1 Introduction

Audio and speech processing systems have steadily risen in importance in the everyday lives of most people in developed countries. From 'Hi-Fi' music systems, through radio to portable music players, audio processing is firmly entrenched in providing entertainment to consumers. Digital audio techniques in particular have now achieved a domination in audio delivery, with CD players, Internet radio, MP3 players and iPods being the systems of choice in many cases. Even within television and film studios, and in mixing desks for 'live' events, digital processing now predominates. Music and sound effects are even becoming more prominent within computer games.

Speech processing has equally seen an upward worldwide trend, with the rise of cellular communications, particularly the European GSM (Global System for Mobile communications) standard. GSM is now virtually ubiquitous worldwide, and has seen tremendous adoption even in the world's poorest regions.

Of course, speech has been conveyed digitally over long distance, especially satellite communications links, for many years, but even the legacy telephone network (named POTS for 'Plain Old Telephone Services') is now succumbing to digitisation in many countries. The *last mile*, the several hundred metres of twisted pair copper wire running to a customer's home, was never designed or deployed with digital technology in mind, and has resisted many attempts over the years to be replaced with optical fibre, Ethernet or wireless links. However with DSL (digital subscriber line – normally asymmetric so it is faster in one direction than the other, hence ADSL), even this analogue twisted pair will convey reasonably high-speed digital signals. ADSL is fast enough to have allowed the rapid growth of Internet telephony services such as Skype which, of course, convey digitised speech.

1.1 Digital audio

Digital processing is now the method of choice for handling audio and speech: new audio applications and systems are predominantly digital in nature. This revolution from analogue to digital has mostly occurred over the past decade, and yet has been a quiet, almost unremarked upon, change.

It would seem that those wishing to become involved in speech, audio and hearing related research or development can perform much, if not all, of their work in the digital domain these days. One of the benefits of digital technology is that the techniques are

relatively device independent: one can create and prototype using one digital processing platform, and then deploy upon another platform. The criteria then for a development platform would be for ease-of-use and testing, while the criteria for a deployment platform may be totally separate: low power, small size, high speed, low cost, etc.

In terms of development ease-of-use, MATLAB running on a PC is chosen by many of those working in the field. It is well designed to handle digital signals, especially the long strings of audio samples. Built-in functions allow most common manipulations to be performed easily, audio recording and playback are equally possible, and the visualisation and plotting tools are excellent. A reduced-price student version is available which is sufficient for much audio work. The author runs MATLAB on both Mac OS-X and Linux platforms for much of his own audio work.

Although there is currently no speech, audio or hearing toolbox provided by The MathWorks® for MATLAB, the Signal Processing Toolbox contains most of the required additional functions, and an open source VOICEBOX is also available from the Department of Electrical and Electronic Engineering, Imperial College, London with many additional useful functions. It is also possible to perform all of the audio and speech processing in this book using the open source developed Octave environment, but would require some small changes to the MATLAB examples given. In terms of capabilities, Octave is less common than MATLAB, lacks the advanced plotting and debugging capabilities, but is otherwise similar.

1.2 Capturing and converting sound

This book is all about sound. Either sound created through the speech production mechanism, or sound as heard by a machine or human. In purely physical terms, sound is a longitudinal wave which travels through air (or a transverse wave in some other media) due to the vibration of molecules. In air, sound is transmitted as a pressure variation, between high and low pressure, with the rate of pressure variation from low, to high, to low again, determining the frequency. The degree of pressure variation (namely the difference between the high and the low) determines the amplitude.

A microphone captures sound waves, often by sensing the deflection caused by the wave on a thin membrane, transforming it proportionally to either voltage or current. The resulting electrical signal is normally then converted to a sequence of coded digital data using an analogue-to-digital converter (ADC). The most common format, pulse coded modulation, will be described in Section 5.1.1.

If this same sequence of coded data is fed through a compatible digital-to-analogue converter (DAC), through an amplifier to a loudspeaker, then a sound may be produced. In this case the voltage applied to the loudspeaker at every instant of time is proportional to the sample value from the computer being fed through the DAC. The voltage on the loudspeaker causes a cone to deflect in or out, and it is this cone which compresses (or rarifies) the air from instant to instant thus initiating a sound wave.

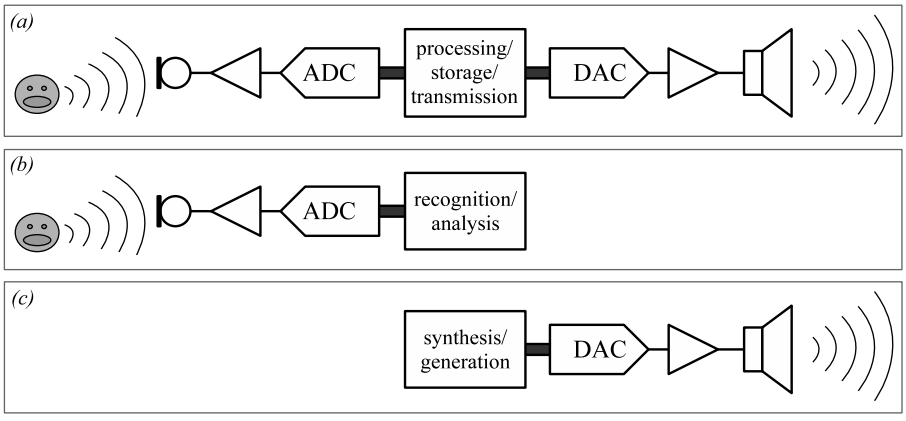

Figure 1.1 Block diagram of three classes of digital audio system showing (a) a complete digital audio processing system comprising (from left to right) an input microphone, amplifier, ADC, digital system, DAC, amplifier and loudspeaker. Variations also exist for systems recognising audio or speech (b), and systems synthesising audio (c).

In fact the process, shown diagrammatically in Figure 1.1(a), identifies the major steps in any digital audio processing system. Audio, in this case speech in free air, is converted to an electrical signal by a microphone, amplified and probably filtered, before being converted into the digital domain by an ADC. Once in the digital domain, these signals can be processed, transmitted or stored in many ways, and indeed may be experimented upon using MATLAB. A reverse process will then convert the signals back into sound.

Connections to and from the processing/storage/transmission system of Figure 1.1 (which could be almost any digital system) may be either serial or parallel, with several standard options being available in either case. Optical and wireless variants are also increasingly popular.

Variations on this basic system, such as shown in Figure 1.1(b) and (c), use a subset of the components for analysis or synthesis of audio. Stereo systems would have two microphones and loudspeakers, and some systems may have many more of either. The very simple amplifier, ADC and DAC blocks in the diagram also hide some of the complexities that would be present in many systems – such as analogue filtering, automatic gain control, and so on, in addition to the type (class) of amplification provided.

Both ADC and DAC are also characterised in different ways: by their sampling rates, technology, signal-to-noise ratio, and dynamic range, usually determined by the number of bits that they output.

1.3 Sampling

Considering a sequence of audio samples, first of all we note that the time spacing between successive samples is almost always designed to be uniform. The frequency of this timing is referred to as the sampling rate, and in Figure 1.1 would be set through

a periodic clock signal fed to the ADC and DAC, although there is no reason why both need the same sample rate – digital processing can be used to change sample rate. Using the well-known Nyquist criterion, the highest frequency that can be unambiguously represented by such a stream of samples is half of the sampling rate.

Samples themselves as delivered by ADC are generally fixed point with a resolution of 16 bits, although 20 bits and even up to 24 bits are found in high-end audio systems. Handling these on computer could utilise either fixed or floating point representation (fixed point meaning each sample is a scaled integer, while floating point allows fractional representation), with a general rule of thumb for reasonable quality being that 20 bits fixed point resolution is desirable for performing processing operations in a system with 16-bit input and output.

In the absence of other factors, the general rule is that an n bit uniformly sampled digital audio signal will have a dynamic range (the ratio of the biggest amplitude that can be represented in the system to the smallest one) of, at best:

$$\mathrm{DR(dB)} = 6.02 \times n. \tag{1.1}$$

For telephone-quality speech, resolutions as low as 8–12 bits are possible depending on the application. For GSM-type mobile phones, 14 bits is common. Telephone-quality, often referred to as toll-quality, is perfectly reasonable for vocal communications, but is not perceived as being of particularly high quality. For this reason, more modern vocal communication systems have tended to move beyond 8 bits sample resolution in practice.

Sample rates vary widely from 7.2 kHz or 8 kHz for telephone-quality audio to 44.1 kHz for CD-quality audio. Long-play style digital audio systems occasionally opt for 32 kHz, and high-quality systems use 48 kHz. A recent trend is to double this to 96 kHz. It is debatable whether a sampling rate of 96 kHz is at all useful to the human ear which can typically not resolve signals beyond about 18 kHz, apart from the rare listeners having *golden ears*.[1] However such systems may be more pet-friendly: dogs are reportedly able to hear up to 44 kHz and cats up to almost 80 kHz.

[1] The die-hard audio enthusiasts who prefer valve amplifiers, pay several years' salary for a pair of loudspeakers, and often claim they can hear above 20 kHz, are usually known as having *golden ears*.

Infobox 1.1 Audio fidelity

Something to note is the inexactness of the entire conversion process: what you hear is a wave impinging on the eardrum, but what you obtain on the computer has travelled some way through air, possibly bounced past several obstructions, hit a microphone, vibrated a membrane, been converted to an electrical signal, amplified, and then sampled. Amplifiers add noise, distortion, and are not entirely linear. Microphones are usually far worse on all counts. Analogue-to-digital converters also suffer linearity errors, add noise, distortion, and introduce quantisation error due to the precision of their voltage sampling process. The result of all this is a computerised sequence of samples that may not be as closely related to the real-world sound as you might expect. Do not be surprised when high-precision analysis or measurements are unrepeatable due to noise, or if delicate changes made to a sampled audio signal are undetectable to the naked ear upon replay.

Table 1.1. Sampling characteristics of common applications.

Application	Sample rate, resolution	Used how
telephony	8 kHz, 8–12 bits	64 kbps A-law or μ-law
voice conferencing	16 kHz, 14–16 bits	64 kbps SB-ADPCB
mobile phone	8 kHz, 14–16 bits	13 kbps GSM
private mobile radio	8 kHz, 12–16 bits	<5 kbps, e.g. TETRA
long-play audio	32 kHz, 14–16 bits	minidisc, DAT, MP3
CD audio	44.1 kHz, 16–24 bits	stored on CDs
studio audio	48 kHz, 16–24 bits	CD mastering
very high end	96 kHz, 20–24 bits	for *golden ears* listening

Sample rates and sampling precisions for several common applications, for humans at least, are summarised in Table 1.1.

1.4 Summary

Most of the technological detail related to the conversion and transmission process is outside the scope of this book, although some excellent resources covering this can be found in the bibliography. Generally, the audio processing specialist is fortunate enough to be able to work with digital audio without being too concerned with how it was captured, or how it will be replayed. Thus, we will confine our discussions throughout the remainder of this text primarily to the processing/storage/transmission, recognition/analysis and synthesis/generation blocks in Figure 1.1, ignoring the messy analogue detail.

Sound, as known to humans, has several attributes. These include time-domain attributes of duration, rhythm, attack and decay, but also frequency domain attributes of tone and pitch. Other, less well-defined attributes, include quality, timbre and tonality. Often, a sound wave conveys meaning: for example a fire alarm, the roar of a lion, the cry of a baby, a peal of thunder or a national anthem.

However, as we have seen, sound sampled by an ADC (at least the more common pulse coded modulation-based ADCs) is simply represented as a vector of samples, with each element in the vector representing the amplitude at that particular instant of time. The remainder of this book attempts to bridge the gap between such a vector of numbers representing audio, and an understanding or interpretation of the meaning of that audio.

Bibliography

- *Principles of Computer Speech*
  I. H. Witten (Academic Press, 1982)
  This book provides a gentle and readable introduction to speech on computer, written in an accessible and engaging style. It is a little dated in the choice of technology presented, but the underlying principles discussed remain unchanged.

- *The Art of Electronics*
  P. Horowitz and W. Hill (Cambridge University Press, 2nd edition 1989)
  For those interested in the electronics of audio processing, whether digital or analogue, this book is a wonderful introduction. It is clearly written, absolutely packed full of excellent information (on almost any aspect of electronics), and a hugely informative text. Be aware though that its scope is large: with over 1000 pages, only a fraction is devoted to audio electronics issues.

- *Digital Signal Processing: A Practical Guide for Engineers and Scientists*
  S. W. Smith (Newnes, 2002)
  Also freely available from www.dspguide.com
  This excellent reference work is available in book form, or directly from the website above. The author has done a good job of covering most of the required elements of signal processing in a relatively easy-to-read way. In general the work lives up to the advertised role of being practically oriented. Overall, a huge amount of information is presented to the reader; however it may not be covered gradually enough for those without a signal processing background.

2 Basic audio processing

Audio is normal and best handled by MATLAB, when stored as a vector of samples, with each individual value being a double-precision floating point number. A sampled sound can be completely specified by the sequence of these numbers plus one other item of information: the sample rate. In general, the majority of digital audio systems differ from this in only one major respect, and that is they tend to store the sequence of samples as fixed-point numbers instead. This can be a complicating factor for those other systems, but an advantage to MATLAB users who have two less considerations to be concerned with when processing audio: namely overflow and underflow.

Any operation that MATLAB can perform on a vector can, in theory, be performed on stored audio. The audio vector can be loaded and saved in the same way as any other MATLAB variable, processed, added, plotted, and so on. However there are of course some special considerations when dealing with audio that need to be discussed within this chapter, as a foundation for the processing and analysis discussed in the later chapters.

This chapter begins with an overview of audio input and output in MATLAB, including recording and playback, before considering scaling issues, basic processing methods, then aspects of continuous analysis and processing. A section on visualisation covers the main time- and frequency-domain plotting techniques. Finally, methods of generating sounds and noise are given.

2.1 Handling audio in MATLAB

Given a high enough sample rate, the double precision vector has sufficient resolution for almost any type of processing that may need to be performed – meaning that one can usually safely ignore quantisation issues when in the MATLAB environment. However there are potential resolution and quantisation concerns when dealing with input to and output from MATLAB, since these will normally be in a fixed-point format. We shall thus discuss input and output: first, audio recording and playback, and then audio file handling in MATLAB.

2.1.1 Recording sound

Recording sound directly in MATLAB requires the user to specify the number of samples to record, the sample rate, number of channels and sample format. For example, to record a vector of double precision floating point samples on a computer with attached or integrated microphone, the following MATLAB command may be issued:

```
speech=wavrecord(16000,8000,1,'double');
```

This records 16 000 samples with a sample rate of 8 kHz, and places them into a 16 000 element vector named `speech`. The '1' argument specifies that the recording is mono rather than stereo. This command only works under Windows, so under Linux or MacOS it is best to use either the MATLAB `audiorecorder()` function, or use a separate audio application to record audio (such as the excellent open source audacity tool), saving the recorded sound as an audio file, to be loaded into MATLAB as we shall see shortly.

Infobox 2.1 Audio file formats

Wave: The wave file format is usually identified by the file extension .wav, and actually can hold many different types of audio data identified by a header field at the beginning of the file. Most importantly, the sampling rate, number of channels and number of bits in each sample are also specified. This makes the format very easy to use compared to other formats that do not specify such information, and thankfully this format is recognised by MATLAB. Normally for audio work, the wave file would contain PCM data, with a single channel (mono), and 16 bits per sample. Sample rate could vary from 8000 Hz up to 48 000 Hz. Some older PC sound cards are limited in the sample rates they support, but 8000 Hz and 44 100 Hz are always supported. 16 000 Hz, 24 000 Hz, 32 000 Hz and 48 000 Hz are also reasonably common.

PCM and **RAW** hold streams of pulse coded modulation data with no headers or gaps. They are assumed to be single channel (mono) but the sample rate and number of bits per sample are not specified in the file – the audio researcher must remember what these are for each .pcm or .raw file that he or she keeps. These can be read from and written to by MATLAB, but are not supported as a distinctive audio file. However these have historically been the formats of choice for audio researchers, probably because research software written in C, C++ and other languages can most easily handle this format.

A-law and **μ-law** are logarithmically compressed audio samples in byte format. Each byte represents something like 12 bits in equivalent linear PCM format. This is commonly used in telecommunications where the sample rate is 8 kHz. Again, however, the .au file extension (which is common on UNIX machines, and supported under Linux) does not contain any information on sample rate, so the audio researcher must remember this. MATLAB does support this format natively.

Other formats include those for compressed music such as MP3 (see Infobox: Music file formats on page 11), MP4, specialised musical instrument formats such as MIDI (musical instrument digital interface) and several hundred different proprietary audio formats.

If using the `audiorecorder()` function, the procedure is first to create an audio recorder object, specifying sample rate, sample precision in bits, and number of channels, then to begin recording:

```
aro=audiorecorder(16000,16,1);
record(aro);
```

At this point, the microphone is actively recording. When finished, stop the recording and try to play back the audio:

```
stop(aro);
play(aro);
```

To convert the stored recording into the more usual vector of audio, it is necessary to use the `getaudiodata()` command:

```
speech=getaudiodata(aro, 'double');
```

Other commands, including `pause()` and `resume()`, may be issued during recording to control the process, with the entire recording and playback operating as background commands, making these a good choice when building interactive speech experiments.

2.1.2 Storing and replaying sound

In the example given above, the 'speech' vector consists of double precision samples, but was recorded with 16-bit precision. The maximum representable range of values in 16-bit format is between −32 768 and +32 767, but when converted to double precision is scaled to lie with a range of +/−1.0, and in fact this would be the most universal scaling within MATLAB so we will use this wherever possible. In this format, a recorded sample with integer value 32 767 would be stored with a floating point value of +1.0, and a recorded sample with integer value −32 768 would be stored with a floating point value of −1.0.

Replaying a vector of sound stored in floating point format is also easy:

```
sound(speech, 8000);
```

It is necessary to specify only the sound vector by name and the sample rate (8 kHz in this case, or whatever was used during recording). If you have a microphone and speakers connected to your PC, you can play with these commands a little. Try recording a simple sentence and then increasing or reducing the sample rate by 50% to hear the changes that result on playback.

Sometimes processing or other operations carried out on an audio vector will result in samples having a value greater than +/−1.0, or in very small values. When replayed using `sound()`, this would result in clipping, or inaudible playback respectively. In such cases, an alternative command will automatically scale the audio vector prior to playback based upon the maximum amplitude element in the audio vector:

```
soundsc(speech, 8000);
```

This command scales in both directions so that a vector that is too quiet will be amplified, and one that is too large will be attenuated. Of course we could accomplish something similar by scaling the audio vector ourselves:

```
sound(speech/max(abs(speech)), 8000);
```

It should also be noted that MATLAB is often used to develop audio algorithms that will be later ported to a fixed-point computational architecture, such as an integer DSP (digital signal processor), or a microcontroller. In these cases it can be important to ensure that the techniques developed are compatible with integer arithmetic instead of floating point arithmetic. It is therefore useful to know that changing the 'double' specified in the use of the `wavrecord()` and `getaudio()` functions above to an 'int16' will produce an audio recording vector of integer values scaled between −32 768 and +32 767.

The audio input and output commands we have looked at here will form the bedrock of much of the process of audio experimentation with MATLAB: graphs and spectrograms (a plot of frequency against time) can show only so much, but even many experienced audio researchers cannot repeatedly recognise words by looking at plots! Perfectly audible sound, processed in some small way, might result in highly corrupt audio that plots alone will not reveal. The human ear is a marvel of engineering that has been designed for exactly the task of listening, so there is no reason to assume that the eye can perform equally as well at judging visualised sounds. Plots can occasionally be an excellent method of visualising or interpreting sound, but often listening is better.

A time-domain plot of a sound sample is easy in MATLAB:

```
plot(speech);
```

although sometimes it is preferred for the x-axis to display time in seconds:

```
plot( [ 1: size(speech) ]  / 8000, speech);
```

where again the sample rate (in this case 8 kHz) needs to be specified.

2.1.3 Audio file handling

In the audio research field, sound files are often stored in a raw PCM (pulse coded modulation) format. That means the file consists of sample values only – with no reference to sample rate, precision, number of channels, and so on. Also, there is a potential endian problem for samples greater than 8 bits in size if they have been handled or recorded by a different computer type.