Chapter 8

Digital 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 in a computer. The purpose of this chapter is to give a brief introduction to digital sound representation and processing.

We start by a short discussion of what sound is, which leads us to the conclusion that sound can be conveniently modelled by functions of a real variable in section 8.1. From mathematics it is known that almost any function can be approximated arbitrarily well by a combination of sines and cosines, and we discuss what this means when it is translated to the context of sound. We then go on and discuss digital sound, and simple operations on digital sound in section 8.2. FInally, we consider compression of sound in sections 8.4 and 8.5.

8.1 Sound

What we perceive as sound corresponds to the physical phenomenon of slight variations in air pressure near our ears. Larger variations mean louder sounds,

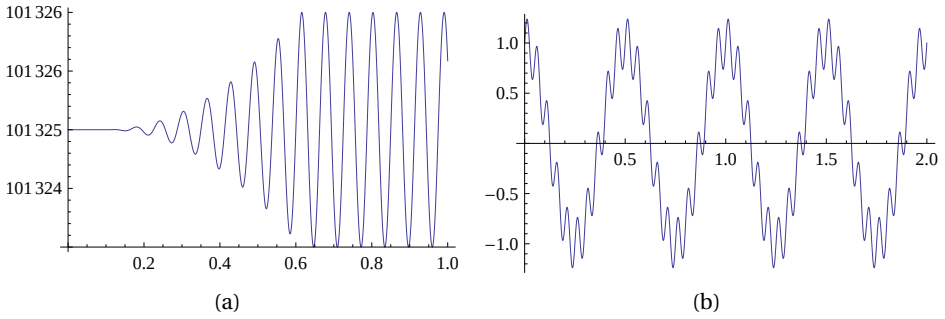


Figure 8.1. Two examples of audio signals.

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. In this section we briefly discuss the basic properties of sound, first the significance of the size of the variations, and then the frequency of the variations. We also consider the important fact that any sound may be considered to be built from very simple basis sounds.

Before we turn to the details, we should be clear about the use of the word *signal* which is often encountered in literature on sound and confuses many.

Observation 8.1. *A sound can be represented by a mathematical function. When a function represents a sound it is often referred to as a* signal.

8.1.1 Loudness: Sound pressure and decibels

An example of a simple sound is shown in figure 8.1a. We observe that the initial air pressure has the value 101 325, and then the pressure starts to vary more and more until it oscillates regularly between the values 101 323 and 101 326. 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 8.2 (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 8.2 explains the values on the vertical axis in figure 8.1a: 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 0.002–2 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 8.1a so that the values vary between -1 and 1. Figure 8.1b shows another sound with a slow, cos-like, variation in air pressure, roughly between -1 and 1. Imposed on this are some smaller and faster variations. This combination of several kinds of vibrations in air pressure is typical for general sounds.

The size of the variations in air pressure 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. The following fact box summarises the previous discussion of what a sound is, and introduces the logarithmic decibel scale.

Fact 8.3 (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.

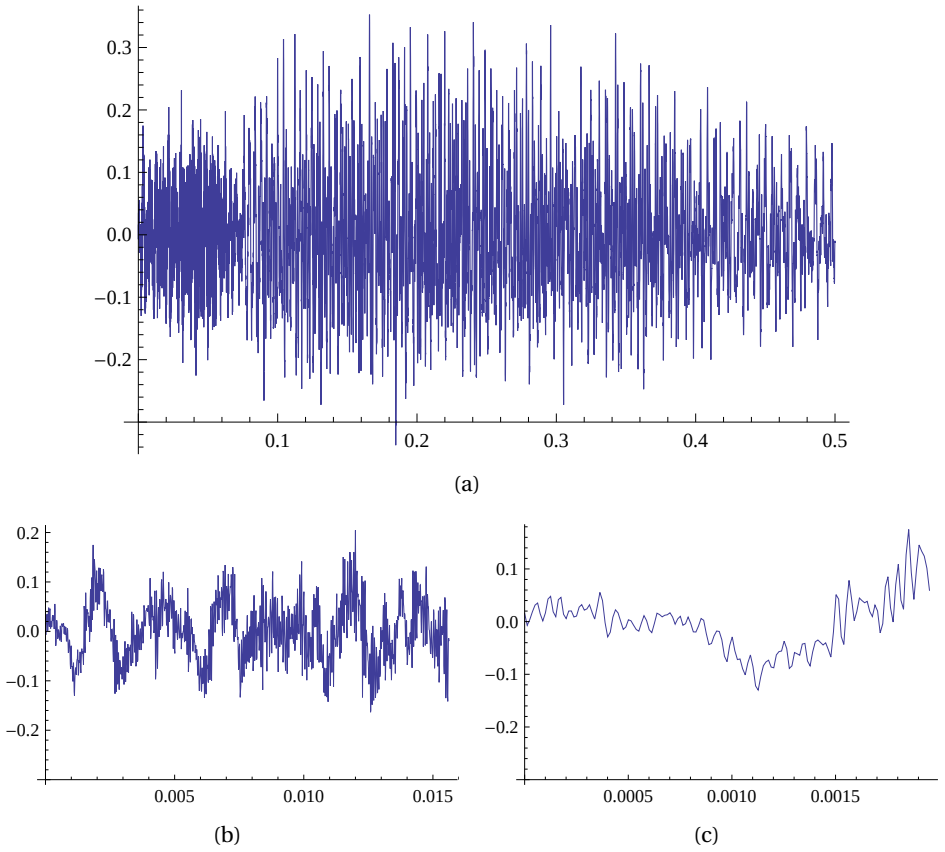


Figure 8.2. Variations in air pressure during parts of a song. Figure (a) shows 0.5 seconds of the song, figure (b) shows just the first 0.015 seconds, and figure (c) shows the first 0.002 seconds.

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 8.1 are synthetic in that they were constructed from mathematical formulas. The sounds in figure 8.2 show the variation in air pressure for a real sound. In (a) there are so many oscillations that it is impossible to see the details, but if we zoom in as in figure (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 figure 8.2c. 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.

8.1.2 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 8.4. *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 8.5. *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 8.6 (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 8.3 illustrates observation 8.6. The function in figure (a) is the plain $\sin t$ which covers one period 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 figure (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.

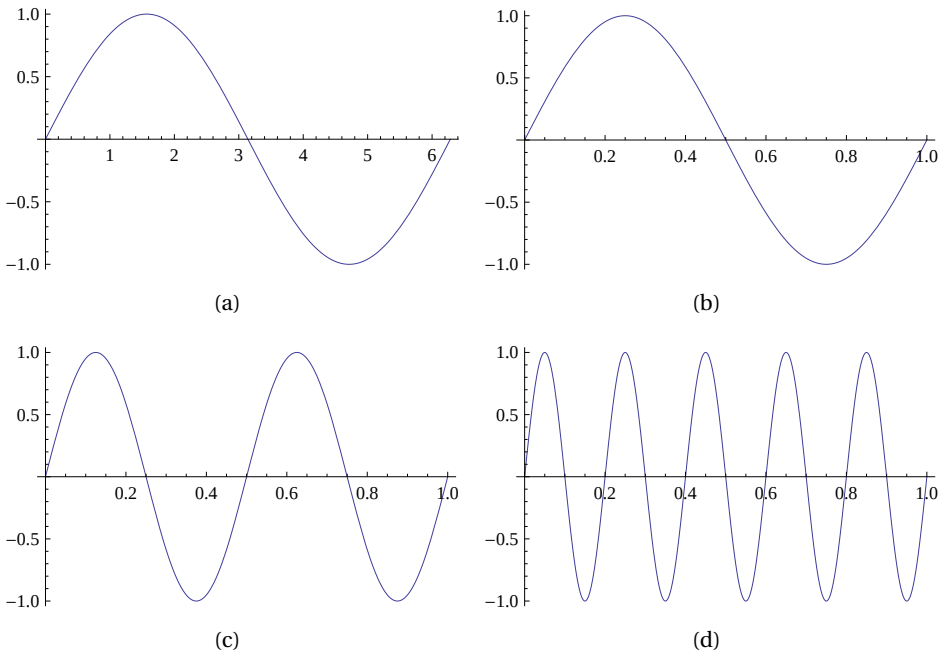


Figure 8.3. Versions of sin with different frequencies. The function in (a) is $\sin t$, the one in (b) is $\sin 2\pi t$, the one in (c) is $\sin 2\pi 2t$, and the one in (d) is $\sin 2\pi 5t$.

Definition 8.7. *The function* $\sin 2\pi\nu t$ *represents a pure tone with frequency* ν*. Frequency is measured in Hz (Herz) which is the same as* s^{-1}*.*

With appropriate software it is easy to generate a sound from a mathematical function; we can 'play' a 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 many other ways in which a function can oscillate regularly. The function in figure 8.1b for example, definitely oscillates 2 times every second, but it does not have frequency 2 Hz since it is not a pure sin function. Likewise, the two functions in figure 8.4 also oscillate twice every second, but are very different from a smooth, trigonometric function. If we play a function like the one in figure (a), but with 440 periods in a second, we hear a sound with the same pitch as $\sin 2\pi 440t$, but it is definitely not pleasant. The sharp corners translate into a rather shrieking, piercing sound. The function in figure (b) leads to a smoother sound than the one in (a), but not as smooth as a pure sin sound.

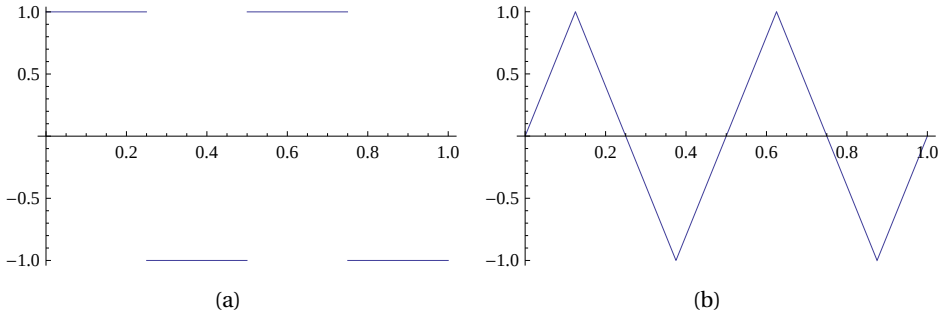


Figure 8.4. Two functions with regular oscillations, but which are not simple, trigonometric functions.

8.1.3 Any function is a sum of sin and cos

A very common tool in mathematics is to approximate general functions by combinations of more standard functions. Perhaps the most well-known example is Taylor series where functions are approximated by combinations of polynomials. In the area of sound it is of more interest to approximate with combinations of trigonometric functions — this is referred to as *Fourier analysis*. The following is an informal version of a very famous theorem.

Theorem 8.8 (Fourier series)**.** *Any reasonable function f can be approximated arbitrarily well on the interval* $[0,1]$ *by a combination*

$$f(t) \approx a_0 + \sum_{k=1}^{N} (a_k \cos 2\pi k t + b_k \sin 2\pi k t), \tag{8.1}$$

by choosing the integer N sufficiently large. The coefficients $\{a_k\}_{k=0}^{N}$ and $\{b_k\}_{k=1}^{N}$ are given by the formulas

$$a_k = \int_0^1 f(t)\cos(2\pi k t)\,dt, \qquad b_k = \int_0^1 f(t)\sin(2\pi k t)\,dt.$$

The series on the right in (8.1) *is called a Fourier series approximation of f.*

An illustration of the theorem is shown in figure 8.5 where a cubic polynomial is approximated by a Fourier series with $N = 9$. Note that 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 8.5b where its periodicity is clearly visible.

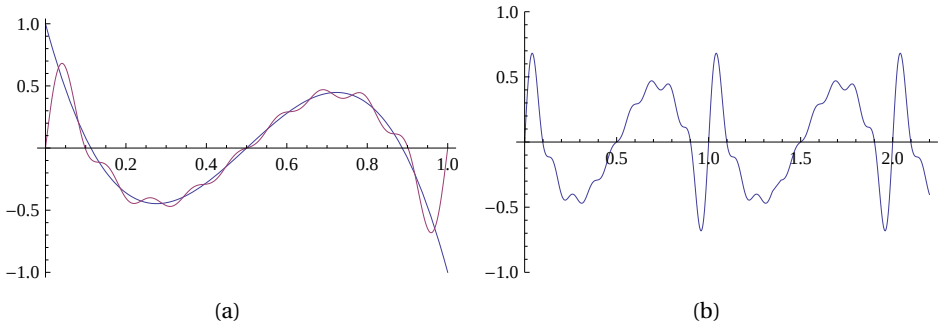


Figure 8.5. Trigonometric approximation of a cubic polynomial on the interval $[0,1]$. In (a) both functions are shown while in (b) the approximation is plotted on the interval $[0,2.2]$.

Since any sound may be considered to be a function, theorem 8.8 can be translated to a statement about sound. We recognise both trigonometric functions on the right in (8.1) as sounds with pure frequency k. The theorem therefore says that any sound may be approximated arbitrarily well by pure sounds with frequencies 0, 1, 2, ..., N, as long as we choose N sufficiently large.

Observation 8.9 (Decomposition of sound into pure tones)**.** *Any sound f is a sum of pure tones with integer frequencies. The amount of each frequency required to form f is the frequency content of f.*

Observation 8.9 makes it possible to explain more precisely what it means that we only perceive sounds with a frequency in the range 20–20 000.

Fact 8.10. *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.*

The most basic consequence of observation 8.9 is that gives us an understanding of how any sound can be built from the simple building blocks of sin and cos. But it is also the basis for many operations on sounds. As an example, consider the function in figure 8.6 (a). Even though this function oscillates 5 times regularly between 1 and -1, the discontinuities mean that it is far from the simple $\sin 2\pi 5t$ which corresponds to a pure tone of frequency 5. If we compute the Fourier coefficients, we find that all the a_k are zero since the function is antisymmetric. The first 100 of the b_k coefficients are shown in figure (c). We note that only $\{b_{10j-5}\}_{j=1}^{10}$ are nonzero, and these decrease in magnitude. Note

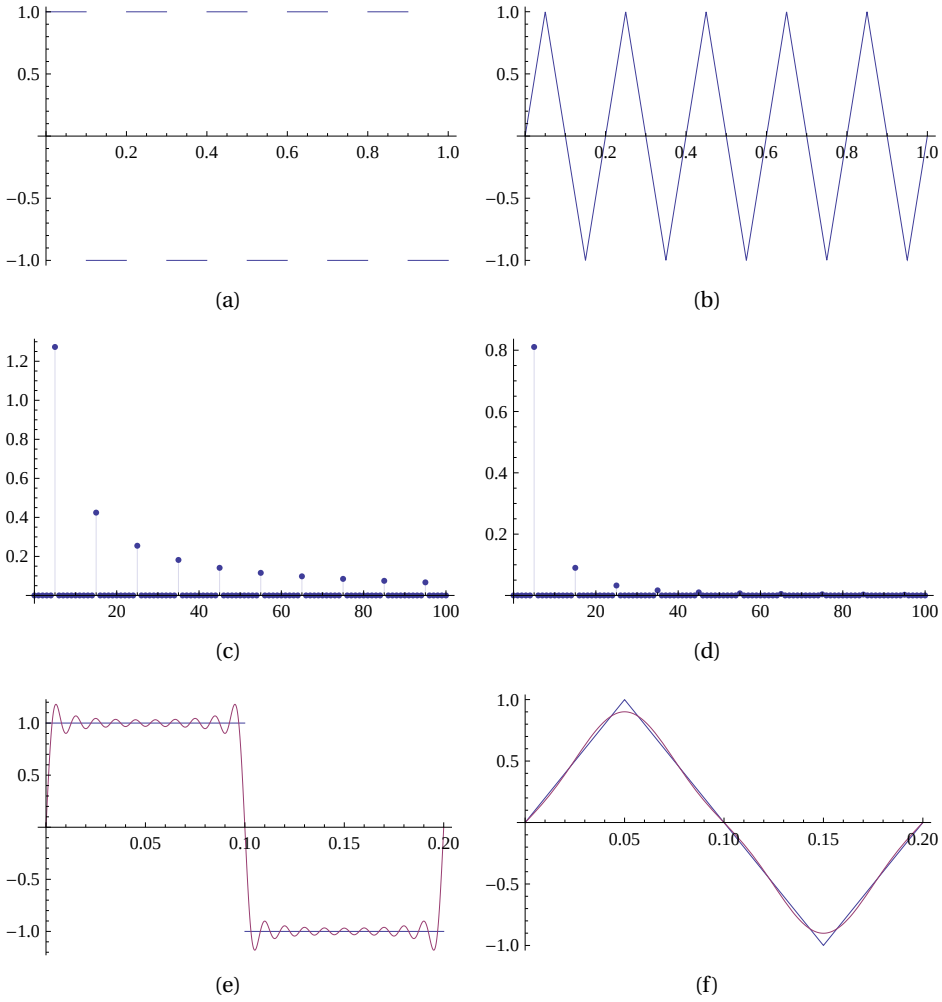


Figure 8.6. Approximations to two periodic functions with Fourier series. Since both functions are antisymmetric, the cos part in (8.1) is zero in both cases (all the a_k are zero). Figure (c) shows $\{a_k\}_{k=0}^{100}$ when f is the function in figure (a), and the plot in (e) shows the resulting approximation (8.1) with $N = 100$. The plots in figures (b), (d), and (e) are similar, except that the approximation in figure (f) corresponds to $N = 20$.

that the dominant coefficient is b_5, which tells us how much there is of the pure tone $\sin 2\pi 5t$ in the square wave in (a). This is not surprising since the square wave oscillates 5 times in a second, but the additional nonzero coefficients pollute the pure sound. As we include more and more of these coefficients, we gradually approach the square wave in (a). Figure (e) shows the corresponding approximation of one period of the square wave.

Figures 8.6 (b), (d), and (f) show the analogous information for a triangular wave. The function in figure (a) is continuous and therefore the trigonometric functions in (8.1) converge much faster. This can be seen from the size of the coefficients in figure (d), and from the plot of the approximation in figure (f). (Here we have only included two nonzero terms. With more terms, the triangular wave and the approximation become virtually indistinguishable.)

From figure 8.6 we can also see how we can use the Fourier coefficients to analyse or improve the sound. Noise in a sound often correspond to the presence of some high frequencies with large coefficients, and by removing these, we remove the noise. For example, in figure (b), we could set all the coefficients except the first one to zero. This would change the unpleasant square wave to the pure tone $\sin 2\pi 5t$ with the same number of oscillations per second. Another common operation is to dampen the treble of a sound. This can be done quite easily by reducing the size of the coefficients corresponding to high frequencies. Similarly, the bass can be adjusted by changing the coefficients corresponding to the lower frequencies.

8.2 Digital sound

In the previous section we considered some basic properties of sound, but it was all in terms of functions defined for all times in some interval. On computers and various kinds of media players the sound is usually digital, and in this section we are going to see what this means.

8.2.1 Sampling

Digital sound is very simple: The air pressure of a sound is measured a fixed number of times per second, and the measurements are stored as numbers in a file.

Definition 8.11 (Digital sound)**.** *A digital sound consists of an array $\boldsymbol{a}$ of numbers,* the samples*, that correspond to measurements of the air pressure of a sound, recorded at a fixed rate of s,* the sample rate*, measurements per second. If the sound is in stereo there will be two arrays $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$, one for each channel. Measuring the sound is also referred to as* sampling *the sound, or analog to digital (AD) conversion.*

There are many different digital sound formats. A couple of them are described in the following two examples.

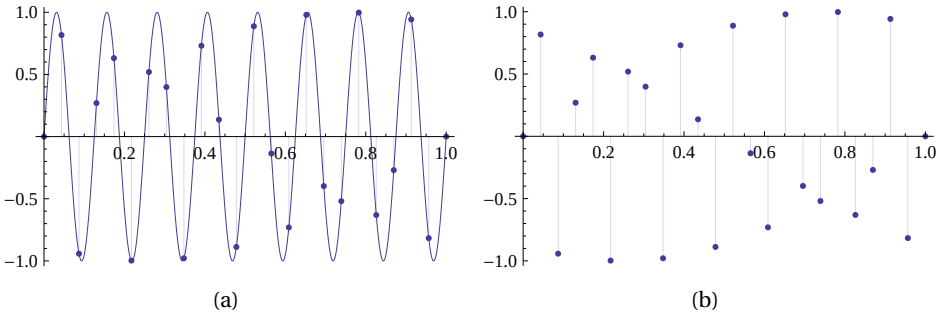


Figure 8.7. An example of sampling. Figure (a) shows how the samples are picked from underlying continuous time function. Figure (b) shows what the samples look like on their own.

Fact 8.12 (CD-format)**.** *The digital sound on a CD has sample rate 44 100, and each measurement is stored as a 16 bit integer.*

Fact 8.13 (GSM-telephone)**.** *The digital sound in GSM mobile telephony has sample rate 8 000, and each measurement is stored as a 13 bit number in a floating-point like format.*

There are many other digital sound formats in use, with sample rates as high as 192 000 and above, using 24 bits and more to store each number.

8.2.2 Limitations of digital audio: The sampling theorem

An example of sampling is illustrated in figure 8.7. When we see the samples on their own in figure (b) it is clear that some information is lost in the sampling process. An important question is therefore how densely we must sample a function in order to not lose too much information.

The difficult functions to sample are those that oscillate quickly, and the challenge is to make sure there are no important features between the samples. By zooming in on a function, we can reduce the extreme situation to something simple. This is illustrated in Figure 8.8. If we consider one period of $\sin 2\pi t$, we see from figure (a) that we need at least two sample points, 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 $[0,1]$.

Suppose now that we have a sound (i.e., a function) whose Fourier series contains terms with frequency at most equal to ν. This means that the function

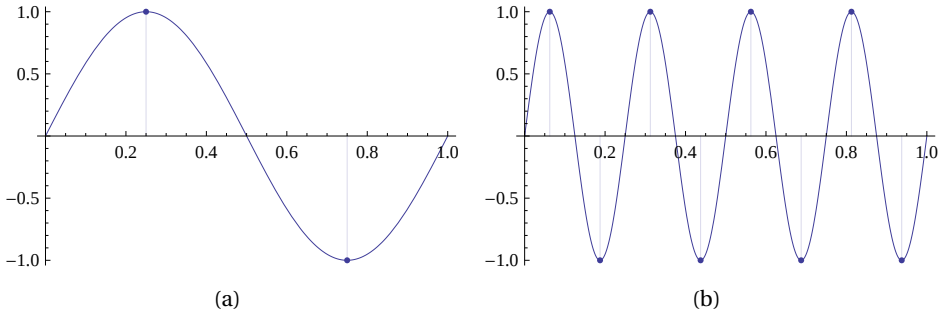


Figure 8.8. Sampling the function $\sin 2\pi t$ with two points, and the function $\sin 2\pi 4t$ with eight points.

in the series that varies most quickly is $\sin 2\pi\nu t$ which requires 2ν sample point per second. This informal observation is the content of an important theorem. We emphasise that the simple argument above is no proof of this theorem; it just shows that it is reasonable.

Theorem 8.14 (Shannon-Nyquist sampling theorem)**.** *A sound that includes frequencies up to ν Hz must be sampled at least 2ν times per second if no information is to be lost.*

The sampling theorem partly explains why the sampling rate on a CD is 44 100. Since the human ear can perceive frequencies up to about 20 000 Hz, the sampling rate must be at least 40 000 to ensure that the highest frequencies are accounted for. The actual sampling rate of 44 100 is well above this limit and ensures that there is some room to smoothly taper off the high frequencies from 20 000 Hz.

8.2.3 Reconstructing the original signal

Before we consider some simple operations on digital sound, we need to discuss a basic challenge: Sound which is going to be played back through an audio system must be defined for continuous time. In other words, we must fill in all the values of the air pressure between two sample points. There is obviously no unique way to do this since there are infinitely many paths for a graph to follow between to given points.

Fact 8.15 (Reconstruction of digital audio)**.** *Before a digital sound can be played through an audio system, the gaps between the sample points must*

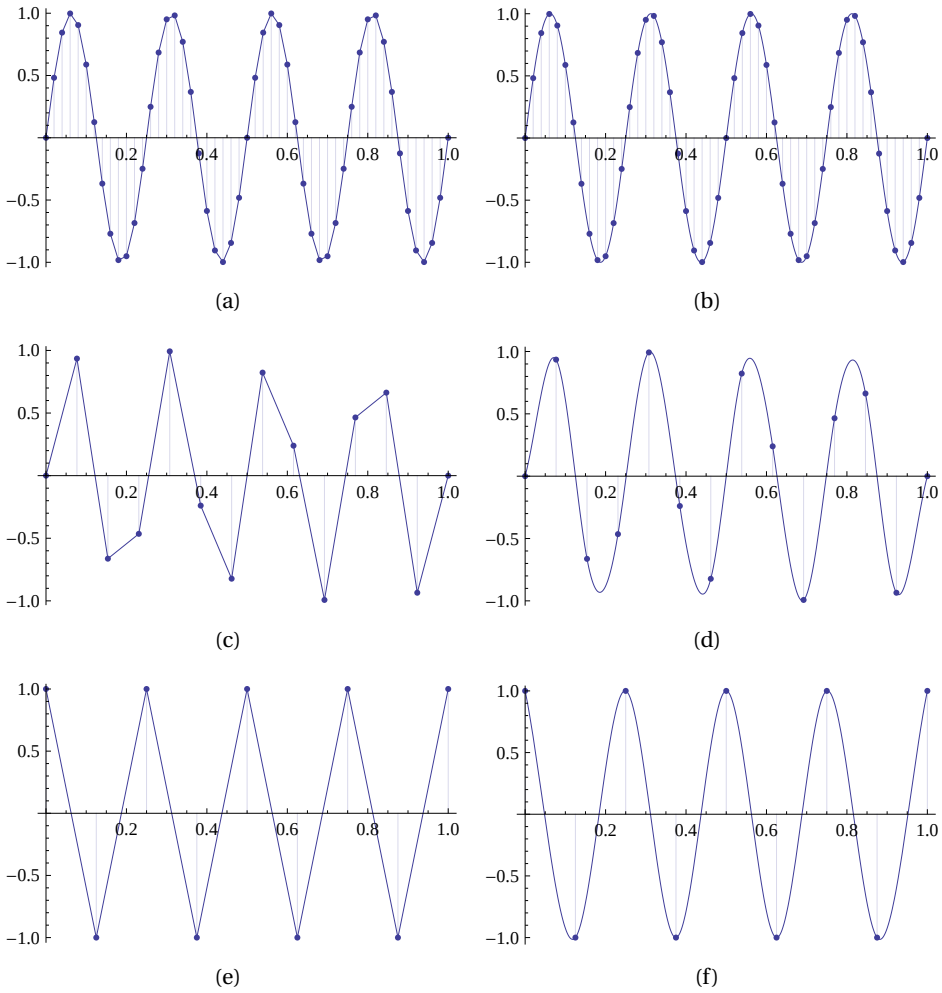


Figure 8.9. Reconstruction of sampled data.

be filled by some mathematical function. This process is referred to as digital to analog (DA) conversion.

Figure 8.9 illustrates two ways to reconstruct an analog audio signal from a digital one. In the top four figures, the points have been sampled from the function $\sin 2\pi 4t$, while in the lower two figures the samples are taken from $\cos 2\pi 4t$. In the first column, neighbouring sample points have been connected by straight lines which results in a piecewise linear function that passes through

(interpolates) the sample points. This works very well if the sample points are close together relative to the frequency of the oscillations, as in figure 8.9a. When the samples are further apart, as in (c) and (e), the discontinuities in the derivative become visible, and we know that this may be heard as noise in the reconstructed signal.

In the second column, the gap between two sample points has been filled with a cubic polynomial, and neighbouring cubic polynomials have been joined smoothly together so that the total function is continuous and has continuous first and second derivative. We see that this works much better and produces a smooth result that is very similar to the original trigonometric signal.

Figure 8.9 illustrates the general principle: If the sampling rate is high, quite simple reconstruction techniques will be sufficient, while if the sampling rate is low, more sophisticated methods for reconstruction will be necessary.

8.3 Simple operations on digital sound

So far we have discussed what digital sound is, the limitations in sampling, and how the information missing in sampled information may be reconstructed. It is now time to see how digital sound can be processed and manipulated.

Recall that a digital sound is just an array of sample values $\boldsymbol{a} = (a_i)_{i=0}^{N}$ together with the sample rate 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.

Playing a sound. Simple operations and computations with sound can be done in any programming environment, but in order to play the sound, it is necessary to use an environment that includes a command like `play(a, s)` (the ocmmand may of course have some other name; it is the functionality that is important). This will simply play the array of samples $\boldsymbol{a}$ using the sample rate s. If no `play`-function is available, 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.

The `play`-function is just a software interface to the sound card in your computer. It basically sends the array of sample values and the 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 8.16. *The basic command in a programming environment that handles sound is a command*

```
play(a, s)
```

which takes as input an array of sample values $\boldsymbol{a}$ *and a sample rate* s*, and plays the corresponding sound through the computer's loudspeakers.*

Changing the sample rate. We can easily play back a sound with a different sample rate than the standard one. If we have a sound $(\boldsymbol{a}, s)$ and we play it with the command `play(a, 2s)`, 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 as in the command `play(a, s/2)`. Then the length of the sound is doubled and all frequencies are halved. This results in low pitch, roaring voices.

Fact 8.17. *A digital sound* $(\boldsymbol{a}, s)$ *can be played back with a double or half sample rate with the commands*

```
play(a, 2s)
play(a, s/2)
```

Playing the sound backwards. At times a popular game as 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 8.18. *Let* $\boldsymbol{a} = \{a_i\}_{i=0}^{N}$ *be the samples of a digital sound. Then the samples* $\boldsymbol{b} = \{b_i\}_{i=0}^{N}$ *of the reverse sound are given by*

$$b_i = a_{N-i}, \quad \text{for } i = 0, 1, \ldots N.$$

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 8.19. *Let $\boldsymbol{a}$ be the samples of a digital sound, normalised so that each sample is a real number in the interval $[-1,1]$. A new sound $\boldsymbol{b}$ with noise added can be obtained by adding a random number to each sample,*

$$b_i = a_i + c\,\texttt{random()}$$

where `random()` *is a function that gives a random number in the interval $[-1,1]$, and c is a constant (usually smaller than 1) that dampens the noise.*

This will produce a general hissing noise similar to the noise you hear on the radio when the reception is bad. The factor c is important, if it is too large the noise will simply drown the signal $\boldsymbol{b}$.

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 8.20. *Let $(\boldsymbol{a}, s)$ be a digital sound. Then the sound $\boldsymbol{b}$ with samples given by*

$$b_i = \begin{cases} a_i, & \text{for } i = 0, 1, \ldots, d-1; \\ a_i + c\,a_{i-d}, & \text{for } i = d, d+1, \ldots, N; \end{cases}$$

will include a echo of the original sound. Here $d = \texttt{round}(ms)$ is the integer closest to ms, and c is a constant which is usually smaller than 1.

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, so there we just copy a_i to b_i.

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 $\boldsymbol{b} = (b_i)_{i=0}^N$ we can compute it as

$$b_i = \begin{cases} a_i, & \text{for } i = 0; \\ (a_{i-1} + a_i + a_{i+1})/3, & \text{for } 0 < i < N; \\ a_i, & \text{for } i = N. \end{cases}$$

This kind of operation is often referred to as *filtering* the sound, and the sequence $\{1/3, 1/3, 1/3\}$ is referred to as a *filter*.

It is reasonable to let the middle sample a_i count more than the neighbours in the average, so an alternative is to compute the average as

$$b_i = \begin{cases} a_i, & \text{for } i = 0; \\ (a_{i-1} + 2a_i + a_{i+1})/4, & \text{for } 0 < i < N; \\ a_i, & \text{for } i = N. \end{cases} \tag{8.2}$$

We can also take averages of more numbers. We note that the coefficients used in (8.2) are taken from row 2 in Pascal's triangle. If we pick coefficients from row 4 instead, the computations become

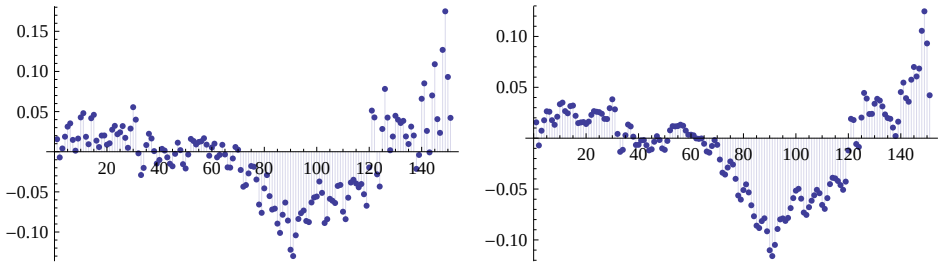
$$b_i = \begin{cases} a_i, & \text{for } i = 0, 1; \\ (a_{i-2} + 4a_{i-1} + 6a_i + 4a_{i+1} + a_{i+2})/16, & \text{for } 1 < i < N-1; \\ a_i, & \text{for } i = N-1, N. \end{cases} \tag{8.3}$$

We have not developed the tools needed to analyse the quality of filters, but it turns out that picking coefficients from a row in Pascal's triangle works very well, and better the longer the filter is.

Observation 8.21. *Let $\boldsymbol{a}$ be the samples of a digital sound, and let $\{c_i\}_{i=0}^{2k}$ be the numbers in row $2k$ of Pascal's triangle. Then the sound with samples $\boldsymbol{b}$ given by*

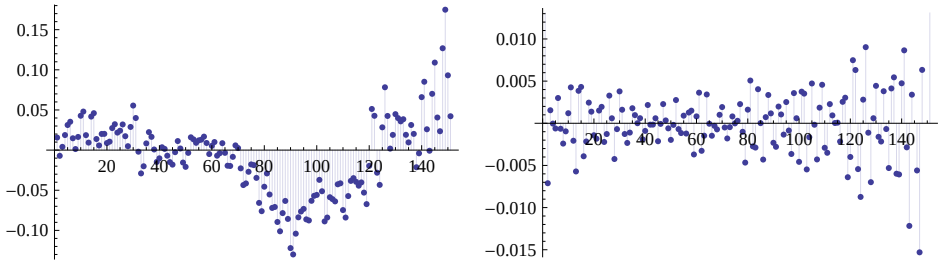
$$b_i = \begin{cases} a_i, & \text{for } i = 0, 1, \ldots, k-1; \\ \left(\sum_{j=0}^{2k} c_j a_{i+j-k}\right) \Big/ 2^k, & \text{for } 1 < i < N-1; \\ a_i, & \text{for } i = N-k+1, N-k+2, \ldots, N. \end{cases} \tag{8.4}$$

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



(a) (b)

Figure 8.10. Reducing the treble. Figure (a) shows the original sound signal, while the plot in (b) shows the result of applying the filter from row 4 of Pascal's triangle.



(a) (b)

Figure 8.11. Reducing the bass. Figure (a) shows the original sound signal, while the plot in (b) shows the result of applying the filter in (8.5).

An example of the result of the averaging is shown in figure 8.10. Figure (a) shows a real sound sampled at CD-quality (44 100 samples per second). Figure (b) shows the result of applying the averaging process in (8.6). We see that the oscillations have been reduced, and if we play the sound it has considerably less treble.

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 in (8.6) to

$$b_i = \begin{cases} a_i, & \text{for } i = 0, 1; \\ (a_{i-2} - 4a_{i-1} + 6a_i - 4a_{i+1} + a_{i+2})/16, & \text{for } 1 < i < N-1; \\ a_i, & \text{for } i = N-1, N. \end{cases} \tag{8.5}$$

An example is shown in figure 8.11. The original signal is shown in figure (a) and the result in figure (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 8.22. *Let $\boldsymbol{a}$ be the samples of a digital sound, and let $\{c_i\}_{i=0}^{2k}$ be the numbers in row $2k$ of Pascal's triangle. Then the sound with samples $\boldsymbol{b}$ given by*

$$b_i = \begin{cases} a_i, & \text{for } i = 0, 1, \ldots, k-1; \\ \left(\sum_{j=0}^{2k} (-1)^{k-j} c_j a_{i+j-k}\right) \Big/ 2^k, & \text{for } 1 < i < N-1; \\ a_i, & \text{for } i = N-k+1, N-k+2, \ldots, N. \end{cases} \tag{8.6}$$

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

8.4 More advanced sound processing

The operations on digital sound described in section 8.3 are simple and can be performed directly on the sample values. We saw in section 8.1.3 that a sound defined for continuous time could be decomposed into different frequency components, see theorem 8.8. The same can be done for digital sound with a digital version of the Fourier decomposition. When the sound has been decomposed into frequency components, the bass and treble can be adjusted by adjusting the corresponding frequencies. This is part of the field of *signal processing*.

8.4.1 The Discrete Cosine Transform

In Fourier analysis a sound is decomposed into sines and cosines. For digital sound a close relative, the Discrete Cosine Transform (DCT) is often used instead. This just decomposes the digital signal into cosines with different frequencies. The DCT is particularly popular for processing the sound before compression, so we will consider it briefly here.

Definition 8.23 (Discrete Cosine Transform (DCT))**.** *Suppose the sequence of numbers $\boldsymbol{u} = \{u_s\}_{s=0}^{n-1}$ are given. The DCT of $\boldsymbol{u}$ is the sequence $\boldsymbol{v}$ whose terms are given by*

$$v_s = \frac{1}{\sqrt{n}} \sum_{r=0}^{n-1} u_r \cos\left(\frac{(2r+1)s\pi}{2n}\right), \quad \text{for } s = 0, \ldots, n-1. \tag{8.7}$$

With the DCT we compute the sequence $\boldsymbol{v}$. It turns out that we can get back to the $\boldsymbol{u}$ sequence by computations that are very similar to the DCT. This is called the inverse DCT.

Theorem 8.24 (Inverse Discrete Cosine Transform)**.** *Suppose that the sequence $\boldsymbol{v} = \{v_s\}_{s=0}^{n-1}$ is the DCT of the sequence $\boldsymbol{u} = \{u_r\}_{r=0}^{n-1}$ as in* (8.7)*. Then $\boldsymbol{u}$ can be recovered from $\boldsymbol{v}$ via the formula*

$$u_r = \frac{1}{\sqrt{n}}\left(v_0 + 2\sum_{s=1}^{n-1} v_s \cos\left(\frac{(2r+1)s\pi}{2n}\right)\right), \quad \text{for } r = 0, \ldots, n-1. \tag{8.8}$$

The two formulas (8.7) and (8.8) allow us to switch back and forth between two different representations of the digital sound. The sequence $\boldsymbol{u}$ is often referred to as representation in the *time domain*, while the sequence $\boldsymbol{v}$ is referred to as representation in the *frequency domain*. There are fast algorithms for performing these operations, so switching between the two representations is very fast.

The new sequence $\boldsymbol{v}$ generated by the DCT tells us how much the sequence $\boldsymbol{u}$ contains of the different frequencies. For each $s = 0, 1, \ldots, n-1$, the function $\cos s\pi t$ is sampled at the points $t_r = (2r+1)/(2n)$ for $r = 0, 1, \ldots, n-1$ which results in the values

$$\cos\left(\frac{s\pi}{2n}\right), \quad \cos\left(\frac{3s\pi}{2n}\right), \quad \cos\left(\frac{5s\pi}{2n}\right), \quad \ldots, \quad \cos\left(\frac{(2n-1)s\pi}{2n}\right).$$

These are then multiplied by the u_r and everything is added together.

Plots of these values for $n = 6$ are shown in figure 8.12. We note that as s increases, the functions oscillate more and more. This means that v_0 gives a measure of how much constant content there is in the data, while (in this particular case where $N = 5$), v_5 gives a measure of how much content there is with maximum oscillation. In other words, the DCT of an audio signal shows the proportion of the different frequencies in the signal.

Once the DCT of $\boldsymbol{u}$ has been computed, we can analyse the frequency content of the signal. If we want to reduce the bass we can decrease the v_s-values with small indices and if we want to increase the treble we can increase the v_s-values with large indices.

8.5 Lossy compression of digital sound

In a typical audio signal there will be most information in the lower frequencies, and some frequencies will be almost completely absent, i.e., some of the v_s-values will be virtually zero. This can exploited for compression: We change

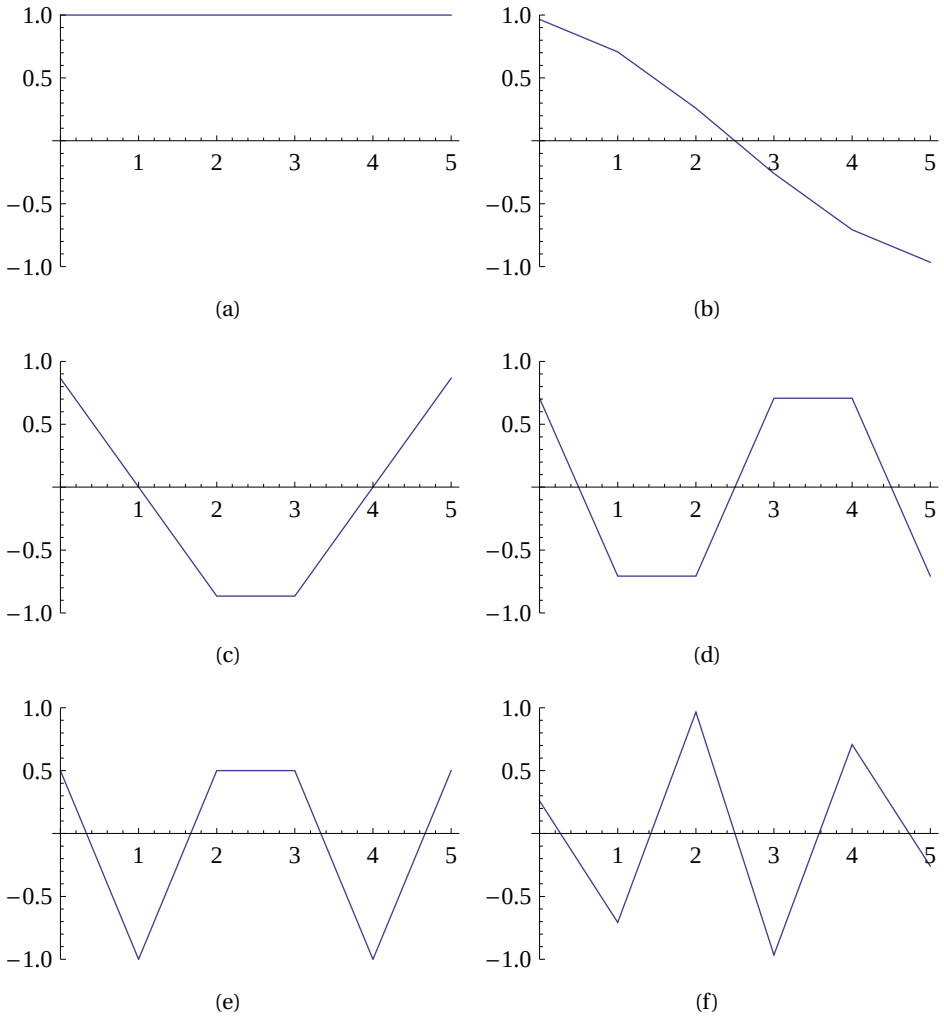


Figure 8.12. The 6 different versions of the cos function used in DCT for $n = 6$. The plots show piecewise linear functions, but this is just to make the plots more readable: Only the values at the integers 0, ..., 5 are used.

the small v_s-values a little bit and set them to 0, and then store the signal by storing the DCT-values. When the sound is to be played back, we first convert the sdjusted DCT-values to the time domain with the inverse DCT as given in theorem 8.24.

Example 8.25. Let us test a naive compression strategy based on the above idea. The plots in figure 8.13 illustrate the principle. A signal is shown in (a) and its

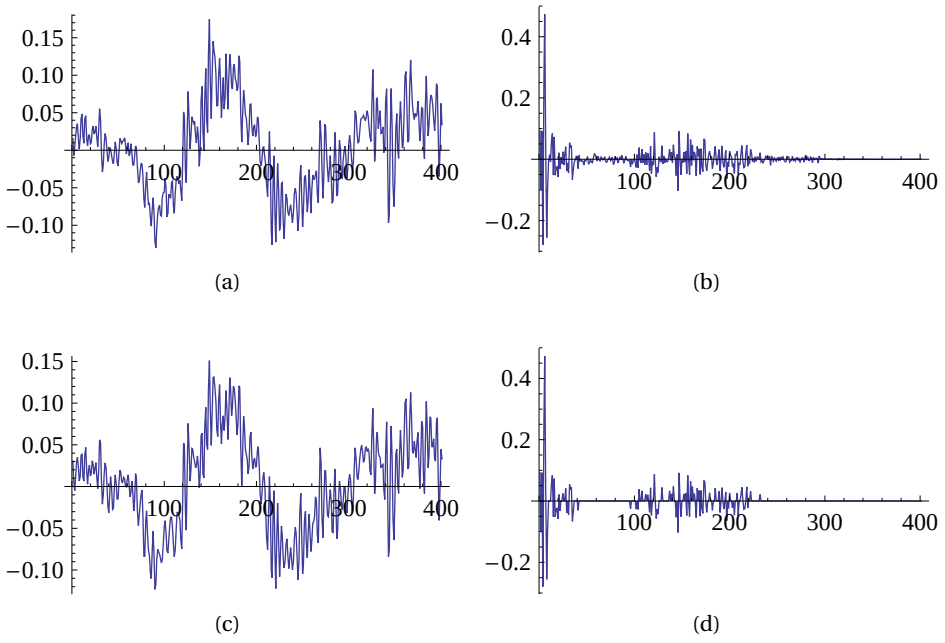


Figure 8.13. The signal in (a) is a small part of a song. The plot in (b) shows the DCT of the signal. In (d), all values of the DCT that are smaller than 0.02 in absolute value have been set to 0, a total of 309 values. In (c) the signal has been reconstructed from these perturbed values of the DCT. Note that all signals are discrete; the values have been connected by straight lines to make it easier to interpret the plots.

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 8.24; the result of this is shown in (c). The two signals in (a) and (b) visually look almost the same even though the signal in (c) can be represented with less than 25 % of the information present in (a).

We test this compression strategy on a data set that consists of 300 001 points. We compute the DCT and set all values smaller than a suitable tolerance to 0. With a tolerance of 0.04, a total of 142 541 values are set to zero. When we then reconstruct the sound with the inverse DCT, we obtain a signal that differs at most 0.019 from the original signal. 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 approach to compression that we have outlined in the above example is essentially what is used in practice. The difference is that commercial software does everything in a more sophisticated way and thereby gets better compres-

sion rates.

Fact 8.26 (Basic idea behind audio compression)**.** *Suppose a digital audio signal* $\boldsymbol{u}$ *is given. To compress* $\boldsymbol{u}$*, perform the following steps:*

1. *Rewrite the signal* $\boldsymbol{u}$ *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 information with some lossless coding method.*

All the lossy compression strategies used in the commercial formats that we review below, use the strategy in fact 8.26. In fact they all use a modified version of the DCT in step 1 and a variant of Huffman coding in step 3. Where they vary the most is probably in deciding what information to remove from the signal. To do this well requires some knowledge of human perception of sound.

8.6 Psycho-acoustic models

In the previous sections, we have outlined a simple strategy for compressing sound. The idea is to rewrite the audio signal in an alternative mathematical representation where many of the values are small, set the smallest values to 0, store this perturbed signal, and code it with a lossless compression method.

This kind of compression strategy works quite well, and is based on keeping the difference between the original signal and the compressed signal small. However, in certain situations a listener will not be able to perceive the sound as being different even if this difference is quite large. This is due to how our auditory system interprets audio signals and is referred to as *psycho-acoustic* effects.

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 Fourier-like 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.

The most obvious psycho-acoustic effect is that 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.

Another phenomenon is *masking effects*. A simple example of this is that a loud sound will make a simultaneous quiet 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.

These kinds of effects are integrated into what is referred to as a *psychoacoustic* model. This model is then used as the basis for simplifying the spectrum of the sound in way that is hardly noticeable to a listener, but which allows the sound to be stored with must less information than the original.

8.7 Digital audio formats

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. 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.

In this section we will give a brief description of some of the most common digital audio formats, both lossy and lossless ones.

8.7.1 Audio sampling — PCM

The basis for all digital sound is sampling of 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 rate and the number format varies for different kinds of audio. 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 64 000 bits per second.

The classical CD-format samples the audio signal 44 100 times per second and stores the samples 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.

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 as opposed to the two stereo channels on CDs).

Both the number of samples per second and the number of bits per sample 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 per sample. For standard telephony we saw that the bit rate is 64000 bits per second or 64 kb/s. 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 me be of different quality.

All the audio formats mentioned so far can be considered raw formats; it is a description of how the sound is digitised. When the information is stored on a computer, the details of how the data is organised must be specified, and there are several popular formats for this.

8.7.2 Lossless formats

The two most common file formats for CD-quality audio are AIFF and WAV, which are both supported by most commercial audio programs. These formats specify in detail how the audio data should be stored in a file. In addition, there is support for including the title of the audio piece, album and artist name and other relevant data. All the other audio formats below (including the lossy ones) also have support for this kind of additional information.

AIFF. *Audio Interchange File Format* was developed by Apple and published in 1988. AIFF supports different sample rates and bit lengths, but is most commonly used for storing CD-quality audio at 44 100 samples per second and 16 bits per sample. No compression is applied to the data, but there is also a vari-

ant that supports lossless compression, namely AIFF-C.

WAV. *Waveform audio data* is a file format developed by Microsoft and IBM. As AIFF, it supports different data formats, but by far the most common is standard CD-quality sound. WAV 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. Microsoft therefore developed the W64 format to remedy this.

Apple Lossless. After Apple's iPods became popular, the company in 2004 introduced a lossless compressed file format called Apple Lossless. This format is used for reducing the size of CD-quality audio files. Apple has not published the algorithm behind the Apple Lossless format, but most of the details have been worked out by programmers working on a public decoder. The compression phase uses a two step algorithm:

1. When the nth sample value x_n is reached, an approximation y_n to x_n is computed, and the error $e_n = x_n - y_n$ is stored instead of x_n. In the simplest case, the approximation y_n would be the previous sample value x_{n-1}; better approximations are obtained by computing y_n as a combination of several of the previous sample values.

2. The error e_n is coded by a variant of the *Rice algorithm*. This is an algorithm which was developed to code integer numbers efficiently. It works particularly well when small numbers are much more likely than larger numbers and in this situation it achieves compression rates close to the entropy limit. Since the sample values are integers, the step above produces exactly the kind of data that the Rice algorithm handles well.

FLAC. *Free Lossless Audio Code* is another compressed lossless audio format. FLAC is free and open source (meaning that you can obtain the program code). The encoder uses an algorithm similar to the one used for Apple Lossless, with prediction based on previous samples and encoding of the error with a variant of the Rice algorithm.

8.7.3 Lossy formats

All the lossy audio formats described below apply a modified version of the DCT to successive groups (frames) of sample values, analyse the resulting values, and perturb them according to a psycho-acoustic model. These perturbed values are then converted to a suitable number format and coded with some lossless coding method like Huffman coding. When the audio is to be played back, this

process has to be reversed and the data translated back to perturbed sample values at the appropriate sample rate.

MP3. Perhaps the best known audio format is MP3 or more precisely *MPEG-1 Audio Layer 3*. This format 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.

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. These MDCT values are then processed according to a psycho-acoustic model and coded efficiently with 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).

AAC. *Advanced Audio Coding* has been presented as the successor to the MP3 format by the principal MP3 developer, Fraunhofer Society. AAC 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 MDCT, just like MP3, but AAC processes 1 024 samples at 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 MDCT 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 net-

work, 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).

Ogg Vorbis. *Vorbis* is an open-source, lossy audio format that was designed to be free of any patent issues and free to use, and to be an improvement on MP3. At our level of detail Vorbis is very similar to MP3 and AAC: It uses the MDCT to transform groups of samples to the frequency domain, it then applies a psycho-acoustic model, and codes the final data with a variant of Huffman coding. In contrast to MP3 and AAC, Vorbis always uses variable length bit rates. The desired quality is indicated with an integer in the range -1 (worst) to 10 (best). Vorbis supports a wide range of sample rates from 8 kHz to 192 kHz and up to 255 channels. In comparison tests with the other formats, Vorbis appear to perform well, particularly at medium quality bit rates.

WMA. *Windows Media Audio* is a lossy audio format developed by Microsoft. WMA is also based on the MDCT and Huffman coding, and like AAC and Vorbis, it was explicitly designed to improve the deficiencies in MP3. WMA supports sample rates up to 48 kHz and two channels. There is a more advanced version, WMA Professional, which supports sample rates up to 96 kHz and 24 bit samples, but this has limited support in popular software and music players. There is also a lossless variant, WMA Lossless. At low bit rates, WMA generally appears to give better quality than MP3. At higher bit rates, the quality of WMA Pro seems to be comparable to that of AAC and Vorbis.