5. The Discrete Fourier Transform

5.1 Introduction

In Section 3, we studied Fourier analysis of discrete-time signals and systems, and looked at the discrete-time Fourier Transform, defined by the following equation:

$$X(\theta) = \sum_{n=-\infty}^{\infty} x(n)e^{-jn\theta}$$

When we calculated the Fourier transform of a discrete-time signal (this signal could be the impulse response of a system, in which case the Fourier transform gives us the frequency response), we saw that it is a continuous function of θ, and furthermore, it is a periodic function with period equal to 2π (or f_{samp} in Hz). While these results are very important, in practice, we generally want to use a computer or embedded microprocessor to calculate Fourier transforms, so strictly speaking, we cannot calculate a "continuous" function of frequency. Instead, we calculate the Fourier Transform of a signal at certain "discrete" frequency points. Furthermore, the definition of the Fourier transform given above assumes that we have knowledge of the discrete-time signal in its entirety (i.e. the summation ranges from $n = -\infty$ to $+\infty$). Again, however, in practice we may only have access to a finite amount of the signal (especially in "real-time" applications, where samples are processed "on the fly"). Therefore, we need to limit the summation to a finite range of n.

These issues lead us to the definition of the Discrete Fourier Transform (DFT), which is the primary topic of this section. We will also examine the use of the DFT for spectral analysis of signals, and will also look at some practical issues relating to this topic.

5.2 Definition of the DFT

Starting with the definition of the Fourier transform from Section 3

$$X(\theta) = \sum_{n=-\infty}^{\infty} x(n)e^{-jn\theta}$$

we note that we can calculate the Fourier transform of $x(n)$ at an arbitrary frequency θ by simply inserting that value of θ into the equation above. In the case of the Discrete Fourier Transform (DFT), we calculate $X(\theta)$ at N values of θ that are evenly spaced between 0 and 2π, i.e.

$$\theta_k = \frac{2\pi}{N}k, \qquad k = 0,1,\ldots,N-1$$

Furthermore, we generally use the DFT to carry out spectral analysis on blocks of the signal of length N, hence, we can write the DFT as:

$$X(k) = X\left(\frac{2\pi}{N}k\right) = \sum_{n=0}^{N-1} x(n)e^{-jn\left(\frac{2\pi}{N}k\right)} \qquad \text{for } k = 0,1,\ldots,N-1$$

Note that, according to the definition of the DFT, N samples in the time-domain yields N samples in the frequency domain.

The frequency resolution of the DFT can be written as:

$$\Delta\theta = \frac{2\pi}{N} \qquad \text{radians}$$

$$\Rightarrow \Delta f = \frac{f_{samp}}{N} \ \text{Hz}$$

where f_{samp} is the sampling frequency.

The Inverse DFT is defined by:

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k) e^{jn\left(\frac{2\pi}{N}k\right)} \qquad \text{for } n = 0,1,\ldots,N-1$$

This has essentially the same form as the Inverse Fourier Transform we studied in Section 3, except that the integral has been replaced by the summation (because the spectrum is "discrete"). The properties of the DFT and Inverse DFT (linearity, convolution etc.) are essentially the same as for the Fourier Transform, and will not be repeated here.

As noted above, the DFT allows us to calculate the Fourier transform of a discrete-time signal in "practical" situations. Note that the maximum frequency at which the DFT is calculated is equal to 2π (strictly speaking, $2\pi - 2\pi/N$, because the range of k is chosen to be from 0 to N-1, however, remember that because of the periodicity of the spectrum of a sampled signal, in the frequency domain, a frequency of 2π is the same as a frequency of 0, so we're not "losing" anything). This suggests that what the DFT does is to "sample" one period of the continuous Fourier transform $X(\theta)$ at defined values of θ. While in principle, the range of k in the DFT equation could be extended beyond N-1, this would not yield any additional information – in the same way that calculating $X(\theta)$ for values of θ greater than 2π does not yield any additional information. Note that, apart from the fact that it is a sampled version of the Fourier transform, the DFT is handled in exactly the same way, i.e. the DFT is still, in general, a complex function of frequency (strictly speaking, k), the magnitude spectrum and phase spectrum can be obtained in the usual way etc. Also, the same symmetry that exists in the Fourier Transform also exists in its discrete version. Furthermore, while we mainly talk about using the DFT for "spectral analysis of signals", it can also be used for system analysis; where the signal being analysed is the impulse response of a system, the DFT can be used to give us the frequency response.

In Section 3, we also noted that the Fourier Transform can also be obtained by evaluating the z-transform on the unit circle in the z-plane, i.e. for $z=e^{j\theta}$. By the same token, the DFT can be obtained by evaluating the z-transform at *specific* values of z:

$$z_k = e^{jk\frac{2\pi}{N}}, \qquad k = 0,1,\ldots,N-1$$

A note about terminology

In Section 3, we referred to the function $X(\theta)$ as the "Fourier Transform of the discrete-time signal" $x(n)$ – remember that this is a continuous function of θ (even though the signal being transformed is sampled). On the other hand, the DFT is a

"sampled" version of this continuous function of frequency (at least over one period of it). To (hopefully) avoid confusion, we will sometimes refer to the continuous function $X(\theta)$ as the "Fourier Transform of a Discrete-Time Signal", with the acronym FTD, in order to distinguish it from the DFT.

5.3 Frequency Resolution

Since the frequency resolution is controlled by the number of time-domain samples used (N), a higher-resolution analysis may be carried out by increasing N. This may be done in either of two ways. Firstly, we can simply use more samples of the time-domain signal, and increase N in this way. However, there are times when this is undesirable, and we will return to this point later. Secondly, we can increase N by adding extra zero-valued samples to the block of samples of $x(n)$ being input to the DFT – this is called zero-padding.

It is useful to recall the "intuitive" explanation of the Fourier Transform (and indeed, the Fourier Series you would have studied in *EE308 Signals and Communications*). Fundamentally, the process of calculating the Fourier transform at a particular frequency amounts to "correlating" the time-domain signal with a sinusoid (or complex exponential) of that particular frequency. This same principle holds whether we are talking about a continuous-time signal (periodic or aperiodic) or a discrete-time signal. In the case of the DFT, for each frequency point k, we are correlating the block of samples of $x(n)$ with a sinusoid

$$e^{-jn\left(\frac{2\pi}{N}k\right)}$$

with frequency $k2\pi/N$, i.e. the frequencies are harmonics of a fundamental equal to $2\pi/N$. The correlation is done by multiplying the block of samples with a block of samples of the complex exponential, and then summing across n. If we increase the block length from N to (say) $2N$ through zero-padding, the extra zero-valued samples do not add "extra" terms to the summation equation in the DFT (we still only have N non-zero valued samples in the block), however, we are now correlating the block of samples of $x(n)$ with complex exponentials with frequency $k2\pi/2N = k\pi/N$. We've halved the "fundamental frequency" but we're now calculating twice the number of frequency points, i.e. $k = 0, 1, \ldots, 2N\text{-}1$. In other words, we've doubled the frequency resolution of the Fourier analysis. The variable k is often referred to as the frequency sample number or "frequency index".

Example 5.1

A signal $x(n)$ consists of four components with frequencies 100, 300, 310 and 500 Hz. The sampling frequency is 10 kHz. Using Matlab, calculate and plot the magnitude spectrum of the signal for a block of 800 points, and show the effect of using zero-padding to increase the block length to 2400 points (i.e. a factor of 3).

Note: Matlab includes a function called `fft` which calculates the DFT of a block of samples using an algorithm called the Fast Fourier Transform (more later) – this is simply a very efficient ("fast") version of the DFT. The function includes a facility for zero-padding where necessary.

A plot of the magnitude spectrum of the signal for an 800-point DFT is shown in Figure 5.1 below. Note that there are clear peaks in the spectrum corresponding to the 100 Hz and 500 Hz components, however, there is only a single peak around 300 Hz, despite the fact that the signal contains two separate components at 300 Hz and 310 Hz.

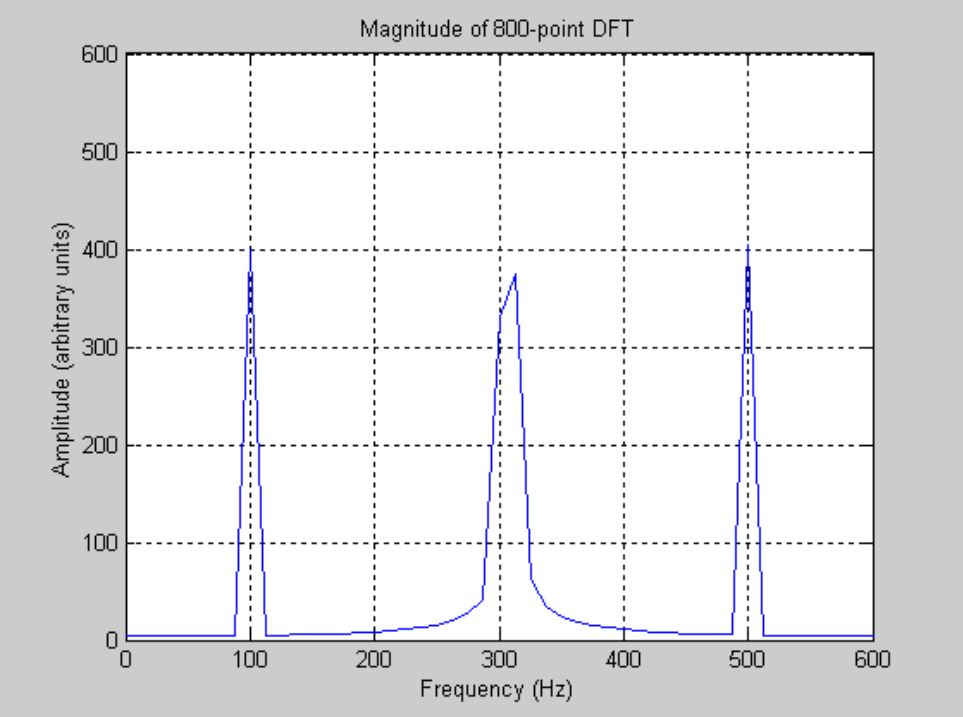

Figure 5.1. Magnitude spectrum of signal in Example 5.1, using 800-point DFT.

Figure 5.2 shows the magnitude spectrum of the signal using a 2400-point DFT:

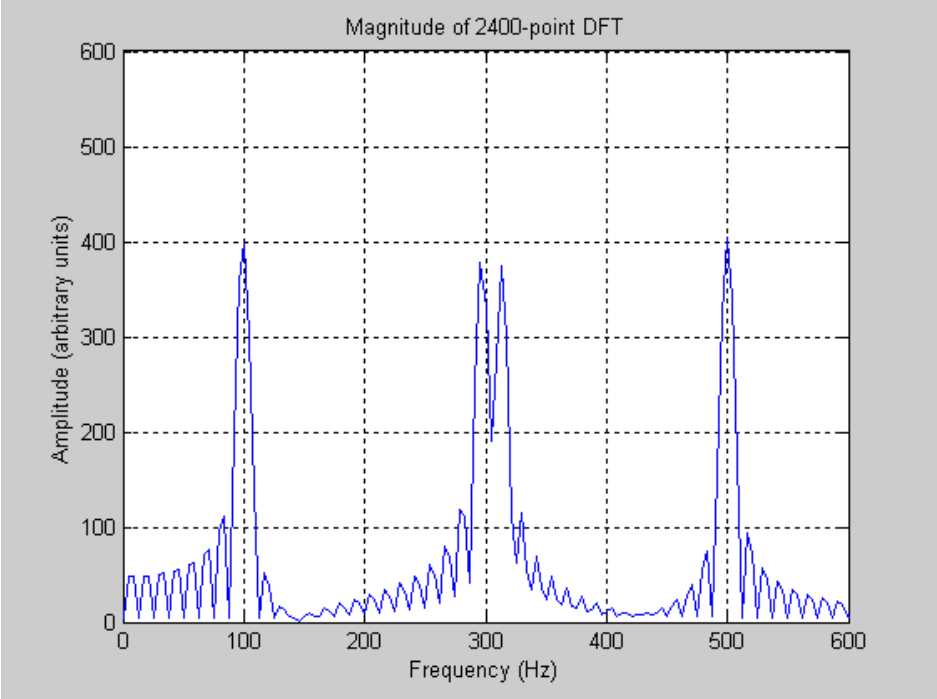

Figure 5.2. Magnitude spectrum of signal in Example 5.1, using 2400-point DFT.

In this case, two separate peaks are visible around 300 Hz, corresponding to the two frequency components. This is because the frequency resolution of the DFT has been increased by a factor of 3, and hence is high enough to distinguish between the two closely-spaced frequency components in the signal. Note that for the 800-point DFT, the frequency resolution is 12.5 Hz, which is too coarse, while for the 2400-point DFT, the resolution is 4.166 Hz, which is sufficient to resolve the 10 Hz separation.

5.4 The Fast Fourier Transform (FFT)

If we look at the equation for the DFT:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-jn\left(\frac{2\pi}{N}k\right)} \qquad \text{for } k = 0,1,\ldots,N-1$$

we see that the calculation of the each of the N frequency points involves the multiplication of each time-domain sample with a sample of a complex exponential, and summation of these products. In the most general case, $x(n)$ may actually be a complex function of time, in which case we have a summation of complex products $x(n)e^{-jn\left(\frac{2\pi}{N}k\right)}$, where each complex multiplication is the equivalent of four real multiplications. However, most practical signals of interest are real, so the product $x(n)e^{-jn\left(\frac{2\pi}{N}k\right)}$ is the multiplication of a real sample sequence by a complex exponential, which reduces to two real multiplications for each value of n.

From the point of view of implementation (in hardware and software), we are generally very interested in the computational complexity of an algorithm (because this translates directly into cost). In the case of DSP algorithms, the bulk of the processing tends to be numerical (rather than "control logic"), so this tends to dominate complexity. Furthermore, estimates of computational complexity generally concentrate more on the number of multiplies that are needed (rather than additions and subtractions), since multipliers are generally much more complex to implement than adders, particularly if the algorithm is being implemented in digital hardware such as FPGA or ASIC (however, this is something of a simplification, and we will return to this later in the course). Hence, the computational complexity of an N-point DFT can be roughly estimated to be $2N^2$ real multiplications (while the number of additions required is $2N(N\text{-}1)$).

Note: As with Fourier analysis for continuous-time waveforms, some efficiencies are possible in the case where the signal $x(n)$ exhibits odd or even symmetry, in which case the amount of computation needed is reduced. However, the most general case is where no such symmetry exists, so we will use this "worst-case" scenario.

For even a modest size of DFT, the computation can be quite significant. However, it turns out that there is a great deal of redundancy in the implementation of the DFT (and the Inverse DFT). This is largely because of redundancy in the calculation of the complex exponential term in the DFT equation:

$$e^{-jn\left(\frac{2\pi}{N}k\right)} = \cos\left(n\frac{2\pi}{N}k\right) - j\sin\left(n\frac{2\pi}{N}k\right)$$

Note that both k and n vary from 0 to N-1, so, as they vary, each of the cosine and sine terms is calculated over and over again (in fact, it can be easily shown that there are only N unique values of sine and cosine required, because of the periodicity of these functions).

The inherent redundancy in the DFT is exploited in an efficient algorithm known as the *Fast Fourier Transform* (FFT). A number of different variants of the FFT exist, for different applications, but they all follow the same basic format. It is important to note that the FFT is simply an efficient way of calculating the DFT, and is not a "new" algorithm – it will give the same results as a "straightforward" implementation of the DFT, only a lot more quickly. In particular, it can be shown that an N-point FFT requires on the order of $2N\log_2(N)$ real multiplies (assuming real $x(n)$). Comparing this with the estimated complexity of the DFT, it can be seen that the saving in computation is on the order of $N/\log_2(N)$. When N is large, this can be very significant.

While the above discussion has focused on the DFT, note that because of the similarity between the DFT and Inverse DFT equations, the same argument applies to the Inverse DFT (there is also an Inverse FFT algorithm).

5.5 Short-time Spectral Analysis

As noted above, the DFT (or FFT) processes data in the form of blocks of length N samples. However, in the real world, most signals of interest have "indeterminate" lengths, so it's difficult to choose a value for N such that the "entire" signal is included (and obviously, taking a value of $N = \infty$ is not feasible). For example, spectral analysis forms an important element of many algorithms for speech processing used in mobile telephony, and clearly, the hardware implementing the spectral analysis has no way of knowing how long a telephone call is going to last. Therefore, it is common for real-time processing of this nature to be carried out by processing the signal in "blocks" (also called "windows" or "frames") of a given length, where the chosen window length is application-dependent. This gives rise to the concept of "short-time" spectral analysis, where a "short" segment of a signal (N samples long) is taken and processed with the DFT, thus giving rise to a spectrum which, technically, only characterises the signal for that time window. For example, in speech processing, a sampling rate of 8 kHz is quite common, and block lengths on the order of 10 to 30 msec are frequently used. This means that spectral analysis would be done on blocks of length 80 to 240 samples (speech processing will be covered in more detail later in the course). Suppose we choose a block length of 128 samples (16 msec) and carry out spectral analysis on the speech signal. This means that we get a "snapshot" of the signal spectrum once every 16 msec, and a series of these snapshots allows us to characterise the signal spectrum over a very long period of time. Of course, we've already noted that the block length (128 in this case) determines the frequency resolution, so in this example, we would have Δf = 62.5 Hz. For some applications, we might need greater resolution, so as noted above, we need to increase the number of samples processed by the DFT. Suppose we want to reduce Δf by a factor of 2. One way of doing this would be to double the block size to 256 samples of speech, however, this would mean we would be taking "snapshots" only once every 32 msec, i.e. we have halved the time resolution. For various reasons, this can be undesirable in speech processing – which is why we often use the alternative method of zero-padding to increase the frequency resolution in the DFT.

Aside: In some applications, greater *time resolution* is also required, and in this case, a common technique is to overlap successive frames by some percentage, i.e. some samples are used in the generation of more than one spectrum. For example, suppose we have blocks of 128 samples, and we overlap by (say) 50%. Then, if we start off with a sample index of 0, the first DFT will be carried out on samples 0 to 127, the second DFT will be carried out on samples 64 to 191 (i.e. samples 64 to 127 are "re-used"), the third DFT will process samples 128 to 255 etc. Overlapping and zero-padding can be "combined" in various ways to increase both time and frequency resolution.

While we are concerned with spectral analysis in this section, the concept of "short-time" analysis also applies to any other form of analysis that we might be interested in doing – the important point to note is that short-time analysis is generally needed for processing "practical" signals (e.g. speech, audio etc.).

5.6 Periodicity Assumption of the DFT

Note that the DFT calculates the spectral content of a block of N samples of a signal at specific frequencies equal to $k2\pi/N$. Put another way, these frequencies can be viewed as harmonics of some fundamental frequency $2\pi/N$. Thus, the DFT calculation is making an "assumption" that the original signal $x(n)$ is periodic (like the Fourier Series), and further, that the block of N samples we are using is equal to one period of the waveform (or an integer number of periods). This can also be seen from the definition of the Inverse DFT given in Section 5.2 above – if this equation is used to calculate values of $x(n)$ for values of n outside the range $0 \le n \le N-1$, it can be seen that the values form a periodic sequence.

Clearly, we cannot guarantee that this will be the case with signals of practical interest. Even for simple signals like a single sinusoid, this will only be true if the block length N is equal to an integer number of periods of the sinusoid.

Example 5.2

Figure 5.3(a) shows the magnitude of the DFT of a block of 500 samples of a sinusoid of frequency 100 Hz (the sampling rate is 10 kHz). Figure 5.3(b) shows the magnitude spectrum of a 480-point DFT of the same signal.

In the first case, the block of 500 samples contains exactly 5 periods of the waveform, so the periodicity assumption of the DFT is satisfied, and there is a single "peak" in the spectrum (corresponding to $k = 5$; note that the x-axis is frequency index in this case). However, for the 480-point DFT is only taking 4.8 periods of the 100 Hz sinusoid, which is not an integer number. Put another way, the DFT "thinks" that 480 points is equal to one period, so it thinks the waveform consists of a series of blocks of these 480 points – which will, of course, contain a discontinuity (because one fifth of a period of the 100 Hz sinusoid is missing from the end). This causes the "spreading" or "leakage" of energy across multiple values of k (this is also apparent in Figure 5.1). This leakage may cause problems in interpreting the results of spectral analysis, so methods of minimising its effect are important – this is the subject of the next sub-section.

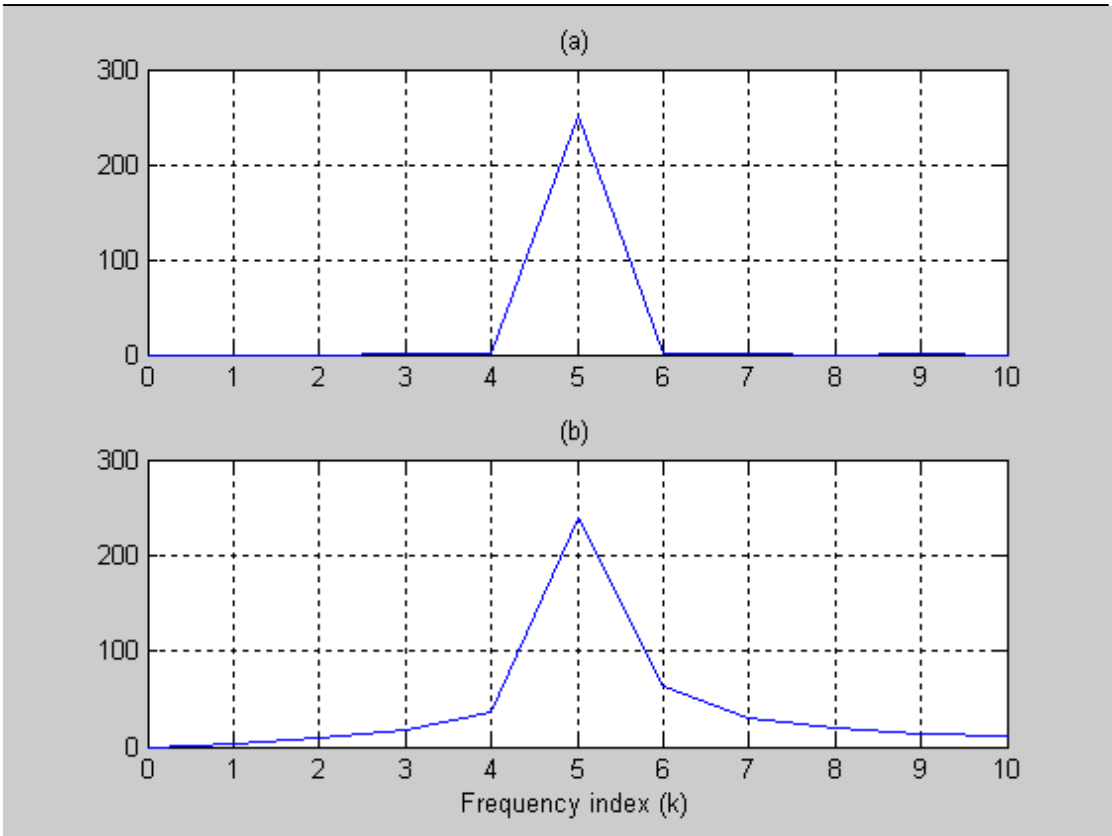

Figure 5.3. Magnitude spectra of (a) 500-point DFT and (b) 480-point DFT, of a 100 Hz sinusoid (the sampling rate is 10 kHz).

5.7 Windowing

We have already seen that the DFT processes data in the form of blocks of length *N* samples, even where each block may be part of a much longer time-domain signal. We have also noted that, in general, the signal in any given block of length *N* samples will not be truly periodic, therefore spectral leakage will occur. One way of looking at this is to note that the "extraction" of a block of *N* samples can be done by multiplying the much longer original by a rectangular "window" (positioned at the point in time where we want to do the analysis). This is illustrated in Figure 5.4, where an arbitrary signal of length 250 samples (actually part of a speech signal) is shown, along with a rectangular window "superimposed" on the signal at an arbitrarily chosen point. Figure 5.4(b) shows the resulting block of N samples that have been extracted by the "windowing operation".

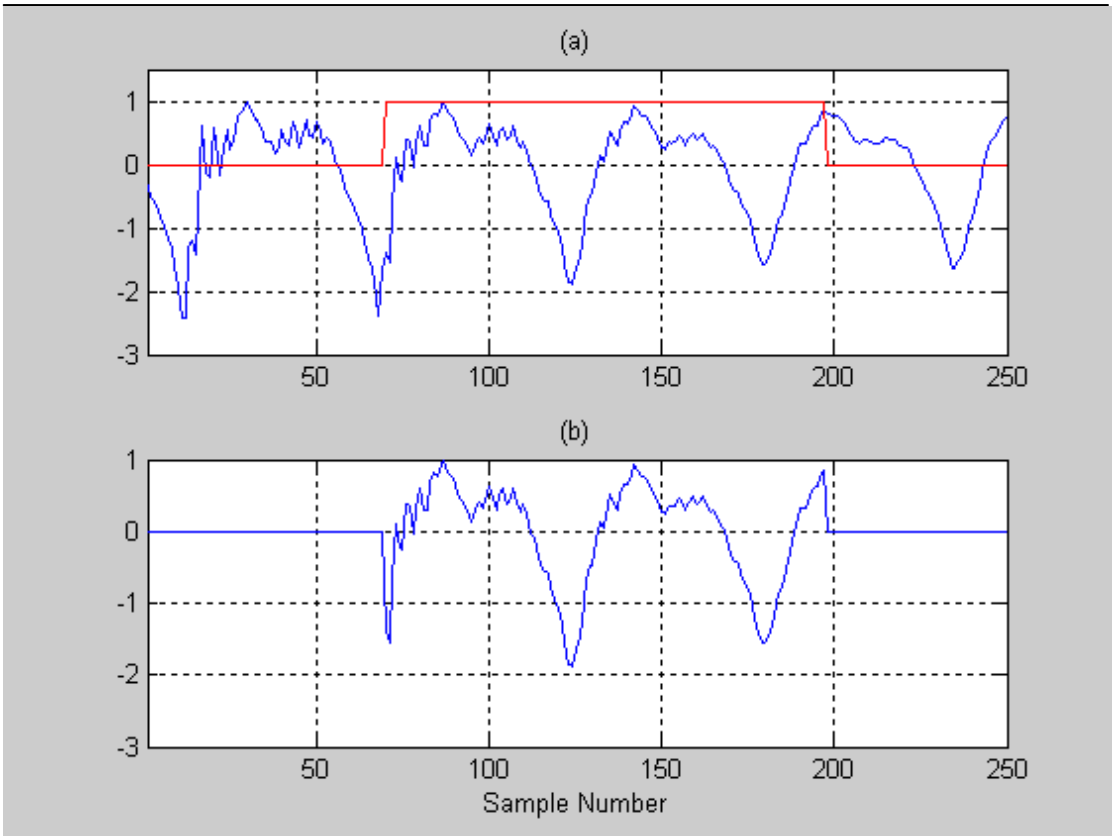

Figure 5.4. (a) Plot of speech signal and 128-point rectangular window, (b) resulting block of 128 samples (not e the "discontinuities" at the window edges).

In this way, the windowing operation can be viewed almost like a "filter", and like any filter, it will affect the spectrum of the original signal. However, unlike a filter in the normal sense of the word, the output is not related to the input through convolution, rather through multiplication. Hence, in the frequency domain, the spectrum of the block of samples can be obtained through convolution (in the frequency domain) of the spectrum of the original (long) signal with the frequency response of the window. Mathematically, if we denote the original signal by $x(n)$, the rectangular window by $w(n)$, and the block of N samples by $x_w(n)$, we can write:

$$x_w(n) = x(n)w(n)$$
$$X_w(\theta) = X(\theta) * W(\theta)$$

For the rectangular window, we note that it is essentially the familiar "gate" function, and hence its Fourier Transform (frequency response) is the *sinc* function, given by:

$$W(\theta) = \frac{\sin\frac{N\theta}{2}}{\sin\left(\frac{\theta}{2}\right)} e^{-j\frac{N-1}{2}\theta}$$

where the complex exponential term gives the phase response while the remaining term gives the magnitude response. The magnitude response of a rectangular window of length 32 points is given in Figure 5.5.

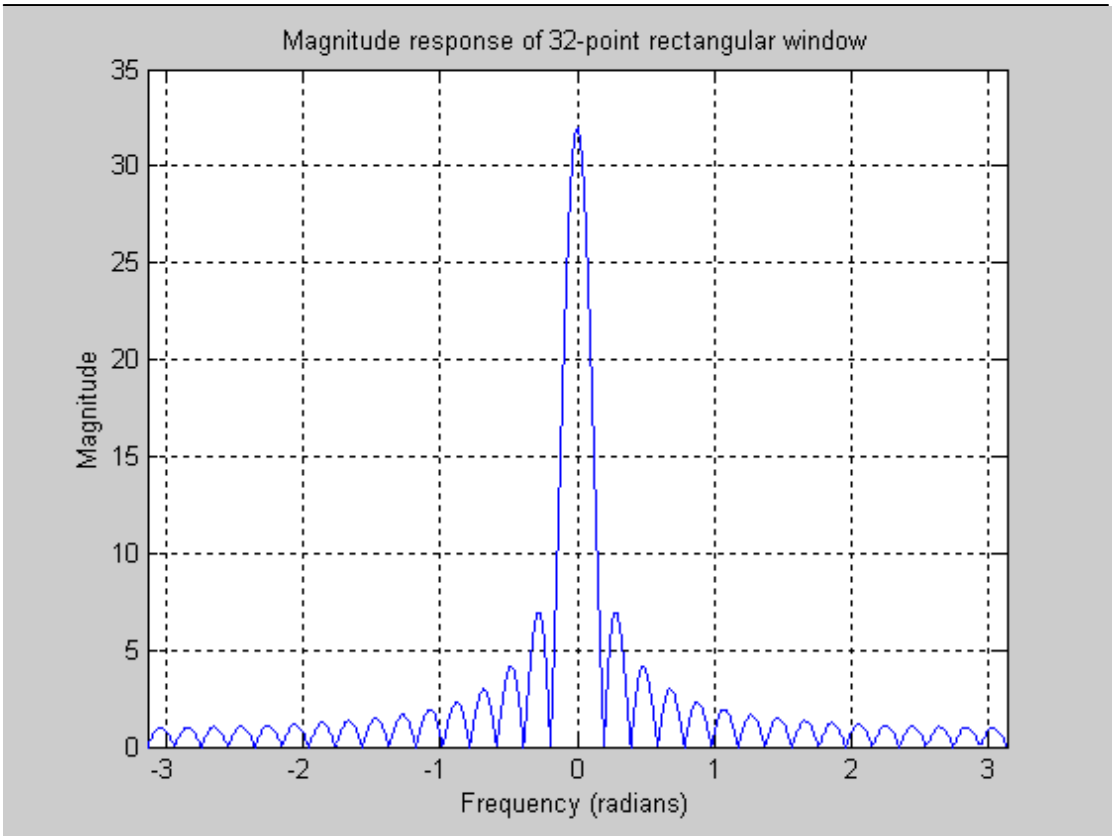

Figure 5.5. Magnitude response of 32-point rectangular window.

The convolution of the original spectrum with the frequency response of the window causes "smearing" of the spectrum. For example, if we take the simple case of a single sinusoid, its "ideal" spectrum would consist of a single non-zero point at the sinusoidal frequency. However, if we window this sinusoid with a rectangular window, the resulting spectrum contains a sinc function centred at the sinusoidal frequency.

Since, for practical reasons, we have to do "short-time" analysis, we are stuck with having to do windowing, however, we are not obliged to use a rectangular window. In fact, the window used in short-time analysis is generally "tapered" to gently reduce the signal sample values to zero at the edges of the window in order to remove the possibility of discontinuities. From the point of view of the signal being analysed, we want to minimise spectral distortion due to the window. This suggests that the "ideal" window is one whose frequency response consists of an impulse at $\theta = 0$, and is zero elsewhere – convolution of such a function with any other function will have no effect. However, all practical windows have a main "lobe" of a certain width, as well as "sidelobes" that reduce in amplitude (like $W(\theta)$ in Figure 5.5 above). Hence, when choosing a window we would like to minimise the width of the main lobe, while minimising the amplitude of the sidelobes; however, these requirements are conflicting and design of a window for a particular application usually involves a suitable tradeoff between main lobe width and sidelobe amplitude.

Some common windows include the following (Matlab functions exist to generate all of these windows):

Bartlett (triangular):

$$w(n) = \begin{cases} \dfrac{2n}{N-1} & 0 \le n \le \dfrac{N-1}{2} \\ 2 - \dfrac{2n}{N-1} & \dfrac{N-1}{2} \le n \le N-1 \\ 0 & \text{otherwise} \end{cases}$$

Hanning:

$$w[n] = \begin{cases} 0.5 - 0.5\cos\left(\dfrac{2\pi n}{N-1}\right) & 0 \le \text{n} \le \text{N -1} \\ 0 & \text{otherwise} \end{cases}$$

Hamming:

$$w[n] = \begin{cases} 0.54 - 0.46\cos\left(\dfrac{2\pi n}{N-1}\right) & 0 \le \text{n} \le \text{N -1} \\ 0 & \text{otherwise} \end{cases}$$

There are several other window types including Kaiser and Blackman.

By way of illustration, a plot of the 128-point Hanning window and its magnitude response are given in Figure 5.6.

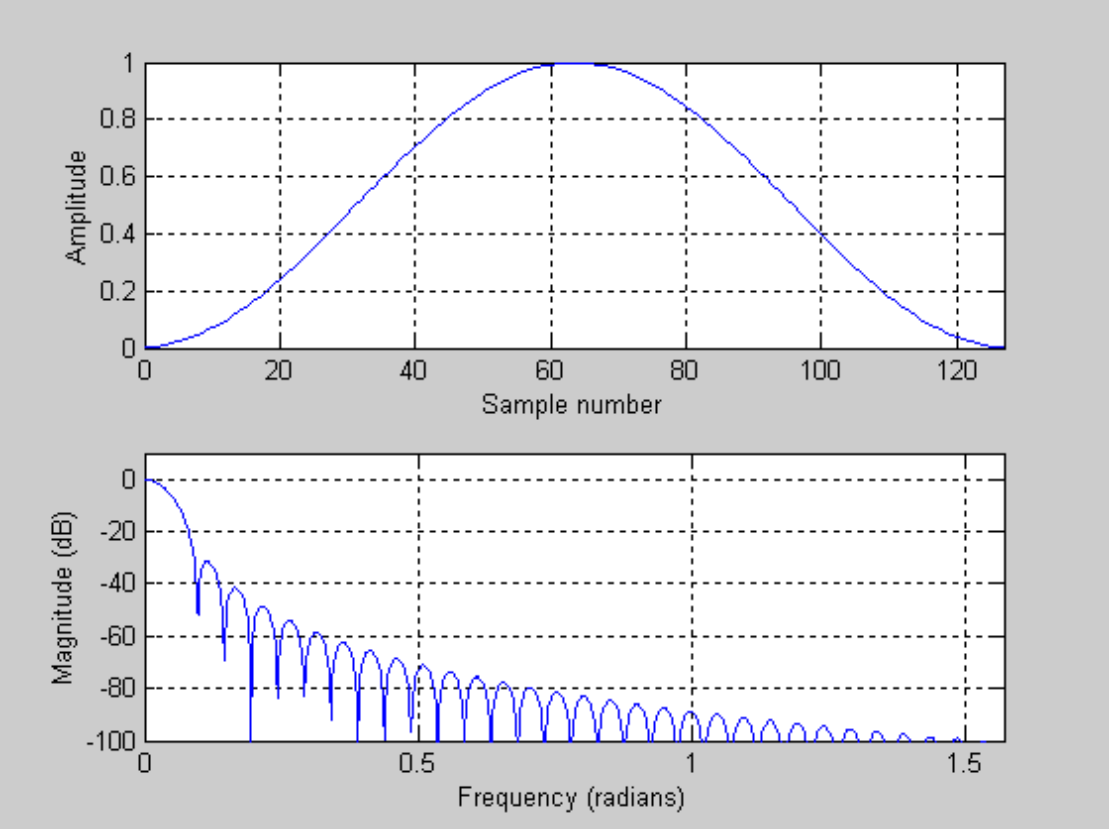

Figure 5.6. Hanning window and its magnitude response.

As noted above, tradeoffs exist between the width of the main lobe, and the amplitude of the sidelobes. The following Table summarises the width of the main lobe (in radians) and the amplitude of the first sidelobe (relative to the peak of the magnitude response) for the different windows:

Window	**Width of main lobe**	**Amplitude of first sidelobe**
Rectangular	$4\pi/N$	-13 dB
Hanning	$8\pi/N$	-32 dB
Hamming	$8\pi/N$	-43 dB
Kaiser	$> 4\pi/N$	-30 to –100 dB (depending on design parameters)

Figure 5.7 shows the block of speech samples from Figure 5.5, before and after windowing by a Hanning window.

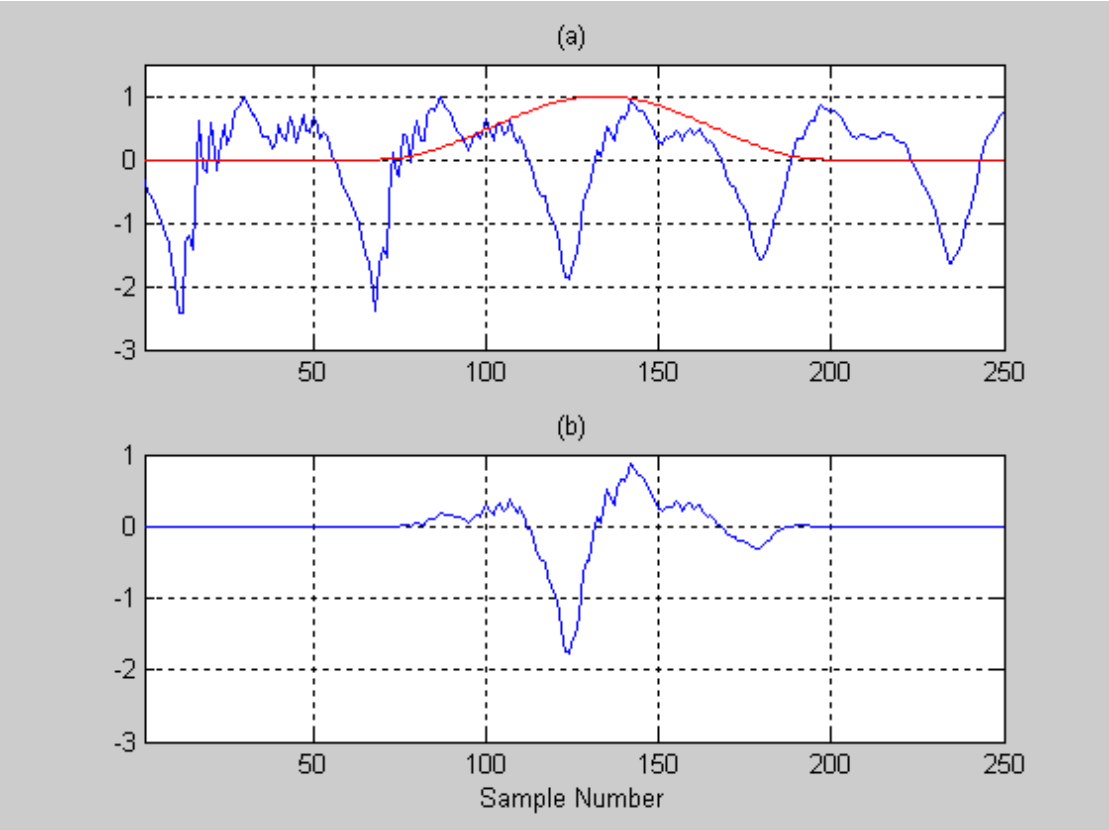

Figure 5.7. (a) Plot of speech signal and 128-point Hanning window, (b) resulting block of 128 samples (note how the signal samples reduce to zero at the window edges).

We noted above that some applications require greater time resolution, and it is common to overlap successive analysis frames by some percentage. Certain

applications also involve re-synthesis of a time-domain signal, and if a "tapered" window is used, it is normal to overlap successive frames by 50% to ensure "perfect reconstruction". If a tapered window is used without overlapping, then significant distortion in the reconstructed signal can result. To illustrate the overlapping, suppose the first sample of the input signal has sample index 0, and we are using frames of 128 samples. Then, the first frame of input samples is indexed from 0 to 127, and we may also index the first frame of output samples from 0 to 127. Because of the windowing, the amplitudes of these samples will reduce to zero at the edges. The second input frame overlaps the first frame by 50%, so it will "re-use" samples 64 to 191 of the input signal. The second output frame from the process must be properly "synchronised" in time, so samples 1 to 64 of the second output frame must be aligned with samples 64 to 191 of the first output frame, and added to these samples. Because both frames have been windowed individually, the net effect is to "cancel out" the reduction in amplitude caused by the windowing. Obviously, not every window can be used for this type of application; in fact, it is a requirement that the sum of multiple windows, where each successive window is "delayed" by the overlap period, sum to 1.

5.8 Use of the DFT in filtering

The primary use of the DFT (normally implemented as the FFT) is in spectral analysis of signals (or analysis of digital filters), as illustrated above. However, the FFT can also be used for filtering. The motivation for this is the observation that convolution in the time-domain corresponds to multiplication in the frequency domain. Implementation of (say) an FIR filter with a very long impulse response in the time-domain may require a great deal of computation. However, if the filtering can be done in the frequency domain, it may result in less computation.

A simplified view of the process in this case would be as follows:

- Use the FFT to process a block of N samples of the filter input signal $x(n)$ to obtain $X(\theta)$.
- Multiply $X(\theta)$ by the frequency response of the filter, $H(\theta)$, to obtain the Fourier Transform (spectrum) of the filter output $Y(\theta)$ (for the current block of samples). In theory, this involves complex multiplications, which in practice means 4 real multiplications for each complex one.
- Transform the frequency-domain data back to the time-domain using the Inverse FFT.

The process is illustrated in Figure 5.8:

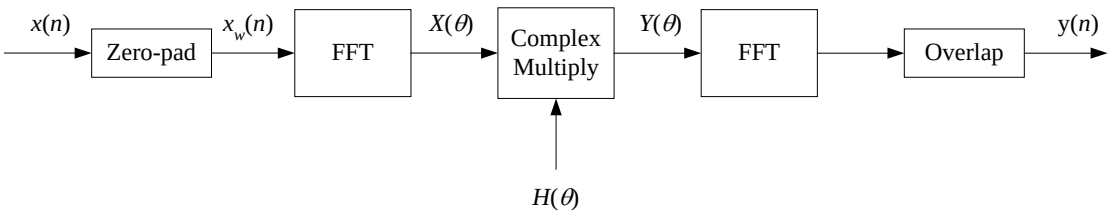

Figure 5.8. Diagram illustrating the procedure for frequency-domain filtering.

To illustrate the potential computational benefits of this approach, let us assume that we have a 512-tap FIR filter. Suppose we choose to use blocks of 256 samples (the greater the block size, the greater the potential saving with the FFT). We will assume that we already have used the FFT to obtain the frequency response of the filter, so we will not count the computation required to do this. Furthermore, we will make the usual assumption that multiplications are more costly than additions (which is a reasonable assumption if implementation is being done using digital hardware).

For the FIR filter, the number of multiplies required to process (say) 1024 samples of the input signal (and produce 1024 samples of output signal) will be 1024*512 = 524288 (since the calculation of each output signal essentially involves the sum of 512 products in the FIR filter). For the frequency-domain filtering approach, the computation required for each frame is as follows (again, concentrating on the multiplications):

- FFT of 256-sample block of input signal = $2N\log_2(N) = 4096$
- Multiplication of $X(\theta)$ by $H(\theta)$ to get $Y(\theta)$ = 256 complex multiplies = 1024 real multiplies
- Inverse transformation of $Y(\theta) = 2N\log_2(N) = 4096$

Therefore, the amount of "raw" computation required per frame is 4096+1024+4096 = 9216 multiplies. For 1024 samples, we have four 256-point frames, so the total computation is 36864 – considerably less than for the FIR filtering approach.

The above description is somewhat simplistic, in that it ignores the assumption of periodicity associated with the DFT (and of course, with the FFT as well). A straightforward implementation as shown above will not result in a *linear* convolution as would be done by an FIR filter, rather it results in a *circular* convolution. Since we want to carry out a linear convolution, some additional care is required in the choice of the values of N, and the "overlapping" of successive frames to achieve this. However, the computational cost associated with this is relatively light. In fact, even if we double the computational load of the FFT approach calculated above, we still have considerably less computation than for the FIR filtering approach.

This general concept of frequency-domain filtering using the FFT is often referred to as fast convolution, while the specific method for processing aperiodic input signals is called the "overlap-add", or "overlap-save" method.