1 Introduction

Audio processing systems have been a part of many people's lives since the invention of the phonograph in the 1870s. The resulting string of innovations sparked by that disruptive technology have culminated eventually in today's portable audio devices such as Apple's iPod, and the ubiquitous MP3 (or similarly compressed) audio files that populate them. These may be listened to on portable devices, computers, as soundtracks accompanying Blu-ray films and DVDs, and in innumerable other places.

Coincidentally, the 1870s saw a related invention – that of the telephone – which has also grown to play a major role in daily life between then and now, and likewise has sparked a string of innovations down the years. Scottish born and educated Alexander Graham Bell was there at their birth to contribute to the success of both inventions. He probably would be proud to know, were he still alive today, that two entire industry sectors, named telecommunications and infotainment, were spawned by the two inventions of phonograph and telephone.

However, after 130 years, something even more unexpected has occurred: the descendents of the phonograph and the descendents of the telephone have converged into a single product called a 'smartphone'. Dr Bell probably would not recognise the third convergence that made all of this possible, that of the digital computer – which is precisely what today's smartphone really is. At heart it is simply a very small, portable and capable computer with microphone, loudspeaker, display and wireless connectivity.

1.1 Computers and audio

The flexibility of computers means that once sound has been sampled into a digital form, it can be used, processed and reproduced in an infinite variety of ways without further degradation. It is not only computers (big or small) that rely on digital audio, so do CD players, MP3 players (including iPods), digital audio broadcast (DAB) radios, most wireless portable speakers, television and film cameras, and even modern mixing desks for 'live' events (and co-incidentally all of these devices contain tiny embedded computers too). Digital music and sound effects are all around us and impact our leisure activities (e.g. games, television, videos), our education (e.g. recorded lectures, broadcasts, podcasts) and our work in innumerable ways to influence, motivate and educate us. Beyond music, we can find examples of digital audio in recorded announcements, the beep of electronic devices, ringing mobile telephones, many alarm sirens, modern

hearing aids, and even the sound conveyed by the old-fashioned telephone network (named POTS for 'plain old telephone service'), which is now a cyborg of interconnected digital and analogue elements.

All of this digital audio impinging on our sensitive ear drums should not cause our mouths to feel neglected: digital speech processing has seen an equally impressive and sustained upward worldwide trend in performance and popularity. The well-charted cellular communications revolution begun by the European GSM (Global System for Mobile communications) standard has now led to mobile phones, cellphones, handphones and smartphones being virtually ubiquitous. Tremendous numbers of mobile phones are sold annually, even in the world's poorest regions, and few would disagree today that a mobile phone – a digital speech computer – is an essential consumer or business device. However, these devices have become more than just mobile telephones, as evidenced by the growing importance of smartphones with large colourful screens; a feature that would be unlikely to be much appreciated by one's ear, which gets the best view of the screen when the device is operating as a more traditional telephone.

It seems at first glance that, as these intelligent, desirable and useful devices worm their way further into our daily life, the relative importance of using the devices to convey 'speech' is reducing, compared with the importance of the visual display. In other words, they are moving away from being primarily telephone devices. In fact, a non-scientific survey of teenagers' attitudes to the essentials of human existence yields something like Figure 1.1, which might support the conjecture that telephony and speech are declining in importance. However, this impression would be untrue for two interesting reasons. The first is that the increasing adoption patterns of smartphones indicate that many of the newer uses (e.g. playing games, watching episodes of television or films, instant chatting) require and rely upon sound and speech just as much as a telephone does. The second is in the growing role of speech-based interfaces. These probably first became prominent with the general public through Apple's Siri, on the iPhone 4s, which could be spoken to using relatively natural language, and which would respond accordingly. Siri was able to order tickets, book flights, find directions, contact friends and relatives, advise on weather, traffic, restaurants and so on. In fact, Siri could be asked almost any question and would attempt to provide an appropriate answer. To many, this began to sound like a kind of electronic personal assistant, or butler. In fact

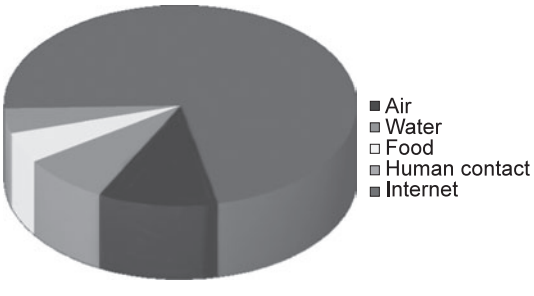

Figure 1.1 Essential human needs – according to teenagers.

the 2011 release of Siri was pre-dated by about 3 years by the Marvel Comics film *Iron Man* in which hero Tony Stark's suit of armour communicates with him using spoken natural language. The system, named J.A.R.V.I.S., is obviously a dramatisation played by an actor rather than a real system; however, it is a placeholder for the archetypal electronic butler of the future. It might not surprise readers to find out that there are currently a significant number of university research groups around the world who are working on making such systems a reality, as well as some large companies such as Google, IBM, iFlytek, Amazon, Samsung, and probably several telecommunications giants too. With this much brainpower and research expenditure, few would doubt that it is purely a matter of time until something (or somebody?) like J.A.R.V.I.S. becomes reality.

1.2 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 two decades, as 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, apart from the interface which captures sound (microphone) and outputs it (loudspeaker). One of the great 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 using another platform, and the behaviour of both systems will be identical. Given that, the criteria for a development platform would then 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 computer 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 very 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 toolbox called VOICEBOX is also available from the Department of Electrical and Electronic Engineering, Imperial College, London, which contains many additional useful functions.[1]

[1] VOICEBOX, released courtesy of Mike Brookes of Imperial College, can be downloaded from www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html

All of the audio and speech processing in this book can also be executed using the open source Octave environment,[2] although some of the MATLAB examples may require a few small changes – usually some vectors will need to be transposed. Octave is less common than the industry standard MATLAB, and lacks one or two of the advanced plotting and debugging capabilities, but is otherwise very similar in capabilities. It is also highly efficient, easily handles parallel processing on a cluster computer, and can be integrated with other languages such as Python for script- or web-based automation.

1.3 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 which varies between high and low levels, 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 pressures) 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 6.1.1.

If this 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 rarefies) the air from instant to instant, thus initiating a sound pressure wave.

In fact the process, shown diagrammatically in Figure 1.2(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 1.2 (which could be almost any digital system) may be either serial or parallel, with many possible connectivity options in practice.

Variations on this basic theme, such as those shown in Figures 1.2(b) and (c), use a subset of the components for analysis or synthesis of audio. Stereo systems would

[2] GNU Octave is an open source alternative to MATLAB, and is available for download from www.gnu.org/software/octave

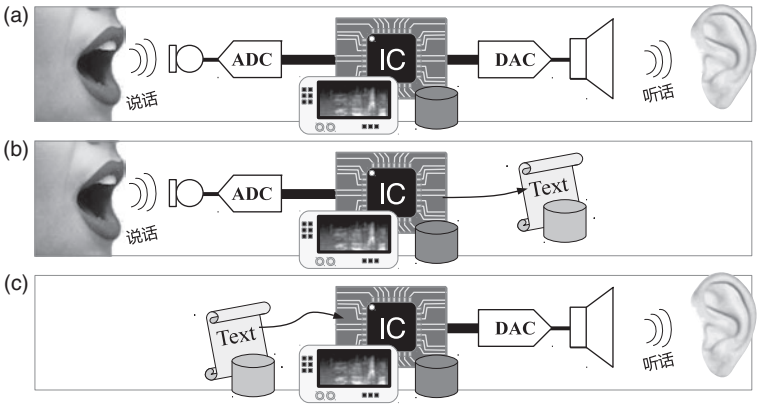

Figure 1.2 Three classes of digital audio system: (a) a complete digital audio processing path incuding (from left to right) an input microphone, amplifier, ADC, processing system, DAC, amplifier and loudspeaker. (b) A system recognising audio or speech. (c) A system that synthesises speech or audio.

have two microphones and loudspeakers, and some systems may have many more of both. 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, conversion technology, signal-to-noise ratio, linearity and dynamic range (which is related to the number of bits that they output in each sample).

1.4 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 1.2 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 the 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 an 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.

Box 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, create distortion, and are not entirely linear. Microphones are usually far worse on all counts. Analogue-to-digital converters also suffer linearity errors, add noise, create 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.

In the absence of other factors, 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:

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

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 human ears, which can typically not resolve signals beyond about 18 kHz even when young – apart from the rare listeners having *golden ears*.[3] 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.

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

1.5 Summary

The technological fine detail related to the conversion and transmission process for audio is outside the scope of this book, which is more concerned with experimentation, analysis and digital speech/audio manipulation. Today's audio processing specialist is very fortunate to be able to work with digital audio without being too concerned with

[3] The die-hard audio enthusiasts, *audiophiles*, 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*.

Table 1.1 Sampling characteristics of common applications.

Application	Sample rate, resolution	How used
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

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 1.2, ignoring the ADCs, DACs, transducers and other messy analogue detail.

However, it is important to remember that sound, as perceived by humans, is a real, continuously varying, noisy, analogue signal that 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. Very 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, a national anthem or a spoken word.

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 sound 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, as well as how that vector of numbers can be manipulated to apply meaningful processing (usually inside MATLAB).

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, along with some simple examples (in BASIC though). By now, this text is definitely 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 is a hugely informative text. Be aware, though, that its scope is large: with over 1000 pages, only a fraction of the book 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 very large amount of good information is presented to the reader, although this may not always be covered gradually enough for those without a signal processing background.

2 Basic audio processing

Most speech and audio researchers use MATLAB as a preferred tool for audio processing, although many of us will make use of other specialised tools from time to time, such as *sox* for command line audio processing[1] (particularly when there are a large number of files to convert or process, something it can do with a single command line option), and the sound capture and editing tool *audacity* which can record, edit, manipulate, convert and play back numerous types of audio file.[2] In fact both of these programs are extremely capable open source tools, having far more options than could be described here. However, while very useful, neither tool can replace the abilities of MATLAB to easily develop scripts that make use of hundreds of built-in functions and operators, and can plot or visualise speech and other sounds in a multitude of ways.

Recorded speech or other sounds are stored within MATLAB (as well as in many other computer-based tools) 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 vector of these numbers as long as one other item of information is known: the sample rate at which the data was recorded. To replay the sampled sound, it is only necessary to sequentially output a voltage proportional to the stored vector information, with a gap between samples equivalent to the inverse of the sample rate.

General audio programs and tools store audio information similarly, except that they tend to use fixed point numbers rather than floating point, which can reduce the storage requirement by a factor of four at the expense of very little degradation – assuming the system is correctly designed. In particular, a consideration of overflow and underflow effects is usually needed when designing a system that uses fixed point storage for audio, whereas in floating point-based tools such as MATLAB this is rarely a concern in practice.

Any operation that MATLAB can perform on a general vector can, in theory, be performed on stored audio. In fact, this is how we typically perform audio processing within MATLAB, and the audio vector can be loaded and saved in much the same way as any other MATLAB variable. Likewise it can be processed, added, plotted, inverted, transformed and so on.

[1] The sox tool can be freely downloaded from http://sox.sourceforge.net/sox.html

[2] Audacity is a free, cross-platform and open source sound audio editor and recorder. It is available for download from http://web.audacityteam.org

However, there are of course some special considerations when dealing with audio that should be discussed within this chapter, as a foundation for later chapters where we shall use MATLAB to implement some real speech and audio-based examples.

This chapter thus begins with an overview of audio input and output in MATLAB, including recording and playback directly within the program. We then consider scaling issues (including overflow and underflow), basic processing methods and the important issues of continuous analysis and processing. A section on visualisation covers the main time- and frequency-domain plotting techniques that we use to look at and understand the signals that we are working with. Finally, some methods of generating sounds and noise are presented, which will be useful in later chapters.

2.1 Sound in MATLAB

With a high enough sample rate, the double precision vector has sufficient resolution for almost any processing that may need to be performed. This means that one can usually safely ignore quantisation issues when processing in MATLAB. However, there are potential resolution and quantisation concerns when we need to input data into, or output data from, MATLAB, including the recording and replaying of sound. It is because input and output will normally be in a fixed point format, as will audio data stored in most files. We will thus look at an overview of audio input and output from MATLAB, starting with audio recording and playback, and then audio file handling.

2.1.1 Recording sound

Recording sound directly in MATLAB has changed in the latest versions, and now requires use of the `audiorecorder()` function. A much recommended alternative is to use a separate audio application to record sound (such as the excellent open source Audacity tool). Audacity is much easier to control and operate than the built-in MATLAB audio commands, and allows intuitive and fast editing of sounds which are displayed by waveform. Once the recorded data is ready to be used in MATLAB it can be exported by Audacity to a file in `.wav` format (or any other standard format), which can then easily be read into MATLAB as we will see in Section 2.1.3.

If Audacity is not used, the `audiorecorder()` function in MATLAB requires an audio recorder object to first be created, specifying sample rate, sample precision in bits, and number of channels, before recording can begin. This is done as follows:

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

At this point, after entering the `record()` command, although there may be no visible indication, the computer should be actively recording sound (always assuming a