero. Translated to the DCT, we should set the DCT coefficients which are small to zero, and we apply the inverse DCT in order to reconstruct the signal in order to play it again. Let us test compression based on this idea. The plots in figure 3.10 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 of theorem ??; 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).



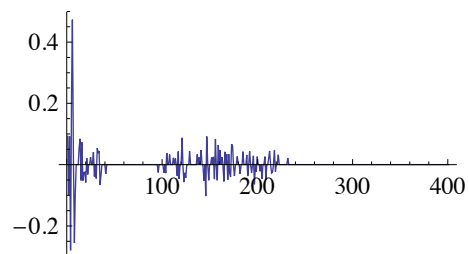
(a)



(b)



(c)



(d)

Figure 3.10: 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.

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=invdct(B);
```

and play the new samples. Finally we can store the signal by storing a `gzip`'ed version of the DCT-values (as 32-bit floating-point numbers) of the 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 much to do with that the DCT, just as the DFT, has a very efficient implementation, as we will see next.

Exercises for Section 3.4

Ex. 1 — In Section 3.4.2 we implemented the function `skipsmallvals`, which ran a DCT on the entire vector. Explain why there are less computation involved in splitting the vector into many parts and performing a DCT for each part. Change the code accordingly.

Ex. 2 — As in Example 3.65, state the exact cartesian form of the DCT matrix for the case $N = 3$.

Ex. 3 — Assume that S is a symmetric digital filter with support $[-E, E]$. Let us, as in Exercise 3.3.10, see how we can make sure that the indices keep inside $[0, N - 1]$. Show that $z_n = (T\mathbf{x})_n$ in this case can be split into the following different formulas, depending on n :

a. $0 \leq n < E$:

$$z_n = T_{0,0}x_n + \sum_{k=1}^n T_{0,k}(x_{n+k} + x_{n-k}) + \sum_{k=n+1}^E T_{0,k}(x_{n+k} + x_{n-k+N}). \quad (3.33)$$

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

$$z_n = T_{0,0}x_n + \sum_{k=1}^E T_{0,k}(x_{n+k} + x_{n-k}). \quad (3.34)$$

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

$$z_n = T_{0,0}x_n + \sum_{k=1}^{N-1-n} T_{0,k}(x_{n+k} + x_{n-k}) + \sum_{k=N-1-n+1}^E T_{0,k}(x_{n+k-N} + x_{n-k}). \quad (3.35)$$

Ex. 4 — Assume that $\{T_{0,-E}, \dots, T_{0,0}, \dots, T_{0,E}\}$ are the coefficients of a symmetric, digital filter S , and let $\mathbf{t} = \{T_{0,1}, \dots, T_{0,E}\}$. Write a function

```
function z=filterT(t,x)
```

which takes the vector \mathbf{t} as input, and returns $\mathbf{z} = T\mathbf{x}$ using the formulas deduced in Exercise 3.

Ex. 5 — Repeat Exercise 1.4.9 by reimplementing the functions `reducetreble` and `reducesbass` using the function `filterT` 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.

Ex. 6 — Using Python, define a class `Transform` with methods `transformImpl` and `itransformImpl`. Define two subclasses of `Transform`, `DCTTransform`, `FFTTransform`, which implements these two functions by calling Python counterparts of `FFTImpl`, `IFFTImpl`, `DCTImpl`, and `IDCTImpl`.

3.5 Summary

We defined the Discrete Fourier transform, which could be thought of as the Fourier series of a vector. We exploited properties of the DFT, which corresponded nicely to the corresponding properties for Fourier series. We defined digital filters, which turned out to be linear transformations diagonalized by the DFT. Also we showed that the techniques from the last section we used to speed up the convergence of the Fourier series, could also be used for the DFT. In this way we arrived at the definition of the DCT.

Chapter 4

Implementation of the DFT and the DCT

The main application of the DFT and the DCT is as tools 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 chapter.

4.1 The Fast Fourier Transform (FFT)

In this section we will discuss the most widely used implementation of the DFT, which is 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 4.1 (FFT algorithm). Let $\mathbf{y} = F_N \mathbf{x}$ be the N -point DFT of \mathbf{x} with N an even number. For any integer n in the interval $[0, N/2 - 1]$ the DFT \mathbf{y} of \mathbf{x} is then given by

$$y_n = \frac{1}{\sqrt{2}} \left((F_{N/2} \mathbf{x}^{(e)})_n + e^{-2\pi i n/N} (F_{N/2} \mathbf{x}^{(o)})_n \right), \quad (4.1)$$

$$y_{N/2+n} = \frac{1}{\sqrt{2}} \left((F_{N/2} \mathbf{x}^{(e)})_n - e^{-2\pi i n/N} (F_{N/2} \mathbf{x}^{(o)})_n \right), \quad (4.2)$$

where $\mathbf{x}^{(e)}, \mathbf{x}^{(o)}$ are the sequences of length $N/2$ consisting of the even and

odd samples of \mathbf{x} , respectively. In other words,

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

Put differently, the formulas (4.1)–(4.2) reduces the computation of an N -point DFT to 2 $N/2$ -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 $0 \leq n \leq N/2 - 1$. We start by splitting the sum in the expression for the DFT into even and odd indices,

$$\begin{aligned}y_n &= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} x_k e^{-2\pi i n k / N} \\ &= \frac{1}{\sqrt{N}} \sum_{k=0}^{N/2-1} x_{2k} e^{-2\pi i n 2k / N} + \frac{1}{\sqrt{N}} \sum_{k=0}^{N/2-1} x_{2k+1} e^{-2\pi i n (2k+1) / N} \\ &= \frac{1}{\sqrt{2}} \frac{1}{\sqrt{N/2}} \sum_{k=0}^{N/2-1} x_{2k} e^{-2\pi i n k / (N/2)} \\ &\quad + e^{-2\pi i n / N} \frac{1}{\sqrt{2}} \frac{1}{\sqrt{N/2}} \sum_{k=0}^{N/2-1} x_{2k+1} e^{-2\pi i n k / (N/2)} \\ &= \frac{1}{\sqrt{2}} \left(F_{N/2} \mathbf{x}^{(e)} \right)_n + \frac{1}{\sqrt{2}} e^{-2\pi i n / N} \left(F_{N/2} \mathbf{x}^{(o)} \right)_n,\end{aligned}$$

where we have substituted $\mathbf{x}^{(e)}$ and $\mathbf{x}^{(o)}$ 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

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

This concludes the proof. \square

It turns out that Theorem 4.1 can be interpreted as a matrix factorization. For this we need to define the concept of a block matrix.

Definition 4.2. Let m_0, \dots, m_{r-1} and n_0, \dots, n_{s-1} be integers, and let $A^{(i,j)}$ be an $m_i \times n_j$ -matrix for $i = 0, \dots, r-1$ and $j = 0, \dots, s-1$. The notation

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

denotes the $(m_0 + m_1 + \dots + m_{r-1}) \times (n_0 + n_1 + \dots + n_{s-1})$ -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 4.3. 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 \mathbf{x} be a column vector. The mapping $\mathbf{x} \rightarrow P\mathbf{x}$ permutes the components of \mathbf{x} so that the even components are placed first and the odd components last,

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

with $\mathbf{x}^{(e)}$, $\mathbf{x}^{(o)}$ defined as in Theorem 4.1.

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 $n = 0, 1, \dots, N/2 - 1$. It is clear from Equation (4.1) that the first half of \mathbf{y} is then given by obtained as

$$\frac{1}{\sqrt{2}} \left(F_{N/2} \mid D_{N/2} F_{N/2} \right) P_N \mathbf{x},$$

and from Equation (4.2) that the second half of \mathbf{y} can be obtained as

$$\frac{1}{\sqrt{2}} \left(F_{N/2} \mid -D_{N/2} F_{N/2} \right) P_N \mathbf{x}.$$

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

Theorem 4.4 (DFT matrix factorization). The Fourier matrix may be factored as

$$F_N = \frac{1}{\sqrt{2}} \left(\begin{array}{c|c} F_{N/2} & D_{N/2} F_{N/2} \\ \hline F_{N/2} & -D_{N/2} F_{N/2} \end{array} \right) P_N. \quad (4.3)$$

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 \mathbf{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

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

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

In an implementation based on formula (4.3), we would first compute $P_N \mathbf{x}$, which corresponds to splitting \mathbf{x} into the even-indexed and odd-indexed samples. The two leftmost blocks in the block matrix in (4.3) 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 by 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 for $N = 4$, say.

4.1.1 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 4.1 and (4.3).

Theorem 4.5 (IDFT matrix factorization). The inverse of the Fourier matrix can be factored as

$$(F_N)^H = \frac{1}{\sqrt{2}} \left(\frac{(F_{N/2})^H}{(F_{N/2})^H} \middle| \frac{E_{N/2}(F_{N/2})^H}{-E_{N/2}(F_{N/2})^H} \right) P_N, \quad (4.4)$$

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

We note that the only difference between the factored forms of F_N and F_N^H is the positive exponent in $e^{2\pi i n/N}$. 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, called `fft` and `ifft`. Note, however, that these functions do not use the normalization $1/\sqrt{N}$ 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.

4.1.2 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 4.6 (Order of an algorithm). Let R_N be the number of operations of a given type (such as multiplication, 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 , which is written $O(f(N))$, if the number of operations

grows as $f(N)$ for large N , or more precisely, if

$$\lim_{N \rightarrow \infty} \frac{R_N}{f(N)} = c > 0.$$

We will also use this notation for functions, and say that a real function g is $O(f(\mathbf{x}))$ if $\lim g(\mathbf{x})/f(\mathbf{x}) = 0$ where the limit mostly will be taken as $\mathbf{x} \rightarrow \mathbf{0}$ (this means that $g(\mathbf{x})$ is much smaller than $f(\mathbf{x})$ when \mathbf{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 4.1. It is clear from the algorithm that

$$M_N = 2M_{N/2} + N/2. \quad (4.5)$$

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 factor $2\pi i/N$ may be computed once and for all outside the loops. We have not counted the multiplications with $1/\text{sqrt}(2)$. 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 multiplying 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

$$M_N = 2M_{N/2} + 2N. \quad (4.6)$$

In Exercise 1 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 4.7 (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.

Since filters are diagonalized by the DFT, it may be tempting to implement a filter by applying an FFT, multiplying with the frequency response, and then apply the IFFT. This is not usually done, however. The reason is that most filters have too few nonzero coefficients for this approach to be efficient — it is then better to use the direct algorithm for the DFT, since this may lead to fewer multiplications than the $O(N \log_2 N)$ required by the FFT.

Exercises for Section 4.1

Ex. 1 — 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 (4.6) 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).

Ex. 2 — 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 writing down this difference equation? Do you have a suggestion to why these multiplications were not counted?

Ex. 3 — Write down a difference equation for computing the number of real additions required by the FFT algorithm.

Ex. 4 — 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:

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

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

- b. Find similar formulas for computing $y_{N/3}, y_{N/3+1}, \dots, y_{2N/3-1}$, and $y_{2N/3}, y_{2N/3+1}, \dots, y_{N-1}$. State a similar factorization of the DFT matrix as in Theorem 4.4, 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.

4.2 Efficient implementations of the DCT

In the preceding section we defined the DCT by expressing it in terms of the DFT. In particular, we can apply efficient implementations of the DFT, which we will shortly look at. However, the way we have defined the DCT, there is a penalty in that we need to compute a DFT of twice the length. 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 4.8 (DCT algorithm). Let $\mathbf{y} = D_N \mathbf{x}$ be the N -point DCT of the vector \mathbf{x} . Then we have that

$$y_n = c_{n,N} \left(\cos \left(\pi \frac{n}{2N} \right) \Re((F_N \mathbf{x}^{(1)})_n) + \sin \left(\pi \frac{n}{2N} \right) \Im((F_N \mathbf{x}^{(1)})_n) \right), \quad (4.7)$$

where $c_{0,N} = 1$ and $c_{n,N} = \sqrt{2}$ for $n \geq 1$, and where $\mathbf{x}^{(1)} \in \mathbb{R}^N$ is defined by

$$\begin{aligned} (\mathbf{x}^{(1)})_k &= x_{2k} \text{ for } 0 \leq k \leq N/2 - 1 \\ (\mathbf{x}^{(1)})_{N-k-1} &= x_{2k+1} \text{ for } 0 \leq k \leq N/2 - 1, \end{aligned}$$

Proof. The N -point DCT of \mathbf{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

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

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

$\mathbf{x}^{(1)}$ as in the text of the theorem, we see that we can write

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

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. \square

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 $Q_{n,n} = c_{n,N} e^{-\pi i n / (2N)}$, the result can also be written on the more compact form

$$\mathbf{y} = D_N \mathbf{x} = \Re \left(Q_N F_N \mathbf{x}^{(1)} \right).$$

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 4.1:

```

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 $\mathbf{x}^{(1)}$ 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).

4.2.1 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 4.8, for $n \geq 1$

$$\begin{aligned} y_n &= c_{n,N} \left(\cos\left(\pi \frac{n}{2N}\right) \Re((F_N \mathbf{x}^{(1)})_n) + \sin\left(\pi \frac{n}{2N}\right) \Im((F_N \mathbf{x}^{(1)})_n) \right) \\ y_{N-n} &= c_{N-n,N} \left(\cos\left(\pi \frac{N-n}{2N}\right) \Re((F_N \mathbf{x}^{(1)})_{N-n}) + \sin\left(\pi \frac{N-n}{2N}\right) \Im((F_N \mathbf{x}^{(1)})_{N-n}) \right) \\ &= c_{n,N} \left(\sin\left(\pi \frac{n}{2N}\right) \Re((F_N \mathbf{x}^{(1)})_n) - \cos\left(\pi \frac{n}{2N}\right) \Im((F_N \mathbf{x}^{(1)})_n) \right), \end{aligned}$$

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

$$\begin{aligned} \cos\left(\pi \frac{n}{2N}\right) y_n + \sin\left(\pi \frac{n}{2N}\right) y_{N-n} &= c_{n,N} \Re((F_N \mathbf{x}^{(1)})_n) \\ \sin\left(\pi \frac{n}{2N}\right) y_n - \cos\left(\pi \frac{n}{2N}\right) y_{N-n} &= c_{n,N} \Im((F_N \mathbf{x}^{(1)})_n). \end{aligned}$$

Adding we get

$$\begin{aligned} c_{n,N} (F_N \mathbf{x}^{(1)})_n &= \cos\left(\pi \frac{n}{2N}\right) y_n + \sin\left(\pi \frac{n}{2N}\right) y_{N-n} + i(\sin\left(\pi \frac{n}{2N}\right) y_n - \cos\left(\pi \frac{n}{2N}\right) y_{N-n}) \\ &= (\cos\left(\pi \frac{n}{2N}\right) + i \sin\left(\pi \frac{n}{2N}\right)) (y_n - i y_{N-n}) = e^{\pi i n / (2N)} (y_n - i y_{N-n}). \end{aligned}$$

This means that $(F_N \mathbf{x}^{(1)})_n = \frac{1}{c_{n,N}} e^{\pi i n / (2N)} (y_n - i y_{N-n})$ for $n \geq 1$. For $n = 0$, since $\Im((F_N \mathbf{x}^{(1)})_n) = 0$ we have that $(F_N \mathbf{x}^{(1)})_0 = \frac{1}{c_{0,N}} y_0$. This means that $\mathbf{x}^{(1)}$ 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 $\frac{1}{c_{n,N}} e^{\pi i n / (2N)} (y_n - i y_{N-n})$:

Theorem 4.9 (IDCT algorithm). Let $\mathbf{x} = (D_N)^T \mathbf{y}$ be the IDCT of \mathbf{y} . and let \mathbf{z} be the vector with component 0 being $\frac{1}{c_{0,N}} y_0$, and the remaining components being $\frac{1}{c_{n,N}} e^{\pi i n / (2N)} (y_n - i y_{N-n})$. Then we have that

$$\mathbf{x}^{(1)} = (F_N)^H \mathbf{z},$$

where $\mathbf{x}^{(1)}$ is defined as in Theorem 4.8.

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

```

MATLAB also has a function 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.

4.2.2 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 4.10 (Number of multiplications required by the DCT and IDCT algorithms). Both the N -point DCT and IDCT factorizations given by Theorem 4.8 and Theorem 4.9 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 4.7, the number of multiplications required by the FFT is $O(2N\log_2 N)$. By Theorem 4.8, 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 $\mathbf{z} = \frac{1}{c_{n,N}} e^{\pi i n/(2N)} (y_n - i y_{N-n})$ seen in Theorem 4.9 should require $4N$ real multiplications to compute. But since the IDFT of \mathbf{z} is real, \mathbf{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 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.

4.2.3 *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 4.1), 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 4.11 (Revised FFT algorithm). Let $\mathbf{y} = F_N \mathbf{x}$ be the N -point DFT of the real vector \mathbf{x} . Then we have that

$$\Re(y_n) = \begin{cases} \frac{1}{\sqrt{2}} \Re((F_{N/2} \mathbf{x}^{(e)})_n) + (ED_{N/4} \mathbf{z})_n & 0 \leq n \leq N/4 - 1 \\ \frac{1}{\sqrt{2}} \Re((F_{N/2} \mathbf{x}^{(e)})_n) & n = N/4 \\ \frac{1}{\sqrt{2}} \Re((F_{N/2} \mathbf{x}^{(e)})_n) - (ED_{N/4} \mathbf{z})_{N/2-n} & N/4 + 1 \leq n \leq N/2 - 1 \end{cases} \quad (4.8)$$

$$\Im(y_n) = \begin{cases} \frac{1}{\sqrt{2}} \Im((F_{N/2} \mathbf{x}^{(e)})_n) & q = 0 \\ \frac{1}{\sqrt{2}} \Im((F_{N/2} \mathbf{x}^{(e)})_n) + (ED_{N/4} \mathbf{w})_{N/4-n} & 1 \leq n \leq N/4 - 1 \\ \frac{1}{\sqrt{2}} \Im((F_{N/2} \mathbf{x}^{(e)})_n) + (ED_{N/4} \mathbf{w})_{n-N/4} & N/4 \leq n \leq N/2 - 1 \end{cases} \quad (4.9)$$

where $\mathbf{x}^{(e)}$ is as defined in Theorem 4.1, where $\mathbf{z}, \mathbf{w} \in \mathbb{R}^{N/4}$ defined by

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

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 $n \geq 1$.

Proof. Taking real and imaginary parts in (4.1) we obtain

$$\begin{aligned}\Re(y_n) &= \frac{1}{\sqrt{2}}\Re((F_{N/2}\mathbf{x}^{(e)})_n) + \frac{1}{\sqrt{2}}\Re((D_{N/2}F_{N/2}\mathbf{x}^{(o)})_n) \\ \Im(y_n) &= \frac{1}{\sqrt{2}}\Im((F_{N/2}\mathbf{x}^{(e)})_n) + \frac{1}{\sqrt{2}}\Im((D_{N/2}F_{N/2}\mathbf{x}^{(o)})_n).\end{aligned}$$

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

$$\begin{aligned}& \Re((D_{N/2}F_{N/2}\mathbf{x}^{(o)})_n) \\ &= \frac{1}{\sqrt{N/2}}\Re(e^{-2\pi in/N} \sum_{k=0}^{N/2-1} (\mathbf{x}^{(o)})_k e^{-2\pi ink/(N/2)}) \\ &= \frac{1}{\sqrt{N/2}}\Re\left(\sum_{k=0}^{N/2-1} (\mathbf{x}^{(o)})_k e^{-2\pi in(k+\frac{1}{2})/(N/2)}\right) \\ &= \frac{1}{\sqrt{N/2}} \sum_{k=0}^{N/2-1} (\mathbf{x}^{(o)})_k \cos\left(2\pi \frac{n(k+\frac{1}{2})}{N/2}\right) \\ &= \frac{1}{\sqrt{N/2}} \sum_{k=0}^{N/4-1} (\mathbf{x}^{(o)})_k \cos\left(2\pi \frac{n(k+\frac{1}{2})}{N/2}\right) \\ &\quad + \frac{1}{\sqrt{N/2}} \sum_{k=0}^{N/4-1} (\mathbf{x}^{(o)})_{N/2-1-k} \cos\left(2\pi \frac{n(N/2-1-k+\frac{1}{2})}{N/2}\right) \\ &= \frac{1}{\sqrt{2}} \frac{1}{\sqrt{N/4}} \sum_{k=0}^{N/4-1} ((\mathbf{x}^{(o)})_k + (\mathbf{x}^{(o)})_{N/2-1-k}) \cos\left(2\pi \frac{n(k+\frac{1}{2})}{N/2}\right) \\ &= (E_0 D_{N/4} \mathbf{z})_n,\end{aligned}$$

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

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

to arrive at $-(E_0 D_{N/4} \mathbf{z})_{N/2-n}$ 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

$$\begin{aligned}
& \Im((D_{N/2}F_{N/2}\mathbf{x}^{(o)})_n) \\
&= \frac{1}{\sqrt{N/2}} \sum_{k=0}^{N/4-1} ((\mathbf{x}^{(o)})_{N/2-1-k} - (\mathbf{x}^{(o)})_k) \sin\left(2\pi \frac{n(k + \frac{1}{2})}{N/2}\right) \\
&= \frac{1}{\sqrt{N/2}} \sum_{k=0}^{N/4-1} ((\mathbf{x}^{(o)})_{N/2-1-k} - (\mathbf{x}^{(o)})_k)(-1)^k \cos\left(2\pi \frac{(N/4-n)(k + \frac{1}{2})}{N/2}\right).
\end{aligned}$$

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 $1 \leq n \leq N/4$ this is $(E_0 D_{N/4} \mathbf{w})_{N/4-n}$, where \mathbf{w} is the vector defined as in the text of the theorem. For $N/4 \leq n \leq N/2 - 1$ we arrive instead at $(E_0 D_{N/4} \mathbf{z})_{n-N/4}$, similarly to as above. This also proves (4.9), and the proof is done. \square

As for Theorem 4.1, 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 4.1.

It is more difficult to obtain a matrix interpretation for Theorem 4.11, 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 4.12 (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 4.11. 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 4.1. We will not prove Theorem 4.12. Instead we will go through the steps in a proof in Exercise 3. The revised FFT has yet a bigger advantage than 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 4.2

Ex. 1 — Write a function

```
function samples=DCTImpl(x)
```

which returns the DCT of the column vector $\mathbf{x} \in \mathbb{R}^{2N}$ 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.

Ex. 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.

Ex. 3 — In this exercise we will compute the number of real multiplications needed by the revised N -point FFT algorithm of Theorem 4.11, denoted M_N .

- a. Explain from the algorithm of Theorem 4.11 that

$$M_N = 2(M_{N/4} + N/2) + M_{N/2} = N + M_{N/2} + 2M_{N/4}. \quad (4.10)$$

- 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

$$x_r = \frac{2}{3}r2^r + C2^r + D(-1)^r.$$

- 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).

4.3 Summary

We obtained an implementation of the DFT which is more efficient in terms of the number of arithmetic operations than a direct implementation of the DFT. We also showed that this could be used for obtaining an efficient implementation of the DCT.

Part II

Wavelets and applications to image processing

Chapter 5

Wavelets

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

1. First of all, the functions used in the approximation are periodic with short periods. In contrast, for most functions encountered in applications the frequency content changes with time. Although Fourier analysis tools also exist for analyzing non-periodic functions, these tools mostly have a theoretical significance and are rarely used in practice, because of lack of efficient implementations.
2. Secondly, all components of the Fourier basis vectors are nonzero — in fact they all have absolute value 1 at all points. This means that, in order to compute a value using the representation in the Fourier basis, we must for each instance in time sum over all N vectors in the basis. This is time-consuming when N is large.

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



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

5.1 Why wavelets?

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

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

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

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

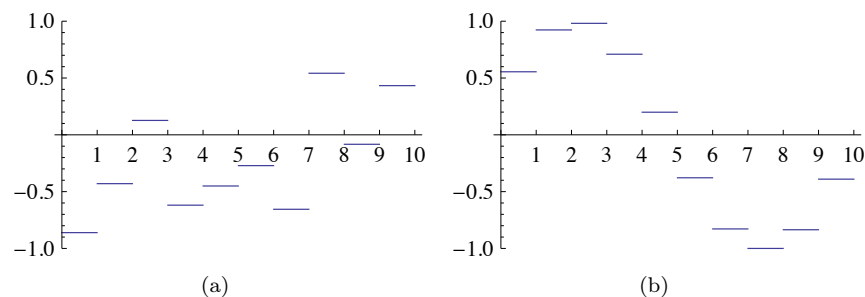


Figure 5.2: Two examples of piecewise constant functions.

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

5.2 Wavelets constructed from piecewise constant functions

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

5.2.1 Resolution spaces

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

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

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

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

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

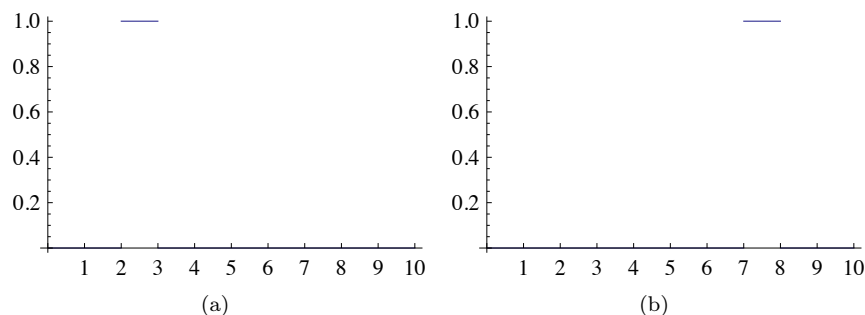


Figure 5.3: The functions ϕ_2 (a) and ϕ_7 (b) in V_0 .

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

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

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

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

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

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

Proof. Two functions ϕ_{n_1} and ϕ_{n_2} with $n_1 \neq n_2$ clearly satisfy $\int \phi_{n_1}(t)\phi_{n_2}(t)dt = 0$ since $\phi_{n_1}(t)\phi_{n_2}(t) = 0$ for all values of x . It is also easy to check that $\|\phi_n\| = 1$ for all n . Finally, any function in V_0 can be written as a linear combination the functions $\phi_0, \phi_1, \dots, \phi_{N-1}$, so the conclusion of the lemma follows. \square

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

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

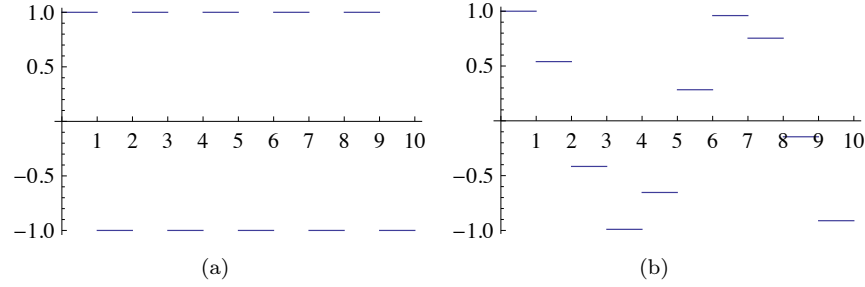


Figure 5.4: The square wave in V_0 (a) and an approximation to $\cos t$ from V_0 .

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

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

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

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

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

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

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

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

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

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

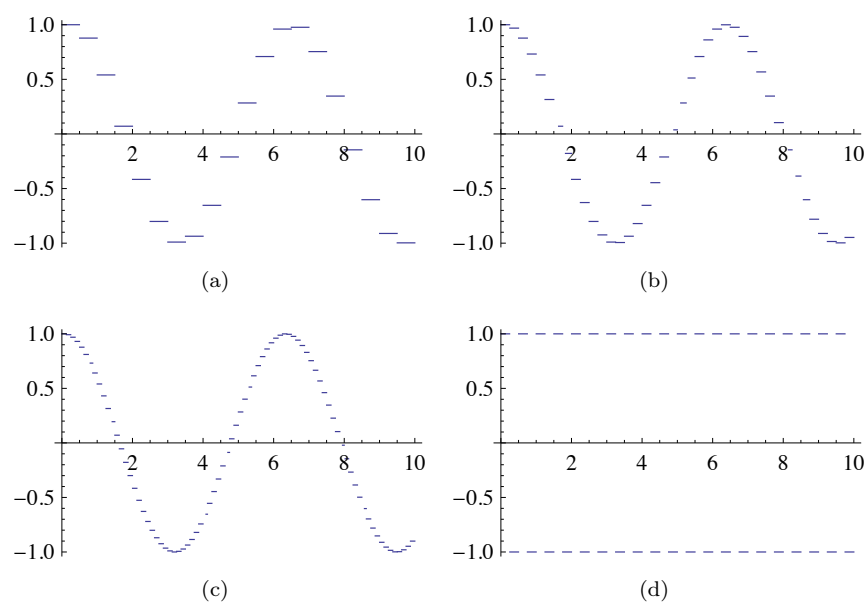


Figure 5.5: Piecewise constant approximations to $\cos t$ on the interval $[0, 10]$ in the spaces V_1 (a), V_2 (b), and V_3 (c). The plot in (d) shows the square wave in V_2 .

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

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

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

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

5.2.2 Function approximation property

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

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

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

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

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

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

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

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

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

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

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

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

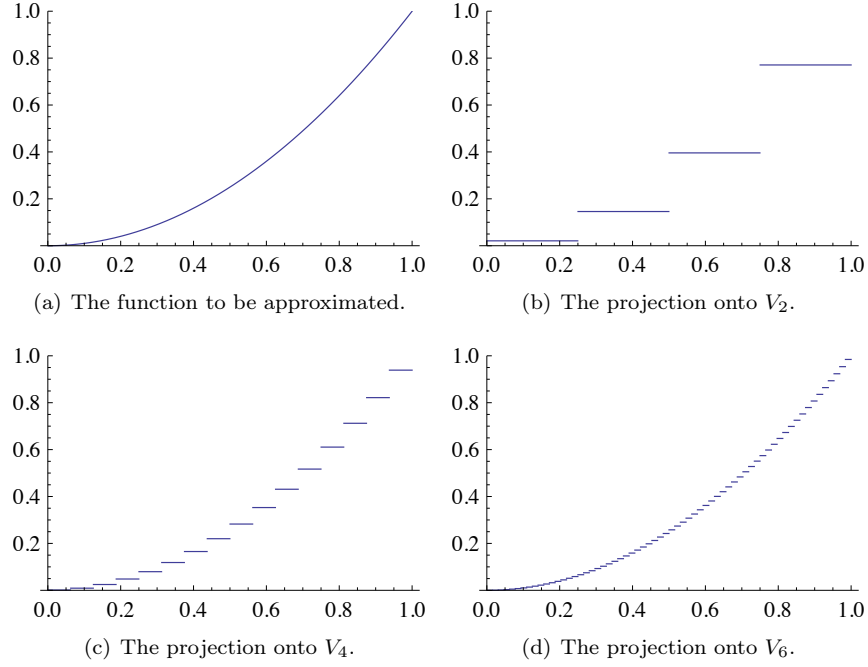


Figure 5.6: Comparison of the function defined by $f(t) = t^2$ on $[0, 1]$ with the projection onto different spaces V_m .

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

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

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

5.2.3 Detail spaces and wavelets

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

We start by observing that since

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

we have

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

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

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

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

Lemma 5.8. The spaces $V_0, V_1, \dots, V_m, \dots$ are nested,

$$V_0 \subset V_1 \subset V_2 \subset \dots \subset V_m \dots$$

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

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

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

If $g_1 \in V_1$ is given by

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

then

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

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

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

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

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

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

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

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

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

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

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

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

which proves (5.8) □

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

onto V_0 . The error is then

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

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

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

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

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

which is (5.12) □

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

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

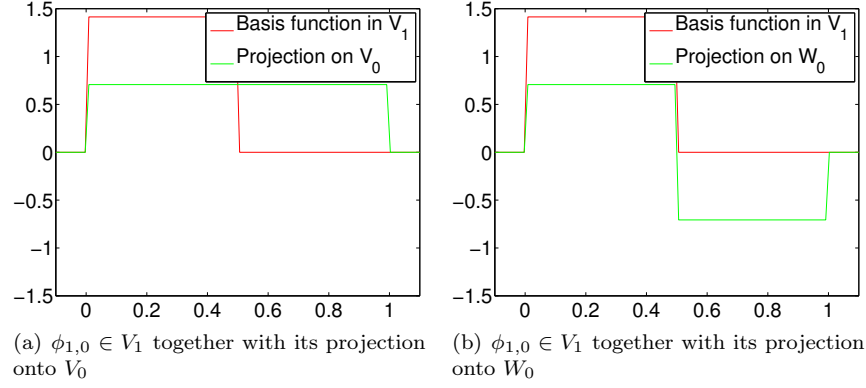


Figure 5.7: The projection of a basis function in V_1 onto V_0 and W_0 .

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

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

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

Lemma 5.14. Define the function ψ by

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

and set

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

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

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

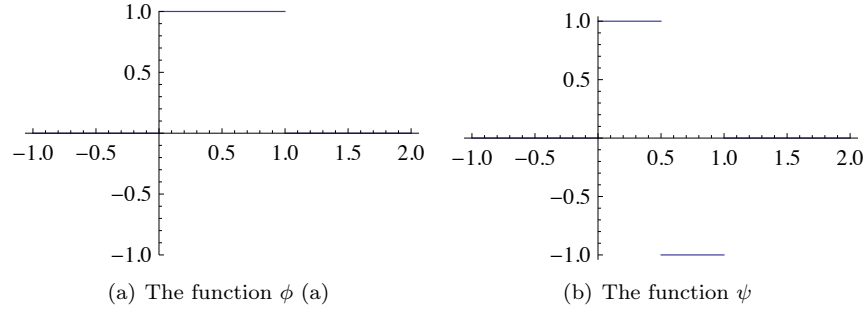


Figure 5.8: The functions we used to analyse the space of piecewise constant functions

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

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

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

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

$$\phi_1 = (\phi_{1,n})_{n=0}^{2N-1} \quad \text{and} \quad (\phi_0, \psi_0) = ((\phi_{0,n})_{n=0}^{N-1}, (\psi_{0,n})_{n=0}^{N-1}).$$

The Discrete Wavelet Transform (DWT) is the change of coordinates from the basis ϕ_1 to the basis (ϕ_0, ψ_0) . If

$$g_1 = \sum_{n=0}^{2N-1} c_{1,n} \phi_{1,n} \in V_1$$

$$g_0 = \sum_{n=0}^{N-1} c_{0,n} \phi_{0,n} \in V_0$$

$$e_0 = \sum_{n=0}^{N-1} w_{0,n} \psi_{0,n} \in W_0$$

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

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

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

Conversely, the Inverse Discrete Wavelet Transform (IDWT) is the change of coordinates from the basis (ϕ_0, ψ_0) to the basis ϕ_1 , and is given by

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

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

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

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

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

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

□

It is common to reorder the basis vectors in (ϕ_0, ψ_0) to

$$\mathcal{C}_1 = \{\phi_{0,0}, \psi_{0,0}, \phi_{0,1}, \psi_{0,1}, \dots, \phi_{0,N-1}, \psi_{0,N-1}\}. \quad (5.22)$$

The subscript 1 is used since \mathcal{C}_1 is a basis for V_1 . This reordering of the basis functions is useful since it makes it easier to write down the change of coordinates matrices. To be more precise, from formulas (5.17)–(5.18) it is apparent that $P_{\phi_1 \leftarrow \mathcal{C}_1}$ is the matrix where

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

is repeated along the main diagonal N times. Also, from formulas (5.15)–(5.16) it is apparent that $P_{\mathcal{C}_1 \leftarrow \phi_1}$ is the same matrix. Such matrices are called *block diagonal matrices*. This particular block diagonal matrix is clearly orthogonal, since it transforms one orthonormal base to another.

Exercises for Section 5.2

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

Ex. 2 — Show that

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

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

Ex. 3 — Show that

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

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

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

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

Ex. 5 — Use lemma 5.9 to write down the matrix for the linear transformation $\text{proj}_{V_0} : V_1 \rightarrow V_0$ relative to the bases ϕ_1 and ϕ_0 . Also, use lemma 5.12 to write down the matrix for the linear transformation $\text{proj}_{W_0} : V_1 \rightarrow W_0$ relative to the bases ϕ_1 and ψ_0 .

Ex. 6 — Show that

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

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

5.3 Multiresolution analysis for piecewise constant functions

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

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

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

5.3.1 Extraction of details at higher resolutions

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

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

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

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

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

and

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

If

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

and $g_m = g_{m-1} + e_{m-1}$, then the change of coordinates from the basis ϕ_m to the basis (ϕ_{m-1}, ψ_{m-1}) is given by

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

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

Conversely, the change of coordinates from the basis (ϕ_{m-1}, ψ_{m-1}) to the basis ϕ_m is given by

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

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

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

$$\mathcal{C}_m = \{\phi_{m-1,0}, \psi_{m-1,0}, \phi_{m-1,1}, \psi_{m-1,1}, \dots, \phi_{m-1,2^{m-1}N-1}, \psi_{m-1,2^{m-1}N-1}\}. \quad (5.30)$$

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

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

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

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

$$\begin{aligned} V_m &= V_{m-1} \oplus W_{m-1} \\ &= V_{m-2} \oplus W_{m-2} \oplus W_{m-1} \\ &\vdots \\ &= V_0 \oplus W_0 \oplus W_1 \oplus \dots \oplus W_{m-2} \oplus W_{m-1}. \end{aligned} \quad (5.31)$$

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

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

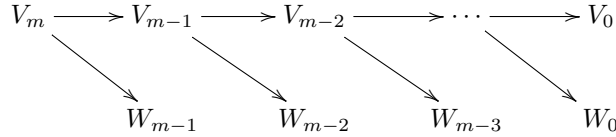
When we above say “basis for $V_0 \oplus W_0 \oplus W_1 \oplus \dots \oplus W_{m-2} \oplus W_{m-1}$ ”, we really mean the canonical basis for this space. In general, the Discrete Wavelet Transform is used to denote a change of coordinates from ϕ_m to the canonical basis, for any m .

Definition 5.20 (m -level Discrete Wavelet Transform). Let F_m denote the change of coordinates matrix from ϕ_m to the canonical basis

$$\phi_0 \oplus \psi_0 \oplus \psi_1 \oplus \dots \oplus \psi_{m-2} \oplus \psi_{m-1}$$

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

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



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

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

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

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

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

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

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

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

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

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

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

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

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

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

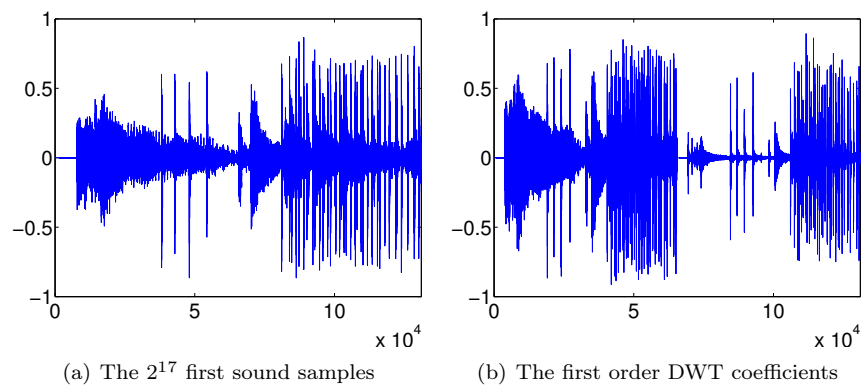


Figure 5.9: The sound samples and the DWT coefficients of the sound `castanets.wav`.

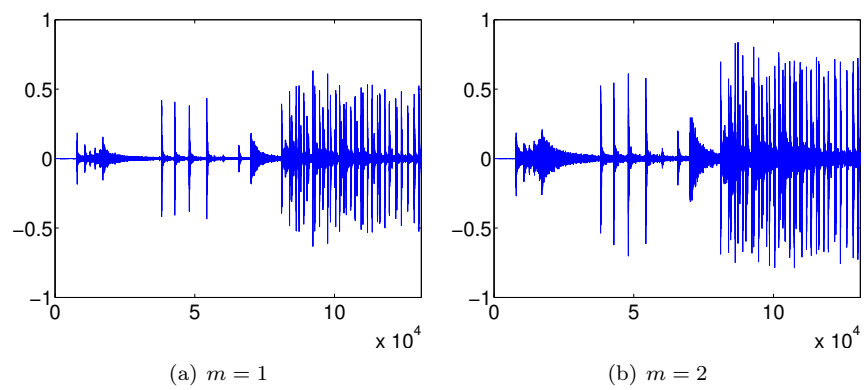


Figure 5.10: The error (i.e. the contribution from $W_0 \oplus W_1 \oplus \dots \oplus W_{m-1}$) in the sound file `castanets.wav`, for different values of m .

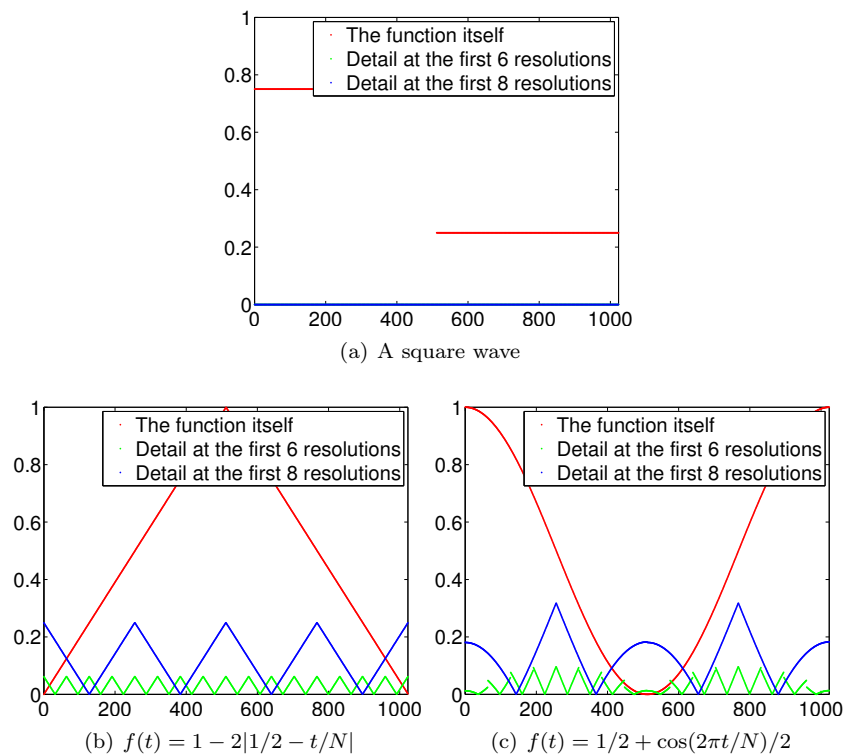


Figure 5.11: The error (i.e. the contribution from $W_0 \oplus W_1 \oplus \dots \oplus W_{m-1}$) for $N = 1024$ for different functions $f(t)$, for different values of m .

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

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

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

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

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

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

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

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

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

5.3.2 Matrix factorization of the DWT

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

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

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

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

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

$$G_m = (P_{\phi_m \leftarrow \mathcal{C}_m})^T$$

$$H_m = P_{\mathcal{C}_m \leftarrow \phi_m}$$

The m -level DWT and IDWT can be expressed as

$$F_m = (P_{2^1 N} H_1 \oplus I_{2^m N - 2^1 N}) (P_{2^2 N} H_2 \oplus I_{2^m N - 2^2 N})$$

$$\cdots (P_{2^{m-1} N} H_{m-1} \oplus I_{2^m N - 2^{m-1} N}) P_{2^m N} H_m,$$

$$(F_m)^{-1} = (P_{2^m N} G_m)^T ((P_{2^{m-1} N} G_{m-1})^T \oplus I_{2^m N - 2^{m-1} N})$$

$$\cdots ((P_{2^2 N} G_2)^T \oplus I_{2^m N - 2^2 N}) ((P_{2^1 N} G_1)^T \oplus I_{2^m N - 2^1 N}).$$

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

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

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

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

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

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

where we used Exercise 5. The result now follows. \square

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

5.3.3 Summary

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

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

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

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

$$V_0 \subset V_1 \subset V_2 \subset \cdots \subset V_m \subset \cdots \quad (5.32)$$

so that

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

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

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

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

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

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

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

Exercises for Section 5.3

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

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

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

$$T_1 \oplus T_2 \oplus \dots \oplus T_n(\mathbf{c}_1 + \mathbf{c}_2 + \dots + \mathbf{c}_n) = T_1(\mathbf{c}_1) \oplus T_2(\mathbf{c}_2) \oplus \dots \oplus T_n(\mathbf{c}_n)$$

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

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

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

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

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

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

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

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

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

- a. Write a function

```
function playDWTlower(m)
```

which

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

- b. Run the function `playDWTlower` for different values of m . For which m can you hear that the sound gets degraded? How does it get degraded? Compare with what you heard through the function `playDFTlower` in Example 3.15, where you performed a DFT on the sound sample instead, and set some of the DFT coefficients to zero.
- c. Do the sound samples returned by `playDWTlower` lie in $[-1, 1]$?

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

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

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

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

5.4 Wavelets constructed from piecewise linear functions

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

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

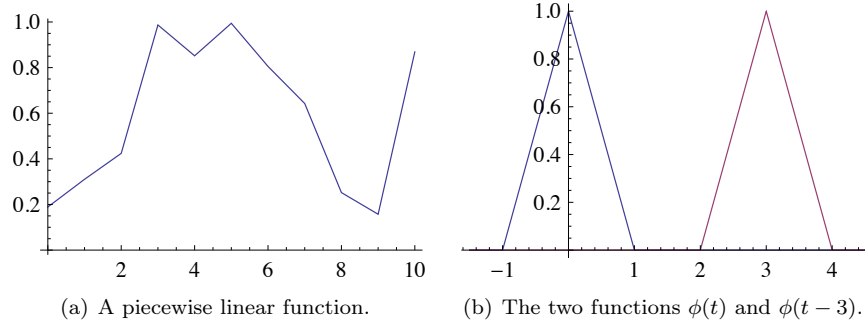


Figure 5.12: Some piecewise linear functions.

compression.

5.4.1 Multiresolution analysis

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

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

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

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

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

Lemma 5.29. Let the function ϕ be defined by

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

and for any $m \geq 0$ set

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

or in vector notation

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

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

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

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

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

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

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

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

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

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

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

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

Lemma 5.31. The piecewise linear resolution spaces are nested,

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

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

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

Let us sum up our findings so far.

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

The next step in the derivation of wavelets is to find formulas that let us express a function given in the basis ϕ_0 for V_0 in terms of the basis ϕ_1 for V_1 .

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

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

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

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

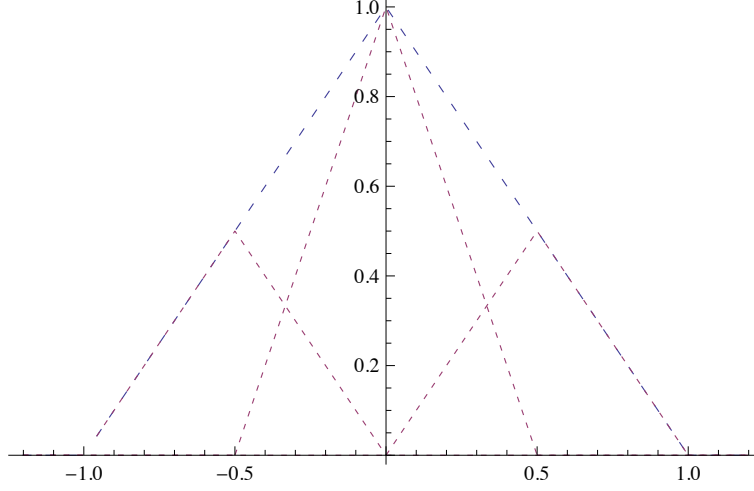
then

$$c_{1,2n} = c_{0,n}, \quad \text{for } n = 0, 1, \dots, N-1; \quad (5.37)$$

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

Proof. Since $\phi_{0,n}$ is in V_0 it may be expressed in the basis ϕ_1 with formula (5.35),

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



The relation (5.36) now follows since

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

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

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

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

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

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

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

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

$$\begin{pmatrix} \mathbf{c}_1^e \\ \mathbf{c}_1^o \end{pmatrix} = \begin{pmatrix} I \\ A_0 \end{pmatrix} \mathbf{c}_0, \quad (5.39)$$

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

$$A_0 = \frac{1}{2} \begin{pmatrix} 1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 1 \end{pmatrix}. \quad (5.40)$$

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

5.4.2 Detail spaces and wavelets

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

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

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

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

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

$$g_0(n) = g_1(n), \quad n = 0, 1, \dots, N-1. \quad (5.42)$$

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

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

$$c_{0,n} = c_{1,2n}, \quad \text{for } n = 0, 1, \dots, N-1, \quad (5.43)$$

and

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

and

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

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

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

and $g_m = g_{m-1} + e_{m-1}$, then the change of coordinates from the basis ϕ_m to the basis (ϕ_{m-1}, ψ_{m-1}) is given by

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

Conversely, the change of coordinates from the basis (ϕ_{m-1}, ψ_{m-1}) to the basis ϕ_m is given by

$$c_{m,2n} = c_{m-1,n}, \quad (5.45)$$

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

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

$$\begin{pmatrix} \mathbf{c}_m^e \\ \mathbf{c}_m^o \end{pmatrix} = \begin{pmatrix} I & 0 \\ A_{m-1} & I \end{pmatrix} \begin{pmatrix} \mathbf{c}_{m-1} \\ \mathbf{w}_{m-1} \end{pmatrix} \quad (5.47)$$

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

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

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

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

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

Using these instead, the two equations above take the form

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

These two relations together give all columns in the change of coordinate matrix $P_{\phi_1 \leftarrow \mathcal{C}_1}$, when the spaces ϕ_m, \mathcal{C}_m instead are defined in terms of the function ψ , and the normalized ϕ . In particular, the first two columns in this matrix are

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

The remaining columns are obtained by shifting this, as in a circulant Toeplitz matrix. Similarly we can compute the change of coordinate matrix the opposite way, $P_{\mathcal{C}_1 \leftarrow \phi_1}$: Equations (5.49) can be written

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

from which it follows that

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

which in the same way as above give the following two first columns in the change of coordinate matrix $P_{\mathcal{C}_1 \leftarrow \phi_1}$:

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

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

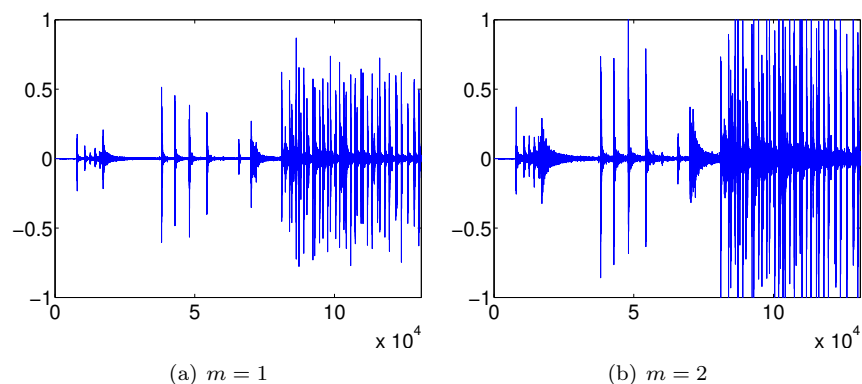


Figure 5.13: The error (i.e. the contribution from $W_0 \oplus W_1 \oplus \dots \oplus W_{m-1}$) in the sound file `castanets.wav`, for different values of m .

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

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

Exercises for Section 5.4

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

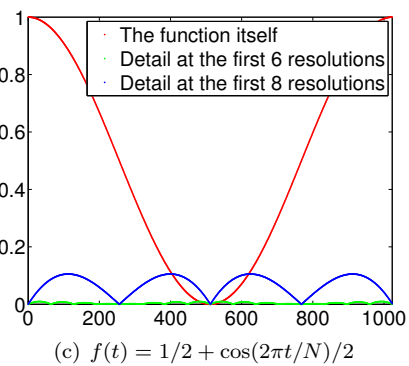
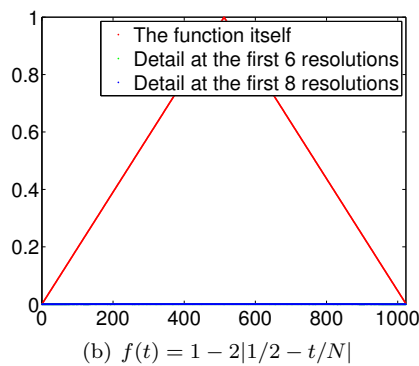
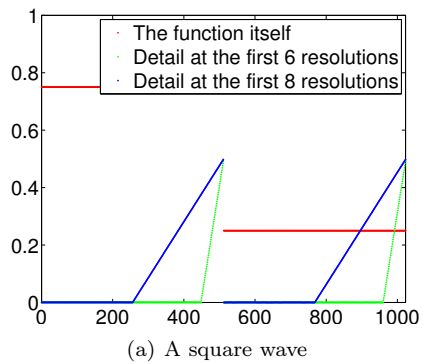


Figure 5.14: The error (i.e. the contribution from $W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}$) for $N = 1024$ for different functions $f(t)$, for different values of m .

Ex. 2 — Show that

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

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

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

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

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

5.5 Alternative wavelets for piecewise linear functions

The direct sum decomposition that we derived in Section 5.4 was very simple, but also has its shortcomings. To see this, set $N = 1$ and consider the space V_{10} , which has dimension 2^{10} , which in most cases will mean that the function g_{10} will be a very good representation of the underlying data. However, when we compute g_{m-1} we just pick every other coefficient from g_m . By the time we get to g_0 we are just left with the first and last coefficient from g_{10} . In some situations this may be adequate, but usually not.

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

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

If f is s times differentiable, it can be represented as $f = P_s(x) + Q_s(x)$, where P_s is a polynomial of degree s , and Q_s is a function which is very small (P_s could for instance be a Taylor series expansion of f). If in addition $\langle t^k, \psi \rangle = 0$, for $k = 1, \dots, s$, we have also that $\langle t^k, \psi_{r,t} \rangle = 0$ for $r \leq s$, so that $\langle P_s, \psi_{r,t} \rangle = 0$

also. This means that (5.52) can be written

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

Here the first sum lies in V_0 . We see that the wavelet coefficients from W_r are $\langle Q_s, \psi_{r,n} \rangle$, which are very small since Q_s is small. This means that the detail in the different spaces W_r is very small, which is exactly what we aimed for. Let us summarize this as follows:

Theorem 5.40 (Vanishing moments). We say that ψ has k vanishing moments if the integrals $\int_{-\infty}^{\infty} t^l \psi(t) dt = 0$ for all $0 \leq l \leq k-1$. If a function $f \in V_m$ is r times differentiable, and ψ has r vanishing moments, then f can be approximated well from V_0 . Moreover, the quality of this approximation improves when r increases.

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

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

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

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

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

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

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

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

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

Lemma 5.42. The function

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

satisfies the conditions

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

Using Equation (5.36), which stated that

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

we get

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

Note that what we did here is equivalent to finding the coordinates of $\hat{\psi}$ in the basis ϕ_1 : Equation (5.55) says that

$$[\hat{\psi}]_{\phi_0 \oplus \psi_0} = (-1/4, -1/4, 0, \dots, 0) \oplus (1, 0, \dots, 0). \quad (5.58)$$

Since the IDWT is the change of coordinates from $\phi_0 \oplus \psi_0$ to ϕ_1 , we could also have computed $[\hat{\psi}]_{\phi_1}$ by taking the IDWT of $(-1/4, -1/4, 0, \dots, 0) \oplus (1, 0, \dots, 0)$. In the next section we will consider more general implementations of the DWT and the IDWT, which we thus can use instead of performing the computation above.

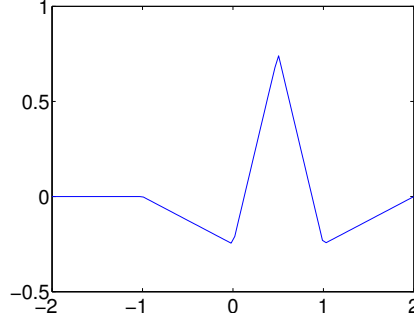


Figure 5.15: The function ψ we constructed as an alternative wavelet for piecewise linear functions.

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

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

These two relations together give all columns in the change of coordinate matrix $P_{\phi_1 \leftarrow \mathcal{C}_1}$, when the spaces ϕ_m, \mathcal{C}_m instead are defined in terms of the function $\hat{\psi}$, and the normalized ϕ . In particular, the first two columns in this matrix are

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

The first column is the same as before, since there was no change in the definition of ϕ . The remaining columns are obtained by shifting this, as in a circulant Toeplitz matrix. Similarly we could compute the change of coordinate matrix the opposite way, $P_{\mathcal{C}_1 \leftarrow \phi_1}$. We will explain how this can be done in the next section. The function ψ is plotted in Figure 5.15.

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

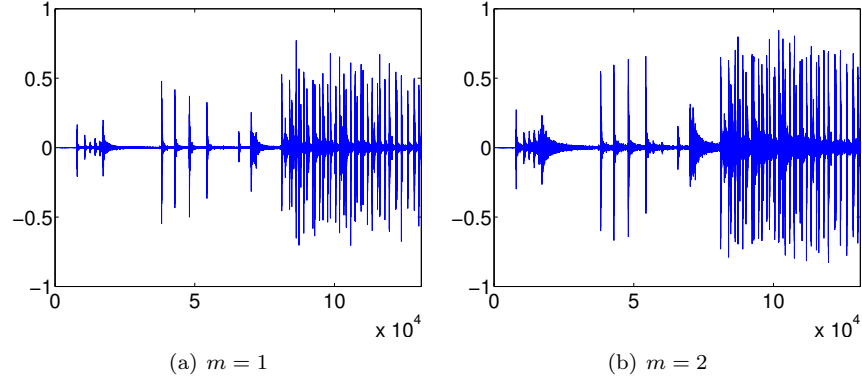


Figure 5.16: The error (i.e. the contribution from $W_0 \oplus W_1 \oplus \dots \oplus W_{m-1}$) in the sound file `castanets.wav`, for different values of m .

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

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

Exercises for Section 5.5

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

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

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

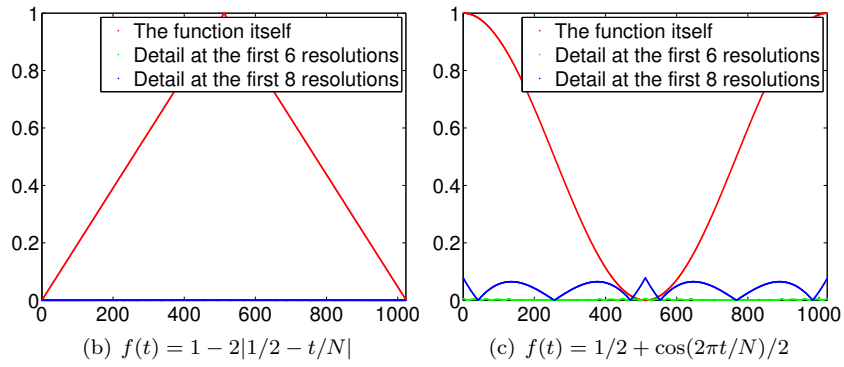
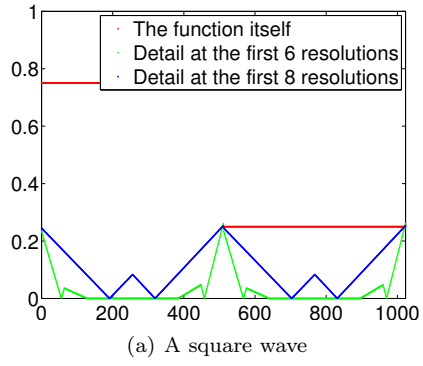


Figure 5.17: The error (i.e. the contribution from $W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}$) for $N = 1024$ for different functions $f(t)$, for different values of m .

b. Show that

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

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

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

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

a. Define

$$a_k = \int_{-1}^1 t^k (1 - |t|) dt$$

$$b_k = \int_0^2 t^k (1 - |t - 1|) dt$$

$$e_k = \int_0^1 t^k (1 - 2|t - 1/2|) dt,$$

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

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

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

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

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

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

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

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

$$d_k = \int_1^3 t^k (1 - |t - 2|) dt$$

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

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

and solve this with Matlab.

- c. Plot the function defined by (5.60), which you found in b.
Hint: If \mathbf{t} is the vector of t -values, and you write
 $(\mathbf{t} > 0) .* (\mathbf{t} \leq 1) .* (1 - 2 * \text{abs}(\mathbf{t} - 0.5))$, you get the points $\phi_{1,1}(t)$.
- d. Explain why the coordinate vector of $\hat{\psi}$ in the basis $\phi_0 \oplus \psi_0$ is

$$[\hat{\psi}]_{\phi_0 \oplus \psi_0} = (-\alpha, -\beta, -\delta, 0, \dots, 0 - \gamma) \oplus (1, 0, \dots, 0).$$

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

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

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

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

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

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

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

- b. Write a function

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

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

5.6 Wavelets and filters

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

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

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

Definition 5.46. We denote by H_0 the (unique) filter with the same first row as $P_{c_m \leftarrow \phi_m}$, and by H_1 the (unique) filter with the same second row as $P_{c_m \leftarrow \phi_m}$.

Using this definition it is clear that

$$\begin{aligned} (P_{c_m \leftarrow \phi_m} \mathbf{c}_m)_k &= (H_0 \mathbf{c}_m)_k && \text{when } k \text{ is even} \\ (P_{c_m \leftarrow \phi_m} \mathbf{c}_m)_k &= (H_1 \mathbf{c}_m)_k && \text{when } k \text{ is odd} \end{aligned}$$

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

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

1. Compute $H_0 \mathbf{c}_m$. The even-indexed entries in the result are the coordinates \mathbf{c}_{m-1} in ϕ_{m-1} .
2. Compute $H_1 \mathbf{c}_m$. The odd-indexed entries in the result are the coordinates \mathbf{w}_{m-1} in ψ_{m-1} .

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

$$\begin{array}{ccccccc}
 V_m & \xrightarrow{H_0} & V_{m-1} & \xrightarrow{H_0} & V_{m-2} & \xrightarrow{H_0} & \cdots \xrightarrow{H_0} V_0 \\
 & \searrow H_1 & & \searrow H_1 & & \searrow H_1 & \\
 & & W_{m-1} & & W_{m-2} & & W_{m-3} & & W_0
 \end{array}$$

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

Definition 5.48. We denote by G_0 the (unique) filter with the same first column as $P_{\phi_m \leftarrow c_m}$, and by G_1 the (unique) filter with the same second column as $P_{\phi_m \leftarrow c_m}$.

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

$$\begin{aligned}
 P_{\phi_m \leftarrow c_m} \begin{pmatrix} c_{m-1,0} \\ w_{m-1,0} \\ c_{m-1,1} \\ w_{m-1,1} \\ \vdots \\ c_{m-1,2^{m-1}N-1} \\ w_{m-1,2^{m-1}N-1} \end{pmatrix} &= P_{\phi_m \leftarrow c_m} \left(\begin{pmatrix} c_{m-1,0} \\ 0 \\ c_{m-1,1} \\ 0 \\ \vdots \\ c_{m-1,2^{m-1}N-1} \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ w_{m-1,0} \\ 0 \\ w_{m-1,1} \\ \vdots \\ 0 \\ w_{m-1,2^{m-1}N-1} \end{pmatrix} \right) \\
 &= P_{\phi_m \leftarrow c_m} \begin{pmatrix} c_{m-1,0} \\ 0 \\ c_{m-1,1} \\ 0 \\ \vdots \\ c_{m-1,2^{m-1}N-1} \\ 0 \end{pmatrix} + P_{\phi_m \leftarrow c_m} \begin{pmatrix} 0 \\ w_{m-1,0} \\ 0 \\ w_{m-1,1} \\ \vdots \\ 0 \\ w_{m-1,2^{m-1}N-1} \end{pmatrix} \\
 &= G_0 \begin{pmatrix} c_{m-1,0} \\ 0 \\ c_{m-1,1} \\ 0 \\ \vdots \\ c_{m-1,2^{m-1}N-1} \\ 0 \end{pmatrix} + G_1 \begin{pmatrix} 0 \\ w_{m-1,0} \\ 0 \\ w_{m-1,1} \\ \vdots \\ 0 \\ w_{m-1,2^{m-1}N-1} \end{pmatrix}.
 \end{aligned}$$

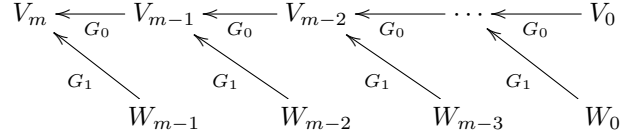
Here we have split a vector into its even-indexed and odd-indexed elements, which correspond to the coefficients from ϕ_{m-1} and ψ_{m-1} , respectively. In the last equation, we replaced with G_0, G_1 , since the multiplications with $P_{\phi_m \leftarrow c_m}$ depend only on the even and odd columns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

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

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

$$c_m = G_0 \begin{pmatrix} c_{m-1,0} \\ 0 \\ c_{m-1,1} \\ 0 \\ \vdots \\ c_{m-1,2^{m-1}N-1} \\ 0 \end{pmatrix} + G_1 \begin{pmatrix} 0 \\ w_{m-1,0} \\ 0 \\ w_{m-1,1} \\ \vdots \\ 0 \\ w_{m-1,2^{m-1}N-1} \end{pmatrix}. \quad (5.62)$$

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



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

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

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

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

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

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

5.6.1 Frequency response for the Haar Wavelet

For the Haar wavelet we saw that, in $P_{\phi_m \leftarrow c_m}$, the matrix

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

repeated along the diagonal. From this it is clear that

$$\begin{aligned} G_0 &= \{1/\sqrt{2}, 1/\sqrt{2}\} \\ G_1 &= \{1/\sqrt{2}, -1/\sqrt{2}\}. \end{aligned}$$

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

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

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

$$\begin{aligned} H_0 &= \{1/\sqrt{2}, 1/\sqrt{2}\} \\ H_1 &= \{1/\sqrt{2}, -1/\sqrt{2}\}, \end{aligned}$$

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

5.6.2 Frequency responses for wavelets of piecewise linear functions

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

$$\begin{aligned} G_0 &= \frac{1}{\sqrt{2}}\{1/2, 1, 1/2\} \\ G_1 &= \frac{1}{\sqrt{2}}\{1\}. \end{aligned} \tag{5.65}$$

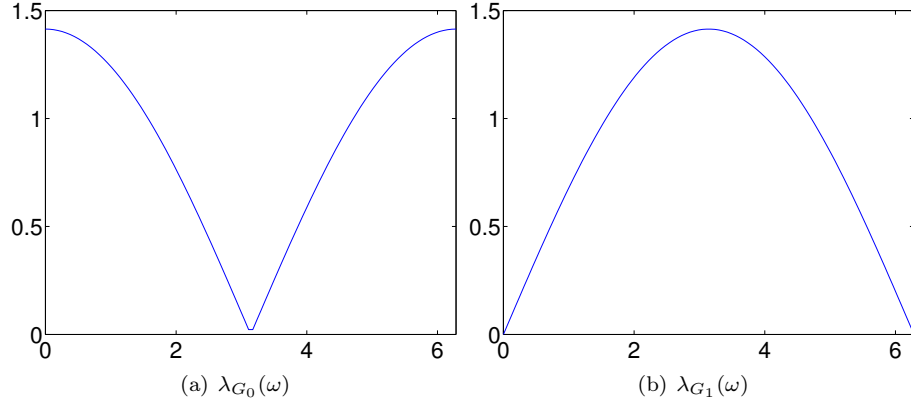


Figure 5.18: The frequency responses for the MRA of piecewise constant functions.

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

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

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

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

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

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

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

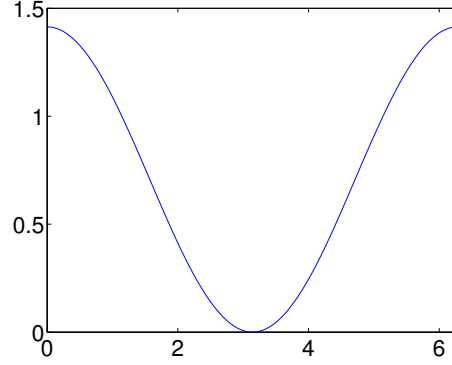


Figure 5.19: The frequency response $\lambda_{G_0}(\omega)$ for the first choice of wavelet for piecewise linear functions

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

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

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

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

so that

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

We now have that

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

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

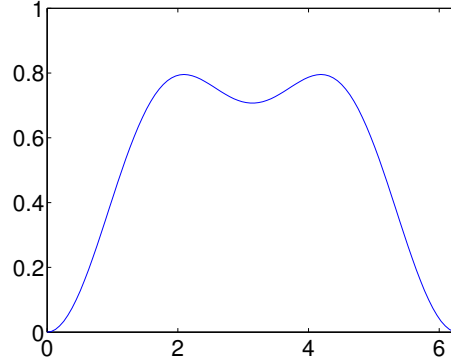


Figure 5.20: The frequency response $\lambda_{G_1}(\omega)$ for the alternative wavelet for piecewise linear functions.

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

5.6.3 Filter-based algorithm for the DWT and the IDWT

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

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

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

$$\begin{aligned} H_0 &= \{h_{0,-k_1}, \dots, h_{0,-1}, \underline{h_{0,0}}, h_{0,1}, \dots, h_{0,k_0}\} \\ H_1 &= \{h_{1,-k_1}, \dots, h_{1,-1}, \underline{h_{1,0}}, h_{1,1}, \dots, h_{1,k_1}\} \end{aligned}$$

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

$$\begin{aligned} \mathbf{a0} &= (h_{0,0}, h_{0,1}, \dots, h_{0,k_0}) \\ \mathbf{a1} &= (h_{1,0}, h_{1,1}, \dots, h_{1,k_1}), \end{aligned}$$

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

$$\begin{aligned} G_0 &= \{g_{0,-l_1}, \dots, g_{0,-1}, \underline{g_{0,0}}, g_{0,1}, \dots, g_{0,l_0}\} \\ G_1 &= \{g_{1,-l_1}, \dots, g_{1,-1}, \underline{g_{1,0}}, g_{1,1}, \dots, g_{1,l_1}\}, \end{aligned}$$

and we write as above

$$\begin{aligned} \mathbf{g0} &= (g_{0,0}, g_{0,1}, \dots, g_{0,l_0}) \\ \mathbf{g1} &= (g_{1,0}, g_{1,1}, \dots, g_{1,l_1}). \end{aligned}$$

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

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

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

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

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

$$h_0 = (h_{0,0}, h_{0,1}, \dots, h_{0,k_0}) = (\sqrt{2}) \quad (5.70)$$

$$h_1 = (h_{1,0}, h_{1,1}, \dots, h_{1,k_1}) = (\sqrt{2}, -\sqrt{2}/2), \quad (5.71)$$

and from Equation (5.65) we see that

$$g_0 = (g_{0,0}, g_{0,1}, \dots, g_{0,l_0}) = (1/\sqrt{2}, 1/(2\sqrt{2})) \quad (5.72)$$

$$g_1 = (g_{1,0}, g_{1,1}, \dots, g_{1,l_1}) = (1/\sqrt{2}). \quad (5.73)$$

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

$$h_0 = (h_{0,0}, h_{0,1}, \dots, h_{0,k_0}) = (3\sqrt{2}/4, \sqrt{2}/4, -\sqrt{2}/8)$$

$$h_1 = (h_{1,0}, h_{1,1}, \dots, h_{1,k_1}) = (\sqrt{2}, -\sqrt{2}/2),$$

and from Equation (5.67) we see that

$$\begin{aligned} \mathbf{g0} &= (g_{0,0}, g_{0,1}, \dots, g_{0,l_0}) = (1/\sqrt{2}, 1/(2\sqrt{2})) \\ \mathbf{g1} &= (g_{1,0}, g_{1,1}, \dots, g_{1,l_1}) = (3/(4\sqrt{2}), -1/(4\sqrt{2}), -1/(8\sqrt{2})). \end{aligned}$$

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

Exercises for Section 5.6

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

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

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

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

Ex. 4 — Write a function

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

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

Ex. 5 — Write a function

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

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

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

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

$$P_{\mathcal{C}_1 \leftarrow \phi_1} = \begin{pmatrix} 1/5 & 1/5 & 1/5 & 0 & 0 & 0 & \cdots & 0 & 1/5 & 1/5 \\ -1/3 & 1/3 & -1/3 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & -1/3 & 1/3 & -1/3 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

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

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

$$P_{\phi_1 \leftarrow \mathcal{C}_1} = \begin{pmatrix} 1/2 & -1/4 & 0 & 0 & \cdots \\ 1/4 & 3/8 & 1/4 & 1/16 & \cdots \\ 0 & -1/4 & 1/2 & -1/4 & \cdots \\ 0 & 1/16 & 1/4 & 3/8 & \cdots \\ 0 & 0 & 0 & -1/4 & \cdots \\ 0 & 0 & 0 & 1/16 & \cdots \\ 0 & 0 & 0 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots \\ 1/4 & 1/16 & 0 & 0 & \cdots \end{pmatrix}$$

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

Ex. 7 — Let us also practice on writing down the change of coordinate matrices from the parameters $\mathbf{h0}, \mathbf{h1}, \mathbf{g0}, \mathbf{g1}$.

- a. Assume that $\mathbf{h0}=[3/8 \ 1/4 \ 1/16]$ and $\mathbf{h1}=[1/2 \ -1/4]$. Write down the compact form for the filters H_0, H_1 . Plot the frequency responses and verify that H_0 is a lowpass filter, and that H_1 is a highpass filter. Also write down the change of coordinate matrix $P_{\mathcal{C}_1 \leftarrow \phi_1}$ for the wavelet corresponding to these filters.
- b. Assume that $\mathbf{g0}=[1/3 \ 1/3]$ and $\mathbf{g1}=[1/5 \ -1/5 \ 1/5]$. Write down the compact form for the filters G_0, G_1 . Plot the frequency responses and verify that G_0 is a lowpass filter, and that G_1 is a highpass filter. Also write down the change of coordinate matrix $P_{\phi_1 \leftarrow \mathcal{C}_1}$ for the wavelet corresponding to these filters.

Ex. 8 — Write a function

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

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

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

- Reimplement the function you made in Exercise 8 so that it instead plays the contribution from the detail spaces. Call the new function `playDWTfilterslowerdifference`.
- In Exercise 5.3.11 we implemented a function `playDWTlowerdifference` for listening to the detail/error when the Haar wavelet is used. In the function `playDWTall` from Example 5.50, replace `playDWTlower` and `playDWTfilterslower` with `playDWTlowerdifference` and `playDWTfilterslowerdifference`. Describe the sounds you hear for different m . Try to explain why the sound seems to get louder when you increase m .

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

- With $\hat{\psi}$ as defined as in Exercise 5.5.2 b., compute the coordinates of $\hat{\psi}$ in the basis ϕ_1 (i.e. $[\hat{\psi}]_{\phi_1}$) with $N = 8$, i.e. compute the IDWT of

$$[\hat{\psi}]_{\phi_0 \oplus \psi_0} = (-\alpha, -\beta, -\delta, 0, 0, 0, 0, -\gamma) \oplus (1, 0, 0, 0, 0, 0, 0, 0),$$

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

- If we redefine the basis \mathcal{C}_1 from $\{\phi_{0,0}, \psi_{0,0}, \phi_{0,1}, \psi_{0,1}, \dots\}$, to $\{\phi_{0,0}, \hat{\psi}_{0,0}, \phi_{0,1}, \hat{\psi}_{0,1}, \dots\}$, the vector you obtained in a. gives us an expression for the second column in $P_{\phi_1 \leftarrow \mathcal{C}_1}$. After redefining the basis like this, the corresponding filter G_1 has changed from that of the piecewise linear wavelet we started with. Use Matlab to state the new filter G_1 with our compact filter notation. Also, plot its frequency response. Hint: Here you are asked to find the unique filter with the same second column as $P_{\phi_1 \leftarrow \mathcal{C}_1}$, i.e. the vector from a..
- Write code which uses Equation (5.68) to find H_0, H_1 from G_0, G_1 , and state these filters with our compact filter notation. Also, state the forms

`h0, h1`, which should be used in calls to `DWTImpl` for our new wavelet. These replace the forms from equations (5.70)-(5.71) in Example 5.50, which we found for the first piecewise linear wavelet.

Hint: Note that the filter G_0 is unchanged from that of the first piecewise linear wavelet (since ϕ is unchanged when compared to the other wavelets for piecewise linear functions).

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

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

5.7 Summary

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

Chapter 6

Digital images

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

6.1 What is an image?

Before we do computations with images, it is helpful to be clear about what an image really is. Images cannot be perceived unless there is some light present, so we first review superficially what light is.

6.1.1 Light

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

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

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

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

6.1.2 Digital output media

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

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

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

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

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

Fact 6.2 (Resolution). The resolution of a medium is the number of dots per inch (dpi). The number of dots per inch for monitors is usually in the range 70–120, while for printers it is in the range 150–4800 dpi. The horizontal and vertical densities may be different. On a monitor the dots are usually referred to as *pixels* (picture elements).

6.1.3 Digital input media

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

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

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

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

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

6.1.4 Definition of digital image

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

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

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

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

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



(a)



(b)



(c)

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

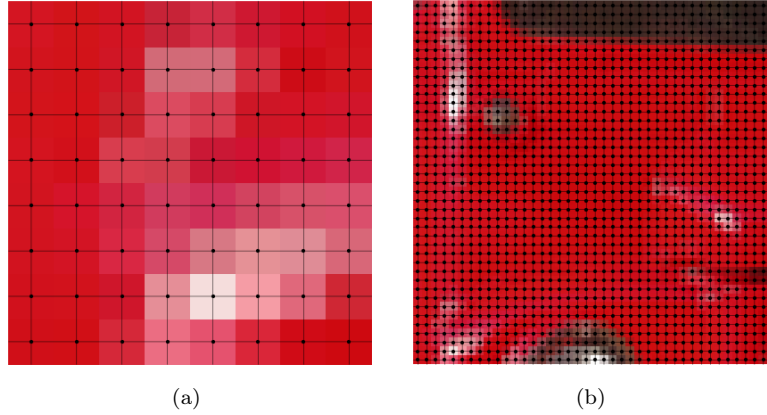


Figure 6.2: Two excerpt of the colour image in figure 6.1. The dots indicate the position of the points (i, j) .

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

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

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

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



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

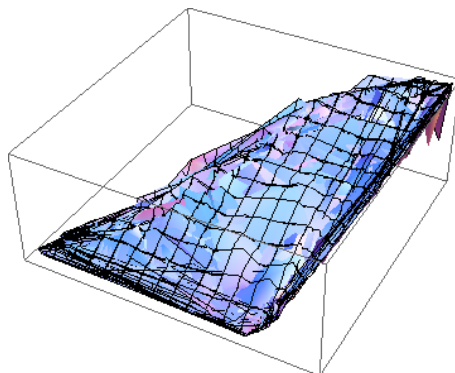


Figure 6.4: A colour image viewed as a parametric surface in space.

6.1.5 Images as surfaces

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

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

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

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

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

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

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

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

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

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

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

6.2 Operations on images

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

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

6.2.1 Images and MATLAB

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

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

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

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

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

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

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

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

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

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

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

Several examples of using this function will be shown below.

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

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

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

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

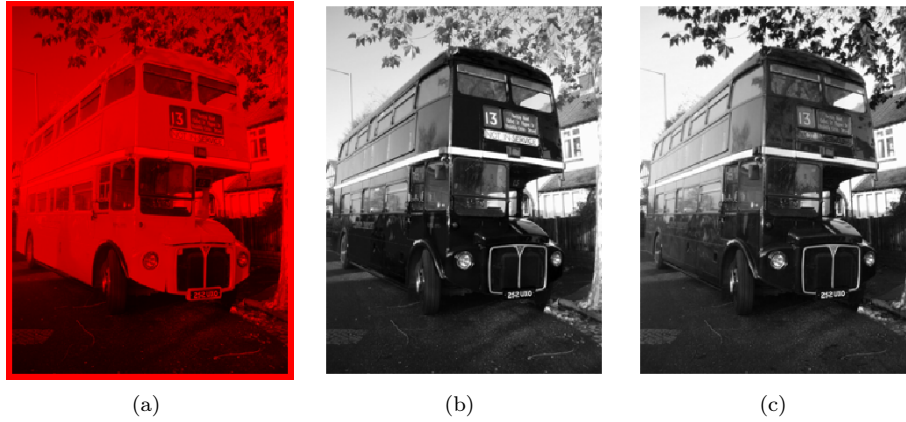



Figure 6.5: The red (a), green (b), and blue (c) components of the colour image in Figure 6.1.

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

The resulting image files are shown in Figure 6.5.

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

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

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

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

Let us sum this up as an algorithm.

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

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

$$q_{i,j} = \frac{\hat{q}_{i,j}}{\max_{k,l} \hat{q}_{k,l}}.$$

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

$$q_{i,j} = \frac{\hat{q}_{i,j}}{\max_{k,l} \hat{q}_{k,l}}.$$

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

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

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

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

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

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

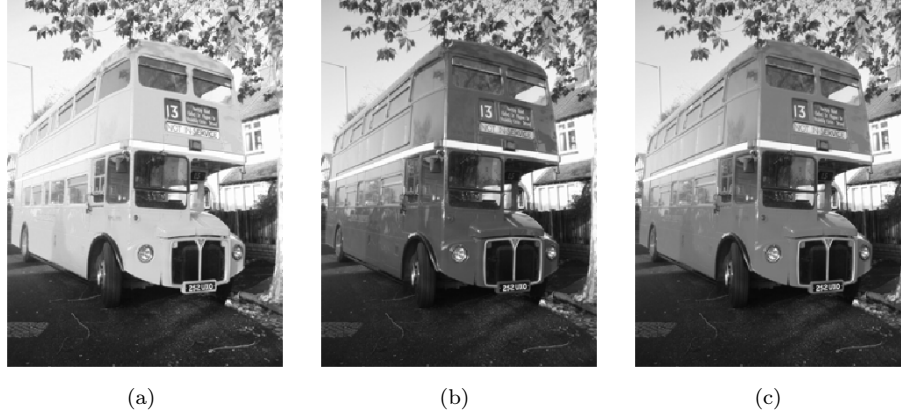


Figure 6.6: Alternative ways to convert the colour image in Figure 6.1 to a grey level image. In (a) each colour triple has been replaced by its maximum, in (b) each colour triple has been replaced by its sum and the result mapped to $[0, 1]$, while in (c) each triple has been replaced by its length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

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

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

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

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

Figure 6.8 shows some examples. The functions in the left plot have quite large derivatives near $x = 0.5$ and will therefore increase the contrast in images with a concentration of intensities with value around 0.5. The functions are all on the form

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

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

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

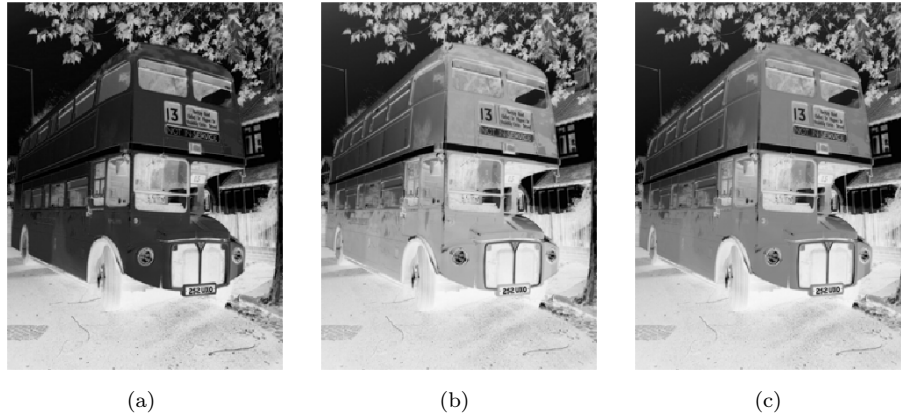


Figure 6.7: The negative versions of the corresponding images in figure 6.6.

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

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

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

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

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

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

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

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

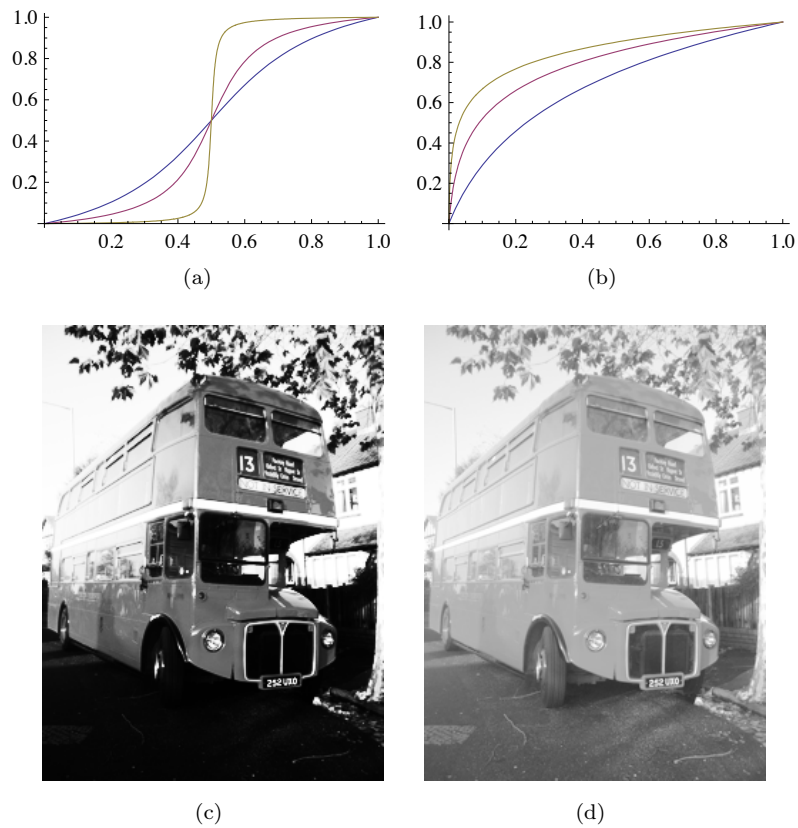


Figure 6.8: The plots in (a) and (b) show some functions that can be used to improve the contrast of an image. In (c) the middle function in (a) has been applied to the intensity values of the image in figure 6.6c, while in (d) the middle function in (b) has been applied to the same image.



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

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

Consider the array of numbers given by

$$\frac{1}{16} \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}. \quad (6.3)$$

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

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

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

A larger filter is given by the array

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

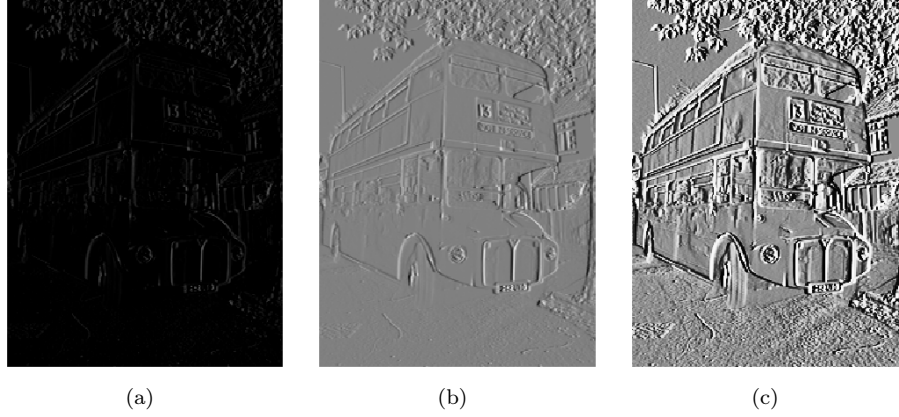


Figure 6.10: The image in (a) shows the partial derivative in the x -direction for the image in 6.6. In (b) the intensities in (a) have been normalised to $[0, 1]$ and in (c) the contrast as been enhanced with the function f_{50} , equation 6.1.

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

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

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

Observation 6.19. Let $P = (p_{i,j})_{i,j=1}^{m,n}$ be a given image. The partial derivative $\partial P / \partial x$ of the image can be computed with the computational molecule

$$\frac{1}{2} \begin{pmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}. \quad (6.6)$$

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

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

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

Observation 6.20. Let $P = (p_{i,j})_{i,j=1}^{m,n}$ be a given image. The partial derivative $\partial P/\partial y$ of the image can be computed with the computational molecule

$$\frac{1}{2} \begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}. \quad (6.7)$$

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

The gradient. The gradient of a scalar function is often used as a measure of the size of the first derivative. The gradient is defined by the vector

$$\nabla P = \left(\frac{\partial P}{\partial x}, \frac{\partial P}{\partial y} \right),$$

so its length is given by

$$|\nabla P| = \sqrt{\left(\frac{\partial P}{\partial x} \right)^2 + \left(\frac{\partial P}{\partial y} \right)^2}.$$

When the two first derivatives have been computed it is a simple matter to compute the gradient vector and its length; the resulting is shown as an image in figure 6.11c.

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

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

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



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

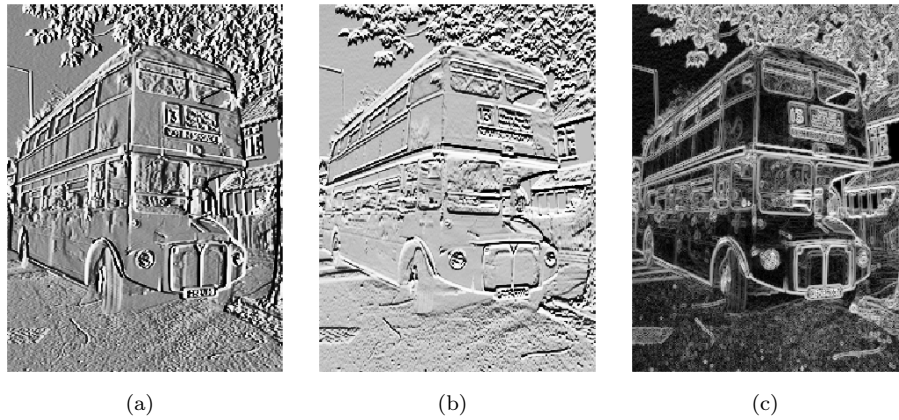


Figure 6.12: The first-order partial derivatives in the x -direction (a) and y -direction (b), and the length of the gradient (c). In all images, the computed numbers have been normalised and the contrast enhanced.

6.2.2 Comparing the first derivatives

Figure 6.12 shows the two first-order partial derivatives and the gradient. If we compare the two partial derivatives we see that the x -derivative seems to emphasise vertical edges while the y -derivative seems to emphasise horizontal edges. This is precisely what we must expect. The x -derivative is large when the difference between neighbouring pixels in the x -direction is large, which is the case across a vertical edge. The y -derivative enhances horizontal edges for a similar reason.

The gradient contains information about both derivatives and therefore emphasises edges in all directions. It also gives a simpler image since the sign of the derivatives has been removed.

6.2.3 Second-order derivatives

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

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

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

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

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

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

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

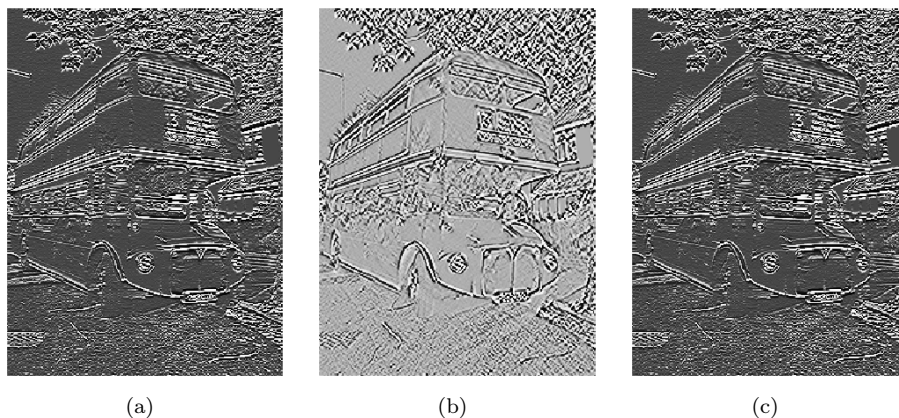


Figure 6.13: The second-order partial derivatives in the x -direction (a) and xy -direction (b), and the y -direction (c). In all images, the computed numbers have been normalised and the contrast enhanced.

Exercises for section 6.2

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

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

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

- a. Write a function `contrastadjust2` which instead uses the function 6.1 to increase the contrast.
- b. Generate the image in Figure 6.8(c) on your own by using your code from Exercise 2, and instead calling the function `contrastadjust2`.

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

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

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

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



Figure 6.14: Secret message

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

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

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

6.3 Adaptations to image processing

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

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

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

6.3.1 Lossless coding

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

6.3.2 Quantization

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

6.3.3 Preprocessing

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

6.3.4 Tiles, blocks, and error resilience

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

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

6.3.5 Metadata

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

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

6.4 Summary

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

Chapter 7

Definition and properties of tensor products

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

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

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

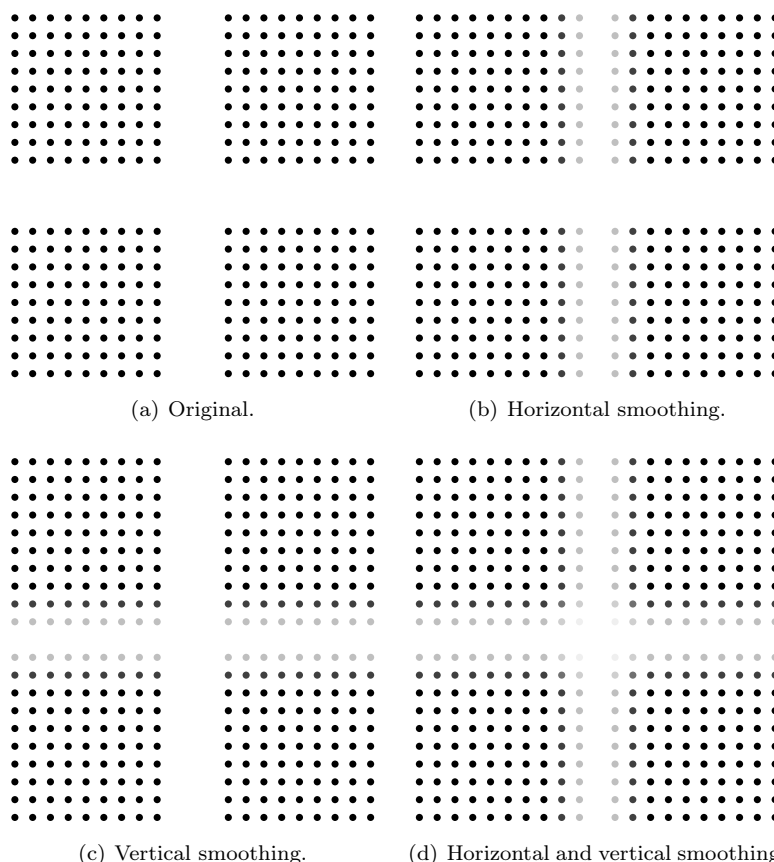


Figure 7.1: The results of smoothing an image with the filter $\{1/4, \underline{1/2}, 1/4\}$ horizontally, vertically, and both. The pixels are shown as disks with intensity corresponding to the pixel values.

7.1 The tensor product of vectors

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

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

one-dimensional operations we can do:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Since

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

we find that

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

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

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

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

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

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

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

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

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

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

Proof. We have that

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

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

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

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

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

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

$$(S \otimes T)X = SXT^T = (T(SX)^T)^T. \quad (7.1)$$

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

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

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

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

```

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

```

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

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

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

Proof. By Theorem 7.5 we have that

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

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

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

$$S \otimes T = (S \otimes I)(I \otimes T) = (I \otimes T)(S \otimes I). \quad (7.2)$$

Moreover, from Theorem 7.5 it follows that

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

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

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

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

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

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

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

Exercises for Section 7.1

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

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

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

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

Ex. 5 — Show that the mapping $F(\mathbf{x}, \mathbf{y}) = \mathbf{x} \otimes \mathbf{y}$ is bi-linear, i.e. that $F(\alpha\mathbf{x}_1 + \beta\mathbf{x}_2, \mathbf{y}) = \alpha F(\mathbf{x}_1, \mathbf{y}) + \beta F(\mathbf{x}_2, \mathbf{y})$, and $F(\mathbf{x}, \alpha\mathbf{y}_1 + \beta\mathbf{y}_2) = \alpha F(\mathbf{x}, \mathbf{y}_1) + \beta F(\mathbf{x}, \mathbf{y}_2)$.

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

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

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

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

7.2 Change of bases in tensor products

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

Theorem 7.10. If $\mathcal{B}_1 = \{\mathbf{v}_i\}_{i=0}^{M-1}$ is a basis for \mathbb{R}^M , and $\mathcal{B}_2 = \{\mathbf{w}_j\}_{j=0}^{N-1}$ is a basis for \mathbb{R}^N , then $\{\mathbf{v}_i \otimes \mathbf{w}_j\}_{(i,j)=(0,0)}^{(M-1,N-1)}$ is a basis for $L_{M,N}(\mathbb{R})$. We denote this basis by $\mathcal{B}_1 \otimes \mathcal{B}_2$.

Proof. Suppose that $\sum_{(i,j)=(0,0)}^{(M-1,N-1)} \alpha_{i,j}(\mathbf{v}_i \otimes \mathbf{w}_j) = \mathbf{0}$. Setting $\mathbf{h}_i = \sum_{j=0}^{N-1} \alpha_{i,j} \mathbf{w}_j$ we get

$$\sum_{j=0}^{N-1} \alpha_{i,j}(\mathbf{v}_i \otimes \mathbf{w}_j) = \mathbf{v}_i \otimes \left(\sum_{j=0}^{N-1} \alpha_{i,j} \mathbf{w}_j \right) = \mathbf{v}_i \otimes \mathbf{h}_i.$$

where we have used the bi-linearity of the tensor product mapping $(\mathbf{x}, \mathbf{y}) \rightarrow \mathbf{x} \otimes \mathbf{y}$ (Exercise 7.1.5). This means that

$$\mathbf{0} = \sum_{(i,j)=(0,0)}^{(M-1,N-1)} \alpha_{i,j}(\mathbf{v}_i \otimes \mathbf{w}_j) = \sum_{i=0}^{M-1} \mathbf{v}_i \otimes \mathbf{h}_i = \sum_{i=0}^{M-1} \mathbf{v}_i \mathbf{h}_i^T.$$

Column k in this matrix equation says $\mathbf{0} = \sum_{i=0}^{M-1} h_{i,k} \mathbf{v}_i$, where $h_{i,k}$ are the components in \mathbf{h}_i . By linear independence of the \mathbf{v}_i we must have that $h_{0,k} = h_{1,k} = \dots = h_{M-1,k} = 0$. Since this applies for all k , we must have that all $\mathbf{h}_i = \mathbf{0}$. This means that $\sum_{j=0}^{N-1} \alpha_{i,j} \mathbf{w}_j = \mathbf{0}$ for all i , from which it follows by linear independence of the \mathbf{w}_j that $\alpha_{i,j} = 0$ for all j , and for all i . This means that $\mathcal{B}_1 \otimes \mathcal{B}_2$ is a basis. \square

In particular, as we have already seen, the standard basis for $L_{M,N}(\mathbb{R})$ can be written $\mathcal{E}_{M,N} = \mathcal{E}_M \otimes \mathcal{E}_N$. This is the basis for a useful convention: For a tensor product the bases are most naturally indexed in two dimensions, rather than the usual sequential indexing. This difference translates also to the meaning

of coordinate vectors, which now are more naturally thought of as coordinate matrices:

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

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

Theorem 7.12 (Change of coordinates in tensor products). Assume that $\mathcal{B}_1, \mathcal{C}_1$ are bases for \mathbb{R}^M , and $\mathcal{B}_2, \mathcal{C}_2$ are bases for \mathbb{R}^N , and that S is the change of coordinates matrix from \mathcal{C}_1 to \mathcal{B}_1 , and that T is the change of coordinates matrix from \mathcal{C}_2 to \mathcal{B}_2 . Both $\mathcal{B}_1 \otimes \mathcal{B}_2$ and $\mathcal{C}_1 \otimes \mathcal{C}_2$ are bases for $L_{M,N}(\mathbb{R})$, and if X is the coordinate matrix in $\mathcal{C}_1 \otimes \mathcal{C}_2$, and Y the coordinate matrix in $\mathcal{B}_1 \otimes \mathcal{B}_2$, then

$$Y = SXT^T. \quad (7.3)$$

Proof. Let \mathbf{c}_{ki} be the i 'th basis vector in \mathcal{C}_k , \mathbf{b}_{ki} the i 'th basis vector in \mathcal{B}_k , $k = 1, 2$. Since any change of coordinates is linear, it is enough to show that it coincides with $X \rightarrow SXT^T$ on the basis $\mathcal{C}_1 \otimes \mathcal{C}_2$. The basis vector $\mathbf{c}_{1i} \otimes \mathbf{c}_{2j}$ has coordinate vector $X = \mathbf{e}_i \otimes \mathbf{e}_j$ in $\mathcal{C}_1 \otimes \mathcal{C}_2$. With the mapping $X \rightarrow SXT^T$ this is sent to

$$SXT^T = S(\mathbf{e}_i \otimes \mathbf{e}_j)T^T = \text{col}_i(S)\text{row}_j(T^T).$$

On the other hand, since column i in S is the coordinates of \mathbf{c}_{1i} in the basis \mathcal{B}_1 , and column j in T is the coordinates of \mathbf{c}_{2j} in the basis \mathcal{B}_2 , we can write

$$\begin{aligned} \mathbf{c}_{1i} \otimes \mathbf{c}_{2j} &= \left(\sum_k S_{k,i} \mathbf{b}_{1k} \right) \otimes \left(\sum_l T_{l,j} \mathbf{b}_{2l} \right) = \sum_{k,l} S_{k,i} T_{l,j} (\mathbf{b}_{1k} \otimes \mathbf{b}_{2l}) \\ &= \sum_{k,l} S_{k,i} (T^T)_{j,l} (\mathbf{b}_{1k} \otimes \mathbf{b}_{2l}) = \sum_{k,l} (\text{col}_i(S)\text{row}_j(T^T))_{k,l} (\mathbf{b}_{1k} \otimes \mathbf{b}_{2l}) \end{aligned}$$

we see that the coordinate vector of $\mathbf{c}_{1i} \otimes \mathbf{c}_{2j}$ in the basis $\mathcal{B}_1 \otimes \mathcal{B}_2$ is $\text{col}_i(S)\text{row}_j(T^T)$. In other words, change of coordinates coincides with $X \rightarrow SXT^T$, and the proof is done. \square

In both cases of tensor products of matrices and change of coordinates in tensor products, we see that we need to compute the mapping $X \rightarrow SXT^T$. This means that we can restate Theorem 7.6 for change of coordinates as follows:

Theorem 7.13 (Implementation of change of coordinates in tensor products). The change of coordinates from $\mathcal{C}_1 \otimes \mathcal{C}_2$ to $\mathcal{B}_1 \otimes \mathcal{B}_2$ can be implemented as follows:

1. For every column in the coordinate matrix in $\mathcal{C}_1 \otimes \mathcal{C}_2$, perform a change of coordinates from \mathcal{C}_1 to \mathcal{B}_1 .
2. Transpose the resulting matrix.
3. For every column in the resulting matrix, perform a change of coordinates from \mathcal{C}_2 to \mathcal{B}_2 .
4. Transpose the resulting matrix.

We can reuse the algorithm from the previous section to implement this. In the following operations on images, we will visualize the pixel values in an image as coordinates in the standard basis, and perform a change of coordinates.

Example 7.14 (Change of coordinates with the DFT). The DFT is one particular change of coordinates which we have considered. The DFT was the change of coordinates from the standard basis to the Fourier basis. The corresponding change of coordinates in a tensor product is obtained by substituting with the DFT as the function implementing change of coordinates in Theorem 7.13. The change of coordinates in the opposite direction is obtained by using the IDFT instead of the DFT.

Modern image standards do typically not apply a change of coordinates to the entire image. Rather one splits the image into smaller squares of appropriate size, called blocks, and perform change of coordinates independently for each block. With the JPEG standard, the blocks are always 8×8 . It is of course not a coincidence that a power of 2 is chosen here, since the DFT takes a simplified form in case of powers of 2.

The DFT values express frequency components. The same applies for the two-dimensional DFT and thus for images, but frequencies are now represented in two different directions. The thing which actually provides compression in many image standards is that frequency components which are small are set to 0. This corresponds to neglecting frequencies in the image which have small contributions. This type of lossy compression has little effect on the human perception of the image, if we use a suitable neglection threshold.

In Figure 7.3 we have applied the two-dimensional DFT to our test image. We have then neglected DFT coefficients which are below certain thresholds, and transformed the samples back to reconstruct the image. When increasing the threshold, the image becomes more and more unclear, but the image is quite clear in the first case, where as much as more than 90% of the samples have been neglected. The blocking effect at the block boundaries is clearly visible.

Example 7.15 (Change of coordinates with the DCT). Similarly to the DFT, the DCT was the change of coordinates from the standard basis to what we

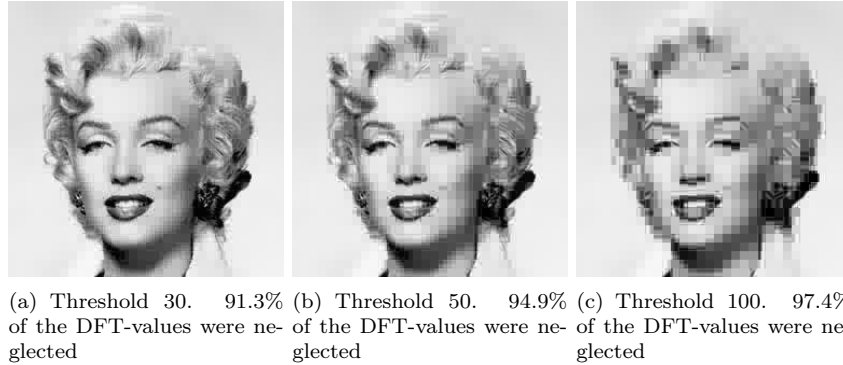


Figure 7.2: The effect on an image when it is transformed with the DFT, and the DFT-coefficients below a certain threshold were neglected.

called the DCT basis. The DCT is used more than the DFT in image processing. Change of coordinates in tensor products between the standard basis and the DCT basis is obtained by substituting with the DCT and the IDCT in Theorem 7.13. The JPEG standard actually applies a two-dimensional DCT to the blocks of size 8×8 , it does not apply the two-dimensional DFT.

If we follow the same strategy for the DCT as for the DFT, so that we neglect DCT-coefficients which are below a given threshold, we get the images shown in Figure 7.3. We see similar effects as with the DFT, but it seems that the latter images are a bit clearer, verifying that the DCT is a better choice than the DFT. It is also interesting to compare with what happens when we drop splitting the image into blocks. Of course, when we neglect many of the DCT-coefficients, we should see some artifacts, but there is no reason to believe that these should be at the old block boundaries. The new artifacts can be seen in Figure 7.4, where the same thresholds as before have been used. Clearly, the new artifacts take a completely different shape.

In the exercises you will be asked to implement functions which generate the images shown in these examples.

Exercises for Section 7.2

Ex. 1 — Implement functions

```
function newx=FFT2Impl(x)
function x=IFF2Impl(newx)
function newx=DCT2Impl(x)
function x=IDCT2Impl(newx)
```

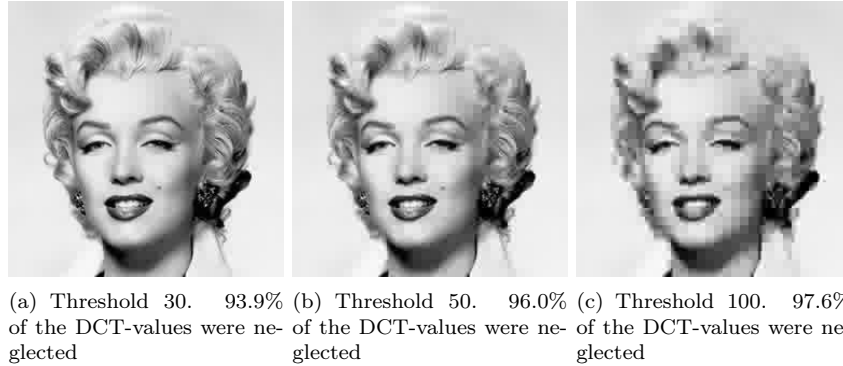


Figure 7.3: The effect on an image when it is transformed with the DCT, and the DCT-coefficients below a certain threshold were neglected.

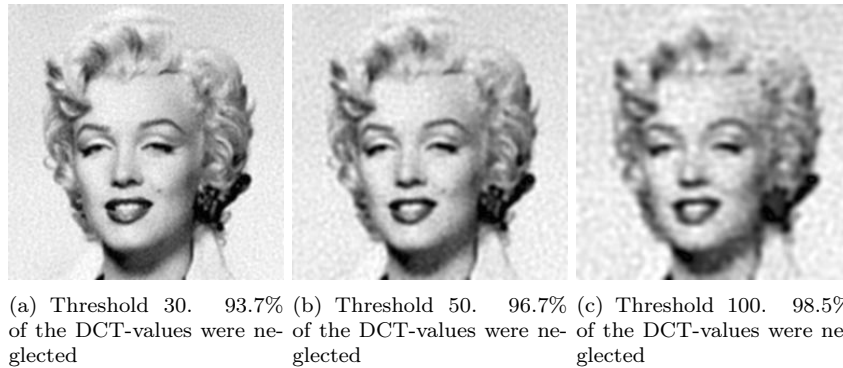


Figure 7.4: The effect on an image when it is transformed with the DCT, and the DCT-coefficients below a certain threshold were neglected. The image has not been split into blocks here.

which implement the two-dimensional DCT, FFT, and their inverses as described in this section. Base your code on the algorithm at the end of Section 7.1.

Ex. 2 — Implement functions

```
function samples=transform2jpeg(x)
function samples=transform2invjpeg(x)
```

which splits the image into blocks of size 8×8 , and performs the DCT2/IDCT2 on each block. Finally run the code

```
function showDCThigher(threshold)
    img = double(imread('mm.gif','gif'));
    newimg=transform2jpeg(img);
    thresholdmatr=(abs(newimg)>=threshold);
    zeroedout=size(img,1)*size(img,2)-sum(sum(thresholdmatr));
    newimg=transform2invjpeg(newimg.*thresholdmatr);
    imageview(abs(newimg));
    fprintf('%i percent of samples zeroed out\n',...
        100*zeroedout/(size(img,1)*size(img,2)));
```

for different threshold parameters, and check that this reproduces the test images of this section, and prints the correct numbers of values which have been neglected (i.e. which are below the threshold) on screen.

7.3 Summary

We defined the tensor product, and saw how this could be used to define operations on images in a similar way to how we defined operations on sound. It turned out that the tensor product construction could be used to construct some of the operations on images we looked at in the previous chapter, which now could be factorized into first filtering the columns in the image, and then filtering the rows in the image. We went through an algorithm for computing the tensor product, and established how we could perform change of coordinates in tensor products. This enables us to define two-dimensional extensions of the DCT and the DFT and their inverses, and we used these extensions to experiment on images.

Chapter 8

Tensor products in a wavelet setting

In Chapter 7 we defined tensor products in terms of vectors, and we saw that the tensor product of two vectors is in fact a matrix. The same construction can be applied to other vector spaces, in particular to vector spaces that are function spaces. As we will see, the tensor product of two univariate function spaces will be a space of functions in two variables. Recall that wavelets are defined in terms of function spaces, so this construction will allow us to define tensor products of wavelets. Through this we will be able to define wavelet transforms that can be applied to images.

Definition 8.1 (Tensor product of function spaces). Let V and W be two vector spaces of functions defined on the intervals $[0, M)$ and $[0, N)$, respectively, and suppose that $f_1 \in V$ and $f_2 \in W$. The tensor product of f_1 and f_2 , denoted $f_1 \otimes f_2$, denotes the function in two variables defined on $[0, M) \times [0, N)$ given by $f_1(t_1)f_2(t_2)$. The function $f_1 \otimes f_2$ is also referred to as the separable extension of f_1 and f_2 to two variables. The tensor product of the two spaces $V \otimes W$ denotes the set of all functions in two variables defined on $[0, M) \times [0, N)$ and on the form $f_1(t_1)f_2(t_2)$, where $f_1 \in V$ and $f_2 \in W$.

We will always assume that the spaces V and W consist of functions which are at least integrable. In this case $V \otimes W$ is also an inner product space, with the inner product given by a double integral,

$$\langle f, g \rangle = \int_0^N \int_0^M f(t_1, t_2)g(t_1, t_2)dt_1dt_2. \quad (8.1)$$

In particular, this says that

$$\begin{aligned}\langle f_1 \otimes f_2, g_1 \otimes g_2 \rangle &= \int_0^N \int_0^M f_1(t_1) f_2(t_2) g_1(t_1) g_2(t_2) dt_1 dt_2 \\ &= \int_0^M f_1(t_1) g_1(t_1) dt_1 \int_0^N f_2(t_2) g_2(t_2) dt_2 = \langle f_1, g_1 \rangle \langle f_2, g_2 \rangle.\end{aligned}\tag{8.2}$$

This means that for tensor products, a double integral can be computed as the product of two one-dimensional integrals.

The tensor product space defined in Definition 8.1 is useful for approximation of functions of two variables if each of the two spaces of univariate functions have good approximation properties.

Idea 8.2. If the spaces V and W can be used to approximate functions in one variable, then $V \otimes W$ can be used to approximate functions in two variables.

We will not state this precisely, but just consider some important examples.

Example 8.3. Let $V = W$ be the space of all polynomials of finite degree. We know that V can be used for approximating many kinds of functions, such as continuous functions, for example by Taylor series. The tensor product $V \otimes V$ consists of all functions on the form $\sum_{i,j} \alpha_{i,j} t_1^i t_2^j$. It turns out that polynomials in several variables have approximation properties analogous to univariate polynomials.

Example 8.4. Let $V = W = V_{N,T}$ be the N th order Fourier space which is spanned by the functions

$$e^{-2\pi i N t/T}, \dots, e^{-2\pi i t/T}, 1, e^{2\pi i t/T}, \dots, e^{2\pi i N t/T}$$

The tensor product space $V \otimes V$ now consists of all functions on the form $\sum_{k,l=0}^N \alpha_{k,l} e^{2\pi i k t_1/T} e^{2\pi i l t_2/T}$. One can show that this space has approximation properties similar to $V_{N,T}$. This is the basis for the theory of Fourier series in two variables.

In the following we think of $V \otimes W$ as a space which can be used for approximating a general class of functions. By associating a function with the vector of coordinates relative to some basis, and a matrix with a function in two variables, we have the following parallel to Theorem 7.10:

Theorem 8.5. If $\{f_i\}_{i=0}^{M-1}$ is a basis for V and $\{g_j\}_{j=0}^{N-1}$ is a basis for W , then $\{f_i \otimes g_j\}_{(i,j)=(0,0)}^{(M-1,N-1)}$ is a basis for $V \otimes W$. Moreover, if the bases for V and W are orthogonal/orthonormal, then the basis for $V \otimes W$ is orthogonal/orthonormal.

Proof. The proof is similar to that of Theorem 7.10: if

$$\sum_{(i,j)=(0,0)}^{(M-1,N-1)} \alpha_{i,j}(f_i \otimes g_j) = 0,$$

we define $h_i(t_2) = \sum_{j=0}^{N-1} \alpha_{i,j} g_j(t_2)$. It follows as before that $\sum_{i=0}^{M-1} h_i(t_2) f_i = 0$ for any t_2 , so that $h_i(t_2) = 0$ for any t_2 due to linear independence of the f_i . But then $\alpha_{i,j} = 0$ also, due to linear independence of the g_j . The statement about orthogonality follows from Equation 8.2. \square

We can now define the tensor product of two bases of functions as before: if $\mathcal{B} = \{f_i\}_{i=0}^{M-1}$ and $\mathcal{C} = \{g_j\}_{j=0}^{N-1}$, we set $\mathcal{B} \otimes \mathcal{C} = \{f_i \otimes g_j\}_{(i,j)=(0,0)}^{(M-1,N-1)}$. Coordinate matrices can also be defined as before: if $f(t_1, t_2) = \sum X_{i,j}(f_i \otimes g_j)(t_1, t_2)$, the coordinate matrix of f is the matrix X with elements $X_{i,j}$. Theorem 7.12 can also be proved in the same way in the context of function spaces. We state this as follows:

Theorem 8.6 (Change of coordinates in tensor products of function spaces). Assume that $\mathcal{B}_1, \mathcal{C}_1$ are bases for V , and $\mathcal{B}_2, \mathcal{C}_2$ are bases for W , and that S is the change of coordinates matrix from \mathcal{C}_1 to \mathcal{B}_1 , and that T is the change of coordinates matrix from \mathcal{C}_2 to \mathcal{B}_2 . Both $\mathcal{B}_1 \otimes \mathcal{B}_2$ and $\mathcal{C}_1 \otimes \mathcal{C}_2$ are bases for $V \otimes W$, and if X is the coordinate matrix in $\mathcal{C}_1 \otimes \mathcal{C}_2$, and Y the coordinate matrix in $\mathcal{B}_1 \otimes \mathcal{B}_2$, then

$$Y = SXT^T. \quad (8.3)$$

8.1 Adopting the tensor product terminology to wavelets

In the remaining part of this chapter we will apply the tensor product construction to wavelets. In particular the spaces V, W from Definition 8.1 are defined from function spaces V_m, W_m , constructed from a given wavelet. We can in particular form the tensor products $\phi_{0,n_1} \otimes \phi_{0,n_2}$. We will assume that

1. the first component ϕ_{0,n_1} has period M (so that $\{\phi_{0,n_1}\}_{n_1=0}^{M-1}$ is a basis for the first component space),
2. the second component ϕ_{0,n_2} has period N (so that $\{\phi_{0,n_2}\}_{n_2=0}^{N-1}$ is a basis for the second component space).

When we speak of $V_0 \otimes V_0$ we thus mean an MN -dimensional space with basis $\{\phi_{0,n_1} \otimes \phi_{0,n_2}\}_{(n_1,n_2)=(0,0)}^{(M-1,N-1)}$, where the coordinate matrices are $M \times N$. This difference in the dimension of the two components is done to allow for images where the number of rows and columns may be different. In the following we

will implicitly assume that the component spaces have dimension M and N , to ease notation. If we use that $\phi_{m-1} \oplus \psi_{m-1}$ also is a basis for V_m , we get the following corollary to Theorem 8.5:

Corollary 8.7. Let ϕ, ψ be a scaling function and a mother wavelet. Then the two sets of tensor products given by

$$\phi_m \otimes \phi_m = \{\phi_{m,n_1} \otimes \phi_{m,n_2}\}_{n_1,n_2}$$

and

$$\begin{aligned} & (\phi_{m-1} \oplus \psi_{m-1}) \otimes (\phi_{m-1} \oplus \psi_{m-1}) \\ &= \{\phi_{m-1,n_1} \otimes \phi_{m-1,n_2}, \\ & \quad \phi_{m-1,n_1} \otimes \psi_{m-1,n_2}, \\ & \quad \psi_{m-1,n_1} \otimes \phi_{m-1,n_2}, \\ & \quad \psi_{m-1,n_1} \otimes \psi_{m-1,n_2}\}_{n_1,n_2} \end{aligned}$$

are both bases for $V_m \otimes V_m$. This second basis is orthogonal/orthonormal whenever the first basis is.

From this we observe that while the one-dimensional wavelet decomposition splits V_m into a direct sum of the two vector spaces V_{m-1} and W_{m-1} , the corresponding two-dimensional decomposition splits $V_m \otimes V_m$ into a direct sum of four tensor product vector spaces which deserve individual names.

Definition 8.8. We define the following tensor product spaces:

1. The space $W_m^{(0,1)}$ spanned by $\{\phi_{m,n_1} \otimes \psi_{m,n_2}\}_{n_1,n_2}$. This is also called the 01-subband, or the LH-subband,
2. The space $W_m^{(1,0)}$ spanned by $\{\psi_{m,n_1} \otimes \phi_{m,n_2}\}_{n_1,n_2}$. This is also called the 10-subband, or the HL-subband,
3. The space $W_m^{(1,1)}$ spanned by $\{\psi_{m,n_1} \otimes \psi_{m,n_2}\}_{n_1,n_2}$. This is also called the 11-subband, or the HH-subband.

The names L and H stand for *Low-pass filters* and *High-pass filters*, reflecting the interpretation of the corresponding filters G_0, G_1, H_0, H_1 as lowpass/high-pass filters. The use of the term subbands comes from the interpretation of these filters as being selective on a certain frequency band. The splitting of $V_m \otimes V_m$ into a direct sum of vector spaces can now be summed up as

$$V_m \otimes V_m = (V_{m-1} \otimes V_{m-1}) \oplus W_m^{(0,1)} \oplus W_m^{(1,0)} \oplus W_m^{(1,1)}. \quad (8.4)$$

Also in the setting of tensor products we refer to $V_{m-1} \otimes V_{m-1}$ as the space of low-resolution approximations. The remaining parts, $W_m^{(0,1)} \oplus W_m^{(1,0)} \oplus W_m^{(1,1)}$,

is referred to as the detail space. Note that the coordinate matrix of

$$\sum_{n_1, n_2=0}^{2^{m-1}N} (c_{m-1, n_1, n_2} (\phi_{m-1, n_1} \otimes \phi_{m-1, n_2}) + w_{m-1, n_1, n_2}^{(0,1)} (\phi_{m-1, n_1} \otimes \psi_{m-1, n_2}) + w_{m-1, n_1, n_2}^{(1,0)} (\psi_{m-1, n_1} \otimes \phi_{m-1, n_2}) + w_{m-1, n_1, n_2}^{(1,1)} (\psi_{m-1, n_1} \otimes \psi_{m-1, n_2})) \quad (8.5)$$

in the basis $(\phi_{m-1} \oplus \psi_{m-1}) \otimes (\phi_{m-1} \oplus \psi_{m-1})$ is

$$\left(\begin{array}{cc|cc} c_{m-1,0,0} & \cdots & w_{m-1,0,0}^{(0,1)} & \cdots \\ \vdots & \vdots & \vdots & \vdots \\ \hline w_{m-1,0,0}^{(1,0)} & \cdots & w_{m-1,0,0}^{(1,1)} & \cdots \\ \vdots & \vdots & \vdots & \vdots \end{array} \right). \quad (8.6)$$

We see that the coordinate matrix is split into four submatrices:

- The c_{m-1} -values, i.e. the coordinates for $V_{m-1} \oplus V_{m-1}$. This is the upper left corner in (8.6), and is also called the 00-subband, or the LL-subband.
- The $w_{m-1}^{(0,1)}$ -values, i.e. the coordinates for $W_{m-1}^{(0,1)}$. This is the upper right corner in (8.6), and corresponds to the LH-subband.
- The $w_{m-1}^{(1,0)}$ -values, i.e. the coordinates for $W_{m-1}^{(1,0)}$. This is the lower left corner in (8.6), and corresponds to the HL-subband.
- The $w_{m-1}^{(1,1)}$ -values, i.e. the coordinates for $W_{m-1}^{(1,1)}$. This is the lower right corner in (8.6), and corresponds to the HH-subband.

The $w_{m-1}^{(i,j)}$ -values are as in the one-dimensional situation often referred to as wavelet coefficients. Let us consider the Haar wavelet as an example.

Example 8.9. If V_m is the vector space of piecewise constant functions on any interval of the form $[k2^{-m}, (k+1)2^{-m})$ (as in the piecewise constant wavelet), $V_m \otimes V_m$ is the vector space of functions in two variables which are constant on any square of the form $[k_1 2^{-m}, (k_1+1)2^{-m}) \times [k_2 2^{-m}, (k_2+1)2^{-m})$. Clearly $\phi_{m, k_1} \otimes \phi_{m, k_2}$ is constant on such a square and 0 elsewhere, and these functions are a basis for $V_m \otimes V_m$.

Let us compute the orthogonal projection of $\phi_{1, k_1} \otimes \phi_{1, k_2}$ onto $V_0 \otimes V_0$. Since the Haar wavelet is orthonormal, the basis functions in (8.4) are orthonormal, and we can thus use the orthogonal decomposition formula to find this projection. Clearly $\phi_{1, k_1} \otimes \phi_{1, k_2}$ has different support from all except one of $\phi_{0, n_1} \otimes \phi_{0, n_2}$. Since

$$\langle \phi_{1, k_1} \otimes \phi_{1, k_2}, \phi_{0, n_1} \otimes \phi_{0, n_2} \rangle = 1/2$$

when the supports intersect, we obtain

$$\text{proj}_{V_0 \otimes V_0} \phi_{1,k_1} \otimes \phi_{1,k_2} = \begin{cases} \frac{1}{2}(\phi_{0,k_1/2} \otimes \phi_{0,k_2/2}) & \text{when } k_1, k_2 \text{ are even} \\ \frac{1}{2}(\phi_{0,k_1/2} \otimes \phi_{0,(k_2-1)/2}) & \text{when } k_1 \text{ is even, } k_2 \text{ is odd} \\ \frac{1}{2}(\phi_{0,(k_1-1)/2} \otimes \phi_{0,k_2/2}) & \text{when } k_1 \text{ is odd, } k_2 \text{ is even} \\ \frac{1}{2}(\phi_{0,(k_1-1)/2} \otimes \phi_{0,(k_2-1)/2}) & \text{when } k_1, k_2 \text{ are odd} \end{cases}$$

So, in this case there were 4 different formulas, since there were 4 different combinations of even/odd. Let us also compute the projection onto the orthogonal complement of $V_0 \otimes V_0$ in $V_1 \otimes V_1$, and let us express this in terms of the $\phi_{0,n}, \psi_{0,n}$, like we did in the one-variable case. Also here there are 4 different formulas. When k_1, k_2 are both even we obtain

$$\begin{aligned} & \phi_{1,k_1} \otimes \phi_{1,k_2} - \text{proj}_{V_0 \otimes V_0}(\phi_{1,k_1} \otimes \phi_{1,k_2}) \\ &= \phi_{1,k_1} \otimes \phi_{1,k_2} - \frac{1}{2}(\phi_{0,k_1/2} \otimes \phi_{0,k_2/2}) \\ &= \left(\frac{1}{\sqrt{2}}(\phi_{0,k_1/2} + \psi_{0,k_1/2}) \right) \otimes \left(\frac{1}{\sqrt{2}}(\phi_{0,k_2/2} + \psi_{0,k_2/2}) \right) - \frac{1}{2}(\phi_{0,k_1/2} \otimes \phi_{0,k_2/2}) \\ &= \frac{1}{2}(\phi_{0,k_1/2} \otimes \phi_{0,k_2/2}) + \frac{1}{2}(\phi_{0,k_1/2} \otimes \psi_{0,k_2/2}) \\ &+ \frac{1}{2}(\psi_{0,k_1/2} \otimes \phi_{0,k_2/2}) + \frac{1}{2}(\psi_{0,k_1/2} \otimes \psi_{0,k_2/2}) - \frac{1}{2}(\phi_{0,k_1/2} \otimes \phi_{0,k_2/2}) \\ &= \frac{1}{2}(\phi_{0,k_1/2} \otimes \psi_{0,k_2/2}) + \frac{1}{2}(\psi_{0,k_1/2} \otimes \phi_{0,k_2/2}) + \frac{1}{2}(\psi_{0,k_1/2} \otimes \psi_{0,k_2/2}). \end{aligned}$$

Here we have used the relation $\phi_{1,k_i} = \frac{1}{\sqrt{2}}(\phi_{0,k_i/2} + \psi_{0,k_i/2})$, which we have from our first analysis of the Haar wavelet. Checking the other possibilities we find similar formulas for the projection onto the orthogonal complement of $V_0 \otimes V_0$ in $V_1 \otimes V_1$ when either k_1 or k_2 is odd. In all cases, the formulas use the basis functions for $W_0^{(0,1)}, W_0^{(1,0)}, W_0^{(1,1)}$. These functions are shown in Figure 8.1, together with the function $\phi \otimes \phi \in V_0 \otimes V_0$.

Example 8.10. If we instead use any of the wavelets for piecewise linear functions, the wavelet basis functions are not orthogonal anymore, just as in the one-dimensional case. The new basis functions are shown in Figure 8.2 for the alternative piecewise linear wavelet.

An immediate corollary of Theorem 8.6 is the following:

Corollary 8.11. Let

$$A_m = P_{(\phi_{m-1} \oplus \psi_{m-1}) \leftarrow \phi_m}$$

$$B_m = P_{\phi_m \leftarrow (\phi_{m-1} \oplus \psi_{m-1})}$$

be the stages in the DWT and the IDWT, and let

$$X = (c_{m,i,j})_{i,j} \quad Y = \begin{pmatrix} (c_{m-1,i,j})_{i,j} & (w_{m-1,i,j}^{(0,1)})_{i,j} \\ (w_{m-1,i,j}^{(1,0)})_{i,j} & (w_{m-1,i,j}^{(1,1)})_{i,j} \end{pmatrix} \quad (8.7)$$

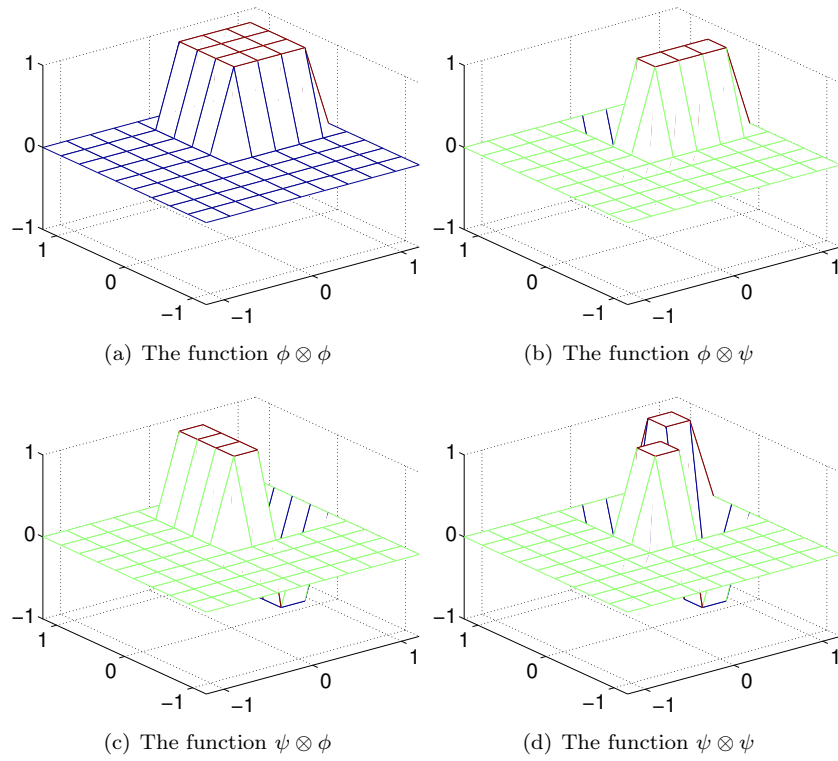


Figure 8.1: The basis functions for $(V_0 \otimes V_0) \oplus W_0^{(0,1)} \oplus W_0^{(1,0)} \oplus W_0^{(1,1)}$ for the Haar wavelet.

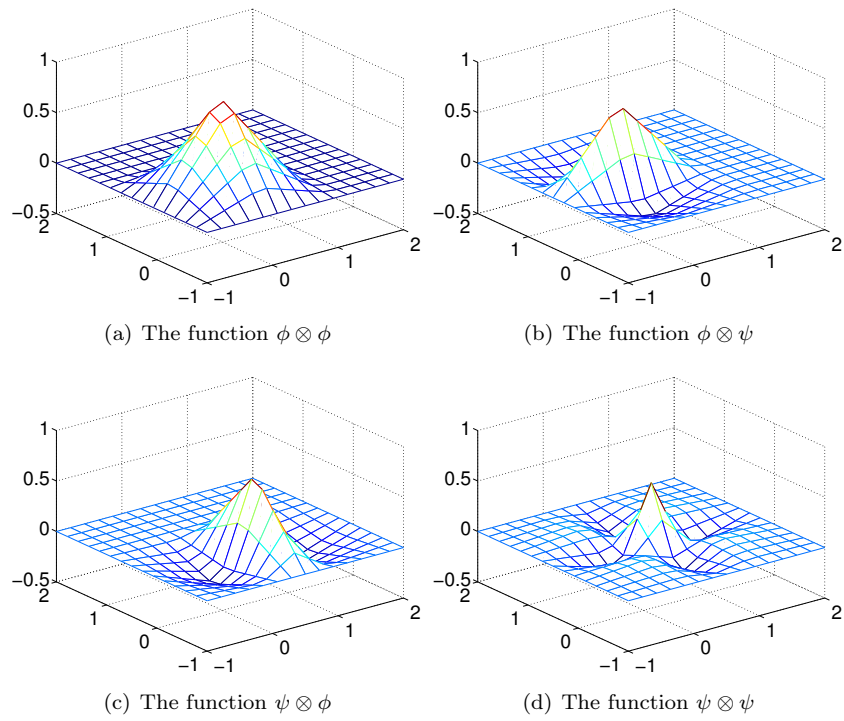


Figure 8.2: The basis functions for $(V_0 \otimes V_0) \oplus W_0^{(0,1)} \oplus W_0^{(1,0)} \oplus W_0^{(1,1)}$ for the alternative piecewise linear wavelet.

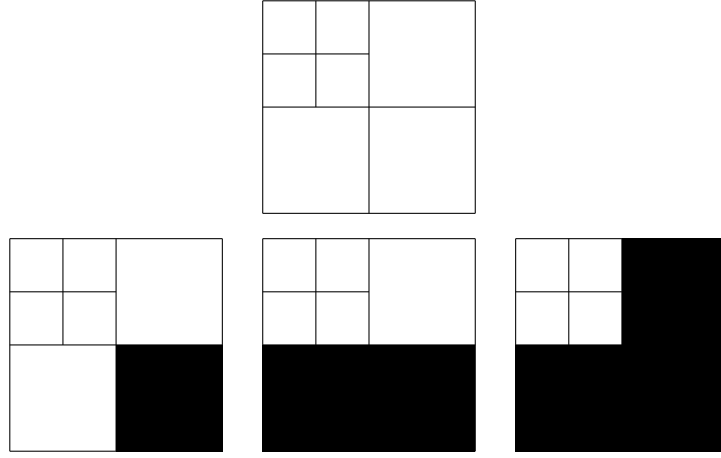


Figure 8.3: Graphical representation of neglecting the wavelet coefficients at the first level. In the first figure, all coefficients are present. Then we remove the ones from $W_1^{(1,1)}$, $W_1^{(1,0)}$, and $W_1^{(0,1)}$, respectively.

be the coordinate matrices in $\phi_m \otimes \phi_m$, and $(\phi_{m-1} \oplus \psi_{m-1}) \otimes (\phi_{m-1} \oplus \psi_{m-1})$, respectively. Then

$$Y = A_m X A_m^T \quad (8.8)$$

$$X = B_m Y B_m^T \quad (8.9)$$

By the m -level two-dimensional DWT/IDWT (or DWT2/IDWT2) we mean the change of coordinates where this is repeated m times as in a DWT/IDWT.

Each stage in DWT2 and IDWT2 can now be implemented by substituting the matrices A_m, B_m above into Theorem 7.13. This implementation can reuse an efficient implementation of the one-dimensional DWT/IDWT. When using many levels of the DWT2, the next stage is applied only to the upper left corner of the matrix, just as the DWT at the next stage only is applied to the first part of the coordinates. At each stage, the upper left corner of the coordinate matrix (which gets smaller at each iteration), is split into four equally big parts. To illustrate this, assume that we have a coordinate matrix, and that we perform the change of basis at two levels, i.e. we start with a coordinate matrix in the basis $\phi_2 \otimes \phi_2$. Figure 8.3 illustrates first the collection of all coordinates, and then the resulting collection of coordinates after removing subbands at the first level successively. The subbands which have been removed are indicated with a black colour. Figure 8.4 illustrates in the same way incremental removal of the subbands at the second level.

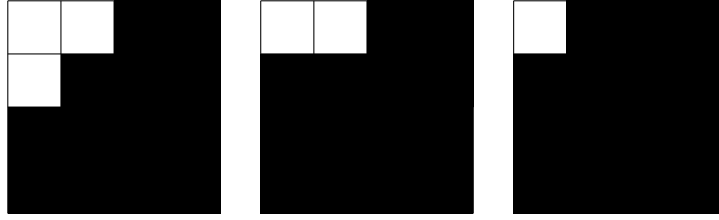


Figure 8.4: Graphical representation of neglecting the wavelet coefficients at the second level. We remove the ones from $W_2^{(1,1)}$, $W_2^{(1,0)}$, and $W_2^{(0,1)}$, respectively.



Figure 8.5: Image of Marilyn Monroe, used in our experiments.

Let us round off this section with some experiments with images using the wavelets we have considered¹. Our theory is applied to images in the following way: We visualize the pixels in the image as coordinates in the basis $\phi_m \otimes \phi_m$ (so that the image has size $(2^m M) \times (2^m N)$), and perform change of coordinates with the DWT2. We can then, just as we did for sound, and for the DCT/DFT-values in images, either set the the part from the $W_k^{(i,j)}$ -spaces (the detail) to zero, or the part from $V_0 \otimes V_0$ (the $M \times N$ -low-resolution approximation) to zero, depending on whether we want to inspect the detail or the low-resolution approximation in the image. Finally we apply the IDWT2 to end up with coordinates in $\phi_m \otimes \phi_m$ again, and display the image with pixel values being these coordinates.

Example 8.12 (Creating thumbnail images). Let us take the sample image of Marilyn Monroe, shown in Figure 8.5, and first use the Haar wavelet. In Exercise 1 you will be asked to implement a function which compute DWT2 for the Haar wavelet. After the DWT2, the upper left submatrices represent the low-resolution approximations from $V_{m-1} \otimes V_{m-1}$, $V_{m-2} \otimes V_{m-2}$, and so on. We can now use the following code to store the low-resolution approximation for $m = 1$:

¹Note also that Matlab has a wavelet toolbox which could be used for these purposes, but we will not go into the usage of this.

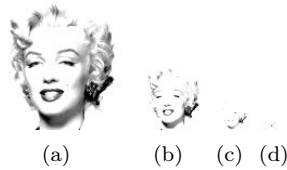


Figure 8.6: The corresponding thumbnail images for the Image of Marilyn Monroe, obtained with a DWT of 1, 2, 3, and 4 levels.

```
img = double(imread('mm.gif','gif'));
[l1,l2]=size(img);
x=DWT2HaarImpl(img,1);
x=x(1:(l1/2),1:(l2/2));
imwrite(uint8(x),'mm1thumbnail.jpg','jpg');
```

In Figure 8.6 the results are shown up to 4 resolutions.

Example 8.13 (Detail and low-resolution approximations with the Haar wavelet). In Exercise 2 you will be asked to implement a function `showDWTlower` which displays the low-resolution approximations to our image test file `mm.gif`, for the Haar wavelet, using functions we implement in the exercises. Let us take a closer look at the images generated. Above we viewed the low-resolution approximation as a smaller image. Let us compare with the image resulting from setting the wavelet detail coefficients to zero, and viewing the result as an image of the same size. In particular, let us neglect the wavelet coefficients as pictured in Figure 8.3 and Figure 8.4. Since the Haar wavelet has few vanishing moments, we should expect that the lower order resolution approximations from V_0 are worse when m increase. Figure 8.7 confirms this for the lower order resolution approximations. Alternatively, we should see that the higher order detail spaces contain more information. In Exercise 3 you will be asked to implement a function `showDWTlowerdifference` which displays the detail components in the image for a given resolution m for the Haar wavelet. The new images when this function is used are shown in Figure 8.8. The black colour indicates values which are close to 0. In other words, most of the coefficients are close to 0, which reflects one of the properties of the wavelet.

Example 8.14 (The alternative piecewise linear wavelet, and neglecting bands in the detail spaces). In Exercise 5 you will be asked to implement a function `showDWTfilterslower` which displays the low-resolution approximations to our image test file `mm.gif`, for any type of wavelet, using functions we implement in the exercises. With this function we can display the result for all the wavelets we have considered up to now, in succession, and at a given resolution, with the following code:

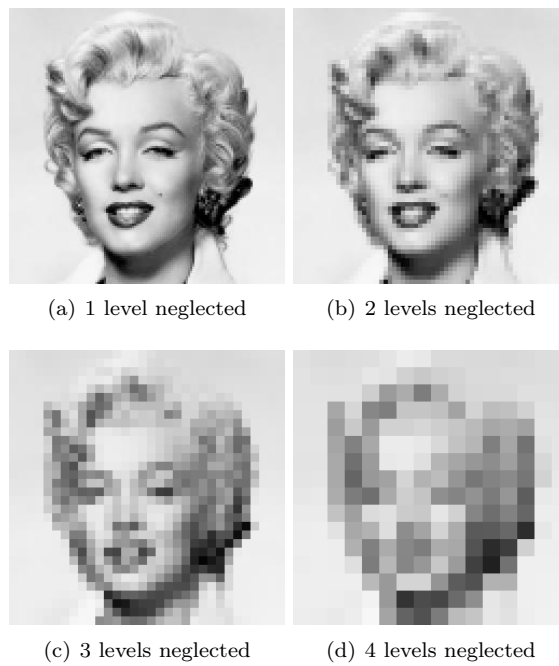


Figure 8.7: Image of Marilyn Monroe, with higher levels of detail neglected for the Haar wavelet.

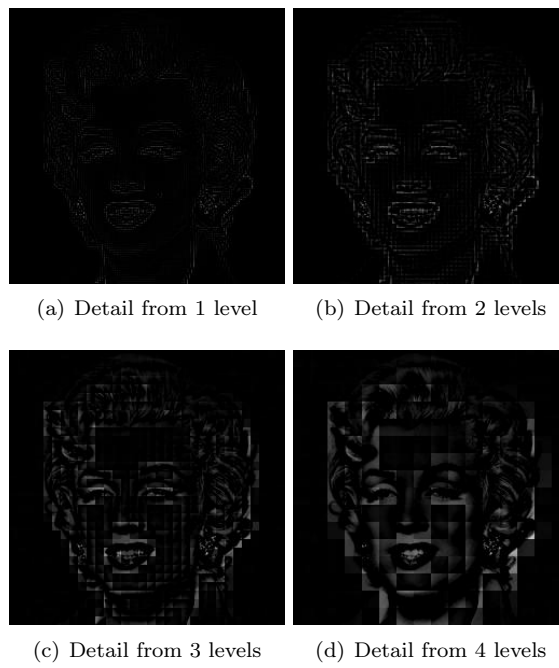


Figure 8.8: The corresponding detail for the images in Figure 8.7, with the Haar wavelet.

```

function showDWTall(m)
disp('Haar wavelet');
showDWTlower(m);
disp('Wavelet for piecewise linear functions');
showDWTfilterslower(m,[sqrt(2)],...
    [sqrt(2) -1/sqrt(2)],...
    [1/sqrt(2) 1/(2*sqrt(2))],...
    [1/sqrt(2)]);
disp('Wavelet for piecewise linear functions, alternative version');
showDWTfilterslower(m,[3/(2*sqrt(2)) 1/(2*sqrt(2)) -1/(4*sqrt(2))],...
    [sqrt(2) -1/sqrt(2)],...
    [1/sqrt(2) 1/(2*sqrt(2))],...
    [3/(4*sqrt(2)) -1/(4*sqrt(2)) -1/(8*sqrt(2))]);

```

The call to `showDWTlower` first displays the result, using the Haar wavelet. The code then moves on to the piecewise linear wavelet and the alternative piecewise linear wavelet. In Example 5.50 we found the parameters `h0,h1,g0,g1` to use for these wavelets. These are then sent as parameters to the function `showDWTfilterslower`, which displays the corresponding image when a wavelet with a given set of filter coefficients are used.

Let us use some other filter than the Haar wavelet. Once the filters are known, and the image has been read from file, the functions `DWT2Impl` and `IDWT2Impl` from Exercise 4, a new image with detail at m levels neglected can be produced with the following code:

```

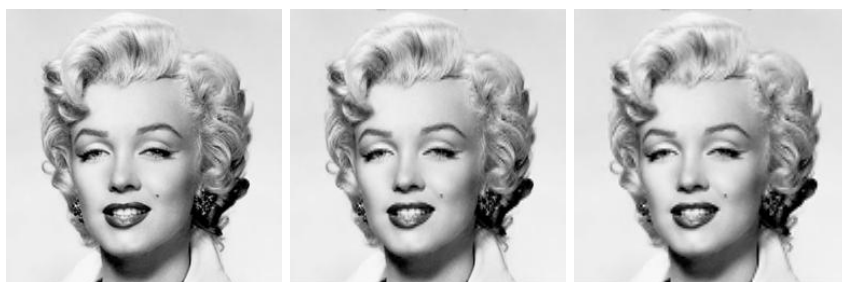
x=DWT2Impl(h0,h1,img,m);
newx=zeros(size(x));
newx(1:(11/2^m),1:(12/2^m))=x(1:(11/2^m),1:(12/2^m));
x=IDWT2Impl(g0,g1,newx,m);

```

We can repeat this for various number of levels, and compare the different images. We can also neglect only parts of the detail, since it at each level is grouped into three bands ($W_m^{(1,1)}$, $W_m^{(1,0)}$, $W_m^{(0,1)}$), contrary to the one-dimensional case. Let us use the alternative piecewise linear wavelet. This is used in the JPEG2000 standard for lossless compression, since the filter coefficients here turned out to be dyadic fractions, which are suitable for lossless operations. The resulting images when the bands on the first level indicated in Figure 8.3 are removed are shown in Figure 8.9. The resulting images when the bands on the second level indicated in Figure 8.4 are removed are shown in Figure 8.10. The image is seen still to resemble the original one, even after two levels of wavelets coefficients have been neglected. This in itself is good for compression purposes, since we may achieve compression simply by dropping the given coefficients. However, if we continue to neglect more levels of coefficients, the result will look poorer. In Figure 8.11 we have also shown the resulting image after the third and fourth level of detail have been neglected. Although we still can see details in the image, the quality in the image is definitely poorer. Although the quality is poorer when we neglect levels of wavelet coefficients, all information is kept if

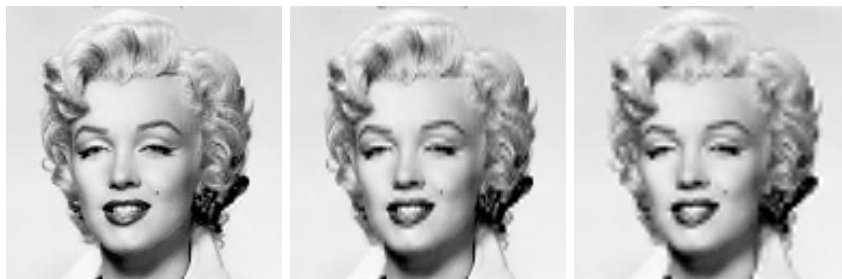


(a) The image unaltered



(b) Resulting image after neglecting detail in $W_1^{(1,1)}$, as illustrated in Figure 8.3(b) (c) Resulting image after neglecting also detail in $W_1^{(1,0)}$, as illustrated in Figure 8.3(c). (d) Resulting image after neglecting also detail in $W_1^{(0,1)}$, as illustrated in Figure 8.3(d).

Figure 8.9: Image of Marilyn Monroe, with various bands of detail at the first level neglected. The alternative piecewise linear wavelet was used.



(a) Resulting image after also neglecting detail in $W_2^{(1,1)}$, as illustrated in Figure 8.10(a). (b) Resulting image after also neglecting detail in $W_2^{(1,0)}$, as illustrated in Figure 8.10(b). (c) Resulting image after also neglecting detail in $W_2^{(0,1)}$, as illustrated in Figure 8.10(c).

Figure 8.10: Image of Marilyn Monroe, with various bands of detail at the second level neglected. The alternative piecewise linear wavelet was used.

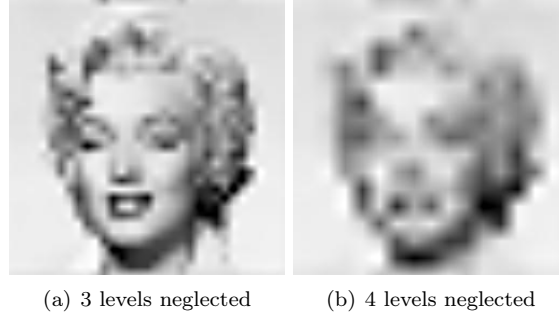


Figure 8.11: Image of Marilyn Monroe, with higher levels of detail neglected. The alternative piecewise linear wavelet was used.

we additionally include the detail/bands. In Figure 8.12, we have shown the corresponding detail for Figure 8.9(d), Figure 8.10(c), and Figure 8.11. Clearly, more detail can be seen in the image when more of the detail is included.

Example 8.15. The JPEG2000 standard uses a more advanced wavelet for lossy compression than the piecewise linear wavelet used for lossless compression. This uses a more advanced scaling function ϕ than the ones we have used, and adds more vanishing moments to it, similarly, to how we did for the alternative piecewise linear wavelet. We will not deduce the expression for this wavelet², only state that it is determined by the filters

$$H_0 = \{0.1562, -0.0985, -0.4569, 1.5589, \underline{3.5221}, 1.5589, -0.4569, -0.0985, 0.1562\}$$

$$H_1 = \{0.0156, -0.0099, \underline{-0.1012}, 0.1909, -0.1012, -0.0099, 0.0156\}.$$

for the DWT, and the filters

$$G_0 = \{-0.0156, -0.0099, 0.1012, \underline{0.1909}, 0.1012, -0.0099, -0.0156\}$$

$$G_1 = \{0.1562, 0.0985, -0.4569, \underline{-1.5589}, 3.5221, -1.5589, -0.4569, 0.0985, 0.1562\}$$

for the IDWT. The length of the filters are 9 and 7 in this case, so that this wavelet is called the CDF 9/7 wavelet (CDF represents the first letters in the names of the inventors of the wavelet). The corresponding frequency responses are

$$\lambda_{G_0}(\omega) = -0.0312 \cos(3\omega), -0.0198 \cos(2\omega), 0.2024 \cos \omega + 0.1909$$

$$\lambda_{H_0}(\omega) = 0.3124 \cos(4\omega) - 0.1970 \cos(3\omega) - 0.9138 \cos(2\omega), 3.1178 \cos \omega + 3.5221.$$

In Figure 8.13 we have plotted these. It is seen that both filters are lowpass

²it can be obtained by hand, but is more easily automated on a computer

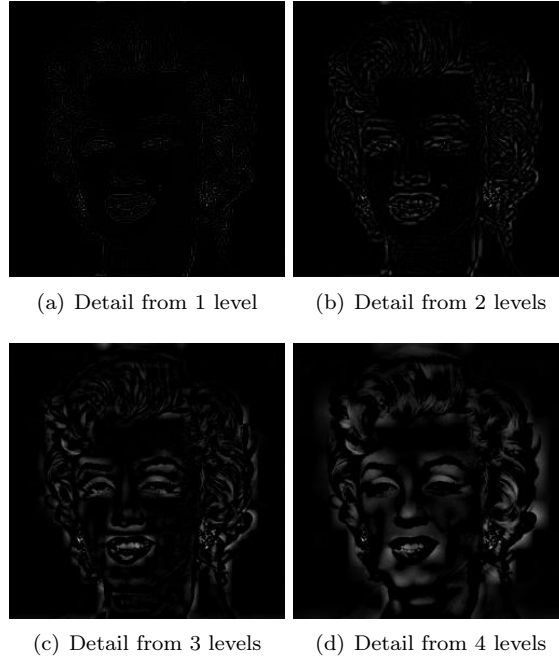


Figure 8.12: The corresponding detail for the image of Marilyn Monroe. The alternative piecewise linear wavelet was used.

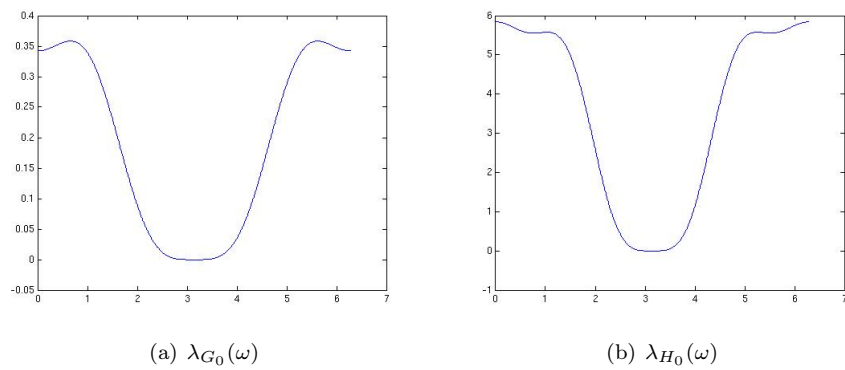


Figure 8.13: The frequency responses for the filters used in lossy compression with JPEG2000.

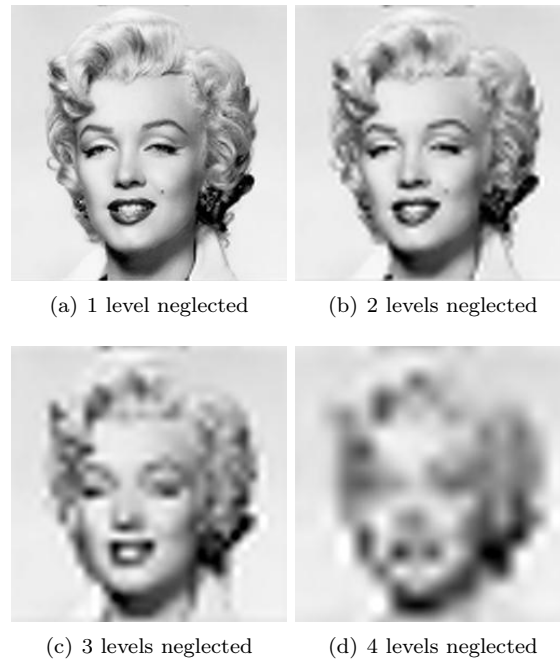


Figure 8.14: Image of Marilyn Monroe, with higher levels of detail neglected. The CDF 9/7 wavelet was used.

filters also here, and that they are closer to an ideal bandpass filter. Although there exist wavelets with better properties when it comes to compression, it can be shown that this wavelet has a good tradeoff between complexity and compression, and is therefore much used. With the CDF 9/7 wavelet, we should see improved images when we discarded the detail in the images. Figure 8.14 confirms this for the lower resolution spaces, while Figure 8.15 confirms this for the higher order detail spaces.

As mentioned, the procedure developed in this section for applying a wavelet transform to an image with the help of the tensor product construction, is adopted in the JPEG2000 standard. This lossy (can also be used as lossless) image format was developed by the Joint Photographic Experts Group and published in 2000. After significant processing of the wavelet coefficients, the final coding with JPEG2000 uses an advanced version of arithmetic coding. At the cost of increased encoding and decoding times, JPEG2000 leads to as much as 20 % improvement in compression ratios for medium compression rates, possibly more for high or low compression rates. The artefacts are less visible than in JPEG and appear at higher compression rates. Although a number of components in JPEG2000 are patented, the patent holders have agreed that the core software should be available free of charge, and JPEG2000 is part of most

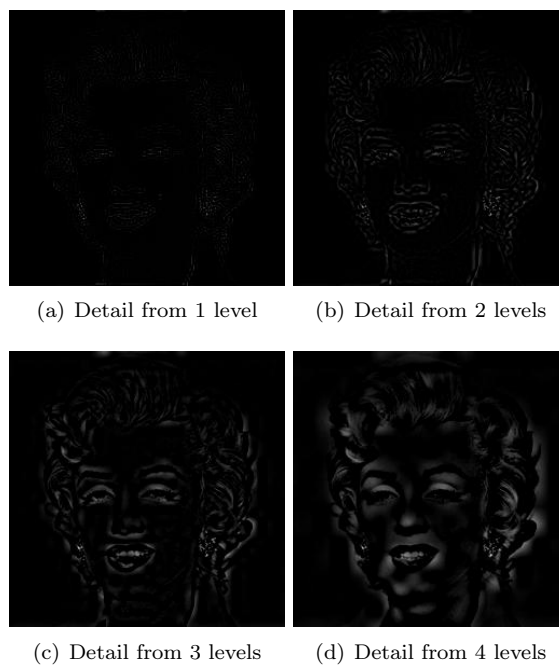


Figure 8.15: The corresponding detail for the image of Marilyn Monroe. The CDF 9/7 wavelet was used.

Linux distributions. However, there appear to be some further, rather obscure, patents that have not been licensed, and this may be the reason why JPEG2000 is not used more. The extension of JPEG2000 files is `.jp2`.

Exercises for Section 8.1

Ex. 1 — Implement functions

```
function xnew=DWT2HaarImpl(x,m)
function x=IDWT2HaarImpl(xnew,m)
```

which implements DWT2 and IDWT2 for the Haar-wavelet.

Ex. 2 — In this exercise we will experiment with applying an m -level DWT to a sound file.

- a. Write a function

```
function showDWTlower(m)
```

which

1. reads the image file `mm.gif`,
 2. performs an m -level DWT2 to the image samples using the function `DWT2HaarImpl`,
 3. sets all wavelet coefficients representing detail to zero (i.e. keep only wavelet coefficients from $V_0 \otimes V_0$),
 4. performs an IDWT2 on the resulting coefficients using the function `IDWT2HaarImpl`,
 5. displays the resuting image.
- b. Run the function `showDWTlower` for different values of m . Describe what you see for different m . degraded? Compare with what you saw with the function `showDCThigher` in Exercise 2, where you performed a DCT on the image samples instead, and set DCT coefficients below a given threshold to zero.
 - c. Do the image samples returned by `showDWTlower` lie in $[0, 255]$?

Ex. 3 — Repeat Exercise 2, but this time instead keep only wavelet coefficients from the detail spaces. Call the new function `showDWTlowerdifference`. What kind of image do you see? Can you recognize the original image in what you see?

Ex. 4 — Implement functions

```
function xnew=DWT2Impl(h0,h1,x,m)
function x=IDWT2Impl(g0,g1,xnew,m)
```

where `DWT2Impl` performs the m -level DWT2 on the image given by `x`, and `IDWT2Impl` performs the m -level IDWT2. The functions should at each stage call `DWTImpl` and `IDWTImpl` with $m = 1$, which you implemented in exercises 4 and 5, and each call to these functions should alter the appropriate upper left submatrix in the coordinate matrix. You can assume that the number of rows and columns both are powers of 2 in this matrix (but they may not be equal).

Ex. 5 — Write a function

```
function showDWTfilterslower(m,h0,h1,g0,g1)
```

which reimplements the function `showDWTlower` from Exercise 2 so that it takes as input the positive parts of the four different filters as in Section 5.6. Look at the result using the different wavelets we have encountered and for different m , using the code from Example 8.13. Can you see any difference from the Haar wavelet? If so, which wavelet gives the best image quality?

Ex. 6 — In this exercise we will change the code in Example 8.13 so that it instead only shows the contribution from the detail spaces.

- a. Reimplement the function you made in Exercise 5 so that it instead shows the contribution from the detail spaces. Call the new function `showDWTfilterslowerdifference`.
- b. In Exercise 3 we implemented a function `showDWTlowerdifference` for looking at the detail/error when the Haar wavelet is used. In the function `showDWTall` from Example 8.13, replace `showDWTlower` and `showDWTfilterslower` with `showDWTlowerdifference` and `showDWTfilterslowerdifference`. Describe the images you see for different m . Try to explain why the images seem to get clearer when you increase m .

8.2 Summary

We extended the tensor product construction to functions by defining the tensor product of functions as a function in two variables. We explained with some examples that this made the tensor product formalism useful for approximation of functions in several variables. We extended the wavelet transform to the tensor product setting, so that it too could be applied to images. We also performed several experiments on our test image, such as creating low-resolution images and neglecting wavelet coefficients. We also used different wavelets, such as the Haar wavelet, the alternative piecewise linear wavelet, and a new wavelet which is much used in lossy compression of images. The experiments confirmed what we previously have proved, that wavelets with many vanishing moments are better suited for compression purposes.

Part III

Nonlinear optimization

Chapter 9

The basics and applications

The problem of minimizing a function of several variables, possibly subject to constraints on these variables, is what optimization is about. So the main problem is easy to state! And, more importantly, such problems arise in many applications in natural science, engineering, economics and business as well as in mathematics itself.

Nonlinear optimization differs from Fourier analysis and wavelet theory in that classical multivariate analysis also is an important ingredient. A recommended book on this, used here at the University of Oslo, is [8] (in Norwegian). It contains a significant amount of fixed point theory, nonlinear equations, and optimization.

There are many excellent books on nonlinear optimization (or nonlinear programming, as it is also called). Some of these books that have influenced these notes are [1, 2, 9, 5, 13, 11]. These are all recommended books for those who want to go deeper into the subject. These lecture notes are particularly influenced by the presentations in [1, 2].

Optimization has its mathematical foundation in linear algebra and multivariate calculus. In analysis the area of convexity is especially important. For the brief presentation of convexity given here the author's own lecture notes [4] (originally from 2001), and the very nice book [14], have been useful sources. But, of course, anyone who wants to learn convexity should study the work by R.T. Rockafellar, see e.g. the classic text [12].

Linear optimization (LP, linear programming) is a special case of nonlinear optimization, but we do not discuss this in any detail here. The reason for this is that we, at the University of Oslo, have a separate course in linear optimization which covers many parts of that subject in some detail.

This first chapter introduces some of the basic concepts in optimization and discusses some applications. Many of the ideas and results that you will find in these lecture notes may be extended to more general linear spaces, even infinite-dimensional. However, to keep life a bit easier and still cover most applications, we will only be working in \mathbb{R}^n .

Due to its character this chapter is a “proof-free zone”, but in the remaining text we usually give full proofs of the main results.

Notation: For $\mathbf{z} \in \mathbb{R}^n$ and $\delta > 0$ define the (closed) ball $\bar{B}(\mathbf{z}; \epsilon) = \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x} - \mathbf{z}\| \leq \epsilon\}$. It consists of all points with distance at most ϵ from \mathbf{z} . Similarly, define the open ball $B(\mathbf{z}; \epsilon) = \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x} - \mathbf{z}\| < \epsilon\}$. A *neighborhood* of \mathbf{z} is a set N containing $B(\mathbf{z}; \epsilon)$ for some $\epsilon > 0$. Vectors are treated as column vectors and they are identified with the corresponding n -tuple, denoted by $\mathbf{x} = (x_1, x_2, \dots, x_n)$. A statement like

$$P(\mathbf{x}) \quad (\mathbf{x} \in H)$$

means that the statement $P(\mathbf{x})$ is true for all $\mathbf{x} \in H$.

9.1 The basic concepts

Optimization deals with finding optimal solutions! So we need to define what this is.

Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be a real-valued function in n variables. The function value is written as $f(\mathbf{x})$, for $\mathbf{x} \in \mathbb{R}^n$, or $f(x_1, x_2, \dots, x_n)$. This is the function we want to minimize (or maximize) and it is often called the *objective function*. Let $\mathbf{x}^* \in \mathbb{R}^n$. Then \mathbf{x}^* is a *local minimum* (or local minimizer) of f if there is an $\epsilon > 0$ such that

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \quad \text{for all } \mathbf{x} \in B(\mathbf{x}^*; \epsilon).$$

So, no point “sufficiently near” \mathbf{x}^* has smaller f -value than \mathbf{x}^* . A *local maximum* is defined similarly, but with the inequality reversed. A stronger notion is that \mathbf{x}^* is a *global minimum* of f which means that

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \quad \text{for all } \mathbf{x} \in \mathbb{R}^n.$$

A *global maximum* satisfies the opposite inequality.

The definition of local minimum has a “variational character”; it concerns the behavior of f near \mathbf{x}^* . Due to this it is perhaps natural that Taylor’s formula, which gives an approximation of f in such a neighborhood, becomes a main tool for characterizing and finding local minima. We present Taylor’s formula, in different versions, in Section 9.3.

An extension of the notion of minimum and maximum is for *constrained* problems where we want, for instance, to minimize $f(\mathbf{x})$ over all \mathbf{x} lying in a given set C . Then $\mathbf{x}^* \in C$ is a *local minimum* of f over the set C , or subject to $\mathbf{x} \in C$ as we shall say, provided no point in C in some neighborhood of \mathbf{x}^* has smaller f -value than \mathbf{x}^* . A similar extension holds for global minimum over C , and for maxima.

Example 9.1. To make these things concrete, consider an example from plane geometry. Consider the point set $C = \{(z_1, z_2) : z_1 \geq 0, z_2 \geq 0, z_1 + z_2 \leq 1\}$ in the plane. We want to find a point $\mathbf{x} = (x_1, x_2) \in C$ which is closest possible to the point $\mathbf{a} = (3, 2)$. This can be formulated as the minimization problem

$$\begin{aligned} &\text{minimize} && (x_1 - 3)^2 + (x_2 - 2)^2 \\ &\text{subject to} && \\ &&& x_1 + x_2 \leq 1 \\ &&& x_1 \geq 0, x_2 \geq 0. \end{aligned}$$

The function we want to minimize is $f(\mathbf{x}) = (x_1 - 3)^2 + (x_2 - 2)^2$ which is a quadratic function. This is the square of the distance between \mathbf{x} and \mathbf{a} ; and minimizing the distance or the square of the distance is equivalent (why?). A minimum here is $\mathbf{x}^* = (1, 0)$. If we instead minimize this function f over \mathbb{R}^2 , the unique global minimum is $\mathbf{x}^* = \mathbf{a} = (3, 2)$. It is useful to study this example and try to solve it geometrically as well as analytically.

In optimization one considers minimization and maximization problems. As

$$\max\{f(\mathbf{x}) : \mathbf{x} \in S\} = -\min\{-f(\mathbf{x}) : \mathbf{x} \in S\}$$

it is clear how to convert a maximization problem into a minimization problem (or vice versa). This transformation may, however, change the properties of the function you work with. For instance, if f is convex (definitions come later!), then $-f$ is not convex (unless f is linear), so rewriting between minimization and maximization may take you out of a class of “good problems”. Note that a minimum or maximum may not exist. A main tool one uses to establish that optimal solutions really exist is the *extreme value theorem* as stated next. You may want to look these notions up in [8].

Theorem 9.2. Let C be a subset of \mathbb{R}^n which is closed and bounded, and let $f : C \rightarrow \mathbb{R}$ be a continuous function.

Then f attains both its (global) minimum and maximum, so these are points $\mathbf{x}^1, \mathbf{x}^2 \in C$ with

$$f(\mathbf{x}^1) \leq f(\mathbf{x}) \leq f(\mathbf{x}^2) \quad (\mathbf{x} \in C).$$

9.2 Some applications

It is useful to see some application areas for optimization. They are many, and here we mention a few in some detail.

9.2.1 Portfolio optimization

The following optimization problem was introduced by Markowitz in order to find an optimal portfolio in a financial market; he later received the Nobel prize in economics¹ (in 1990) for his contributions in this area:

$$\begin{aligned} & \text{minimize} && \alpha \sum_{i,j \leq n} c_{ij} x_i x_j - \sum_{j=1}^n \mu_j x_j \\ & \text{subject to} && \sum_{j=1}^n x_j = 1 \\ & && x_j \geq 0 \quad (j \leq n). \end{aligned}$$

The model may be understood as follows. The decision variables are x_1, x_2, \dots, x_n where x_i is the fraction of a total investment that is made in (say) stock i . Thus one has available a set of stocks in different companies (Statoil, IBM, Apple etc.) or bonds. The fractions x_i must be nonnegative (so we consider no short sale) and add up to 1. The function f to be minimized is

$$f(\mathbf{x}) = \alpha \sum_{i,j \leq n} c_{ij} x_i x_j - \sum_{j=1}^n \mu_j x_j.$$

It can be explained in terms of random variables. Let R_j be the return on stock j , this is a random variable, and let $\mu_j = \mathbb{E}R_j$ be the expectation of R_j . So if X denotes the random variable $X = \sum_{j=1}^n x_j R_j$, which is the return on our portfolio (= mix among investments), then $\mathbb{E}X = \sum_{j=1}^n \mu_j x_j$ which is the second term in f . The minus sign in front explains that we really want to maximize the expected return. The first term in f is there because just looking at expected return is too simple. We want to spread our investments to reduce the risk. The first term in f is the variance of X multiplied by a weight factor α ; the constant c_{ij} is the covariance of R_i and R_j and c_{ii} is the variance of R_i . The covariance of R_i and R_j is defined as $\mathbb{E}(R_i - \mu_i)(R_j - \mu_j)$.

So f is a weighted difference of variance and expected return. This is what we want to minimize. The optimization problem is to minimize a quadratic function subject to linear constraints. We shall discuss theory and methods for such problems later.

In order to use such a model one needs to find good values for all the parameters μ_j and c_{ij} ; this is done using historical data from the stock markets. The weight parameter α is often varied and the optimization problem is solved for each such “interesting” value. This makes it possible to find a so-called efficient frontier of expectation versus variance for optimal solutions.

The Markowitz model is a useful tool for financial investments, and now extensions and variations of the model exist, e.g., by using different ways of measuring risk. All such models involve a balance between risk and expected return.

¹The precise term is “Sveriges Riksbank Prize in Economic Sciences in Memory of Alfred Nobel”

9.2.2 Fitting a model

In many applications one has a mathematical model of some phenomenon where the model has some parameters. These parameters represent a flexibility of the model, and they may be adjusted so that the model explains the phenomenon best possible.

To be more specific consider a model

$$y = F_{\alpha}(\mathbf{x})$$

for some function $F_{\alpha} : \mathbb{R}^m \rightarrow \mathbb{R}$. Here $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_n) \in \mathbb{R}^n$ is a parameter vector (so we may have several parameters). Perhaps there are natural constraints on the parameter, say $\alpha \in A$ for a given set A in \mathbb{R}^n .

For instance, consider

$$y = \alpha_1 \cos x_1 + x_2^{\alpha_2}$$

so here $n = m = 2$, $\alpha = (\alpha_1, \alpha_2)$ and $F_{\alpha}(\mathbf{x}) = \alpha_1 \cos x_1 + x_2^{\alpha_2}$ where (say) $\alpha_1 \in \mathbb{R}$ and $\alpha_2 \in [1, 2]$.

The general model may also be thought of as

$$y = F_{\alpha}(\mathbf{x}) + \text{error}$$

since it is usually a simplification of the system one considers. In statistics one specifies this error term as a random variable with some (partially) known distribution. Sometimes one calls y the *dependent variable* and \mathbf{x} the *explaining variable*. The goal is to understand how y depends on \mathbf{x} .

To proceed, assume we are given a number of observations of the phenomenon given by points

$$(\mathbf{x}^i, y^i) \quad (i = 1, 2, \dots, m).$$

meaning that one has observed y^i corresponding to $\mathbf{x} = \mathbf{x}^i$. We have m such observations. Usually (but not always) we have $m \geq n$. The *model fit problem* is to adjust the parameter α so that the model fits the given data as good as possible. This leads to the optimization problem

$$\text{minimize } \sum_{i=1}^m (y^i - F_{\alpha}(\mathbf{x}^i))^2 \quad \text{subject to } \alpha \in A.$$

The optimization variable is the parameter α . Here the model error is quadratic (corresponding to the Euclidean norm), but other norms are also used.

This optimization problem above is a constrained nonlinear optimization problem. When the function F_{α} depends linearly on α , which often is the case in practice, the problem becomes the classical *least squares approximation problem* which is treated in basic linear algebra courses. The solution is then characterized by a certain linear system of equations, the so-called normal equations.

9.2.3 Maximum likelihood

A very important problem in statistics, arising in many applications, is parameter estimation and, in particular, *maximum likelihood estimation*. It leads to optimization.

Let Y be a “continuous” real-valued random variable with probability density $p_x(y)$. Here x is a parameter (often one uses other symbols for the parameter, like ξ , θ etc.). For instance, if Y is a normal (Gaussian) variable with expectation x and variance 1, then $p_x(y) = \frac{1}{\sqrt{2\pi}} e^{-(y-x)^2/2}$ and

$$\mathbb{P}(a \leq Y \leq b) = \int_a^b \frac{1}{\sqrt{2\pi}} e^{-(y-x)^2/2} dy$$

where \mathbb{P} denotes probability.

Assume Y is the outcome of an experiment, and that we have observed $Y = y$ (so y is a known real number or a vector, if several observations were made). On the basis of y we want to *estimate* the value of the parameter x which “explains” best possible our observation $Y = y$. We have now available the probability density $p_x(\cdot)$. The function $x \rightarrow p_x(y)$, for fixed y , is called the *likelihood* function. It gives the “probability mass” in y as a function of the parameter x . The *maximum likelihood* problem is to find a parameter value x which maximizes the likelihood, i.e., which maximizes the probability of getting precisely y . This is an optimization problem

$$\max_x p_x(y)$$

where y is fixed and the optimization variable is x . We may here add a constraint on x , say $x \in C$ for some set C , which may incorporate possible knowledge of x and assure that $p_x(y)$ is positive for $x \in C$. Often it is easier to solve the equivalent optimization problem of maximizing the logarithm of the likelihood function

$$\max_x \ln p_x(y)$$

This is a nonlinear optimization problem. Often, in statistics, there are several parameters, so $\mathbf{x} \in \mathbb{R}^n$ for some n , and we need to solve a nonlinear optimization problem in several variables, possibly with constraints on these variables. If the likelihood function, or its logarithm, is a concave function, we have (after multiplying by -1) a convex optimization problem. Such problems are easier to solve than general optimization problems. This will be discussed later.

As a specific example assume we have the linear statistical model

$$y = A\mathbf{x} + \mathbf{w}$$

where A is a given $m \times n$ matrix, $\mathbf{x} \in \mathbb{R}^n$ is an unknown parameter, $\mathbf{w} \in \mathbb{R}^m$ is a random variable (the “noise”), and $\mathbf{y} \in \mathbb{R}^m$ is the observed quantity. We assume that the components of \mathbf{w} , i.e., w_1, w_2, \dots, w_m are independent and identically

distributed with common density function p on \mathbb{R} . This leads to the likelihood function

$$p_{\mathbf{x}}(y) = \prod_{i=1}^m p(y_i - \mathbf{a}_i \mathbf{x})$$

where \mathbf{a}_i is the i 'th row in A . Taking the logarithm we obtain the maximum likelihood problem

$$\max \sum_{i=1}^m \ln p(y_i - \mathbf{a}_i \mathbf{x}).$$

In many applications of statistics it is central to solve this optimization problem numerically.

9.2.4 Optimal control problems

Recall that a discrete dynamical system is an equation

$$\mathbf{x}_{t+1} = h_t(\mathbf{x}_t) \quad (t = 0, 1, \dots)$$

where $\mathbf{x}_t \in \mathbb{R}^n$, \mathbf{x}_0 is the initial solution, and h_t is a given function for each t . We here think of t as time and \mathbf{x}_t is the state of the process at time t . For instance, let $n = 1$ and consider $h_t(\mathbf{x}) = a\mathbf{x}$ ($t = 0, 1, \dots$) for some $a \in \mathbb{R}$. Then the solution is $\mathbf{x}_t = a^t \mathbf{x}_0$. Another example is when A is an $n \times n$ matrix, $\mathbf{x}_t \in \mathbb{R}^n$ and $h_t(\mathbf{x}) = A\mathbf{x}$ for each t . Then the solution is $\mathbf{x}_t = A^t \mathbf{x}_0$. For the more general situation, where the system functions h_t may be different, it may be difficult to find an explicit solution for \mathbf{x}_t . Numerically, however, we compute \mathbf{x}_t simply in a for-loop by computing \mathbf{x}_0 , then $\mathbf{x}_1 = f_1(\mathbf{x}_0)$ and then $\mathbf{x}_2 = f_2(\mathbf{x}_1)$ etc.

Now, consider a dynamical system where we may “control” the system in each time step. We restrict the attention to a finite time span, $t = 0, 1, \dots, T$. A proper model is then

$$\mathbf{x}_{t+1} = h_t(\mathbf{x}_t, \mathbf{u}_t) \quad (t = 0, 1, \dots, T-1)$$

where \mathbf{x}_t is the state of the system at time t and the new variable \mathbf{u}_t is the control at time t . We assume $\mathbf{x}_t \in \mathbb{R}^n$ and $\mathbf{u}_t \in \mathbb{R}^m$ for each t (but these things also work if these vectors lie in spaces of different dimensions). Thus, when we choose the controls $\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_{T-1}$ and \mathbf{x}_0 is known, the sequence $\{\mathbf{x}_t\}$ of states is uniquely determined. Next, assume there are given functions $f_t : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$ that we call cost functions. We think of $f_t(\mathbf{x}_t, \mathbf{u}_t)$ as the “cost” at time t when the system is in state \mathbf{x}_t and we choose control \mathbf{u}_t . The *optimal control* problem is

$$\begin{aligned} & \text{minimize} && f_T(\mathbf{x}_T) + \sum_{t=0}^{T-1} f_t(\mathbf{x}_t, \mathbf{u}_t) \\ & \text{subject to} && \mathbf{x}_{t+1} = h_t(\mathbf{x}_t, \mathbf{u}_t) \quad (t = 0, 1, \dots, T-1) \end{aligned} \tag{9.1}$$

where the control is the sequence $(\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_{T-1})$ to be determined. This problem arises in many applications, in engineering, finance, economics etc. We now rewrite this problem. First, let $\mathbf{u} = (\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_T) \in \mathbb{R}^N$ where $N = Tn$. Since, as we noted, \mathbf{x}_t is uniquely determined by \mathbf{u} , there is a function \mathbf{v}_t such that $\mathbf{x}_t = \mathbf{v}_t(\mathbf{u})$ ($t = 1, 2, \dots, T$); \mathbf{x}_0 is given. Therefore the total cost may be written

$$f_T(\mathbf{x}_T) + \sum_{t=0}^{T-1} f_t(\mathbf{x}_t, \mathbf{u}_t) = f_T(\mathbf{v}_T(\mathbf{u})) + \sum_{t=0}^{T-1} f_t(\mathbf{v}_t(\mathbf{u}), \mathbf{u}_t) := f(\mathbf{u})$$

which is a function of \mathbf{u} . Thus, we see that the optimal control problem may be transformed to the unconstrained optimization problem

$$\min_{\mathbf{u} \in \mathbb{R}^N} f(\mathbf{u})$$

Sometimes there may be constraints on the control variables, for instance that they each lie in some interval, and then the transformation above results in a constrained optimization problem.

9.2.5 Linear optimization

This is not an application, but rather a special case of the general nonlinear optimization problem where all functions are linear. A *linear optimization* problem, also called *linear programming*, has the form

$$\begin{array}{ll} \text{minimize} & \mathbf{c}^T \mathbf{x} \\ \text{subject to} & \\ & A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}. \end{array} \quad (9.2)$$

Here A is an $m \times n$ matrix, $\mathbf{b} \in \mathbb{R}^m$ and $\mathbf{x} \geq \mathbf{0}$ means that $x_i \geq 0$ for each $i \leq n$. So in linear optimization one minimizes (or maximizes) a linear function subject to linear equations and nonnegativity on the variables. Actually, one can show any problem with constraints that are linear equations and/or linear inequalities may be transformed into the form above. Such problems have a wide range of application in science, engineering, economics, business etc. Applications include portfolio optimization and many planning problems for e.g. production, transportation etc. Some of these problems are of a combinatorial nature, but linear optimization is a main tool here as well.

We shall not treat linear optimization in detail here since this is the topic of a separate course, INF-MAT3370 Linear optimization. In that course one presents some powerful methods for such problems, the simplex algorithm and interior point methods. In addition one considers applications in network flow models and game theory.

9.3 Multivariate calculus and linear algebra

We first recall some useful facts from linear algebra.

The *spectral theorem* says that if A is a real symmetric matrix, then there is an orthogonal matrix V (i.e., its columns are orthonormal) and a diagonal matrix D such that

$$A = VDV^T.$$

The diagonal of D contains the eigenvalues of A , and A has an orthonormal set of eigenvectors (the columns of V).

A real symmetric matrix is *positive semidefinite*² if $\mathbf{x}^T A \mathbf{x} \geq 0$ for all $\mathbf{x} \in \mathbb{R}^n$. The following statements are equivalent

- (i) A is positive semidefinite,
- (ii) all eigenvalues of A are nonnegative,
- (iii) $A = W^T W$ for some matrix W .

Similarly, a real symmetric matrix is *positive definite* if $\mathbf{x}^T A \mathbf{x} > 0$ for all nonzero $\mathbf{x} \in \mathbb{R}^n$. The following statements are equivalent

- (i) A is positive definite,
- (ii) all eigenvalues of A are positive,
- (iii) $A = W^T W$ for some invertible matrix W .

Every positive definite matrix is therefore invertible.

We also recall some central facts from multivariate calculus. They will be used repeatedly in these notes. Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be a real-valued function defined on \mathbb{R}^n . The *gradient* of f at \mathbf{x} is the n -tuple

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, \dots, \frac{\partial f(\mathbf{x})}{\partial x_n} \right).$$

We will always identify an n -tuple with the corresponding column vector³. Of course, the gradient only exists if all the partial derivatives exist. Second order information is contained in a matrix: assuming f has second order partial derivatives we define the *Hessian matrix*⁴ $\nabla^2 f(\mathbf{x})$ as the $n \times n$ matrix whose (i, j) 'th entry is

$$\frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j}.$$

If these second order partial derivatives are continuous, then we may switch the order in the derivations, and $\nabla^2 f(\mathbf{x})$ is a symmetric matrix.

²See Section 7.2 in [7]

³This is somewhat different from [8], since the gradient there is always considered as a row vector

⁴See Section 5.9 in [8]

For vector-valued functions we also need the derivative. Consider the vector-valued function F given by

$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} F_1(\mathbf{x}) \\ F_2(\mathbf{x}) \\ \vdots \\ F_n(\mathbf{x}) \end{bmatrix}$$

so $F_i : \mathbb{R}^n \rightarrow \mathbb{R}$ is the i th component function of \mathbf{F} . \mathbf{F}' denotes the *Jacobi matrix*⁵, or simply the *derivative*, of \mathbf{F}

$$\mathbf{F}'(x) = \begin{bmatrix} \frac{\partial F_1(\mathbf{x})}{\partial x_1} & \frac{\partial F_1(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial F_1(\mathbf{x})}{\partial x_n} \\ \frac{\partial F_2(\mathbf{x})}{\partial x_1} & \frac{\partial F_2(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial F_2(\mathbf{x})}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_n(\mathbf{x})}{\partial x_1} & \frac{\partial F_n(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial F_n(\mathbf{x})}{\partial x_n} \end{bmatrix}$$

The i th row of this matrix is therefore the gradient of F_i , now viewed as a row vector.

Next we recall Taylor's theorems from multivariate calculus⁶:

Theorem 9.3 (First order Taylor theorem). Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be a function having continuous partial derivatives in some ball $B(\mathbf{x}; r)$. Then, for each $\mathbf{h} \in \mathbb{R}^n$ with $\|\mathbf{h}\| < r$ there is some $t \in (0, 1)$ such that

$$f(\mathbf{x} + \mathbf{h}) = f(\mathbf{x}) + \nabla f(\mathbf{x} + t\mathbf{h})^T \mathbf{h}.$$

The next one is known as Taylor's formula, or the second order Taylor's theorem⁷:

Theorem 9.4 (Second order Taylor theorem). Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be a function having second order partial derivatives that are continuous in some ball $B(\mathbf{x}; r)$. Then, for each $\mathbf{h} \in \mathbb{R}^n$ with $\|\mathbf{h}\| < r$ there is some $t \in (0, 1)$ such that

$$f(\mathbf{x} + \mathbf{h}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{h} + \frac{1}{2} \mathbf{h}^T \nabla^2 f(\mathbf{x} + t\mathbf{h}) \mathbf{h}.$$

This may be shown by considering the one-variable function $g(t) = f(\mathbf{x} + t\mathbf{h})$ and applying the chain rule and Taylor's formula in one variable.

⁵See Section 2.6 in [8]

⁶This theorem is also the mean value theorem of functions in several variables, see Section 5.5 in [8]

⁷See Section 5.9 in [8]

There is another version of the second order Taylor theorem in which the Hessian is evaluated in \mathbf{x} and, as a result, we get an error term. This theorem shows how f may be approximated by a quadratic polynomial in n variables⁸:

Theorem 9.5 (Second order Taylor theorem, version 2). Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be a function having second order partial derivatives that are continuous in some ball $B(\mathbf{x}; r)$. Then there is a function $\epsilon : \mathbb{R}^n \rightarrow \mathbb{R}$ such that, for each $\mathbf{h} \in \mathbb{R}^n$ with $\|\mathbf{h}\| < r$,

$$f(\mathbf{x} + \mathbf{h}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{h} + \frac{1}{2} \mathbf{h}^T \nabla^2 f(\mathbf{x}) \mathbf{h} + \epsilon(\mathbf{h}) \|\mathbf{h}\|^2.$$

Here $\epsilon(\mathbf{y}) \rightarrow 0$ when $\mathbf{y} \rightarrow 0$.

Using the O -notation from Definition 4.6, the very useful approximations we get from Taylor' theorems can thus be summarized as follows:

Taylor approximations:

$$\begin{aligned} \text{First order:} \quad f(\mathbf{x} + \mathbf{h}) &= f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{h} + O(\|\mathbf{h}\|) \\ &\approx f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{h}. \\ \text{Second order:} \quad f(\mathbf{x} + \mathbf{h}) &= f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{h} + \frac{1}{2} \mathbf{h}^T \nabla^2 f(\mathbf{x}) \mathbf{h} + O(\|\mathbf{h}\|^2) \\ &\approx f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{h} + \frac{1}{2} \mathbf{h}^T \nabla^2 f(\mathbf{x}) \mathbf{h}. \end{aligned}$$

We introduce notation for these approximations

$$\begin{aligned} T_f^1(\mathbf{x}; \mathbf{x} + \mathbf{h}) &= f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{h} \\ T_f^2(\mathbf{x}; \mathbf{x} + \mathbf{h}) &= f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{h} + \frac{1}{2} \mathbf{h}^T \nabla^2 f(\mathbf{x}) \mathbf{h} \end{aligned}$$

As we shall see, one can get a lot of optimization out of these approximations!

We also need a Taylor theorem for vector-valued functions, which follows by applying Taylor' theorem above to each component function:

Theorem 9.6 (First order Taylor theorem for vector-valued functions). Let $\mathbf{F} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ be a vector-valued function which is continuously differentiable in a neighborhood N of \mathbf{x} . Then

$$\mathbf{F}(\mathbf{x} + \mathbf{h}) = \mathbf{F}(\mathbf{x}) + \mathbf{F}'(\mathbf{x}) \mathbf{h} + O(\|\mathbf{h}\|)$$

when $\mathbf{x} + \mathbf{h} \in N$.

⁸See Section 5.9 in [8]

Finally, if $\mathbf{F} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ and $\mathbf{G} : \mathbb{R}^k \rightarrow \mathbb{R}^n$ we define the *composition* $\mathbf{H} = \mathbf{F} \circ \mathbf{G}$ as the function $\mathbf{H} : \mathbb{R}^k \rightarrow \mathbb{R}^m$ by $\mathbf{H}(\mathbf{x}) = \mathbf{F}(\mathbf{G}(\mathbf{x}))$. Then, under natural differentiability assumption the following *chain rule*⁹ holds:

$$\mathbf{H}'(\mathbf{x}) = \mathbf{F}'(\mathbf{G}(\mathbf{x}))\mathbf{G}'(\mathbf{x}).$$

Here the right-hand side is a product of two matrices, the respective Jacobi matrices evaluated in the right points.

Finally, we discuss some notions concerning the convergence of sequences.

Definition 9.7 (Linear convergence). We say that a sequence $\{\mathbf{x}_k\}_{k=1}^\infty$ converges to \mathbf{x}^* *linearly* (or that the convergence speed is linear) if there is a $\gamma < 1$ such that

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \leq \gamma \|\mathbf{x}_k - \mathbf{x}^*\| \quad (k = 0, 1, \dots).$$

A faster convergence rate is *superlinear convergence* which means that

$$\lim_{k \rightarrow \infty} \|\mathbf{x}_{k+1} - \mathbf{x}^*\| / \|\mathbf{x}_k - \mathbf{x}^*\| = 0$$

A special type of superlinear convergence is *quadratic convergence* where

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \leq \gamma \|\mathbf{x}_k - \mathbf{x}^*\|^2 \quad (k = 0, 1, \dots)$$

for some $\gamma < 1$.

Exercises for Section 9.3

Ex. 1 — Give an example of a function $f : \mathbb{R} \rightarrow \mathbb{R}$ with 10 global minima.

Ex. 2 — Consider the function $f(x) = x \sin(1/x)$ defined for $x > 0$. Find its local minima. What about global minimum?

Ex. 3 — Let $f : X \rightarrow \mathbb{R}_+$ be a function (with nonnegative function values). Explain why it is equivalent to minimize f over $x \in X$ or minimize $f^2(x)$ over X .

Ex. 4 — Consider $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ given by $f(\mathbf{x}) = (x_1 - 3)^2 + (x_2 - 2)^2$. How would you explain to *anyone* that $\mathbf{x}^* = (3, 2)$ is a minimum point?

⁹See Section 2.7 in [8]

Ex. 5 — The *level sets* of a function $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ are sets of the form $L_\alpha = \{\mathbf{x} \in \mathbb{R}^2 : f(\mathbf{x}) = \alpha\}$. Let $f(\mathbf{x}) = (1/4)(x-1)^2 + (x-3)^2$. Draw the level sets in the plane for $\alpha = 10, 5, 1, 0.1$.

Ex. 6 — The *sublevel set* of a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is the set $S_\alpha(f) = \{\mathbf{x} \in \mathbb{R}^n : f(\mathbf{x}) \leq \alpha\}$, where $\alpha \in \mathbb{R}$. Assume that $\inf\{f(\mathbf{x}) : \mathbf{x} \in \mathbb{R}^n\} = \eta$ exists.

- What happens to the sublevel sets S_α as α decreases? Give an example.
- Show that if f is continuous and there is an \mathbf{x}' such that with $\alpha = f(\mathbf{x}')$ the sublevel set $S_\alpha(f)$ is bounded, then f attains its minimum.

Ex. 7 — Consider the portfolio optimization problem in Subsection 9.2.1.

- Assume that $c_{ij} = 0$ for each $i \neq j$. Find, analytically, an optimal solution. Describe the set of all optimal solutions.
- Consider the special case where $n = 2$. Solve the problem (hint: eliminate one variable) and discuss how minimum point depends on α .

Ex. 8 — Later in these notes we will need the expression for the gradient of functions which are expressed in terms of matrices.

- Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be defined by $f(\mathbf{x}) = \mathbf{q}^T \mathbf{x} = \mathbf{x}^T \mathbf{q}$, where \mathbf{q} is a vector. Show that $\nabla f(\mathbf{x}) = \mathbf{q}$, and that $\nabla^2 f(\mathbf{x}) = \mathbf{0}$.
- Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be the quadratic function $f(\mathbf{x}) = (1/2)\mathbf{x}^T A \mathbf{x}$. Show that $\nabla f(\mathbf{x}) = A\mathbf{x}$, and that $\nabla^2 f(\mathbf{x}) = A$.

Ex. 9 — Consider $f(\mathbf{x}) = f(x_1, x_2) = x_1^2 + 3x_1x_2 - 5x_2^2 + 3$. Determine the first order Taylor approximation to f at each of the points $(0, 0)$ and $(2, 1)$.

Ex. 10 — Let $A = \begin{bmatrix} 1 & 2 \\ 2 & 8 \end{bmatrix}$. Show that A is positive definite. (Try to give two different proofs.)

Ex. 11 — Show that if A is positive definite, then its inverse is also positive definite.

Chapter 10

A crash course in convexity

Convexity is a branch of mathematical analysis dealing with convex sets and convex functions. It also represents a foundation for optimization.

We just summarize concepts and some results. For proofs one may consult [4] or [14], see also [1].

10.1 Convex sets

A set $C \subseteq \mathbb{R}^n$ is called *convex* if $(1 - \lambda)\mathbf{x} + \lambda\mathbf{y} \in C$ whenever $\mathbf{x}, \mathbf{y} \in C$ and $0 \leq \lambda \leq 1$. Geometrically, this means that C contains the line segment between each pair of points in C , so, loosely speaking, a convex set contains no “holes”.

For instance, the ball $B(\mathbf{a}; \delta) = \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x} - \mathbf{a}\| \leq \delta\}$ is a convex set. Let us show this. Recall the triangle inequality which says that $\|\mathbf{u} + \mathbf{v}\| \leq \|\mathbf{u}\| + \|\mathbf{v}\|$ whenever $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$. Let $\mathbf{x}, \mathbf{y} \in B(\mathbf{a}; \delta)$ and $\lambda \in [0, 1]$. Then

$$\begin{aligned} \|((1 - \lambda)\mathbf{x} + \lambda\mathbf{y}) - \mathbf{a}\| &= \|(1 - \lambda)(\mathbf{x} - \mathbf{a}) + \lambda(\mathbf{y} - \mathbf{a})\| \\ &\leq \|(1 - \lambda)(\mathbf{x} - \mathbf{a})\| + \|\lambda(\mathbf{y} - \mathbf{a})\| \\ &= (1 - \lambda)\|\mathbf{x} - \mathbf{a}\| + \lambda\|\mathbf{y} - \mathbf{a}\| \\ &\leq (1 - \lambda)\delta + \lambda\delta = \delta. \end{aligned}$$

Therefore $B(\mathbf{a}; \delta)$ is convex.

Every linear subspace is also a convex set, as well as the translate of every subspace (which is called an affine set). Some other examples of convex sets in \mathbb{R}^2 are shown in Figure 10.1. We will come back to why each of these sets are convex later. Another important property is that the intersection of a family of convex sets is a convex set.

By a *linear system* we mean a finite system of linear equations and/or linear inequalities involving n variables. For example

$$x_1 + x_2 = 3, x_1 \geq 0, x_2 \geq 0$$

is a linear system in the variables x_1, x_2 . The solution set is the set of points $(x_1, 3 - x_1)$ where $0 \leq x_1 \leq 3$. The set of solutions of a linear system is called

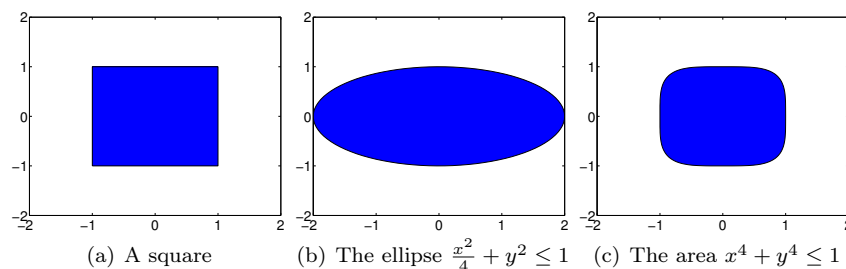


Figure 10.1: Examples of some convex sets.

a *polyhedron*. These sets often occur in optimization. Thus, a polyhedron has the form

$$P = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \leq \mathbf{b}\}$$

where $A \in \mathbb{R}^{m,n}$ and $\mathbf{b} \in \mathbb{R}^m$ (m is arbitrary, but finite) and \leq means componentwise inequality. There are simple techniques for rewriting any linear system in the form $A\mathbf{x} \leq \mathbf{b}$.

Proposition 10.1. Every polyhedron is a convex set.

The square from Figure 10.1(a) is defined by the inequalities $-1 \leq x, y \leq 1$. It is therefore a polyhedron, and therefore convex. The next result shows that convex sets are preserved under linear maps.

Proposition 10.2. If $T : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is a linear transformation, and $C \subseteq \mathbb{R}^n$ is a convex set, then the image $T(C)$ of this set is also convex.

10.2 Convex functions

The notion of a convex function also makes sense for real-valued functions of several variables. Consider a real-valued function $f : C \rightarrow \mathbb{R}$ where $C \subseteq \mathbb{R}^n$ is a convex set. We say that f is *convex* provided that

$$f((1 - \lambda)\mathbf{x} + \lambda\mathbf{y}) \leq (1 - \lambda)f(\mathbf{x}) + \lambda f(\mathbf{y}) \quad (\mathbf{x}, \mathbf{y} \in \mathbb{R}^n, 0 \leq \lambda \leq 1) \quad (10.1)$$

(This inequality holds for all \mathbf{x}, \mathbf{y} and λ as specified). Due to the convexity of C , the point $(1 - \lambda)\mathbf{x} + \lambda\mathbf{y}$ lies in C , so the inequality is well-defined. The geometrical interpretation in one dimension is that whenever you take two points on the graph of f , say $(x, f(x))$ and $(y, f(y))$, the graph of f restricted to the line segment $[x, y]$ lies below the line segment in \mathbb{R}^{n+1} between the two chosen points. A function g is called *concave* if $-g$ is convex.

Every linear function is convex. Some other examples of convex functions in n variables are

- $f(\mathbf{x}) = L(\mathbf{x}) + \alpha$ where L is a linear function from \mathbb{R}^n into \mathbb{R} (a linear functional) and α is a real number. Such a function is called an *affine function* and it may be written $f(\mathbf{x}) = \mathbf{c}^T \mathbf{x} + \alpha$ for a suitable vector \mathbf{c} .
- $f(\mathbf{x}) = \|\mathbf{x}\|$ (Euclidean norm). That this is convex can be proved by writing $\|(1 - \lambda)\mathbf{x} + \lambda\mathbf{y}\| \leq \|(1 - \lambda)\mathbf{x}\| + \|\lambda\mathbf{y}\| = (1 - \lambda)\|\mathbf{x}\| + \lambda\|\mathbf{y}\|$. In fact, the same argument can be used to show that *every* norm defines a convex function. Such an example is the l_1 -norm, also called the *sum norm*, defined by $\|\mathbf{x}\|_1 = \sum_{j=1}^n |x_j|$.
- $f(\mathbf{x}) = e^{\sum_{j=1}^n x_j}$ (see Exercise 7).
- $f(\mathbf{x}) = e^{h(\mathbf{x})}$ where $h : \mathbb{R}^n \rightarrow \mathbb{R}$ is a convex function.
- $f(\mathbf{x}) = \max_i g_i(\mathbf{x})$ where $g_i : \mathbb{R}^n \rightarrow \mathbb{R}$ is an affine function ($i \leq m$). This means that the pointwise maximum of affine functions is a convex function. Note that such convex functions are typically not differentiable everywhere. A more general result is that the pointwise supremum of an arbitrary family of affine functions (or even convex functions) is convex. This is a very useful fact in convexity and its applications.

The following result is an exercise to prove, and it gives a method for proving convexity of a function.

Proposition 10.3. Assume that $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is convex and $\mathbf{H} : \mathbb{R}^m \rightarrow \mathbb{R}^n$ is affine. Then the composition $f \circ \mathbf{H}$ is convex, where $(f \circ \mathbf{H})(\mathbf{x}) := f(\mathbf{H}(\mathbf{x}))$.

The next result is often used, and is called *Jensen's inequality*. It can be shown using induction.

Theorem 10.4 (Jensen's inequality). Let $f : C \rightarrow \mathbb{R}$ be a convex function defined on a convex set $C \subseteq \mathbb{R}^n$. If $\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^r \in C$ and $\lambda_1, \dots, \lambda_r \geq 0$ satisfy $\sum_{j=1}^r \lambda_j = 1$, then

$$f\left(\sum_{j=1}^r \lambda_j \mathbf{x}^j\right) \leq \sum_{j=1}^r \lambda_j f(\mathbf{x}^j). \quad (10.2)$$

A point of the form $\sum_{j=1}^r \lambda_j \mathbf{x}^j$, where the λ_j 's are nonnegative and sum to 1, is called a *convex combination* of the points $\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^r$. One can show that a set is convex if and only if it contains all convex combinations of its points.

Finally, one connection between convex sets and convex functions is the following fact whose proof is an exercise.

Proposition 10.5. Let $C \subseteq \mathbb{R}^n$ be a convex set and consider a convex function $f : C \rightarrow \mathbb{R}$. Let $\alpha \in \mathbb{R}$. Then the “sublevel” set

$$\{\mathbf{x} \in C : f(\mathbf{x}) \leq \alpha\}$$

is a convex set.

10.3 Properties of convex functions

A convex function may not be differentiable in every point. However, one can show that a convex function always has one-sided directional derivatives at any point. But what about continuity?

Theorem 10.6. Let $f : C \rightarrow \mathbb{R}$ be a convex function defined on an open convex set $C \subseteq \mathbb{R}^n$. Then f is continuous on C .

However, a convex function may be discontinuous in points on the *boundary* of its domain. For instance, the function $f : [0, 1] \rightarrow \mathbb{R}$ given by $f(0) = 1$ and $f(x) = 0$ for $x \in (0, 1]$ is convex, but discontinuous at $x = 0$. Next we give a useful technique for checking that a function is convex.

Theorem 10.7. Let f be a real-valued function defined on an open convex set $C \subseteq \mathbb{R}^n$ and assume that f has continuous second-order partial derivatives on C .

Then f is convex if and only if the Hessian matrix $\nabla^2 f(\mathbf{x})$ is positive semidefinite for each $\mathbf{x} \in C$.

With this result it is straightforward to prove that the remaining sets from Figure 10.1 are convex. They can be written as sublevel sets of the functions $f(x, y) = \frac{x^2}{4} + y^2$, and $f(x, y) = x^4 + y^4$. For the first of these the level sets are ellipses, and are shown in Figure 10.2, together with f itself. One can quickly verify that the Hessian matrices of these functions are positive semidefinite. It follows from Proposition 10.5 that the corresponding sets are convex.

An important class of convex functions consists of (certain) quadratic functions. Let $A \in \mathbb{R}^{n \times n}$ be a symmetric matrix which is positive semidefinite and consider the quadratic function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ given by

$$f(\mathbf{x}) = (1/2) \mathbf{x}^T A \mathbf{x} - \mathbf{b}^T \mathbf{x} = (1/2) \sum_{i,j} a_{ij} x_i x_j - \sum_{j=1}^n b_j x_j.$$

(If $A = \mathbf{0}$, then the function is linear, and it may be strange to call it quadratic. But we still do this, for simplicity.) Then (Exercise 9.3.8) the Hessian matrix of f is A , i.e., $\nabla^2 f(\mathbf{x}) = A$ for each $\mathbf{x} \in \mathbb{R}^n$. Therefore, by Theorem 10.7 is a convex function.

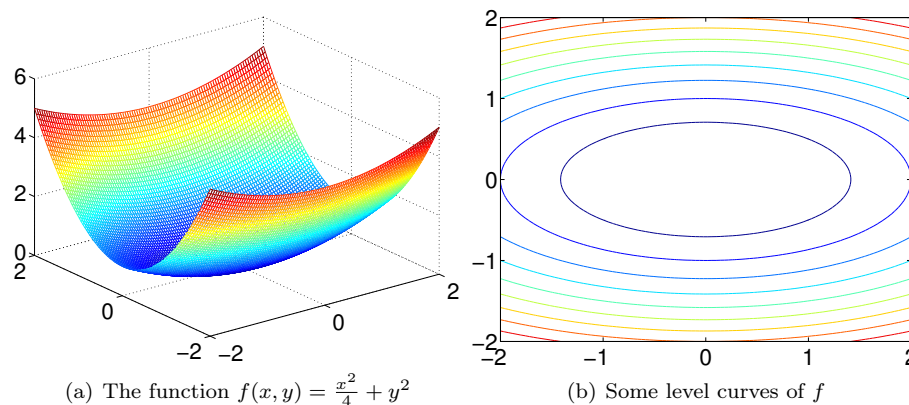


Figure 10.2: A function and its level curves.

We remark that sometimes it may be easy to check that a symmetric matrix A is positive semidefinite. A (real) symmetric $n \times n$ matrix A is called *diagonally dominant* if $|a_{ii}| \geq \sum_{j \neq i} |a_{ij}|$ for $i = 1, \dots, n$. These matrices arise in many applications, e.g. splines and differential equations. It can be shown that every symmetric diagonally dominant matrix is positive semidefinite. For a simple proof of this fact using convexity, see [3]. Thus, we get a simple criterion for convexity of a function: check if the Hessian matrix $\nabla^2 f(\mathbf{x})$ is diagonally dominant for each \mathbf{x} . Be careful here: this matrix may be positive semidefinite without being diagonally dominant!

We now look at differentiability properties of convex functions.

Theorem 10.8. Let f be a real-valued convex function defined on an open convex set $C \subseteq \mathbb{R}^n$. Assume that all the partial derivatives $\partial f(\mathbf{x})/\partial x_1, \dots, \partial f(\mathbf{x})/\partial x_n$ exist at a point $\mathbf{x} \in C$. Then f is differentiable at \mathbf{x} .

A convex function may not be differentiable everywhere, but it is differentiable “almost everywhere”. More precisely, for a convex function defined on an open convex set in \mathbb{R}^n , the set of points for which f is not differentiable has Lebesgue measure zero. We do not go into further details on this here, but refer to e.g. [5] for a proof and a discussion.

Another characterization of convex functions that involves the gradient may now be presented.

Theorem 10.9. Let $f : C \rightarrow \mathbb{R}$ be a differentiable function defined on an open convex set $C \subseteq \mathbb{R}^n$. Then the following conditions are equivalent:

- | | | |
|-------|--|--|
| (i) | f is convex. | |
| (ii) | $f(\mathbf{x}) \geq f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0)$ | for all $\mathbf{x}, \mathbf{x}_0 \in C$. |
| (iii) | $(\nabla f(\mathbf{x}) - \nabla f(\mathbf{x}_0))^T(\mathbf{x} - \mathbf{x}_0) \geq 0$ | for all $\mathbf{x}, \mathbf{x}_0 \in C$. |

This theorem is important. Property (ii) says that the first-order Taylor approximation of f at \mathbf{x}_0 (which is the right-hand side of the inequality) always underestimates f . This result has interesting consequences for optimization as we shall see later.

Exercises for section 10.3

Ex. 1 — Let $S = \{(x, y, z) : z \geq x^2 + y^2\} \subset \mathbb{R}^3$. Sketch the set and verify that it is a convex set.

Ex. 2 — Let $f : S \rightarrow \mathbb{R}$ be a differentiable function, where S is an open set in \mathbb{R} . Check that f is convex if and only if $f''(x) \geq 0$ for all $x \in S$.

Ex. 3 — Prove Proposition 10.3.

Ex. 4 — Prove Proposition 10.5.

Ex. 5 — Explain how you can write the LP problem $\max \{\mathbf{c}^T \mathbf{x} : A\mathbf{x} \geq \mathbf{b}, B\mathbf{x} = \mathbf{d}, \mathbf{x} \geq \mathbf{0}\}$ as an LP problem of the form

$$\max\{\mathbf{c}^T \mathbf{x} : H\mathbf{x} \leq \mathbf{h}, \mathbf{x} \geq \mathbf{0}\}$$

for suitable matrix H and vector \mathbf{h} .

Ex. 6 — Let $\mathbf{x}_1, \dots, \mathbf{x}_t \in \mathbb{R}^n$ and let C be the set of vectors of the form

$$\sum_{j=1}^t \lambda_j \mathbf{x}_j$$

where $\lambda_j \geq 0$ for each $j = 1, \dots, t$. Show that C is convex. Make a sketch of such a set in \mathbb{R}^3 .

Ex. 7 — Show that $f(\mathbf{x}) = e^{\sum_{j=1}^n x_j}$ is a convex function.

Ex. 8 — Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be a convex function and let $\alpha \in \mathbb{R}$. Show that the sublevel set $S_\alpha(f) = \{\mathbf{x} \in \mathbb{R}^n : f(\mathbf{x}) \leq \alpha\}$ is a convex set.

Ex. 9 — Assume that f and g are convex functions defined on an interval I . Determine which of the functions following functions that are convex or concave:

- a. λf where $\lambda \in \mathbb{R}$,
- b. $\min\{f, g\}$,
- c. $|f|$.

Ex. 10 — Let $f : [a, b] \rightarrow \mathbb{R}$ be a convex function. Show that

$$\max\{f(x) : x \in [a, b]\} = \max\{f(a), f(b)\}$$

i.e., a convex function defined on closed real interval attains its maximum in one of the endpoints.

Ex. 11 — Let $f : \langle 0, \infty \rangle \rightarrow \mathbb{R}$ and define the function $g : \langle 0, \infty \rangle \rightarrow \mathbb{R}$ by $g(x) = xf(1/x)$. Why is the function $x \rightarrow xe^{1/x}$ convex?

Ex. 12 — Let $C \subseteq \mathbb{R}^n$ be a convex set and consider the distance function d_C defined by $d_C(x) = \inf\{\|x - y\| : y \in C\}$. Show that d_C is a convex function.

Chapter 11

Nonlinear equations

A basic mathematical problem is to solve a system of equations in several unknowns (variables). There are numerical methods that can solve such equations, at least within a small error tolerance. We shall briefly discuss such methods here; for further details, see [6, 11].

11.1 Equations and fixed points

In linear algebra one works a lot with linear equations in several variables, and Gaussian elimination is a central method for solving such equations. There are also other faster methods, so-called iterative methods, for linear equations. But what about *nonlinear equations*? For instance, consider the system in two variables x_1 and x_2 :

$$\begin{aligned}x_1^2 - x_1x_2^{-3} + \cos x_1 &= 1 \\5x_1^4 + 2x_1^3 - \tan(x_1x_2^8) &= 3\end{aligned}$$

Clearly, such equations can be very hard to solve. The general problem is to solve the equation

$$\mathbf{F}(\mathbf{x}) = \mathbf{0} \tag{11.1}$$

for a given function $\mathbf{F} : \mathbb{R}^n \rightarrow \mathbb{R}^n$. If $\mathbf{F}(\mathbf{x}) = \mathbf{0}$ we call \mathbf{x} a *root* of \mathbf{F} (or of the equation). The example above is equivalent to finding roots in $\mathbf{F}(\mathbf{x}) = (F_1(\mathbf{x}), F_2(\mathbf{x}))$ where

$$\begin{aligned}F_1(\mathbf{x}) &= x_1^2 - x_1x_2^{-3} + \cos x_1 - 1 \\F_2(\mathbf{x}) &= 5x_1^4 + 2x_1^3 - \tan(x_1x_2^8) - 3\end{aligned}$$

In particular, if $\mathbf{F}(\mathbf{x}) = A\mathbf{x} - \mathbf{b}$ where A is an $n \times n$ matrix and $\mathbf{b} \in \mathbb{R}^n$, then we are back to linear equations (a square system). More generally one may consider equations $\mathbf{G}(\mathbf{x}) = \mathbf{0}$ where $\mathbf{G} : \mathbb{R}^n \rightarrow \mathbb{R}^m$, but we here only discuss the case $m = n$.

Often the problem $\mathbf{F}(\mathbf{x}) = \mathbf{0}$ has the following form, or may be rewritten to it:

$$\mathbf{K}(\mathbf{x}) = \mathbf{x}. \quad (11.2)$$

for some function $\mathbf{K} : \mathbb{R}^n \rightarrow \mathbb{R}^n$. This corresponds to the special choice $\mathbf{F}(\mathbf{x}) = \mathbf{K}(\mathbf{x}) - \mathbf{x}$. A point $\mathbf{x} \in \mathbb{R}^n$ such that $\mathbf{x} = \mathbf{K}(\mathbf{x})$ is called a *fixed point* of the function \mathbf{K} . In finding such a fixed point it is tempting to use the following iterative method: choose a starting point \mathbf{x}_0 and repeat the following iteration

$$\mathbf{x}_{k+1} = \mathbf{K}(\mathbf{x}_k) \quad \text{for } k = 1, 2, \dots \quad (11.3)$$

This is called a *fixed-point iteration*. We note that if \mathbf{K} is continuous and this procedure converges to some point \mathbf{x}^* , then \mathbf{x}^* must be a fixed point. The fixed-point iteration is an extremely simple algorithm, and very easy to implement. Perhaps surprisingly, it also works very well for many such problems. Let $\epsilon > 0$ denote a small error tolerance used for stopping the process, e.g. 10^{-6} .

Fixed-point algorithm:

1. Choose an initial point \mathbf{x}_0 , let $\mathbf{x} = \mathbf{x}_0$ and err = 1.
2. while err > ϵ do
 - (i) Compute $\mathbf{x}_1 = \mathbf{K}(\mathbf{x})$
 - (ii) Compute err = $\|\mathbf{x}_1 - \mathbf{x}\|$
 - (iii) Update $\mathbf{x} := \mathbf{x}_1$

When does the fixed-point iteration work? Let $\|\cdot\|$ be a fixed norm, e.g. the Euclidean norm, on \mathbb{R}^n . We say that the function $\mathbf{K} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is a *contraction* if there is a constant $0 \leq c < 1$ such that

$$\|\mathbf{K}(\mathbf{x}) - \mathbf{K}(\mathbf{y})\| \leq c\|\mathbf{x} - \mathbf{y}\| \quad (\mathbf{x}, \mathbf{y} \in \mathbb{R}^n).$$

We also say that \mathbf{K} is *c-Lipschitz* in this case. The following theorem is called the *Banach contraction principle*. It also holds in Banach spaces, i.e., complete normed vector spaces (possibly infinite-dimensional).

Theorem 11.1. Assume that \mathbf{K} is *c-Lipschitz* with $0 < c < 1$. Then \mathbf{K} has a unique fixed point \mathbf{x}^* . For any starting point \mathbf{x}_0 the fixed-point iteration (11.3) generates a sequence $\{\mathbf{x}_k\}_{k=0}^{\infty}$ that converges to \mathbf{x}^* . Moreover

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \leq c\|\mathbf{x}_k - \mathbf{x}^*\| \quad \text{for } k = 0, 1, \dots \quad (11.4)$$

so that

$$\|\mathbf{x}_k - \mathbf{x}^*\| \leq c^k \|\mathbf{x}_0 - \mathbf{x}^*\|.$$

Proof. First, note that if both \mathbf{x} and \mathbf{y} are fixed points of \mathbf{K} , then

$$\|\mathbf{x} - \mathbf{y}\| = \|\mathbf{K}(\mathbf{x}) - \mathbf{K}(\mathbf{y})\| \leq c\|\mathbf{x} - \mathbf{y}\|$$

which means that $\mathbf{x} = \mathbf{y}$ (as $c < 1$); therefore \mathbf{K} has at most one fixed point. Next, we compute

$$\|\mathbf{x}_{k+1} - \mathbf{x}_k\| = \|\mathbf{K}(\mathbf{x}_k) - \mathbf{K}(\mathbf{x}_{k-1})\| \leq c\|\mathbf{x}_k - \mathbf{x}_{k-1}\| = \cdots \leq c^k\|\mathbf{x}_1 - \mathbf{x}_0\|$$

so

$$\begin{aligned} \|\mathbf{x}_m - \mathbf{x}_0\| &= \left\| \sum_{k=0}^{m-1} (\mathbf{x}_{k+1} - \mathbf{x}_k) \right\| \leq \sum_{k=0}^{m-1} \|\mathbf{x}_{k+1} - \mathbf{x}_k\| \\ &\leq \left(\sum_{k=0}^{m-1} c^k \right) \|\mathbf{x}_1 - \mathbf{x}_0\| \leq (1/(1-c))\|\mathbf{x}_1 - \mathbf{x}_0\| \end{aligned}$$

From this we derive that $\{\mathbf{x}_k\}$ is a Cauchy sequence; as we have

$$\begin{aligned} \|\mathbf{x}_{s+m} - \mathbf{x}_s\| &= \|\mathbf{K}(\mathbf{x}_{s+m-1}) - \mathbf{K}(\mathbf{x}_{s-1})\| \leq c\|\mathbf{x}_{s+m-1} - \mathbf{x}_{s-1}\| = \cdots \\ &\leq c^s\|\mathbf{x}_m - \mathbf{x}_0\| \leq (c^s/(1-c))\|\mathbf{x}_1 - \mathbf{x}_0\|. \end{aligned}$$

and $0 < c < 1$. Any Cauchy sequence in \mathbb{R}^n has a limit point, so $\mathbf{x}_m \rightarrow \mathbf{x}^*$ for some $\mathbf{x}^* \in \mathbb{R}^n$. We now prove that the limit point \mathbf{x}^* is a (actually, the) fixed point:

$$\begin{aligned} \|\mathbf{x}^* - \mathbf{K}(\mathbf{x}^*)\| &\leq \|\mathbf{x}^* - \mathbf{x}_m\| + \|\mathbf{x}_m - \mathbf{K}(\mathbf{x}^*)\| \\ &= \|\mathbf{x}^* - \mathbf{x}_m\| + \|\mathbf{K}(\mathbf{x}_{m-1}) - \mathbf{K}(\mathbf{x}^*)\| \\ &\leq \|\mathbf{x}^* - \mathbf{x}_m\| + c\|\mathbf{x}_{m-1} - \mathbf{x}^*\| \end{aligned}$$

and letting $m \rightarrow \infty$ here gives $\|\mathbf{x}^* - \mathbf{K}(\mathbf{x}^*)\| \leq 0$ so $\mathbf{x}^* = \mathbf{K}(\mathbf{x}^*)$ as desired.

Finally,

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| = \|\mathbf{K}(\mathbf{x}_k) - \mathbf{K}(\mathbf{x}^*)\| \leq c\|\mathbf{x}_k - \mathbf{x}^*\| \leq c^{k+1}\|\mathbf{x}_0 - \mathbf{x}^*\|$$

which completes the proof. \square

We see that $\mathbf{x}_k \rightarrow \mathbf{x}^*$ *linearly*, and that Equation (11.4) gives an estimate on the convergence speed.

11.2 Newton's method

We return to the main problem (11.1). Our goal is to present Newton's method, a highly efficient iterative method for solving this equation. The method constructs a sequence

$$\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \dots$$

in \mathbb{R}^n which, hopefully, converges to a root \mathbf{x}^* of \mathbf{F} , so $\mathbf{F}(\mathbf{x}^*) = \mathbf{0}$. The idea is to linearize \mathbf{F} at the current iterate \mathbf{x}_k and choose the next iterate \mathbf{x}_{k+1} as a zero of this linearized function. The first order Taylor approximation of \mathbf{F} at \mathbf{x}_k is

$$T_{\mathbf{F}}^1(\mathbf{x}_k; \mathbf{x}) = \mathbf{F}(\mathbf{x}_k) + \mathbf{F}'(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k).$$

We solve $T_F^1(\mathbf{x}_k; \mathbf{x}) = \mathbf{0}$ for \mathbf{x} and define the next iterate as $\mathbf{x}_{k+1} = \mathbf{x}$. This gives

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{F}'(\mathbf{x}_k)^{-1} \mathbf{F}(\mathbf{x}_k) \quad (11.5)$$

which leads to Newton's method. One here assumes that the derivative \mathbf{F}' is known analytically. Note that we do not (and hardly ever do!) compute the inverse of the matrix \mathbf{F}' .

Newton's method for nonlinear equations:

1. Choose an initial point \mathbf{x}_0 .
2. For $k = 0, 1, \dots$ do
 - (i) Find the direction \mathbf{p} by solving $\mathbf{F}'(\mathbf{x}_k)\mathbf{p} = -\mathbf{F}(\mathbf{x}_k)$
 - (ii) Update: $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}$

In the main step, which is to compute \mathbf{p} , one needs to solve an $n \times n$ linear system of equations where the coefficient matrix is the Jacobi matrix of \mathbf{F} , evaluated at \mathbf{x}_k . In MAT1110 [8] we implemented the following code for Newton's method for nonlinear equations:

```
function x=newtonmult(x0,F,J)
% Performs Newtons method in many variables
% x: column vector which contains the start point
% F: computes the values of F
% J: computes the Jacobi matrix
epsilon=0.0000001; N=30; n=0;
x=x0;
while norm(F(x)) > epsilon && n<=N
    x=x-J(x)\F(x);
    fval = F(x);
    fprintf('itnr=%2d x=[%13.10f,%13.10f] F(x)=[%13.10f,%13.10f]\n',...
        n,x(1),x(2),fval(1),fval(2))
    n = n + 1;
end
```

This code also terminates after a given number of iterations, and when a given accuracy is obtained. Note that this function should work for any function \mathbf{F} , since it is a parameter to the function.

The convergence of Newton's method may be analyzed using fixed point theory since one may view Newton's method as a fixed point iteration. Observe that the Newton iteration (11.5) may be written

$$\mathbf{x}_{k+1} = \mathbf{G}(\mathbf{x}_k)$$

where \mathbf{G} is the function

$$\mathbf{G}(\mathbf{x}) = \mathbf{x} - \mathbf{F}'(\mathbf{x})^{-1} \mathbf{F}(\mathbf{x})$$

From this it is possible to show that if the starting point is sufficiently close to the root, then Newton's method will converge to this root at a linear convergence rate. With more clever arguments one may show that the convergence rate of Newton's method is even faster: it has superlinear convergence. Actually, for many functions one even has quadratic convergence rate. The proof of the following convergence theorem relies purely on Taylor's theorem.

Theorem 11.2. Assume that Newton's method with initial point \mathbf{x}_0 produces a sequence $\{\mathbf{x}_k\}_{k=0}^\infty$ which converges to a solution \mathbf{x}^* of (11.1). Then the convergence rate is superlinear.

Proof. From Taylor's theorem for vector-valued functions, Theorem 9.6, in the point \mathbf{x}_k we have

$$\mathbf{0} = \mathbf{F}(\mathbf{x}^*) = \mathbf{F}(\mathbf{x}_k + (\mathbf{x}^* - \mathbf{x}_k)) = \mathbf{F}(\mathbf{x}_k) + \mathbf{F}'(\mathbf{x}_k)(\mathbf{x}^* - \mathbf{x}_k) + O(\|\mathbf{x}_k - \mathbf{x}^*\|)$$

Multiplying this equation by $\mathbf{F}'(\mathbf{x}_k)^{-1}$ (which is assumed to exist!) gives

$$\mathbf{x}_k - \mathbf{x}^* - \mathbf{F}'(\mathbf{x}_k)^{-1}\mathbf{F}(\mathbf{x}_k) = O(\|\mathbf{x}_k - \mathbf{x}^*\|)$$

Combining this with the Newton iteration $\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{F}'(\mathbf{x}_k)^{-1}\mathbf{F}(\mathbf{x}_k)$ we get

$$\mathbf{x}_{k+1} - \mathbf{x}^* = O(\|\mathbf{x}_k - \mathbf{x}^*\|).$$

So

$$\lim_{k \rightarrow \infty} \|\mathbf{x}_{k+1} - \mathbf{x}^*\| / \|\mathbf{x}_k - \mathbf{x}^*\| = 0$$

This shows the superlinear convergence. □

The previous result is interesting, but it does not say *how near* to the root the starting point need to be in order to get convergence. This is the next topic. Let $\mathbf{F} : U \rightarrow \mathbb{R}^n$ where U is an open, convex set in \mathbb{R}^n . Consider the conditions on the derivative \mathbf{F}'

$$\begin{aligned} (i) \quad & \|\mathbf{F}'(\mathbf{x}) - \mathbf{F}'(\mathbf{y})\| \leq M\|\mathbf{x} - \mathbf{y}\| \quad \text{for all } \mathbf{x}, \mathbf{y} \in U \\ (ii) \quad & \|\mathbf{F}'(\mathbf{x}_0)\| \leq K \quad \text{for some } \mathbf{x}_0 \in U \end{aligned} \tag{11.6}$$

where K and L are some constants. Here $\|\mathbf{F}'(\mathbf{x}_0)\|$ denotes the *operator norm* of the square matrix $\mathbf{F}'(\mathbf{x}_0)$ which is defined as

$$\|\mathbf{F}'(\mathbf{x}_0)\| = \sup_{\|\mathbf{x}\|=1} \|\mathbf{F}'(\mathbf{x}_0)\mathbf{x}\|$$

and it measures how much the operator $\mathbf{F}'(\mathbf{x}_0)$ may increase the size of vectors. The following convergence result for Newton's method is known as *Kantorovich' theorem*.

Theorem 11.3 (Kantorovich' theorem). Let $\mathbf{F} : U \rightarrow \mathbb{R}^n$ be a differentiable function satisfying (11.6). Assume that $\bar{B}(\mathbf{x}_0; 1/(KL)) \subseteq U$ and that

$$\|\mathbf{F}'(\mathbf{x}_0)^{-1}\mathbf{F}(\mathbf{x}_0)\| \leq 1/(2KL).$$

Then $\mathbf{F}'(\mathbf{x})$ is invertible for all $\mathbf{x} \in B(\mathbf{x}_0; 1/(KL))$ and Newton's method with initial point \mathbf{x}_0 will produce a sequence $\{\mathbf{x}_k\}_{k=0}^{\infty}$ contained in $B(\mathbf{x}_0; 1/(KL))$ and $\lim_{k \rightarrow \infty} \mathbf{x}_k = \mathbf{x}^*$ for some limit point $\mathbf{x}^* \in \bar{B}(\mathbf{x}_0; 1/(KL))$ with

$$\mathbf{F}(\mathbf{x}^*) = \mathbf{0}.$$

A proof of this theorem is quite long (but not very difficult to understand) [8].

One disadvantage with Newton's method is that one needs to know the Jacobi matrix \mathbf{F}' explicitly. For complicated functions, or functions being the output of a simulation, the derivative may be hard or impossible to find. The *quasi-Newton method*, also called the *secant-method*, is then a good alternative. The idea is to approximate $\mathbf{F}'(\mathbf{x}_k)$ by some matrix B_k and to compute the new search direction from

$$B_k \mathbf{p} = -\mathbf{F}(\mathbf{x}_k)$$

A practical method for finding these approximations B_1, B_2, \dots is *Broyden's method*. Provided that the previous iteration gave \mathbf{x}_k , with Broyden's method we compute \mathbf{x}_{k+1} by following the search direction, define $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ and $\mathbf{y}_k = \mathbf{F}(\mathbf{x}_{k+1}) - \mathbf{F}(\mathbf{x}_k)$, and compute B_{k+1} from B_k by the formula

$$B_{k+1} = B_k + (1/\mathbf{s}_k^T \mathbf{s}_k)(\mathbf{y}_k - B_k \mathbf{s}_k)\mathbf{s}_k^T. \quad (11.7)$$

It can be shown that B_k approximates the Jacobi matrix $\mathbf{F}'(\mathbf{x}_k)$ well in each iteration. Moreover, the update given in (11.7) can be done efficiently (it is a rank one update of B_k).

Algorithm: Broyden's method:

1. Choose an initial point \mathbf{x}_0 , and an initial B_0 .
2. For $k = 0, 1, \dots$ do
 - (i) Find direction \mathbf{p}_k by solving $B_k \mathbf{p} = -\mathbf{F}(\mathbf{x}_k)$
 - (ii) Use line search (see Section 12.2) along direction \mathbf{p}_k to find α_k
 - (iii) Update: $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k$

$$\mathbf{s}_k := \mathbf{x}_{k+1} - \mathbf{x}_k$$

$$\mathbf{y}_k := \mathbf{F}(\mathbf{x}_{k+1}) - \mathbf{F}(\mathbf{x}_k)$$
 compute B_{k+1} from (11.7).

Note that this algorithm also computes an α through what we call a *line search*, to attempt to find the optimal distance to follow the search direction.

We do not here specify how this line search can be performed. Also, we do not specify how the initial values can be chosen. For B_0 , any approximation of the Jacobian of \mathbf{F} at \mathbf{x}_0 can be used, using a numerical differentiation method of your own choosing. One can show that Broyden's method, under certain assumptions, also converges superlinearly, see [11].

Exercises for Section 11.2

Ex. 1 — Show that the problem of solving nonlinear equations (11.1) may be transformed into a nonlinear optimization problem. (Hint: Square each component function and sum these up!)

Ex. 2 — Let $T : \mathbb{R} \rightarrow \mathbb{R}$ be given by $T(x) = (3/2)(x - x^3)$. Draw the graph of this function, and determine its fixed points. Let x^* denote the largest fixed point. Find, using your graph, an interval I containing x^* such that the fixed point algorithm with an initial point in I will guaranteed converge towards x^* . Then try the fixed point algorithm with starting point $x_0 = \sqrt{5/3}$.

Ex. 3 — Let $\alpha \in \mathbb{R}_+$ be fixed, and consider $f(x) = x^2 - \alpha$. Then the zeros are $\pm\sqrt{\alpha}$. Write down the Newton's iteration for this problem. Let $\alpha = 2$ and compute the first three iterates in Newton's method when $x_0 = 1$.

Ex. 4 — For any vector norm $\|\cdot\|$ on \mathbb{R}^n , we can more generally define a corresponding *operator norm* for $n \times n$ matrices by

$$\|A\| = \sup_{\|\mathbf{x}\|=1} \|A\mathbf{x}\|.$$

- a. Explain why this supremum is attained.
- b. Consider the vector norm $\|\mathbf{x}\| = \|\mathbf{x}\|_1 = \sum_{j=1}^n |x_j|$ on \mathbb{R}^n . For $n = 2$, draw the sublevel set $\{\mathbf{x} \in \mathbb{R}^2 : \|\mathbf{x}\|_1 \leq 1\}$. Compute the corresponding operator norm $\|A\|$ where A is an $n \times n$ matrix.

Ex. 5 — Consider a linear map $T : \mathbb{R}^n \rightarrow \mathbb{R}^m$ given by $T(\mathbf{x}) = A\mathbf{x}$ where A is an $n \times n$ matrix. When is T a contraction, using the operator norm defined in the previous exercise?

Ex. 6 — Test the function `newtonmult` on the equations given initially in Section 11.1.

Ex. 7 — In this exercise we will implement Broyden's method with Matlab.

- a. Given a value \mathbf{x}_0 , implement a function which computes an estimate of $\mathbf{F}'(\mathbf{x}_0)$ by estimating the partial derivatives of \mathbf{F} , using a numerical differentiation method and step size of your own choosing.
- b. Implement a function

```
function x=broyden(x0,F)
```

which returns an estimate of a zero of \mathbf{F} using Broyden's method. Your method should set B_0 to be the matrix obtained from the function in a. Just indicate where line search along the search direction should be performed in your function, without implementing it. The function should work as `newtonmult` in that it terminates after a given number of iterations, or after precision of a given accuracy has been obtained.

Chapter 12

Unconstrained optimization

How can we know whether a given point \mathbf{x}^* is a minimum, local or global, of some given function $f : \mathbb{R}^n \rightarrow \mathbb{R}$? And how can we find such a point \mathbf{x}^* ?

These are, of course, some main questions in optimization. In order to give good answers to these questions we need *optimality conditions*. They provide tests for optimality, and serve as the basis for algorithms. We here focus on differentiable functions; the corresponding results for the nondifferentiable case are more difficult (but they exist, and are based on convexity, see [5, 13]).

For unconstrained problems it is not difficult to find powerful optimality conditions from Taylor's theorem for functions in several variables.

12.1 Optimality conditions

In order to establish optimality conditions in unconstrained optimization, Taylor's theorem is the starting point, see Section 9.3. We only consider minimization problems, as maximization problems are turned into minimization problems by multiplying the function f by -1 .

First we look at some necessary optimality conditions.

Theorem 12.1. Assume that $f : \mathbb{R}^n \rightarrow \mathbb{R}$ has continuous partial derivatives, and assume that \mathbf{x}^* is a local minimum of f . Then

$$\nabla f(\mathbf{x}^*) = \mathbf{0}. \quad (12.1)$$

If, moreover, f has continuous second order partial derivatives, then $\nabla^2 f(\mathbf{x}^*)$ is positive semidefinite.

Proof. Assume that \mathbf{x}^* is a local minimum of f and that $\nabla f(\mathbf{x}^*) \neq \mathbf{0}$. Let $\mathbf{h} = -\alpha \nabla f(\mathbf{x}^*)$ where $\alpha > 0$. Then $\nabla f(\mathbf{x}^*)^T \mathbf{h} = -\alpha \|\nabla f(\mathbf{x}^*)\|^2 < 0$ and by continuity of the partial derivatives of f , $\nabla f(\mathbf{x})^T \mathbf{h} < 0$ for all \mathbf{x} in some

neighborhood of \mathbf{x}^* . From Theorem 9.3 (first order Taylor) we obtain

$$f(\mathbf{x}^* + \mathbf{h}) - f(\mathbf{x}^*) = \nabla f(\mathbf{x}^* + t\mathbf{h})^T \mathbf{h} \quad (12.2)$$

for some $t \in (0, 1)$ (depending on α). By choosing α small enough, the right-hand side of (12.2) is negative (as just said), and so $f(\mathbf{x}^* + \mathbf{h}) < f(\mathbf{x}^*)$, contradicting that \mathbf{x}^* is a local minimum. This proves that $\nabla f(\mathbf{x}^*) = \mathbf{0}$.

To prove the second statement, we get from Theorem 9.4 (second order Taylor)

$$\begin{aligned} f(\mathbf{x}^* + \mathbf{h}) &= f(\mathbf{x}^*) + \nabla f(\mathbf{x}^*)^T \mathbf{h} + \frac{1}{2} \mathbf{h}^T \nabla^2 f(\mathbf{x}^* + t\mathbf{h}) \mathbf{h} \\ &= f(\mathbf{x}^*) + \frac{1}{2} \mathbf{h}^T \nabla^2 f(\mathbf{x}^*) \mathbf{h} \end{aligned} \quad (12.3)$$

If $\nabla^2 f(\mathbf{x}^*)$ is not positive semidefinite, there is an \mathbf{h} such that $\mathbf{h}^T \nabla^2 f(\mathbf{x}^*) \mathbf{h} < 0$ and, by continuity of the second order partial derivatives, $\mathbf{h}^T \nabla^2 f(\mathbf{x}) \mathbf{h} < 0$ for all \mathbf{x} in some neighborhood of \mathbf{x}^* . But then (12.3) gives $f(\mathbf{x}^* + \mathbf{h}) - f(\mathbf{x}^*) < 0$; a contradiction. This proves that $\nabla^2 f(\mathbf{x})$ is positive semidefinite. \square

The two necessary optimality conditions in Theorem 12.1 are called the *first-order* and the *second-order* conditions, respectively. The first-order condition says that the gradient must be zero at \mathbf{x}^* , and such a point is often called a *stationary point*. The second-order condition may be interpreted by f being "convex locally" at \mathbf{x}^* , although this is not a precise term. A stationary point which is neither a local minimum or a local maximum is called a *saddle point*. So, every neighborhood of a saddle point contains points with larger and points with smaller f -value.

Theorem 12.1 gives a connection to nonlinear equations. In order to find a stationary point we may solve $\nabla f(\mathbf{x}) = \mathbf{0}$, which is a $n \times n$ (usually nonlinear) system of equations. (The system is linear whenever f is a quadratic function.) One may solve this equation, for instance, by Newton's method and thereby get a candidate for a local minimum. Sometimes this approach works well, in particular if f has a unique local minimum and we have an initial point "sufficiently close". However, there are other better methods which we discuss later.

It is important to point out that any algorithm for finding a minimum of f *has to* be able to find a stationary point. Therefore algorithms in this area are typically iterative and move to gradually better points where the norm of the gradient becomes smaller, and eventually almost equal to zero.

As an example consider a convex quadratic function

$$f(\mathbf{x}) = (1/2) \mathbf{x}^T A \mathbf{x} - \mathbf{b}^T \mathbf{x}$$

where A is the (symmetric) Hessian matrix (constant equal to) A and this matrix is positive semidefinite. Then $\nabla f(\mathbf{x}) = A\mathbf{x} - \mathbf{b}$ so the first-order necessary optimality condition is

$$A\mathbf{x} = \mathbf{b}$$

which is a linear system of equations. If f is *strictly convex*, which happens when A is positive definite, then A is invertible and the unique solution is $\mathbf{x}^* = A^{-1}\mathbf{b}$. Thus, there is only one candidate for a local (and global) minimum, namely $\mathbf{x}^* = A^{-1}\mathbf{b}$. Actually, this is indeed a unique global minimum, but to verify this we need a suitable argument. One way is to use convexity (with results presented later) or an alternative is to use *sufficient* optimality conditions which we discuss next. The linear system $A\mathbf{x} = \mathbf{b}$, when A is positive definite, may be solved by several methods. A popular, and very fast, method is the *conjugate gradient method*. This method, and related methods, are discussed in detail in the course INF-MAT4360 *Numerical linear algebra* [10].

In order to present a sufficient optimality condition we need a result from linear algebra. Recall from linear algebra that a symmetric positive definite matrix has only real eigenvalues and all these are positive.

Proposition 12.2. Let A be an $n \times n$ symmetric positive definite matrix, and let $\lambda_n > 0$ denote its smallest eigenvalue. Then

$$\mathbf{h}^T A \mathbf{h} \geq \lambda_n \|\mathbf{h}\|^2 \quad (\mathbf{h} \in \mathbb{R}^n).$$

Proof. By the spectral theorem there is an orthogonal matrix V (containing the orthonormal eigenvectors as its columns) such that

$$A = V D V^T$$

where D is the diagonal matrix with the eigenvalues $\lambda_1, \dots, \lambda_n$ on the diagonal. Let $\mathbf{h} \in \mathbb{R}^n$ and define $\mathbf{y} = V^T \mathbf{h}$. Then $\|\mathbf{y}\| = \|\mathbf{h}\|$ and

$$\mathbf{h}^T A \mathbf{h} = \mathbf{h}^T V D V^T \mathbf{h} = \mathbf{y}^T D \mathbf{y} = \sum_{j=1}^n \lambda_j y_j^2 \geq \lambda_n \sum_{i=1}^n y_i^2 = \lambda_n \|\mathbf{y}\|^2 = \lambda_n \|\mathbf{h}\|^2.$$

□

Next we consider *sufficient* optimality conditions in the general differentiable case. These conditions are used to prove that a candidate point (say, found by an algorithm) is really a local minimum.

Theorem 12.3. Assume that $f : \mathbb{R}^n \rightarrow \mathbb{R}$ has continuous second order partial derivatives in some neighborhood of a point \mathbf{x}^* . Assume that $\nabla f(\mathbf{x}^*) = \mathbf{0}$ and $\nabla^2 f(\mathbf{x}^*)$ is positive definite. Then \mathbf{x}^* is a local minimum of f .

Proof. From Theorem 9.5 (second order Taylor) and Proposition 12.2 we get

$$\begin{aligned} f(\mathbf{x}^* + \mathbf{h}) &= f(\mathbf{x}^*) + \nabla f(\mathbf{x}^*)^T \mathbf{h} + \frac{1}{2} \mathbf{h}^T \nabla^2 f(\mathbf{x}^*) \mathbf{h} + \epsilon(\mathbf{h}) \|\mathbf{h}\|^2 \\ &\geq f(\mathbf{x}^*) + \frac{1}{2} \lambda_n \|\mathbf{h}\|^2 + \epsilon(\mathbf{h}) \|\mathbf{h}\|^2 \end{aligned}$$

where $\lambda_n > 0$ is the smallest eigenvalue of $\nabla^2 f(\mathbf{x}^*)$. Dividing here by $\|\mathbf{h}\|^2$ gives

$$(f(\mathbf{x}^* + \mathbf{h}) - f(\mathbf{x}^*)) / \|\mathbf{h}\|^2 = \frac{1}{2}\lambda_n + \epsilon(\mathbf{h})$$

Since $\lim_{\mathbf{h} \rightarrow \mathbf{0}} \epsilon(\mathbf{h}) = 0$, there is an r such that for $\|\mathbf{h}\| < r$, $|\epsilon(\mathbf{h})| < \lambda_n/4$. This implies that

$$(f(\mathbf{x}^* + \mathbf{h}) - f(\mathbf{x}^*)) / \|\mathbf{h}\|^2 \geq \lambda_n/4$$

for all \mathbf{h} with $\|\mathbf{h}\| < r$. This proves that \mathbf{x}^* is a local minimum of f . \square

We remark that the proof of the previous theorem actually shows that \mathbf{x}^* is a *strict* local minimum of f meaning that $f(\mathbf{x}^*)$ is strictly smaller than $f(\mathbf{x})$ for all other points \mathbf{x} in some neighborhood of \mathbf{x}^* . Note the difference between the necessary and the sufficient optimality conditions: a necessary condition is that $\nabla^2 f(\mathbf{x})$ is positive semidefinite, while a part of the sufficient condition is the stronger property that $\nabla^2 f(\mathbf{x})$ is positive definite.

Let us see what happens when we work with a convex function.

Theorem 12.4. Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be a convex function. Then a local minimum is also a global minimum. If, in addition, f is differentiable, then a point \mathbf{x}^* is a local (and then global) minimum of f if and only if

$$\nabla f(\mathbf{x}^*) = \mathbf{0}.$$

Proof. Let \mathbf{x}_1 be a local minimum. If \mathbf{x}_1 is not a global minimum, there is an $\mathbf{x}_2 \neq \mathbf{x}_1$ with $f(\mathbf{x}_2) < f(\mathbf{x}_1)$. Then for $0 < \lambda < 1$

$$f((1 - \lambda)\mathbf{x}_1 + \lambda\mathbf{x}_2) \leq (1 - \lambda)f(\mathbf{x}_1) + \lambda f(\mathbf{x}_2) < f(\mathbf{x}_1)$$

and this contradicts that $f(\mathbf{x}) \geq f(\mathbf{x}_1)$ for all \mathbf{x} in a neighborhood of \mathbf{x}^* . Therefore \mathbf{x}_1 must be a global minimum.

Assume f is convex and differentiable. Due to Theorem 12.1 we only need to show that if $\nabla f(\mathbf{x}^*) = \mathbf{0}$, then \mathbf{x}^* is a local and global minimum. So assume that $\nabla f(\mathbf{x}^*) = \mathbf{0}$. Then, from Theorem 10.9 we have

$$f(\mathbf{x}) \geq f(\mathbf{x}^*) + \nabla f(\mathbf{x}^*)^T (\mathbf{x} - \mathbf{x}^*)$$

for all $\mathbf{x} \in \mathbb{R}^n$. If $\nabla f(\mathbf{x}^*) = \mathbf{0}$, this directly shows that \mathbf{x}^* is a global minimum. \square

12.2 Methods

Algorithms for unconstrained optimization are iterative methods that generate a sequence of points with gradually smaller values on the function f which is to be minimized. There are two main types of algorithms in unconstrained optimization:

- *Line search methods:* Here one first chooses a *search direction* \mathbf{d}_k from the current point \mathbf{x}_k , using information about the function f . Then one chooses a *step length* α_k so that the new point

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$$

has a small, perhaps smallest possible, value on the halfline $\{\mathbf{x}_k + \alpha \mathbf{d}_k : \alpha \geq 0\}$. α_k describes how far one should go along the search direction. The problem of choosing α_k is a one-dimensional optimization problem. Sometimes we can find α_k exactly, and in such cases we refer to the method as *exact line search*. In cases where α_k can not be found analytically, algorithms can be used to approximate how we can get close to the minimum on the halfline. The method is then referred to as *backtracking line search*.

- *Trust region methods:* In these methods one chooses an approximation \hat{f}_k to the function in some neighborhood of the current point \mathbf{x}_k . The function \hat{f}_k is simpler than f and one minimizes \hat{f}_k (in the mentioned neighborhood) and let the next iterate \mathbf{x}_{k+1} be this minimizer.

These types are typically both based on quadratic approximation of f , but they differ in the order in which one chooses search direction and step size. In the following we only discuss the first type, the line search methods.

A very natural choice for search direction at a point \mathbf{x}_k is the negative gradient, $\mathbf{d}_k = -\nabla f(\mathbf{x}_k)$. Recall that the direction of maximum increase of a (differentiable) function f at a point \mathbf{x} is $\nabla f(\mathbf{x})$, and the direction of maximum decrease is $-\nabla f(\mathbf{x})$. To verify this, Taylor's theorem gives

$$f(\mathbf{x} + \mathbf{h}) = f(\mathbf{x}) + \nabla f(\mathbf{x}) \cdot \mathbf{h} + \frac{1}{2} \mathbf{h}^T \nabla^2 f(\mathbf{x} + t\mathbf{h}) \mathbf{h}.$$

So, for small \mathbf{h} , the first order term dominates and we would like to make this term small. By the Cauchy-Schwarz inequality¹

$$\nabla f(\mathbf{x}) \cdot \mathbf{h} \geq -\|\nabla f(\mathbf{x})\| \|\mathbf{h}\|$$

and equality holds for $\mathbf{h} = -\alpha \nabla f(\mathbf{x})$ for some $\alpha \geq 0$. In general, we call \mathbf{h} a *descent direction* at \mathbf{x} if $\nabla f(\mathbf{x}) \cdot \mathbf{h} < 0$. Thus, if we move in a descent direction from \mathbf{x} and make a sufficiently small step, the new point has a smaller f -value. With this background we shall in the following focus on *gradient methods* given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k \tag{12.4}$$

where the direction \mathbf{d}_k satisfies

$$\nabla f(\mathbf{x}_k) \cdot \mathbf{d}_k < 0 \tag{12.5}$$

There are two gradient methods we shall discuss:

¹The Cauchy-Schwarz' inequality says: $|\mathbf{u} \cdot \mathbf{v}| \leq \|\mathbf{u}\| \|\mathbf{v}\|$ for $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$.

- If we choose the search direction $\mathbf{d}_k = -\nabla f(\mathbf{x}_k)$, we get the *steepest descent method*

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k).$$

In each step it moves in the direction of the negative gradient. Sometimes this gives slow convergence, so other methods have been developed where other choices of direction \mathbf{d}_k are made.

- An important method is *Newton's method*

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla^2 f(\mathbf{x}_k)^{-1} \nabla f(\mathbf{x}_k). \quad (12.6)$$

This is the gradient method with $\mathbf{d}_k = -\nabla^2 f(\mathbf{x}_k)^{-1} \nabla f(\mathbf{x}_k)$; this vector \mathbf{d}_k is called the *Newton step*. The so-called *pure Newton method* is when one simply chooses step size $\alpha_k = 1$ for each k . To interpret this method consider the second order Taylor approximation of f in \mathbf{x}_k

$$f(\mathbf{x}_k + \mathbf{h}) \approx T_f^2(\mathbf{x}_k; \mathbf{x}_k + \mathbf{h}) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T \mathbf{h} + (1/2) \mathbf{h}^T \nabla^2 f(\mathbf{x}_k) \mathbf{h}$$

If we minimize this quadratic function w.r.t. \mathbf{h} , assuming $\nabla^2 f(\mathbf{x}_k)$ is positive definite, we get (see Exercise 7)

$$\mathbf{h} = -\nabla^2 f(\mathbf{x}_k)^{-1} \nabla f(\mathbf{x}_k)$$

which explains the Newton step.

In the following we follow the presentation in [1]. In a gradient method we need to *choose the step length*. This is the one-dimensional optimization problem

$$\min\{f(\mathbf{x} + \alpha \mathbf{d}) : \alpha \geq 0\}.$$

Sometimes (maybe not too often) we may solve this problem exactly. Most practical methods try some candidate α 's and pick the one with smallest f -value. Note that it is not necessary to compute the exact minimum (this may take too much time). The main thing is to assure that we get a sufficiently large decrease in f without making a too small step.

A popular step size rule is the *Armijo Rule*. Here one chooses (in advance) parameters s , a reduction factor β satisfying $0 < \beta < 1$, and $0 < \sigma < 1$. Define the integer

$$m_k = \min\{m : m \geq 0, f(\mathbf{x}_k) - f(\mathbf{x}_k + \beta^m s \mathbf{d}_k) \geq -\sigma \beta^m s \nabla f(\mathbf{x}_k)^T \mathbf{d}_k\} \quad (12.7)$$

and choose step length $\alpha_k = \beta^{m_k} s$. Here σ is typically chosen very small, e.g. $\sigma = 10^{-3}$. The parameter s fixes the search for step size to lie within the interval $[0, s]$. This can be important: for instance, we can set s so small that the initial step size we try is within the domain of definition for f . According to [1] β is usually chosen in $[1/10, 1/2]$. In the literature one may find a lot more information about step size rules and how they may be adjusted to the methods for finding search direction, see [1], [11].

Now, we return to the *choice of search direction* in the gradient method (12.4). A main question is whether it generates a sequence $\{\mathbf{x}_k\}_{k=1}^{\infty}$ which converges to a stationary point \mathbf{x}^* , i.e., where $\nabla f(\mathbf{x}^*) = \mathbf{0}$. It turns out that this may not be the case; one needs to be careful about the choice of \mathbf{d}_k to assure this convergence. The problem is that if \mathbf{d}_k tends to be nearly orthogonal to $\nabla f(\mathbf{x}_k)$ one may get into trouble. For this reason one introduces the following notion:

Definition 12.5 (Gradient related). $\{\mathbf{d}_k\}$ is called *gradient related* to $\{\mathbf{x}_k\}$ if for any subsequence $\{\mathbf{x}_{k_p}\}_{p=1}^{\infty}$ of $\{\mathbf{x}_k\}$ converging to a nonstationary point, then the corresponding subsequence $\{\mathbf{d}_{k_p}\}_{p=1}^{\infty}$ of $\{\mathbf{d}_k\}$ is bounded and $\limsup_{p \rightarrow \infty} \nabla f(\mathbf{x}_{k_p})^T \mathbf{d}_{k_p} < 0$.

What this condition assures is that $\|\mathbf{d}_k\|$ is not too small or large compared to $\|\nabla f(\mathbf{x}_k)\|$ and that the angle between the vectors \mathbf{d}_k and $\nabla f(\mathbf{x}_k)$ is not too close to 90° . The proof of the following theorem may be found in [1].

Theorem 12.6. Let $\{\mathbf{x}_k\}_{k=0}^{\infty}$ be generated by the gradient method (12.4), where $\{\mathbf{d}_k\}_{k=0}^{\infty}$ is gradient related to $\{\mathbf{x}_k\}_{k=0}^{\infty}$ and the step size α_k is chosen using the Armijo rule. Then every limit point of $\{\mathbf{x}_k\}_{k=0}^{\infty}$ is a stationary point.

We remark that in Theorem 12.6 the same conclusion holds if we use exact minimization as step size rule, i.e., $f(\mathbf{x}_k + \alpha \mathbf{d}_k)$ is minimized exactly with respect to α .

A very important property of a numerical algorithm is its convergence speed. Let us consider the steepest descent method first. It turns out that the convergence speed for this algorithm is very well explained by its performance on minimizing a quadratic function, so therefore the following result is important. In this theorem A is a symmetric positive definite matrix with eigenvalues $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n > 0$.

Theorem 12.7. If the steepest descent method $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)$ using exact line search is applied to the quadratic function $f(\mathbf{x}) = \mathbf{x}^T A \mathbf{x}$ where A is positive definite, then (the minimum value is 0 and)

$$f(\mathbf{x}_{k+1}) \leq m_A f(\mathbf{x}_k)$$

where $m_A = ((\lambda_1 - \lambda_n)/(\lambda_1 + \lambda_n))^2$.

The proof may be found in [1]. Thus, if the largest eigenvalue is much larger than the smallest one, m_A will be nearly 1 and one typically have slow convergence. In this case we have $m_A \approx \text{cond}(A)$ where $\text{cond}(A) = \lambda_1/\lambda_n$ is the *condition number* of the matrix A . So the rule is: if the condition number of A is small we get fast convergence, but if $\text{cond}(A)$ is large, there will be

slow convergence. A similar behavior holds for most functions f because locally near a minimum point the function is very close to its second order Taylor approximation in \mathbf{x}^* which is a quadratic function with $A = \nabla^2 f(\mathbf{x}^*)$.

Thus, Theorem 12.7 says that the sequence obtained in the steepest descent method converges linearly to a stationary point (at least for quadratic functions).

We now turn to Newton's method.

Newton's method for unconstrained optimization:

1. Choose an initial point \mathbf{x}_0 .
2. For $k = 1, 2, \dots$ do
 - (i) (Newton step) $\mathbf{d}_k := -\nabla^2 f(\mathbf{x})^{-1} \nabla f(\mathbf{x})$; $\eta = -\nabla f(\mathbf{x})^T \mathbf{d}_k$
 - (ii) (Stopping criterion) If $\eta/2 < \epsilon$: stop.
 - (iii) (Line search) Use backtracking line search to find step size α_k
 - (iv) (Update) $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{d}_k$

Recall that the pure Newton step minimizes the second order Taylor approximation of f at the current iterate \mathbf{x}_k . Thus, if the function we minimize is quadratic, we are done in one step. Similarly, if the function can be well approximated by a quadratic function, then one would expect fast convergence.

We shall give a result on the convergence of Newton's method (see [2] for further details). When A is symmetric, we let $\lambda_{\min}(A)$ denote that smallest eigenvalue of A .

For the convergence result we need a lemma on strictly convex functions. Assume that \mathbf{x}_0 is a starting point for Newton's method and let $S = \{\mathbf{x} \in \mathbb{R}^n : f(\mathbf{x}) \leq f(\mathbf{x}_0)\}$. We shall assume that f is continuous and convex, and this implies that S is a closed convex set. We also assume that f has a minimum point \mathbf{x}^* which then must be a global minimum. Moreover the minimum point will be unique due to a strict convexity assumption on f . Let $f^* = f(\mathbf{x}^*)$ be the optimal value.

The following lemma says that for a convex functions as just described, a point is nearly a minimum point (in terms of the f -value) whenever the gradient is small in that point.

Lemma 12.8. Assume that f is convex as above and that $\lambda_{\min}(\nabla^2 f(\mathbf{x})) \geq m$ for all $\mathbf{x} \in S$. Then

$$f(\mathbf{x}) - f^* \leq \frac{1}{2m} \|\nabla f(\mathbf{x})\|^2. \quad (12.8)$$

Proof. From Theorem 9.4, the second order Taylor' theorem, we have for each $\mathbf{x}, \mathbf{y} \in S$

$$f(\mathbf{y}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T (\mathbf{y} - \mathbf{x}) + (1/2)(\mathbf{y} - \mathbf{x})^T \nabla^2 f(\mathbf{z})(\mathbf{y} - \mathbf{x})$$

for suitable \mathbf{z} on the line segment between \mathbf{x} and \mathbf{y} . Here a lower bound for the quadratic term is $(m/2)\|\mathbf{y} - \mathbf{x}\|^2$, due to Proposition 12.2. Therefore

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x}) + (m/2)\|\mathbf{y} - \mathbf{x}\|^2.$$

Now, fix \mathbf{x} and view the expression on the right-hand side as a quadratic function of \mathbf{y} . This function is minimized for $\mathbf{y}^* = \mathbf{x} - (1/m)\nabla f(\mathbf{x})$. So, by inserting $\mathbf{y} = \mathbf{y}^*$ above we get

$$\begin{aligned} f(\mathbf{y}) &\geq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y}^* - \mathbf{x}) + (m/2)\|\mathbf{y}^* - \mathbf{x}\|^2 \\ &= f(\mathbf{x}) - \frac{1}{2m}\|\nabla f(\mathbf{x})\|^2 \end{aligned}$$

This holds for every $\mathbf{y} \in S$ so letting $\mathbf{y} = \mathbf{x}^*$ gives

$$f^* = f(\mathbf{x}^*) \geq f(\mathbf{x}) - \frac{1}{2m}\|\nabla f(\mathbf{x})\|^2$$

which proves the desired inequality. \square

In the following convergence result we consider a function f as in Lemma 12.8. Moreover, we assume that the Hessian matrix is Lipschitz continuous over S ; this is essentially a bound on the third derivatives of f . We do not give the complete proof (it is quite long), but consider some of the main ideas. Recall the definition of the set S from above. Recall that the *spectral norm* of a square matrix A is defined by

$$\|A\|_2 = \max_{\|\mathbf{x}\|=1} \|A\mathbf{x}\|.$$

It is a fact that $\|A\|_2$ is equal to the largest singular value of A .

Theorem 12.9. Let f be convex and twice continuously differentiable and assume that

- (i) $\lambda_{\min}(\nabla^2 f(\mathbf{x})) \geq m$ for all $\mathbf{x} \in S$.
- (ii) $\|\nabla^2 f(\mathbf{x}) - \nabla^2 f(\mathbf{y})\|_2 \leq L\|\mathbf{x} - \mathbf{y}\|$ for all $\mathbf{x} \in S$.

Moreover, assume that f has a minimum point \mathbf{x}^* . Then Newton's method generates a sequence $\{\mathbf{x}_k\}_{k=0}^\infty$ that converges to \mathbf{x}^* . From a certain k' the convergence speed is quadratic.

Proof. Define $f^* = f(\mathbf{x}^*)$. It is possible to show that there are numbers η and $\gamma > 0$ with $0 < \eta \leq m^2/L$ such that the following holds for each k :

- (i) If $\|\nabla f(\mathbf{x}_k)\| \geq \eta$, then

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) - \gamma. \quad (12.9)$$

- (ii) If $\|\nabla f(\mathbf{x}_k)\| < \eta$, then backtracking line search gives $\alpha_k = 1$ and

$$\frac{L}{2m^2}\|\nabla f(\mathbf{x}_{k+1})\| \leq \left(\frac{L}{2m^2}\|\nabla f(\mathbf{x}_k)\|\right)^2. \quad (12.10)$$

We omit the proof of this fact; it may be found in [2].

We may now prove that if $\|\nabla f(\mathbf{x}_k)\| < \eta$, then also $\|\nabla f(\mathbf{x}_{k+1})\| < \eta$. This follows from (ii) above and the fact (assumption) $\eta \leq m^2/L$. Therefore, as soon as case (ii) occurs in the iterative process, in all the remaining iterations case (ii) will occur. Actually, as soon as case (ii) “kicks in” quadratic convergence starts as we shall see now. So assume that case (ii) occurs from a certain k . (Below we show that such k must exist.)

Define $\mu_l = \frac{L}{2m^2} \|\nabla f(\mathbf{x}_l)\|$ for each $l \geq k$. Then $0 \leq \mu_k < 1/2$ as $\eta \leq m^2/L$. From what we just saw and (12.10)

$$\mu_{l+1} \leq \mu_l^2 \quad (l \geq k).$$

So (by induction)

$$\mu_l \leq \mu_k^{2^{l-k}} \leq (1/2)^{2^{l-k}} \quad (l = k+1, \dots).$$

Next, from Lemma 12.8

$$f(\mathbf{x}_k) - f^* \leq \frac{1}{2m} \|\nabla f(\mathbf{x}_l)\|^2 \leq \frac{2m^3}{L^2} (1/2)^{2^{l-k+1}} \quad (l \geq k).$$

This inequality shows that $f(\mathbf{x}_l) \rightarrow f^*$, and since the minimum point is unique, we must have $\mathbf{x}_l \rightarrow \mathbf{x}^*$. Moreover, it follows that the convergence is quadratic.

It only remains to explain why case (ii) above indeed occurs for some k . In each iteration of type (i) f is decreased by at least γ , as seen from equation (12.10), so the number of such iterations must be bounded by

$$(f(\mathbf{x}_0) - f^*)/\gamma$$

which is a finite number. Finally, the proof of the statements in connection with (i) and (ii) above is quite long and one derives several inequalities using the convexity properties of f . \square

From the proof it is also possible to say something about how many iterations that are needed to reach a certain accuracy. In fact, if $\epsilon > 0$ a bound on the number of iterations until $f(\mathbf{x}_k) \leq f^* + \epsilon$ is

$$(f(\mathbf{x}_0) - f^*)/\gamma + \log_2 \log_2 \frac{2m^3}{\epsilon L^2}.$$

Here γ is the parameter introduced in the proof above. The second term in this expression (the logarithmic term) grows very slowly as ϵ is decreased, and it may roughly be replaced by the constant 6. So, whenever the second stage (case (ii) in the proof) occurs, the convergence is extremely fast, it takes about 6 more Newton iterations. Note that quadratic convergence means, roughly, that the number of correct digits in the answer doubles for every iteration.

Exercises for Section 12.2

Ex. 1 — Consider the function $f(x_1, x_2) = x_1^2 + ax_2^2$ where $a > 0$ is a parameter. Draw some of the level sets of f (for different levels) for each a in the set $\{1, 4, 100\}$. Also draw the gradient in a few points on these level sets.

Ex. 2 — State and prove a theorem similar to Theorem 12.1 for maximization problems.

Ex. 3 — Let $f(\mathbf{x}) = \mathbf{x}^T A \mathbf{x}$ where A is a symmetric $n \times n$ matrix. Assume that A is indefinite, so it has both positive and negative eigenvalues. Show that $\mathbf{x} = \mathbf{0}$ is a saddlepoint of f .

Ex. 4 — Let $f(x_1, x_2) = 4x_1 + 6x_2 + x_1^2 + 2x_2^2$. Find all stationary points and determine if they are minimum, maximum or saddlepoints. Do the same for the function $g(x_1, x_2) = 4x_1 + 6x_2 + x_1^2 - 2x_2^2$.

Ex. 5 — The function $f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$ is called the *Rosenbrock function*. Compute the gradient and the Hessian matrix at every point \mathbf{x} . Find every local minimum. Also draw some of the level sets (contour lines) of f using Matlab.

Ex. 6 — Let $f(\mathbf{x}) = (1/2)\mathbf{x}^T A \mathbf{x} - \mathbf{b}^T \mathbf{x}$ where A is a positive definite $n \times n$ matrix. Consider the steepest descent method applied to the minimization of f , where we assume exact line search is used. Assume that the search direction happens to be equal to an eigenvector of A . Show that then the minimum is reached in just one step.

Ex. 7 — Consider the second order Taylor approximation

$$T_f^2(\mathbf{x}; \mathbf{x} + \mathbf{h}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{h} + (1/2)\mathbf{h}^T \nabla^2 f(\mathbf{x}) \mathbf{h}.$$

- Show that $\nabla_{\mathbf{h}} T_f^2 = \nabla f(\mathbf{x})^T + \nabla^2 f(\mathbf{x}) \mathbf{h}$.
- Minimizing T_f^2 with respect to \mathbf{h} implies solving $\nabla_{\mathbf{h}} T_f^2 = \mathbf{0}$, i.e. $\nabla f(\mathbf{x})^T + \nabla^2 f(\mathbf{x}) \mathbf{h} = \mathbf{0}$ from a.. If $\nabla^2 f(\mathbf{x})$ is positive definite, explain that it is invertible, so that this equation has the unique solution $\mathbf{h} = -\nabla^2 f(\mathbf{x}_k)^{-1} \nabla f(\mathbf{x}_k)$, as previously noted for the Newton step.

Ex. 8 — Implement the steepest descent method. Test the algorithm on the functions in exercises 4 and 5. Use different starting points.

Ex. 9 — Implement a function

```
function alpha=armijorule(f,df,x,d)
```

which returns α chosen according to the Armijo rule for a function f with the given gradient, at point x , with search direction d . The function should compute m_k from Equation (12.7) with $\beta = 0.2$, $s = 0.5$, $\sigma = 10^{-3}$, and return $\alpha = \beta^{m_k} s$.

Ex. 10 — Write a function

```
[xopt,numit]=newtonbacktrack(f,df,d2f,x0)
```

which performs Newton's method for unconstrained optimization. The input parameters are the function, its gradient, its Hesse matrix, and the initial point. The function should also return the number of iterations, and at each iteration write the corresponding function value. The function should use backtracking line search with the function `armijorule` from the previous exercise. Test the algorithm on the functions in exercises 4 and 5. Use different starting points.

Chapter 13

Constrained optimization - theory

In this chapter we consider constrained optimization problems. A general optimization problem is

$$\text{minimize } f(\mathbf{x}) \text{ subject to } \mathbf{x} \in S$$

where $S \subseteq \mathbb{R}^n$ is a given set and $f : S \rightarrow \mathbb{R}$. We here focus on a very general optimization problem which often occurs in applications. Consider the *nonlinear optimization problem* with equality/inequality constraints

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}) \\ &\text{subject to} && \\ &&& h_i(\mathbf{x}) = 0 \quad (i \leq m) \\ &&& g_j(\mathbf{x}) \leq 0 \quad (j \leq r) \end{aligned} \tag{13.1}$$

where f, h_1, h_2, \dots, h_m and g_1, g_2, \dots, g_r are continuously differentiable functions from \mathbb{R}^n into \mathbb{R} . A point \mathbf{x} satisfying all the $m + r$ constraints will be called *feasible*. Thus, we look for a feasible point with smallest f -value.

Our goal is to establish optimality conditions for this problem, starting with the special case with only equality constraints. Then we discuss algorithms for solving this problem. Our presentation is strongly influenced by [2] and [1].

13.1 Equality constraints and the Lagrangian

Consider the nonlinear optimization problem with equality constraints

$$\begin{array}{ll}
\text{minimize} & f(\mathbf{x}) \\
\text{subject to} & \\
& h_i(\mathbf{x}) = 0 \quad (i \leq m)
\end{array} \tag{13.2}$$

where f and h_1, h_2, \dots, h_m are continuously differentiable functions from \mathbb{R}^n into \mathbb{R} . We introduce the vector field $\mathbf{H} = (h_1, h_2, \dots, h_m)$, so $\mathbf{H} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ and $\mathbf{H}(\mathbf{x}) = (h_1(\mathbf{x}), h_2(\mathbf{x}), \dots, h_m(\mathbf{x}))$.

We first establish necessary optimality conditions for this problem. A point $\mathbf{x} \in \mathbb{R}^n$ is called *regular* if the gradient vectors $\nabla h_i(\mathbf{x}^*)$ ($i \leq m$) are linearly independent.

Theorem 13.1. Let \mathbf{x}^* be a local minimum in problem (13.1) and assume that \mathbf{x}^* is a regular point. Then there is a unique vector $\boldsymbol{\lambda}^* = (\lambda_1^*, \lambda_2^*, \dots, \lambda_m^*) \in \mathbb{R}^m$ such that

$$\nabla f(\mathbf{x}^*) + \sum_{i=1}^m \lambda_i^* \nabla h_i(\mathbf{x}^*) = \mathbf{0}. \tag{13.3}$$

If f and each h_i are twice continuously differentiable, then the following also holds

$$\mathbf{h}^T (\nabla^2 f(\mathbf{x}^*) + \sum_{i=1}^m \lambda_i^* \nabla^2 h_i(\mathbf{x}^*)) \mathbf{h} \geq 0 \quad \text{for all } \mathbf{h} \in T(\mathbf{x}^*) \tag{13.4}$$

where $T(\mathbf{x}^*)$ is the subspace $T(\mathbf{x}^*) = \{\mathbf{h} \in \mathbb{R}^n : \nabla h_i(\mathbf{x}^*) \cdot \mathbf{h} = 0 \ (i \leq m)\}$.

The numbers λ_i^* in this theorem are called the *Lagrangian multipliers*. Note that the Lagrangian multiplier vector $\boldsymbol{\lambda}^*$ is unique; this follows directly from the linear independence assumption as \mathbf{x}^* is assumed regular. The theorem may also be stated in terms of the *Lagrangian function* $L : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$ given by

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_{i=1}^m \lambda_i h_i(\mathbf{x}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{H}(\mathbf{x}) \quad (\mathbf{x} \in \mathbb{R}^n, \boldsymbol{\lambda} \in \mathbb{R}^m).$$

Then

$$\nabla_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}) = \nabla f(\mathbf{x}) + \sum_i \lambda_i \nabla h_i$$

$$\nabla_{\boldsymbol{\lambda}} L(\mathbf{x}, \boldsymbol{\lambda}) = \mathbf{H}(\mathbf{x}).$$

Therefore, the first order conditions in Theorem 13.1 may be rewritten as follows

$$\nabla_{\mathbf{x}} L(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \mathbf{0}, \quad \nabla_{\boldsymbol{\lambda}} L(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \mathbf{0}.$$

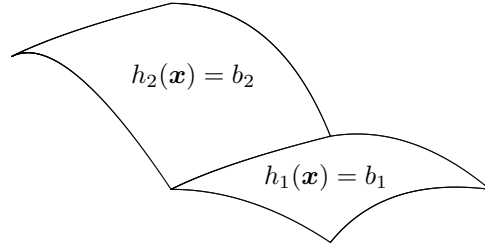


Figure 13.1: The two surfaces $h_1(\mathbf{x}) = b_1$ og $h_2(\mathbf{x}) = b_2$ intersect each other in a curve. Along this curve the constraints are fulfilled

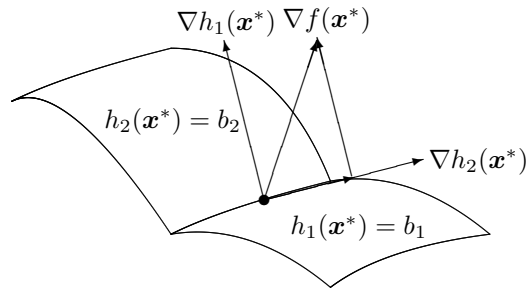


Figure 13.2: $\nabla f(\mathbf{x}^*)$ as a linear combination of $\nabla h_1(\mathbf{x}^*)$ and $\nabla h_2(\mathbf{x}^*)$

Here the second equation simply means that $\mathbf{H}(\mathbf{x}) = \mathbf{0}$. These two equations say that $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$ is a *stationary point for the Lagrangian*, and it is a system of $n + m$ (possibly nonlinear) equations in $n + m$ variables.

We may interpret the theorem in the following way. At the point \mathbf{x}^* the linear subspace $T(\mathbf{x}^*)$ consist of the “first order feasible directions”. Actually, if each h_i is linear, then $T(\mathbf{x}^*)$ consists of those \mathbf{h} such that $\mathbf{x}^* + \mathbf{h}$ is feasible, i.e., $h_i(\mathbf{x}^* + \mathbf{h}) = 0$ for each $i \leq m$. Thus, (13.3) says that in a local minimum \mathbf{x}^* the gradient $\nabla f(\mathbf{x}^*)$ is orthogonal to the subspace $T(\mathbf{x}^*)$ of the first order feasible variations. This is reasonable since otherwise there would be a feasible direction in which f would decrease. In Figure 13.1 we have plotted a curve where two constraints are fulfilled. In Figure 13.2 we have then shown an interpretation of Theorem 13.1.

Note that this necessary optimality condition corresponds to the condition $\nabla f(\mathbf{x}^*) = \mathbf{0}$ in the unconstrained case. The second condition (13.4) is a sim-

ilar generalization of the second order condition in Theorem 12.1 (saying that $\nabla^2 f(\mathbf{x}^*)$ is positive semidefinite).

It is possible to prove the theorem by eliminating variables based on the equations and thereby reducing the problem to an unconstrained one. Another proof, which we shall present below is based on the *penalty approach*. This approach is also interesting as it leads to algorithms for actually solving the problem.

Proof. (Theorem 13.1) For $k = 1, 2, \dots$ consider the modified objective function

$$F^k(\mathbf{x}) = f(\mathbf{x}) + (k/2)\|\mathbf{H}(\mathbf{x})\|^2 + (\alpha/2)\|\mathbf{x} - \mathbf{x}^*\|^2$$

where \mathbf{x}^* is the local minimum under consideration, and α is a positive constant. The second term is a penalty term for violating the constraints and the last term is there for proof technical reasons. As \mathbf{x}^* is a local minimum there is an $\epsilon > 0$ such that $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in \bar{B}(\mathbf{x}^*; \epsilon)$. Choose now an optimal solution \mathbf{x}^k of the problem $\min\{F^k(\mathbf{x}) : \mathbf{x} \in \bar{B}(\mathbf{x}^*; \epsilon)\}$; the existence here follows from the extreme value theorem (F^k is continuous and the ball is compact). For every k

$$F^k(\mathbf{x}^k) = f(\mathbf{x}^k) + (k/2)\|\mathbf{H}(\mathbf{x}^k)\|^2 + (\alpha/2)\|\mathbf{x}^k - \mathbf{x}^*\|^2 \leq F^k(\mathbf{x}^*) = f(\mathbf{x}^*).$$

By letting $k \rightarrow \infty$ in this inequality we conclude that $\lim_{k \rightarrow \infty} \|\mathbf{H}(\mathbf{x}^k)\| = 0$. So every limit point $\bar{\mathbf{x}}$ of the sequence $\{\mathbf{x}^k\}$ satisfies $\mathbf{H}(\bar{\mathbf{x}}) = \mathbf{0}$. The inequality above also implies (by dropping a term on the left-hand side) that $f(\mathbf{x}^k) + (\alpha/2)\|\mathbf{x}^k - \mathbf{x}^*\|^2 \leq f(\mathbf{x}^*)$ for all k , so by passing to the limit we get

$$f(\bar{\mathbf{x}}) + (\alpha/2)\|\bar{\mathbf{x}} - \mathbf{x}^*\|^2 \leq f(\mathbf{x}^*) \leq f(\bar{\mathbf{x}})$$

where the last inequality follows from the facts that $\bar{\mathbf{x}} \in \bar{B}(\mathbf{x}^*; \epsilon)$ and $\mathbf{H}(\bar{\mathbf{x}}) = \mathbf{0}$. Clearly, this gives $\bar{\mathbf{x}} = \mathbf{x}^*$. We have therefore shown that the sequence $\{\mathbf{x}^k\}$ converges to the local minimum \mathbf{x}^* . Since \mathbf{x}^* is the center of the ball $\bar{B}(\mathbf{x}^*; \epsilon)$, the points \mathbf{x}^k lie in the interior of S for suitably large k . The conclusion is then that \mathbf{x}^k is the *unconstrained* minimum of F^k when k is sufficiently large. We may therefore apply Theorem 12.1 so $\nabla F^k(\mathbf{x}^k) = \mathbf{0}$, so

$$\mathbf{0} = \nabla F^k(\mathbf{x}^k) = \nabla f(\mathbf{x}^k) + k\mathbf{H}'(\mathbf{x}^k)^T \mathbf{H}(\mathbf{x}^k) + \alpha(\mathbf{x}^k - \mathbf{x}^*). \quad (13.5)$$

Here \mathbf{H}' denotes the Jacobi matrix of \mathbf{H} . For suitably large k the matrix $\mathbf{H}'(\mathbf{x}^k)\mathbf{H}'(\mathbf{x}^k)^T$ is invertible (as the rows of $\mathbf{H}'(\mathbf{x}^k)$ are linearly independent due to $\text{rank}(\mathbf{H}'(\mathbf{x}^*)) = m$ and a continuity argument). Multiply equation (13.5) by $(\mathbf{H}'(\mathbf{x}^k)\mathbf{H}'(\mathbf{x}^k)^T)^{-1}\mathbf{H}'(\mathbf{x}^k)$ to obtain

$$k\mathbf{H}(\mathbf{x}^k) = -(\mathbf{H}'(\mathbf{x}^k)\mathbf{H}'(\mathbf{x}^k)^T)^{-1}\mathbf{H}'(\mathbf{x}^k)(\nabla f(\mathbf{x}^k) + \alpha(\mathbf{x}^k - \mathbf{x}^*)).$$

Letting $k \rightarrow \infty$ we see that the sequence $\{k\mathbf{H}(\mathbf{x}^k)\}$ is convergent and its limit point $\boldsymbol{\lambda}^*$ is given by

$$\boldsymbol{\lambda}^* = -(\mathbf{H}'(\mathbf{x}^*)\mathbf{H}'(\mathbf{x}^*)^T)^{-1}\mathbf{H}'(\mathbf{x}^*)\nabla f(\mathbf{x}^*).$$

Finally, by passing to the limit in (13.5) we get

$$\mathbf{0} = \nabla f(\mathbf{x}^*) + \mathbf{H}'(\mathbf{x}^*)^T \boldsymbol{\lambda}^*$$

This proves the first part of the theorem; we omit proving the second part which may be found in [1]. \square

The first order necessary condition (13.3) along with the constraints $\mathbf{H}(\mathbf{x}) = \mathbf{0}$ is a system of $n + m$ equations in the $n + m$ variables x_1, x_2, \dots, x_n and $\lambda_1, \lambda_2, \dots, \lambda_m$. One may use e.g. Newton's method for solving these equations and find a candidate for an optimal solution. But usually there are better numerical methods for solving the optimization (13.1), as we shall see soon.

Necessary optimality conditions are used for finding a candidate solution for being optimal. In order to verify optimality we need *sufficient* optimality conditions.

Theorem 13.2. Assume that f and \mathbf{H} are twice continuously differentiable functions. Moreover, let \mathbf{x}^* be a point satisfying the first order necessary optimality condition (13.3) and the following condition

$$\mathbf{y}^T \nabla^2 L(\mathbf{x}^*, \boldsymbol{\lambda}^*) \mathbf{y} > 0 \quad \text{for all } \mathbf{y} \neq \mathbf{0} \text{ with } \mathbf{H}'(\mathbf{x}^*)^T \mathbf{y} = 0 \quad (13.6)$$

where $\nabla^2 L(\mathbf{x}^*, \boldsymbol{\lambda}^*)$ is the Hessian of the Lagrangian function with second order partial derivatives with respect to \mathbf{x} . Then \mathbf{x}^* is a (strict) local minimum of f subject to $\mathbf{H}(\mathbf{x}) = \mathbf{0}$.

This theorem may be proved (see [1] for details) by considering the *augmented Lagrangian function*

$$L_c(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{H}(\mathbf{x}) + (c/2) \|\mathbf{H}(\mathbf{x})\|^2 \quad (13.7)$$

where c is a positive scalar. This is in fact the Lagrangian function in the modified problem

$$\text{minimize } f(\mathbf{x}) + (c/2) \|\mathbf{H}(\mathbf{x})\|^2 \quad \text{subject to } \mathbf{H}(\mathbf{x}) = \mathbf{0} \quad (13.8)$$

and this problem must have the same local minima as the problem of minimizing $f(\mathbf{x})$ subject to $\mathbf{H}(\mathbf{x}) = \mathbf{0}$. The objective function in (13.8) contains the *penalty term* $(c/2) \|\mathbf{H}(\mathbf{x})\|^2$ which may be interpreted as a penalty (increased function value) for violating the constraint $\mathbf{H}(\mathbf{x}) = \mathbf{0}$. In connection with the proof of Theorem 13.2 based on the augmented Lagrangian one also obtains the following interesting and useful fact: *if \mathbf{x}^* and $\boldsymbol{\lambda}^*$ satisfy the sufficient conditions in Theorem 13.2 then there exists a positive \bar{c} such that for all $c \geq \bar{c}$ the point \mathbf{x}^* is also a local minimum of the augmented Lagrangian $L_c(\cdot, \boldsymbol{\lambda}^*)$.* Thus, the original constrained problem has been converted to an unconstrained one involving the augmented Lagrangian. And, as we know, unconstrained problems are easier to solve (solve the equations saying that the gradient is equal to zero).

13.2 Inequality constraints and KKT

We now consider the general nonlinear optimization problem where there are both equality and inequality constraints. The problem is then

$$\begin{aligned} & \text{minimize} && f(\mathbf{x}) \\ & \text{subject to} && \\ & && h_i(\mathbf{x}) = 0 \quad (i \leq m) \\ & && g_j(\mathbf{x}) \leq 0 \quad (j \leq r) \end{aligned} \tag{13.9}$$

We assume, as usual, that all these functions are continuously differentiable real-valued functions defined on \mathbb{R}^n . In short form we write the constraints as $\mathbf{H}(\mathbf{x}) = \mathbf{0}$ and $\mathbf{G}(\mathbf{x}) \leq \mathbf{0}$ where we let $\mathbf{H} = (h_1, h_2, \dots, h_m)$ and $\mathbf{G} = (g_1, g_2, \dots, g_r)$.

A main difficulty in problems with inequality constraints is to determine which of the inequalities that are active in an optimal solution. If we knew the active inequalities, we would essentially have a problem with only equality constraints, $\mathbf{H}(\mathbf{x}) = \mathbf{0}$ plus the active equalities, i.e., a problem of the form discussed in the previous section. For very small problems (solvable by hand-calculation) a direct method is to consider all possible choices of active inequalities and solve the corresponding equality-constrained problem by looking at the Lagrangian function.

Interestingly, one may also transform the problem (13.9) into the following equality-constrained problem

$$\begin{aligned} & \text{minimize} && f(\mathbf{x}) \\ & \text{subject to} && \\ & && h_i(\mathbf{x}) = 0 \quad (i \leq m) \\ & && g_j(\mathbf{x}) + z_j^2 = 0 \quad (j \leq r). \end{aligned} \tag{13.10}$$

We have introduced extra variables z_j , one for each inequality. The square of these variables represent slack in each of the original inequalities. Note that there is no sign constraint on z_j . Clearly, the problems (13.9) and (13.10) are equivalent. This transformation can also be useful computationally. Moreover, it is useful theoretically as one may apply the optimality conditions from the previous section to problem (13.10) to derive the theorem below (see [1]).

We now present a main result in nonlinear optimization. It gives optimality conditions for this problem, and these conditions are called the *Karush-Kuhn-Tucker conditions*, or simply the *KKT conditions*. In order to present the KKT conditions we introduce the *Lagrangian function* $L : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^r \rightarrow \mathbb{R}$ given by

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) + \sum_{i=1}^m \lambda_i h_i(\mathbf{x}) + \sum_{j=1}^r \mu_j g_j(\mathbf{x}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{H}(\mathbf{x}) + \boldsymbol{\mu}^T \mathbf{G}(\mathbf{x}).$$

The gradient of L with respect to \mathbf{x} is given by

$$\nabla_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \nabla f(\mathbf{x}) + \sum_{i=1}^m \lambda_i \nabla h_i(\mathbf{x}) + \sum_{j=1}^r \mu_j \nabla g_j(\mathbf{x}).$$

The Hessian matrix of L at $(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ containing second order partial derivatives of L with respect to \mathbf{x} will be denoted by $\nabla_{\mathbf{x}\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$. Finally, the indices of the active inequalities at \mathbf{x} is denoted by $A(\mathbf{x})$, so $A(\mathbf{x}) = \{j \leq r : g_j(\mathbf{x}) = 0\}$. A point \mathbf{x} is called *regular* if $\{\nabla h_1(\mathbf{x}), \dots, \nabla h_m(\mathbf{x})\} \cup \{\nabla g_i(\mathbf{x}) : i \in A(\mathbf{x})\}$ is linearly independent.

In the following theorem the first part contains necessary conditions while the second part contains sufficient conditions for optimality.

Theorem 13.3. Consider problem (13.9) with the usual differentiability assumptions.

(i) Let \mathbf{x}^* be a local minimum of this problem and assume that \mathbf{x}^* is a regular point. Then there are unique Lagrange multiplier vectors $\boldsymbol{\lambda}^* = (\lambda_1^*, \lambda_2^*, \dots, \lambda_m^*)$ and $\boldsymbol{\mu}^* = (\mu_1^*, \mu_2^*, \dots, \mu_r^*)$ such that

$$\begin{aligned} \nabla_{\mathbf{x}} L(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) &= \mathbf{0} \\ \mu_j^* &\geq 0 & (j \leq r) \\ \mu_j^* &= 0 & (j \notin A(\mathbf{x}^*)). \end{aligned} \tag{13.11}$$

If f , g and h are twice continuously differentiable, then the following also holds

$$\mathbf{y}^T \nabla_{\mathbf{x}\mathbf{x}}^2 L(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \mathbf{y} \geq 0 \tag{13.12}$$

for all \mathbf{y} with $\nabla h_i(\mathbf{x}^*)^T \mathbf{y} = 0$ ($i \leq m$) and $\nabla g_j(\mathbf{x}^*)^T \mathbf{y} = 0$ ($j \in A(\mathbf{x}^*)$).

(ii) Assume that \mathbf{x}^* , $\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^*$ are such that \mathbf{x}^* is a feasible point and (13.11) holds. Assume, moreover, that (13.12) holds with strict inequality for each \mathbf{y} . Then \mathbf{x}^* is a (strict) local minimum in problem (13.9).

Proof. We shall derive this result from Theorem 13.1.

(i) By assumption \mathbf{x}^* is a local minimum of problem (13.9), and \mathbf{x}^* is a regular point. Consider the *constrained* problem

$$\begin{aligned}
& \text{minimize} && f(\mathbf{x}) \\
& \text{subject to} && \\
& && h_i(\mathbf{x}) = 0 \quad (i \leq m) \\
& && g_j(\mathbf{x}) = 0 \quad (j \in A(\mathbf{x}^*))
\end{aligned} \tag{13.13}$$

which is obtained by removing all inactive constraints in \mathbf{x}^* . Then \mathbf{x}^* must be a local minimum in (13.13); otherwise there would be a point \mathbf{x}' in the neighborhood of \mathbf{x}^* which is feasible in (13.13) and satisfying $f(\mathbf{x}') < f(\mathbf{x}^*)$. By choosing \mathbf{x}' sufficiently near \mathbf{x}^* we would get $g_j(\mathbf{x}') < 0$ for all $j \in A(\mathbf{x}^*)$, contradicting that \mathbf{x}^* is a local minimum in (13.9). Therefore we may apply Theorem 13.1 to problem (13.13) and by regularity of \mathbf{x}^* there must be unique Lagrange multiplier vectors $\boldsymbol{\lambda}^* = (\lambda_1^*, \lambda_2^*, \dots, \lambda_m^*)$ and μ_j^* ($j \in A(\mathbf{x}^*)$) such that

$$\nabla f(\mathbf{x}^*) + \sum_{i=1}^m \lambda_i^* \nabla h_i(\mathbf{x}^*) + \sum_{j \in A(\mathbf{x}^*)} \mu_j^* \nabla g_j(\mathbf{x}^*) = \mathbf{0}$$

By defining $\mu_j = 0$ for $j \notin A(\mathbf{x}^*)$ we get (13.11), except for the nonnegativity of μ .

The remaining part of the theorem may be proved, after some work, by studying the equality-constrained reformulation (13.10) of (13.9) and applying Theorem 13.1 to (13.10). The details may be found in [1]. \square

The KKT conditions have an interesting geometrical interpretation. They say that $-\nabla f(\mathbf{x}^*)$ may be written as linear combination of the gradients of the h_i 's plus a nonnegative linear combination of the gradients of the g_j 's that are active at \mathbf{x}^* .

We remark that the assumption that \mathbf{x}^* is a regular point may be too restrictive in some situations, for instance there may be more than n active inequalities in \mathbf{x}^* . There exist several other weaker assumptions that assure the existence of Lagrangian multipliers (and similar necessary conditions). Let us briefly say a bit more on this matter.

Definition 13.4 (Tangent vector). Let $C \subseteq \mathbb{R}^n$ and let $\mathbf{x} \in C$. A vector $\mathbf{d} \in \mathbb{R}^n$ is called a *tangent* (vector) to C at \mathbf{x} if there is a sequence $\{\mathbf{x}^k\}$ in C and a sequence $\{\alpha_k\}$ in \mathbb{R}_+ such that

$$\lim_{k \rightarrow \infty} (\mathbf{x}^k - \mathbf{x}) / \alpha_k = \mathbf{d}.$$

The set of tangent vectors at \mathbf{x} is denoted by $T_C(\mathbf{x})$.

$T_C(\mathbf{x})$ always contains the zero vector and it is a cone, meaning that it contains each positive multiple of its vectors. Consider now problem (13.9) and let C be the set of feasible solutions (those \mathbf{x} satisfying all the equality and inequality constraints).

Definition 13.5 (Linearized feasible directions). A *linearized feasible direction* at $\mathbf{x} \in C$ is a vector \mathbf{d} such that

$$\begin{aligned} \mathbf{d} \cdot \nabla h_i(\mathbf{x}) &= 0 \quad (i \leq m) \\ \mathbf{d} \cdot \nabla g_j(\mathbf{x}) &= 0 \quad (j \in A(\mathbf{x}^*)). \end{aligned}$$

Let $LF_C(\mathbf{x})$ be the set of all linearized feasible directions at \mathbf{x} .

So, if we move from \mathbf{x} along a linearized feasible direction with a suitably small step, then the new point is feasible if we only care about the *linearized* constraints at \mathbf{x} (the first order Taylor approximations) of each h_i and each g_j for active constraints at \mathbf{x} , i.e., those inequality constraints that hold with equality. With this notation we have the following lemma. The proof may be found in [11] and it involves the implicit function theorem from multivariate calculus [8].

Lemma 13.6. Let $\mathbf{x}^* \in C$. Then $T_C(\mathbf{x}^*) \subseteq LF_C(\mathbf{x}^*)$. If \mathbf{x}^* is a regular point, then $T_C(\mathbf{x}^*) = LF_C(\mathbf{x}^*)$.

The purpose of constraint qualifications is to assure that $T_C(\mathbf{x}^*) = LF_C(\mathbf{x}^*)$. This property is central for obtaining the necessary optimality conditions discussed above. An important example is when C is defined only by *linear constraints*, i.e., each h_i and c_j is a linear function. Then $T_C(\mathbf{x}^*) = LF_C(\mathbf{x}^*)$ holds for each $\mathbf{x} \in C$.

For a more thorough discussion of these matters, see e.g. [11, 1].

In the remaining part of this section we discuss some examples; the main tool is to establish the KKT conditions.

Example 13.7. Consider the one-variable problem: minimize $f(x)$ subject to $x \geq 0$, where $f : \mathbb{R} \rightarrow \mathbb{R}$ is a differentiable convex function. We here let $g_1(x) = -x$ and $m = 0$. The KKT conditions then become: there is a number μ such that $f'(x) - \mu = 0$, $\mu \geq 0$ and $\mu = 0$ if $x > 0$. This is one of the (rare) occasions where we can eliminate the Lagrangian variable μ via the equation $\mu = f'(x)$. So the optimality conditions are: $x \geq 0$ (feasibility), $f'(x) \geq 0$, and $f'(x) = 0$ if $x > 0$ (x is an interior point of the domain so the derivative must be zero), and if $x = 0$ we must have $f'(0) \geq 0$.

Example 13.8. More generally, consider the problem to minimize $f(\mathbf{x})$ subject to $\mathbf{x} \geq \mathbf{0}$, where $f : \mathbb{R}^n \rightarrow \mathbb{R}$. So here $C = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{x} \geq \mathbf{0}\}$ is the nonnegative orthant. We have that $g_i(x) = -x_i$, so that $\nabla g_i = -\mathbf{e}_i$. The KKT conditions say that $-\nabla f(\mathbf{x}^*)$ is a nonnegative combination of $-\mathbf{e}_i$ for i so that $x_i = 0$. In other words, $\nabla f(\mathbf{x}^*)$ is a nonnegative combination of \mathbf{e}_i for i so that $x_i = 0$. This means that

$$\begin{aligned} \partial f(\mathbf{x}^*)/\partial x_i &= 0 \quad \text{for all } i \leq n \text{ with } \mathbf{x}_i^* > 0, \text{ and} \\ \partial f(\mathbf{x}^*)/\partial x_i &\geq 0 \quad \text{for all } i \leq n \text{ with } \mathbf{x}_i^* = 0. \end{aligned}$$

If we interpret this for $n = 3$ we get the following cases:

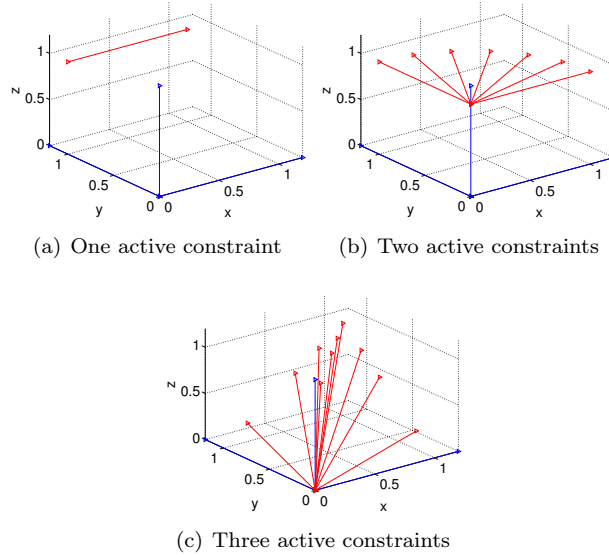


Figure 13.3: The different possibilities for ∇f in a minimum of f , under the constraints $\mathbf{x} \geq \mathbf{0}$.

- No active constraints: This means that $x, y, z > 0$. The KKT-conditions say that all partial derivatives are 0, so that $\nabla f(\mathbf{x}^*) = \mathbf{0}$. This is reasonable, since these points are internal points.
- One active constraint, such as $x = 0, y, z > 0$. The KKT-conditions say that $\partial f(\mathbf{x}^*)/\partial y = \partial f(\mathbf{x}^*)/\partial z = 0$, so that $\nabla f(\mathbf{x}^*)$ points in the positive direction of \mathbf{e}_1 , as shown in Figure 13.3(a).
- Two active constraints, such as $x = y = 0, z > 0$. The KKT-conditions say that $\partial f(\mathbf{x}^*)/\partial z = 0$, so that $\nabla f(\mathbf{x}^*)$ lies in the cone spanned by $\mathbf{e}_1, \mathbf{e}_2$, i.e. $\nabla f(\mathbf{x}^*)$ lies in the first quadrant of the xy -plane, as shown in Figure 13.3(b).
- Three active constraints: This means that $x = y = z = 0$. The KKT conditions say that $\nabla f(\mathbf{x}^*)$ is in the cone spanned by $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$, as shown in Figure 13.3(c).

In all cases $\nabla f(\mathbf{x}^*)$ points into a cone spanned by gradients corresponding to the active inequalities (in general, by a cone we mean the set of all linear combinations of a set of vectors, with positive coefficients). Note that for the third case above, we are used to finding minimum values from before: if we restrict f to values where $x = y = 0$, we have a one-dimensional problem where we want to minimize $g(z) = f(x, y, z)$, which is equivalent to finding z so that $g'(z) = \partial f(\mathbf{x}^*)/\partial z = 0$, as stated by the KKT-conditions.

Example 13.9. Consider a quadratic optimization problem with linear equality constraints

$$\begin{aligned} &\text{minimize} && (1/2) \mathbf{x}^T D \mathbf{x} - \mathbf{q}^T \mathbf{x} \\ &\text{subject to} && A \mathbf{x} = \mathbf{b} \end{aligned}$$

where D is positive semidefinite and $A \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$. This is a special case of (13.15) where $f(\mathbf{x}) = (1/2) \mathbf{x}^T D \mathbf{x} - \mathbf{q}^T \mathbf{x}$. Then $\nabla f(\mathbf{x}) = D \mathbf{x} - \mathbf{q}$ (see Exercise 9.8). Thus, the KKT conditions are: there is some $\boldsymbol{\lambda} \in \mathbb{R}^m$ such that $D \mathbf{x} - \mathbf{q} + A^T \boldsymbol{\lambda} = \mathbf{0}$. In addition, the vector \mathbf{x} is feasible so we have $A \mathbf{x} = \mathbf{b}$. Thus, solving the quadratic optimization problem amounts to solving the linear system of equations

$$D \mathbf{x} + A^T \boldsymbol{\lambda} = \mathbf{q}, \quad A \mathbf{x} = \mathbf{b}$$

which may be written as

$$\begin{bmatrix} D & A^T \\ A & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{q} \\ \mathbf{b} \end{bmatrix}. \quad (13.14)$$

Under the additional assumption that D is positive definite and A has full row rank, one can show that the coefficient matrix in (13.14) is invertible so this system has a unique solution $\mathbf{x}, \boldsymbol{\lambda}$. Thus, for this problem, we may write down an explicit solution (in terms of the inverse of the block matrix). Numerically, one finds \mathbf{x} (and the Lagrangian multiplier $\boldsymbol{\lambda}$) by solving the linear system (13.14) by e.g. Gaussian elimination or some faster (direct or iterative) method.

Example 13.10. Consider an extension of the previous example by allowing linear inequality constraints as well:

$$\begin{aligned} &\text{minimize} && (1/2) \mathbf{x}^T D \mathbf{x} - \mathbf{q}^T \mathbf{x} \\ &\text{subject to} && A \mathbf{x} = \mathbf{b} \\ &&& \mathbf{x} \geq \mathbf{0} \end{aligned}$$

Here D , A and \mathbf{b} are as above. Then $\nabla f(\mathbf{x}) = D \mathbf{x} - \mathbf{q}$ and $\nabla g_k(\mathbf{x}) = -\mathbf{e}_k$. Thus, the KKT conditions for this problem are: there are $\boldsymbol{\lambda} \in \mathbb{R}^m$ and $\boldsymbol{\mu} \in \mathbb{R}^n$ such that $D \mathbf{x} - \mathbf{q} + A^T \boldsymbol{\lambda} - \boldsymbol{\mu} = \mathbf{0}$, $\boldsymbol{\mu} \geq \mathbf{0}$ and $\mu_k = 0$ if $x_k > 0$ ($k \leq n$). We eliminate $\boldsymbol{\mu}$ from the first equation and obtain the equivalent condition: there is a $\boldsymbol{\lambda} \in \mathbb{R}^m$ such that $D \mathbf{x} + A^T \boldsymbol{\lambda} \geq \mathbf{q}$ and $(D \mathbf{x} + A^T \boldsymbol{\lambda} - \mathbf{q})_k \cdot x_k = 0$ ($k \leq n$). In addition, we have $A \mathbf{x} = \mathbf{b}$, $\mathbf{x} \geq \mathbf{0}$. This problem may be solved numerically, for instance, by a so-called active set method, see [9].

Example 13.11. Linear optimization is a problem of the form

$$\text{minimize } \mathbf{c}^T \mathbf{x} \quad \text{subject to } A\mathbf{x} = \mathbf{b}, \quad \mathbf{x} \geq \mathbf{0}$$

This is a special case of the convex programming problem (13.15) where $g_j(\mathbf{x}) = -x_j$ ($j \leq n$). Here $\nabla f(\mathbf{x}) = \mathbf{c}$ and $\nabla g_k(\mathbf{x}) = -\mathbf{e}_k$. Let \mathbf{x} be a feasible solution. The KKT conditions state that there are vectors $\boldsymbol{\lambda} \in \mathbb{R}^m$ and $\boldsymbol{\mu} \in \mathbb{R}^n$ such that $\mathbf{c} + A^T \boldsymbol{\lambda} - \boldsymbol{\mu} = \mathbf{0}$, $\boldsymbol{\mu} \geq \mathbf{0}$ and $\mu_k = 0$ if $x_k > 0$ ($k \leq n$). Here we eliminate $\boldsymbol{\mu}$ and obtain the equivalent set of KKT conditions: there is a vector $\boldsymbol{\lambda} \in \mathbb{R}^m$ such that $\mathbf{c} + A^T \boldsymbol{\lambda} \geq \mathbf{0}$, $(\mathbf{c} + A^T \boldsymbol{\lambda})_k \cdot x_k = 0$ ($k \leq n$). These conditions are the familiar optimality conditions in linear optimization theory. The vector $\boldsymbol{\lambda}$ is feasible in the so-called dual problem and complementary slack holds. We do not go into details on this here, but refer to the course INF-MAT3370 *Linear optimization* where these matters are treated in detail.

13.3 Convex optimization

A *convex optimization* problem is to minimize a convex function f over a convex set C in \mathbb{R}^n . These problems are especially attractive, both from a theoretic and algorithmic perspective.

First, let us consider some general results.

Theorem 13.12. Let $f : C \rightarrow \mathbb{R}$ be a convex function defined on a convex set $C \subseteq \mathbb{R}^n$.

1. Then every local minimum of f over C is also a global minimum.
2. If f is continuous and C is closed, then the set of local (and therefore global) minimum points of f over C is a closed convex set.
3. Assume, furthermore, that $f : C \rightarrow \mathbb{R}$ is differentiable and C is open. Let $\mathbf{x}^* \in C$. Then $\mathbf{x}^* \in C$ is a local (global) minimum if and only if $\nabla f(\mathbf{x}^*) = \mathbf{0}$.

Proof. 1.) The proof of property 1 is exactly as the proof of the first part of Theorem 12.4, except that we work with local and global minimum of f over C .

2.) Assume the set C^* of minimum points is nonempty and let $\alpha = \min_{\mathbf{x} \in C} f(\mathbf{x})$. Then $C^* = \{\mathbf{x} \in C : f(\mathbf{x}) \leq \alpha\}$ is a convex set, see Proposition 10.5. Moreover, this set is closed as f is continuous.

3.) This follows directly from Theorem 10.9. \square

Next, we consider a quite general convex optimization problem which is of the form (13.9):

$$\begin{array}{ll}
\text{minimize} & f(\mathbf{x}) \\
\text{subject to} & A\mathbf{x} = \mathbf{b} \\
& g_j(\mathbf{x}) \leq 0 \quad (j \leq r)
\end{array} \tag{13.15}$$

where all the functions f and g_j are differentiable convex functions, and $A \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$. Let C denote the feasible set of problem (13.15). Then C is a convex set, see Proposition 10.5. A special case of (13.15) is linear optimization.

An important concept in convex optimization is *duality*. To briefly explain this introduce again the Lagrangian function $L : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}_+^r \rightarrow \mathbb{R}$ given by

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T (A\mathbf{x} - \mathbf{b}) + \boldsymbol{\nu}^T \mathbf{G}(\mathbf{x}) \quad (\mathbf{x} \in \mathbb{R}^n, \boldsymbol{\lambda} \in \mathbb{R}^m, \boldsymbol{\nu} \in \mathbb{R}_+^r)$$

Remark: we use the variable name $\boldsymbol{\nu}$ here in stead of the $\boldsymbol{\mu}$ used before because of another parameter $\boldsymbol{\mu}$ to be used soon. Note that we require $\boldsymbol{\nu} \geq \mathbf{0}$.

Define the new function $g : \mathbb{R}^m \times \mathbb{R}_+^r \rightarrow \mathbb{R}$ by

$$g(\boldsymbol{\lambda}, \boldsymbol{\nu}) = \inf_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$$

Note that this infimum may sometimes be equal to $-\infty$ (meaning that the function $\mathbf{x} \rightarrow L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$ is unbounded below). The function g is the pointwise infimum of a family of affine functions in $(\boldsymbol{\lambda}, \boldsymbol{\mu})$, one function for each \mathbf{x} , and this implies that g is a concave function. We are interested in g due to the following fact, which is easy to prove. It is usually referred to as *weak duality*.

Lemma 13.13. Let \mathbf{x} be feasible in problem (13.15) and let $\boldsymbol{\lambda} \in \mathbb{R}^m, \boldsymbol{\nu} \in \mathbb{R}_+^r$ where $\boldsymbol{\nu} \geq \mathbf{0}$. Then

$$g(\boldsymbol{\lambda}, \boldsymbol{\nu}) \leq f(\mathbf{x}).$$

Proof. For $\boldsymbol{\lambda} \in \mathbb{R}^m, \boldsymbol{\nu} \in \mathbb{R}_+^r$ with $\boldsymbol{\nu} \geq \mathbf{0}$ and \mathbf{x} feasible in problem (13.15) we have

$$\begin{aligned}
g(\boldsymbol{\lambda}, \boldsymbol{\nu}) &\leq L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) \\
&= f(\mathbf{x}) + \boldsymbol{\lambda}^T (A\mathbf{x} - \mathbf{b}) + \boldsymbol{\nu}^T \mathbf{G}(\mathbf{x}) \\
&\leq f(\mathbf{x})
\end{aligned}$$

as $A\mathbf{x} = \mathbf{b}$, $\boldsymbol{\nu} \geq \mathbf{0}$ and $\mathbf{G}(\mathbf{x}) \leq \mathbf{0}$. □

Thus, $g(\boldsymbol{\lambda}, \boldsymbol{\nu})$ provides a lower bound on the optimal value in (13.15). It is natural to look for a best possible such lower bound and this is precisely the so-called *dual problem* which is

$$\begin{aligned}
& \text{maximize} && g(\boldsymbol{\lambda}, \boldsymbol{\nu}) \\
& \text{subject to} && \\
& && \boldsymbol{\nu} \geq \mathbf{0}.
\end{aligned} \tag{13.16}$$

Actually, in this dual problem, we may further restrict the attention to those $(\boldsymbol{\lambda}, \boldsymbol{\nu})$ for which $g(\boldsymbol{\lambda}, \boldsymbol{\nu})$ is finite.

The original problem (13.15) will be called the *primal problem*. It follows from Lemma 13.13 that

$$g^* \leq f^*$$

where f^* denotes the optimal value in the primal problem and g^* the optimal value in the dual problem. If $g^* < f^*$, we say that there is a *duality gap*. Note that the derivation above, and weak duality, holds for *arbitrary* functions f and g_j ($j \leq r$). The concavity of g also holds generally.

The dual problem is useful when the dual objective function g may be computed efficiently, either analytically or numerically. Duality provides a powerful method for proving that a solution is optimal or, possibly, near-optimal. If we have a feasible \mathbf{x} in (13.15) and we have found a dual solution $(\boldsymbol{\lambda}, \boldsymbol{\nu})$ with $\boldsymbol{\nu} \geq \mathbf{0}$ such that

$$f(\mathbf{x}) = g(\boldsymbol{\lambda}, \boldsymbol{\nu}) + \epsilon$$

for some ϵ (which then has to be nonnegative), then we can conclude that \mathbf{x} is “nearly optimal”, it is not possible to improve f by more than ϵ . Such a point \mathbf{x} is sometimes called *ϵ -optimal*, where the case $\epsilon = 0$ means optimal.

So, how good is this duality approach? For *convex problems* it is often perfect as the next theorem says. We omit most of the proof, see [5, 1, 14]). For nonconvex problems one should expect a duality gap. Recall that $\mathbf{G}'(\mathbf{x})$ denotes the Jacobi matrix of $\mathbf{G} = (g_1, g_2, \dots, g_r)$ at \mathbf{x} .

Theorem 13.14. Consider convex optimization problem (13.15) and assume this problem has a feasible point satisfying

$$g_j(\mathbf{x}') < 0 \quad (j \leq r).$$

Then $f^* = g^*$, so there is no duality gap. Moreover, \mathbf{x} is a (local and global) minimum in (13.15) if and only if there are $\boldsymbol{\lambda} \in \mathbb{R}^m$ and $\boldsymbol{\nu} \in \mathbb{R}^r$ with $\boldsymbol{\nu} \geq \mathbf{0}$ and

$$\nabla f(\mathbf{x}) + A^T \boldsymbol{\lambda} + \mathbf{G}'(\mathbf{x})^T \boldsymbol{\nu} = \mathbf{0}$$

and

$$\nu_j g_j(\mathbf{x}) = 0 \quad (j \leq r).$$

Proof. We only prove the second part (see the references above). So assume that $f^* = g^*$ and the infimum and supremum are attained in the primal and dual problems, respectively. Let \mathbf{x} be a feasible point in the primal problem.

Then \mathbf{x} is a minimum in the primal problem if and only if there are $\boldsymbol{\lambda} \in \mathbb{R}^m$ and $\boldsymbol{\nu} \in \mathbb{R}^r$ such that all the inequalities in the proof of Lemma 13.13 hold with equality. This means that $g(\boldsymbol{\lambda}, \boldsymbol{\nu}) = L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$ and $\boldsymbol{\nu}^T \mathbf{G}(\mathbf{x}) = 0$. But $L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$ is convex in \mathbf{x} so it is minimized by \mathbf{x} if and only if its gradient is the zero vector, i.e., $\nabla f(\mathbf{x}) + \boldsymbol{\lambda}^T A + \mathbf{G}'(\mathbf{x})^T \boldsymbol{\nu} = \mathbf{0}$. This leads to the desired characterization. \square

The assumption stated in the theorem, that $g_j(\mathbf{x}') < 0$ for each j , is called the *weak Slater condition*.

Finally, we mention a theorem on convex optimization which is used in several applications.

Theorem 13.15. Let $\mathbf{x}^* \in C$. Then \mathbf{x}^* is a (local and therefore global) minimum of f over C if and only if

$$\nabla f(\mathbf{x}^*)^T (\mathbf{x} - \mathbf{x}^*) \geq 0 \quad \text{for all } \mathbf{x} \in C. \quad (13.17)$$

Proof. Assume first that $\nabla f(\mathbf{x}^*)^T (\mathbf{x} - \mathbf{x}^*) < 0$ for some $\mathbf{x} \in C$. Consider the function $g(\epsilon) = f(\mathbf{x}^* + \epsilon(\mathbf{x} - \mathbf{x}^*))$ and apply the mean value theorem to this function. Thus, for every $\epsilon > 0$ there exists an $s \in [0, 1]$ with

$$f(\mathbf{x}^* + \epsilon(\mathbf{x} - \mathbf{x}^*)) = f(\mathbf{x}^*) + \epsilon \nabla f(\mathbf{x}^* + s\epsilon(\mathbf{x} - \mathbf{x}^*))^T (\mathbf{x} - \mathbf{x}^*).$$

Since $\nabla f(\mathbf{x}^*)^T (\mathbf{x} - \mathbf{x}^*) < 0$ and the gradient function is continuous (our standard assumption!) we have for sufficiently small $\epsilon > 0$ that $\nabla f(\mathbf{x}^* + s\epsilon(\mathbf{x} - \mathbf{x}^*))^T (\mathbf{x} - \mathbf{x}^*) < 0$. This implies that $f(\mathbf{x}^* + \epsilon(\mathbf{x} - \mathbf{x}^*)) < f(\mathbf{x}^*)$. But, as C is convex, the point $\mathbf{x}^* + \epsilon(\mathbf{x} - \mathbf{x}^*)$ also lies in C and so we conclude that \mathbf{x}^* is not a local minimum. This proves that (13.17) is necessary for \mathbf{x}^* to be a local minimum of f over C .

Next, assume that (13.17) holds. Using Theorem 10.9 we then get

$$f(\mathbf{x}) \geq f(\mathbf{x}^*) + \nabla f(\mathbf{x}^*)^T (\mathbf{x} - \mathbf{x}^*) \geq f(\mathbf{x}^*) \quad \text{for every } \mathbf{x} \in C$$

so \mathbf{x}^* is a (global) minimum. \square

Exercises for section 13.3

Ex. 1 — In the plane consider a rectangle R with sides of length x and y and with perimeter equal to α (so $2x + 2y = \alpha$). Determine x and y so that the area of R is largest possible.

Ex. 2 — Consider the optimization problem

$$\text{minimize } f(x_1, x_2) \text{ subject to } (x_1, x_2) \in C$$

where $C = \{(x_1, x_2) \in \mathbb{R}^2 : x_1, x_2 \geq 0, 4x_1 + x_2 \geq 8, 2x_1 + 3x_2 \leq 12\}$. Draw the feasible set C in the plane. Find the set of optimal solutions in each of the cases given below.

- a. $f(x_1, x_2) = 1$.
- b. $f(x_1, x_2) = x_1$.
- c. $f(x_1, x_2) = 3x_1 + x_2$.
- d. $f(x_1, x_2) = (x_1 - 1)^2 + (x_2 - 1)^2$.
- e. $f(x_1, x_2) = (x_1 - 10)^2 + (x_2 - 8)^2$.

Ex. 3 — Solve

$$\max\{x_1 x_2 \cdots x_n : \sum_{j=1}^n x_j = 1\}.$$

Ex. 4 — Let $S = \{\mathbf{x} \in \mathbb{R}^2 : \|\mathbf{x}\| = 1\}$ be the unit circle in the plane. Let $\mathbf{a} \in \mathbb{R}^2$ be a given point. Formulate the problem of finding a nearest point in S to \mathbf{a} as a nonlinear optimization problem. How can you solve this problem directly using a geometrical argument?

Ex. 5 — Let S be the unit circle as in the previous exercise. Let $\mathbf{a}_1, \mathbf{a}_2$ be two given points in the plane. Let $f(\mathbf{x}) = \sum_{i=1}^2 \|\mathbf{x} - \mathbf{a}_i\|^2$. Formulate this as an optimization problem and find its Lagrangian function L . Find the stationary points of L , and use this to solve the optimization problem.

Ex. 6 — Solve

$$\text{minimize } x_1 + x_2 \text{ subject to } x_1^2 + x_2^2 = 1.$$

using the Lagrangian, see Theorem 13.1. Next, solve the problem by eliminating x_2 (using the constraint).

Ex. 7 — Let $g(x_1, x_2) = 3x_1^2 + 10x_1x_2 + 3x_2^2 - 2$. Solve

$$\min\{\|(x_1, x_2)\| : g(x_1, x_2) = 0\}.$$

Ex. 8 — Same question as in previous exercise, but with $g(x_1, x_2) = 5x_1^2 - 4x_1x_2 + 4x_2^2 - 6$.

Ex. 9 — Let f be a two times differentiable function $f : \mathbb{R}^n \rightarrow \mathbb{R}$. Consider the optimization problem

$$\text{minimize } f(\mathbf{x}) \text{ subject to } x_1 + x_2 + \cdots + x_n = 1.$$

Characterize the stationary points (find the equation they satisfy).

Ex. 10 — Consider the previous exercise. Explain how to convert this into an unconstrained problem by eliminating x_n . Find an

Ex. 11 — Let A be a real symmetric $n \times n$ matrix. Consider the optimization problem

$$\max\{x^T A x : \|x\| = 1\}$$

Rewrite the constraint as $\|x\| = 1$ and show that an optimal solution of this problem must be an eigenvector of A . What can you say about the Lagrangian multiplier?

Ex. 12 — Solve

$$\min\{(1/2)(x_1^2 + x_2^2 + x_3^2) : x_1 + x_2 + x_3 \leq -6\}.$$

Hint: Use KKT and discuss depending on whether the constraint is active or not.

Ex. 13 — Solve

$$\min\{(x_1 - 3)^2 + (x_2 - 5)^2 + x_1 x_2 : 0 \leq x_1, x_2 \leq 1\}.$$

Ex. 14 — Solve

$$\min\{x_1 + x_2 : x_1^2 + x_2^2 \leq 2\}.$$

Ex. 15 — Use Theorem 13.15 to find optimality conditions for the convex optimization problem

$$\min\{f(x_1, x_2, \dots, x_n) : x_j \geq 0 \ (j \leq n), \sum_{j=1}^n x_j \leq 1\}$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is a differentiable convex function.

Chapter 14

Constrained optimization - methods

In this final chapter we present numerical methods for solving nonlinear optimization problems. This is a huge area, so we can here only give a small taste of it! The algorithms we present are known good methods.

14.1 Equality constraints

We here consider the nonlinear optimization problem with linear equality constraints

$$\begin{array}{ll} \text{minimize} & f(\mathbf{x}) \\ \text{subject to} & A\mathbf{x} = \mathbf{b} \end{array} \quad (14.1)$$

Newton's method may be applied to this problem. The method is very similar to the unconstrained case, but with two modifications. First, the initial point \mathbf{x}_0 must be chosen so that it is feasible, i.e., $A\mathbf{x}_0 = \mathbf{b}$. Next, the search direction \mathbf{d} must be such that the new iterate is feasible as well. This means that $A\mathbf{d} = \mathbf{0}$, so the search direction lies in the nullspace of A .

The second order Taylor approximation of f at an iterate \mathbf{x}_k is

$$T_f^1(\mathbf{x}_k; \mathbf{x}_k + \mathbf{h}) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T \mathbf{h} + (1/2) \mathbf{h}^T \nabla^2 f(\mathbf{x}_k) \mathbf{h}$$

and we want to minimize this w.r.t. \mathbf{h} subject to the constraint

$$A(\mathbf{x}_k + \mathbf{h}) = \mathbf{b}$$

This is a quadratic optimization problem in \mathbf{h} with a linear equality constraint ($A\mathbf{h} = \mathbf{0}$) as in Example 13.9. The KKT conditions for this problem are thus

$$\begin{bmatrix} \nabla^2 f(\mathbf{x}_k) & A^T \\ A & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{h} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} -\nabla f(\mathbf{x}_k) \\ \mathbf{0} \end{bmatrix}$$

where $\boldsymbol{\lambda}$ is the Lagrange multiplier. The Newton step is only defined when the coefficient matrix in the KKT problem is invertible. In that case, the problem has a unique solution $(\mathbf{h}, \boldsymbol{\lambda})$ and we define $\mathbf{d}_{Nt} = \mathbf{h}$ and call this the *Newton step*.

Newton's method for solving (14.1) may now be described as follows. Again $\epsilon > 0$ is a small stopping criterion.

Newton's method for linear equality constrained optimization:

1. Choose an initial point \mathbf{x}_0 satisfying $A\mathbf{x}_0 = \mathbf{b}$ and let $\mathbf{x} = \mathbf{x}_0$.
2. repeat
 - (i) Compute the Newton step \mathbf{d}_{Nt} and $\eta := \mathbf{d}_{Nt}^T \nabla^2 f(\mathbf{x}) \mathbf{d}_{Nt}$.
 - (ii) If $\eta^2/2 < \epsilon$: stop.
 - (iii) Use backtracking line search to find step size α
 - (iv) Update $\mathbf{x} := \mathbf{x} + \alpha \mathbf{d}_{Nt}$

This leads to an algorithm for Newton's method for linear equality constrained optimization which is very similar to the function `newtonbacktrack` from Exercise 12.2.10. We do not state a formal convergence theorem for this method, but it behaves very much like Newton's method for unconstrained optimization. Actually, it can be seen that the method just described corresponds to eliminating variables based on the equations $A\mathbf{x} = \mathbf{b}$ and using the unconstrained Newton method for the resulting (smaller) problem. So as soon as the solution is "sufficiently near" an optimal solution, the convergence rate is quadratic, so extremely few iterations are needed in this final stage.

14.2 Inequality constraints

We here briefly discuss an algorithm for inequality constrained nonlinear optimization problems. The presentation is mainly based on [2, 11]. We restrict the attention to convex optimization problems, but many of the ideas are used for nonconvex problems as well.

The method we present is an *interior-point method*, more precisely, an *interior-point barrier method*. This is an iterative method which produces a sequence of points lying in the relative interior of the feasible set. The barrier idea is to approximate the problem by a simpler one in which constraints are replaced by a penalty term. The purpose of this penalty term is to give large objective

function values to points near the (relative) boundary of the feasible set, which effectively becomes a barrier against leaving the feasible set.

Consider again the convex optimization problem

$$\begin{aligned} & \text{minimize} && f(\mathbf{x}) \\ & \text{subject to} && \\ & && A\mathbf{x} = \mathbf{b} \\ & && g_j(\mathbf{x}) \leq 0 \quad (j \leq r) \end{aligned} \tag{14.2}$$

where A is an $m \times n$ matrix and $\mathbf{b} \in \mathbb{R}^m$. The feasible set here is $F = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{b}, g_j(\mathbf{x}) \leq 0 \ (j \leq r)\}$. We assume that the weak Slater condition holds, and therefore by Theorem 13.14 the KKT conditions for problem (14.2) are

$$\begin{aligned} A\mathbf{x} &= \mathbf{b}, \quad g_j(\mathbf{x}) \leq 0 \quad (j \leq r) \\ \boldsymbol{\nu} &\geq \mathbf{0}, \quad \nabla f(\mathbf{x}) + A^T \boldsymbol{\lambda} + \mathbf{G}'(\mathbf{x})^T \boldsymbol{\nu} = \mathbf{0} \\ \nu_j g_j(\mathbf{x}) &= 0 \quad (j \leq r). \end{aligned} \tag{14.3}$$

So, \mathbf{x} is a minimum in (14.2) if and only if there are $\boldsymbol{\lambda} \in \mathbb{R}^m$ and $\boldsymbol{\nu} \in \mathbb{R}^r$ such that (14.3) holds.

Let us state an algorithm for Newton's method for linear equality constrained optimization with inequality constraints. Before we do this there is one final problem we need to address: The α we get from backtracking line search may be so that $\mathbf{x} + \alpha \mathbf{d}_{Nt}$ do not satisfy the inequality constraints (in the exercises you will be asked to verify that this is the case for a certain function). The problem comes from that the iterates $\mathbf{x}_k + \beta^m \mathbf{s} \mathbf{d}_k$ from Armijo's rule do not necessarily satisfy the inequality constraints. However, we can choose m large enough so that all succeeding iterates satisfy these constraints. We can reimplement the function `armijorule` to address this as follows:

```
function alpha=armijoruleg1g2(f,df,x,d,g1,g2)
    beta=0.2; s=0.5; sigma=10^(-3);
    m=0;
    while (g1(x+beta^m*s*d)>0 || g2(x+beta^m*s*d)>0)
        m=m+1;
    end
    while (f(x)-f(x+beta^m*s*d) < -sigma *beta^m*s *(df(x))'*d)
        m=m+1;
    end
    alpha = beta^m*s;
```

Here `g1` and `g2` are function handles which represent the inequality constraints, and we have added a first loop, which secures that m is so large that the inequality constraints are satisfied. The rest of the code is as in the function `armijorule`. After this we can also modify the function `newtonbacktrack` from Exercise 12.2.10 to a function `newtonbacktrackg1g2` in the obvious way, so that the inequality constraints are passed to `armijoruleg1g2`:


```

function [x,numit]=newtonbacktrackg1g2LEC(f,df,d2f,A,b,x0,g1,g2)
    epsilon=10^(-3);
    x=x0;
    maxit=100;
    for numit=1:maxit
        matr=[d2f(x) A'; A zeros(size(A,1))];
        vect=[-df(x); zeros(size(A,1),1)];
        solvedvals=matr\vect;
        d=solvedvals(1:size(A,2));
        eta=d'*d2f(x)*d;
        if eta^2/2<epsilon
            break;
        end
        alpha=armijoruleg1g2(f,df,x,d,g1,g2);
        x=x+alpha*d;
    end

```

Both these function work in all cases where there are exactly two inequality constraints.

The interior-point barrier method is based on an approximation of problem (14.2) by the *barrier problem*

$$\begin{aligned}
 & \text{minimize} && f(\mathbf{x}) + \mu\phi(\mathbf{x}) \\
 & \text{subject to} && \\
 & && A\mathbf{x} = \mathbf{b}
 \end{aligned} \tag{14.4}$$

where

$$\phi(\mathbf{x}) = -\sum_{j=1}^r \ln(-g_j(\mathbf{x}))$$

and $\mu > 0$ is a parameter (in \mathbb{R}). The function ϕ is called the (*logarithmic barrier function*) and its domain is the relative interior of the feasible set

$$F^\circ = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{b}, g_j(\mathbf{x}) < 0 \ (j \leq r)\}.$$

The same set F° is the feasible set of the barrier problem. The key properties of the barrier function are:

- ϕ is concave, i.e. $-\phi$ is a convex function. This may be shown from the definition using that g_j is convex and the fact that the logarithm function is concave and increasing.
- If $\{\mathbf{x}_k\}$ is a sequence in F° such that $g_j(\mathbf{x}_k) \rightarrow 0$ for some $j \leq r$, then $\phi(\mathbf{x}_k) \rightarrow \infty$. This is the barrier property.

- ϕ is twice differentiable and

$$\nabla\phi(\mathbf{x}) = \sum_{j=1}^r \frac{1}{(-g_j(\mathbf{x}))} \nabla g_j(\mathbf{x}) \quad (14.5)$$

and

$$\nabla^2\phi(\mathbf{x}) = \sum_{j=1}^r \frac{1}{g_j^2(\mathbf{x})} \nabla g_j(\mathbf{x}) \nabla g_j(\mathbf{x})^T + \sum_{j=1}^r \frac{1}{(-g_j(\mathbf{x}))} \nabla^2 g_j(\mathbf{x}) \quad (14.6)$$

The idea here is that for points \mathbf{x} near the boundary of F the value of $\phi(\mathbf{x})$ is very large. So, an iterative method which moves around in the interior F° of F will typically avoid points near the boundary as the logarithmic penalty term makes the function value $f(\mathbf{x}) + \mu\phi(\mathbf{x})$ very large.

The interior point method consists in solving the barrier problem, using Newton's method, for a sequence $\{\mu_k\}$ of (positive) barrier parameters; these are called the *outer iterations*. The solution \mathbf{x}_k found for $\mu = \mu_k$ is used as the starting point in Newton's method in the next outer iteration where $\mu = \mu_{k+1}$. The sequence $\{\mu_k\}$ is chosen such that $\mu_k \rightarrow 0$. When μ is very small, the barrier function approximates the "ideal" penalty function $\eta(x)$ which is zero in F and $-\infty$ when one of the inequalities $g_j(\mathbf{x}) \leq 0$ is violated.

A natural question is why one bothers to solve the barrier problems for more than one single μ , typically a very small value. The reason is that it would be hard to find a good starting point for Newton's method in that case; the Hessian matrix of $\mu\phi$ is typically ill-conditioned for small μ .

Assume now that the barrier problem has a *unique* optimal solution $\mathbf{x}(\mu)$; this is true under reasonable assumptions that we shall return to. The point $\mathbf{x}(\mu)$ is called a *central point*. Assume also that Newton's method may be applied to solve the barrier problem. The set of points $\mathbf{x}(\mu)$ for $\mu > 0$ is called the *central path*; it is a path (or curve) as we know it from multivariate calculus. In order to investigate the central path we prefer to work with the equivalent problem¹ to (14.4) obtained by multiplying the objection function by $1/\mu$, so

$$\begin{aligned} &\text{minimize} && (1/\mu)f(\mathbf{x}) + \phi(\mathbf{x}) \\ &\text{subject to} && \\ &&& A\mathbf{x} = \mathbf{b}. \end{aligned} \quad (14.7)$$

A central point $\mathbf{x}(\mu)$ is characterized by

$$\begin{aligned} A\mathbf{x}(\mu) &= \mathbf{b} \\ g_j(\mathbf{x}(\mu)) &< 0 \quad (j \leq r) \end{aligned}$$

and the existence of $\boldsymbol{\lambda} \in \mathbb{R}^m$ (the Lagrange multiplier vector) such that

$$(1/\mu)\nabla f(\mathbf{x}(\mu)) + \nabla\phi(\mathbf{x}(\mu)) + A^T\boldsymbol{\lambda} = \mathbf{0}$$

¹Equivalent here means the same minimum points.

i.e.,

$$(1/\mu)\nabla f(\mathbf{x}(\mu)) + \sum_{j=1}^r \frac{1}{(-g_j(\mathbf{x}))} \nabla g_j(\mathbf{x}) + A^T \boldsymbol{\lambda} = \mathbf{0}. \quad (14.8)$$

A fundamental question is: how far from being optimal is the central point $\mathbf{x}(\mu)$? We now show that duality provides a very elegant way of answering this question.

Theorem 14.1. For each $\mu > 0$ the central point $\mathbf{x}(\mu)$ satisfies

$$f^* \leq f(\mathbf{x}(\mu)) \leq f^* + r\mu.$$

Proof. Define $\boldsymbol{\nu}(\mu) = (\nu_1(\mu), \dots, \nu_r(\mu)) \in \mathbb{R}^r$ and $\boldsymbol{\lambda}(\mu) \in \mathbb{R}^m$ by

$$\begin{aligned} \nu_j(\mu) &= -\mu/g_j(\mathbf{x}(\mu)), & (j \leq r); \\ \boldsymbol{\lambda}(\mu) &= \mu\boldsymbol{\lambda}. \end{aligned} \quad (14.9)$$

We want to show that the pair $(\boldsymbol{\lambda}(\mu), \boldsymbol{\nu}(\mu))$ is a feasible solution in the dual problem to (14.2), see Section 13.3. So there are two properties to verify, that $\boldsymbol{\nu}(\mu)$ is nonnegative and that $\mathbf{x}(\mu)$ minimizes the Lagrangian function for the given $(\boldsymbol{\lambda}(\mu), \boldsymbol{\nu}(\mu))$. The first property is immediate: as $g_j(\mathbf{x}(\mu)) < 0$ and $\mu > 0$, we get $\nu_j(\mu) = -\mu/g_j(\mathbf{x}(\mu)) > 0$ for each j . Concerning the second property, note first that the Lagrangian function $L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T(A\mathbf{x} - \mathbf{b}) + \boldsymbol{\nu}^T \mathbf{G}(\mathbf{x})$ is convex in \mathbf{x} for given $\boldsymbol{\lambda}$ and $\mu \geq 0$. Thus, \mathbf{x} minimizes this function if and only if $\nabla_{\mathbf{x}} L = \mathbf{0}$. Now,

$$\nabla_{\mathbf{x}} L(\mathbf{x}(\mu), \boldsymbol{\lambda}(\mu), \boldsymbol{\nu}(\mu)) = \nabla f(\mathbf{x}(\mu)) + A^T \boldsymbol{\lambda}(\mu) + \sum_j \nu_j(\mu) \nabla g_j(\mathbf{x}(\mu)) = \mathbf{0}$$

by (14.8) and the definition of the dual variables (14.9). This shows that $(\boldsymbol{\lambda}(\mu), \boldsymbol{\nu}(\mu))$ is a feasible solution to the dual problem.

By weak duality Lemma 13.13 we therefore obtain

$$\begin{aligned} f^* &\geq g(\boldsymbol{\lambda}(\mu), \boldsymbol{\nu}(\mu)) \\ &= L(\mathbf{x}(\mu), \boldsymbol{\lambda}(\mu), \boldsymbol{\nu}(\mu)) \\ &= f(\mathbf{x}(\mu)) + \boldsymbol{\lambda}(\mu)^T(A\mathbf{x}(\mu) - \mathbf{b}) + \sum_{j=1}^r \nu_j(\mu) g_j(\mathbf{x}(\mu)) \\ &= f(\mathbf{x}(\mu)) - r\mu \end{aligned}$$

which proves the result. \square

This theorem is very useful and shows why letting $\mu \rightarrow 0$ (more accurately $\mu \rightarrow 0^+$) is a good idea.

Corollary 14.2. The central path has the following property

$$\lim_{\mu \rightarrow 0} f(\mathbf{x}(\mu)) = f^*.$$

In particular, if f is continuous and $\lim_{\mu \rightarrow 0} \mathbf{x}(\mu) = \mathbf{x}^*$ for some \mathbf{x}^* , then \mathbf{x}^* is a global minimum in (14.2).

Proof. This follows from Theorem 14.1 by letting $\mu \rightarrow 0$. The second part follows from

$$f(\mathbf{x}^*) = f(\lim_{\mu \rightarrow 0} \mathbf{x}(\mu)) = \lim_{\mu \rightarrow 0} f(\mathbf{x}(\mu)) = f^*$$

by the first part and the continuity of f ; moreover \mathbf{x}^* must be a feasible point by elementary topology. \square

After these considerations we may now present the interior-point barrier method. It uses a tolerance $\epsilon > 0$ in its stopping criterion.

Interior-point barrier method:

1. Choose an initial point $\mathbf{x} = \mathbf{x}_0$ in F° , $\mu = \mu^0$ and $\alpha < 1$.
2. while $r\mu > \epsilon$ do
 - (i) (Centering step) Using initial point \mathbf{x} find the solution $\mathbf{x}(\mu)$ of (14.4)
 - (ii) (Update) $\mathbf{x} := \mathbf{x}(\mu)$
 - (iii) (Decrease μ) $\mu := \alpha\mu$.

This leads to the following algorithm for the internal point barrier method for the case of equality constraints, and 2 inequality constraints:

```
function xopt=IPBopt(f,g1,g2,df,dg1,dg2,d2f,d2g1,d2g2,A,b,x0)
    xopt=x0;
    mu=1;
    alpha=0.1;
    r=2;
    epsilon=10^(-3);
    numitouter=0;
    while (r*mu>epsilon)
        [xopt,numit]=newtonbacktrackg1g2LEC(...
            @(x)(f(x)-mu*log(-g1(x))-mu*log(-g2(x))),...
            @(x)(df(x) - mu*dg1(x)/g1(x) - mu*dg2(x)/g2(x)),...
            @(x)(d2f(x) + mu*dg1(x)*dg1(x)'/g1(x)^2) ...
                + mu*dg2(x)*dg2(x)'/g2(x)^2 - mu*d2g1(x)/g1(x)...
                - mu*d2g2(x)/g2(x) ),A,b,xopt,g1,g2);
        mu=alpha*mu;
```

```

numitouter=numitouter+1;
fprintf('Iteration %i:',numitouter);
fprintf(' (%f,%f)\n',xopt,f(xopt));
end

```

Note that we here have inserted the expressions from Equation 14.5 and Equation 14.6 for the gradient and the Hesse matrix of the barrier function. The input are f , g_1 , g_2 , their gradients and their Hesse matrices, the matrix A , the vector \mathbf{b} , and an initial feasible point \mathbf{x}_0 . The function calls `newtonbacktrackg1g2LEC`, and returns the optimal solution \mathbf{x}^* . It also gives some information on the values of f during the iterations. The iterations used in Newton's method is called the *inner iterations*. There are different implementation details here that we do not discuss very much. A typical value on α is 0.1. The choice of the initial μ^0 can be difficult, if it is chosen too large, one may experience many outer iterations. Another issue is how accurately one solves (14.4). It may be sufficient to find a near-optimal solution here as this saves inner iterations. For this reason the method is also called a *path-following method*; it follows in the neighborhood of the central path.

Finally, it should be mentioned that there exists a variant of the interior-point barrier method which permits an infeasible starting point. For more details on this and various implementation issues one may consult [2] or [11].

Example 14.3. Consider the function $f(x) = x^2 + 1$, $2 \leq x \leq 4$. Minimizing f can be considered as the problem of finding a minimum subject to the constraints $g_1(x) = 2 - x \leq 0$, and $g_2(x) = x - 4 \leq 0$. The barrier problem is to minimize the function

$$f(x) + \mu\phi(x) = x^2 + 1 - \mu \ln(x - 2) - \mu \ln(4 - x).$$

Some of these are drawn in Figure 14.1, where we clearly can see the effect of decreasing μ in the barrier function: The function converges to f pointwise, except at the boundaries. It is easy to see that $x = 2$ is the minimum of f under the given constraints, and that $f(2) = 5$ is the minimum value. There are no equality constraints in this case, so that we can use the barrier method with Newton's method for unconstrained optimization, as this was implemented in Exercise 12.2.10. We need, however, to make sure also here that the iterates from Armijo's rule satisfy the inequality constraints. In fact, in the exercises you will be asked to verify that, for the function f considered here, some of the iterates from Armijo's rule do not satisfy the constraints.

It is straightforward to implement a function `newtonbacktrackg1g2` which implements Newton's method for two inequality constraints and no equality constraints (this can follow the implementation of the function `newtonbacktrack` from Exercise 12.2.10, and use the function `armijoruleg1g2`, just as the function `newtonbacktrackg1g2LEC`). This leads to the following algorithm for the interior point barrier method for the case of no equality constraints, but 2 inequality constraints:

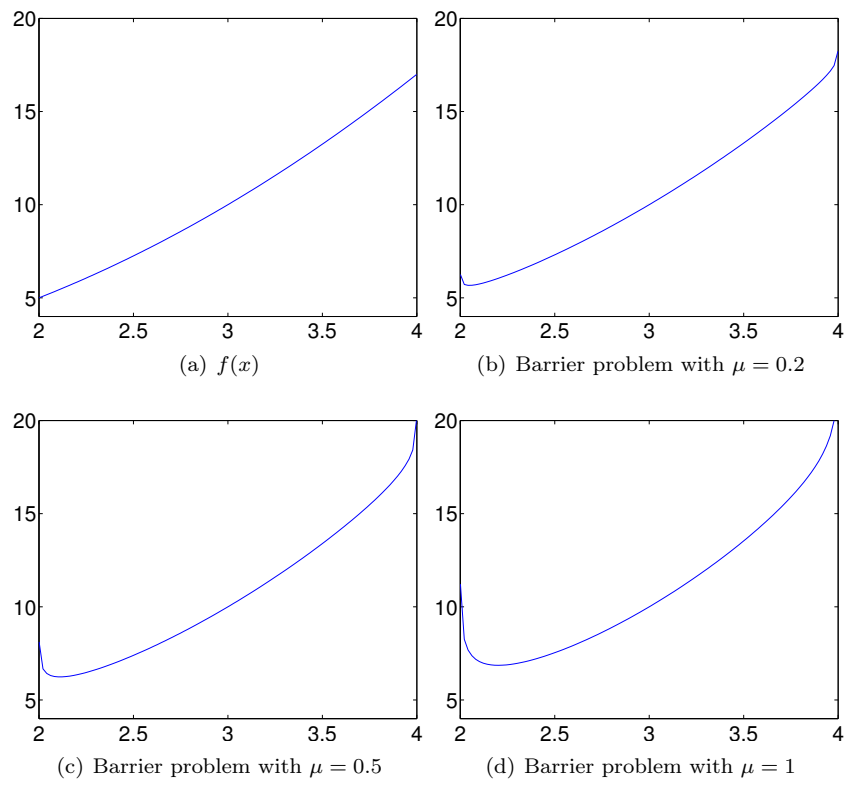


Figure 14.1: The function from Example 14.3 and some of its barrier functions.

```

function xopt=IPBopt2(f,g1,g2,df,dg1,dg2,d2f,d2g1,d2g2,x0)
    xopt=x0;
    mu=1; alpha=0.1; r=2; epsilon=10^(-3);
    numitouter=0;
    while (r*mu>epsilon)
        [xopt,numit]=newtonbacktrackg1g2(...
            @(x)(f(x)-mu*log(-g1(x))-mu*log(-g2(x))),...
            @(x)(df(x) - mu*dg1(x)/g1(x) - mu*dg2(x)/g2(x)),...
            @(x)(d2f(x) + mu*dg1(x)*dg1(x)'/(g1(x)^2) ...
                + mu*dg2(x)*dg2(x)'/(g2(x)^2) ...
                - mu*d2g1(x)/g1(x) - mu*d2g2(x)/g2(x) ),xopt,g1,g2);
        mu=alpha*mu;
        numitouter=numitouter+1;
        fprintf('Iteration %i:',numitouter);
        fprintf(' (%f,%f)\n',xopt,f(xopt));
    end

```

Note that this function also prints a summary for each of the outer iterations, so that we can see the progress in the barrier method. We can now find the minimum of f with the following code, where we have substituted with Matlab functions for f , g_i , their gradients, and their Hesse matrices.

```

IPBopt2(@(x)(x.^2+1),@(x)(2-x),@(x)(x-4),...
        @(x)(2*x),@(x)(-1),@(x)(1),...
        @(x)(2),@(x)(0),@(x)(0),3)

```

Running this code gives a good approximation to the minimum $x = 2$ after 4 outer iterations.

Exercises for Section 14.2

Ex. 1 — Consider problem (14.1) in Section 14.1. Verify that the KKT conditions for this problem are as stated there.

Ex. 2 — Define the function $f(x, y) = x + y$. We will attempt to minimize f under the constraints $y - x = 1$, and $x, y \geq 0$

- Find A , \mathbf{b} , and functions g_1, g_2 so that the problem takes the same form as in Equation (14.2).
- Draw the contours of the barrier function $f(x, y) + \mu\phi(x, y)$ for $\mu = 0.1, 0.2, 0.5, 1$, where $\phi(x, y) = -\ln(-g_1(x, y)) - \ln(-g_2(x, y))$.
- Solve the barrier problem analytically using the Lagrange method.
- It is straightforward to find the minimum of f under the mentioned constraints. State a simple argument for finding this minimum.
- State the KKT conditions for finding the minimum, and solve these.

- f. Show that the central path converges to the same solution which you found in d. and e..

Ex. 3 — Use the function `IPBopt` to verify the solution you found in Exercise 2. Initially you must compute a feasible starting point x_0 .

Ex. 4 — State the KKT conditions for finding the minimum for the constrained problem of Example 14.3, and solve these. Verify that you get the same solution as in Example 14.3.

Ex. 5 — In the function `IPBopt2`, replace the call to the function `newtonbacktrackg1g2` with a call to the function `newtonbacktrack`, with the obvious modification to the parameters. Verify that the code does not return the expected minimum in this case.

Ex. 6 — Consider the function $f(x) = (x-3)^2$, with the same constraints $2 \leq x \leq 4$ as in Example 14.3. Verify in this case that the function `IPBopt2` returns the correct minimum regardless of whether you call `newtonbacktrackg1g2` or `newtonbacktrack`. This shows that, at least in some cases where the minimum is an interior point, the iterates from Newton's method satisfy the inequality constraints as well.

Nomenclature

$\check{\mathbf{x}}$	Symmetric extension of a vector
$\hat{\mathbf{x}}$	DFT of the vector \mathbf{x}
$\lambda_{S,n}$	Frequency response of a filter
λ_S	Frequency response of a filter
\oplus	Direct sum
\otimes	Tensor product
$\mathbf{x}^{(e)}$	Vector of even samples
$\mathbf{x}^{(o)}$	Vector of odd samples
ϕ_m	Wavelet basis, before transform
E_d	Filter which delays with d samples
F_N	$N \times N$ -DCT matrix
F_N	$N \times N$ -Fourier matrix
$O(f(\mathbf{x}))$	Order of a function
$O(N)$	Order of an algorithm
S^f	Matrix with the columns reversed
$V_{N,T}$	N 'th order Fourier space
$W_m^{(0,1)}$	Resolution m Complementary wavelet space, LH
$W_m^{(1,0)}$	Resolution m Complementary wavelet space, HL
$W_m^{(1,1)}$	Resolution m Complementary wavelet space, HH
\mathcal{C}_m	Wavelet basis, after transform, reordered
$\mathcal{D}_N = \{\mathbf{d}_0, \mathbf{d}_1, \dots, \mathbf{d}_{N-1}\}$	N -point DCT basis for \mathbb{R}^N

$\mathcal{D}_{N,T}$ Order N Fourier basis for $V_{N,T}$

$\mathcal{E}_N = \{\mathbf{e}_0, \mathbf{e}_1, \dots, \mathbf{e}_{N-1}\}$ Standard basis for \mathbb{R}^N

$\mathcal{F}_{N,T}$ Order N complex Fourier basis for $V_{N,T}$

Mathematics index

- AD conversion, 17
- affine function, 263
- algebra, 78
- analysis, 37
 - equations, 37
- angular frequency, 76
- Armijo rule, 281

- band-limited
 - function, 66
- bandpass filter, 84
- barrier method, 306
- barrier problem, 308
- bit rate, 18
- block, 211
- block diagonal matrices, 140
- block matrix, 110

- central path, 309
- central point, 309
- chain rule, 259
- change of coordinates
 - in a wavelet basis, 150
 - in tensor product, 221
- Complex Fourier coefficients, 46
- concave, 262
- condition number, 282
- continuous signal, 9
- contraction, 269
- convex combination, 263
- convex function (multivariate), 262
- convex optimization, 299
- convex set, 261
- convolution
 - sequences, 80
 - vectors, 80
- coordinate matrix, 221

- DCT basis, 99
- DCT coefficients, 100
- DCT matrix, 100
- detail space, 135
- diagonally dominant, 265

- digital
 - sound, 9, 17
- digital filter, 72
- Direct sum
 - canonical basis, 144
 - linear transformations, 152
 - matrices, 150
- direct sum
 - vector spaces, 136
- Dirichlet conditions, 38
- Discrete Cosine transform, 100
- Discrete Wavelet Transform, 144
- dual problem, 300
- duality, 300
- duality gap, 301

- error-resilient, 211

- FFT factorization, 111
- filter coefficients, 71
- fixed point, 269
- Fourier analysis, 33
- Fourier coefficients, 36, 57
- Fourier domain, 37
- Fourier matrix, 57
- Fourier series, 35
- Fourier space, 35
- Fourier transform
 - discrete, 57
- frequency domain, 37
- frequency response, 72

- gradient, 256
- gradient method, 280
- gradient related, 282

- Haar wavelet, 146
- Hessian matrix, 256
- highpass filter, 84

- ideal highpass filter, 84
- ideal lowpass filter, 84
- IDFT, 58
- impulse response, 74

- interior point method, 306
- interpolating polynomial, 66
- interpolation formula, 68
 - ideal
 - periodic functions, 68
- Inverse Discrete Wavelet Transform, 144
- Jacobi matrix, 257
- Jensen 's inequality, 263
- Karush-Kuhn-Tucker conditions, 293
- KKT conditions, 293
- Lagrangian function, 289
- least square error, 35
- length, 79
- linear convergence, 259
- linear optimization, 255
- linear phase
 - digital filters, 89
- linearized feasible direction, 296
- Lipschitz, 269
- logarithmic barrier function, 308
- lowpass filter, 84
- LTI filters, 89
- maximum, 249
- maximum likelihood, 253
- minimum, 249
- mother wavelets, 138
- MRA-matrix, 174
- multiresolution analysis, 151
- multiresolution model, 128
- Newton's method, 271, 281, 283, 306
- nonlinear optimization problem, 288
- nonlinear equations, 268
- objective function, 249
- optimal control, 254
- optimality conditions, 277
- Order N complex Fourier basis for $V_{N,T}$, 45
- Order N Fourier basis for $V_{N,T}$, 37
- Order of a function, 113
- Order of an algorithm, 113
- Parallel computing
 - with the DCT, 119
 - with the DWT, 211
 - with the FFT, 114
 - with the revised FFT, 123
- penalty term, 292
- perfect reconstruction system, 91
- polyhedron, 262
- positive definite, 256
- positive semidefinite, 256
- primal problem, 301
- pure digital tones, 56
- quasi-Newton method, 273
- reconstruction system, 91
- regular point, 289
- samples, 17
- sampling, 17
 - frequency, 17
 - period, 17
 - rate, 17
- scaling function, 132, 151
- separable extension, 226
- sound channel, 20
- square wave, 16
- stationary point, 277
- steepest descent method, 281
- strictly convex, 278
- subband
 - HH, 229
 - HL, 229
 - LH, 229
 - LL, 230
- superlinear convergence, 259
- symmetric extension, 96
- synthesis, 37
 - equation, 37
 - vectors, 57
- tangent vector, 295
- tensor product, 212
 - of function spaces, 226
 - of functions, 226
 - of matrices, 214

- of vectors, 214
- tile, 210
- time domain, 37
- time-invariant, 88
- Toeplitz matrix, 70
 - circulant, 70
- triangle wave, 16
- vector frequency response, 76
- Wavelet basis, after transform, re-ordered, 143
- Wavelet basis, before transform, 143
- wavelet coefficients, 144
- weak duality, 300
- weak Slater condition, 302

Index for MATLAB commands

audioplayer, 19

conv, 29, 80

dct, 119

double, 194

fft, 112

idct, 119

ifft, 112

imageview, 195

imread, 194

imwrite, 194

play, 19

playblocking, 19

rand, 23

uint8, 194

wavread, 20

wavwrite, 20

Bibliography

- [1] D.P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1999.
- [2] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.
- [3] G. Dahl. A note on diagonally dominant matrices. *Linear Algebra and its Appl.*, 317(1-3):217–224, 2000.
- [4] G. Dahl. *An introduction to convexity*. Report, University of Oslo, 2010.
- [5] J. B. Hiriart-Urruty and C. Lemaréchal. *Convex analysis and minimization algorithms I*. Springer, 1993.
- [6] C. T. Kelley. *Iterative Methods for Linear and Nonlinear Equations*. SIAM, 1995.
- [7] D. C. Lay. *linear algebra and its applications (4th edition)*. Addison Wesley, 2011.
- [8] T. Lindstrøm and K. Hveberg. *Flervariabel analyse med lineær algebra*. Pearson, 2011.
- [9] D.G. Luenberger. *Linear and nonlinear programming*. Addison-Wesley, 1984.
- [10] T. Lyche. *Numerical linear algebra*. Report, University of Oslo, 2010.
- [11] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer, 2006.
- [12] R.T. Rockafellar. *Convex Analysis*. Princeton University Press, 1970.
- [13] A. Ruszczyński. *Nonlinear optimization*. Princeton University Press, 2006.
- [14] R. Webster. *Convexity*. Oxford University Press, Oxford, 1994.