CHAPTER 1

Introduction

1.1 THE SCOPE OF DIGITAL SIGNAL PROCESSING

1.1.1 BACKGROUND

The extraordinary growth of microelectronics and computing has had a major impact on digital signal processing (DSP). Powerful DSP techniques are now used to analyze and process signals and data arising in many areas of engineering and science, medicine, economics, and the social sciences. Most people trained in a numerate discipline can expect to encounter DSP at some stage in their careers, and there is every sign that the trend will continue.

DSP is concerned with the numerical manipulation of signals and data in sampled form. Using such elementary operations as digital storage and delay, addition, subtraction, and multiplication by constants, we can produce a wide variety of useful functions. For example, we may need to detect trends in data; to extract a wanted signal from unwanted 'noise'; or to assess the frequencies present in a signal.

Many readers of this book will have a background in electronic and electrical engineering. Others will come to DSP via the other branches of engineering, physics, computer science, or mathematics. And some of you will be interested in data processing applied to economics and the social sciences. We therefore assume no knowledge of electronics or computer hardware. Our approach is to introduce the basic theory and applications of DSP, illustrating important concepts and design techniques with the help of computer programs. The programs, listed in both C and PASCAL in Appendix A1, are suitable for a wide of personal computers — and, of course, more powerful machines. It is not essential that you run the programs for yourself, because their graphical outputs are reproduced at various points in the text. However, we hope that many of you will have the opportunity to do so.

The use of general-purpose computers is straightforward and convenient for illustrating DSP theory and applications. However if high-speed real-time signal processing is required, it may be essential to use special-purpose digital hardware. Intermediate between these two extremes come programmable microprocessors, possibly attached to a general-purpose host computer. Many manufacturers now produce DSP microprocessors, tailored to the requirements of signal processing. The use of such devices, and the design of dedicated DSP

hardware, is a fast-growing field. So although we concentrate here on general-purpose computers, we must always remember that they are only a part of modern DSP.

Various terms are used to describe signals in the DSP environment. By *discrete-time signal*, we mean a signal which is defined only for a particular set of instants in time, or *sampling instants*. A good example would be the midday temperature at a certain place, measured on successive days. Discrete-time signals may be subdivided into two categories: *sampled-data signals*, which display a continuous range of amplitude values; and *digital signals*, in which the amplitude values are quantized in a series of finite steps. Signals stored or processed in a computer are digital, because each sample value is represented by a finite-length binary code. However, in practice the terms discrete-time signal and digital signal are often used interchangeably.

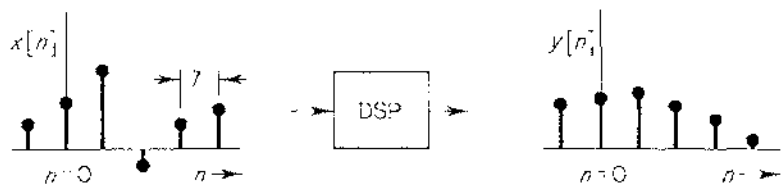

Figure 1.1 Input and output signals of a digital signal processor.

Two discrete-time signals are shown in Figure 1.1. Here $x[n]$ is the sampled input to a digital signal processor, and $y[n]$ is the sampled output. Successive input signal values $\ldots\ x[-1],\ x[0],\ x[1],\ x[2]\ \ldots$ may be thought of as regularly-spaced samples of an underlying analog signal, and are separated from one another by the system sampling interval T. The integer variable n denotes the sample number, with $n = 0$ corresponding to some convenient time origin or reference.

Note that x and y are not *necessarily* time functions. For example they may be functions of distance. However, for convenience we generally assume that n is a time variable.

In effect we may regard $x[n]$ and $y[n]$ as sequences of numerical data, which are binary-coded for the purposes of signal processing. Although we restrict ourselves in this book to one-dimensional signals, two-dimensional (or even multi-dimensional) DSP is of increasing practical importance. A good example is the processing of pictures and images, such as those obtained from X-ray brain-scanners or from satellite reconnaissance. Fortunately the techniques of one-dimensional DSP can be extended to such applications without too much difficulty.

Figure 1.2 illustrates a typical DSP scheme. It is often said that we live in an analog world, and most practical signals start off in analog form. Examples might be variations in voltage, temperature, pressure, or light intensity. If the signal is not inherently electrical, it is first converted to a proportional voltage fluctuation by a suitable transducer. The transducer provides the analog input to the signal processing chain. Very often, the first stage in the chain is an analog filter, designed to limit the frequency range of the signal prior to sampling. The reasons for this will become clear in Section 1.2.

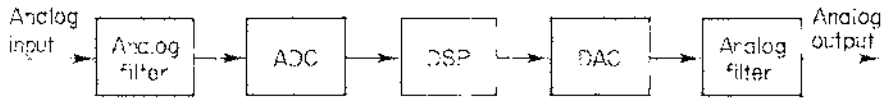

Figure 1.2 A typical DSP scheme.

The signal is next sampled and converted into a binary code by an analog-to-digital converter (ADC). After digital signal processing it may be changed back into analog form using a digital-to-analog converter (DAC). The final processing stage is often another analog filter, designed to remove sharp transitions from the DAC output.

There are a number of possible variations on the above theme. For example, after DSP the signal may be used to drive a computer display, or it may be transmitted in binary form to a remote terminal or location. Another important variation concerns data processing applications in such fields as economics and the social sciences. Here the numerical data representing the input signal may be available on a computer disk, or it may be entered on a keyboard by a computer operator. In such cases sampling and analog-to-digital conversion is an inherent part of the data entry process. Similarly, a computer print-out of processed sample values may take the place of digital-to-analog conversion.

To an electronic engineer, the DSP scheme in Figure 1.2 may seem rather a complicated way of proceeding. There are, after all, many well-established signal processing techniques based on analog electronic circuits and components. Why do we go to the trouble of converting signals into digital form, and then back again? There are several good reasons:

Signals and data of many types are increasingly stored in digital computers, and transmitted in digital form from place to place. In many cases it makes sense to process them digitally as well.

Digital processing is inherently stable and reliable. It also offers certain technical possibilities not available with analog methods.

Rapid advances in integrated circuit design and manufacture are producing ever more powerful DSP devices at decreasing cost.

In many case DSP is used to process a number of signals simultaneously. This may be done by *interlacing* samples obtained from the various signal channels — a technique known as *time-division-muliplexing*.

Of course, those of you who are interested in data processing applications will know that there is no realistic substitute for digital methods. The nature of your numerical data, and the way in which it is gathered, make computer storage and processing the obvious choice.

1.1.2 SOME PRACTICAL APPLICATIONS

We hope that you now have an idea of the general flavor of DSP. In this section we illustrate some practical applications drawn from different disciplines. Our aim is to provide motivation, and to show something of the power and

flexibility of DSP methods. We certainly do not expect you to understand all the details at this stage.

Our first illustration should be quite easy to understand, regardless of your background. Figure 1.3 shows changes in the dollar price of gold between late 1979 and 1983. The heavy, rapidly fluctuating, curve represents the price recorded day by day. There are more than 1500 values altogether, so for convenience we have joined the samples together to give a continuous curve.

It is often helpful to smooth such 'raw data', to reduce rapid fluctuations and reveal underlying trends. A widely-used technique is to estimate a *moving average*. Each smoothed value is computed as the average of a number of preceding raw-data values, and the process is repeated sample by sample through the record. The figure shows the 200-day moving average (with individual values again joined together for convenience). 200-Day averages are popular with analysts of financial and stock market data for investigating long-term trends. Clearly, such processing gives a very pronounced smoothing action.

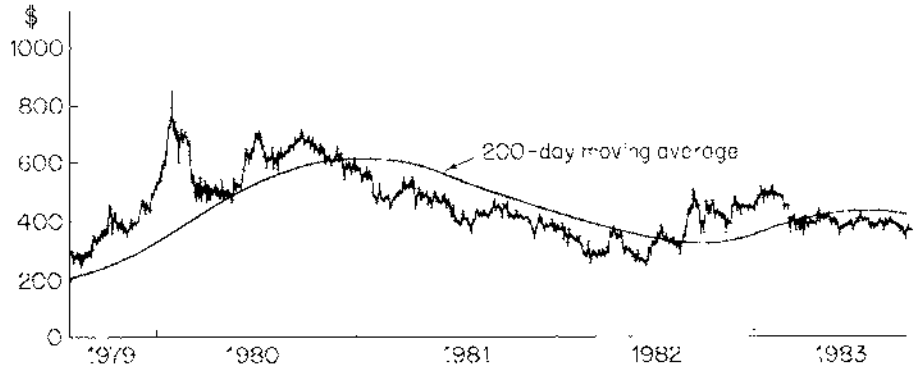

Figure 1.3 The dollar price of gold, 1979–1983.

From the DSP point of view, the daily gold price provides samples of an input signal $x[n]$. The smoothed values represent output samples $y[n]$ from the moving-average process. The 200-day average may be estimated using the algorithm:

$$y[n] = \frac{1}{200}\{x[n] + x[n-1] + x[n-2] + \cdots x[n-199]\}$$

$$= 0.005 \sum_{k=0}^{199} x[n-k] \tag{1.1}$$

It is a simple matter to store the raw data in a computer, and to write a program for calculating $y[n]$. Every time a new value of x becomes available, we can produce a new value of y.

Equation (1.1) is a *recurrence formula*, which is used over and over again to find successive values of y. It is also called a *difference equation*. It is *nonrecursive*, because each output sample is computed solely from input values. And

the equation defines the operation of a simple *digital filter*, which in this case produces a smoothing action on the input data. Much of DSP is concerned with the design of digital filters, which can be tailored to produce a wide variety of processing functions.

You may have noticed that the above moving-average algorithm is not very efficient. The computation of each output value is almost exactly the same as the one before it, except that the most recent input sample is included, and the most distant one is discarded. Using this idea, we can generate a much more efficient algorithm. Equation (1.1) shows that the *next* smoothed output value must be:

$$y[n+1] = 0.005\{x[n+1] + x[n] + x[n-1] + \cdots x[n-198]\}$$
$$+ y[n] + 0.005\{x[n+1] - x[n-199]\}$$

Since this is a recurrence formula which applies for *any* value of n, we may subtract 1 from each term in square brackets, giving:

$$y[n] = y[n-1] + 0.005\{x[n] - x[n-200]\} \tag{1.2}$$

Equation (1.2) confirms that we can estimate each output sample by *updating the previous output* $y[n-1]$. The equation defines a *recursive* version of the filter which is much more efficient, requiring far fewer additions/subtractions. Note that equations (1.1) and (1.2) are equivalent from the signal processing point of view.

Although Figure 1.3 refers to financial data, similar smoothing operations are widely used in DSP. For example, electronic engineers often need to remove unwanted interference, or 'noise', from a relatively slowly-varying signal. A smoothing filter used for this purpose is known as a *low-pass filter*. That is, it passes (transmits) low frequencies, representing slow fluctuations; but reduces high frequencies.

In medicine, the electrical activity of the heart can be recorded using electrodes placed on the chest. The resulting *electrocardiogram* (EKG) is often contaminated by rapid fluctuations due to electrical activity in nearby muscles. These, too, may be reduced by processing with a low-pass filter. We therefore see that the type of DSP action shown in Figure 1.3 has a wide range of practical uses. In fact there are many types of digital low-pass filter, and the simple moving-average design we have described would not be suitable for most applications.

Mention of the EKG leads to our next illustration of DSP. Figure 1.4(a) shows a typical EKG waveform, corresponding to a single heartbeat. In part (b) of the figure it is badly contaminated – not in this case by muscle 'noise', but by sinusoidal interference at mains supply frequency (60 Hz in the USA, 50 Hz in Europe). This is quite a common problem when recording biomedical signals, due to pick-up in electrode leads. It must obviously be reduced before the signal can have diagnostic value. Since biomedical signals are often stored in digital computers, we have the opportunity to reduce the interference with a digital filter.

Before describing the filter algorithm, we should explain the form of the sampled signals in the figure. They have been drawn by computer, and each sample value is represented by a thin vertical bar. In this case there are 640

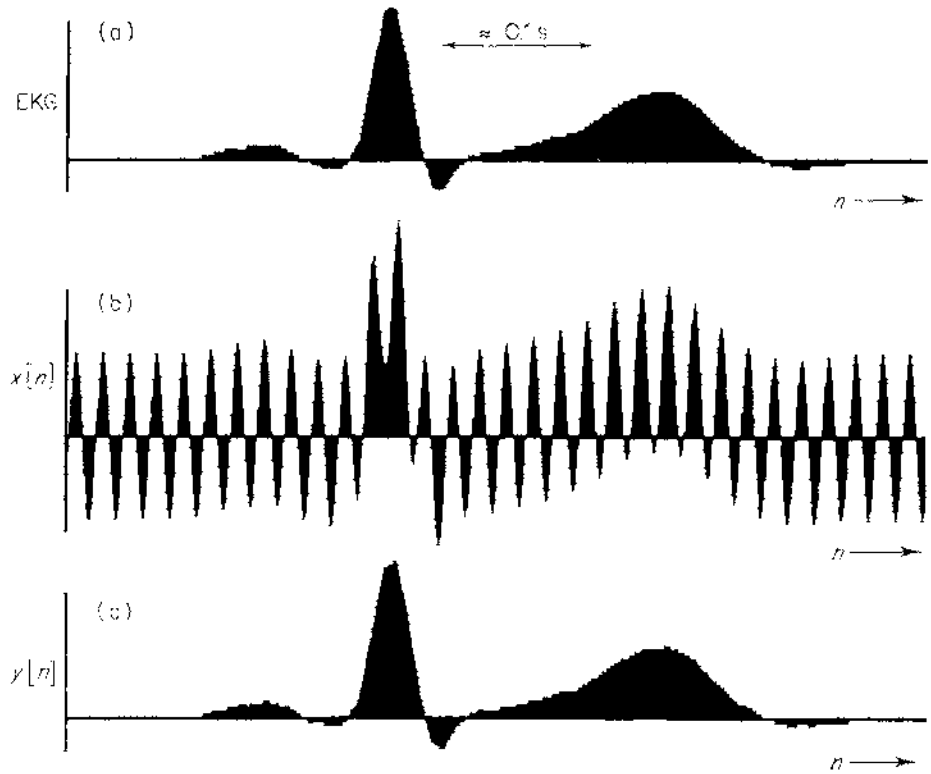

Figure 1.4 Removing mains-frequency interference from an electocardiogram (EKG) *(abscissa: 640 samples).*

sample values across the page, and the vertical bars are contiguous. This gives the plots their solid appearance. You will meet many more examples of such computer plots in this book, and will have the opportunity to try later programs for yourself on a personal computer. Note that in most cases there will be 320 sample values (or less) across the page, sometimes with gaps between adjacent vertical bars. The figure caption will make clear how many samples are plotted.

Let us now return to the problem of filtering the EKG. In this case we do not need a smoothing (low-pass) design, but one which rejects a narrow band of frequencies centered at mains supply frequency. This is called a *bandstop*, or *notch*, filter. The following recurrence formula may be used:

$$y[n] = 1.8523\,y[n-1] - 0.94833\,y[n-2] + x[n] - 1.9021\,x[n-1] + x[n-2] \tag{1.3}$$

Figure 1.4(c) shows the dramatic effect of this recursive filter on the contaminated signal of part (b). The interference has been greatly reduced, without significantly distorting the signal waveform.

At this stage we certainly do not expect you to understand how equation (1.3) has been derived. The illustration is intended to whet the appetite for some DSP theory in later chapters! However, we should just add that a digital

filter is always designed for a particular sampling rate. If the interference is at 60 Hz, this algorithm is effective at a sampling frequency of 1200 samples per second (1.2 kHz), if it is at 50 Hz, the algorithm is effective at 1000 samples per second (1 kHz).

Our third introductory example is based on a computer program which you can try out for yourself. It shows the action of a simple *bandpass* filter — that is, a filter which passes a particular band of frequencies relatively strongly. By the time you have finished reading this section, you will therefore have met examples of low-pass, bandstop, and bandpass filters.

Bandpass filters are useful in many signal processing applications. A good example is electronic communications. Many communications systems — for example, radio, television, or data channels — send information from place to place within a well-defined frequency band. Signals can therefore be separated from one another at the receiving end on the basis of their different frequencies. Furthermore, interference or noise picked up during transmission often occupies a wide frequency range, and can be reduced with a suitable bandpass filter.

The recursive filter we have chosen is defined by:

$$y[n] = 1.5\, y[n-1] - 0.85\, y[n-2] + x[n] \tag{1.4}$$

This particular algorithm is designed to transmit a sampled sinusoidal signal relatively strongly when each cycle (or *period*) of the sinusoid contains about ten sample values. If there are more, or less, samples per period the output signal is reduced in amplitude. Computer Program no.1 in Appendix A1 produces a graphical output of the form shown in Figure 1.5. At the top is a sinusoidal input signal $x[n]$ of constant amplitude but variable frequency. It starts off with about twenty samples per period, decreasing to about six samples per period on the right-hand side. As it 'sweeps through' the frequency corresponding to ten samples per period, the output signal $y[n]$ displays an amplitude peak. By altering the multiplier coefficients in equation (1.4) we could easily achieve a more (or less) pronounced bandpass effect.

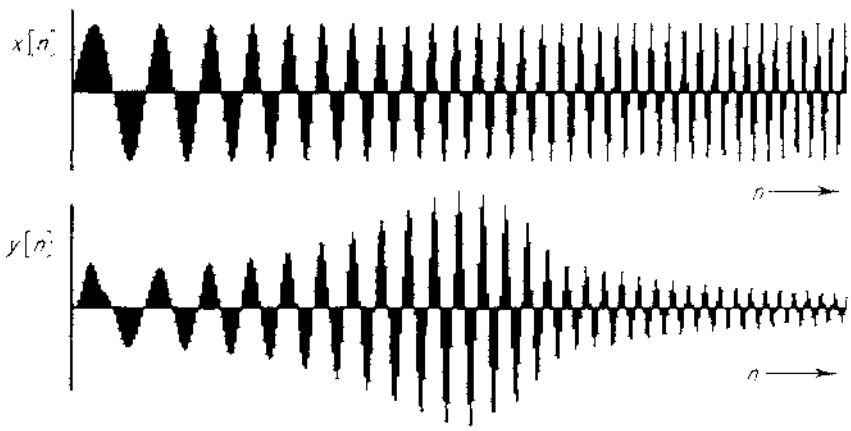

Figure 1.5 The action of a digital bandpass filter *(abscissa: 320 samples).*

If you refer to Appendix A1, you will see that Program no.1 has three main parts:

(a) It generates the 'swept-frequency' input signal. (You need not concern yourself with how this is done.)
(b) It uses an iterative loop to implement equation (1.4) over and over again for different values of n.
(c) It produces a graphical output of the type shown in Figure 1.5.

Part (b) is the most important from our point of view, since it shows just how easily a general-purpose computer can be programmed as a digital filter.

The four recurrence formulae discussed in this section should give you some idea of the possibilities offered by DSP. We hope that you are feeling sufficiently tantalized to want to tackle some theory! However, before we really start this in Chapter 2, we need to cover some further introductory aspects of digital signals and systems.

1.2 SAMPLING AND ANALOG-TO-DIGITAL CONVERSION

We have already mentioned the conversion from analog to digital signals, and back again, and have illustrated a DSP scheme in Figure 1.2. We should now look at the important processes of sampling and analog to digital conversion rather more carefully.

Suppose that an analog signal is to be represented by a set of equally-spaced samples. How often should it be sampled? An intuitive answer is suggested by Figure 1.6. Part (a) shows a signal sampled at a rate which is clearly too low to pick out the more rapid fluctutations. In part (b), however, sampling appears unnecessarily fast. A great many sample values would have to be stored, processed, or transmitted. Can we choose some intermediate sampling rate which is adequate, without being excessive?

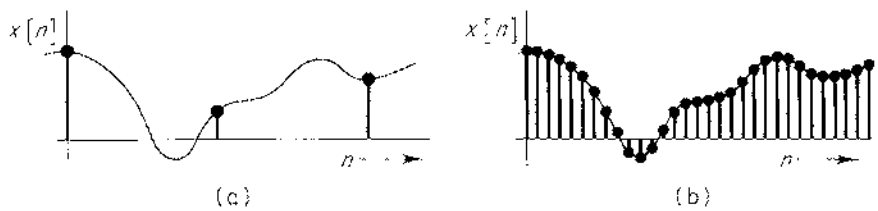

Figure 1.6 Sampling an analog signal.

Let us assume, for the moment, that the analog signal represents a speech waveform from a microphone. Then it will typically contain significant frequencies between about 300 Hz and 3 kHz — the range catered for by many telephone systems. Since the fastest fluctuations in the waveform correspond to the highest frequencies, it is clear that these frequencies will be lost if sampling is too slow. This deduction is confirmed by Shannon's famous *Sampling Theorem*, which may be stated as follows:

> An analog signal containing components up to some maximum frequency f_1 Hz may be completely represented by regularly-spaced samples, provided the sampling rate is at least $2f_1$ samples per second.

Thus a speech waveform having $f_1 = 3$ kHz should be sampled at least 6000 times per second. Note that this corresponds to *two samples per period* of the highest frequency present. To take another example, we might need to sample a television signal having a maximum frequency of 5 MHz (5×10^6 Hz). The Sampling Theorem tells us that it should be sampled at least 10 million times per second.

If we use the minimum rate specified by the theorem, then the sampling interval T is clearly:

$$T = \frac{1}{2f_1} \tag{1.5}$$

Alternatively, if we have a digital system with sampling interval T, the maximum analog frequency which it can represent is:

$$f_1 = \frac{1}{2T} \text{ Hz}, \quad \text{or} \quad \omega_1 = 2\pi f_1 = \frac{\pi}{T} \text{ radians/sec} \tag{1.6}$$

At first sight it may seem surprising that a set of instantaneous samples can ever give a *complete* representation of an analog signal. It implies that we could recover the original signal perfectly from its sample values if we wished. This is indeed the case, and means that sampling need not involve any loss of information. We now present, in simplified form, the reasoning behind this extremely important idea.

Figure 1.7(a) represents the frequency distribution, or *spectrum*, of an analog speech signal. Its precise shape is unimportant; what matters is that there are no significant components above some maximum frequency f_1 — here taken as 3 kHz. (We have drawn the spectral *magnitude* $|H(f)|$, ignoring the *phases* of the various components. Once again, this is adequate for the present discussion.) Note that we have shown the spectrum as an *even* function, extending to negative frequencies. This widely-used representation results from expressing each frequency component as the sum of two exponentials. Thus a component $A\cos(\omega t)$ of amplitude A and frequency ω radians per second may be written as:

$$A\cos(\omega t) = \frac{A}{2}\exp(j\omega t) + \frac{A}{2}\exp(j\{-\omega\}t) \tag{1.7}$$

and represented by *two* spectral components of amplitude $A/2$ at frequencies $\pm\omega$.

It may be shown that sampling the analog signal causes the original spectrum to repeat around multiples of the sampling frequency. This is illustrated in part (b) of the figure for a sampling frequency of 8000 samples per second (rather higher than required by the Sampling Theorem). The original spectrum repeats around 8 kHz, 16 kHz, and so on. We may think of the spectral repetitions, which in theory extend to infinitely high frequencies, as a consequence

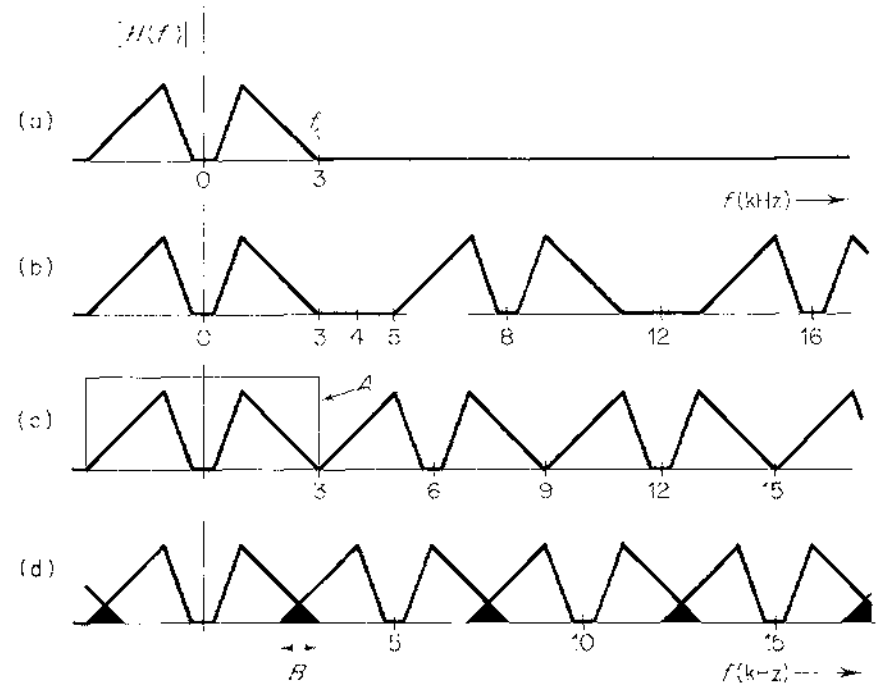

Figure 1.7 The effects of sampling on a signal spectrum.

of representing a smooth analog waveform by a set of narrow samples, or *impulses*.

In principle we could recover the spectrum shown in part (a) — and hence the original analog signal — with a low-pass filter. Such a *reconstituting filter* should transmit components up to 3 kHz, but reject all the spectral repetitions caused by sampling. Since the first of the repetitions in part (b) of the figure begins at 5 kHz, the filter should eliminate components at 5 kHz and above.

What happens if we reduce the sampling rate to the minimum value demanded by the Sampling Theorem — in this example, 6 kHz? Figure 1.7(c) shows the resulting spectrum. The repetitions now occur around 6 kHz, 12 kHz, and so on. In the region of 3 kHz they touch, but do not overlap, so it is still just theoretically possible to recover the original spectrum by low-pass filtering. The required filter characteristic is labeled 'A' in the figure. It must have an infinitely sharp cut off at 3 kHz, and is referred to as the 'ideal' reconstituting filter.

If we reduce the sampling frequency further — say to 5 kHz — we get overlap between successive spectral repetitions. The effect, known as *aliasing*, is illustrated in part (d) of the figure. The overlap in the region of 3 kHz (labeled 'B') corrupts the original spectrum, which cannot now be recovered. Information has been lost through inadequate sampling.

We have discussed the problem in relation to a speech signal with f_1 = 3 kHz. But the same basic ideas apply to a very wide variety of signals, data, and sampling rates.

We therefore see that the Sampling Theorem defines the minimum sampling

rate which avoids spectral overlap, or aliasing. In practice we choose a somewhat higher sampling rate — for example, speech signals having f_1 = 3 kHz are commonly sampled at 8 kHz. The main reason is that we cannot make an ideal reconstituting filter with an infinitely sharp cut-off. By sampling rather faster than the minimum rate we introduce a *guard band* between adjacent spectral repetitions (for example, the range 3–5 kHz in part (b) of the figure). The reconstituting filter can now have a more gentle cut-off. Also we have a small safety margin, in case the maximum frequency f_1 has been underestimated.

While on the subject of sampling, we should introduce a little more terminology. The maximum frequency f_1 contained in an analog signal is widely referred to as the *Nyquist frequency*. The minimum sampling rate ($2f_1$ samples per second) at which it can theoretically be recovered is known as the *Nyquist rate*. And the so-called *folding frequency*, which equals half the sampling frequency actually used, is the highest frequency which can be adequately represented according to the Sampling Theorem. When the theorem is just obeyed, the Nyquist and folding frequencies are equal.

Referring back to Figure 1.2, it is now clear why our general DSP scheme includes an analog filter on the input side. Known as an *anti-aliasing filter*, it has low-pass characteristics and is designed to ensure that the maximum frequency $f_1 = 1/2T$ is not exceeded.

Let us now consider the sampling process in a little more detail. In most electronic DSP applications, sampling is performed by an analog-to-digital converter (ADC), which also transforms the stream of samples into a binary code. It is simple to show that a binary code N bits long allows 2^N separate numbers, or signal values, to be represented. Thus if $N = 8$ we may encode $2^8 = 256$ discrete values; if $N = 16$ we may encode $2^{16} = 65\,536$ values, and so on.

Analog signals normally take on a continuous range of amplitudes, so when they are sampled and binary-coded small amplitude errors are bound to be introduced. This is illustrated by Figure 1.8. For simplicity, we have assumed

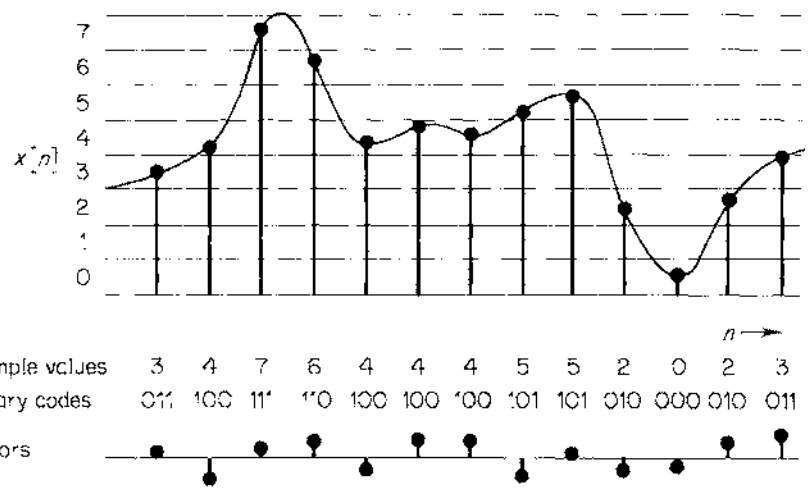

Figure 1.8 Converting an analog signal into a binary code.

a 3-bit code, which can represent just $2^3 = 8$ separate sample values. The total amplitude range is therefore divided into eight *quantization levels* or 'slots', and each sample value is allocated the appropriate binary code. The sequence of coded values now represents the original signal, and may be processed digitally.

The maximum error introduced into each sample value by this process is plus or minus half a quantization level. With a 3-bit code this equals $\pm\frac{1}{16}$ of the total available amplitude range. In most practical applications such large errors would be unacceptable, and a longer binary code would be used. Note that the magnitudes of successive errors, also shown in the figure, are often more or less random and independent of one another. We can therefore regard them as unwanted *quantization noise*, introduced by the coding process. It can not subsequently be removed.

Note that we are assuming the quantization to be uniform: that is, the various quantization levels are assumed equally spaced. Nonuniform quantization is sometimes encountered — for example, in coding schemes for digital communication systems, and in the roundoff of floating point numbers. However it is more difficult to describe and visualize, and is not covered in this book.

Quantization also occurs when a signal is coded 'manually' rather than automatically. For example, you may wish to record some midday temperature values, and enter them into a computer via a keyboard. You make a decision about the accuracy of the data, and quantize the numerical values accordingly. Thus a true temperature value of 18.27 °C (if you can read it that accurately!) might be entered as 18.3 °C, or just as 18 °C — and so on. Once again, the sample values have been quantized, and some quantization noise has been introduced. Of course, in this case the samples are converted into a binary code inside the computer. And if you attempt to enter an extremely accurate value, say 18.2 737 948 65, the amount of quantization will probably be determined by the binary wordlength of the computer itself.

Analog-to-digital conversion therefore always degrades a signal to some extent. This may appear to be a serious drawback of digital processing. However we must remember that if the quantization errors are too large, then the number of bits in the binary code can be increased. Secondly, an analog signal always has some noise or uncertainty associated with it anyway, so a small amount of quantization noise need not cause significant degradation.

Apart from input quantization effects, DSP algorithms generally introduce a certain amount of computational error. Essentially this is because a finite wordlength is used to represent signal values, coefficients, and numerical results within the machine. For example the bandstop filter previously defined by equation (1.3) involves coefficients specified to 4 decimal places. The operation of the filter will obviously be affected if the sample values and coefficients are imperfectly represented, or if the results of computation are rounded off. The analysis of such effects is complicated. Fortunately they are hardly ever a serious handicap when processing signals on a general-purpose computer, and we do not consider them further in this book. However, if you get involved in fast DSP techniques implemented with special-purpose hardware, you will probably meet them again!

After a signal has been processed digitally, it may be appropriate to convert back to an analog voltage using a digital-to-analog converter (DAC). A DAC

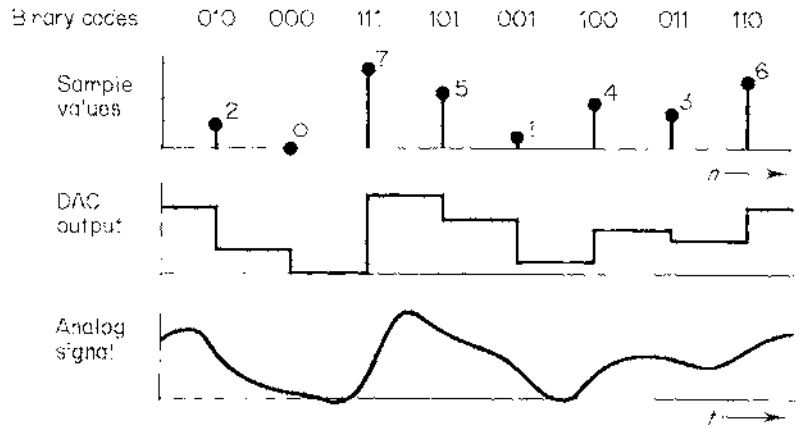

Figure 1.9 Digital-to-analog conversion.

usually produces an output of the *zero-order-hold* type, also known as *sample-and-hold*. This is illustrated in Figure 1.9. Each signal sample value, corresponding to the binary code delivered to the DAC input, is 'held' during the following sampling interval. The resulting staircase waveform is adequate for many practical applications (including the computer plots illustrated in this book). If a 'true' analog output is required, a further smoothing filter must be employed. The fully reconstituted signal is then as shown at the bottom of the figure.

1.3 BASIC TYPES OF DIGITAL SIGNAL

Our next task is to describe some basic types of digital signal. You may feel that these signals, with their rather simple waveforms and straightforward mathematical descriptions, are of limited value. After all, Figures 1.3 and 1.4 have shown examples of real-life signals which are very complicated in form and structure, so we begin by summarizing the main reasons why basic signals are so valuable:

Complicated, real-life, signals may generally be considered as the summation of a number of simpler, basic, signals.

Many useful DSP algorithms are *linear*. The response of a linear algorithm, or processor, to a number of signals applied simultaneously equals the summation of its responses to each signal applied separately. Thus if we can define the response of a linear processor to basic signals, we can predict its response to more complicated ones.

Basic signals are simple to describe and generate, and are widely used as *test signals* for investigating the properties of linear processors and systems.

We might note in passing that this book concentrates exclusively on linear DSP techniques. Nonlinear ones are certainly important, and have found many practical applications. But their theoretical treatment is much more difficult, and cannot easily be covered in an introductory text.

1.3.1 STEPS, IMPULSES, AND RAMPS

Digital step and impulse functions are among the most important of all basic signals. The ramp function is rather less so, but since it is closely related to the other two, we will also describe it in this section.

The *unit step function* $u[n]$ is defined as:

$$\begin{aligned} u[n] &= 0, \quad n<0 \\ u[n] &= 1 \quad n\geq 0 \end{aligned} \tag{1.8}$$

It is shown in Figure 1.10(a). This signal plays a valuable role in the analysis and testing of digital signals and processors.

A further basic signal, which is probably even more important than the unit step, is the *unit impulse function* $\delta[n]$. It is shown in part (b) of the figure, and is defined as:

$$\begin{aligned} \delta[n] &= 0, \quad n\neq 0 \\ \delta[n] &= 1, \quad n=0 \end{aligned} \tag{1.9}$$

Also known as the *unit sample*, $\delta[n]$ consists of an isolated unit-valued sample at $n=0$, surrounded on both sides by zeros. We shall see later that this signal is of the greatest value in DSP theory and practice.

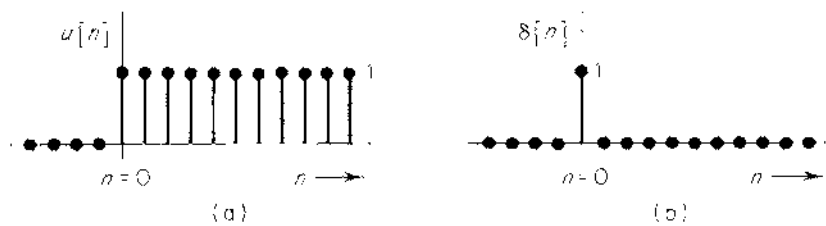

Figure 1.10 Basic digital signals: (a) the unit step function, and (b) the unit impulse function.

$u[n]$ and $\delta[n]$ are closely related to one another. $u[n]$ is said to be the *running sum* of $\delta[n]$. This is simple to visualize. If we start at the left-hand side of Figure 1.10(b), and move to the right summing the sample values of $\delta[n]$ as we go, we generate the unit step function $u[n]$. The relationship may be written formally as:

$$u[n] = \sum_{m=-\infty}^{n} \delta[m] \tag{1.10}$$

Conversely, we can easily generate $\delta[n]$ from $u[n]$. The recurrence formula:

$$\delta[n] = u[n] - u[n-1] \tag{1.11}$$

holds good for all integer values of n. Therefore $\delta[n]$ is referred to as the *first-order difference* of $u[n]$.

Ramp functions also deserve a mention here. A digital ramp signal rises, or falls, linearly with the variable n. It is sometimes used as a test signal, or for approximating a practical signal by a set of straight line sections. The *unit*

ramp function $r[n]$ is defined as:

$$r[n] = n\,u[n] \tag{1.12}$$

Since $u[n]$ is zero for $n<0$, so also is the ramp function. Note that $r[n]$ is a running sum of $u[n]$; alternatively, $u[n]$ is a first-order difference of $r[n]$. The unit-ramp signal is shown in Figure 1.11.

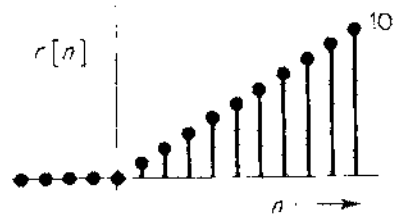

Figure 1.11 The unit ramp function.

A number of worked examples are included in this book, to help you consolidate your understanding of the main text. In the first of these, we look at the effects of scaling, shifting, and time-reversing steps, impulses, and ramps; and we also show how other simple digital signals may be built up by summation of such basic functions.

Example 1.1 **Find expressions for the various signals shown in Figure 1.12.**

Solution

(a) This is a unit step function which has been scaled (weighted) by a factor of -2; it starts at $n=-4$, rather than $n=0$; and it is time-reversed.
Now:
Scaling by -2 gives the function $-2u[n]$.
Time-shifting so that it starts at $n=4$ gives the function $-2u[n=-4]$.
Reversal gives the function $-2u[-n-4]$. Hence the required function is:

$$x[n] = -2u[-n-4]$$

(b) This digital 'rectangular pulse' may be considered as the summation, or superposition, of a unit step starting at $n=-3$, and an equal but opposite unit step starting at $n=5$. Hence:

$$x[n] = u[n+3] - u[n-5]$$

(c) The signal is a weighted unit impulse of value 8, time-shifted to occur at $n=6$. Hence:

$$x[n] = 8\delta[n-6]$$

(d) This signal may be considered as the superposition of two ramp functions. One starts at $n=-6$ and has a slope of 2. Its upward trend

may be stopped by adding another ramp starting at $n = -2$, with a slope of -2. Hence:

$$x[n] = 2r[n+6] - 2r[n+2]$$

Note: The answers we have given are not the only possible ones.

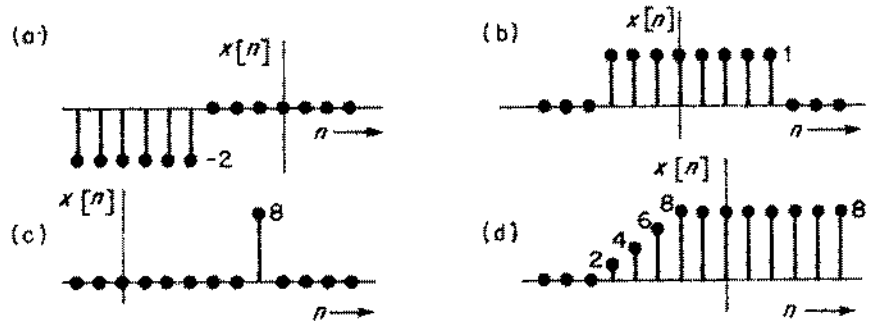

Figure 1.12

You have now met the basic ideas and mathematical notation of steps, impulses, and ramps. Some further properties of these signals — particularly of impulses — will be developed in later chapters.

1.3.2 EXPONENTIALS, SINES, AND COSINES

Exponential signals — and their close relations, sine and cosine waves — are extremely important in many branches of engineering and applied science. They are widely used in analysis, and often occur in the natural world and in technology. In the DSP context, sampled exponentials are central to a powerful set of *frequency-domain techniques* used to analyze and process signals, and to design systems. These form the main material for Chapters 3 and 4. It is therefore important to become familiar with such signals. Those of you who have previously worked with their analog (continuous-time) counterparts will notice many similarities, and a few differences.

We start by considering a function of the form:

$$x[n] = A \exp(\beta n) \tag{1.13}$$

where A and β are constants, and exp denotes the exponential function. Although we restrict ourselves to real values of A, we will allow β to be real, imaginary, or complex.

Suppose that β is real. Figure 1.13 shows the forms of signal when it is negative, zero, and positive. When $\beta < 0$, we get a *decaying exponential*. In theory it falls towards zero as n increases, and rises without limit for negative n. (We have, of course, shown only a few sample values of each function.) When $\beta = 0$, all sample values are equal. The signal is just a sampled 'steady level', often referred to in electronic engineering as a *DC level*. If $\beta > 0$, we get a *rising exponential* which grows without limit as n increases. In all three cases, the sample value at $n = 0$ equals the constant A.

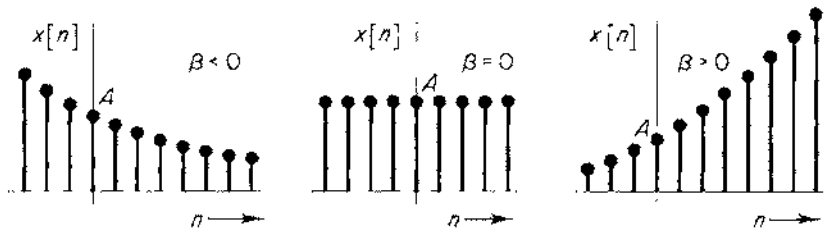

Figure 1.13 Basic digital signals: real exponentials.

Such real exponentials are theoretically *eternal* — they continue forever in both directions. In practice, however, we often work with signals which are zero prior to some reference instant, normally taken as $n=0$. An exponential of this type may be written as:

$$\begin{aligned} x[n] &= A \exp(\beta n), \qquad n \geq 0 \\ &= 0, \qquad\qquad n < 0 \end{aligned} \tag{1.14}$$

Alternatively, we can make use of the fact that the unit step function $u[n]$ is zero for $n<0$, and unity for $n \geq 0$. Therefore if we multiply, or *modulate*, an eternal exponential by $u[n]$ we specify the function:

$$x[n] = A \exp(\beta n)\, u[n] \tag{1.15}$$

This signal is effectively 'switched on' at $n=0$ by the unit step function. Equations (1.14) and (1.15) are equivalent.

Another interesting property of sampled real exponentials is that successive sample values form a simple geometric progression — each value equals that of its neighbor, multiplied by a constant. We may show this by rewriting equation (1.13) as:

$$x[n] = A \exp(\beta n) = AB^n, \qquad \text{where } B = \exp(\beta)$$

Hence

$$x[n+1] = AB^{n+1} = Bx[n] \tag{1.16}$$

If we next allow the constant β in equation (1.13) to be purely imaginary, we can generate sine and cosine signals. Suppose $\beta = j\Omega$, where $j = \sqrt{-1}$ and Ω is a real constant. The resulting signal $x_1[n]$ is given by:

$$x_1[n] = A \exp(jn\Omega) = A \cos(n\Omega) + jA \sin(n\Omega) \tag{1.17}$$

It contains a cosinusoidal real part and a sinusoidal imaginary part. This is a little awkward, since (in this book at least!) we deal mainly with real signals. Note, however, that if we put $\beta = -j\Omega$, we define another signal:

$$x_2[n] = A \exp(-jn\Omega) = A \cos(n\Omega) - jA \sin(n\Omega) \tag{1.18}$$

Now cosines are *even* functions, thus $\cos(-n\Omega) = \cos(n\Omega)$; and sines are *odd* functions, thus $\sin(-n\Omega) = -\sin(n\Omega)$. Hence by adding $x_1[n]$ and $x_2[n]$ together we form the signal:

$$x_1[n] + x_2[n] = 2A \cos(n\Omega) \tag{1.19}$$

giving:

$$A\cos(n\Omega) = \tfrac{1}{2}\{x_1[n] + x_2[n]\} = \frac{A}{2}\exp(jn\Omega) + \frac{A}{2}\exp(-jn\Omega) \tag{1.20}$$

This result shows that a sampled cosine signal can be made up from a *pair* of sampled imaginary exponentials.

In a similar way we may write:

$$x_1[n] - x_2[n] = 2jA\sin(n\Omega)$$

giving:

$$A\sin(n\Omega) = \frac{A}{2j}\exp(jn\Omega) - \frac{A}{2j}\exp(-jn\Omega) \tag{1.21}$$

Therefore a real, sampled, sine signal may also be made up from a pair of imaginary exponentials. We now see why exponentials, sines, and cosines are so closely related.

What are the differences between sampled sines and cosines, and their analog counterparts? You probably know that analog sine and cosine signals are oscillatory, and *periodic*. That is to say, they repeat indefinitely along the time axis. For example, the signal $A\sin(\omega t)$ is wave-like, and repeats every $2\pi/\omega$ seconds. Its peak value, or amplitude, is A; and its frequency is ω radians per second, or $\omega/2\pi$ Hz.

Sampled sines and cosines, however, are not necessarily periodic. Although their sample values lie on a periodic *envelope*, the numerical values may not form a repetitive sequence. We illustrate this in Figure 1.14. Part (a) shows a discrete-time sinusoid, and part (b) a cosinusoid, both of which *are* periodic. But part (c), although its samples lie along a sinusoidal envelope, does not have repeating numerical values. Indeed, it is hard to tell merely by inspection whether it is a sampled sinusoid or not.

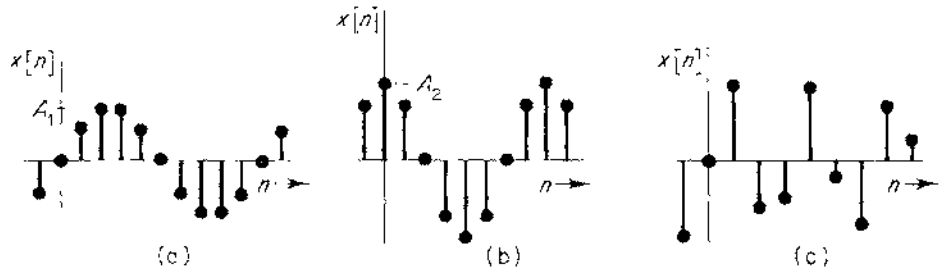

Figure 1.14 Basic digital signals: sines and cosines.

Intuitively, it is clear that exact repetition will only occur if the sampling interval bears some simple relationship to the repetition time, or period, of the underlying analog waveform. We may quantify this idea by returning to equation (1.17). For the moment let us assume that $x[n]$ *is* truly periodic, and repeats every N sample values. This means that:

$$\begin{aligned} x[n] &= A\exp(jn\Omega) = A\exp(j\{n+N\}\Omega) \\ &= A\exp(jn\Omega)\exp(jN\Omega) \end{aligned} \tag{1.22}$$

and hence that:

$$\exp(jN\Omega) = 1 \tag{1.23}$$

The last equation only holds good if $N\Omega$ is a multiple of 2π. In other words there must be an integer m such that:

$$N\Omega = 2\pi m, \quad \text{or} \quad \frac{\Omega}{2\pi} = \frac{m}{N} \tag{1.24}$$

Hence $x[n]$ is only periodic if $\Omega/2\pi$ is a rational number (the ratio of two integers). Otherwise its sample values do not repeat. We have derived this result for an imaginary exponential, but it applies equally well to sines and cosines.

A second difference between analog and digital sinusoidal signals concerns the question of time and frequency scales. When we write a digital signal as $x[n] = A \sin(n\Omega)$, n is assumed to be a dimensionless integer variable. Therefore the constant Ω must be measured in radians rather than radians per second. Strictly speaking it is not a frequency — although we generally describe it as such. However our formulation for $x[n]$ allows us to define instead the number of *samples in each cycle, or period,* of the sinusoid. Since one complete period corresponds to:

$$n\Omega = 2\pi, \quad \text{or} \quad n = 2\pi/\Omega \tag{1.25}$$

there must be $2\pi/\Omega$ samples per period, regardless of the time or frequency scales of the signal. For example, referring to Figure 1.14(a) we see that the sinusoid has exactly ten samples per period. Therefore $2\pi/\Omega = 10$, or $\Omega = \pi/5$. The signal is given by $x[n] = A_1 \sin(n\pi/5)$. Similarly, the cosinusoid in part (b) of the figure has eight samples per cycle, and is given by $x[n] = A_2 \cos(n\pi/4)$.

In some cases, however, we really do need to work in terms of time and frequency. At the beginning of section 1.2 we introduced the Sampling Theorem, which states that an analog signal with frequency components up to some maximum frequency f_1 Hz may be represented by samples taken at a rate of at least $2f_1$ samples per second. Clearly, when considering this theorem we *are* interested in time and frequency scales. A good way of introducing them is to work in terms of the sampling interval T. The sampling instants are given by:

$$t = nT, \qquad n = \ldots -2, -1, 0, 1, 2, 3 \ldots \tag{1.26}$$

and the sampling frequency is:

$$f_s = \frac{1}{T} \tag{1.27}$$

We now write our digital signal as:

$$x[n] \quad \sin(n\Omega) \quad \sin(n\omega T) \tag{1.28}$$

Since nT represents time in seconds, ω must be an angular frequency in radians per second. If f is the corresponding frequency in hertz, then $\omega \quad 2\pi f$ and equation (1.28) may be written as:

$$x[n] = \sin\left(\frac{n2\pi f}{f_s}\right) \tag{1.29}$$

We can now generate the sample series $x[n]$ if we know the frequency f of the underlying analog sinusoid, and the sampling frequency f_s. Our sampled signal has been redefined in terms of frequency in hertz, and time in seconds.

It may be helpful to mention the practical importance of sines and cosines once again. Such signals arise widely in engineering and science, and in the natural world: the mains electricity supply; the signal generators used in electronic laboratories; oscillations in electronic circuits containing capacitance and inductance; the vibrations of tuning forks and mechanical structures; and various types of wave motion. Many of these signals can be stored and processed in digital computers. But from our point of view, it is also very important that other, *completely different*, signals may be considered as the summation of a number of sines and cosines of appropriate amplitudes and frequencies. Seen in this light, sines and cosines can even be useful for DSP applications in economics and the social sciences!

We end our survey of exponential and sinusoidal signals by letting the constant β in equation (1.13) become complex. You can probably imagine what will happen. We have seen that *real* values of β give signals which rise or fall exponentially. *Imaginary* values produce sines and cosines. Therefore *complex* values may be expected to produce oscillatory signals with rising or falling amplitudes.

Consider the signal:

$$x[n] = A \exp(\beta n) = A \exp(\{\beta_0 + j\Omega\}n) \tag{1.30}$$

where β_0 is a real part, and $j\Omega$ the imaginary part, of β. Hence:

$$x[n] = A \exp(\beta_0 n) \exp(j\Omega n) \tag{1.31}$$

The frequency term $\exp(j\Omega n)$ is multiplied, or modulated, by the rising (or falling) amplitude term $A \exp(\beta_0 n)$. Of course, we again have the difficulty that $\exp(j\Omega n)$ is not a real function of n. We therefore need to consider a pair of imaginary exponentials which produce a real sine or cosine. The argument is just as before, leading to functions of the form:

$$x[n] = A \exp(\beta_0 n) \sin(n\Omega)$$

or

$$x[n] = A \exp(\beta_0 n) \cos(n\Omega) \tag{1.32}$$

Typical examples are given in parts (c) and (d) of the worked example below.

These signals are also important in practice. Sinusoidal oscillations with decaying amplitudes are often observed in circuits and filters — both analog and digital. They also arise in vibrating mechanical systems. Exponentially increasing oscillations tend to occur in *unstable* systems — including digital processors which have been badly designed! We shall have more to say about this later.

If you require further familiarization with sampled exponentials and sinusoids, you may like to try Computer Program no.2 in Appendix A1. This

simple program draws signals of the general form $x[n] = \exp(\beta_0 n] \cos(n\Omega)$ on the screen, allowing you to control β_0 and Ω. Note that if $\Omega=0$, $x[n]$ is a real exponential. If $\beta_0=0$, it is a cosine. Whereas if neither is zero, $x[n]$ is an exponentially rising (or falling) cosine. (Suitable values to start with are β_0 0.01 and $\Omega=0.2$.)

Example 1.2 **Sketch carefully, and label, the following signals:**

(a) $x[n] = \exp(0.2n)$

(b) $x[n] = \cos\left(\frac{\pi n}{4}\right)$

(c) $x[n] = \exp\left(\frac{n}{15}\right) \sin\left(\frac{\pi n}{6}\right)$

(d) $x[n] = \exp\left(\frac{-n}{5}\right) \cos(n]\, u[n]$

Solution: **The signals are shown in Figure 1.15, and should be self-explanatory.**

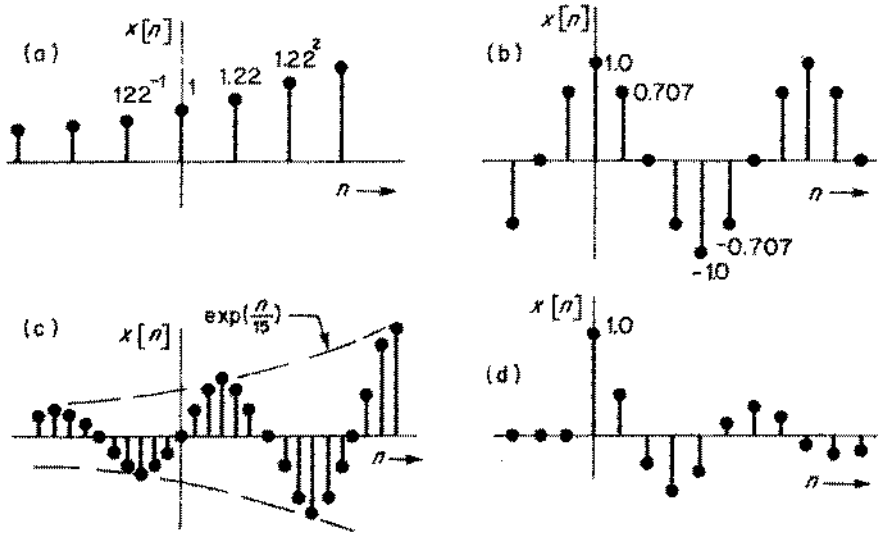

Figure 1.15

1.4 AMBIGUITY IN DIGITAL SIGNALS

A digital signal $x[n]$ generally represents some underlying analog function. However, if we start with a given set of sample values, it is clear that a huge

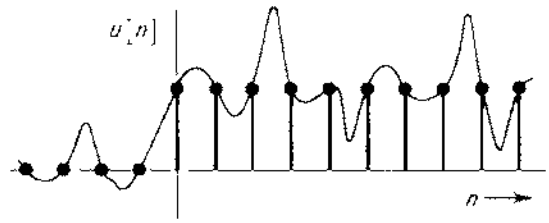

Figure 1.16 A unit step function, and one of the many analog signals which can be drawn through its sample points.

variety of analog signals *could* be drawn through them. Consider, for example, the unit step function $u[n]$. Although we may think of it as being the sampled equivalent of an analog step function, Figure 1.16 shows another analog waveform with the same sample values. It is clear that digital signals are, in an important sense, ambiguous.

You may feel that the above example is rather exaggerated, and that a sampled signal should have essentially the same 'shape' as its analog counterpart. However, we must remember that many sampled functions – including random noise — simply do not have an obvious underlying shape. Nor can we necessarily expect the systems and computers which handle them to interpret our intentions correctly.

The difficulty is made worse when we realize that sampled sinusoids also involve ambiguity — even though we restrict ouselves to a particular shape of signal. Figure 1.17 shows two of the many analog sinusoidal waves which fit the given set of sample values. Here, then, is a further difference between digital sinusoids and their analog counterparts.

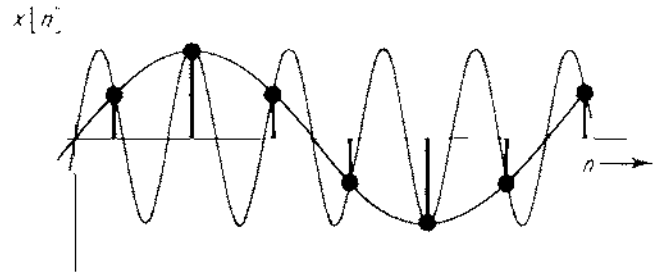

Figure 1.17 Ambiguity in digital sinusoids.

We can clarify the situation by considering the complex exponential signal:

$$x[n] = \exp(j\{\Omega + 2\pi\}n) \tag{1.33}$$

This may be written as:

$$x[n] = \exp(j\Omega n)\exp(j2\pi n) = \exp(j\Omega n) \tag{1.34}$$

Therefore a sampled exponential of frequency Ω is *identical* to those of frequency $\Omega \pm 2\pi$, $\Omega \pm 4\pi$, and so on. Although we have shown this for exponentials, the same is true for sines and cosines. As we increase the frequency of a digital sinusoid from zero, its rate of oscillation initially *increases*. Beyond $\Omega = \pi$ it

starts to *decrease* again. At $\Omega = 2\pi$, the rate of oscillation is again zero · · and so on, indefinitely. Ambiguity arises because a digital sinusoid can represent any one of an infinite set of underlying frequencies, separated from one another by 2π.

Program no.3 in Appendix A1 illustrates this important point. It generates and plots a digital sinusoid whose frequency Ω increases in ten equal steps across the screen. (For convenience we use a cosine rather than a sine.) The signal is given by:

$$x[n] = \cos(n\Omega) = \cos\left(\frac{n2\pi m}{8}\right) \tag{1.35}$$

with the integer m increasing from 0 to 9 inclusive.

The plot is reproduced in Figure 1.18. We have also marked the values of Ω against it. On the left-hand side $\Omega = 0$, and on the right $\Omega = 2.25\pi$. As expected, the rate of oscillation increases as far as $\Omega = \pi$, then decreases again. At $\Omega = 2\pi$, the signal is the same as at $\Omega = 0$. At $\Omega = 2.25\pi$ it is the same as at 0.25π, and so on. You may think of the underlying analog frequency as continuously increasing from left to right.

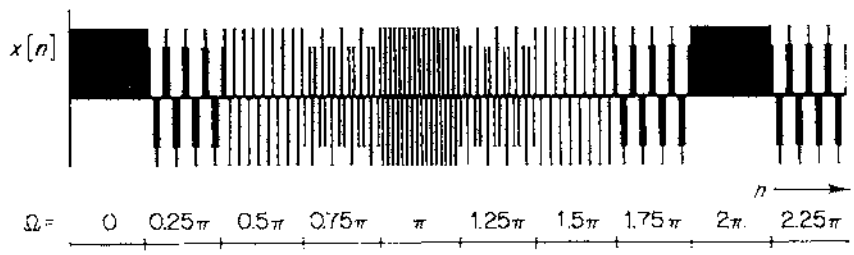

Figure 1.18 A digital cosinusoid whose frequency is increased in ten equal increments *(abscissa: 370 samples).*

The ambiguity problem is closely related to our discussion of the Sampling Theorem and aliasing in Section 1.2. To obey the theorem we must make the sampling frequency at least twice the highest frequency present in the underlying analog waveform ($f_s \geq 2f_1$). This is equivalent to taking at least two samples per period of the highest frequency. Now two samples per period corresponds to $\Omega = \pi$ (see equation (1.25)). Hence there is no ambiguity over which frequency is intended within the range $\Omega = 0$ to $\Omega = \pi$. It is only when we go outside this range — as in Figure 1.18 · · that we get into difficulty. In effect the Sampling Theorem resolves ambiguity by restricting the allowable frequency range of the underlying analog signal.

1.5 DIGITAL PROCESSORS

It is now time to consider some basic aspects of digital signal processors. By 'processor' we mean any system which carries out a DSP function. It may be a general-purpose computer programmed for a particular DSP task, a

microprocessor, dedicated digital hardware – or a combination of these. Since this book's emphasis is on the use of general-purpose computers, we will generally think of a processor as a computer plus relevant software. Incidentally, we make no particular distinction between the terms 'processor' and 'system', using them interchangeably.

1.5.1 LINEAR TIME-INVARIANT (LTI) SYSTEMS

This book covers only *linear* DSP. Nonlinear processing is certainly important, but its theoretical treatment is much more difficult. Fortunately, linear processing includes a wide range of DSP algorithms of great practical interest.

A linear system, or processor, may be defined as one which obeys the *Principle of Superposition*. The principle may be stated as follows:

> If an input consisting of the sum of a number of signals is applied to a linear system, then the output is the sum, or *superposition*, of the system's responses to each signal considered separately.

Let us suppose than an input $x_1[n]$, applied to a digital processor, produces the output $y_1[n]$; and that input $x_2[n]$ produces $y_2[n]$. Then the processor is linear if its response to $\{x_1[n] + x_2[n]\}$ is $\{y_1[n] + y_2[n]\}$. Furthermore, linearity implies that the response to an input $ax_1[n]$ is $ay_1[n]$, where a is a constant coefficient, or multiplier (also called a *weighting factor*). To generalize, the weighted sum of inputs:

$$ax_1[n] + bx_2[n] + cx_3[n] + \ldots \tag{1.36}$$

must produce the corresponding weighted sum of outputs:

$$ay_1[n] + by_2[n] + cy_3[n] + \ldots \tag{1.37}$$

You may feel that the property of linearity appears obvious, or even trivial. So we should point out straightaway that many quite simple processes are *not* linear. For example, suppose we have a system which *squares* each sample value applied to its input. Thus for two different inputs applied separately, we have:

$$y_1[n] = (x_1[n])^2 \quad \text{and} \quad y_2[n] = (x_2[n])^2 \tag{1.38}$$

When the same two inputs are summed, and applied simultaneously, the output is:

$$\begin{aligned} y_3[n] &= (x_1[n] + x_2[n])^2 \\ &= (x_1[n])^2 + (x_2[n])^2 + 2x_1[n]x_2[n] \end{aligned} \tag{1.39}$$

This is clearly *not* the sum of its responses to $x_1[n]$ and $x_2[n]$ applied separately.

A major property of linear systems, which is closely related to the Principle of Superposition, is known as *frequency-preservation*. It means that if we apply an input signal containing certain frequencies to a linear system, the output can contain only the same frequencies, and no others. The property depends upon the fact that a sampled sinusoid, applied to any linear processor, always produces a similar form of output, *at the same frequency as the input*. If we apply

a complicated signal containing many sinusoidal frequencies, the Principle of Superposition tells us that the output must be the sum of the outputs due to each input frequency, considered separately. The output therefore contains only those frequencies present in the input.

We can easily demonstrate that this property does not hold for the squaring system mentioned above. Suppose a signal $\sin(n\Omega)$ is applied to its input. Its output is therefore:

$$\begin{aligned} y[n] &= (x[n])^2 = \sin^2(n\Omega) \\ &= \tfrac{1}{2}\{1 - \cos(2n\Omega)\} = \tfrac{1}{2} + \tfrac{1}{2}\cos(n2\Omega) \end{aligned} \tag{1.40}$$

There are two separate, additive, frequency components in the output: one at zero frequency, of amplitude $\frac{1}{2}$, the other at frequency 2Ω, also of amplitude $\frac{1}{2}$. There is no component in the output at the frequency Ω.

We next describe *time-invariance*. A time-invariant system is one whose properties do not vary with time. The only effect of a time-shift in an input signal to the system is a corresponding time-shift in its output. The majority of technological systems and processes are of this type — including the DSP algorithms we cover in this book. Thus we concern ourselves exclusively with *linear time-invariant (LTI)* systems.

We can illustrate the above discussion with a simple example. Consider a cash register, of the type used at the check-out of a supermarket. It is a digital processor, which adds up the prices of individual items (the input signal) to produce a total (the output signal). We may think of it as a form of *digital integrator*, or *accumulator*, which estimates the running sum:

$$y[n] = x[n] + x[n-1] + x[n-2] + \ldots \tag{1.41}$$

Of course, no sensible cash register works nonrecursively — repeating the complete calculation every time a new item is included! It keeps a running total, which is updated. This is done using the equivalent recursive algorithm:

$$y[n] = y[n-1] + x[n] \tag{1.42}$$

when a customer checks out, the cash register is reset to zero, and the summation starts again.

It is fairly obvious that such a device is an LTI system. Its linearity may be deduced as follows. Suppose one customer checks out and the cash register estimates a total payment of y_1. The next customer's total is y_2. Now suppose the two customers are friends, and they decide to pool their goods before checking out. The grand total will be $(y_1 + y_2)$, which is the superposition of the two previous totals calculated separately. The cash register is also time-invariant, because its performance is (we hope!) unaffected by the time of day.

The cash register clearly involves storage/delay and addition of numerical values. More generally, digital LTI processors involve the following types of operation on input and/or output samples.

storage/delay
addition/subtraction
multiplication by constants

As another example, consider the digital notch filter previously illustrated in Figure 1.4. Equation (1.3) gave its recurrence formula, or difference equation, as:

$$y[n] = 1.8523\, y[n-1] - 0.94833\, y[n-2] + x[n] - 1.9021\, x[n-1] + x[n-2] \tag{1.43}$$

Although we are not yet in a position to explain how the filter works, the equation shows that it requires only the above types of operation. We can safely conclude that the filter is a LTI processor.

This is a convenient moment to show how digital LTI systems can be represented in block diagram form. Figure 1.19(a) illustrates the nonrecursive version of the cash register equation — see equation (1.41). Input samples are stored and delayed successively by an amount equal to the sampling interval T. This produces the values $x[n-1]$, $x[n-2]$, $x[n-3]$ ... Tapping points lead to an adder unit, whose output is $y[n]$. The figure implies that input samples are equally spaced in time. Although this is unlikely to be true of a cash register, regular sampling is used in most types of digital processor.

Part (b) of the figure shows the recursive version of the cash register equation — see equation (1.42). The current output $y[n]$ is *fed back* via a single delay unit T to produce the previous output $y[n-1]$, and this is updated by the current input $x[n]$ in an adder unit.

In part (c) we show a block diagram for the digital notch filter defined by

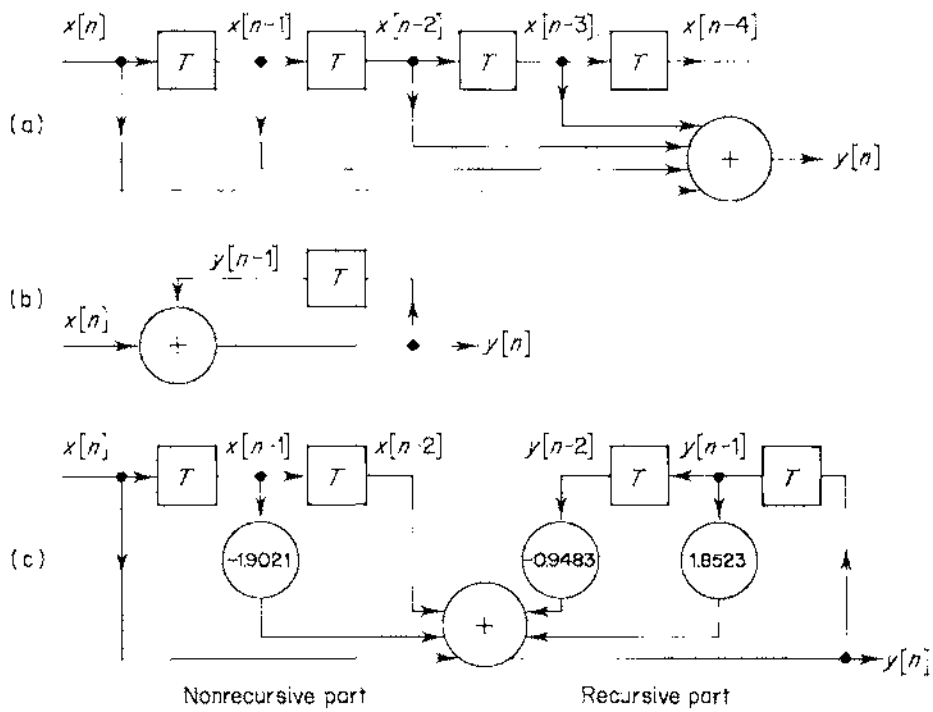

Figure 1.19 System block diagrams of digital signal processors.

equation (1.43). The current output $y[n]$ depends on three input sample values, weighted and added in the nonrecursive part of the filter; and on two previous outputs, provided by the recursive part. As before, the recursive part involves feedback.

When a general-purpose computer is used to implement a DSP algorithm, it *behaves as if* it were such a set of digital building blocks — delay/storage units, multipliers, and adders. This is achieved by the software. On the other hand when a DSP system is realized in special-purpose hardware, its architecture is likely to resemble such a block diagram much more closely. We should, however, bear in mind that sample values are represented in binary form, and that each block must process multi-bit codes.

You should by now have a good idea of what a digital LTI system is, and how it may be represented - - either as a recurrence formula, or as a block diagram. In the final part of this chapter we cover some additional ideas about digital processors, and introduce some more terminology.

1.5.2 OTHER SYSTEM PROPERTIES

We have already mentioned that LTI systems obey the Principle of Superposition, and have the property of frequency preservation. They possess other important properties, including *association* and *commutation*. The associative property means that we may analyze a complicated LTI system by breaking it down into a number of simpler subsystems. Alternatively, we can synthesize an overall system - perhaps a very complicated one - - by designing a number of independent subsystems. We know that, when we put them all together, overall performance can be predicted by straightforward rules of association. These rules will be explained in later chapters.

The commutative property of LTI systems means that if subsystems are arranged in series, or *cascade*, then they may be rearranged in any order without affecting overall performance. This can sometimes offer advantages — for example, it may reduce complexity.

Although the associative and commutative properties of LTI systems may appear rather obvious, they are not generally shared by nonlinear systems.

There are four other properties of systems which are often mentioned in the DSP literature. They are: *causality; stability; invertibility;* and *memory*. A processor may or may not possess them, independently of whether it is LTI or not. We now briefly summarize each in turn.

In a *causal* system, the output signal depends only on present and/or previous values of the input. You might assume that practical signal processors are always causal, because they cannot anticipate the future! However, if we record a signal or data and subsequently process it by computer, the software need not be causal. Of course it is reasonable to object that the complete recording/processing system *is* causal. But at this point we may become entangled in semantics!

A *stable* system is one which produces a finite, or *bounded*, output in response to a bounded input. This means that if the system is disturbed from its resting state by an input signal which does not grow without limit, or *diverge*, then

the output must not diverge either. In Section 1.3.2 we considered some exponential signals which grow without limit (Figures 1.13 and 1.15). Clearly, if a system produced such a signal in response to a bounded input (such as an impulse), it would be unstable. In practice the output of an unstable system must sooner or later cease to diverge, due to some *limiting effect* (such as numerical 'overflow' in a computer). We shall see later that instability is closely associated with the idea of feedback.

We now discuss *invertibility*. If a digital processor with input $x[n]$ gives an output $y[n]$, then its *inverse* would produce $x[n]$ if fed with $y[n]$. Most practical systems are invertible. An exception in the 'squaring device' mentioned earlier, for which $y[n] = (x[n])^2$. Knowledge of an output value $y[n]$ does not allow us to find $x[n]$ unambiguously, since it could be positive or negative.

A processor possesses *memory* if its present output $y[n]$ depends upon one or more previous input values $x[n-1]$, $x[n-2]$ ... In other words it must contain storage/delay elements, such as those shown in Figure 1.19. Interesting DSP systems invariably possess memory.

We can summarize the above discussion by saying that the main interest of this book is in LTI systems which are also causal, stable, invertible, and possess memory.

Example 1.3 $x[n]$ and $y[n]$ are the input and output signals of a DSP system. Determine which of the following properties are possessed by systems defined by the recurrence formulae (a) to (d) below:

linearity	time-invariance
causality	stability
invertibility	memory

(a) $y[n] = 3x[n] - 4x[n-1]$
(b) $y[n] = 2y[n-1] + x[n+2]$
(c) $y[n] = nx[n]$
(d) $y[n] = \cos(x[n])$

Solution

(a) The output is a weighted sum of present and previous inputs. It is bounded if the input is bounded. A given input sequence is unambiguously related to the output sequence. Therefore the system possesses all six properties mentioned above.

(b) The present output depends on a future input, so the system is not causal. If the input signal ceases, the output goes on rising without limit, since each output value is twice the previous one. Therefore the system is unstable. But it possesses the other four properties mentioned above, namely: linearity, time-invariance, invertibility and memory.

(c) The output depends on the present input only, so the system has no memory. Since it also depends on the independent variable, the

system is time-variant. But it possesses the properties of linearity, causality, stability, and invertibility.

(d) A cosine function is periodic, so many different values of $x[n]$ would produce the same value of $y[n]$. Hence the system is not invertible. Neither is it linear, because if (for example) we double $x[n]$, we do not double $y[n]$. Also, since $y[n]$ depends only on $x[n]$, the system has no memory. However, it is time-invariant, causal, and stable.

PROBLEMS

SECTION 1.2

Q1.1 An analog television signal is to be coded by an ADC for transmission via a digital communication system. The signal contains significant frequencies up to 6 MHz. The quantization error in any one sample value, introduced by coding, must be less than 1 per cent of the total amplitude range of the ADC. Uniform quantization is to be used.

Assuming the Sampling Theorem is obeyed, what is the minimum rate of transmission of binary pulses, expressed in bits per second?

Q1.2 The analog television signal in Q1.1 fluctuates in amplitude, sometimes filling only 20 per cent of the amplitude range of the ADC. If the amount of quantization noise, *relative to the signal*, is to be no worse than before, what minimum transmission rate is required?

Q1.3 Repeat problem Q1.1, but for an electrocardiogram (EKG) signal containing significant components up to 200 Hz. The quantization error introduced by coding should again be less than 1 per cent of the ADC's amplitude range.

SECTION 1.3.1

Q1.4 Figure Q1.4 shows a digital signal $x[n]$. Sketch and label carefully the following signals:

(a) $x[n-2]$

(b) $x[3-n]$

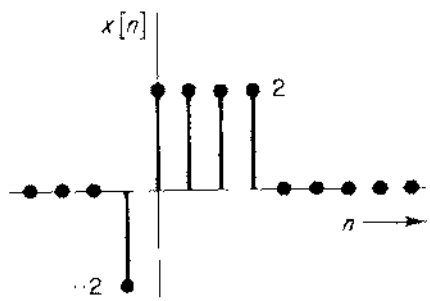

Figure Q1.4

(c) $x[n-1]\,u[n]$
(d) $x[n-1]\,\delta[n]$
(e) $x[1-n]\,\delta[n-2]$

Q1.5 If $x_1[n]$ is an even signal, such that $x_1[n] = x_1[-n]$; and $x_2[n]$ is an odd signal, such that $x_2[n] = -x_2[-n]$; then show that:

(a) $(x_1[n]\,x_2[n])$ is an odd signal
(b) $(x_2[n])^2$ is an even signal

(c) $\sum_{n=-\infty}^{\infty} x_2[n] = 0$

Q1.6 Find mathematical expressions for the various step, impulse, and ramp signals illustrated in Figure Q1.6.

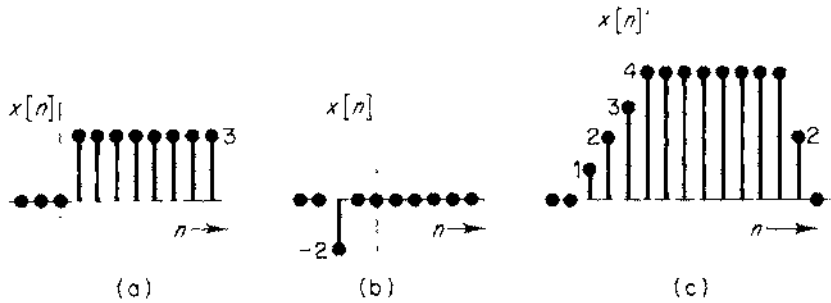

Figure Q1.6

Q1.7 Sketch, and label carefully, the following signals:

(a) $-u[n-2]$
(b) $u[n+1] + \delta[n]$
(c) $2u[n+2] - u[3-n]$
(d) $r[n] - 2r[n-3]$

SECTION 1.3.2

Q1.8 Determine which of the following signals is strictly periodic. If a signal is periodic, find its period.

(a) $x[n] = \sin\left(\frac{\pi n}{9}\right)$

(b) $x[n] = \sin(n\pi^2)$

(c) $x[n] = \cos\left(\frac{\pi n^2}{15}\right)$

(d) $x[n] = \sin\left(\frac{\pi n}{5} + \pi\right) + \cos\left(\frac{\pi n}{10} - \pi\right)$

Q1.9 Sketch carefully portions of the following signals:

(a) $x[n] = 3 \exp(-n/2)$
(b) $x[n] = \exp(n/5)$
(c) $x[n] = \sin\left(\frac{\pi n}{6}\right) u[n]$
(d) $x[n] = \exp(-n/4) \cos\left(\frac{\pi n}{2}\right) u[n]$

SECTION 1.5

Q1.10 $x[n]$ and $y[n]$ are the input and output signals of a digital processor. Determine which of the following properties are exhibited by each of the systems defined below: linearity, time-invariance, causality, stability, and memory.

(a) $y[n] = x[5-n]$
(b) $y[n] = x[n]\, x[n-3]$
(c) $y[n] = x[n] + x[n-1] + 3x[n-2]$
(d) $y[n] = 1.5\, y[n-1] + x[n-1]$
(e) $y[n] = nx[n]$

Q1.11 Which of the following systems are invertible? If a system is not invertible, find two values of the input which produce the same output.

(a) $y[n] = x[2-n]$
(b) $y[n] = x[n]\, x[n-1]$
(c) $y[n] = \sin(nx[n])$

Q1.12 Figure Q1.12 shows a block diagram of a digital processor. Assuming $y[n] = 0$ for $n < 0$, sketch the output from the processor when the input is (a) the unit impulse function $\delta[n]$, and (b) the unit step function $u[n]$. Assume the constant multiplier α is between 0 and 1. What would happen if $\alpha > 1$, or $\alpha < -1$?

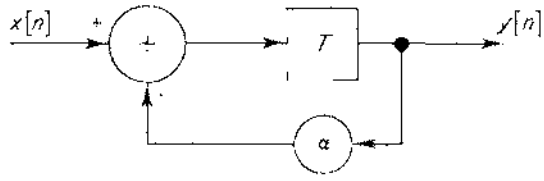

Figure Q1.12

Q1.13 Draw a block diagram for the digital bandpass filter defined by equation (1.4) in the main text.

Q1.14 Draw a block diagram for a digital processor with the following recurrence formula. Distinguish clearly between its nonrecursive and recursive parts.

$$y[n] = 1.625\, y[n-1] - 0.934\, y[n-2] + 0.5\, x[n] - 0.1\, x[n-2]$$