In this lecture, I will be examining the impact of extracting a portion of a signal and find the spectrum of this extracted portion instead of the signal. This process of taking a portion of signal is known as "windowing".

Then I will discuss the calculation of the Fourier transform of a signal on a computer using discrete method, known as Discrete Fourier transform or DFT.

Finally I will explain how to calculate energy of a signal in the frequency, instead of the time, domain.
Extracting a portion of signal from an everlasting sinusoidal signal is the same as multiplying the everlasting sinewave with a rectangular function as shown.

As mentioned in the last lecture, multiplying in the time domain is equivalent to an operation known as “convolution” in the frequency domain. We will not examine convolution until a later lecture.

It is sufficient for you to know for now that convolution results in the spectrum of the original sinewave, which is two spikes (impulses) at $\pm \omega_0$, being modified by the spectrum of the rectangular function as shown above.

The impact of applying the rectangular window (i.e. function) in the time domain is to spread out the energy of the sinusoid around the impulses.
Let us examine the magnitude spectrum of the rectangular function (in dB). It has a main region where most of the energy lies. This is call the mainlobe of the spectrum. In addition, energy is also spread to its neighbouring lobes, the sidelobes. This is known as “leakages” – energy leaked from the main frequency component.

The reason for such energy spreading to higher frequency is that the rectangular window causes discontinuity (or abrupt changes) at the edges of the window. For example, in Lab 2 exercise 3, you have discovered the difference between the spectrum for a 1000Hz and a 1100Hz sinewave with 1000 samples extracted and sampled to 8000Hz. The 1000Hz has 125 exact cycles in the rectangular window. However the 1100Hz sinewave has 137.5 cycles extracted by the rectangular window.
Instead of using a rectangular window function to “extract” portion of the signal to analyse, it is far better to apply a smooth window function, where the edges of the window tails off gradually. In the lab we, use a Hamming window. Shown here is the plot of the Hanning window, which is really very similar.

There are two impact on applying such a window:

1. It reduces the sidelobes substantially and therefore reduces the leakages.
2. The total energy is reduced, but energy of the signal at each frequency relative to each other is unaffected. In other words, if your unwindowed signal is the sum of two sinusoids \( \omega_1 \) and \( \omega_2 \), the magnitudes of both are reduced by using one of these windows, but their ratio remains the same.
Here is a table of the various windows commonly used as a function of time, the width of the mainlobe (the narrower, the better), the rate at which the sidelobe energy falls away (the faster rate the better), and the height of the first sidelobe.
Next we will examine how the computer calculate the spectrum – using discrete values instead of continuous values.

First remind yourself the definition of Fourier Transform.
When you use the Matlab function `fft(sig)` to compute the spectral component values, you perform the Discrete Fourier Transform (DFT) calculation using a fast algorithm. The fast algorithm is known as Fast Fourier Transform. How this is done is not important. It is sufficient to know that a straight calculation of DFT on a N sample signal has a computation complexity of order \( N^2 \). Using FFT, we reduce this order \( N \log N \). Say, if \( N \) 1000, using FFT instead of DFT would be over 300 times faster!

To compute the FFT or DFT of a time domain signal, we first extract the signal with an appropriate window (N samples in a window of \( T_0 \)), then make up a periodic signal as shown. (This is the formulation used inside the computer algorithm. The signal does not really exists as everlasting periodic signal does not happen in real life!)

The DFT/FFT algorithm basically compute the Fourier coefficients of this repetitive signal using the standard Fourier series equation as shown.

The maths here just shows that the effect of DFT is to take discrete samples of the Fourier Transform of our windowed signal at frequency steps of \( f_0 = 1/T_0 \).
Let us examine DFT from a conceptual, instead of mathematical, point of view. When we take a continuous time signal \( x(t) \) and obtain its spectrum using Fourier transform, we get \( X(\omega) \).

If we sample the continuous time signal, the resulting discrete time signal has a different spectrum as compared to that of the original signal. In fact, the original spectrum is repeated indefinitely at each frequency location \( \pm n f_s \), where \( f_s \) is the sampling frequency and \( n \) is all integers (plus or minus) \( n = \pm 1, \pm 2 \ldots \).

This result is very important. It says that the sampling process modifies the spectrum of the signal in this particular way that has implications to how often we need to sample (i.e. what \( f_s \) to choose), how we can avoid corrupt the signal through the sampling process, and finally, how can we get back the original continuous time signal from the discrete time signal.
The most important message of this slide is how to work out \( N_0 \) and \( T_0 \) for a given sampling frequency \( f_s \) (sampling period \( T = 1/f_s \)).

The frequency step (for each successive frequency bin in the FFT result) = \( 1/T_0 \).

Given that the sampling frequency is \( f_s \), the number of samples you need is:

\[
N_0 = T_0 \times f_s.
\]

For example, in the Lab, we were using \( f_s = 8000 \text{Hz} \) and \( N_0 = 1000x \)
In other words, when we perform DFT, we create a periodic signal using a windowed version of the original (long) signal. Because of this action of constructing a period signal from the original portion of signal, we get a spectrum with discrete frequency bins. This is in contrast with Fourier transform, which produces a continuous frequency spectrum (with ALL frequencies).

DFT is a computer based algorithm, therefore everything must be discrete.

The previous slide shows that the effect of DFT is to SAMPLE the continuous spectrum from the Fourier transform, and the rate of sampling (in frequency) is given by the frequency step $1/T_0 = fs/N_0$.

It is helpful to consider DFT as looking at the spectrum of the original signal through a picket fence. You only see samples of the spectrum at discrete frequencies. It also means that you may miss peak components if they fall between the sampling points (between the fence gaps and blocked by the fence posts).

Consider Lab 1, exercise 2. In this Lab, we sample the signal at 8000 Hz, and we extract 1000 samples to analyse using DFT (actually we used FFT, the fast version of DFT in Matlab).

The frequency resolution (step) is $8000\text{Hz}/1000 = 8\text{Hz}$. The DFT algorithm will generate a spectrum between -4000Hz and +4000Hz in steps of 8Hz. This yields 1000 frequency bins. In our Matlab code, magnitude[1] = 0Hz or dc value. magnitude[2] = 8Hz bin etc.. Now if you use the tuning fork app to generate a 1000Hz tone, you will see a clear spike in magnitude[126] (exactly $125 \times 8 = 1000\text{Hz}$). However, if you use a 1100Hz tone, the signal falls between magnitude[138] and
magnitude[139] (bin 138.5 = 1100/8 + 1). So what you should see is the energy shared in the two neighbouring frequency bins.
Here is a formal mathematical definition of Discrete Fourier Transform. You are not really required to remember this. If you compare this with the formal definition of the Fourier transform (not the discrete version), they look very similar except that

1. x(t) now becomes x[nT] because samples are discrete. T is the sampling interval and T = 1/fs.
2. \( \omega_0 \) is the fundamental frequency as we repeat the extract signals every \( T_0 \). \( \omega_0 = 2\pi f_0 = 2\pi / T_0 \).
3. DFT produces results \( X_r \), the amount of signal at the \( r \)th frequency bin, whose frequency is \( r\Omega_0 \), where \( \Omega_0 \) is the discrete frequency step.
The next three slides are about energy of a signal.

We have seen before in Lecture 1 and Lab 2 Exercise 4 that we can compute the energy of a signal by integrating the square of the signal \(x^2(t)\) over the duration where the signal is non-zero.

Parseval’s Theorem basically says that you can do this energy calculation equally well in the frequency domain. The important result here is the equation:

\[
E_x = \int_{-\infty}^{\infty} |x(t)|^2 dt = \frac{1}{2\pi} \int_{-\infty}^{\infty} |X(\omega)|^2 d\omega
\]

We can compute energy of the signal by integrating over all the spectral frequencies (\(\omega = -\infty \) to \(\omega = +\infty\)) the square of the magnitude of the spectrum. The \(1/2\pi\) is a scaling factor to account for the fact that we using radians/sec as frequency and not Hz (cycles/sec).
Therefore we can view the plot of $|X(\omega)|^2$ as the energy spectral density of the signal. We can then ask the question: “how much energy is between two frequencies?” by integrating under the curve as shown.

We can this energy spectral density because the unit can be seen as energy per unit bandwidth (in Hz).
For real signals, it can be shown that the spectrum is symmetrical about 0, i.e. \( X(\omega) = X(-\omega) \). Therefore we can simplify the calculation of energy between two frequencies \( \omega_1 \) and \( \omega_2 \) as shown.
Here is an example of how to use Parseval’s Theorem to provide useful results.

We are interested in the bandwidth of a signal which contain 95% of the energy. The signal in question is \( x(t) = e^{-at} u(t) \).

This is done by performing FT on \( x(t) \) using the FT table.

Using Parseval’s Theorem we can compute the integral and calculate the total energy of the signal \( E_x \).

The rest is pretty straightforward.