1. Introduction to Digital Signal Processing

1.1 What is Digital Signal Processing (DSP)?

From your earlier studies, you will be familiar with "signals" – how to describe them mathematically, how to generate them, and how to process them. For example, in EE221/EE222 Electrical Circuits and Systems, you will have studied simple signals like sinusoids, the unit impulse and the unit step, as well as more complex signals, and you will have learned how to describe these signals in terms of both time-domain equations and also transform-domain representations (e.g. Laplace Transform). You will also have learned how to determine the frequency content of signals by means of Fourier analysis.

You will also be familiar with "systems" – how to analyse their behaviour, how to specify them, and how to design them. For example, in EE308 Signals and Communications, you will have learned how to describe systems in terms of their impulse responses and their transform-domain representations (i.e, their transfer functions, or Laplace Transforms). You will also have learned how to analyse the behaviour of systems, most commonly by calculating the frequency response of the system (by means of the Fourier Transform). Furthermore, you will have studied the flip side of system analysis, namely system design. In particular, you will have learned how to "design" a signal processing system (e.g. a low–pass filter), usually by specifying its desired frequency response, following which you will have obtained a desired transfer function (e.g. using the Butterworth design methodology), after which you will have synthesised a practical realisation of the filter (either using purely passive components – resistors, capacitors, inductors - or perhaps using active techniques like the Sallen-Key low-pass filter configuration). For the most part, the signals and systems you will have studied will have been "analogue", i.e. signals are continuous-time and also continuous in amplitude – for convenience, we will refer to this as Analogue Signal Processing (ASP).

Digital Signal Processing (DSP) is largely concerned with doing essentially the same things – signal analysis and processing, system analysis and system design – but using digital techniques instead of traditional analogue techniques. In particular, signals and systems are represented in digital form, which means they can be easily manipulated using computer-based techniques. For example, if you want to implement a low-pass filter, ASP techniques would require you to build up a circuit using (say) discrete resistors, capacitors and op-amps. Using DSP, on the other hand, you can implement the filter using a software program running on a PC, or perhaps using digital logic implemented on an FPGA.

It is important to note that DSP is simply another way of handling signals and systems, and the concepts that apply to analogue signal processing still apply – an impulse response means exactly the same in DSP as it does in ASP, poles are still poles, zeros are still zeros etc. All you're doing is using different techniques for processing the same signals. Also, note that most signals of interest are analogue in nature (e.g. speech, audio, radio signals), so DSP requires some means of converting the signal of interest from analogue to digital form before processing, and then (usually) converting the result back to analogue form. For example, in a mobile phone, the speech signal originates as an analogue signal (in the form of a voice

pressure wave), which is converted into digital form before processing by complex algorithms embedded in the mobile handset. At the receiving end, the digital signal that has been transmitted across the telephone network is converted back into an analogue speech signal, before being sent to its final destination (i.e. the listener's ear).

Of course, there's no point doing this unless there are advantages (otherwise, why bother?).

1.2 Benefits of using DSP

1. **Reproducibility and Consistency**. Unlike ASP, DSP does not suffer from problems of differences in component values due to natural component tolerances (e.g. resistor values are typically specified to 1%, 5% or 10% accuracy). This means that any number of different "copies" of the DSP algorithm will yield identical results. ASP also suffers from problems of component drift over time (e.g. due to temperature variations), which causes algorithm behaviour to vary over time. This does not occur with DSP.
2. **Flexibility**. Changing a DSP function (e.g. designing a low-pass filter with a different cutoff frequency) is simply a matter of changing a few numbers in a software implementation, or perhaps re-programming an FPGA (i.e. there's no hardware change). In ASP, such a change would require different resistors or capacitors, which is much more problematic.
3. **Cost**. DSP is well-placed to take advantage of continuing improvements in VLSI technology, results in higher density, faster and lower cost digital implementations. Trends in analogue silicon technology have not followed the same rapid pace.
4. **Superior performance**. There are certain applications that are simply not practical using ASP, or can only be implemented with poor performance, but are much more feasible with DSP. Examples include linear phase filters, speech and audio processing algorithms (e.g. MP3), and speech recognition systems.

It is important to note that there are still some areas where ASP is more appropriate. In particular, applications that involve signals with very wide bandwidths can be quite difficult to implement with DSP, mainly because such signals require very high sampling rates. Also, signal processing at radio frequencies (hundreds of MHz or GHz) still needs a significant amount of ASP.

1.3 A Signal Processing System

A block diagram of a typical DSP system is shown in Figure 1.1. This system takes an analogue signal, $x(t)$, and carries out some processing on it to yield the output signal $y(t)$.

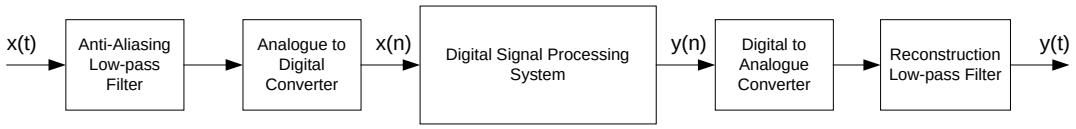

Figure 1.1. Block diagram of a typical DSP system.

The major elements of this system are as follows:

Anti-Aliasing Low-pass Filter: The function of this analogue filter is to constrain the bandwidth of the analogue signal $x(t)$ such that it contains no frequency components greater than the cut-off frequency of the filter (the reason for this will be explained in more detail below).

Analogue to Digital Converter (ADC): This samples the analogue signal at a certain sampling rate, thus producing the discrete-time ("digital") signal $x(n)$.

DSP System: This is the core signal-processing element of the system, and carries out the actual processing of the sampled signal (e.g. speech encoding and transmission in a mobile telephony application).

Digital to Analogue Converter (DAC): This takes the output of the DSP System, $y(n)$, and converts it back into analogue form.

Reconstruction Low-pass Filter: This is another analogue filter whose function is to "smooth" the output of the DAC, to produce a nice "clean" signal that is then sent to its final destination (e.g. the earpiece of a mobile phone).

1.4 Discrete-Time Signals

1.4.1 Sampling

Continuous-time ("analogue") signals are defined for every point in time (e.g. Figure 1.2(a)), however, discrete-time ("digital") signals are only strictly defined at specific instants of time, and are equal to zero at all other times. Normally, the instants of time at which discrete-time signals are specified are equal to integer multiples of the sampling period (which is the reciprocal of the sampling frequency). For example, Figure 1.2(b) shows the discrete-time signal which is obtained by sampling the signal in Figure 1.2(a) at a sampling frequency of 8 kHz (i.e. the sampling period is 125 μsec). While the discrete-time signal consists of "impulses" with specific voltage values, the underlying shape of the analogue signal can be clearly seen.

An intuitive interpretation of sampling is that the analogue signal is sampled every T seconds, where T is the sampling period (125 μsec in Figure 1.2), to generate a sequence of numbers $x(n)$, i.e.

$$x(n) = x(t)\Big|_{t=nT}$$

where n is an integer, usually referred to as the *sample index*.

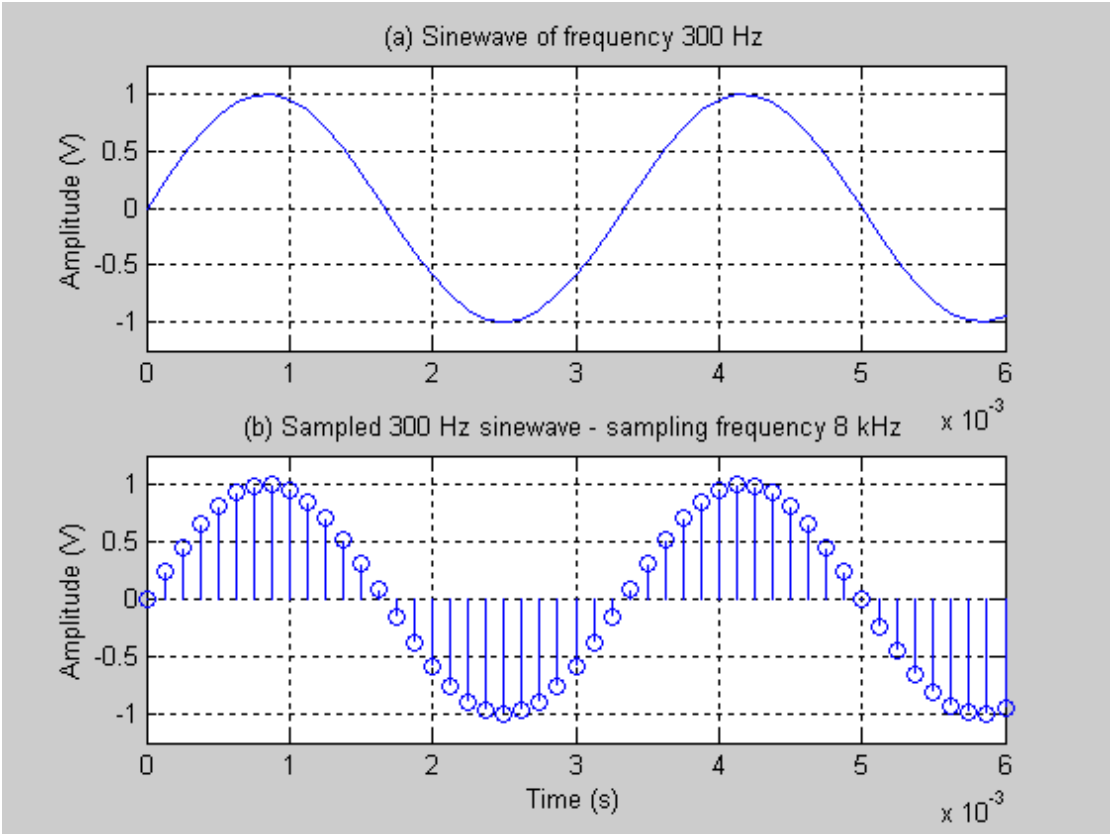

Figure 1.2. Example of a continuous-time signal, and its sampled version.

Note: Strictly-speaking, when converting an analogue signal to digital form, two operations are carried out. In the first instance, sampling of the analogue signal at the sampling frequency takes place, resulting in a signal that is discrete in time, but is still technically a voltage value that has "infinite" accuracy or resolution. Each sampled voltage value is then usually converted into a binary number with a certain number of bits to produce a quantised version of the original sample (see EE308 Signals and Communications) – this quantisation has to be done so that the sampled signal can actually be processed by a computer or digital hardware. For example, in speech processing applications, the use of 12-bit ADCs is common, so each sample is converted into a 12-bit integer (normally 2's complement), with a value of –2048 to +2047. If the input voltage range of the ADC is (say) +/-0.5V (1 V peak-to-peak), then a voltage sample of (say) 0.34 volts will result in an integer sample value of approximately 1392 (depending on the specifics of operation of the ADC). Essentially, what has happened is that the original continuous voltage signal has been converted into a string of integer numbers that can be readily manipulated by computer.

For the most part in this course, we will examine DSP without being concerned about the effects of quantisation, and will simply assume that all numbers are represented using floating-point arithmetic. Technically, the signal is still quantised because even floating-point numbers are still represented by a finite number of bits, but the accuracy of floating-point arithmetic is such that any quantisation error can be ignored. However, quantisation places a fundamental limitation on the performance of DSP systems implemented in hardware and software, and it's important to be aware of this fact.

In addition, we will use the terms "discrete-time", "sampled" and "digital" interchangeably (even though strictly-speaking, "digital" refers to a signal that has undergone both sampling and quantisation). Also, when plotting discrete-time signals, the "correct" way of doing this is to plot the individual "impulses" of the discrete-time signal. However, for convenience, we will often "join the dots" to produce a pseudo-continuous waveform so that the shape of the underlying signal can be more readily seen.

1.4.2 Reconstruction

In principle, if the analogue signal is adequately sampled (more below), then it should be possible to accurately reproduce the original analogue signal $x(t)$ from the sequence of samples $x(n)$ (ignoring any quantisation noise that may have occurred in the sampling process). Mathematically, the so-called reconstruction theorem can be expressed as follows:

$$x(t) = \sum_{n=-\infty}^{\infty} x(n)\text{sinc}\left(\frac{t-nT}{T}\right)$$

where T is the sampling period, and sinc(x) is the well-known sinc function, given by:

$$\text{sinc}(x) = \sin(\pi x)/(\pi x)$$

Intuitively, what this means is that the continuous time signal is reconstructed from the sampled sequence, by adding together an infinite set of sinc-functions. Each sinc function is delayed a by a time interval equal to the sampling period, and weighted by the corresponding sample value. For example, Figure 1.3 shows a plot of one such sinc-function, for a sampling frequency of 100 Hz (T=10 msec). In this case, the sinc function has been delayed by 5 sample periods. It can also be seen that the zero-crossings of the sinc-functions are located at multiples of the sampling period. Effectively, the sinc functions enable "interpolation" between the samples, resulting in a smooth "analogue" waveform.

In practice, the combination of the DAC and reconstruction low-pass filter in Figure 1.1 carry out the task of regenerating an analogue signal from a sampled sequence. Of course, these circuits do not actually directly implement the mathematical form of the reconstruction theorem given above. Instead, in simple terms, the DAC operates as follows. It takes each sample (in binary form) and outputs the corresponding voltage value (basically, the inverse of the ADC operation described earlier). It holds this voltage value constant for a period equal to the sampling period, after which time, of course, the next sample arrives from the Digital Signal Processor and the process repeats. So, in effect, the DAC itself produces a "staircase" waveform, with discontinuities ("steps") at the border between sampling periods (the DAC is a so-called "Zero Order Hold" circuit). These discontinuities mean that the staircase waveform contains high frequencies (recall Fourier analysis from EE308 Signals and Communications), so the reconstruction low-pass filter removes these high frequencies to produce a "smooth" waveform, which should be bandlimited to less than half the sampling frequency.

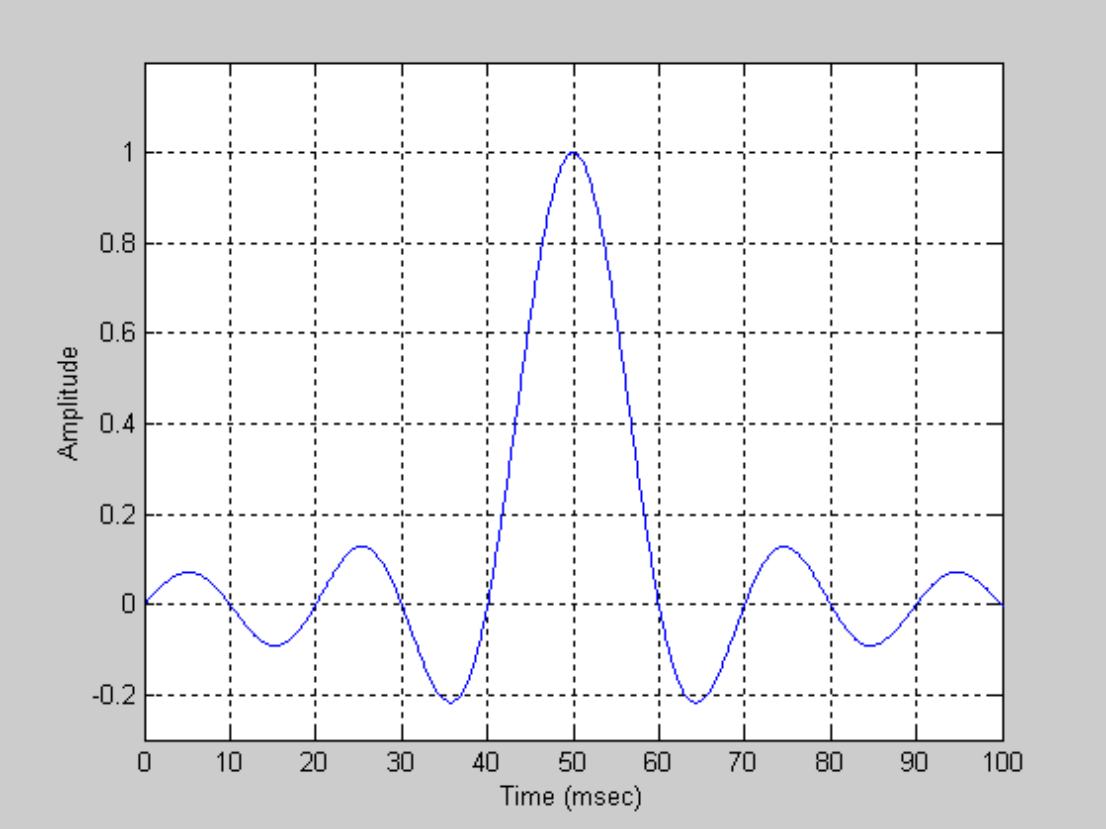

Figure 1.3. Plot of sinc function (sampling frequency 100 Hz)

1.4.3 Choice of Sampling Frequency

In this Section, we will briefly discuss how to choose the sampling frequency. In later sections, we will examine this topic in a bit more detail (particularly in the frequency domain), but for now, we will simply state the theorem that determines the choice of sampling frequency – the famous Sampling Theorem, one form of which is as follows:

> *When sampling an analogue signal with a maximum frequency content of f_{max}, the sampling frequency f_{samp} must be greater than or equal to $2f_{max}$ in order to adequately preserve the information in the signal.*

(Both Harry Nyquist and Claude Shannon contributed to the development of sampling theory, so the sampling theorem is sometimes called either the Nyquist Sampling Theorem or Shannon's Sampling Theorem.)

An alternative way of expressing the Sampling Theorem is as follows:

> *If an analogue signal is sampled at a rate of f_{samp}, the analogue signal must not contain frequency components greater than $f_{samp}/2$, in order to adequately preserve the information in the signal.*

The maximum analogue frequency is often called the "Nyquist Frequency", and the minimum acceptable sampling rate – which is twice the Nyquist Frequency – is often called the "Nyquist Rate".

If the sampling frequency is too low (less than the Nyquist Rate), then there is ambiguity as to what frequencies the sampled signal actually contains. To take a

simple example, suppose we have an analogue sinusoid of frequency 500 Hz. According to the sampling theorem, we should sample this at a rate of at least 1000 Hz. However, suppose we sample this signal at a rate of only 750 Hz. If we try to reconstruct the analogue signal, we will find that we obtain a signal of frequency 250 Hz. This is illustrated in Figure 1.4, where the dashed line shows the waveform that would be produced by interpolation between the 750 Hz samples. Put another way, we sampled an analogue sinusoid of frequency ($f_{samp}/2$)+A Hz, where A is 125 Hz, and the samples we obtained were the same ones we would have obtained if the original analogue sinusoid actually had a frequency of ($f_{samp}/2$)-A Hz. These two frequencies sitting A Hz on either side of $f_{samp}/2$ are called "aliases" of each other. The apparent "folding" of the 500 Hz waveform back into a 250 Hz waveform is called "aliasing". As far as the digital signal processing system is concerned, it thinks it's dealing with a signal of frequency 250 Hz, not 500 Hz.

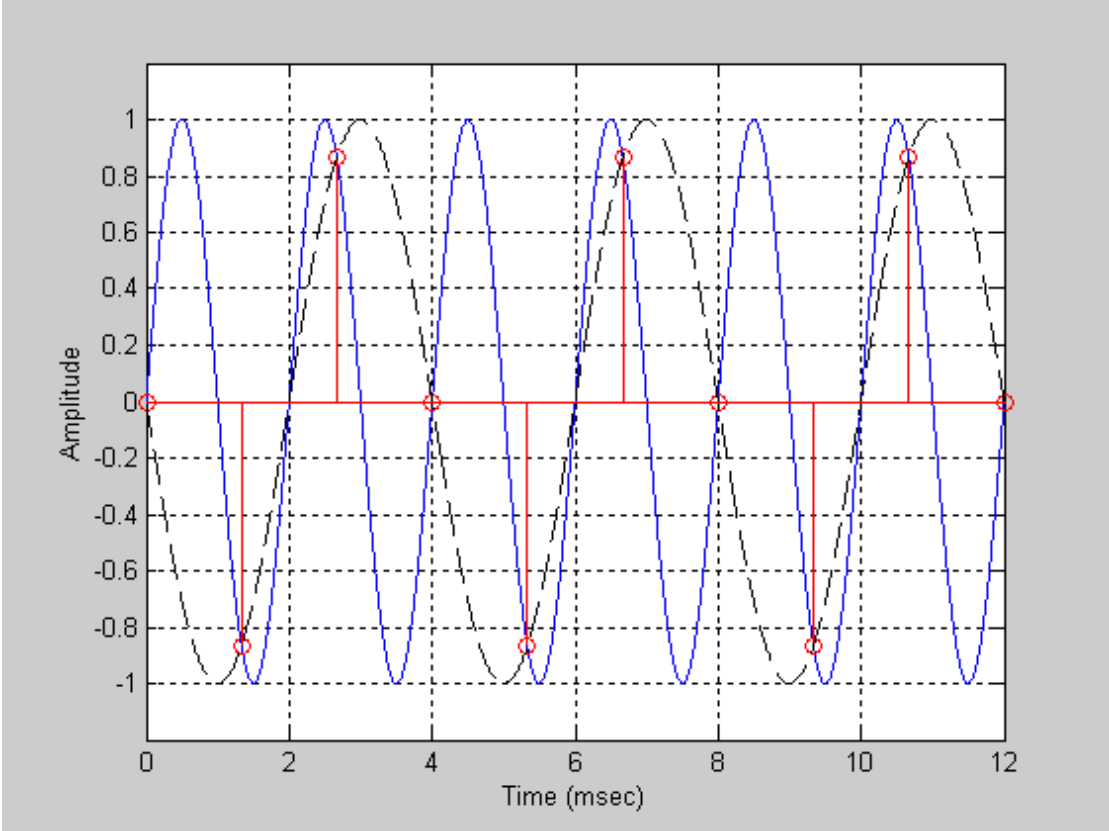

Figure 1.4. Illustration of aliasing (solid line – original 500 Hz analogue signal, dashed line – "aliased" 250 Hz reconstructed signal, samples taken at 750 Hz are indicated by "impulses").

Even if the sampling frequency is chosen so that it is greater than twice the highest frequency of interest, aliasing or "folding" of analogue frequency components greater than $f_{samp}/2$ back into the signal bandwidth below $f_{samp}/2$ is still possible – at the very least, the analogue signal will contain some amount of noise (however small). This noise is generally broadband in nature ("white"), so obviously it's going to contain some energy above half the sampling frequency. Alternatively, the signal itself may contain high frequency components that are deemed to be of no use for the particular application, so we want to remove these to keep the sampling rate down. A good example of this is in sampling speech signals for telephony, where conversational

speech contains frequencies up to 6 or 7 kHz (albeit with decreasing energy at higher frequencies). However, speech signals are generally sampled at 8 kHz, implying that the frequencies of primary interest are below 4 kHz – the higher frequencies must be discarded before sampling. In either of these situations, the aliased frequency components from above $f_{samp}/2$ essentially cause distortion in the frequency band of interest below $f_{samp}/2$. Once this aliasing distortion occurs, there's really no way of removing it.

The primary purpose of the Anti-Aliasing Low-Pass Filter in Figure 1.1 is to carry out band limiting of the analogue signal, to minimise aliasing. The anti-aliasing filter itself can never completely bandlimit the signal either, because that would require an ideal "brick wall" filter, which is not realisable in practice. So, the objective in the design of anti-aliasing filters is to make sure whatever aliasing distortion may occur is so small that it will have minimal impact on the application. For example, in speech applications with a sampling frequency of 8 kHz, we may require that any frequency components above 4 kHz are attenuated by at least 30 dB before sampling (so the "signal to aliasing noise ratio" is at least 30 dB). Furthermore, since we cannot design a brick wall filter with a cutoff frequency of 4 kHz and a transition bandwidth of 0, we have to design a filter with a non-zero transition band – in speech applications, a cutoff frequency of 3.4 kHz is generally used.

Notes

1. As described above, the sampling theorem relates the minimum sampling frequency to the maximum frequency of the analogue signal. Technically, it's the bandwidth (rather than just the maximum frequency component) that's of interest, however, for the most part we will be dealing with signals that are "low pass" in nature, so this form of the sampling theorem is fine for our purposes. We will return to sampling band-pass signals at a later stage.
2. As a general rule, it's desirable to keep the sampling rate as low as possible for the particular application (i.e. we generally try to sample as low as the Nyquist Rate). The main advantages of keeping the sampling rate low are the fact that we're minimising the amount of "digital" information we're generating, thus reducing storage requirements (for the same duration, fewer samples means less storage), and we're also reducing the computational requirements of whatever we're using to implement our DSP system (lower sampling rate means fewer instructions per second needed, which means lower cost hardware/microprocessors). However, there are times when increasing the sampling rate to some multiple of the Nyquist Rate is beneficial. For example, going back to our speech application above, to use a sampling rate of 8 kHz we require an anti-aliasing filter with cutoff frequency 3.4 kHz, and with attenuation of >30 dB at 4 kHz, which is a reasonably tough specification – the transition band is only 600 Hz wide (what order would be required for a Butterworth filter?). However, if we were to sample at (say) 32 kHz instead of 8 kHz, then we need >30 dB attenuation at 16 kHz. If we retain a cutoff frequency of 3.4 kHz, our transition band is now 16-3.4 = 12.6 kHz wide, which means we can use a much simpler analogue anti-aliasing filter. Of course, we would still like to reduce the sampling frequency of the sampled signal to 8 kHz from 32 kHz, which means that we still have to low-pass filter the signal, albeit in its sampled form. However, this is generally much easier

to do using DSP techniques that with the original analogue anti-aliasing filter. (Note that the principles of aliasing still apply here even though we've already sampled the signal, because we're reducing the sampling frequency, so aliasing can still occur; viewed another way, we could also say that the original sampling of the analogue signal in the ADC was a reduction of the sampling frequency from a sampling rate of "infinity" to 32 kHz). The topic of "oversampling" will be treated in detail in *EE444 Communications and Signal Processing Applications* in Semester 2.

1.5 Important Digital Signals

1.5.1 Unit Impulse

This is of fundamental importance in DSP, not least because it enables us to obtain the impulse response of a discrete-time system (much as the Dirac delta function is used with analogue systems). It is defined by:

$$\delta(n) = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases}$$

The unit impulse also has a delayed version, defined by:

$$\delta(n-k) = \begin{cases} 1, & n = k \\ 0, & n \neq k \end{cases}$$

where k is the sample delay. The unit impulse and delayed unit impulse (with k=4) are plotted in Figure 1.3 (a) and (b) respectively.

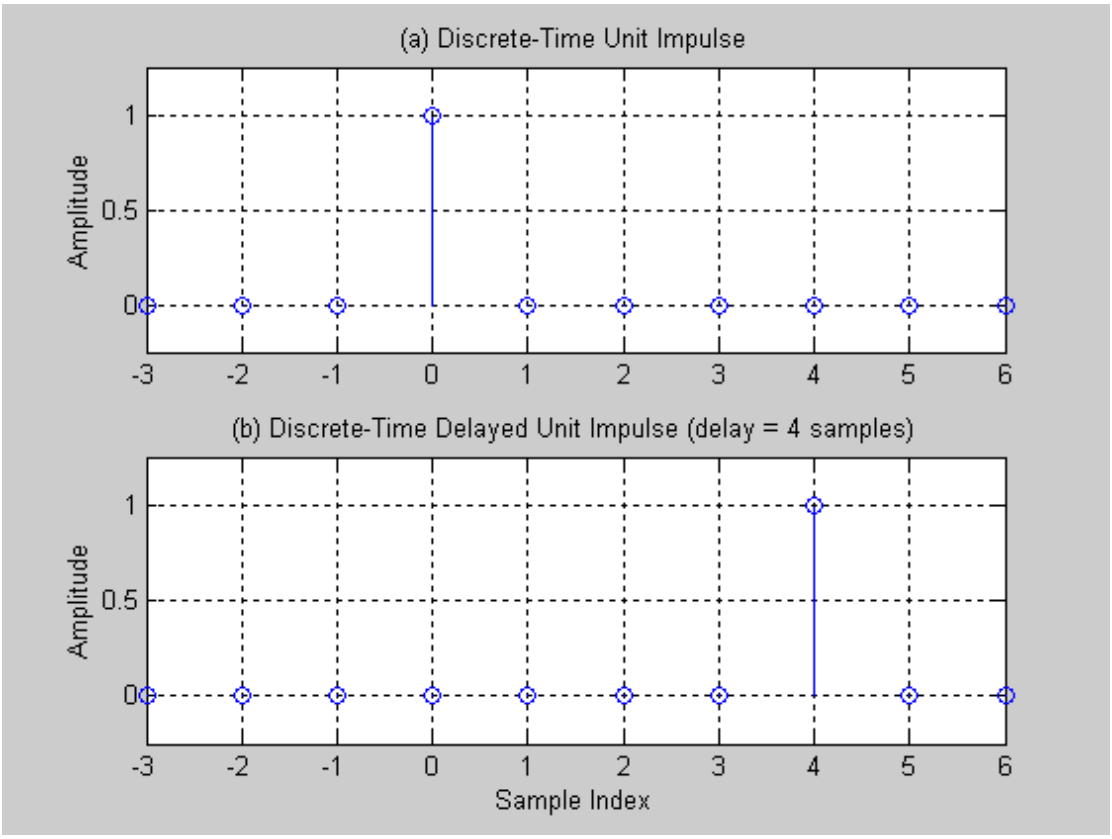

Figure 1.3. Plot of Unit Impulse and Delayed Unit Impulse.

1.5.2 Unit Step Function

The discrete-time unit step is simply a "sampled" version of the analogue unit step, and is defined by:

$$u(n) = \begin{cases} 1, & n \geq 0 \\ 0, & n < 0 \end{cases}$$

while the delayed unit step is defined by:

$$u(n-k) = \begin{cases} 1, & n \geq k \\ 0, & n < k \end{cases}$$

The unit step and delayed unit step (k=2) are plotted in Figure 1.4.

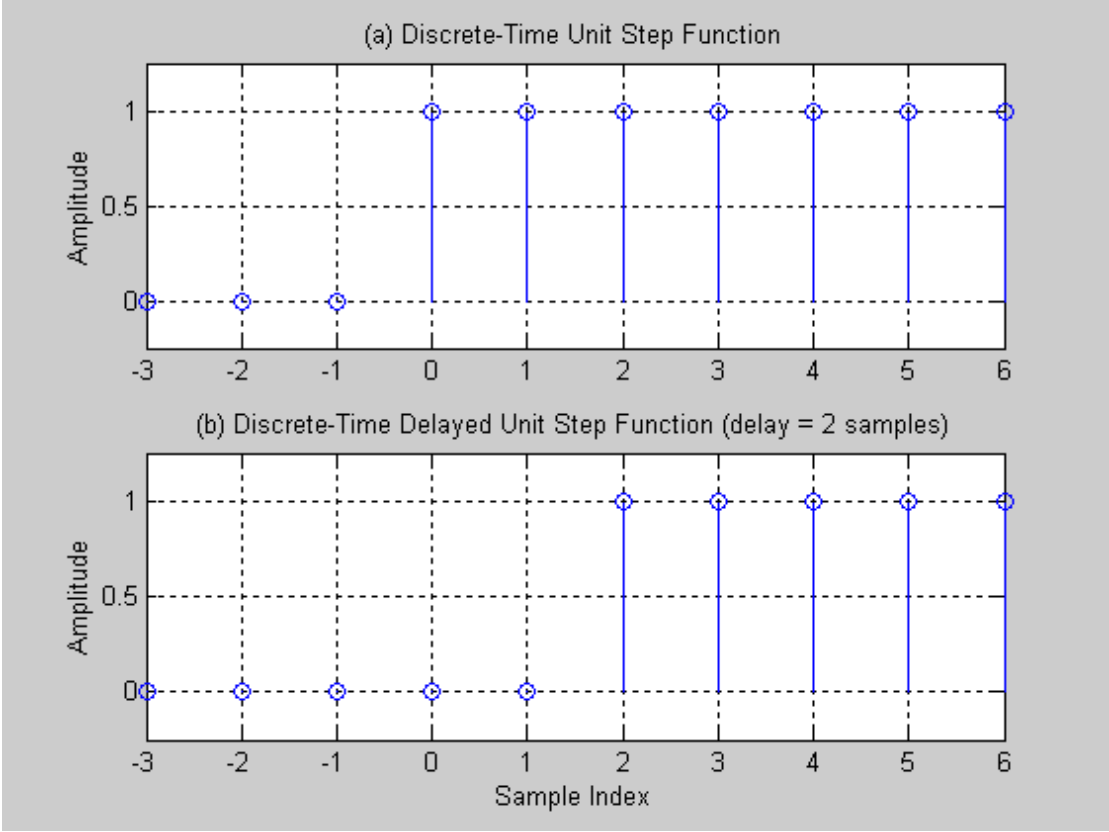

Figure 1.4. Discrete-time Unit Step and Delayed Unit Step.

1.5.3 Sampled Exponential

The sampled exponential is defined by:

$$x(n) = a^n u(n)$$

where a is some constant. Note that multiplication by $u(n)$ (the unit step function) means that the sampled exponential is zero for negative values of the sample index. The exact form of the sampled exponential depends on the value of the constant a. This dependency can be summarised as shown in Table 1.1, and plotted in Figure 1.5:

Value of *a*	Form of function
Positive, less than 1	Decaying
Negative, less than 1	Decaying oscillation
Positive, greater than 1	Increasing ("unstable")
Negative, greater than 1	Increasing oscillation

Table 1.1. Dependency of sampled exponential behaviour on constant *a*.

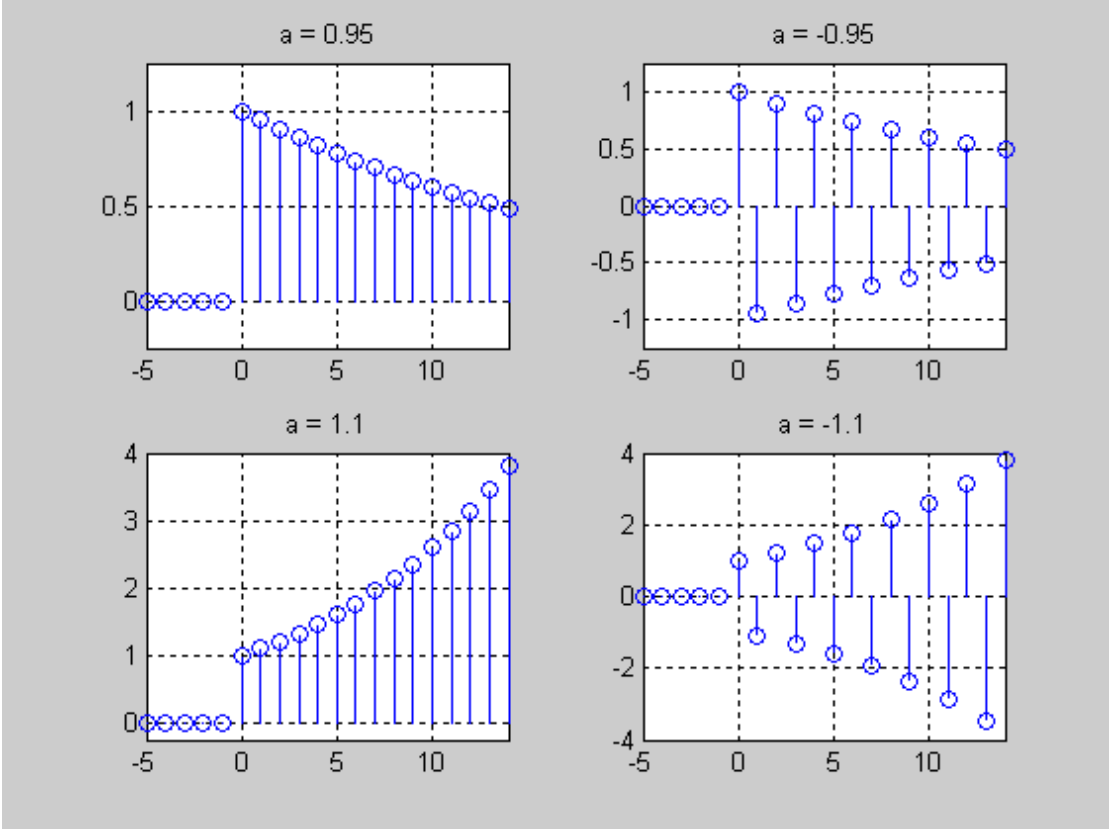

Figure 1.5. Plot of sampled exponential for different values of *a*.

1.5.4 Discrete-Time Sinusoidal Signal

The discrete-time sinusoidal signal is derived from the underlying analogue sinusoidal signal:

$$x(t) = A\sin(\omega_a t)$$

where A is the amplitude of the sinusoid, and ω_a is the frequency in radians per second. By sampling this signal every T seconds, we obtain:

$$x(n) = A\sin(\omega_a nT) = A\sin(2\pi n f_a/f_s)$$

where f_a is the frequency of the sinusoid (in Hz), f_s is the sampling frequency (in Hz), and n is the sample index as before. For example, Figure 1.2 shows a sinusoid with frequency 300 Hz, being sampled at a rate of 8 kHz.

The quantity $2\pi f_a/f_s$ has a special meaning in the context of DSP, and is often referred to as the *digital frequency* or *relative frequency*, with symbol θ (other symbols may also be used in textbooks etc.). Therefore, the sampled sinusoid can be re-written as:

$$x(n) = A\sin(n\theta)$$

The units of θ are radians, rather than radians/s as you're used to dealing with. Intuitively, you might expect this to be the case, because the argument of the trigonometric sin function is $n\theta$, and n is a dimensionless integer. Note that as the underlying analogue frequency f_a increases from 0 Hz (DC) up to the sampling frequency f_s, the value of θ increases from 0 to 2π.

A digital *sequence* is said to be periodic, with period N, if N is the smallest integer for which:

$$x(n+N) = x(n)$$

For example, the sampled sinusoid is periodic with period N samples, if

$$A\sin([n+N]\theta) = A\sin(n\theta)$$

This can only be satisfied if $N\theta$ is an integer multiple of 2π, i.e. $N\theta = 2\pi k$, where k is an integer. Expressed another way,

$$N = \frac{2\pi k}{\theta} = \frac{f_s}{f_a}k$$

Since N must be an integer (and k is an integer), this means that the ratio of the sampling frequency to the analogue frequency must also be an integer. In other words, in the case of a sinusoid, there will be an exact integer number of digital samples in every period of the underlying analogue waveform. For example, a 100 Hz analogue sinewave sampled at 8 kHz will have 80 samples per period of the sinewave, and this set of 80 samples will repeat itself over and over again.

Note: Care must be taken when relating periodicity of the sampled sequence to the periodicity of the underlying analogue waveform. The description above relates to conditions where the sampled sequence has the same period (in seconds) as the underlying analogue waveform. However, this is not always the case. A sampled sequence may not have an exact integer number of samples per period of the waveform, but this does not necessarily mean that the underlying waveform is not periodic. For example, for the 300 Hz sinewave shown in Figure 1.2, the number of samples per period is 8000/300 = 26.6667 (not an integer), but the underlying waveform is still periodic (it's a 300 Hz sinewave with period ~3.3 msec). Of course, according to the definition of periodicity of a sampled sequence above, this particular sequence has a period of 80 samples, which is 10 msec – not the same as the period of the underlying analogue waveform (so be careful!). Generally-speaking, we're more concerned with the periodicity of the underlying analogue waveform – after all, that's what we're processing – and we're less concerned with periodicity of the sampled sequence (though there are exceptions to this, which we'll deal with as we go along).

Matlab Example 1.1

Write Matlab code to generate 100 samples of a discrete-time sinewave of amplitude 3 volts, with sampling frequency of 8 kHz and underlying "analogue" frequency 1 kHz.

Suggested Solution

```
% Some constants
fs = 8000;      % sampling frequency
fa = 1000;      % "analogue" frequency
N = 100;        % number of samples required
A = 3;          % amplitude
% First, do it the "brute force" way, with a for loop (like in C)
for n = 1:N,
        x1(n) = A * sin(2*pi*n*fa/fs);
end;
% Now, do it taking advantage of Matlab's inherent vector handling ability
% first, generate a vector containing the sample index values (1 to N)
sample_index = 1:N;
x2 = sin(2.*pi.*sample_index.*fa./fs);
% plot
figure;
stem(x2);       % plot "impulses"
grid on;
```

Exercise 1: Repeat this exercise for an analogue frequency of 1100 Hz. Examine the sequence of samples, and determine the period of the *sampled sequence*.

Exercise 2: In the above example, the first sample index is 1, which implies that the sample sequence starts at time 125 μsec. Modify the code (both methods) to generate 100 samples starting at time 0.

Matlab Example 1.2

An Amplitude Modulated (AM) waveform is given by the following equation:

$$x(t) = A[1 + m\cos(\omega_m t)]\cos(\omega_c t)$$

where ω_c is the carrier frequency (in radians/s), ω_m is the modulating frequency, A is the carrier amplitude, and m is the modulation index. The digital equivalent is:

$$x(n) = A[1 + m\cos(\omega_m nT)]\cos(\omega_c nT) = A[1 + m\cos(n\theta_m)]\cos(n\theta_c)$$

where T is the sampling period (note that the digital equivalent was obtained from the analogue version by replacing time t with nT). Write Matlab code to generate the digital AM waveform, given the carrier, modulating and sampling frequencies in Hz.

Suggested solution

```
% Some constants
fs = 10e3;
fc = 1000;
fm = 80;
m = 0.8;
A = 1;
Nsamp = 500;
% Calculate the digital frequencies
theta_m = 2*pi*fm/fs;
theta_c = 2*pi*fc/fs;

% Generate the AM waveform ("efficiently")
sample_index = 0:nsamp-1;
x = A.*(1+m.*cos(sample_index.*theta_m)).*cos(sample_index.*theta_c);

% plot
figure;
plot(x);
grid on;
```

1.5.5 An Alternative View of Digital Signals

In Section 1.5.1, we came across the most basic of discrete-time signals, the unit impulse (and its delayed version). From your earlier studies in EE221/EE222

Electrical Circuits and Systems (and other places), you will have come across the *sifting property* of the Dirac delta function, which can be stated as follows:

$$\int_{-\infty}^{\infty} x(t)\delta(t-\tau)dt = x(\tau)$$

which basically says that the integral of the product of a function $x(t)$ and a Dirac delta function with delay τ, is equal to the value of the function at the same delay τ.

In discrete-time notation, the corresponding equation is:

$$x(n)\delta(n-k) = x(k)\delta(n-k)$$

i.e. if we multiply an arbitrary signal by a delayed unit impulse function, the result is a signal that is zero for all values of the sample index except $n=k$, and its value here is equal to the value of the original arbitrary function at that sample index.

The logical follow-on to this is the fact that we can represent any arbitrary sequence by means of an infinite sum of delayed unit impulses, each one weighted by the corresponding sample value of the arbitrary sequence. For example, suppose we have an arbitrary, finite-duration sequence, $x(n)$, as shown in Figure 1.6.

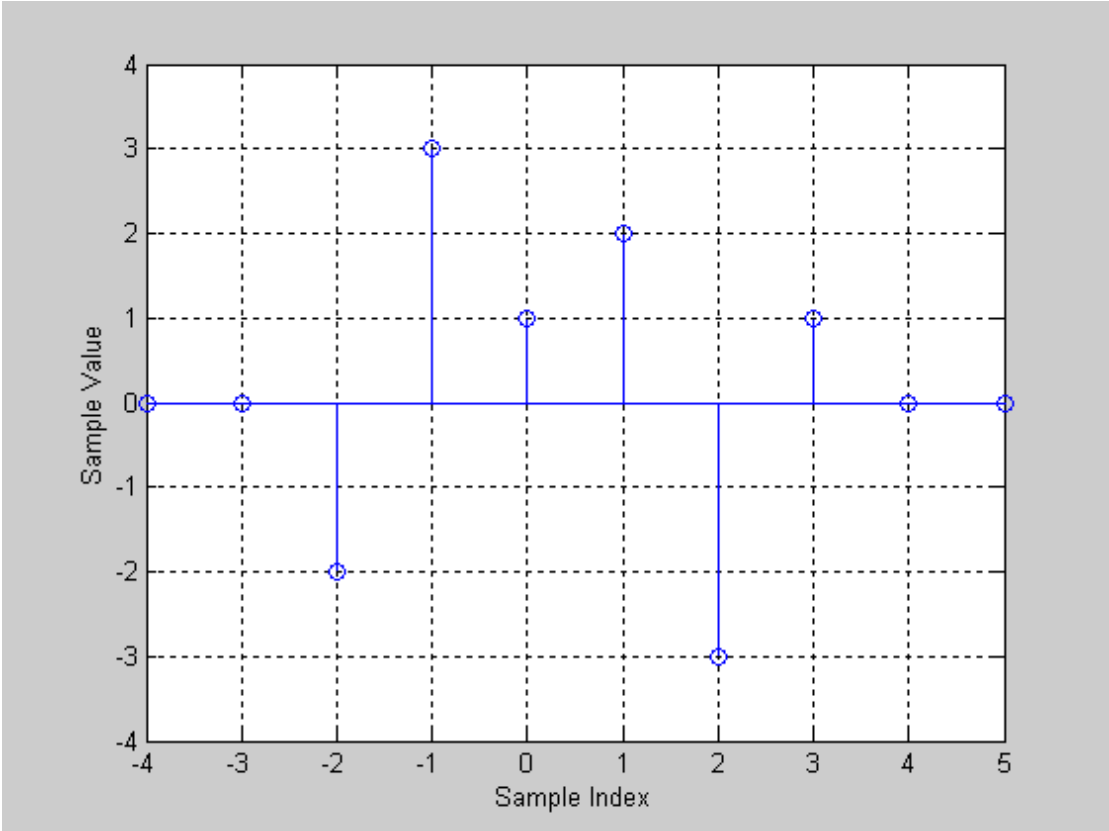

Figure 1.6. Arbitrary finite-duration sample sequence.

This sequence can be re-written as follows:

$$x(n) = -2\delta(n+2) + 3\delta(n+1) + \delta(n) + 2\delta(n-1) - 3\delta(n-2) + \delta(n-3)$$

More generally, arbitrary sequences can be expressed as:

$$x(n) = \sum_{k=-\infty}^{\infty} x(k)\delta(n-k)$$

We will return to this important point later.

1.6 Linear Time-Invariant (LTI) Discrete-Time Systems

1.6.1 Introduction

We now turn our attention to Linear, Time-Invariant Discrete-Time systems, which have many of the same characteristics as their continuous-time counterparts. In essence, LTI systems have the following two fundamental characteristics:

Linearity:

If the response of the system to an input $x_1(n)$ is $y_1(n)$, and its response to x_2(n) is y_2(n), then its response to $a_1x_1(n) + a_2x_2(n)$ will be $a_1y_1(n) + a_2y_2(n)$.

Time Invariance

If the response of a system to an input $x(n)$ is $y(n)$, then its response to $x(n\text{-}N)$ will by $y(n\text{-}N)$.

In the context of DSP, LTI discrete-time systems are usually referred to as "digital filters". Such filters can be constructed from three fundamental mathematical operations (see Figure 1.7):

- Addition (or subtraction)
- Multiplication (usually of a signal by a constant)
- Time delay, i.e. delaying a digital signal by one or more sample periods.

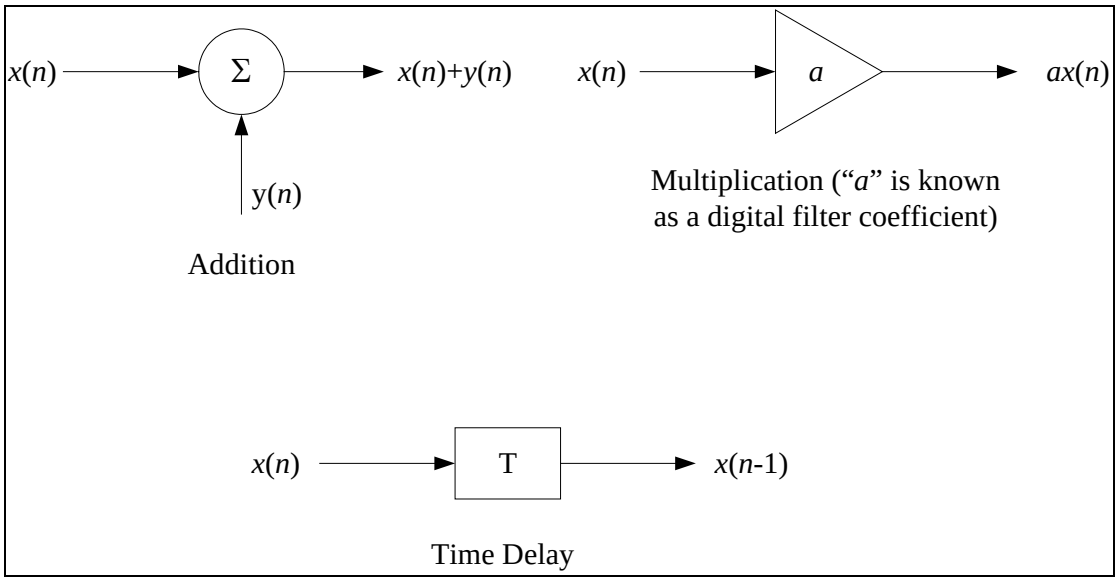

Figure 1.7. Fundamental operations in digital filters.

Digital filters may be specified or described in a number of different ways (as can continuous-time or "analogue" filters):

- Block diagram
- Difference equation

- Impulse response
- Frequency response
- Z-transform (more later)
- Pole-zero diagram

We will examine some of these different forms in this section, and others will be discussed in later sections.

1.6.2 Block Diagram

A very commonly used graphical means of describing a digital filter is the block diagram form, whereby the behaviour of the filter is described using the basic operations listed above (i.e. addition, multiplication and time-delay). For example, Figures 1.8(a) and (b) shows the block diagrams for very simple first-order digital filters:

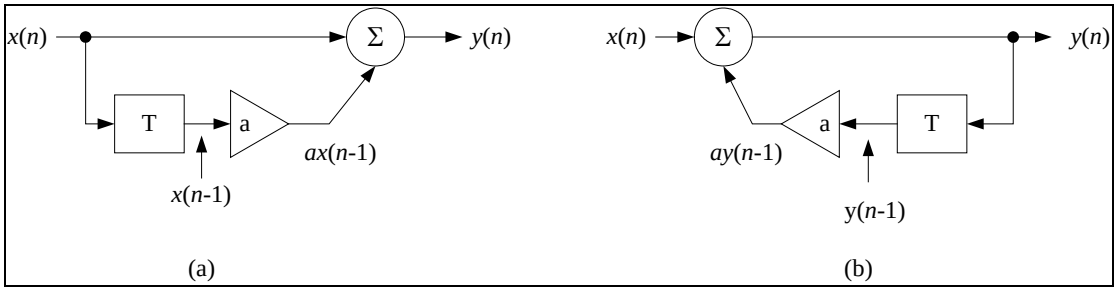

Figure 1.8. Block diagrams of simple digital filters.

1.6.3 Difference Equation

A difference equation is a simple mathematical statement of how the output, $y(n)$, of a digital filter is calculated. It is based on some or all of the following quantities:

- The current input, $x(n)$
- Previous input samples $x(n\text{-}1)$, $x(n\text{-}2)$ etc.
- Previous output samples $y(n\text{-}1)$, $y(n\text{-}2)$ etc.
- Various constants that are used to multiply these samples – these constants in effect determine the behaviour of the filter, and are usually referred to as **digital filter coefficients**.

The difference equation may be readily derived from the block diagram. For example, the difference equation corresponding to Figure 1.8(a) is:

$$y(n) = x(n) + ax(n-1)$$

which depends only on the current input sample and the previous input sample. The difference equation corresponding to Figure 1.8(b) is:

$$y(n) = x(n) + ay(n-1)$$

which depends only on the current input sample and the previous **output** sample.

A simple classification of digital filters is into two categories:

- Non-recursive, where the output depends only on the current and previous inputs

- Recursive, where the output depends not only on the current and (possibly) previous inputs, but also on previous outputs, i.e. there is feedback or recursion from the output to the input.

Figure 1.8(a) is an example of a non-recursive filter, while Figure 1.8(b) is an example of a recursive filter. Table 1.1 shows how the first few samples of the filter in Figure 1.8(a) can be calculated, while Table 1.2 shows how the output of Figure 1.8(b) can be obtained – in both cases, an arbitrary input signal is used, and the digital filter coefficient a has a value of 0.9. Of course, the difference equation may also be readily calculated using a programming language like C or Matlab.

Note that the difference equation plays a role in digital systems that is essentially the same as the role played by differential equations in continuous-time systems.

n	$x(n)$	$x(n-1)$	$ax(n-1)$	$y(n)$
0	1	0	0	1
1	0.5	1	0.9	1.4
2	-1.3	0.5	0.45	-0.85
3	0.6	-1.3	-1.17	-0.57
4	1.1	0.6	0.54	1.64
		etc.		

Table 1.1. Evaluation of difference equation of filter in Figure 1.8(a), for a=0.9

n	$x(n)$	$y(n-1)$	$ay(n-1)$	$y(n)$
0	1	0	0	1
1	0.5	1	0.9	1.4
2	-1.3	1.4	1.26	-0.04
3	0.6	-0.04	-0.036	0.564
4	1.1	0.564	0.508	1.608
		etc.		

Table 1.2. Evaluation of difference equation of filter in Figure 1.8(b), for a=0.9

Causality

Generally-speaking, we will be primarily interested in signals that are non-zero only for values of the sample index, n, greater than or equal to zero, i.e. most signals of interest are assumed to "begin" at n=0 (corresponding to a time of 0 seconds). This is simply a restatement of the concept of *causality*, which you will probably have come across before. In the case of digital filters, causality means that the filter output can only depends on current and previous values of the input, and previous values of the output, and not on future samples like $x(n+2)$, $y(n+5)$ etc. In the real world, this is intuitively obvious – after all, how can the current output sample of a filter depend on an input sample that hasn't yet appeared, or depend on an output sample that's not supposed to be calculated for a while yet? An exception to this rule is the case where a block of signal samples have been recorded and stored, for later processing – in this

case, the digital filter could have access to "future" samples, but this is generally not what we mean.

Note that in the above difference equation calculations, we assume that $x(n\text{-}1)$, $y(n\text{-}1)$ etc. are zero for $n=0$. In other words, the filter input signal and output signal are assumed to be zero for "negative" values of the sample index. This is an assumption that is frequently made when calculating the output of a digital filter (for the input signal, of course, it's simply causality). It basically means that the digital filter is starting from a "clean" state (zero initial conditions). This doesn't have to be the case, of course, as we sometimes want to know the response of a digital filter to a particular input signal in a situation where the system may not have finished responding to a previously-applied input, i.e. the difference equation has non-zero initial conditions. However, such cases will be rare.

Exercise 1.1

Draw the block diagrams, and calculate the first 5 output samples, for the digital filters represented by the following difference equations:

(a) $y(n) = x(n) + 0.5x(n-1) - 0.2x(n-2)$

(b) $y(n) = 0.5x(n) - 0.6x(n-1) + 0.1x(n-3)$

(c) $y(n) = x(n) + 0.6x(n-1) - 0.1x(n-2) + 0.3y(n-1) - 0.3y(n-2)$

In all cases, the input signal is the unit step function.

1.6.4 Impulse Response

The impulse response of a digital filter, $h(n)$, is the response of the filter to an input consisting of the unit impulse function, $\delta(n)$, and it plays a role in discrete-time systems that is exactly the same as its role in continuous-time systems. If the impulse response of a system is known, it is possible to calculate the system response to any input sequence, $x(n)$. We have already seen that any arbitrary sequence can be represented by a sum of weighted, delayed unit impulses, i.e.

$$x(n) = \sum_{k=-\infty}^{\infty} x(k)\delta(n-k)$$

We know from the time-invariance property of an LTI system that the response of the system to a delayed unit impulse $\delta(n\text{-}k)$ will be a delayed version of the impulse response, i.e. $h(n\text{-}k)$. Further, from the linearity property, we know that the response of the system to a weighted sum of inputs will be a weighted sum of the responses of the system to each of the individual inputs. Therefore, we can write the response of a system to an arbitrary input $x(n)$ as follows:

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k)$$

or, in the case of a causal input (normally the case):

$$y(n) = \sum_{k=0}^{\infty} x(k)h(n-k)$$

This equation called the *convolution sum* of two sequences $x(n)$ and $h(n)$, and the operation of convolution is given the symbol "*", i.e.:

$$y(n) = x(n) * h(n) = \sum_{k=0}^{\infty} x(k)h(n-k)$$

assuming a causal input. Convolution is commutative, i.e. $x(n)*h(n) = h(n)*x(n)$. Also, convolution as applied to discrete-time systems has exactly the same meaning as when applied to continuous-time systems.

As a further important point, by definition, the unit impulse is applied to the system at sample index n=0. Hence, we expect that the impulse response is non-zero only for values of n greater than or equal to zero, i.e. $h(n)$ is zero for n<0. Such an impulse response is said to be causal. Again, you would expect this to be the case – after all, how could the system produce a response to an input that hasn't yet been applied?

The process of convolving two sequences, $x(n)$ and $h(n)$, can be described by the following steps:

1. Re-write $x(n)$ and $h(n)$ as $x(k)$ and $h(k)$; all we are doing here is changing the variable used for the sample index.
2. "Fold" over (or reverse) $h(k)$ in time, to get $h(-k)$.
3. Delay h($-k$) by n samples to get $h(n-k)$.
4. Multiply $x(k)$ by $h(n-k)$, to obtain $x(k)h(n-k)$.
5. Sum this product over all k to obtain $y(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k)$; in practice, $x(k)$ and $h(k)$ will often be of finite duration, thus limiting the range of summation
6. Repeat for all values of n. Again, because the two sequences are of finite duration, this will limit the range of values of n for which $y(n)$ has to be evaluated.

Note: As a general rule, given two finite-duration sequences of length N_1 and N_2 samples, the sequence resulting from their convolution will have $N_1 + N_2 - 1$ non-zero samples.

Exercise 1.2

Calculate the convolution of the following two causal sequences, where each sequence starts at n=0):

$x(n) = \{1, 1, 1, 1, 1\}$, $h(n) = \{2, 1\}$

Matlab Example 1.3

Calculate the convolution of the following two causal sequences, where each sequence starts at n=0):

$x(n) = \{0.5, 0.5, 0.5\}$, $h(n) = \{3, 2, 1\}$

Suggested solution

```
% specify waveforms
x = [0.5 0.5 0.5];
h = [3, 2, 1];
% do convolution using Matlab "conv" function
y = conv(x, h);
% plot
figure;
stem(y);
grid on;
```

Exercise 1.3

Calculate the impulse responses of each of the filters described by the difference equations in Exercise 1.1 above.

In Section 1.6.3, digital filters were classified into non-recursive and recursive types, depending on whether feedback was present. An alternative classification based on the impulse response is the following:

- Finite Impulse Response (FIR). As the name suggests, these are digital filters for which the impulse response is of finite duration, i.e. $h(n)$ has a finite number of non-zero samples.
- Infinite Impulse Response (IIR). These are filters for which the impulse response is of infinite duration.

Generally-speaking non-recursive filters are FIR in nature, for example, Figure 1.8(a) above (why?), while recursive filters are IIR (Figure 1.8(b)). However, there are some exceptions to the latter. For example, it is possible to design recursive filters that have a finite-duration impulse response.

Exercise 1.4

Show that the samples of the impulse response of an FIR filter are the same as the digital filter coefficients.

1.6.5 Stability of Digital Filters

In this section, we briefly examine the issue of stability of digital filters. First of all, we want to define a *bounded* signal. A digital signal $x(n)$ is bounded, if there exists some finite number M for which

$$|x(n)| < M < \infty$$

Stability of a digital filter may be defined in a number of ways:

- A digital filter is Bounded Input Bounded Output (BIBO) stable if a bounded input sequence $x(n)$ produces an output sequence $y(n)$ that is also bounded, i.e.

$$|y(n)| < K < \infty$$

- Alternatively, since the response of the filter is critically dependent on the impulse response, we require that the impulse response be absolutely summable, i.e.

$$\sum_{k=0}^{\infty} |h(k)| < \infty$$

  where, as usual, we assume a causal impulse response.

In practice, we normally want systems that will remain stable for all the possible operating conditions of the system.

Exercise 1.4

Draw the following impulse responses, and state if the digital filters to which they correspond are (i) causal or non-causal, (ii) stable or unstable.

$$h_1(n) = \left(\frac{2}{3}\right)^n u(n)$$

$$h_2(n) = \left(\frac{3}{2}\right)^n u(n)$$

$$h_3(n) = \left(\frac{2}{3}\right)^{-n} u(-n)$$

$$h_4(n) = \left(\frac{3}{2}\right)^{-n} u(-n)$$