Worksheet 4 — Digital Signal Processing

1 Introduction to Digital Signal Processing

The new and fast growing field of digital signal processing (DSP) is concerned with the processing of signals that have been converted into digital data. DSP techniques have found application in many areas including image processing, instrumentation and control, data compression, audio, telecommunications and biomedicine to name but a few. Although primarily a subject area of electronic engineering, an understanding of DSP can be extremely valuable to anyone involved in science and technology, including physicists! Modern physics experiments invariably use a *data acquisition* (DAQ) system to collect and record data in digital form. The methods of DSP are then used to extract the required information from the data. This might be done in real-time as the data is acquired or 'off-line' as the first step in the analysis of the data.

1.1 Discrete-time Systems

A signal processor can be pictured as a 'black box' with an input that receives a signal and an output that transmits a version of that signal after it has been transformed in some way. A digital signal processor receives and processes *discrete-time* signals *i.e.* signals that have been *sampled* at regular time intervals[1]. A sampled signal is basically a sequence of numbers and we can write it in the form x_n where $n = 0, 1, 2, \ldots$. Systems that process such discrete-time signals are known as discrete-time systems.

A discrete-time system is said to be *linear* if it obeys the principle of superposition. That is, the response of the system to two or more inputs is equal to the sum of the responses to each input acting separately in the absence of other inputs. For example, if a discrete-time input signal a_n gives rise to the output signal c_n and b_n gives rise to d_n then the input signal $a_n + b_n$ produces $c_n + d_n$. A discrete-time system is said to be *time invariant* if its output does not depend on the time that the input is applied. For example, if the input signal x_n gives the output y_n then the time-shifted input signal x_{n-k} will give the output y_{n-k}. When a discrete-time system is both linear and time invariant its output y_n for an input signal x_n is given by the convolution sum

$$y_n = \sum_{k=0}^{\infty} h_k x_{n-k} \tag{1}$$

[1] signals need not be time based but for the purposes of this document we are going to always talk in terms of time based signals

where h_k is the *impulse response* of the system. If the input signal is in the form of an impulse *i.e.* $x_n = \ldots, 0, 0, 1, 0, 0, \ldots$ the output of the system is h_k (although shifted in time.) Any input signal can be considered as a sequence of impulses with different amplitudes so, by the principle of superposition, the output is a sum of the responses to the individual input impulses. Hence, the output is the convolution of the input signal with impulse response described by Equation 1. It is important to appreciate that the values of impulse response completely define the system. Once you know the impulse response you know everything about the system and can calculate the output for any input signal.

An alternative and complementary way of looking at signals and the response of systems is in the frequency domain. Signals can be described by the amplitude and phase of their frequency components and systems can be described by their frequency response. Converting between time domain and frequency domain descriptions involves the Fourier transform. A time domain signal $x(t)$ and its frequency domain equivalent $X(\omega)$ form a Fourier transform pair and are related by:

$$\begin{aligned} X(\omega) &= \mathcal{F}[x(t)] \\ x(t) &= \mathcal{F}^{-1}[X(\omega)] \end{aligned}$$

where $\mathcal{F}$ represents the Fourier transform and $\mathcal{F}^{-1}$ represents the inverse Fourier transform. In general, $X(\omega)$ is complex and can be written in the amplitude-phase form $|X(\omega)| \exp(i\theta_X(\omega))$ where $\theta_X(\omega)$ is the frequency dependent phase angle. Similarly, a system's impulse response $h(t)$ forms a Fourier transform pair with the system's frequency response or *transfer function*, $H(\omega)$:

$$\begin{aligned} H(\omega) &= \mathcal{F}(h(t)) \\ h(t) &= \mathcal{F}^{-1}(H(\omega)). \end{aligned}$$

In the frequency domain, the output of a system, $Y(\omega)$, is calculated by multiplying the input, $X(\omega)$, by the transfer function $H(\omega)$ *i.e.*

$$Y(\omega) = H(\omega)X(\omega). \tag{2}$$

Since $H(\omega)$ is a complex function it can modify both the amplitude spectrum and the phase spectrum of the input signal. Equation 2 is the frequency domain equivalent of convoluting the input signal in the time domain with the impulse response, as can be seen by taking the Fourier transform of $y(t) = h(t) * x(t)$ where the $*$ operator represents convolution:

$$\begin{aligned} \mathcal{F}[y(t)] &= \mathcal{F}[h(t) * x(t)] \\ \mathcal{F}[y(t)] &= \mathcal{F}[h(t)]\,\mathcal{F}[x(t)] \\ Y(\omega) &= H(\omega)X(\omega). \end{aligned}$$

In this working we have made use of the *convolution theorem* which states that convolution in the time domain is equivalent to multiplication in the frequency domain i.e.

$$\mathcal{F}[x_1(t) * x_2(t)] = X_1(\omega)X_2(\omega). \tag{3}$$

Similarly, multiplication in the time domain is equivalent to convolution in the frequency domain *i.e.*

$$\mathcal{F}[x_1(t)x_2(t)] = X_1(\omega) * X_2(\omega). \tag{4}$$

When thinking about signal processing problems it can be very helpful to keep these two relationships in mind.

1.2 Sampling

Before being processed by a digital signal processor, continuous analogue signals must be converted into a stream of numbers which represent the original signal by the processes of *sampling* and *digitisation*. Sampling involves taking the value of the signal at regular time intervals and digitisation involves converting the sampled signal values into numbers. At both of these stages there is a loss of information and there are resulting side effects. An understanding of the side effects is essential when considering DSP systems.

The sampled signal can be considered as only being non-zero at the regular sampling instants and zero at other times. In the time domain, this is equivalent to multiplying the continuous analogue signal by an infinite sequence of Dirac delta functions spaced by the sampling period $1/f_s$, where f_s is the sampling frequency; let us call this the sampling function. It is interesting to see what this does in the frequency domain. As has already been stated in Section 1.1, multiplying in the time domain is equivalent to convolution in the frequency domain so we need to calculate the Fourier transform of the input signal and the Fourier transform of the sampling function and then convolute them. In general the input signal will consist of a range of frequencies but there will be an upper limit to this range (perhaps because of the limited frequency response of a transducer.) The sampling function is a set of Dirac delta functions as shown in Figure 1. The Fourier transform of this sampling function is an infinite sequence of Dirac deltas spaced by the sampling frequency. Note that only a few of these are shown in the figure. The result of the convolution of these two Fourier transforms is illustrated in Figure 1. Notice that the frequency spectrum of the input signal repeats at multiples of the sampling frequency (only a few are shown on the figure).

This pattern of signal appearing in upper and lower sidebands around repeated harmonics of the sampling frequency, f_s, can be understood as follows. In the time domain, the sampled signal is formed from the multiplication of a continuous signal $d(t)$, by the sampling function, $s(t)$, which is a train of delta functions at interval $1/f_s$. The periodic sampling pulse train, $s(t)$, can be expanded in a Fourier series as

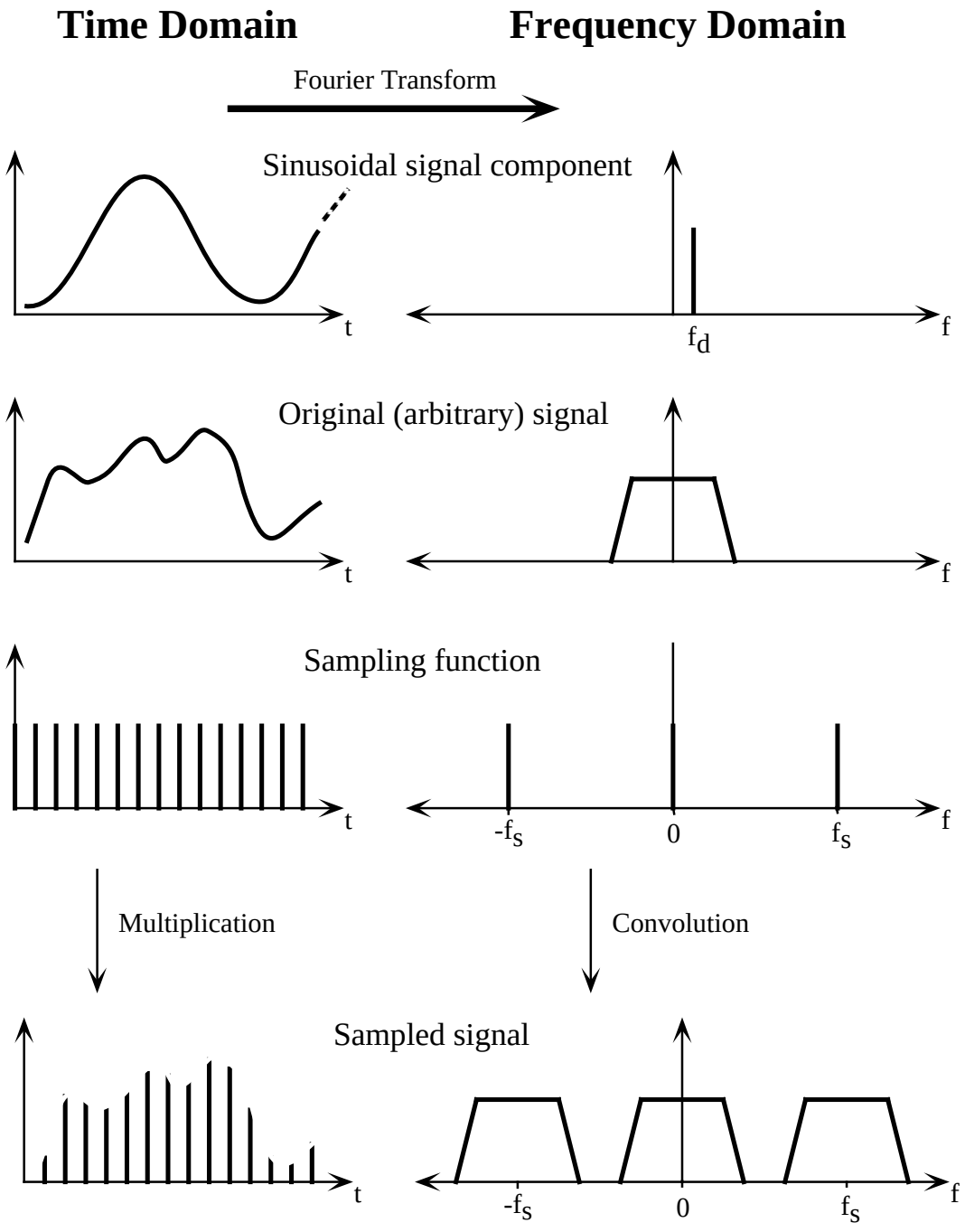

Figure 1: The effect of sampling a signal.

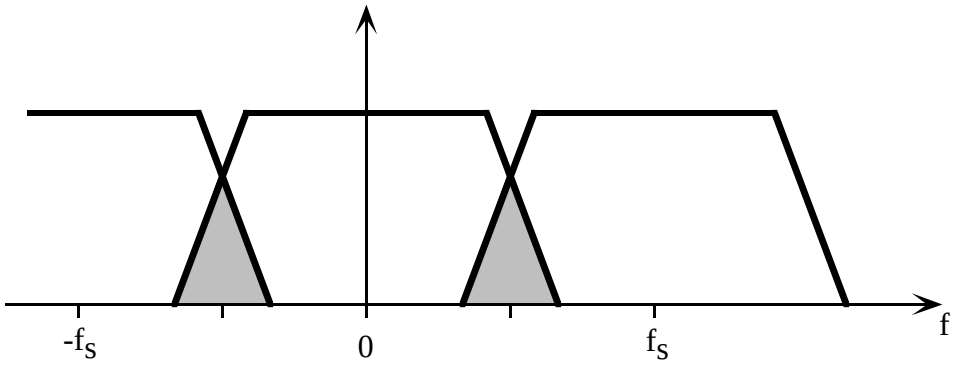

Figure 2: Aliasing in the frequency domain.

$$s(t) = a_0 + a_1 \cos \omega_s t + a_2 \cos 2\omega_s t + \cdots$$

If $d(t)$ is a single sinusoidal component, $\cos \omega_d t$, then the sampled waveform is given by

$$d_s(t) = a_0 \cos \omega_d t + \frac{a_1}{2} \cos(\omega_s - \omega_d)t + \frac{a_1}{2} \cos(\omega_s + \omega_d)t + \frac{a_2}{2} \cos(2\omega_s - \omega_d)t + \frac{a_2}{2} \cos(2\omega_s + \omega_d)t + \cdots$$

where the $n\omega_s \pm \omega_d$ terms follow from $\cos(A \pm B) = \cos A \cos B \mp \sin A \sin B$.

The same expansion around harmonics of f_s applies to every frequency component of $d(t)$, hence the repeated symmetric spectrum illustrated in Figure 1, which extends in both directions from each integer multiple of the sampling spectrum.

Problems occur if the input signal contains frequencies greater than $f_s/2$, the so called *Nyquist frequency*. In this case the repeated spectra start to overlap as illustrated in Figure 2 and it becomes impossible to distinguish between input frequencies greater than $f_s/2$ and those less than $f_s/2$ in the overlap. This effect is called *aliasing*. The conclusion of this analysis is that frequencies greater than $f_s/2$ cannot be recovered once the signal has been sampled at f_s and so the input signal should not contain such frequencies. In practise this is achieved by applying a low pass filter at the input of sampling circuit to remove all frequency components above $f_s/2$. If the signal contains useful information at high frequency then it is necessary to increase the sampling frequency.

The side-effect of digitisation is the addition of quantisation noise to the signal. The size of the noise is determined by the precision of the numbers used to represent the signal. This precision should be made good enough that the added noise is insignificant compared with other sources of noise in the signal.

1.3 The Fast Fourier Transform

The fast Fourier transform (FFT) is very useful tool for estimating the frequency content of signals. The algorithm is highly efficient and using modern microprocessors it is possible to do real-time frequency analysis of signals in the audio frequency range. The FFT is a special form of the more general discrete Fourier transform (DFT). In turn, the DFT is a discrete-time form of the Fourier transform. That is, the input time domain data is in the form of a sequence of discrete values and the output is a set of discrete frequency amplitude-phase values. The DFT and Fourier transform are therefore related but are not exactly equivalent.

The DFT of a sequence of N discrete-time values x_n, where $n = 0, 1, \ldots N-1$, is given by:

$$X_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x_n e^{i2\pi kn/N} \tag{5}$$

where $k = 0, 1, \ldots N-1$. Hence, the DFT returns N complex values which represent the amplitude and phase for the harmonic frequencies $f = kf_s/N$ where f_s is the sampling frequency. The inverse DFT is given by:

$$x_n = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} X_k e^{-i2\pi kn/N}. \tag{6}$$

The FFT is mathematically identical to the DFT but the number of data points is restricted to 2^i where i is an integer. FFTs are much faster because the algorithm takes advantage of computational redundancies in the DFT.

The discrete nature of the DFT results in side effects which need to be appreciated when using the DFT (or FFT) to analyse signals. The first problem is *aliasing* and has already been discussed in Section 1.2. If the input signal contains any frequencies greater than $f_s/2$ they will appear as frequency components less than $f_s/2$. This problem can be solved by increasing the sampling frequency until the frequencies of interest are below the Nyquist frequency.

The second problem occurs because a signal component with frequency not exactly equal to one of the harmonic frequencies $f = kf_s/N$ cannot be properly represented. The result is that its amplitude is shared between nearby harmonics. This effect can be reduced by increasing the number of data points either by analysing more points or, if that is not possible, by adding zero values to the end of the data. This improves the spectral resolution of the DFT by reducing the spacing of harmonic frequencies, Δf, since for an N-point DFT $\Delta f = f_s/N$.

The third problem is *spectral leakage* and is the result of analysing only the finite time interval N/f_s. Effectively, what we have done is multiply a signal of infinite duration by a *window function* to give a signal of finite duration. In the frequency domain this is equivalent to convoluting the frequency spectrum of the signal with the Fourier transform

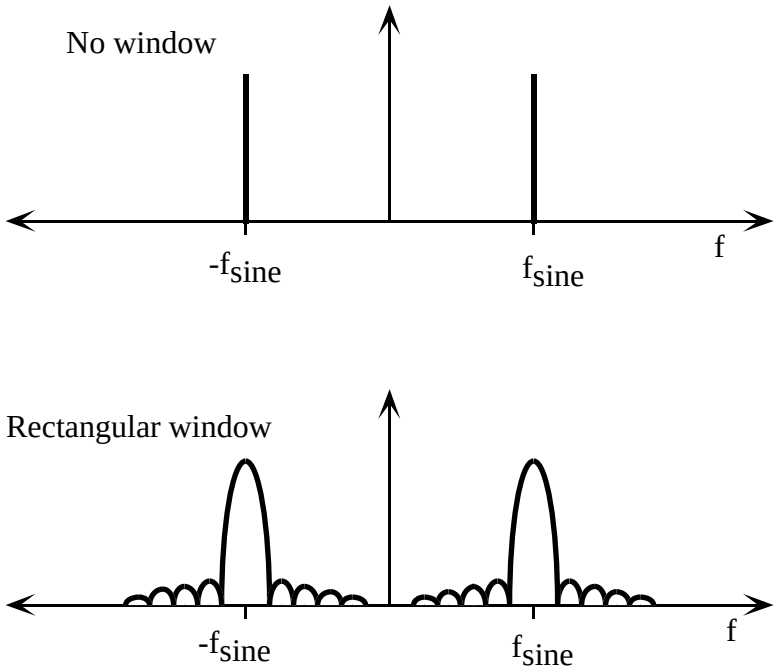

Figure 3: The effect of a rectangular window on a sine wave in the frequency domain.

of the window function. In the simplest case this window function is rectangular and the input signal's spectrum gets convoluted with the sinc function as shown in Figure 3. This function has a finite width and has very long tails so that each frequency component in the signal 'leaks' across several frequency bands. This problem can be improved by using window functions other than a rectangle and by increasing the width of the time window.

1.4 Digital Filters

Digital filters are discrete-time systems that modify the amplitude and/or phase of signals in a frequency dependent way. The reason for using a filter is usually to extract only the frequencies of interest from a signal. Real signals are always contaminated by noise to some extent and filters are needed to remove the noise. The great advantage of digital filtering over using analogue filters is that virtually any kind of digital filter can be realized. The properties of a filter are determined completely by its impulse response and digital filters can be designed to have any arbitrary impulse response.

There are two main classes of digital filter; *finite impulse response* (FIR) filters and *infinite impulse response* (IIR) filters. As shown in Section 1.1 the output of a digital filter can be calculated from its input using Equation 1. If the impulse response of the filter is infinite in length the filter is considered to be an IIR filter. In this case it is impractical to calculate the output directly using Equation 1. However, if the impulse

response is of finite length the filter is a FIR filter and the output can be calculated directly. It is common to refer to the values of the impulse response of a FIR filter as *filter coefficients*.

Designing FIR filters is relatively straightforward. The first step is to specify the required frequency response. A simple and useful example is the ideal low pass filter response $H_D(\omega)$ shown in Figure 4. A filter with this ideal response removes all frequencies above the cut-off angular frequency ω_c. Here, the frequency range has been normalised so that the sampling angular frequency is 2π. Notice also that the frequency response repeats because of the discrete-time nature of the signals. Given the frequency response we need to calculate the impulse response so that we can implement the filter using Equation 1. As has been discussed in Section 1.1 the frequency response, or transfer function, and the impulse response form a Fourier transform pair. The impulse response can be calculated by taking the inverse Fourier transform of $H_D(\omega)$. If we do this we get the result:

$$\begin{aligned} h_n &= \frac{\omega_c}{\pi}\frac{\sin(n\omega_c)}{n\omega_c}, \qquad n \neq 0 \\ &= \frac{\omega_c}{\pi} \qquad\qquad\qquad\quad n = 0 \end{aligned} \tag{7}$$

where n is an integer and $-\infty < n < +\infty$. Immediately we see that we have a problem because an infinite number of values are required. To produce a practical filter it is necessary to use only a 'window' of values of h_n around $n = 0$. This is equivalent to multiplying the impulse response by a rectangular window and the result is that the frequency response deviates from the ideal low pass response. Rather than multiply by a rectangular window, other window functions can be used to reduce some of these problems in the same way that spectra from the FFT can be improved by using a suitable window function.

It is highly recommended that you do some supplementary reading on digital signal processing before attempting this worksheet. Search for keywords "Digital Signal Processing" or "Digital Filters". Some example titles are listed at the end of the worksheet (any one of these should contain useful analyses of sampling and filtering — and there are many similar texts).

2 Exercises

Week 5, Session 1/2

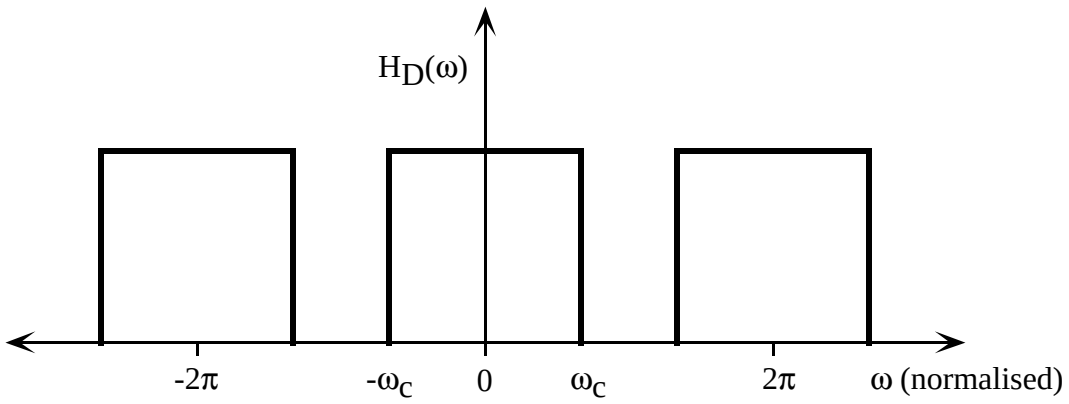

Figure 4: Ideal low pass filter frequency response.

2.1 Sampling

Use Mathcad to simulate sampling of a sine wave of frequency f at a sampling frequency $f_s = 100$ Hz using the form $\sin(2\pi fnT)$ where $f = 10$ Hz and $T = 1/f_s$ and plot the sample values on a graph[2]. Note that sampled data is best stored in an array of samples, than as a function of time or sample number; this approach will be needed for the later exercises. *Describe the appearance of the sampled signal as you increase f from zero to the Nyquist frequency $f_s/2$ and then above the Nyquist frequency.* Compare graphs for frequencies separated by multiples of the sampling frequency ie $(f + N f_s)$ where N is an integer. Also try some values of negative frequencies in the formula as well as positive frequencies. *At each stage try to justify what you observe by thinking in the frequency domain.*

2.2 Discrete-time Systems

We are going to investigate the properties of a discrete-time system which has the following finite impulse response:

$$\begin{aligned} h_0 &= 0.081 \\ h_1 &= 0.247 \\ h_2 &= 0.344 \\ h_3 &= 0.247 \\ h_4 &= 0.081. \end{aligned}$$

Using equation 1, plot the output of the system for a unit impulse input ie $x_k = 1$ where $k \geq 4$ and all other x_k=0. Verify that this is in fact the impulse response given above

[2]This may sound rather imposing, but all it actually means is plot a sine wave with time-steps of 0.01 seconds. After all, all you do when you plot a function is plot it's value at a few discrete points — the 'curve' you see is because with the standard plot options, the discrete points are joined by a line.

shifted in time (remember that Mathcad's arrays run from zero by default.)

Investigate the response of the system to a sampled input sine wave (such as those produced in the first exercise) and changing the frequency of the input between zero and the Nyquist frequency. *Comment on the amplitude and phase-delay response you observe, plotting the amplitude response as a function of frequency.* NB. The extraction of frequency response can be "automated" by making the input and output as 2D arrays with dimensions of sample number and frequency index (where the input data is generated with a frequency taken from an array f_i with values covering the range 0–50 Hz). The maximum point on the output waveform for input frequency i can then be determined using the $max()$ function acting on the i^{th} column of output values. A single column of a matrix can be addressed by the Superscript operator (p 592), where $A^{<i>}$ is a vector identical to the i^{th} column of matrix A. To avoid transient effects, apply the $max()$ function to only the latter part of the output for each frequency, making sure that it is calculated over at least one full cycle of the waveform.

Finally, try a square wave of different frequencies (starting with $f < 5$Hz) and *comment on what you see, particularly how the shape of the output is different to the input (think in terms of the frequeny components of the input.)* A regular square wave with period T can be generated using a construction such as:

$$sq(t) = \text{if}(mod(t, T) < T/2, 1, -1)$$

Week 6, Session 1

2.3 Using the Fast Fourier Transform

Mathcad includes the built-in functions *fft* and *ifft* for performing the Fast Fourier Transform and inverse Fast Fourier Transform respectively. Note the *fft* only works if there are 2^n samples.

Use the *fft* function to analyse a sampled sine wave generated using:

$$x_n = \sin(2\pi f n / f_s)$$

where N is the number of sample points, n is an integer such that $0 \leq n \leq N - 1$, and f is the frequency of the sine wave normalised to f_s (f_s should initially take the value N, so that the sampled input covers 1 second of data). Since the output *fft* is complex, you should look at the *modulus* of the individual components. Start by using $N = 64$ and $f = 6.0$ and *comment on what you see.* Then try $f = 6.1$ and *comment on what you see.* Try changing the number of data points, N, you use (without changing the sample frequency, f_s, and vice versa, change the sample frequency for the same number of data points, and *comment on what you find. Comment on what happens as you slowly change*

f to 6.1 and up to 7.0? Can you explain why integer values of f are free of spectral leakage? Caution: when re-using arrays with different data, be sure that a long array from a previous task does not still contain "old" data — by default the FFT looks at the whole array it was supplied with, even if you have only refilled the first M elements.

It should be clear by now that there are effects which limit the spectral resolution that can be achieved using a FFT. The finite resolution limits our ability to resolve individual spectral components. Look at the FFT spectrum of two sine waves given by:

$$x_n = \sin(2\pi f_1 n/N) + 0.1\sin(2\pi f_2 n/N)$$

where $f_1 = 4.1$ and $f_2 = 7$ using $N = 64$ data points. It is very hard to resolve the smaller sine wave because of spectral leakage from the larger sine wave. So far, we have been using a rectangular window of width N samples. Now see what happens when you multiply the data by the Hamming window function using $x_n = x_n \dot{w}_n$ The Hamming window function is given by:

$$w_n = 0.5 + 0.5\cos(\frac{2\pi(n - N/2)}{N}). \tag{8}$$

Plot the windowed version of x_n and *comment on the difference between this and the original x_n plot.* Then perform the *fft* and *comment on the changes you see in the frequency response and try to explain them.* You might also like to try (optionally) other window functions such as the Blackman window by

$$w_n = 0.42 + 0.5\cos(\frac{2\pi(n - N/2)}{N-1}) + 0.08\cos(\frac{4\pi(n - N/2)}{N-1}). \tag{9}$$

Week 6, Session 2

2.4 Designing a Digital Filter

Use equation 7 to calculate the impulse response for a low pass finite impulse response (FIR) filter with a normalised cut-off frequency $w_c = 2\pi/10$. Use a rectangular window of width 128 points to start with. Arrange the centre of the impulse response h_n so that it is a maximum at $n = 64$. Look at the amplitude response of the filter by using the *fft* function to calculate the transfer function from the impulse response. The logarithmic plotting option will allow you to look at the frequency response above the cut-off frequency. Try changing the width of the rectangular window (always using 2^n samples) containing the impulse response; initially, simply pad-out a longer window with zero values, but then increase the number of non-zero values in the impulse response (re-centering the impulse response in the new window. *Comment on what you see.* Now multiply the impulse response by the Hamming window and *comment on the changes you see and try to justify them.*

2.5 Using a Digital Filter

Finally, add some positive and negative amplitude noise to a sine wave with the *rnd* function then use your FIR filter to reduce the noise. The frequency of the input sine wave clearly should be below the cut-off frequency in order to obtain some output. Plot the noisy input signal and your filtered output on top of each other and *comment on the results* . Use the FFT to analyse the frequency content of the noisy signal *before* and *after* it is filtered by the FIR filter.

Suggested reading:

A. Bateman and W. Yates, Digital Signal Processing Design, Pitman, 1988.
M. Bellanger, Digital Processing of Signals, 2nd Ed., John Wiley and Sons, 1988.
J. Dunlop and D.G. Smith, Telecommunications Engineering, 3rd Ed., Chapman and Hall, 1994.
R.W. Hamming, Digital Filters, 2nd Ed, Prentice-Hall, 1983.
L.B. Jackson, Digital Filters and Signal Processing, Kluwer, 1986.
R. Kuc, Introduction to Digital Signal Processing, McGraw-Hill, 1988.
P.A.Lynn, Introductory Digital Signal Processing with Computer Applications, Wiley, 1989.
A. Peled, Digital Signal Processing: Theory, Design and Implementation, Wiley, 1976.