The Discrete Fourier Transform - A practical approach

David Dorran
Technological University Dublin, david.dorran@tudublin.ie

Table of Contents

How to use this series of documents

"Digital Signal Processing Foundations" provides a gentle introduction to the world of DSP. This series of documents deals with topics relevant to DSP and they provide links to online video content in an effort to make some perhaps tricky concepts easier to understand for the reader.

If you are completely new to DSP then I'd recommend you take a look at the foundations document first. Also note, that if you have already read the foundations document you can skip straight to the section entitled "How to use Octave/Matlab's fft function".

I believe that the approach of integrating text with video takes advantage of the unique visualisations that video material has to offer with the more in-depth detail and speed of review that text-based material does. Most of the video material relates to my own youtube channel, which at the time of writing (2021) had over 16,000 subscribers and 3 million views.

I also provide Octave/Matlab code examples throughout the document and I'd encourage anyone who wants to develop practical DSP skills to download Octave, which is available free of charge, and implement your ideas.

My intention is to continue this series so as to deal with the major elements of DSP, such as correlation, the Z-Transform and so on. I'll post updates to this series at http://www.pzdsp.com/docs if you want to check out any new additions.

If you find any of this work useful I'd be most grateful if you could cite the relevant resource when appropriate to provide recognition.

Regards,

David

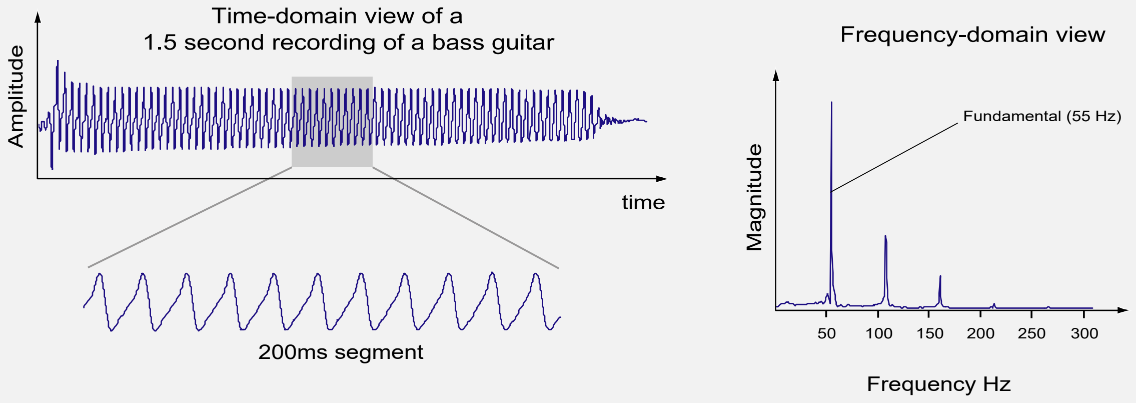
An Introduction to the Frequency-Domain

When someone plays the guitar different sounds are created because the guitar strings vibrate or oscillate at different frequencies. A similar effect can be heard if you stretch an elastic band between your fingers and pluck it and you'd notice that changing either the length or the tension of the band would alter the frequency of the sound since this causes the band to vibrate at a different rate or frequency.

When something is oscillating a repeating pattern is being produced over time. This can be seen with a vibrating elastic band as it moves backwards and forwards through its initial position. Take a look at the following link to clearly see this effect in a slow motion video of guitar strings as they oscillate: pzdsp.com/vid11 .

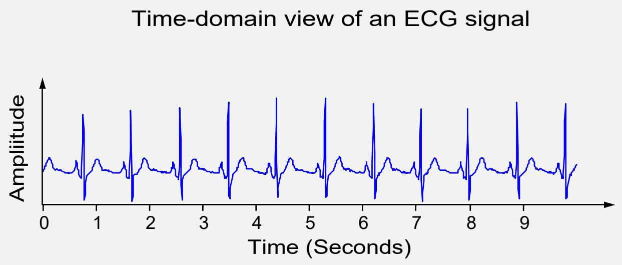
The repeating nature associated with the movement of a guitar string can also be seen in a plot of the audio signal it produces, as shown below, where the amplitude of a bass guitar audio signal is seen to move up and down over time as the strings vibrate. You should note that the rate of oscillation of the string is the same as the rate of oscillation of the audio signal since it is the string vibrations that cause pressure variations in the air which we perceive as sound (The audio recording of the bass guitar signal shown above can be downloaded from pzdsp.com/sig1). The change in air pressure can also be picked up by a microphone and stored on a computer as a discrete signal i.e. a sequence of numbers that were obtained by measuring the sound pressure at regular time intervals.

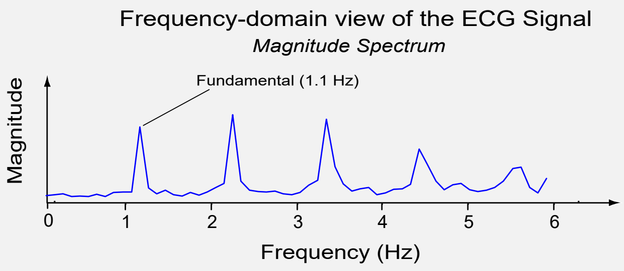
The frequency-domain representation of a signal is a convenient way of showing the oscillation rate associated with a signal, as explained in the following paragraph.



From the figure above, the sound pressure oscillates after the initial 'attack' or transient component at the start of the signal. This plot of pressure variation over time is referred to as a time-domain plot and by looking closely at this plot you can see that the time to complete one cycle of an oscillation is about 1.82 milliseconds (approx. 11 cycles over a 200ms segment). In other words the cycle is repeating about 55 times every second. To the right of the time-domain plot is a plot of the *magnitude spectrum* which is a frequency-domain representation that can be used to quickly determine the rate of oscillations in time-domain signals. The three relatively large 'spikes' shown in the magnitude spectrum represent the fundamental frequency (55 Hz) and the first two harmonics (110 Hz and 165 Hz). You should notice that you can tell the rate of oscillation (55 Hz) quite easily when you look at the signal in the frequency-domain; much more quickly and easily than by analysing the period of the time-domain signal.

This type of repeating pattern doesn't just happen with audio signals and it can be observed in many signals, including those from our heart. Your heart will beat at particular rate, or frequency, depending on what you are doing and your heart rate will increase if you go for a run or cycle. Engineers and scientists (and musicians and doctors!) are often analysing the repeating nature of signals and the frequency-domain view of a signal shows the frequency of the repeating patterns in a convenient graph.





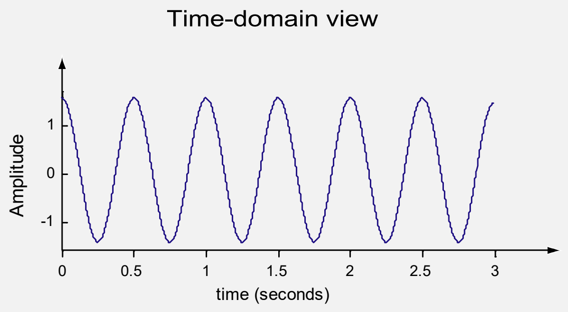
The frequency-domain view of a signal provides another way of analysing a signal which can provide valuable insight into a signals' behaviour. I find it useful to relate this to the way an architect has different drawings of a building depending on who she is dealing with: A client would find it easier to visualise what the building would look like by examining a 3-D view of the building; while a builder would require detailed plans in order to construct the building. Both sets of drawings are representations of the same building and both have their uses. It's the same with the time-domain and frequency-domain views of signals – both represent the same signal and both can be very useful when analysing signals. Here's a link to a video which demonstrates the benefit of both the time-domain view and frequency-domain view of a signal pzdsp.com/vid12.

Frequency-domain graphs of signals are very easy to create using software tools like Octave and Matlab and they make use of Fourier analysis techniques to extract frequency information from a time-domain signal (more on this later!). The basic principle behind all of the Fourier analysis techniques is that any signal can be broken down into a set of sinusoidal signals and this concept is explored further in the next couple of subsections.

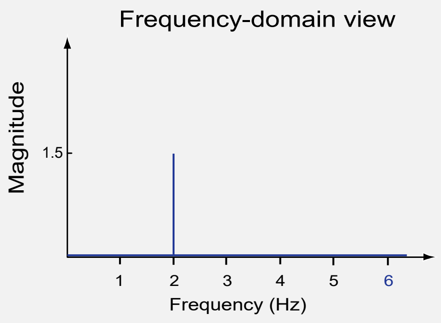
What are sinusoids?

A sinusoid is a waveform that oscillates smoothly over time (see the plot below) and is associated with many signals that occur in nature. For example, when you whistle you create pressure variations in the air which have a sinusoidal shape or of you were to allow an object attached to the end of a spring bounce up and down then the motion of the object would also be sinusoidal (see pzdsp.com/vid13). Even more interestingly it turns out that sinusoids are a fundamental building block of any signal so it's worth spending some time getting used to what they look like and how they can be represented mathematically. This fact was shown mathematically by a French mathematician called Jean Baptiste Joseph Fourier (1768-1830).

There are three features of sinusoidal waveforms that you'll need to be comfortable with to fully appreciate Fourier analysis: *frequency*, *amplitude* and *phase offset*.



A sinusoidal waveform of amplitude 1.5 and frequency 2 Hz



The figure above shows a time-domain plot of a cosine waveform to the left and its corresponding **magnitude spectrum** to the right. From the time-domain view notice that the sinusoids amplitude oscillates between 1.5 and -1.5 which means that the amplitude of the sinusoid is 1.5. You'll notice that the sinusoid is repeating every 0.5 seconds, in other words it has a period of 0.5 seconds, which means that it has a frequency of 2 Hz. I'd recommend you check out the interactive animation at pzdsp.com/sinusoids in order to get a clearer idea about these parameters.

The frequency-domain plot of the sinusoid above shows a single spike at a frequency of 2 Hz. Anytime you have a time-domain plot of a single sinusoid you will observe a single 'spike' in the frequency-domain and the position of the spike on the frequency axis corresponds to the frequency of the sinusoid. The magnitude (height) of the 'spike' is proportional to the amplitude of the sinusoid. You'll see examples of signals with more than one sinusoid present in the next section.

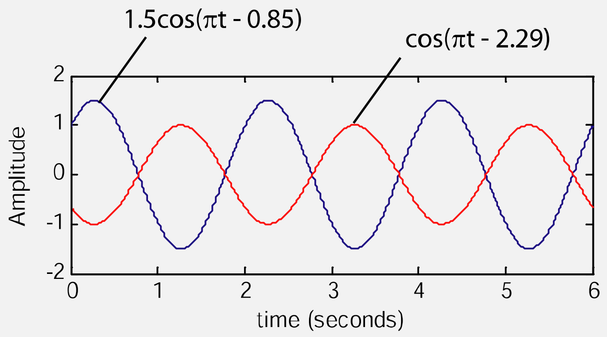
Octave code to create a plot of a sinusoid:

```
A = 1.5;
f = 2;
phi = 0;
duration = 1; %1 second
T = 1/f;
t=0:T/100:duration;
x = A*cos(2*pi*f*t + phi);
plot(t,x)
xlabel('time (seconds)')
ylabel('Amplitude')
```

Before we look at the phase associated with this sinusoid lets first take a look at a mathematical function often used to represent a sinusoid which is shown below:

$$x(t) = A\cos(2\pi f t + \varphi)$$

The A parameter specifies the amplitude of the sinusoid; f specifies the frequency and φ (Greek letter phi) parameter specifies the phase offset (also referred to as the initial phase or phase). The t variable represents time and the mathematical expression is evaluated for a range of values of t in order to create a time-domain signal. So, if you wanted to recreate the plot of the sinusoid shown above you'd substitute A with 1.5, f with 2 and φ with 0 to give $x(t) = 1.5\mathbf{cos}(4\pi t)$, and then you could evaluate this for a number of values of t before finally plotting your graph of $x(t)$ against time.



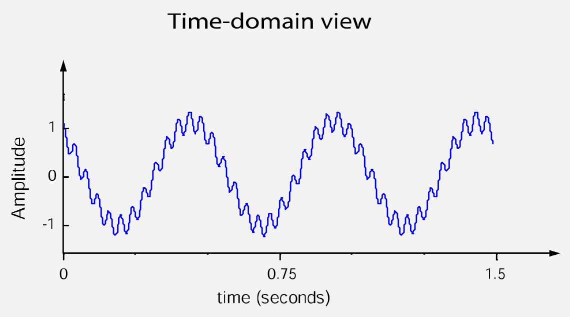
You should notice that when the phase value is zero that the waveform will be a maximum when t=0 and every period of the waveform after that. Changing the phase will change the times when the maximum of the sinusoid will occur. You should try this out for yourself using the code above and you should also observe that adding 2π to any phase offset value you try out will produce the exact same waveform. For example, the waveform produced when the phase offset is set to 1.4 will be the same as the waveform produced when the phase is set to 1.4+2π, or 1.4+4π, or even 1.4-2π for that matter. In fact, you will find that for any integer k the following relationship holds:

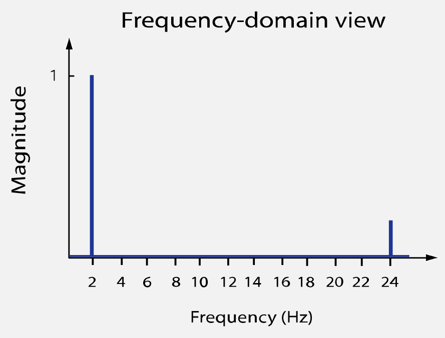
$$A\cos(2\pi f t + \varphi) = A\cos(2\pi f t + \varphi + k2\pi)$$

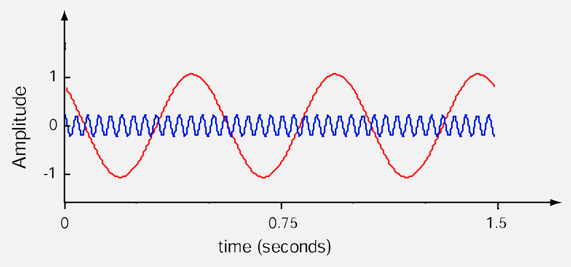
All signals can be decomposed into sinusoids

The French mathematician Jean-Baptiste Joseph Fourier showed that any signal can be recreated by adding sinusoidal signals together. (See pzdsp.com/vid14 and pzdsp.com/vid15 for video tutorials/demonstrations on this concept).

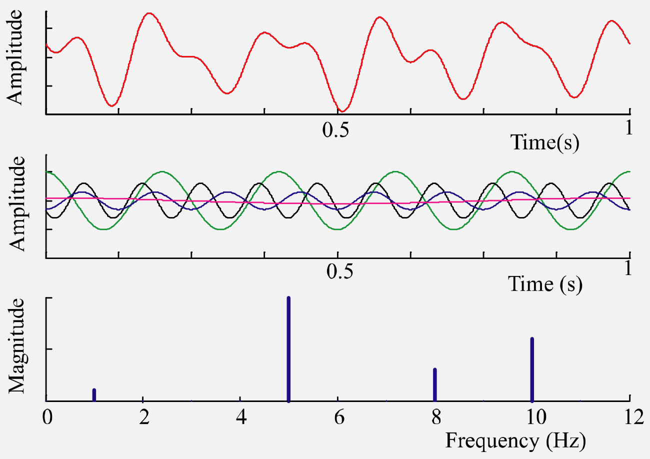
The frequency-domain view of a signal provides a way to visualise the sinusoids that make up a signal i.e. the sinusoids that when added together reproduce the original signal. The magnitude spectrum shows the amplitudes of the various sinusoids which make up a signal, while the phase spectrum shows the phases of the sinusoids which make up a signal.







The figure above shows a waveform (top) which is a plot of the time-domain signal produced when the two sinusoids shown below it are added together. The frequency-domain view of this signal contains two spikes; the spike at 2 Hz is larger than the one at 24 Hz because the 2 Hz sinusoid is larger (5 times larger) than the 24 Hz sinusoid.



The figure to the right shows the magnitude spectrum of a signal in the bottom plot; with the time-domain view of the same signal shown in the top plot. Each of the 'spikes' in the magnitude spectrum represents a sinusoid (there are 4 in total indicating the presence of 4 sinusoids in the signal; in other words the signal could be reproduced by adding four sinusoids together). Each of the four sinusoids, which when summed together produce the time-domain signal shown in the top plot, are shown in the middle plot. The green sinusoid has 5 cycles over the one second duration of the segment shown and therefore has a frequency of 5 Hz; it has the largest amplitude, as can also be seen in the corresponding magnitude spectrum plot where the 'spike' shown at 5 Hz is the largest. It can also be seen in the magnitude spectrum that the 'spike' at 8 Hz is less than half the height of the 5 Hz spike; this can also be seen in the middle plot whereby the sinusoid with 8 cycles in one second has an amplitude of less than a half the amplitude of the 5 Hz sinusoid.

The phase values for each of the sinusoids present in the signal are 0, 0, 3.14, 2.13 radians for the 1, 5, 8, and 10 Hz components. These phase values are phase shifts relative to cosine waveforms. A plot of the phase spectrum shows the phase values plotted against frequency in a similar way to the magnitude spectrum showing the magnitude values plotted against frequency.

If you would like to see a practical application of the frequency-domain then take a look at pzdsp.com/vid12.

A note on the duration of a sinusoid

From the mathematical description of a sinusoid a sinusoidal waveform exists for all instances of time. In this document I show plots of sinusoidal segments which have a finite duration and you'll notice that I still refer to these plots of sinusoidal segments as sinusoids, which is, strictly speaking, incorrect but makes the document a bit easier to read.

How to use Octave/Matlab's fft function

The fft function can be used to determine the amplitude, frequencies and phases of the sinusoids that a signal is comprised of and is frequently used to obtain a frequency-domain plot of a signal. In this section I'll explain how to use the fft function without getting into detail on its inner workings. You should note that the fft function is an implementation of the Discrete Fourier Transform algorithm which is described in detail in the next section.

In this section I'll first show how to create a frequency-domain plot of the bass guitar signal used in the Introduction, then I'll provide another example which provides more insight on how to use the fft to analyse a signal which is based on the popular video on the subject that I created in 2012 pzdsp.com/vid16.

Using Octave/Matlab's fft function to analyse a bass guitar signal

The following code can be used to load in an audio signal and plot its frequency content. The audio file in the example can be downloaded from pzdsp.com/sig1 and you should make sure the audio file is stored/saved in the 'present working directory' – this can be determined by typing pwd at the command line.

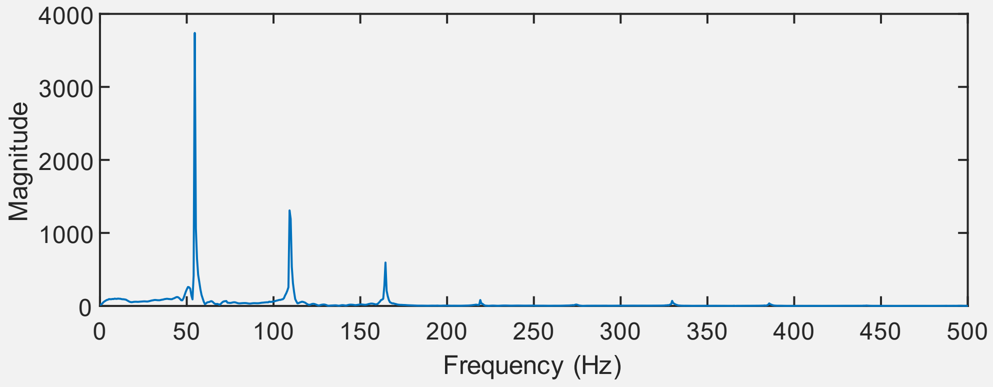
```
>> [b fs]= audioread('bass_note.wav');  % the variable b
contains the audio samples. The audioread function also
returns the sampling rate,fs, which is 44100 in this case
>> B = fft(b); % the fft returns 67822 complex numbers
which are stored in the array variable B. Note that, by
convention, capital letters are used to store frequency-
domain information while lowercase are used for time-
domain. There are 67822 samples in the time-domain signal
b. The fft function returns the same number of values as
are in the signal being analysed i.e. the time-domain
signal b in this case.
>> B_mags = abs(B); % the abs function determines the
magnitudes of the 67822 complex numbers.
```

The second line in the code above is the one that does all the hard work. The fft function analyses the time-domain signal b (as described in detail in the next section) to determine the magnitudes and phases of the sinusoids required to reproduce the time-domain signal b.

One of the most common ways to visually analyse the frequency content of signal is to plot the magnitudes of the values returned by the fft function, which provides a plot of the magnitude spectrum. You should note that there are numerous ways to plot the magnitude spectrum and the video available at pzdsp.com/vid17 provides a detailed explanation on how to do so. The following code shows one common method of plotting the magnitude spectrum against frequency using units of hertz, where the xlim([0 500]) command limits the range of frequencies being displayed to be from 0 to 500 Hz.

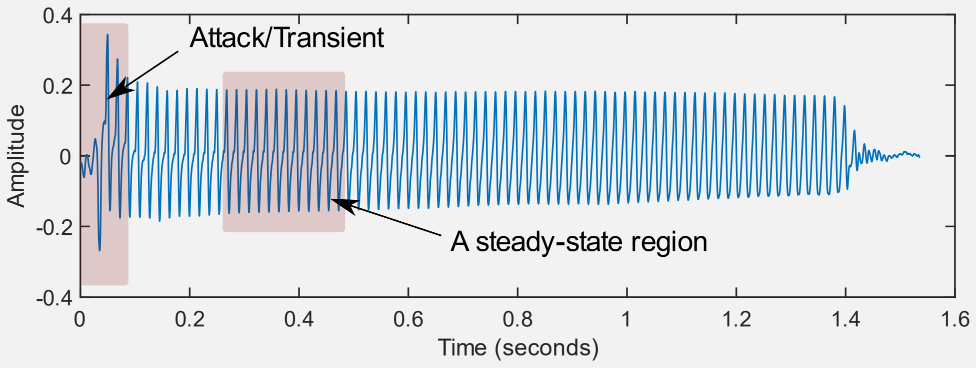
```
>> plot([0:length(b)-1]/length(b)*fs , B_mags)
>> xlim([0 500]); %limit the range of frequencies to be
from 0 to 500 Hz
>> xlabel('Frequency (Hz) '); ylabel('Magnitude');
```

A more detailed explanation of this code can be found from pzdsp.com/vid17. This code will produce the following plot in which the fundamental and first two harmonics of the guitar note can be clearly seen. There is a fundamental frequency component at about 55Hz, with strongly present harmonics at 110 Hz and 155 Hz, as indicated by the large spikes at these frequencies



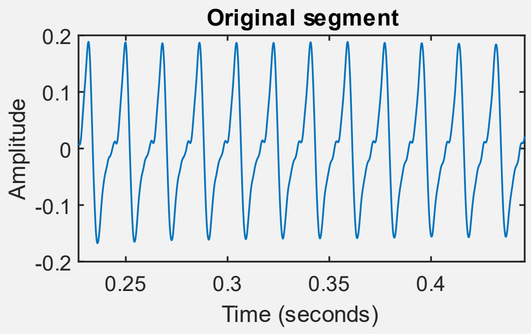
The time-domain view of this signal can be plotted using the following code, and you can see that the signal contains a sharp attack/transient element from when the bass guitar string was plucked.

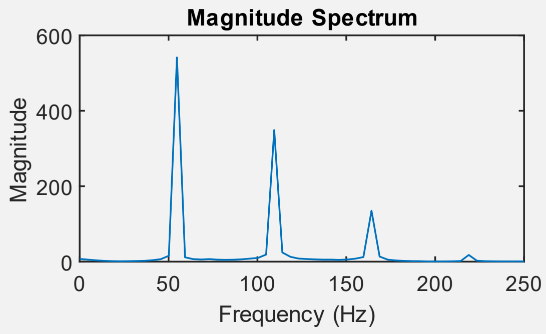
```
>> [b fs]= audioread('bass_note.wav');
>> t = [0:length(b)-1]*1/fs;
>> plot(t,b)
>> xlabel('Time (seconds)'); ylabel('Amplitude')
```

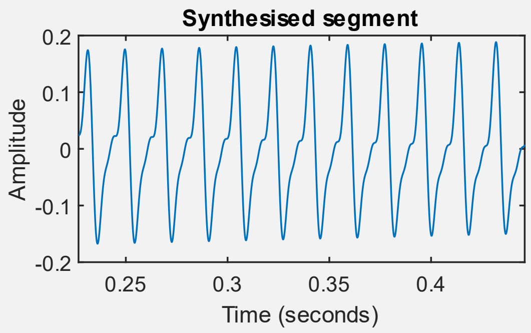


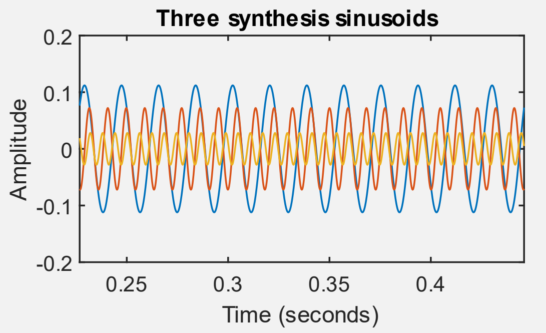
The figure above also highlights a steady-state region in which the signal is reasonably stationary. This steady-state/stationary segment can be reproduced reasonably well by adding just three sinusoidal components, as shown by the code and plots below. Note that the code doesn't show how the amplitudes, frequencies and phases of the three sinusoidal components are determined, however, how this could be done can be appreciated after reading through the entire document!

```
>> [ip fs]= audioread('bass_note.wav');
>> N = 9670;
>> stationary_seg = ip(10000:10000+N-1);
>> t = [0:N-1]/fs; t_offset = 10000/fs;
>> subplot(2,2,1); plot(t+t_offset, stationary_seg)
>> title('Original segment')
>> xlabel('Time (seconds)'); ylabel('Amplitude');
>> subplot(2,2,2);
>> plot([0:N-1]/N*fs , abs(fft(stationary_seg)))
>> xlim([0 250]); %limit frequencies from 0 to 250 Hz
>> xlabel('Frequency (Hz) '); ylabel('Magnitude');
>> title('Magnitude Spectrum')
>> fundamental = 0.112*cos(2*pi*54.7*t-0.82);
>> harmonic1 = 0.072*cos(2*pi*109.7*t+3);
>> harmonic2 = 0.028*cos(2*pi*164.6*t + 0.83);
>> synth_sig =  fundamental + harmonic1  + harmonic2 ;
>> subplot(2,2,3);plot(t+t_offset, synth_sig);
>> xlabel('Time (seconds)'); ylabel('Amplitude');
>> title('Synthesised segment')
>> subplot(2,2,4); plot(t+t_offset, fundamental);
>> hold on ; plot(t+t_offset, harmonic1);
>> plot(t+t_offset, harmonic2);
>> xlabel('Time (seconds)'); ylabel('Amplitude');
>> title('Three synthesis sinusoids')
```







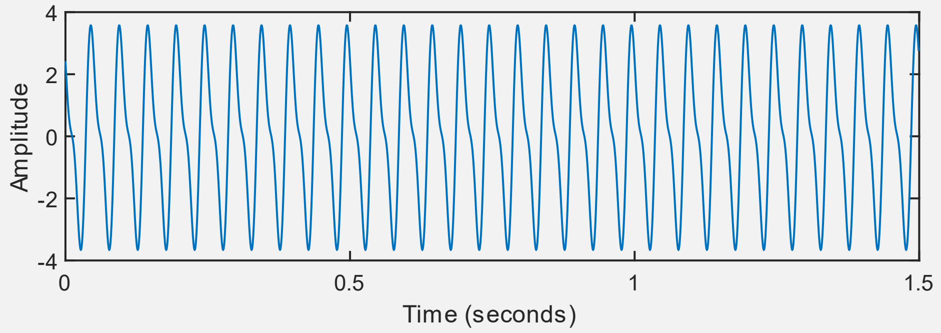


Interpreting the output of the fft function

The fft function returns a sequence of complex numbers, and these complex numbers describe the amplitude and phases of the sinusoidal waveforms that a time-domain signal is comprised of. In this section I'll attempt to explain how to interpret the complex numbers returned by the fft function using some synthesised example signals, as explained in pzdsp.com/vid16.

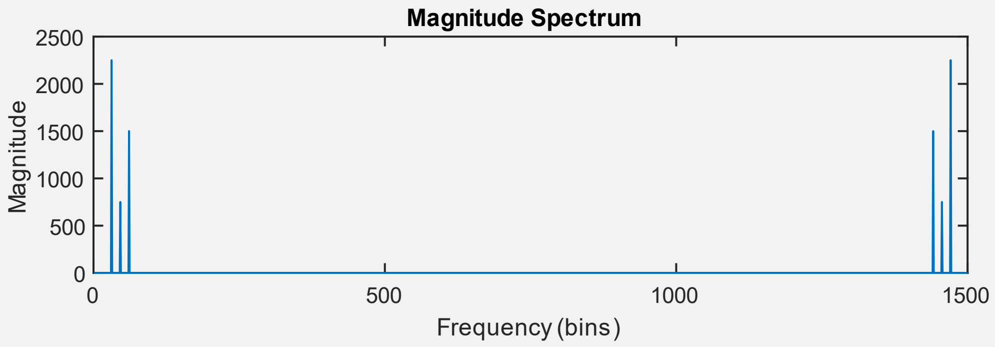
Let's start by synthesising a time-domain signal that contains three sinusoids. The fft function should be able to determine the amplitudes and phases of these three sinusoids so let's see how it does it.

```
>> fs = 1000;
>> t = 0 : 1/fs :1.5-1/fs;
>> x = 3*cos(2*pi*20*t + 0.2) + 1*cos(2*pi*30*t -0.3) +
2*cos(2*pi*40*t + 2.4);
>> plot(t,x);
>> xlabel('Time (seconds) ');
>> ylabel('Amplitude');
```



We know that the time-domain signal shown above contains three sinusoids of frequencies 20 Hz, 30 Hz and 40 Hz with phase offsets of 0.2 radians, -0.3 radians and 2.4 radians, respectively, and amplitudes 3, 1, and 2, respectively. The time-domain signal contains 1500 samples (sampling rate is 1000 Hz) and when we apply the fft function to this signal 1500 complex numbers are returned. If we plot the magnitudes of these 1500 complex numbers, as shown below, we can see three 'spikes' on the left hand side of the plot with another three spikes 'mirrored' on the right hand side. The three pairs of 'spikes' represent the three sinusoidal components that the original signal is comprised of.

```
>> X = fft(x);
>> plot(abs(X)); xlabel('Frequency (bins)');
ylabel('Magnitude');
>> title('Magnitude Spectrum')
```



The amplitude of the 'spikes' correspond to the amplitude of the sinusoids. Referring to the 'spikes' on the left-hand side; the spike furthest to the left corresponds to the 20 Hz sinusoid which has the largest amplitude; the middle 'spike' has the lowest amplitude and corresponds to the 30 Hz sinusoid; while the 'spike' to the right of the grouping is twice the amplitude of the middle 'spike' and corresponds to the sinusoid with a frequency of 40 Hz.

If we took a closer look at the values of the variable X we'd see that they are complex numbers that contain a lot of zero values. While showing all 1500 values is impractical we can use the following matlab code to look at a few:

```
>> X(30:32) % X(30)     X(31)       X(32)
0+0j      2205.15+447j      0+0j
>> X(45:47)  % X(45)     X(46)       X(47)
0+0j      716.5-221.64j      0+0j
>> X(60:62)  % X(60)     X(61)       X(62)
0+0j      -1106.09+1013.19j      0+0j
```

We can see that there are three non-zero values at indices 31, 46 and 61. The magnitudes of these values are 2250, 750 and 1500, respectively, and these values can also be determined from the plot of the magnitude spectrum by examining the amplitude of each of the 'spikes'. This should make sense, since the plot of the magnitude spectrum is simply a visual representation of the magnitudes of the variable X. It's worth noting that if we divide these magnitude values by 750 (which is half the number of values in X) then we get a result of 3, 1 and 2 which exactly match the amplitudes of the three sinusoids that the synthesised signal is comprised of.

The phase angles of the complex numbers of X at indices 31, 46 and 61 are 0.2 radians, -0.3 radians and 2.4 radians, respectively. By referring to the code which synthesised the time-domain signal we can see that these phase angles directly correspond to the phases of the sinusoids that the synthesised signal is comprised of.

So, we can see that the fft function can determine the amplitudes and phase of the sinusoids that a signal is comprised. The remaining piece of information is the frequency of each of those sinusoids and to determine the frequency we have to examine the indices of the non-zero values of X i.e. 31, 46 and 61. Before continuing its important to note that matlab and octave index the first value of an array with the number 1, while mathematicians (and most other programming) languages will index the first element of an array with the number 0. The values of the array returned by the fft function (stored in the variable X) are referred to as bin values, with the first element of the array being referred to as bin number 0. The non-zero values of the variable X occurring at indices 31, 46 and 61 therefore correspond to bin numbers 30, 45 and 60. These bin numbers are related to frequency with the bin numbers associated with the left-hand side of the magnitude spectrum being converted to frequency in hertz using the following formula:

$$f = k.f_s/N$$

where f is the frequency associated with bin k, f_s is the sampling frequency and N is the total number of bins (which is equal to the number of values in the variable X).

Using this formula for bin values k set to 30, 45 and 60 gives frequencies 20 Hz, 30 Hz and 40 Hz, which correspond to the frequencies of the sinusoids used to synthesise the time-domain signal.

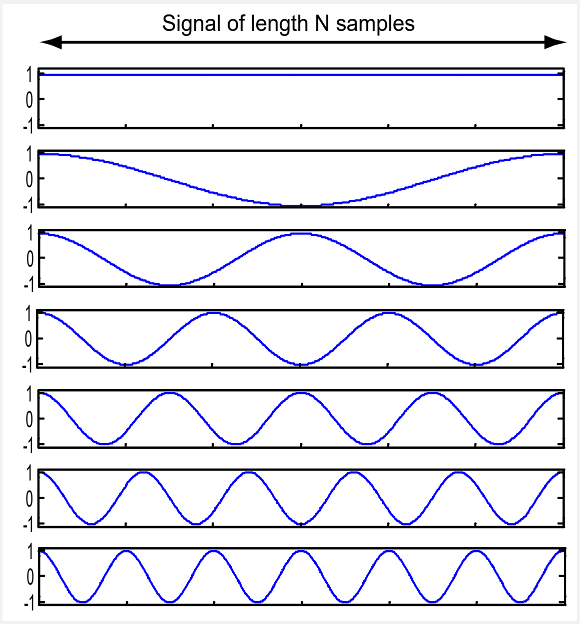
Before moving on to understanding how the DFT works (and the fft function is simply an efficient implementation of the DFT) its important to appreciate how to interpret the values returned by the DFT (the fft function).

I would like to say that for many scientists and engineers a deep understanding of the DFT is not required to undertake some basic frequency analysis and an ability to have a high-level interpretation of the output of the DFT would be adequate in a large number of cases. For those readers who feel they may not require a detailed understanding of the DFT I would recommend skipping ahead to the section entitled “The DFT in Practice – zero-padding and windowing”.

The Discrete Fourier Transform

The Discrete Fourier Transform (DFT) is a mathematical process that can be used to determine the sinusoids that a discrete signal is comprised of. It basically works out the amplitudes and phases of the sinusoids that will reproduce the original discrete signal if those sinusoids were added together.

Before getting into the detail of the DFT it's worth noting a couple of features associated with the sinusoids that the DFT uses to reproduce a discrete signal. Firstly, **the sinusoids only have an integer number of cycles** over N samples, where N is the number of samples in the discrete signal being analysed.

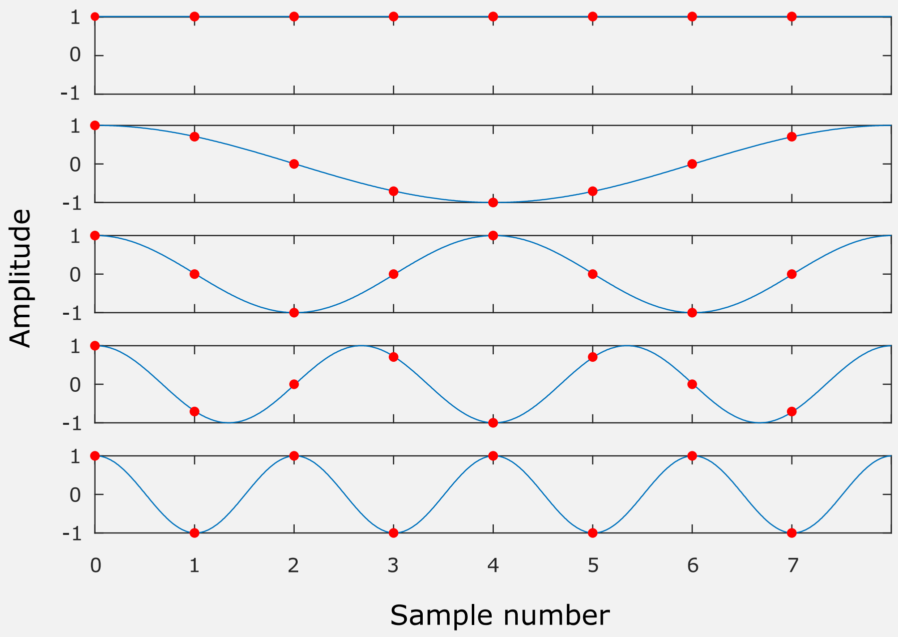


The figure to the right shows some examples of such sinusoids. The second plot from the top shows a sinusoid with 1 cycle over N samples; the next one down has 2 cycles over N samples; and so on. The plot at the very top is a special case and has 0 cycles over N samples, in other words, a DC signal that has a constant amplitude.

The second key feature is that a discrete signal of length N samples **will require at most N/2+1 of these 'integer cycle' sinusoids to reproduce the original discrete-time signal.**

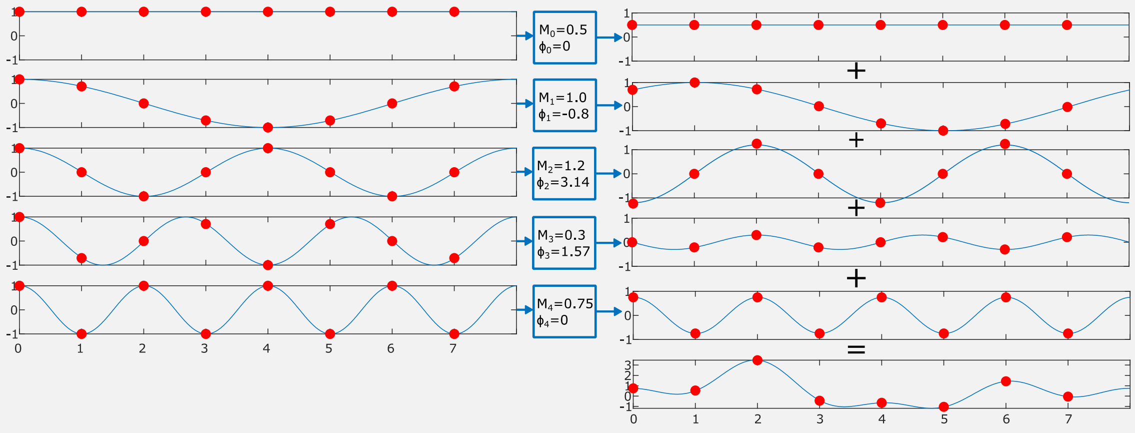
To understand the operation of the DFT, it's useful to think of the frequency of the sinusoids in terms of radians per sample rather than Hertz. For example the sinusoid which oscillates at a rate of one cycle over N samples (i.e. the sinusoid second from top in the above figure) has a frequency of $2\pi/N$ radians per sample, since a sinusoid with one cycle over N samples undergoes 2π radians of a revolution over N samples. Similarly, the sinusoid at the bottom of the figure has a frequency of $12\pi/N$ radians per sample, since it underdoes 6 cycles over N samples i.e. 6 times 2π radians. Take a look at the video at pzdsp.com/vid24 for an overview of angular frequency.

Given the above two features, any discrete-time signal of 8 samples could be reproduced by summing just 5 sampled sinusoids. You should note that no matter what the values of the 8 samples are, or what sampling rate was used to capture the 8 samples, the frequency of the sinusoids will be same when expressed in radians per sample. The key point is, we can reproduce any discrete-time signal containing 8 samples using the 5 sinusoids shown in the figure below. All we have to do is modify the amplitude and phase of the 5 sinusoids in a particular way. These sinusoids are referred to as **sinusoidal basis functions**, so there are 5 sinusoidal basis functions which are modified in terms of amplitude and phase before they are summed together to reproduce the original discrete signal. You should note that all the sinusoidal basis functions have an amplitude of 1 and have a phase shift of 0, relative to a cosine waveform.



It’s worth noting that the red dots shown in the above figure are the samples of sinusoids which have an integer number of cycles over 8 samples and that the discrete signals associated with each basis function are simply a sequence of 8 numbers. For example, the discrete signal associated with the basis function with 4 cycles over 8 samples at the bottom of the figure is the sequence 1, -1, 1, -1, 1, -1, 1, -1. The blue lines are shown to make it easier to visualise the sinusoidal nature of these basis functions.

To make this a little bit clearer it can be useful to look at an example discrete-time signal and its associated modified basis functions.



The plot shown in the bottom right corner of the figure above shows the discrete signal given by the sequence

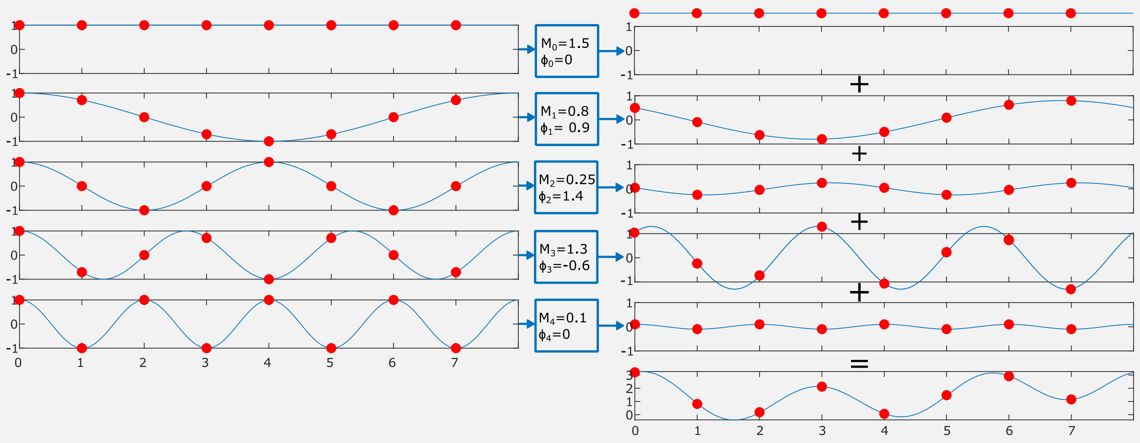
[0.7467 0.5378 3.4674 -0.4475 -0.6467 -1.0378 1.4326 -0.0525].

The red dots show the discrete sample points. The 5 basis functions are shown to the left of the figure. Each basis function is modified in terms of magnitude and phase, as shown by the waveforms to the right. For example, the basis function that has 2 cycles over 8 samples (the third waveform from the top) has a phase shift of 3.14 radians applied to it and its amplitude is scaled by a factor 1.2 i.e. increased by 20%.

The figure below shows a second example where the discrete-time signal in this case is given by the sequence

[3.2127 0.8225 0.1968 2.1293 0.0723 1.4848 2.9182 1.1634]

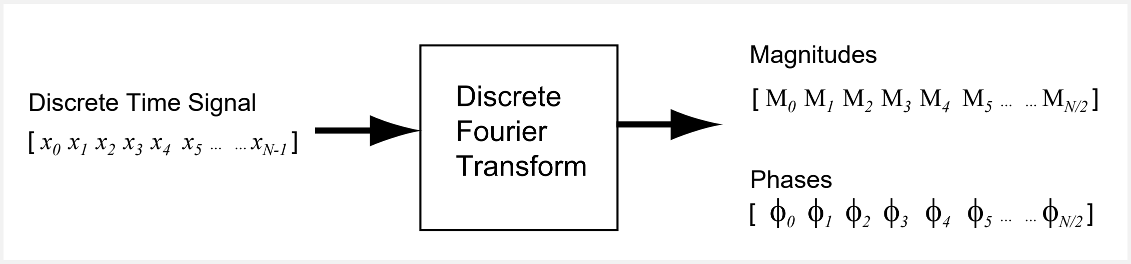
Since both of the example signals contain 8 samples, they both can be reproduced by modifying the same set of sinusoidal basis functions, however the way in which each of the basis functions are modified is different for each example.



The key point to appreciate is that <u>any</u> sequence of 8 samples can be reproduced by modifying the 5 basis functions shown in the left of the figures above. Each unique sequence of 8 samples will have a unique set of magnitude and phase modification parameters though.

What the DFT does

The DFT determines how the magnitude and phase of the sinusoidal basis functions should be modified so that they will reproduce the original signal when the modified basis functions are added together.


Discrete Time Signal
$[x_0\ x_1\ x_2\ x_3\ x_4\ x_5 \ldots \ldots x_{N-1}]$
Discrete Fourier Transform
Magnitudes
$[M_0\ M_1\ M_2\ M_3\ M_4\ M_5 \ldots \ldots M_{N/2}]$
Phases
$[\phi_0\ \phi_1\ \phi_2\ \phi_3\ \phi_4\ \phi_5 \ldots \ldots \phi_{N/2}]$

In practice, when the DFT algorithm is applied to a discrete-time signal, it will output a sequence of complex numbers where the magnitudes of the complex numbers correspond to the magnitudes that each basis functions should be multiplied by. Similarly, the phase of each complex number corresponds to the phase shift that should be imparted on each sinusoidal basis function. I'll show an example of this in action in the paragraphs below but before I do I'd like to point out that most DFT algorithms are implemented so that they output N complex numbers rather than N/2 + 1. You'll find that the 'additional' N/2

-1 complex numbers are somewhat redundant though so, for the moment, don't worry too much about them!

Let's take a look at how the DFT is used on a discrete signal with 8 samples using a matlab/octave example – the complete mathematical calculations are available in the next section but at this stage it's not necessary to follow the mathematical detail, rather it's more beneficial to focus on what the DFT outputs at this stage. The code below shows the variable x being set equal to an array of 8 values, this represents the 8 samples of a discrete signal. The variable x is then processed by a function called fft (fft is an acronym for fast Fourier transform, which is an efficient implementation of the DFT) and you can see that in the fft function returns, or outputs, 8 complex numbers.

```
>> x = [3.2127    0.8225    0.1968    2.1293    0.0723    1.4848    2.9182
1.1634];  % a discrete-time signal of length 8 samples
>> X = fft(x) % the fft returns, or outputs, 8 complex
numbers. The last 3 are the conjugate of the earlier
values and are generally not required
12.0+0.0i   1.9891+2.5067i   0.17+0.9854i   4.2917-2.9361i    0.8+0.0i
4.2917+2.9361i   0.17-0.9854i   1.9891-2.5067i
>> amps = abs(X(1:5)) %determine the magnitude of the
first 5 values of X
12.0000    3.2000    1.0000    5.2    0.8000
>> phases =angle(X(1:5))  % determine the phase values of
the first 5 values of X
0    0.9000    1.4000   -0.6000        0
```

The matlab example above demonstrates how to use the fft function to determine the set of amplitude and phase values of the sinusoids that the time-domain signal x is comprised of. The example signal used is the same as the one shown the figure above (top of pg 17). Referring to the figure, the phase values determined by the fft function in the code above directly correspond to ϕ_0, ϕ_1, ϕ_2, ϕ_3 and ϕ_4 in the figure. Note that the phase of the DC component is zero, which is the first phase value returned by matlab's angle function. The amplitude values returned by matlab's abs function require some additional scaling: the first and last magnitude values of those listed are divided by N, while the remainder are divided by N/2. The first and last values will always be scaled by N regardless of the length of the signal being analysed, while the remaining values are scaled by N/2. For the sake of completeness I feel I have to mention that the reason some of the sinusoids are scaled by

N/2 is because the 'somewhat redundant' sinusoids I mentioned above aren't included. I realise this comment is perhaps a distraction for some readers and you can dismiss it if it causes any confusion.

To verify that these values represent the amplitudes and phases of the sinusoids that x comprises of example matlab code is shown below which adds appropriately scaled and phase shifted sinusoids together. *Note: a discrete sinusoid with k cycles over N samples is given by the expression* $\cos\left(\frac{2\pi k}{N}n\right)$.

```
>> N = 8;
>> n = 0 : N-1; % vector of sample numbers
>> dc_component = amps(1)*cos(0*n)/N;
>> one_cycle = amps(2)*cos(2*pi*n/N+phases(2))/(N/2);
>> two_cycle = amps(3)*cos(2*pi*n*2/N+ phases(3))/(N/2);
>> three_cycle = amps(4)*cos(2*pi*n*3/N +
phases(4))/(N/2);
>> four_cycle = amps(5)*cos(2*pi*n*4/N + phases(5))/N;
>> sinusoids_summed = dc_component + one_cycle + two_cycle
+       three_cycle + four_cycle % add the five components
to reproduce the original signal
3.2127    0.8225    0.1968    2.1293    0.0723    1.4848    2.9182    1.1634
```

As can be seen by the numerical values shown on the last line above, the sinusoids added together reproduce the original signal x.

The short example sequence of 8 samples above is useful in terms of being able to appreciate the numerical values that the DFT returns, however, in practice, scientists and engineers usually analyse discrete-time signals which are hundreds, thousands or even millions of samples long. The reader is referred back to the section on 'How to use Octave/Matlab's fft function' to see a practical audio example in which a bass guitar signal is analysed.

A note on summing cosine and sine waveforms

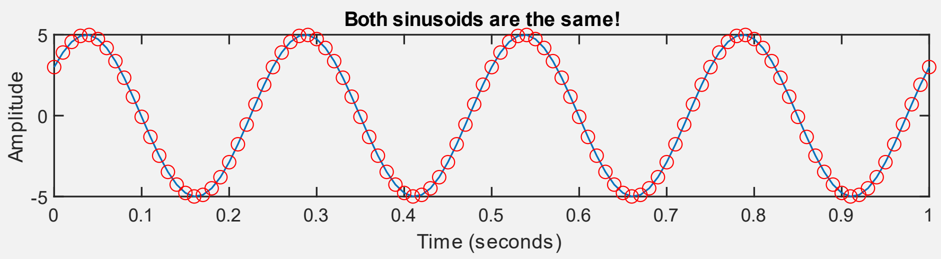
It's worth noting that a sinusoid given by $A\cos(2\pi ft + \varphi)$ can also be represented as the sum of a sine waveform and a cosine waveform of the same frequency, f, with no phase offsets i.e.

$$A\cos(2\pi ft + \varphi) = A_c\cos(2\pi ft) - A_s\sin(2\pi ft)$$

where $A = \sqrt{(A_c^2 + A_s^2)}$ and $\varphi = \text{atan2}(A_s, A_c)$. *(I won't provide the proof of this here but the interested reader could show this from Ptolemy's identities).*

The following example code will help illustrate this, whereby two sinusoids synthesised in two different ways are shown to be equivalent.

```
>> t=0 : 0.01 : 1;
>> f = 4;
>> x1 = 3*cos(2*pi*f*t)  + 4*sin(2*pi*f*t); % sinusoid 1
>> x2 = sqrt(3^2+4^2)*cos(2*pi*f*t + atan2(-4,3)); %
sinusoid 2 : 5cos(8πt – 0.9273)
>> plot(t,x1); hold on;
>> plot(t,x2, ' ro');  % plot in red circles
>> xlabel('Time (seconds) ');ylabel('Amplitude');
>> title('Both sinusoids are the same! ')
```

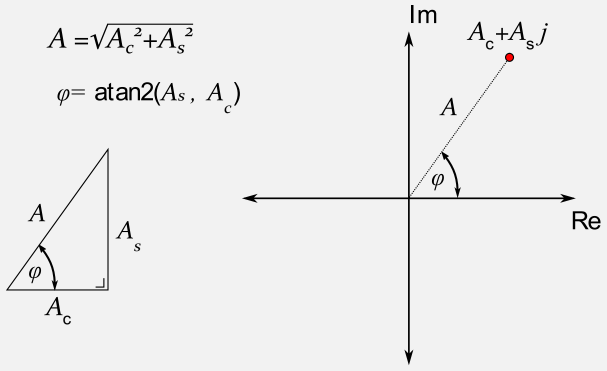


So, any sinusoidal waveform (with any phase shift) can be further decomposed into a sine waveform and cosine waveform with zero phase shifts.

You'll notice that the relationship between the amplitude of the sinusoidal waveform A and the amplitudes of the sine and cosine waveforms (As and Ac, respectively) can be visualised using a right-angle triangle or an argand diagram (Readers familiar with phasor representations will also appreciate this!), as shown below.

$$A = \sqrt{A_c^2 + A_s^2}$$

$$\varphi = \text{atan2}(A_s, A_c)$$



The key point to note from this is that the values returned from the fft function are complex numbers, and while the previous section used the magnitude and phase values of these complex numbers to apply an appropriate amplitude scaling and phase shift to sinusoidal basis functions, an alternative approach is to use the real and imaginary parts of the complex numbers to appropriately scale cosine and sine waveforms with no phase shift. Compare the following code with that of page 21 – you'll notice I get the same result using this alternative approach.

```
>> real_vals = real(X(1:5)) %determine the real values of
the first 5 values of X
>> imag_vals = imag(X(1:5)) %determine the imaginary
values of the first 5 values of X
>> N = 8;
>> n = 0 : N-1; % vector of sample numbers
>> dc_component = amps(1)*cos(0*n)/N;
>> one_cycle = (real_vals (2)*cos(2*pi*n/N)  - imag_vals
(2)*sin(2*pi*n/N)  )/(N/2);
>> two_cycle = (real_vals (3)*cos(2*pi*2*n/N)  - imag_vals
(3)*sin(2*pi*2*n/N)  )/(N/2);
>> three_cycle = (real_vals(4)*cos(2*pi*3*n/N)  -
imag_vals (4)*sin(2*pi*3*n/N)  )/(N/2);
>> four_cycle = (real_vals (5)*cos(2*pi*4*n/N)  -
imag_vals (5)*sin(2*pi*4*n/N)  )/N;
>> sinusoids_summed = dc_component + one_cycle + two_cycle
+      three_cycle + four_cycle % add the five components
to reproduce the original signal
3.2127    0.8225    0.1968    2.1293    0.0723    1.4848    2.9182    1.1634
```

The reason I've gone through this cosine/sine perspective is because the inner workings of the DFT are more readily explained from this perspective.

The inner workings of the DFT

A brief overview of how the DFT works can be seen at pzdsp.com/vid26, which some readers may find useful before getting into the mathematics.

Mathematically the DFT is described by the following equation

$$X[k]=\sum_{n=0}^{N-1} x[n].e^{-j\omega_K n}$$

where $x[n]$ is the discrete time signal being analysed; e is Euler's number (approx. 2.71828 and from Euler's formula $e^{j\theta} = \cos(\theta)+j\sin(\theta)$); k is the so-called bin number, which is an integer; ω_k is the frequency associated with bin k, which is given by $\omega_k = 2\pi k/N$ radians per sample; N is length of signal $x[n]$ being analysed. $X[k]$ is typically evaluated over a range of values of k from 0 up to N, however 0 up to N/2 is usually all that is required.

I'll attempt to provide a visually intuitive perspective on this algorithm later but before doing that it can be useful to see a worked example of the algorithm so let's return to our example of a discrete signal given by the sequence of numbers [3.2127 0.8225 0.1968 2.1293 0.0723 1.4848 2.9182 1.1634]. **Don't worry too much about following the calculations too closely – I just want you to appreciate that all that's going on is a series of multiplications and additions using complex numbers!**

Note that $\omega_k = 0$ when $k=0$, so $e^{-j\omega_k n} = 1$, since $e^{-0} = 1$. Therefore $X[0] = \sum_{n=0}^{7} x[n] = x[0]+x[1]+x[2]+x[3]+x[4]+x[5]+x[6]+x[7]$. Summing these values gives us a result of **12 for *X*[*0*]**.

When $k=1$, $\omega_k = \pi/4$. Therefore $X[1] = \sum_{n=0}^{7} x[n]e^{-\frac{j\pi n}{4}} = x[0]\,e^{-\frac{j\pi 0}{4}} + x[1]\,e^{-\frac{j\pi}{4}} + x[2]\,e^{-\frac{j\pi}{2}} + x[3]\,e^{-\frac{j\pi 3}{4}} + x[4]\,e^{-j\pi} + x[5]\,e^{-\frac{j\pi 5}{4}} + x[6]\,e^{-\frac{j\pi 3}{2}} + x[7]\,e^{-\frac{j\pi 7}{4}}$. This is equal to $x[0] + x[1](1/\sqrt{2} - 1/\sqrt{2}j) + x[2](-j) + x[3]\,(-1/\sqrt{2} - 1/\sqrt{2}j) + x[4](-1) + x[5]\,(-1/\sqrt{2} + 1/\sqrt{2}j) + x[6]\,(j) + x[7]\,(1/\sqrt{2} + 1/\sqrt{2}j)$. Summing these values gives us a result of **1.9891 + 2.5067*j* for *X*[*1*]**.

When $k=2$, $\omega_k = \pi/2$. Therefore $X[2] = \sum_{n=0}^{7} x[n]e^{-\frac{j\pi n}{2}} = x[0]\,e^{-\frac{j\pi 0}{2}} + x[1]\,e^{-\frac{j\pi}{2}} + x[2]\,e^{-j\pi} + x[3]\,e^{-\frac{j\pi 3}{2}} + x[4]\,e^{-j2\pi} + x[5]\,e^{-\frac{j\pi 5}{2}} + x[6]\,e^{-j\pi 3} + x[7]\,e^{-\frac{j\pi 7}{2}}$. This is equal to $x[0] + x[1](-j) + x[2](-1) + x[3]\,(j) + x[4](1) + x[5]\,(-j) + x[6]\,(-1) + x[7]\,(j)$. Summing these values gives us a result of **0.1700 + 0.9854*j* for *X*[*2*]**.

When $k=3$, $\omega_k = 3\pi/4$. Therefore $X[3] = \sum_{n=0}^{7} x[n]e^{-\frac{j3\pi n}{4}} = x[0]\, e^{-\frac{j3\pi 0}{4}} + x[1]\, e^{-\frac{j3\pi}{4}} + x[2]\, e^{-\frac{j3\pi}{2}} + x[3]\, e^{-\frac{j\pi 9}{4}} + x[4]\, e^{-j3\pi} + x[5]\, e^{-\frac{j\pi 15}{4}} + x[6]\, e^{-\frac{j\pi 9}{2}} + x[7]\, e^{-\frac{j\pi 21}{4}}$. This is equal to $x[0] + x[1](-1/\sqrt{2} - 1/\sqrt{2}j) + x[2](j) + x[3]\,(1/\sqrt{2} - 1/\sqrt{2}j) + x[4](-1) + x[5]\,(1/\sqrt{2} + 1/\sqrt{2}j) + x[6]\,(-j) + x[7]\,(-1/\sqrt{2} + 1/\sqrt{2}j)$. Summing these values gives us a result of **4.2917 - 2.9361*j* for *X*[*3*]**.

When $k=4$, $\omega_k = \pi$. Therefore $X[4] = \sum_{n=0}^{7} x[n]e^{-j\pi n} = x[0]\, e^{-j\pi 0} + x[1]\, e^{-j\pi} + x[2]\, e^{-j\pi 2} + x[3]\, e^{-j\pi 3} + x[4]\, e^{-j\pi 4} + x[5]\, e^{-j\pi 5} + x[6e^{-j\pi 6} + x[7]\, e^{-j\pi 7}$. This is equal to $x[0]\,(1) + x[1](-1) + x[2](1) + x[3](-1) + x[4](1) + x[5](-1) + x[6]\,(1) + x[7]\,(-1)$. Summing these values gives us a result of **0.8 - 0*j* for *X*[*4*]**.

The above calculations show that the DFT is just undertaking a number of multiplications and summations – that's all! There isn't anything too mysterious there really and it can be useful to remind yourself of that. When the DFT is applied to longer signals there are just going to be a greater amount of multiplications and additions but the process doesn't change. It is worth pointing out that there are efficient ways to do these calculations that take advantage of the fact that some calculations don't need to be repeated (for example, you'll notice in the calculations above that $x[3]$ and $x[5]$ are both multiplied by $1/\sqrt{2}$ when k is 1 and also when k is 3. Efficient algorithms, such as the fast Fourier transform, take advantage of this type of redundancy by only doing the multiplication once and storing the result for later use. You should note that the output of the DFT is same as any of the efficient implementations, such as the fast Fourier transform, the only difference is how the DFT outputs are calculated.

It's useful to establish some terminology in order to discuss the DFT and in particular its useful to become comfortable with the terms *bin number* and *bin value*. The variable k used in the DFT calculations above is referred to as the bin number. For example, the bin value associated with bin number 0 is 12 i.e. $X[0] = 12+0j$. Bin number 3 has a value of 4.2917 - 2.9361j i.e. $X[3] = 4.2917 - 2.9361j$.

Bin numbers are always integers. Bin number k is associated with sinusoidal waveforms that oscillate k cycles over N samples (as can be appreciated from the code and plots in the paragraphs below). So, bin number 2 is associated with a sinusoid that has a frequency of 2 cycles over N samples; bin number 3 is associated with a sinusoid that has a frequency of 3 cycles over N samples and so on. When you are dealing with signals that contain more samples then you'll find that you have more DFT bin values, which basically just indicates that you need more sinusoidal waveforms to create longer signals. Make sure that you remember that a bin number *k* is always associated with a sinusoid that has *k* cycles over N samples, and that *k* is always an integer.

While it's nice to see how to calculate the output of the DFT it's perhaps more insightful to be able to visualise what the algorithm is doing and this is easier if we use Euler's formula ($e^{j\theta} = \cos(\theta)+j\sin(\theta)$) to allow us break the DFT equation into two parts, as shown below.

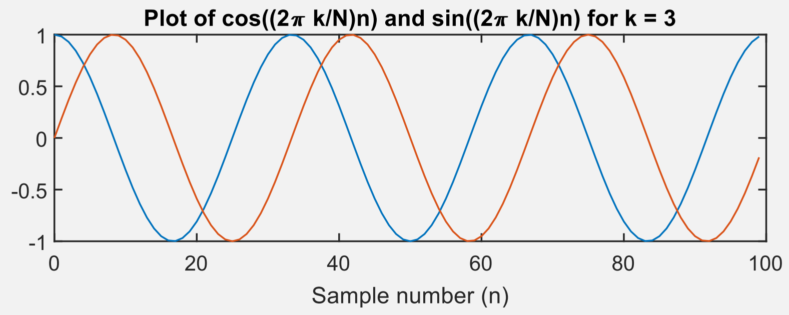
$$X[k] = \sum_{n=0}^{N-1} x[n].e^{-j\omega_K n}$$
$$= \sum_{n=0}^{N-1} x[n]\cos(\omega_K n) - j\sum_{n=0}^{N-1} x[n]\sin(\omega_K n)$$

since $e^{-j\omega_K n} = \cos(\omega_K n) - j\sin(\omega_K n)$, from eulers formula.

Looking at the left-hand summation in the expression above it can be seen that the discrete signal $x[n]$ is being multiplied by $\cos(\omega_k n)$ and the samples of the resulting signal (sequence of numbers) are being summed. A similar operation is taking place with a sine waveform with the right-hand summation term, but note that the summation is multiplied by $-j$ i.e. $-\sqrt{-1}$.

It's important to visualise what the signal $\cos(\omega_k n)$ looks like, where $\omega_k = 2\pi k/N$, and if you evaluated this expression for $0 \leq n < N$, you'd find that that the waveform associated with $\cos(\omega_k n)$ is sinusoidal and that the sinusoid has exactly *k* cycles over N samples, as shown in the example code below. The signal $\sin(\omega_k n)$ is, of course, similar, except the waveforms associated with this expression will be $\pi/2$ radians out of phase with its cosine counterpart.

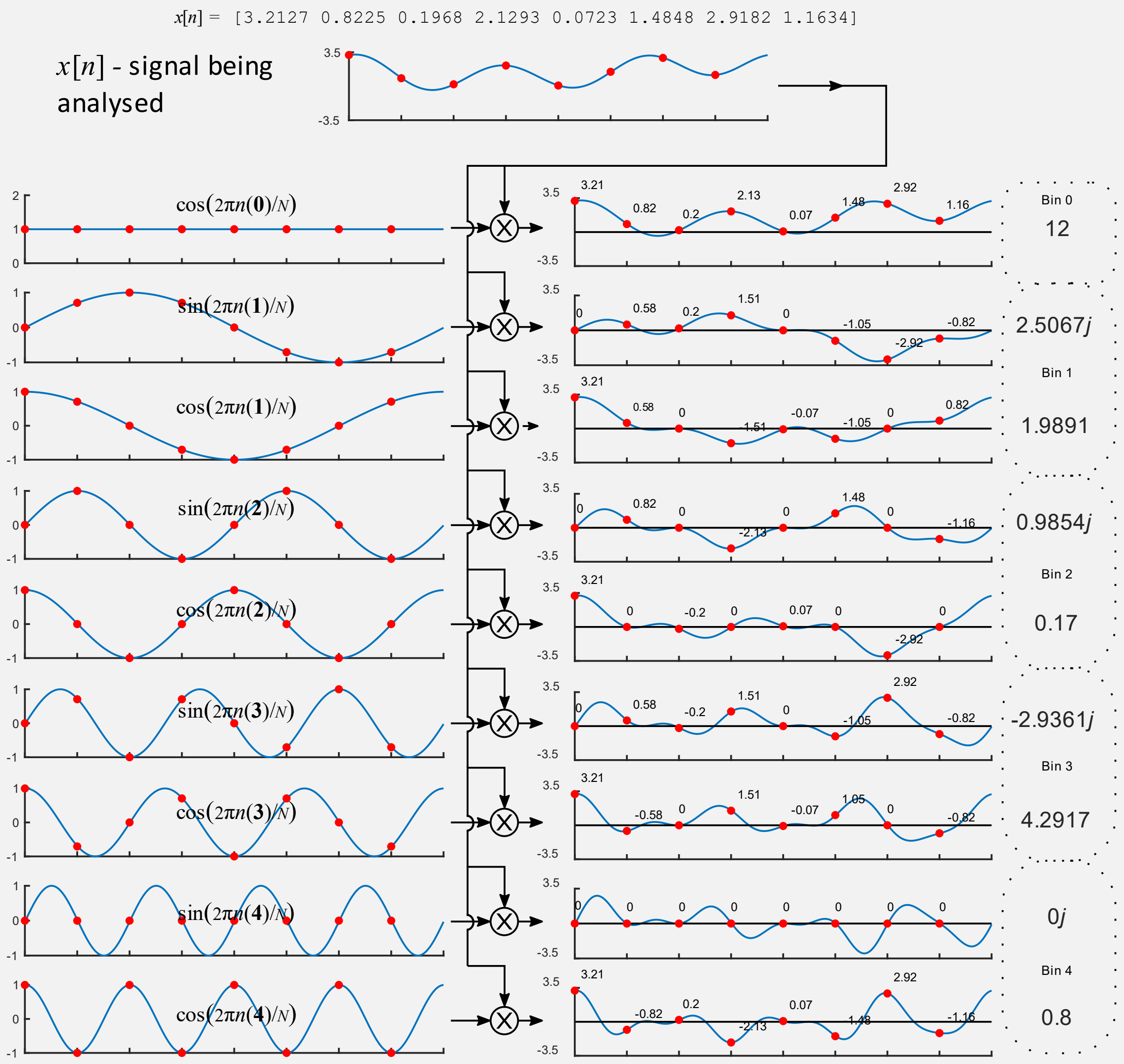
```
>> N = 100;
>> n = 0 : N-1; % vector of sample numbers
>> k = 3;
>> x1 = cos((2*pi*k/N) *n);
>> x2 = sin((2*pi*k/N) *n);
>> plot(n, x1, n, x2);
>> title('Plot of cos((2{\pi} k/N)n) and sin((2{\pi}
k/N)n) for k = 3')
```



The figure on the next page attempts to capture the DFT process visually. It shows a discrete signal $x[n]$ at the top being multiplied by a set of sine and cosine waveforms (referred to as basis functions) to the left, where the basis functions all have an integer number of cycles over N samples. The result of the multiplication is shown in the waveforms to the right and the samples of these waveforms (note the sample values displayed are rounded to 2 decimal places) are then summed to give a numerical result, with the 'sine' terms multiplied by *-j*, as per the DFT equation. These numerical results correspond to DFT bin values, where the imaginary part of the bin values are associated with sine basis functions and the real part of the bin values are associated with the cosine basis function. You should note how these values relate to the calculations shown earlier in this section, for example $X[1] = 1.9891+2.5067j$, which correspond to the results associated with the sine and cosine basis functions with 1 cycle over 8 samples.

The magnitude of the numerical result obtained by multiplying and summing gives an indication of how 'strongly present' a particular basis function is in the signal $x[n]$ and this process of multiplying and summing is known as correlation (see pzdsp.com/vid25 for and overview of correlation). Looking at the figure we can see that the cosine basis function with 3 cycles is the most 'strongly present' sinusoidal basis function in the signal $x[n]$, since it is associated with the largest numerical result (4.2917) shown on the right, in terms of magnitude. A visual inspection of the plot of $x[n]$ also shows a waveform that appears to have pattern that repeats approximately three times; this is not a coincidence! The fact that bin 0 has a value of 12 means that the signal has a positive DC offset i.e. the mean of the samples is greater than zero. If bin 0 has a value of zero this indicates that the mean of the samples is zero while a negative result for bin 0 indicates the mean of the samples is negative i.e. a negative DC offset.

It's worth noting that the frequency associated with bin k is given by $k.fs/\mathrm{N}$ Hz, since there are k cycles over N samples, and for a sampling rate fs this equates to k cycles over N/fs seconds, and the unit of Hertz relates to the number of cycles per second.



At this stage I'd like to highlight that the result of summing the samples of the waveform produced by multiplying the sine wave basis functions by the signal $x[n]$ is also multiplied by $-j$ i.e. $-\sqrt{-1}$. The reason for this can be appreciated by referring back to the DFT equation, where it can be seen that sine basis functions are multiplied by $-j$.

$$X[k]=\sum_{n=0}^{N-1} x[n].e^{-j\omega_K n}$$
$$=\sum_{n=0}^{N-1} x[n]\cos(\omega_K n)-j\sum_{n=0}^{N-1} x[n]\sin(\omega_K n)$$

Referring back to the calculations shown earlier in this section, we can see that the DFT outputs a sequence of complex numbers with the real values associated with how 'strongly present' cosine basis functions are in the signal and imaginary values associated with the 'strength' of the sine terms. Recall also, from the section entitled 'A note on summing cosine and sine waveforms', that if a cosine and a sine term are present in a signal that this is equivalent to a sinusoidal waveform of the same frequency with a phase shift being present i.e.

$$A\cos(\omega_k n + \varphi) = A_c\cos(\omega_k n) - A_s\sin(\omega_k n)$$

where $A = \sqrt{(A_c^2 + A_s^2)} = | A_c + jA_s |$ and $\varphi = \text{atan2}(A_s, A_c) = \angle(A_c + jA_s)$

This allows us to consider the fact that we obtained a correlation result of 4.2917 and -2.9361j for the three-cycle cosine and sine basis functions as being equivalent to indicating that a sinusoid given by the following equation is present in the signal:

$$5.2\cos(2\pi 3n/\text{N} - 0.6) = 4.2917\cos(2\pi 3n/\text{N}) + 2.9361\sin(2\pi 3n/\text{N})$$

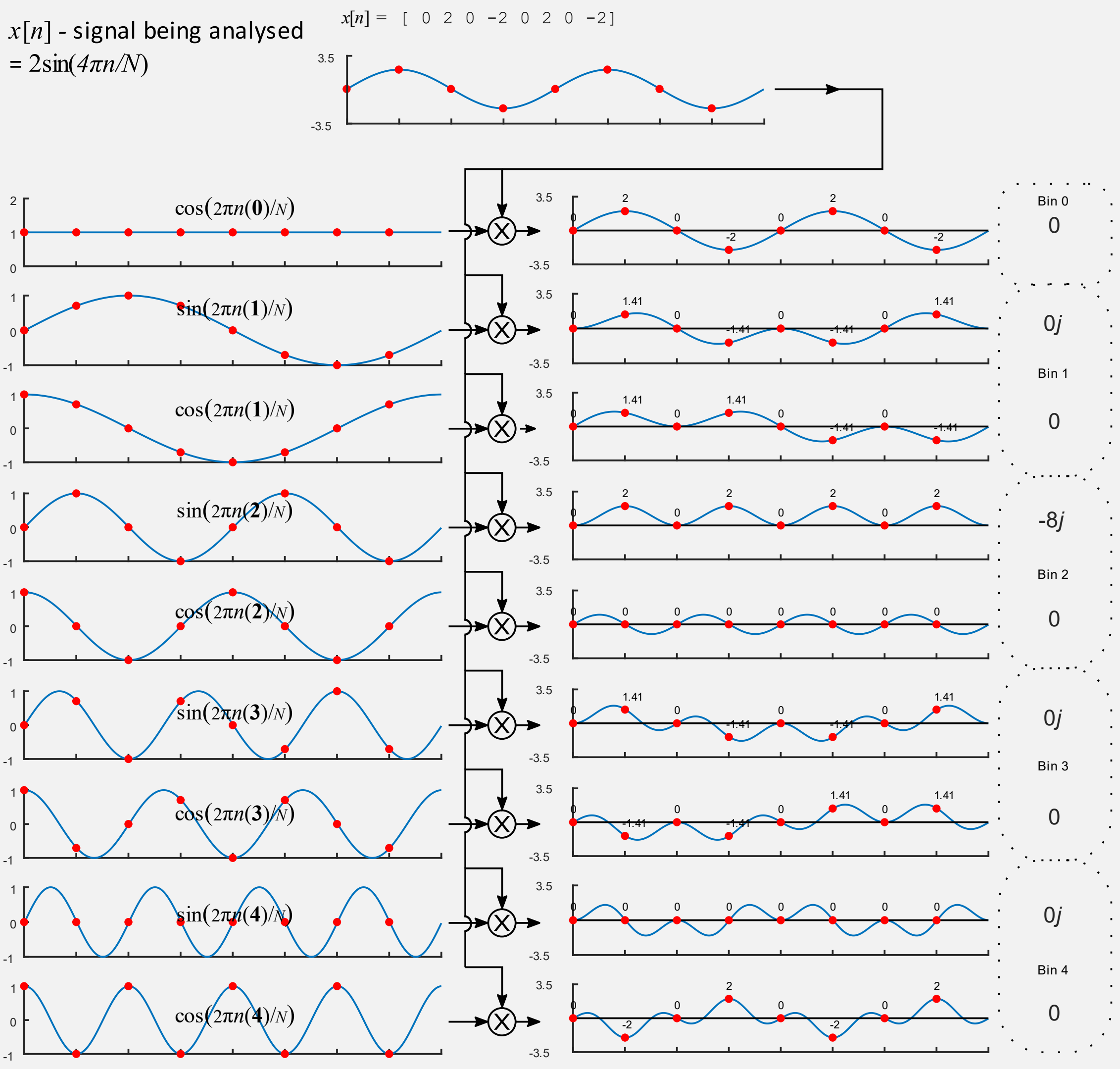
At this stage you should appreciate that the DFT identifies the presence of sinusoidal components in a signal by correlating the signal of interest with a set of sine and cosine basis functions. In the next section we'll take a look some specific signals with a view to being able to predict/interpret what the DFT outputs for 'well-defined' signals.

DFT analysis of well-defined signals

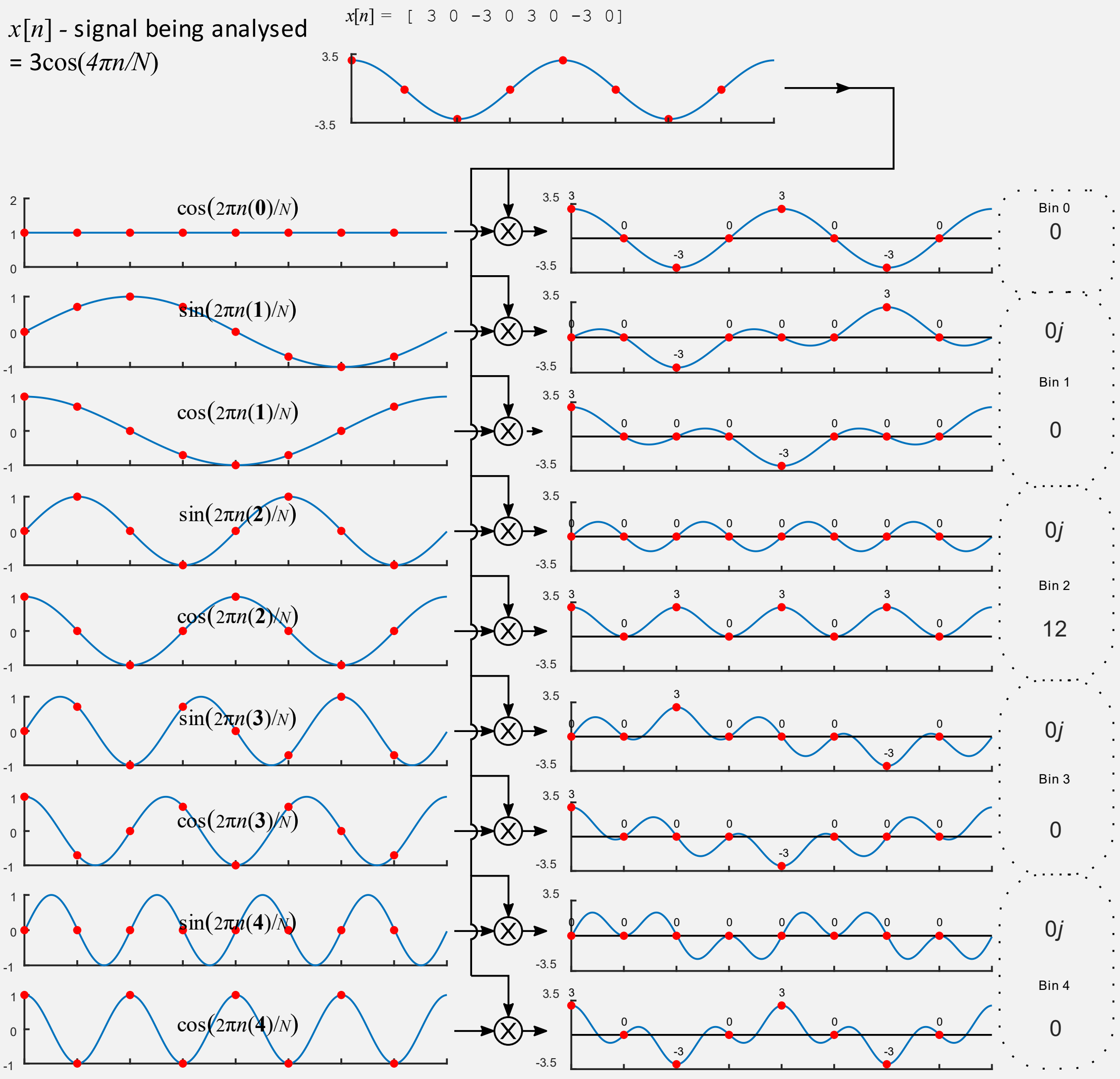
It can be worthwhile considering a few particular examples to appreciate how the DFT works and what it outputs. The remainder of this section examines four such examples:

- A sine waveform with 2 cycles over N samples and amplitude 2 i.e. $x[n] = 2\sin(2\pi 2n/\text{N})$
- A cosine waveform with 2 cycles over N samples and amplitude 3 i.e. $x[n] = 3\cos(2\pi 2n/\text{N})$
- A signal comprising of a sine waveform with 1 cycle over N samples and a cosine waveform with 3 cycles over N samples with amplitudes 2 and 0.5, respectively i.e. $x[n] = 2\sin(2\pi 1n/\text{N}) + 0.5\cos(2\pi 3n/\text{N})$
- A cosine waveform with a phase shift of -1.0854 and a frequency of 3 cycles over N samples i.e. $x[n] = 3\cos(2\pi 3n/\text{N} - 1.0854)$

For each of these four examples I'll show a visual representation of how the DFT would process each one and in the text I'll point out the key features I'd like you to focus on. You'll notice that in all four cases that the sinusoidal basis functions shown to the left of the figures do not change, since in all four cases we are dealing with signals that have 8 samples i.e. N = 8.

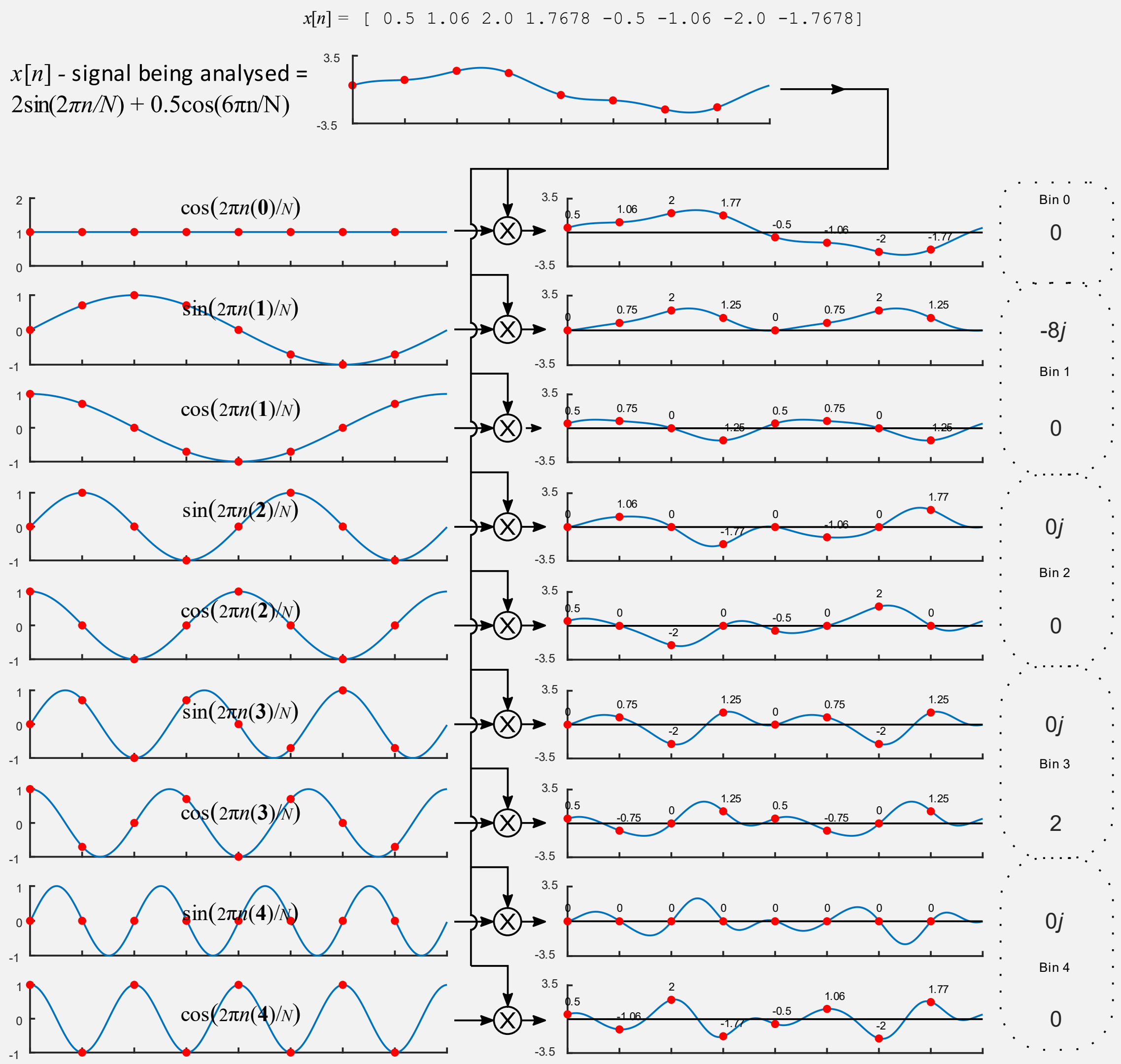

$x[n]$ - signal being analysed
= $2\sin(4\pi n/N)$
$x[n]$ = [0 2 0 -2 0 2 0 -2]
$\cos(2\pi n(\mathbf{0})/N)$
$\sin(2\pi n(\mathbf{1})/N)$
$\cos(2\pi n(\mathbf{1})/N)$
$\sin(2\pi n(\mathbf{2})/N)$
$\cos(2\pi n(\mathbf{2})/N)$
$\sin(2\pi n(\mathbf{3})/N)$
$\cos(2\pi n(\mathbf{3})/N)$
$\sin(2\pi n(\mathbf{4})/N)$
$\cos(2\pi n(\mathbf{4})/N)$
Bin 0
0
0j
Bin 1
0
-8j
Bin 2
0
0j
Bin 3
0
0j
Bin 4
0

For the first two examples you can see that the DFT bin values (the values to the right) are zero except for those values associated with basis functions that have 2 cycles over N samples. You'll also notice that for the first two examples that the amplitude of the sinusoidal waveforms can be determined from the DFT output by dividing the value of bin number 2 by N/2 = 4. This is an important fact to note – the magnitude of the bin values are proportional to amplitudes of the sinusoidal basis functions present in a signal. If a bin value is zero then this indicates that the sinusoidal basis function associated with that bin value is also zero. In other words a particular basis function is not present in a signal if its associated bin value is zero.



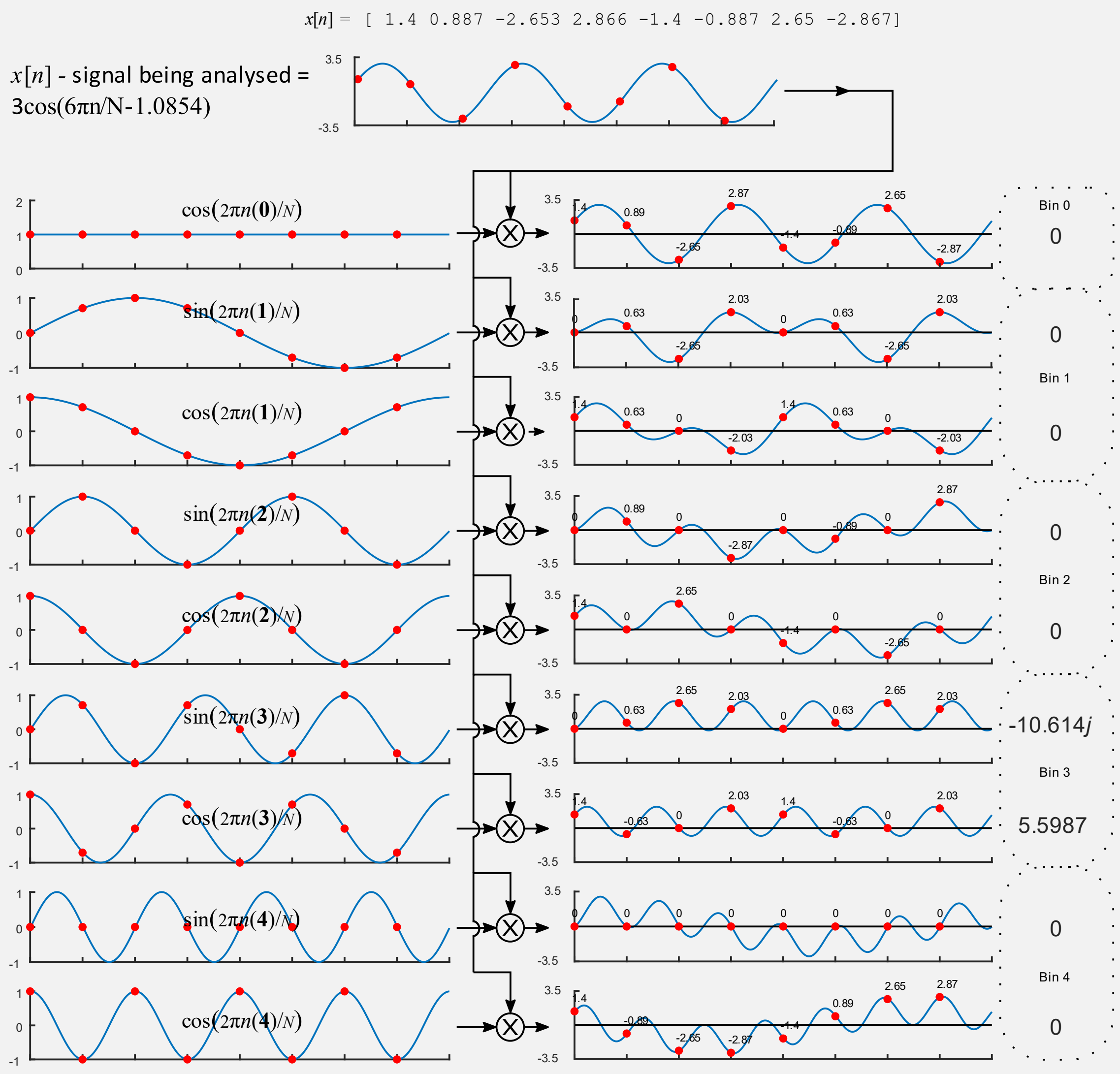
$x[n]$ - signal being analysed = $3\cos(4\pi n/N)$
$x[n]$ = [3 0 −3 0 3 0 −3 0]
$\cos(2\pi n(\mathbf{0})/N)$
$\sin(2\pi n(\mathbf{1})/N)$
$\cos(2\pi n(\mathbf{1})/N)$
$\sin(2\pi n(\mathbf{2})/N)$
$\cos(2\pi n(\mathbf{2})/N)$
$\sin(2\pi n(\mathbf{3})/N)$
$\cos(2\pi n(\mathbf{3})/N)$
$\sin(2\pi n(\mathbf{4})/N)$
$\cos(2\pi n(\mathbf{4})/N)$
Bin 0
0
0j
Bin 1
0
0j
Bin 2
12
0j
Bin 3
0
0j
Bin 4
0

The third example shows a case where the signal being analysed is comprised of two scaled basis functions of different frequencies. As you might expect, the DFT bin values are non-zero for the bins associated with these two sinusoids. Bin number 1 has a value of 0-8j, where the imaginary value indicates that the signal being analysed contains a sine waveform which oscillates at a frequency of one cycles every N samples. Bin number 3 has a value of 2+0j, indicating that the signal being analysed contains a cosine waveform which oscillates at a frequency of 3 cycles every N samples and that it is one quarter the amplitude of the sine waveform.


$x[n]$ = [0.5 1.06 2.0 1.7678 -0.5 -1.06 -2.0 -1.7678]
$x[n]$ - signal being analysed = $2\sin(2\pi n/N) + 0.5\cos(6\pi n/N)$
$\cos(2\pi n(\mathbf{0})/N)$
$\sin(2\pi n(\mathbf{1})/N)$
$\cos(2\pi n(\mathbf{1})/N)$
$\sin(2\pi n(\mathbf{2})/N)$
$\cos(2\pi n(\mathbf{2})/N)$
$\sin(2\pi n(\mathbf{3})/N)$
$\cos(2\pi n(\mathbf{3})/N)$
$\sin(2\pi n(\mathbf{4})/N)$
$\cos(2\pi n(\mathbf{4})/N)$
Bin 0
0
-8j
Bin 1
0
0j
Bin 2
0
0j
Bin 3
2
0j
Bin 4
0

The fourth and last example is a case where the signal being analysed is comprised of a cosine waveform of amplitude 3 with a phase shift -1.0854 radians. From our discussion earlier in this section it should be appreciated that this waveform can be considered to be a sum of a sine waveform and a cosine waveform. The equation below shows this relationship mathematically.

$$3\cos(6\pi n/\mathrm{N} - 1.0854) = 2.6535\cos(6\pi n/\mathrm{N}) + 1.3997\sin(6\pi n/\mathrm{N})$$



We can see that the bin values of the DFT are all zero except for the case of bin 3. This indicates that the signal only contains sinusoids that oscillate at 3 cycles over N samples, as you should expect. The fact that bin number 3 contains both real and imaginary non-zero values indicates that both a sine waveform and cosine are present in the signal.

As before the amplitudes of the sinusoidal basis functions should be divided by N/2 = 4 in order to determine the amplitude of the basis functions should be scaled if you wanted to reproduce the original signal by summing the scaled basis functions.

You'll notice that the amplitudes of the scaled basis functions are equal to the amplitudes of the cosine and sine waveforms shown in the equation given earlier.

You should also notice the magnitude of the complex number associated with bin 3 i.e. 5.5987 -10.614*j*, scaled by N/2, is equal to the amplitude of the phase shifted cosine waveform and that the phase shift is equal to the angle associated with this complex number.

Key points and a brief note on negative frequencies

The take away points from the previous section are that discrete signals of length N samples can be reconstructed by adding together sinusoids and that the DFT can be used to determine the nature of the sinusoids that will, when summed, reproduce the original signal. A key feature of the sinusoidal waveforms is that they all have an integer number of cycles over N samples.

There are two related perspectives on the sinusoidal summing process. From one perspective you can consider the sinusoids to be N/2 + 1 cosine waveforms which have a particular amplitude and phase shift applied, and that the magnitude and phase of the complex values that DFT outputs correspond to the amplitude and phase shift that should be applied to each of the N/2 + 1 cosine waveforms.

From a second perspective the sinusoids that are summed are sine waveforms and cosine waveforms with no phase shift. In this case there are a total of N waveforms to be summed. The real part of the complex values the DFT outputs corresponds the amplitude of the cosine waveforms and the imaginary part corresponds the amplitude of the sine waveform, which should be further multiplied by -1.

Both of these perspectives are equivalent since a cosine of any amplitude with a phase shift is mathematically equal to the sum of zero-phase shifted cosine and sine waveform of certain amplitudes (see the section entitled "A note on summing cosine and sine waveforms").

A third perspective worth considering is that of the discrete signals being a sum of what are referred to as complex exponentials, which are described mathematically by the expression $e^{-j\omega t}$. Complex exponentials are trickier to visualise than sinusoidal waveforms because they occupy complex space (see pzdsp.com/vid27 for a way to visualise them) but they are fundamentally a 'more correct' way of thinking about the DFT as they can be used to recreate any signal even those that are a sequence of complex numbers rather than the discrete signals that are a sequence of real valued numbers that we have considered so far. I'd like to stress that this perspective isn't essential to understanding and using the DFT to analyse signals, however many text books do refer to complex exponentials (and their associated negative frequencies) and they are required in certain contexts (quadrature signal analysis for example) so the interested reader may find the following useful. You should feel free to skip on to the next section without any loss in continuity if you prefer though.

Complex exponentials have the general form of $Ae^{j(\omega t+\varphi)}$, where ω represents angular frequency, t represents time φ represents a phase shift, and they are related to sinusoidal waveforms by Euler's formula, $e^{j\theta} = \cos(\theta)+j\sin(\theta)$.

Since $\cos(-\theta) = \cos(\theta)$ and $\sin(-\theta) = -\sin(\theta)$ then

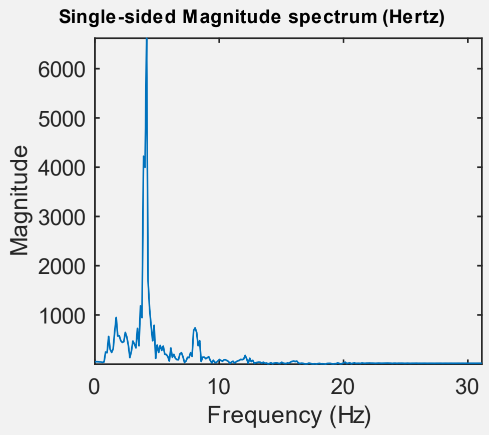
$$\cos(\theta) = \frac{e^{j\theta} + e^{-j\theta}}{2}$$

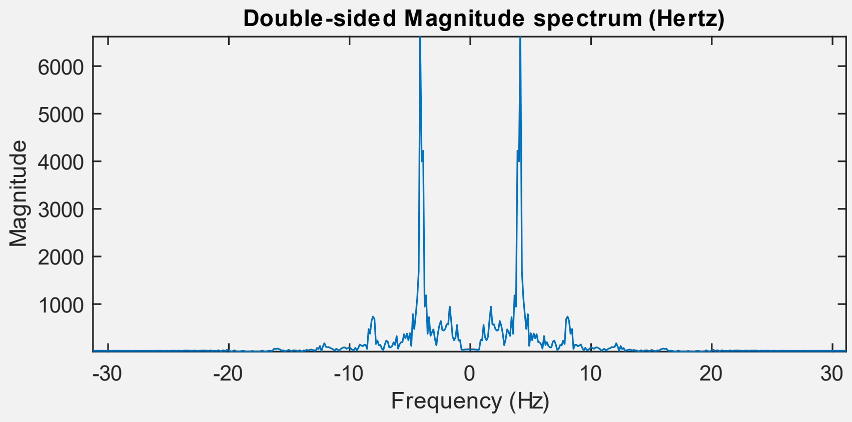
Therefore

$$A\cos(\omega t + \varphi) = \frac{Ae^{j(\omega t+\varphi)} + Ae^{-j(\omega t+\varphi)}}{2}$$

The equation above shows that any phase shifted cosine waveform can be considered as being the sum of two complex exponentials, $Ae^{j(\omega t+\varphi)}$ and $Ae^{-j(\omega t+\varphi)}$. You'll notice that the two complex exponential terms are very similar except that one has a negative sign in its exponent. This negative sign controls the direction of rotation of the complex exponential, which can be understood by referring to pzdsp.com/vid27.

Earlier in this document I provided a number of frequency-domain plots of signals which showed what is known as the magnitude spectrum of the signal. The magnitude spectrum shows the magnitudes of the sinusoids that are present in a signal, with magnitude shown on the vertical axis of the plot and frequency along the horizontal axis. It also very common to show the magnitude spectrum as an illustration of the magnitudes of the complex exponentials in a signal and in this case you will see negative frequencies as well as positive frequencies. An example of this is shown below with details on how to create these plots available at pzdsp.com/vid28. You should note that the concept of negative frequency is only relevant in the context of complex exponentials where positive and negative frequency complex exponentials rotate in the opposite direction in complex space as illustrated in pzdsp.com/vid27.





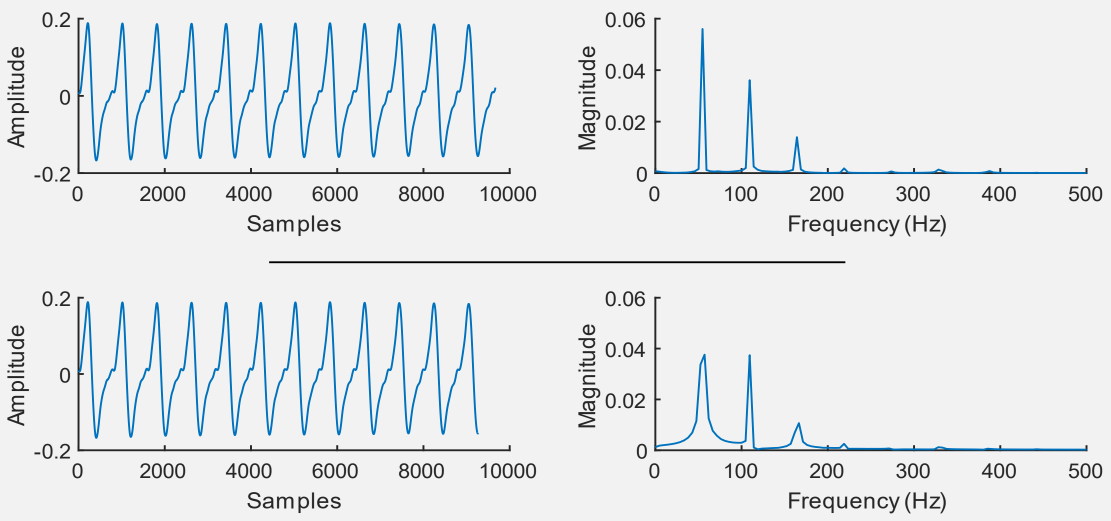
The DFT in Practice – zero-padding and windowing

In the previous section I used some well-defined signals to show how the DFT works. The signals I used were synthesised so that they all were perfectly sinusoidal and they all had an integer number of cycles over 8 samples. I'm sure you can appreciate that this is very unlikely to ever occur in a real-world situation.

At this stage I'd like to highlight a couple of issues with the DFT by going back to a practical example whereby we are analysing the frequency content of a bass guitar signal (see page 9). These issues are also explained in pzdsp.com/vid29. I've used the code below to plot the magnitude spectrums of two very similar segments of the bass guitar signal, as can be seen in the plots below. One of the time-domain segments is just 400 samples longer than the other but they share 9270 of the exact same samples. What's perhaps surprising is that the magnitude spectrums of these two signals are significantly different, despite the fact that they are so similar in the time-domain. You'll notice that the amplitude of the frequency content associated with the fundamental frequency (first peak on the left) is very different. Also, in one magnitude spectrum the spectral shape of the fundamental is similar to the first two harmonics while in the other one they are quite different.

```
>> [ip fs]= audioread('bass_note.wav'); %from
pzdsp.com/sig1
>> N = 9670; %change this to 9270 for the second signal
>> seg = ip(10000:10000+N-1);
>> plot(seg)
>> xlabel('Samples');
>> ylabel('Amplitude');
>> fax = [0:N-1]*fs/N;
>> figure; plot(fax, abs(fft(seg))/N);
>> xlabel('Frequency (Hz)');
>> ylabel('Magnitude');
>> xlim([0 500]) % just show the first 500 Hz
```

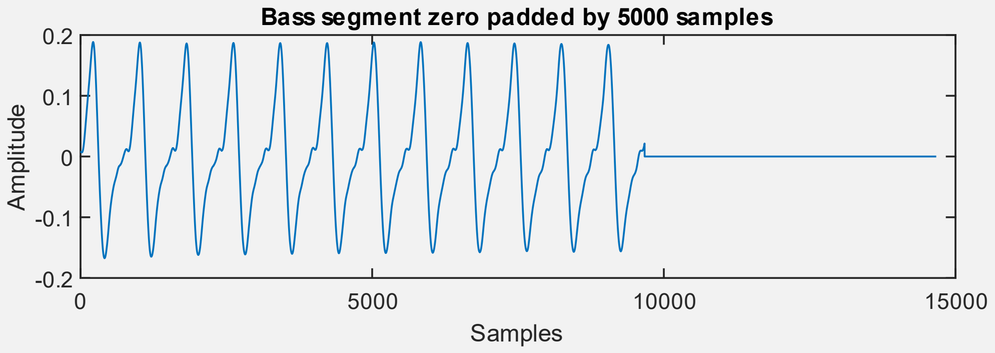
This difference between magnitude spectrums can be problematic since engineers and scientists are often looking for patterns in signals (in both time and frequency domains) to diagnose problems or to extract useful information. If the shape (or pattern) of the spectral information associated with a signal is heavily dependent on the length of the time-domain signal being analysed then the frequency-domain perspective becomes less useful in practice. Luckily techniques exist to deal with this issue!



The difference in the magnitude spectrums is caused by a phenomenon known as spectral leakage (spectral spreading), and in the next two sub-sections I'll explain what causes this phenomenon but before I do I'd like to show you how DSP practitioners deal with this issue using two techniques known as "windowing" and "zero-padding". Later on, I'll explain why these techniques achieve the desired effect, of ensuring that the length of the time-domain signal does not significantly impact the magnitude spectrum, but for now I'd just like you to see their effect and how straight-forward they are to implement.

Zero-padding is simply the process of appending samples of zero amplitude to a signal as show in the code and plot below. In this case I have appended 5000 samples of zero amplitude to the bass guitar segment. In other words, I have zero-padded the segment by 5000 samples.

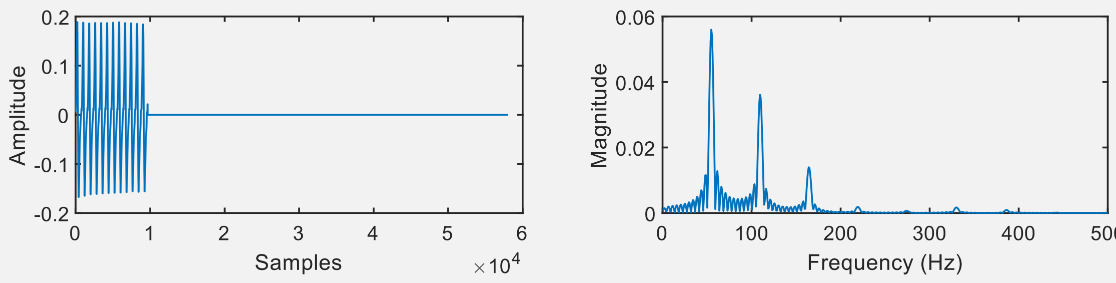
```
>> [ip fs]= audioread('bass_note.wav'); %from
pzdsp.com/sig1
>> N = 9670; %change this to 9270 for the second signal
>> seg = ip(10000:10000+N-1);
>> zpad_seg = [seg ; zeros(5000,1)]; %zero-padding
>> plot(zpad_seg)
>> xlabel('Samples');
>> ylabel('Amplitude');
>> title('Bass segment zero padded by 5000 samples')
```

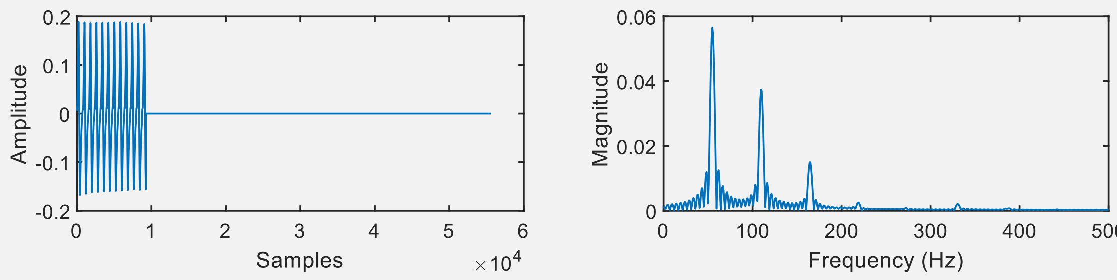


In practice it can be beneficial to zero-pad by much larger amounts. An appreciation for how much you should zero-pad will be provided later on in this document. The following code and plots show the impact of zero-padding the two bass segments dealt with earlier. In this case zero-padding is five times the length of the original signal.

```
>> [ip fs]= audioread('bass_note.wav');
>> N = 9670; %change this to 9270 for the second plot
>> seg = ip(10000:10000+N-1);
>> zpad_seg = [seg; zeros(N*5,1)]; %append N*5 zeros
>> plot(zpad_seg)
>> N2 = length(zpad_seg);
>> xlabel('Samples');
>> ylabel('Amplitude');
>> fax = [0:N2-1]*fs/N2;
>> plot(fax, abs(fft(zpad_seg))/N);
>> xlabel('Frequency (Hz)');
>> ylabel('Magnitude');
>> xlim([0 500]) % just show first 500 Hz
```

The time-domain plots below are the same as the ones shown below except they are zero-padded i.e. one time-domain signal contains 9670 samples of the bass signal while the other contains 9270. You can see that the frequency content of both zero-padded signals look much more similar than the frequency-domain plots above. They will become even more similar if the amount of zero-padding is increased, with the increase in zero-padding resulting in a greater number of fft computations being required i.e. it will take the fft longer to compute!

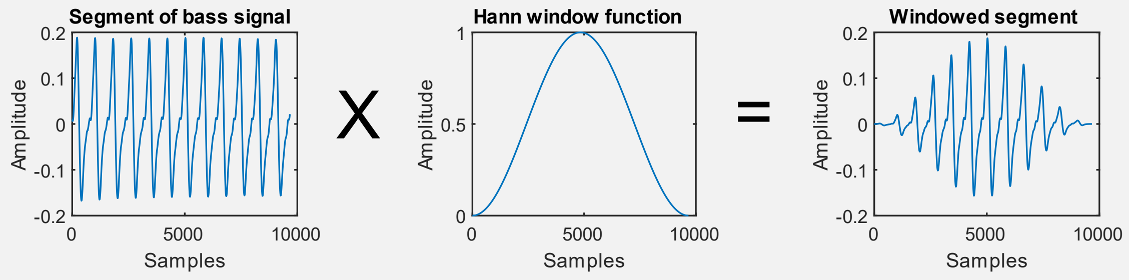




You'll notice the frequency domain plots have 3 relatively wide 'peaks' (referred to as main-lobes) centred at the fundamental (55 Hz) and the first two harmonics (110 Hz and 165 Hz). There are also numerous narrower and smaller 'peaks' (referred to as side-lobes) either side of the main-lobes. These side-lobes are considered as interference and can be problematic when taking measurements and reducing them is often desirable. This desired reduction in the magnitude of the side-lobes can be achieved using a processing called 'windowing'.

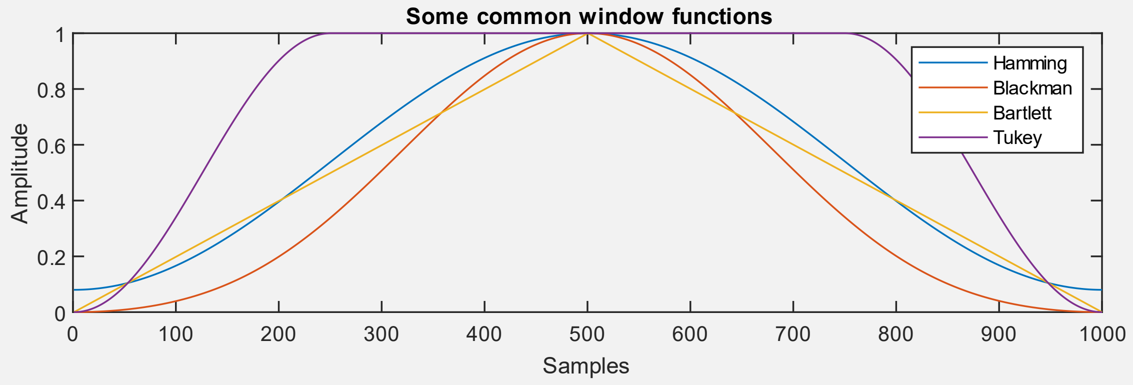
Windowing a signal involves multiplying it by another signal (referred to as a windowing function) to produce a new windowed signal, as shown in the code and plots below.

```
>> [ip fs]= audioread('bass_note.wav');
>> N = 9670;
>> seg = ip(10000:10000+N-1);
>> subplot(1,3,1);plot(seg)
>> xlabel('Samples'); ylabel('Amplitude');
>> title('Segment of bass signal')
>> hanning_win = hanning(N); % built-in function
>> subplot(1,3,2);plot(hanning_win)
>> xlabel('Samples'); ylabel('Amplitude');
>> title('Hann window function')
>> windowed_seg = seg.*hanning_win;
>> subplot(1,3,3);plot(windowed_seg)
>> xlabel('Samples'); ylabel('Amplitude');
>> title('Windowed segment')
```

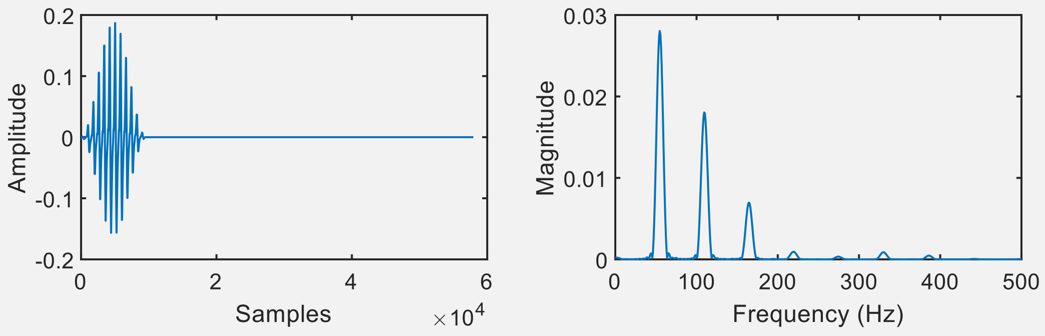


As can be seen above, the effect of windowing a signal is to alter the shape of a signal so that the start and end of the original signal are closer to zero. Windowing has the effect of superimposing an 'amplitude envelope' on the original signal which matches the shape of the window function. The example above makes use of the Hann window function but there are a large number of window functions with some common ones shown below. You'll notice that all the window functions shown taper off towards the beginning and end.

```
>> plot(hamming(1000));
>> hold on
>> plot(blackman(1000));
>> plot(bartlett(1000));
>> plot(tukeywin(1000));
>> ylabel('Amplitude'); xlabel('Samples')
>> legend({'Hamming', 'Blackman','Bartlett','Tukey'})
```

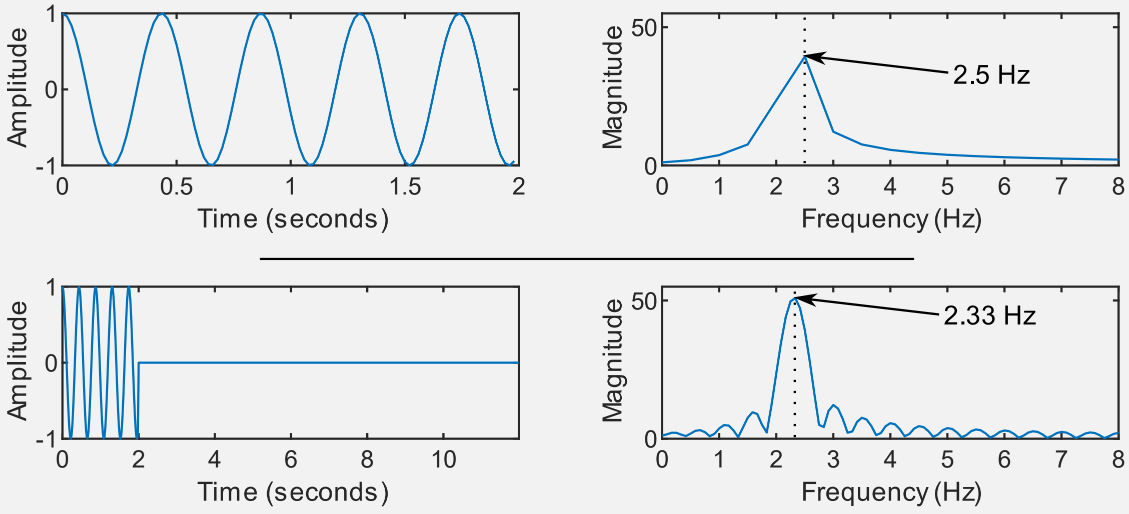


The code below shows the effect of windowing the bass guitar segment has in the frequency-domain, whereby the side-lobes, discussed above, are reduced. Notice, however, that the amplitudes of the main-lobes are wider and that they are also lower in amplitude, compared to the plots provided earlier (they are half the amplitude), as a consequence of the windowing process. A Hann window function is used in this example and if a different window was used you would find that there would be different levels of side-lobe attenuation and main-lobe width. There is generally a trade-off in terms of main-lobe width against side-lobe magnitude in the choice of window i.e. a narrow main-lobe (typically desirable) comes at the expense of more significant side-lobes (typically undesirable).



Windowing and zero-padding are reasonably easy to implement and their effect is effect is quite dramatic, so they are both routinely used in practice. By windowing and zero-padding DSP practitioners will produce consistent looking frequency-domain plots in which frequency components will not significantly interfere with other components. In many cases it can be sufficient just to be aware of the benefits of these two techniques without fully appreciating why side-lobe artefacts occur and how they can be 'controlled'. The next sections provide more detail on how and why zero-padding and windowing work.

Before moving on to the last few sections it is worth noting another practical benefit of zero-padding, which relates to our ability to estimate the frequency of sinusoidal components using the DFT. The code below synthesises a sinusoidal segment of frequency 2.3 Hz and duration 2 seconds, using a sampling rate of 50 Hz, and shows the magnitude spectrums of this segment for two cases, with and without zero padding. You'll notice that the location of the maximum peak is at 2.5 Hz for the non-zero padded spectrum while it is closer to the actual frequency of 2.3 Hz (highlighted with the dotted line in the figure) when zero-padding is employed. By zero-padding the sinusoidal segment we are improving the DFT frequency resolution which, in turn, improves our ability to accurately determine the frequency of sinusoidal components present in a signal. The peak of the magnitude spectrum of the bottom plot (associated with the zero-padded signal) occurs at a frequency of 2.33 Hz which is much closer to the actual frequency of 2.3 Hz.

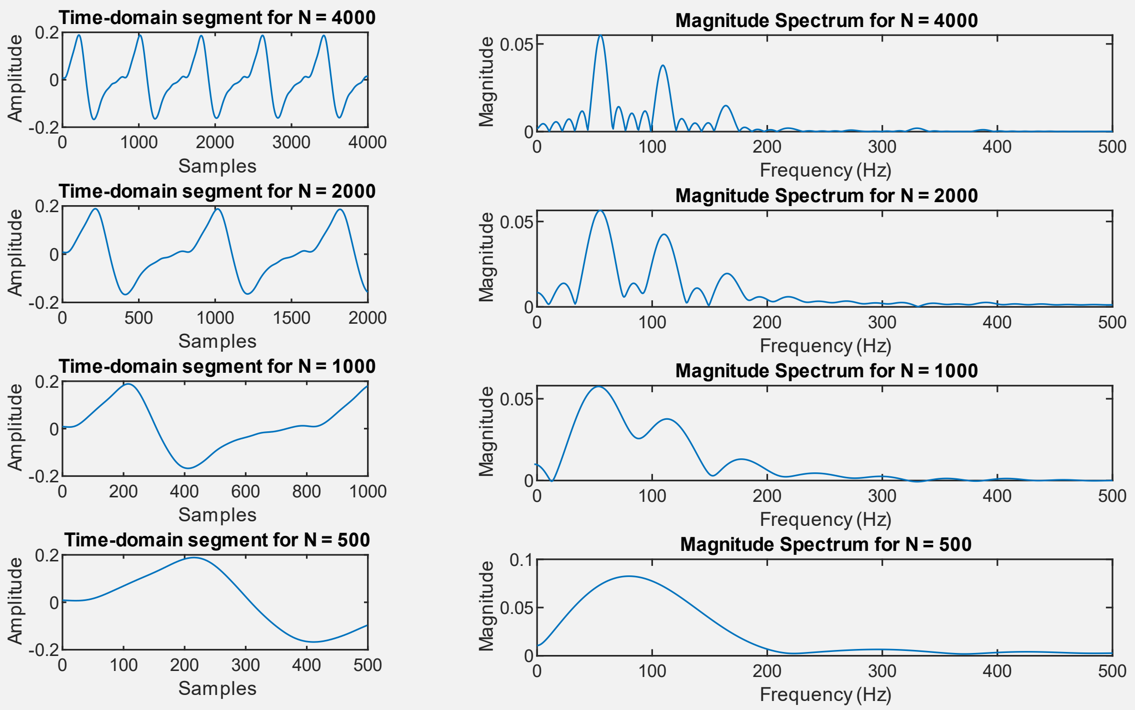


A further improvement in frequency estimate accuracy could be achieved by zero-padding by a greater amount. By zero-padding we are improving what is referred to as the DFT bin frequency resolution, which is given by *fs/N*, where *N* is the length of the zero-padded signal. The DFT bin frequency resolution dictates the maximum error in frequency measurements, so by increasing the length of the signal you can reduce the frequency estimate error. When no zero-padding is applied the bin resolution is 50/100 = 0.5 Hz (*fs* = 50; *N* = 100), however, by zero-padding by 500 samples *N* then becomes 600 so the bin resolution becomes a smaller value given by 50/600 = 0.0833 Hz.

Time-frequency resolution trade-off

The choice of segment length, prior to zero-padding, affects our ability to identify and analyse sinusoidal frequency components (see pzdsp.com/vid32 for an overview on this). The code below analyses segments of the bass guitar audio signal and can be used to produce the four magnitude spectrum plots below (the second line of the code should be modified to reproduce each of the four plots).

```
>> [ip fs]= audioread('bass_note.wav');
>> N = 4000; % change this to generate the spectrum plots
>> seg = ip(10000:10000+N-1);
>> zpad_seg = [seg; zeros(50000,1)]; % zero-pad
>> N2 = length(zpad_seg);
>> fax = [0:N2-1]*fs/N2;
>> plot(fax, abs(fft(zpad_seg))/N);
>> xlabel('Frequency (Hz)'); ylabel('Magnitude');
>> xlim([0 500]) % just show first 500 Hz
>> title(['Magnitude Spectrum for N = ' num2str(N)])
```

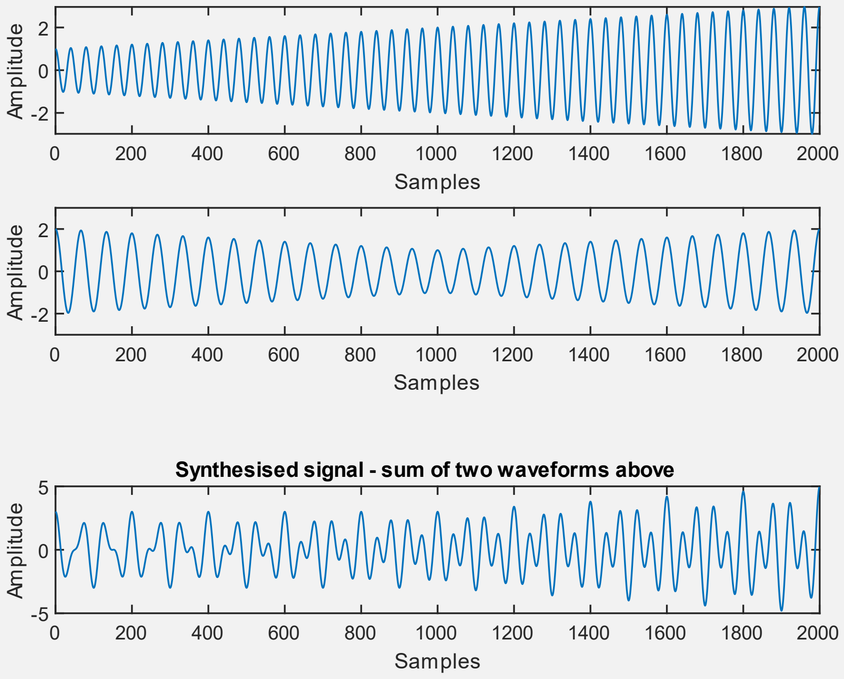


You'll notice in the four plots that it's more difficult to clearly see the fundamental and first two harmonics as the variable *N* (the segment length prior to zero-padding) gets smaller. This is because the main-lobe width gets wider, as do the side-lobes (an explanation of why this occurs is provided later in this document), and the separation between the fundamental and harmonics becomes less distinctive. When *N* is 500 (bottom right plot) the fundamental and harmonics cannot be clearly identified making it impossible to measure any frequency characteristics of the three harmonically related components. Even when *N* is 2000, when both the fundamental and harmonics can be clearly seen, you'll notice that the magnitude (height) of the main-lobe associated with the first harmonic has been increased in comparison to the top plot, when N=4000. This is predominantly due to interference from the side-lobes associated with the fundamental frequency. You should note that this interference could be reduced through an appropriate windowing process, as explained later in this document.

From the plots above its clear to see that if we have more time-domain samples we can get a clearer picture of the frequency-domain representation of that signal. Intuitively this should make sense, since the frequency-domain is essentially a measure of the rate of change of the time-domain and the more time-domain samples we have the more 'rate of change' between samples we can measure. Taking this perspective to its extreme we can consider how much frequency-domain information we could extract from a single sample i.e. when N = 1. The answer is 'very little' – just the DC level, which is of no real interest in isolation from other samples.

From the analysis above alone, it would appear that the length of segment should always be as long as possible, however if the segment is too long this can also cause problems in interpreting the magnitude spectrum. To appreciate why this is the case let's return to a synthesised example, whereby two sinusoidal waveforms are present but the amplitude of the waveforms are slowly changing, as shown in the code and corresponding time-domain plots below. Note that the variables `amp_envelope1` and `amp_envelope2` can be used to determine accurate amplitudes of the two sinusoidal waveforms.

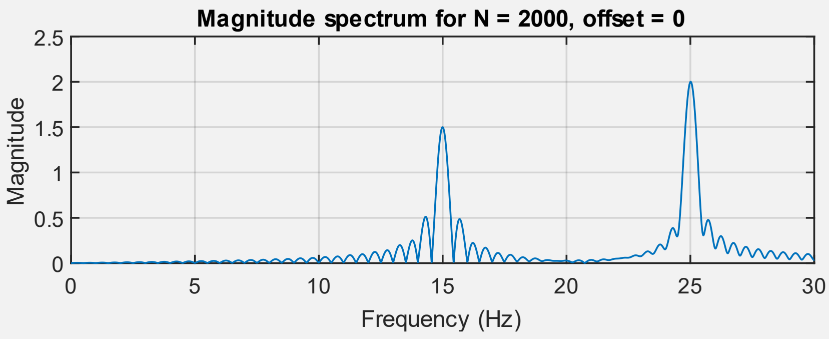
```
>> fs = 1000; N = 2000; t = [0:N-1]/fs;
>> amp_envelope1 = t+1;
>> amp_envelope2 = 2-t;
>> amp_envelope2(N/2+1:end)=fliplr(amp_envelope2(1:N/2));
>> waveform1 = cos(2*pi*25*t).*amp_envelope1;
>> waveform2 = cos(2*pi*15*t).*amp_envelope2;
>> synth_sig = waveform1+waveform2;
>> subplot(3,1,1); plot([0:N-1], waveform1);
>> xlabel('Samples');ylabel('Amplitude');
>> subplot(3,1,2); plot([0:N-1], waveform2);
>> xlabel('Samples');ylabel('Amplitude');
>> subplot(3,1,3); plot([0:N-1], synth_sig);
>> xlabel('Samples');ylabel('Amplitude');
>> title('Synthesised signal - sum of waveforms above')
```



For the purpose of understanding the impact in the choice of segment length, let's imagine a situation where we only have access to the synthesised signal and we want to determine the evolution of the amplitude of both sinusoidal waveforms using the DFT.

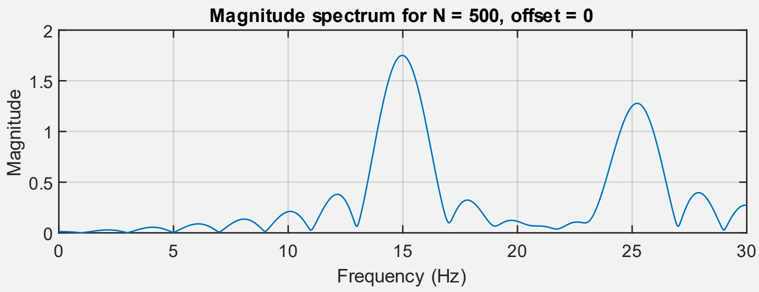
If we applied a DFT to the entire synthesised signal (bottom plot above) then the magnitude (height) of the two main-lobes in the magnitude spectrum would indicate the average amplitude of the two sinusoids, as shown by the code and plot below. We can see that the average amplitude of the 15 Hz frequency component is lower in amplitude than the 25 Hz component, which corresponds to our expectation since the average amplitude of the 25 Hz component is 2 (`mean(amp_envelope1)`), while the average amplitude of the 15 Hz component is 1.5 (`mean(amp_envelope2)`). This plot clearly cannot be used to determine how the amplitudes of the sinusoids evolve as it simply shows the average over the 2000 samples.

```
>> offset = 0; % change this to reproduce other plots
>> N = 2000; % change this to reproduce other plots
>> zpad_win_sig =  [synth_sig(1+offset:N+offset)
zeros(1,1000000)];
>> N2 = length(zpad_win_sig);
>> fax = [0:N2-1]*fs/N2;
>> plot(fax, abs(fft(zpad_win_sig))/(N/2))
>> xlim([0 30]); ylabel('Magnitude'); xlabel('Frequency
(Hz)')
>> title(['Magnitude spectrum for N = ' num2str(N)  ',
offset = ' num2str(offset)])
```

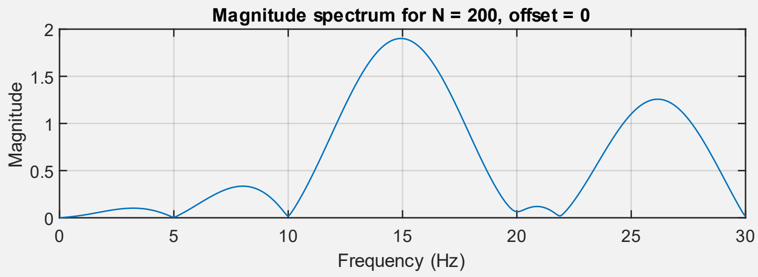


A better indication of the amplitude evolution of both sinusoidal components can be obtained by segmenting the waveform into smaller segments (these segments are also often referred to as windows or frames). The magnitude spectrum plot below is associated with the first 500 samples of the entire 2000 samples of the signal synthesised. The magnitude of the main-lobe associated with the 15 Hz component is approx 1.75 while the 25 Hz component is approx. 1.25. These magnitudes represent the average (approximately!) of the sinusoidal waveforms over the first 500 samples and also correspond to the amplitude of the sinusoidal waveforms at the centre of the segment i.e. at sample number 250, since the increasing and decreasing amplitudes either side of this centre point will have the effect of cancelling each other out. Notice, however, that the main-lobe associated with the 25 Hz component is not centred at exactly 25 Hz due to side-lobe interference associated with the 15 Hz component. This interference will increase if the segments get shorter since the main-lobe widths increase and their centres become closer to each other, as explained later in this document.

Note that the benefit of using synthesised signals is that we can determine the exact amplitudes of the sinusoids by referring to the code used to synthesise the signals. The amplitudes at sample 250 will be `amp_envelope1(250)`, which is 1.249, and `amp_envelope1(250)`, which is 1.751.



Let's see what happens when the segment length is further reduced to down to 200 samples. From analysis of the time-domain waveforms the amplitude of the 15 Hz sinusoid after 100 samples will be 1.1 and the 25 Hz sinusoid will be 1.9 (these amplitudes can also be determined from code using `amp_envelope1(100)` and `amp_envelope1(100))`. Examining the plot of magnitude spectrum of the first 200 samples, shown below, indicates that the magnitude of the 25 Hz component is about 1.25 rather than 1.1. Also notice how the centre of the main-lobe associated with the 25 Hz component is now shifted further away from 25 Hz to 26 Hz. Both these inaccuracies (in terms of amplitude and frequency) can be attributed to side-lobe interference associated with the 15 Hz component.



The key point to take away from these examples is that a shorter analysis segment will reduce the effect of averaging and can provide a more accurate estimate of the local amplitude of each sinusoidal component within the synthesised signal. However, shorter segments also result in more significant interference from side-lobes, and even main-lobes for "extremely" short segments (see segment length guidelines 2 paragraphs down for a sense of what constitutes an adequate length segment). While the windowing techniques, introduced earlier, and described in more detail later on, can reduce the effects of side-lobe interference their effects will always be somewhat present. Note that for extremely short segments the main-lobes associated with components will interfere with each other, and meaningful frequency measurements can become impossible to determine.

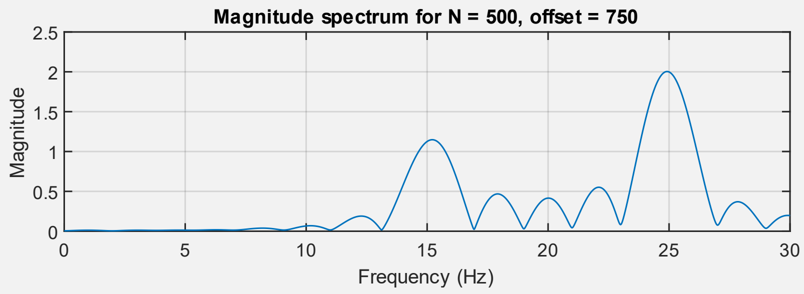
This trade-off in terms of the desire to choose a short segment to reduce the impact of averaging with the desire to minimise interference by choosing a longer segment is referred to as a time-frequency resolution trade-off. A long segment length improves frequency resolution (our ability to visually 'space-out' or 'separate' components along the frequency axis, which also impacts the interference between components) at the cost of more accurate time resolution (our ability to accurately estimate the characteristics of the frequency components at a particular time).

A rough guideline for selecting a minimum segment length is to ensure that the segment length equates to twice the period of the lowest frequency component present in the signal, for periodic signals. When dealing with a number of non-harmonically related quasi-sinusoidal waveforms you would need to determine the 'likely' frequency of all the waveforms and find the smallest frequency difference between these sinusoids, then the segment length should be twice the inverse of this minimum frequency difference. For example, if sinusoidal components of 100 Hz, 107 Hz and 109 Hz where present in a signal the segment length should equate to 0.5 seconds, since the smallest frequency difference is 2 Hz (109– 107 Hz).

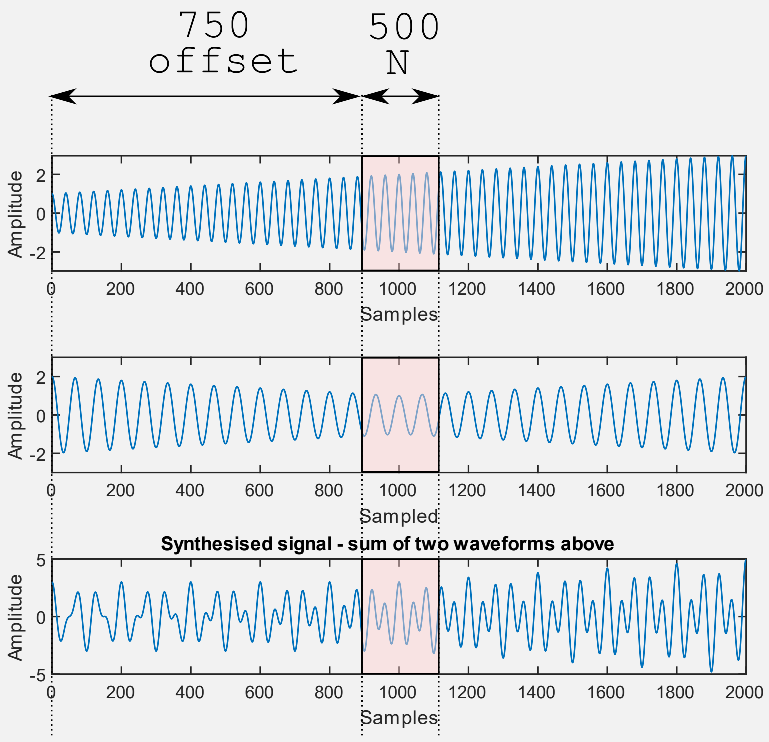
Of course, these guidelines depend on prior knowledge of the frequencies present in the signal and engineers and scientists may rely on expert input for this information. For example, with ECG (heart signal) analysis scientists would rely on medical experts knowledge of the physiological limits of human heart rate. Audio engineers might rely on the frequency range of certain musical genres in their analysis. In the event that you had little or no prior knowledge of the expected frequency characteristic of a signal, estimates of the frequency components present could be determined experimentally using the DFT's using a variety of segment lengths, with the optimum segment length determined through refined experimentation.

Before concluding this section, I would like to highlight that the amplitude evolution of the synthesised signal can be determined by adjusting the `offset` variable in the first line of the code provided above. By incrementing the `offset` variable you will be able to estimate the amplitudes of the two sinusoidal components at different times/samples.

I would also like to point out that the magnitude of the main-lobes won't always correspond to the amplitude of the sinusoidal waveforms at the centre of the segment selected (even taking side-lobe interference into consideration!). In the example below, the offset is 750 and the segment length is 500, and it can seen that magnitudes of the main-lobes are 1.2 for the 15 Hz component and 2 for the 25 Hz component. The actual amplitudes of the sinusoidal waveforms at the centre of the segment are 1 (i.e. `amp_envelope2(1000)`) and 2 (i.e. `amp_envelope1(1000)`). The reason the 15 Hz measurement is incorrect is because the average amplitude of the waveform is greater than 1, which should be apparent from a visual inspection. This contrasts with the 25 Hz component in which the average amplitude is equal to the amplitude at the centre of the segment.

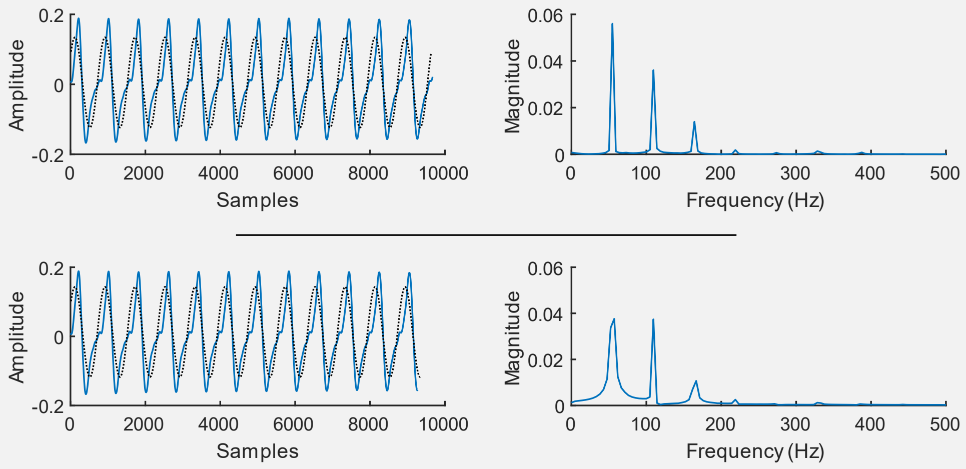


The plot below explicitly highlights the time-domain segment associated with the magnitude spectrum shown above, highlighted in pink.



Spectral leakage

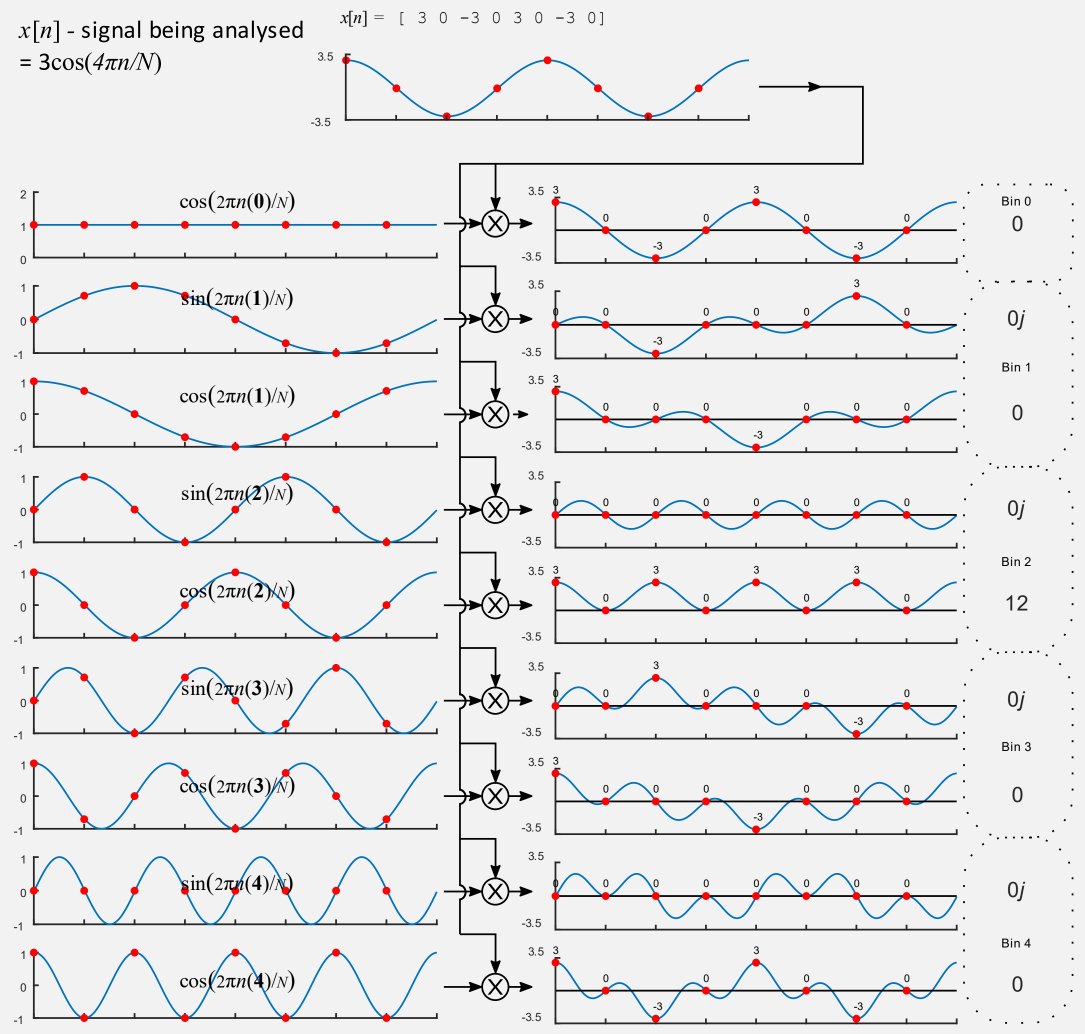
Spectral leakage refers to a spread of energy across the frequency axis. Referring to the figure below (a modified version of a figure shown earlier) the magnitude spectrum in the bottom right shows a spread of energy across a wider range of DFT bins in comparison to the spectrum in the top right, particularly in the region of the fundamental frequency of 55 Hz. The figure has been adapted from the earlier one to include the sinusoidal waveform associated with the fundamental frequency (shown as a dotted line). What you should notice is that in the top left time-domain plot the fundamental has close to 12 complete cycles, while the bottom left plot contains 11.5 cycles of the fundamental. You'll observe a significant spread of energy across DFT bins if there are sinusoidal components present in a signal that do not have an integer number of cycles within the length of the signal being analysed by the DFT. The reason for this can be appreciated by returning to a few synthesised examples.



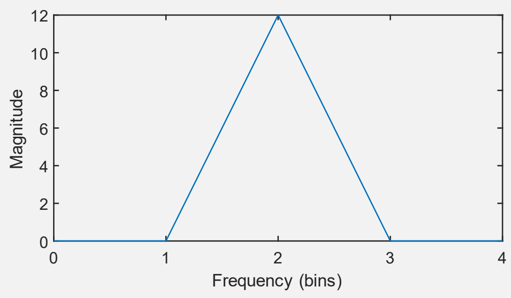
When we looked at how the DFT works earlier we saw that when a sinusoidal segment which had an integer k number of cycles over N samples was analysed then DFT bin k was non-zero. This is because there is a strong correlation between the sinusoidal waveform being analysed and the DFT basis functions which also have k cycles over N samples. In the example below (page 52), bin 2 is non-zero, while the other bins shown are zero.

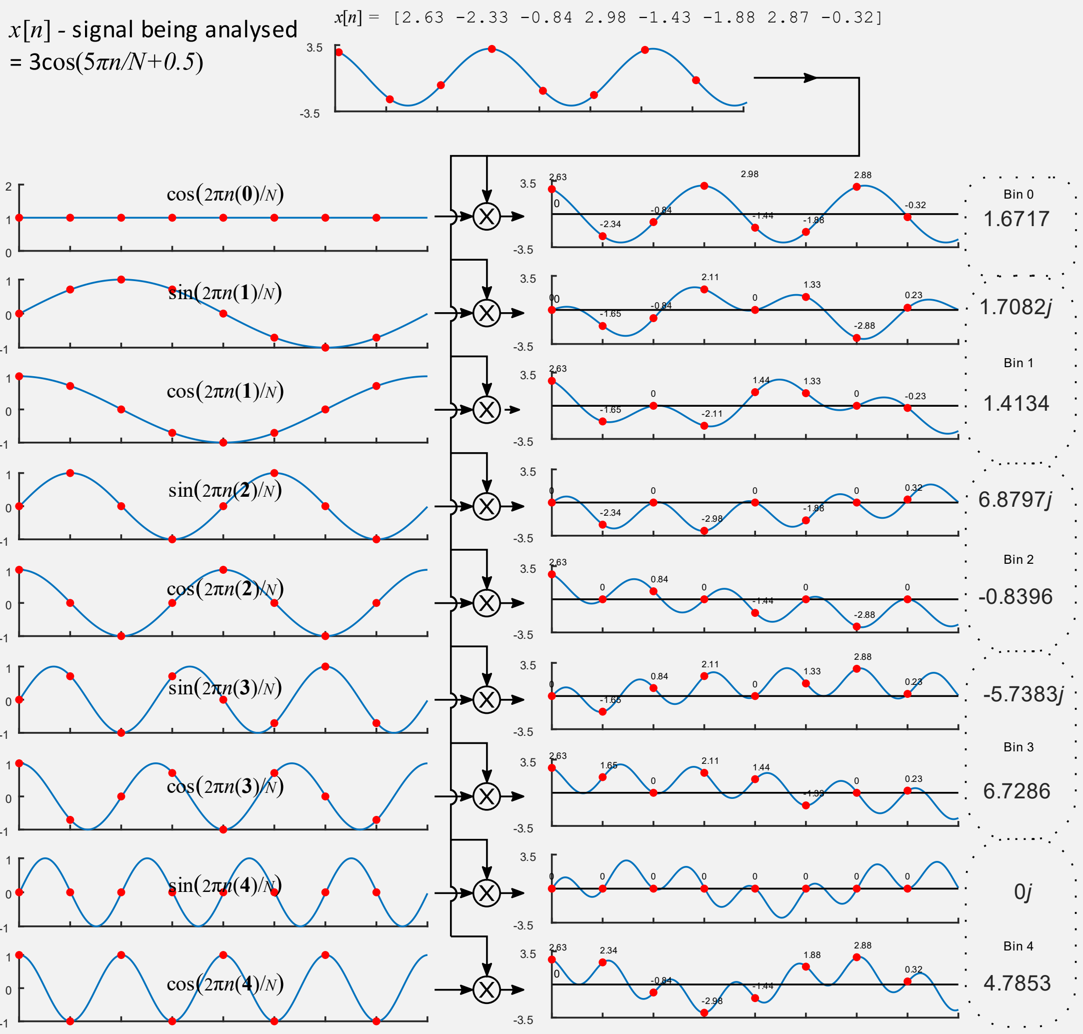
When we are dealing with a sinusoidal segment that contains a non-integer number of cycles (page 53) then the waveform will correlate with a number of DFT basis functions. Consider the next example in which the waveform being analysed is sinusoidal and has 2.5 cycles over 8 samples. In this case the DFT basis functions that have two cycles and three cycles over 8 samples will be strongly correlated with this waveform and therefore will result in non-zero values for both bins 2 and 3. Bins 0, 1 and 4 are also correlated but to a lesser extent.

$x[n]$ - signal being analysed
= 3cos($4\pi n/N$)

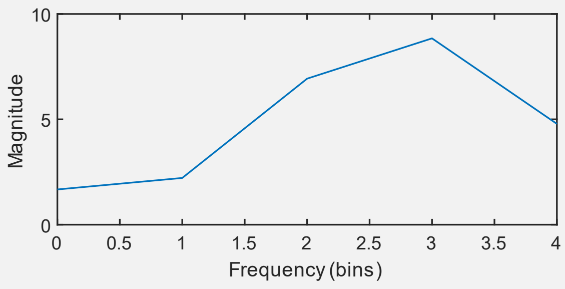


```
>> x = [3 0 -3 0 3 0 -3 0]; %3*cos(4*pi*n/8)
>> X = fft(x);
>> plot([0:4],abs(X(1:5)))
>> xlabel('Frequency (bins)'); ylabel('Magnitude');
```





```
>> n = 0:7; x = 3*cos(5*pi*n/8+0.5);
>> X = fft(x);
>> plot([0:4],abs(X(1:5)))
>> xlabel('Frequency (bins)'); ylabel('Magnitude');
```

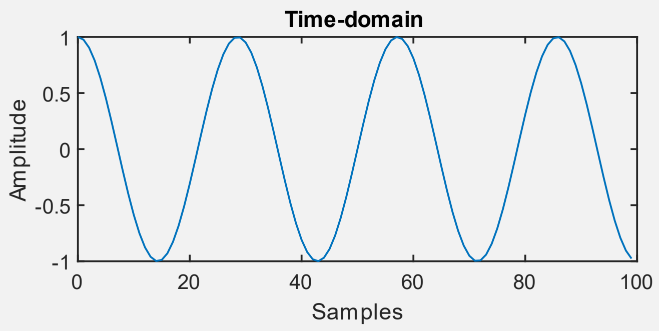


The important point to take from the above examples is that whenever you analyse a signal that contains sinusoidal components that do not have an integer number of cycles over the length of the signal being analysed by the DFT, then you will get a spread of spectral energy across a number of DFT bins. The reason this spread of energy occurs is because the signal being analysed is correlated with a number of basis functions.

One of the problem's with spectral leakage is that the amount of spectral spread is dependent upon the frequency of the signal being analysed, and since we generally don't know the frequency of components in the signal being analysed then the shape of the spectrum becomes unpredictable and more difficult to interpret. As we saw earlier, this problem can be alleviated by zero-padding and in the following subsection we'll try to get a better understanding of how and why zero-padding works. A video explanation on zero-padding is available at pzdsp.com/vid30.

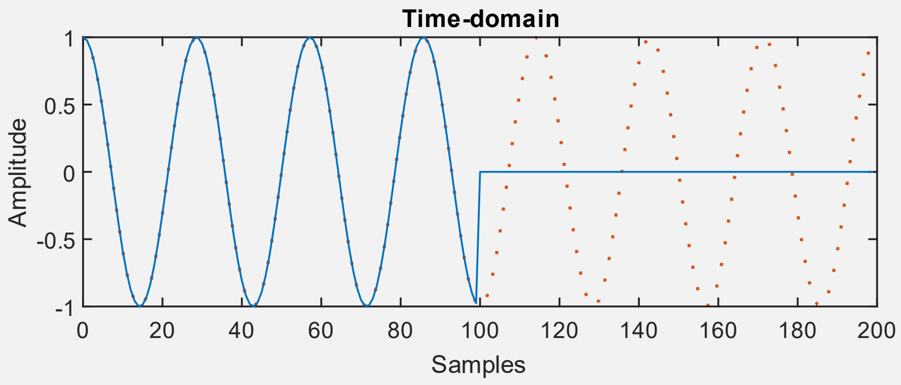
How and why zero-padding works

Before you can appreciate why zero-padding works you must first appreciate that the DFT essentially correlates, or measures the similarity of, a signal of interest with sinusoidal basis functions **that only have an integer number of cycles over the length of the signal being analysed** (as described earlier)**.** Consider the case where the signal being analysed by the DFT is a segment of a sinusoidal waveform which contains 3.5 cycles over 100 samples, as Illustrated below. If the DFT is applied to this signal none of the basis functions will 'match' the frequency of this signal, since the DFT is restricted to correlating (which is a mathematical method of comparing/matching) the signal with basis functions that have an integer number of cycles over 100 samples. The closest frequency basis functions are those with 3 and 4 cycles over 100 samples.

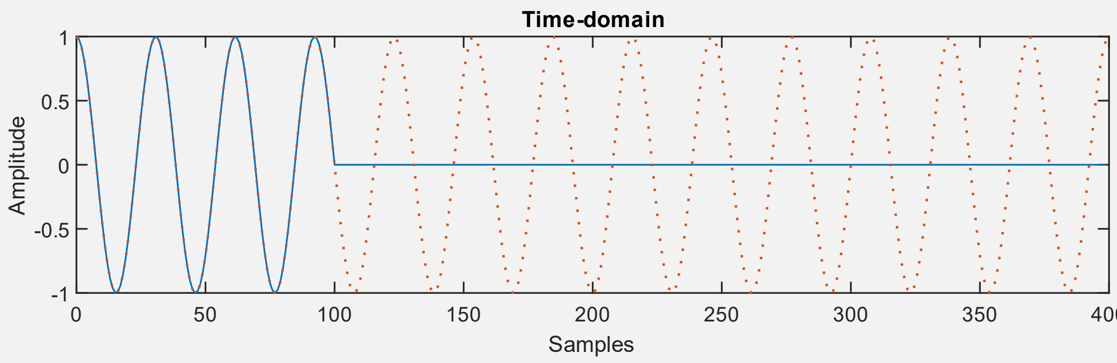


At this point it's worth pointing out that the frequency of a sinusoid with k cycles over N samples has k cycles over $N.T$ (or N/fs) seconds, where T is the sampling interval (in seconds) and is equal to 1/*fs* where *fs* is the sampling rate, in units of Hertz. We can therefore express the frequency of a sinusoid with *k* cycles over N samples as *k.fs*/*N* Hz. While we don't explicitly need to specify a sampling rate it can be helpful to use one to make the numbers more meaningful, so for the example above if the sampling rate was 1000 Hz the sinusoidal waveform would be 35 Hz.

Now let's consider the case where the signal is zero-padded by 100 samples, as shown by the blue waveform in the plot below. When the DFT is applied to this zero-padded signal the basis function that has 7 cycles over 200 samples (shown as a dotted waveform in the figure below) will 'match' the frequency of the zero-padded signal. Zero-padding the signal to twice its original length results in twice the number of basis functions being correlated. This increase in the number of basis function provides the DFT with a greater opportunity to 'match' the frequency of components in the signal being analysed. Note that the frequency of the basis function with 7 cycles over 200 samples is equivalent to 35 Hz, for a sampling rate of 1000 Hz.



Now consider a third case where the DFT is analysing a segment of a sinusoidal waveform that has 3.25 cycles over 100 samples (32.5 Hz, for a sampling rate of 1000 Hz). How many zeros would need to be appended to this signal to provide the DFT with an opportunity to 'match' this frequency component? If you're thinking 300 then you're correct! If you zero-padded by 300 the signal will become 400 samples long and the basis function that has 13 cycles over 400 samples will 'match' the frequency of the signal of interest (as shown in the plot below). Note that a basis function with 13 cycles over 400 samples has a frequency of 32.5 Hz, for a sampling rate of 1000 Hz). You should also appreciate that zero padding by 700, 1100, 1500, etc. would also provide the DFT with an opportunity to 'match' a basis function to the frequency of the signal of interest in this case.

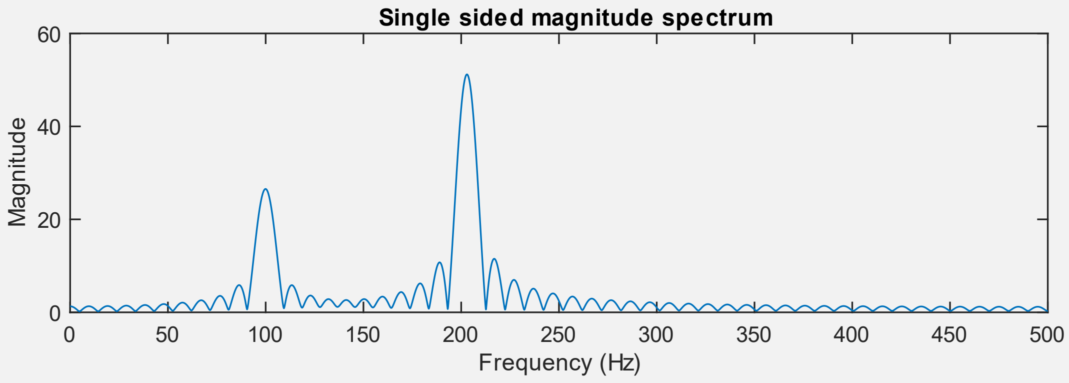


It's important to appreciate that the more you zero-pad a signal the greater the opportunity the DFT has of accurately 'matching' the frequency components present in a signal. Also, if you were able to zero-pad by an infinite amount then the DFT would have the opportunity to 'match' any sinusoidal frequency component. Of course, zero-padding by an infinite amount isn't possible in practice but by going through this thought experiment you can appreciate why it can be useful to zero-pad by 'large' amounts (within the computation constraints of your computer).

When we zero-pad by 'large' amounts a predictable pattern in the magnitude spectrum emerges. You'll notice a particular 'gherkin-like' shape (referred to as a main-lobe) appear in the region of each sinusoidal component present in the signal. This main-lobe is accompanied by smaller and narrower side-lobes. We've seen this shape earlier when we examined the bass guitar signal, and it also appears in the synthesised example shown below.

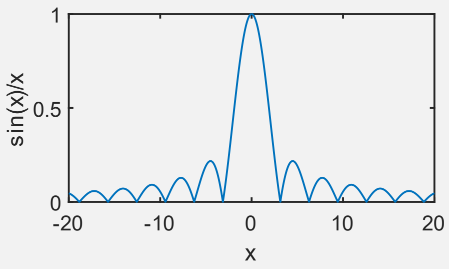
The following code synthesises a signal that is comprised of two sinusoidal segments of frequencies 100 Hz (i.e. 10 cycles over 100 samples at a sampling rate of 1000 Hz) and 200 Hz. The lower frequency sinusoid is half the amplitude of the higher one. One million samples are then padded to the signal prior to applying a DFT, using the fft function.

```
>> N = 100;n = 0:N-1;
>> fs = 1000;
>> x = 0.5*cos(2*pi*10*n/N)+cos(2*pi*20.3*n/N);
>> x_zpad = [x zeros(1,1000000)];
>> N_zpad = length(x_zpad);
>> fax_zpad = [0:N_zpad-1]*fs/N_zpad;
>> plot(fax_zpad, abs(fft(x_zpad)))
>> xlim([0 fs/2])
>> xlabel('Frequency (Hz)'); ylabel('Magnitude');
```



This shape associated with each sinusoidal component is the magnitude of a sinc function i.e. sin(x)/x, as illustrated by the code and plot below.

```
>> x = -20:.01:20;
>> plot(x,abs(sin(x)./x))
>> xlabel('x')
>> ylabel('sin(x)/x')
```



The reason why this shape, as opposed to any other shape, appears at the location of each sinusoidal component will be explained in the next section on windowing.

At this stage I would like to highlight an interesting feature relating to the spectrum associated with the heavily zero-padded signal. If you were to take the DFT of the non-zero-padded signal and examine any of the DFT bin values returned, you would find that a corresponding bin value also exists in the spectrum associated with the heavily zero-padded signal. For example, consider DFT bin number 10 of the non-zero padded signal, which is associated with a frequency of 100 Hz (10 cycles over 100 samples at a sampling rate of 1000 Hz). The value of bin 10 is 26.5328 + 0.7427j (see code below), and the magnitude of this complex number is 26.5432. The same magnitude could be determined by reading the magnitude spectrum of the heavily zero-padded signal at a frequency of 100 Hz (remember bin 10 is associated with a frequency of 10*fs*/N = 100 Hz).

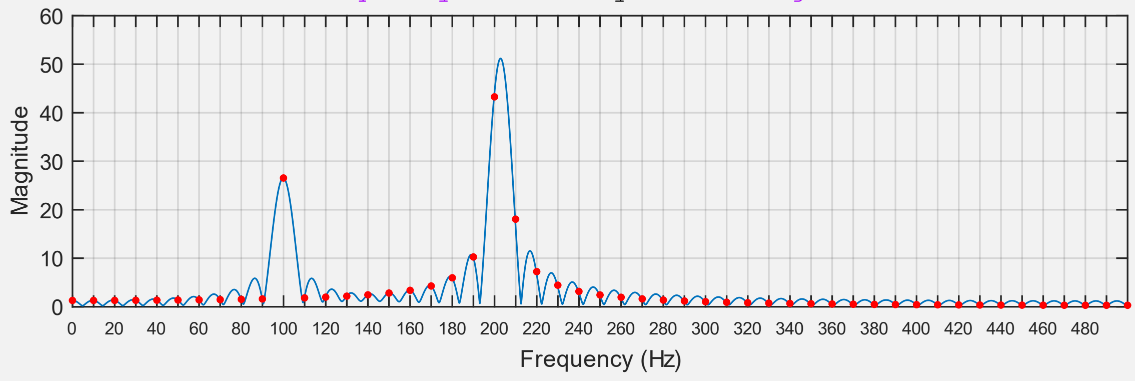
```
>> N = 100; n = 0:N-1; fs = 1000;
>> x = [0.5*cos(2*pi*10*n/N)+cos(2*pi*20.3*n/N)];
>> X = fft(x);
>> abs(X(11)) % bin 10 is at index 11

         26.5432
```

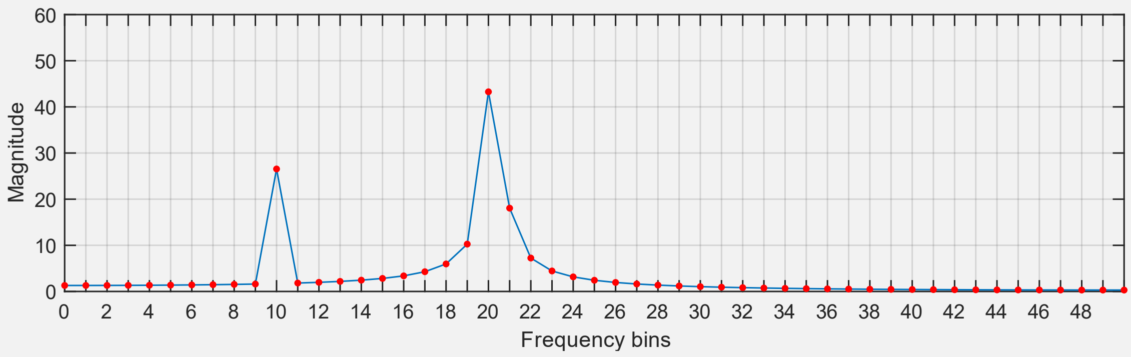
The same approach could be taken for any bin. If we consider that bin *k* is associated with a frequency of *k.fs/N* then bins 0,1,2, …, …, 48, 49 are associate with frequencies 0 Hz, 10 Hz, 20 Hz, …, …, 480 Hz, 490 Hz, since *fs* =1000 Hz and *N* = 100. We could therefore determine the DFT bin magnitudes associated with the non-zero padded signal by simply measuring the magnitudes of the spectrum associated with the heavily zero-padded spectrum at the frequencies listed above.

The code and plot below reinforce this concept that the DFT of a non-zero padded signal is a sampled subset of the DFT associated with the same signal which has been heavily zero-padded. The red dots on the plot correspond the DFT of the non-zero padded signal, all of which lie on magnitude spectrum associated with the DFT of the heavily zero-padded signal.

```
>> N = 100;
>> n = 0:N-1;
>> fs = 1000;
>> x = 0.5*cos(2*pi*10*n/N)+cos(2*pi*20.3*n/N);
>> x_zpad = [x zeros(1,1000000)];
>> N_zpad = length(x_zpad);
>> fax_zpad = [0:N_zpad-1]*fs/N_zpad;
>> plot(fax_zpad, abs(fft(x_zpad)));
>> xlim([0 fs/2])
>> xlabel('Frequency (Hz)'); ylabel('Magnitude');
>> hold on; fax = [0:N-1]*fs/N;
>> plot(fax,abs(fft(x)),'r.', 'Markersize',10)
>> figure
>> plot([0:N-1], abs(fft(x))); hold on
>> plot([0:N-1], abs(fft(x)),'r.','Markersize',10)
>> xlabel('Frequency bins'); ylabel('Magnitude');
```

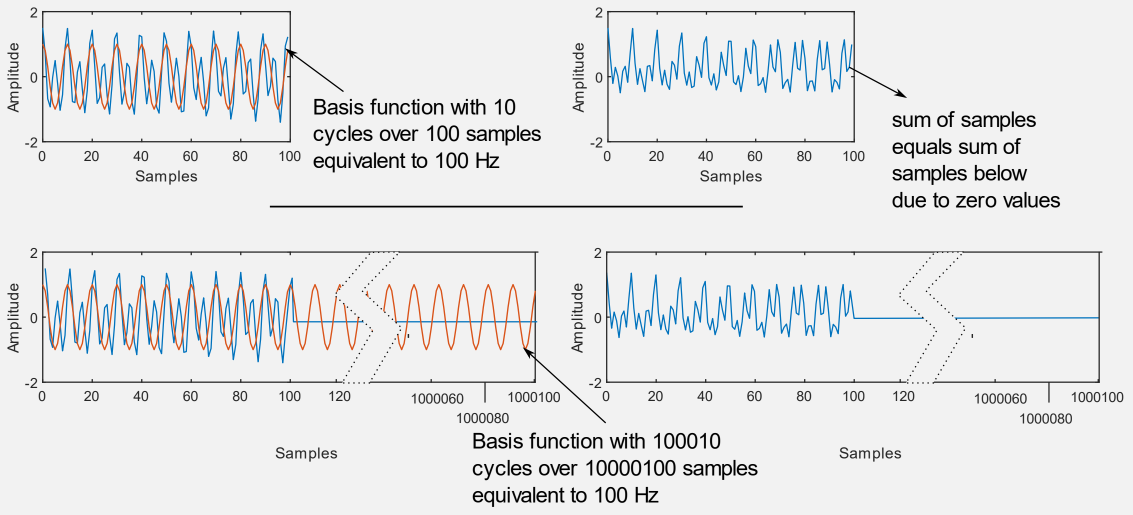


The next plot below is just the DFT of the non-zero padded signal, shown in isolation for comparison purposes.



To appreciate why the DFT of the non-zero padded signal can be considered as a sampled version of the DFT of a heavily zero-padded signal lets take a closer look at the DFT in operation for one bin in particular i.e. bin 10. To determine the DFT value of bin 10 we need to multiply the signal by a cosine basis function which has 10 cycles over 100 samples (we would also need to multiply a sine wave of the same frequency but for the purpose of this explanation it isn't necessary). The plot below shows both the signal and the cosine basis function to the left and the result of the multiplication to the right. The plots further down are zero-padded equivalents illustrating 1 million zero valued samples being appended to the signal.

```
>> N = 100;n = 0:N-1;
>> fs = 1000;
>> x = 0.5*cos(2*pi*10*n/N)+cos(2*pi*20.3*n/N);
>> cosine_basis_10 = cos(2*pi*10*n/N);
>> subplot(1,4,1); plot(n,x); hold on
>> plot(n,cosine_basis_10)
>> xlabel('Samples'); ylabel('Amplitude');
>> subplot(1,4,3);
>> plot(n, cosine_basis_10.*x)
>> xlabel('Samples'); ylabel('Amplitude');
>> ylim([-2 2])
>> sum(cosine_basis_10.*x)
     26.5328
```



```
>> x_zpad = [x zeros(1,1000000)];
>> N_zpad = length(x_zpad);
>> n2 = 0: N_zpad-1;
>> cosine_basis_100010 = cos(2*pi*100010*n2/N_zpad);
>> sum(cosine_basis_100010.*x_zpad)
      26.5328
```

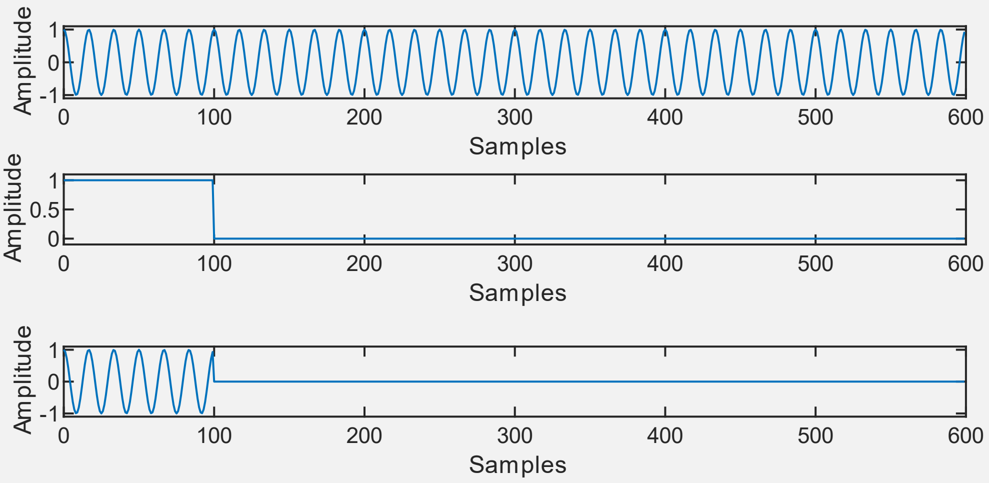
What you need to appreciate is that the value of DFT bin number 10 of the non zero-padded signal is identical to the value of bin number 100010 of the zero-padded signal because the 1 million zeros in the bottom right plot above accumulate to zero and don't affect the DFT result. The same process could be undertaken for any DFT bin, there's nothing particularly special about bin 10. With this in mind you should appreciate that any of the DFT bin values associated with the non zero-padded signal have an equivalent bin value in the DFT applied to the heavily zero-padded signal.

The significance of the above discussion is that while the magnitude spectrum associated with the DFT of non-zero padded signals can appear somewhat unpredictable, since they are influenced by the length of signal being analysed, there is an underlying structure when you consider the DFT of a signal as being a sampled version of the DFT of the same signal which has been infinitely zero-padded.

How and why windowing affects main-lobes and side-lobes

The key to understanding how and why windowing affects main-lobes and side-lobes is to consider that the bottom plot, shown below, is equivalent to the top plot multiplied by the middle plot (see pzdsp.com/vid31). You'll recognise that the bottom plot can also be considered as being a zero-padded version of a sinusoidal segment which has 6 cycles over 100 samples. The middle plot is a rectangular pulse function, which is simply a sequence of ones followed by a sequence of zeros. So, the process of zero-padding can also be considered as a process of multiplying one waveform by a rectangular pulse waveform. This rectangular pulse waveform is also referred to as a rectangular window function.

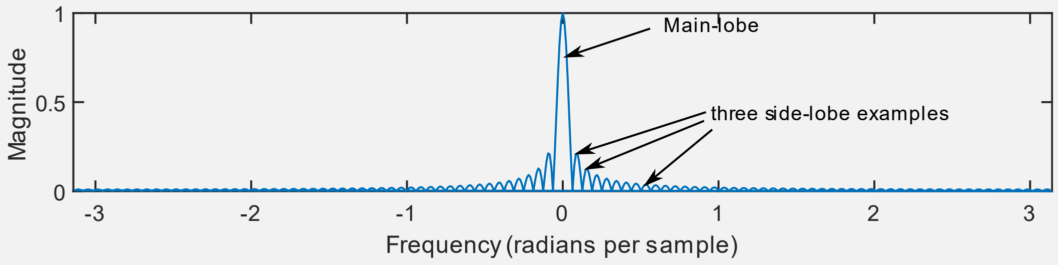
```
>> N = 100;  n = 0: N-1;
>> N_zeros = 500;
>> n = 0:N+N_zeros-1;
>> x_top = cos(2*pi*6*n/N);
>> rect_win = [ones(1,N) zeros(1,N_zeros)];
>> x_zpad = x_top.*rect_win;
>> subplot(3,1,1); plot(n, x_top);
>> xlabel('Samples');ylabel('Amplitude');
>> subplot(3,1,2); plot(n, rect_win);
>> xlabel('Samples');ylabel('Amplitude');
>> subplot(3,1,3); plot(n, x_zpad);
>> xlabel('Samples');ylabel('Amplitude');
```



It should be appreciated that this perspective of zero-padding (as being a process of multiplying two waveforms) is valid for any signal, not just the sinusoidal example shown above.

If we plot the double-sided magnitude spectrum of the rectangular pulse function you'll notice that the shape of the spectrum has a wide main-lobe and narrow side-lobes, in terms of frequency. The main-lobe is twice the width of the side-lobes.

```
>> W = fft(rect_win);
>> fax = [0:length(W)-1]*2*pi/length(W);
>> inds = find(fax>pi); fax(inds) = fax(inds)-2*pi;
>> plot(fax, abs(W)/N);
>> xlabel('Frequency (radians per sample)')
>> ylabel('Magnitude');
```

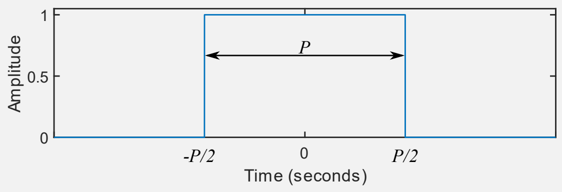


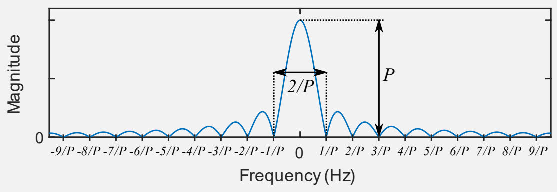
This spectral shape is the magnitude of a sinc-like function whereby sinc(x) = sin(x)/x. Taking a Fourier transform of a rectangular pulse reveals where this shape derives from mathematically.

For the following derivation it's convenient to centre the pulse of width P seconds around time t=0 and then the time-shifting property of the Fourier transform (i.e. $\mathscr{F}\{f(t\text{-P}/2)\} = \mathscr{F}\{f(t)\, e^{-j\omega P/2}\}$) could be used to apply the relevant phase shift. Note, however, that the time-shifting property does not impact the magnitude spectrum i.e. $|\mathscr{F}\{f(t\text{-P}/2)\}| = |\mathscr{F}\{f(t)\}|$, so it is not a required step to understand the magnitude spectrum features.

$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t} = \int_{-\frac{P}{2}}^{\frac{P}{2}} e^{-j\omega t}$, since $x(t)$ is 1 from $-P/2$ to $P/2$ and zero elsewhere.

Therefore, $X(\omega) = \frac{e^{-j\omega\frac{P}{2}} - e^{j\omega\frac{P}{2}}}{-j\omega} = \frac{2\sin\left(\omega\frac{P}{2}\right)}{\omega} = \frac{\sin(\pi f P)}{\pi f}$, since $\omega = 2\pi f$

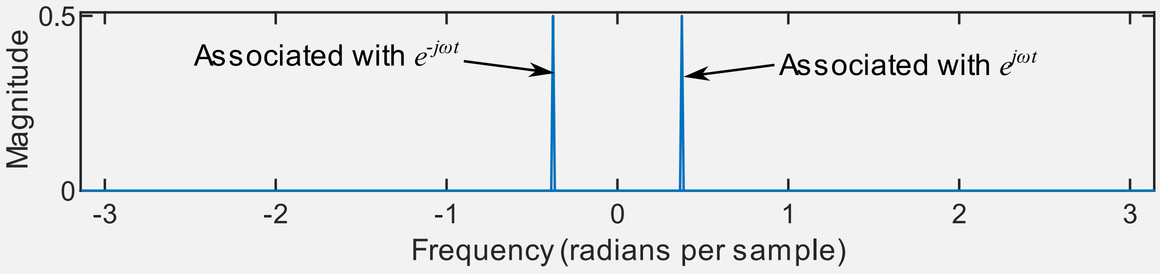




From this analysis you should appreciate that the width of the main-lobe is inversely proportional to the length of the pulse i.e. the main-lobe becomes narrower in frequency as the pulse length gets longer in time, as we have seen previously in the section entitled "Time-frequency resolution trade-off".

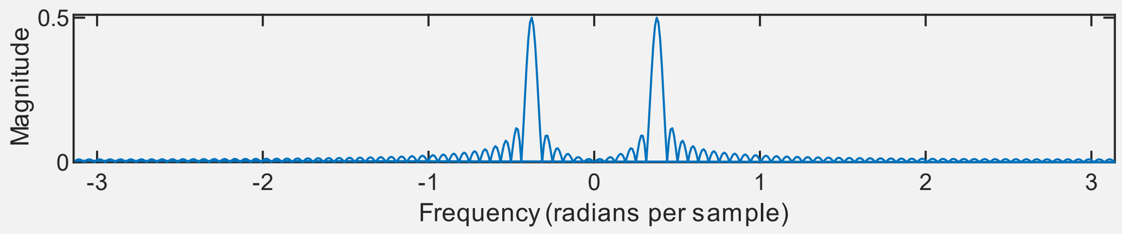
If we plot the double-sided magnitude spectrum of the sinusoidal waveform in the top plot we can see two 'spikes' which relate to the positive and negative complex exponentials associated with the waveform i.e. $\cos(\omega t) = (e^{j\omega t} + e^{-j\omega t})/2$ (see the section entitled "Key points and a brief note on negative frequencies").

```
>> X_top = fft(x_top);
>> plot(fax, abs(X_top)/length(X_top));
>> xlabel('Frequency (radians per sample)')
>> ylabel('Magnitude');
```



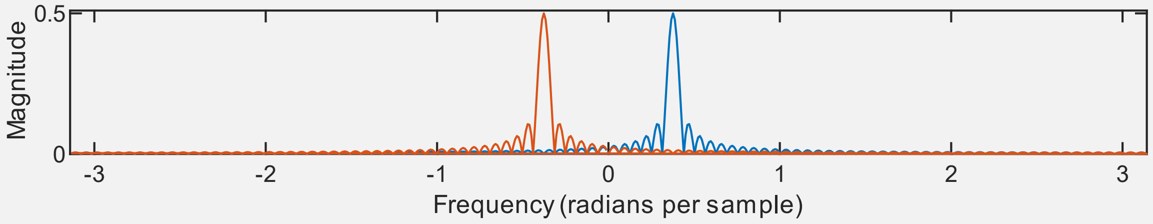
Plotting the double-sided magnitude spectrum of the zero-padded bottom plot shows that the 'sinc' function associated with the rectangular window is 'superimposed' and centred at the locations of the 'spikes' associated with the sinusoidal waveform. An explanation as to why this happens follows later.

```
>> X_zpad = fft(x_zpad);
>> plot(fax, abs(X_zpad)/N);
>> xlabel('Frequency (radians per sample)')
>> ylabel('Magnitude');
```



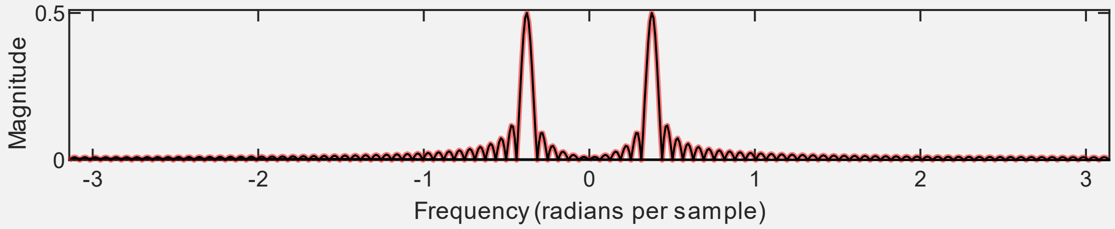
You'll notice that the shape of the both "sinc" functions have been noticeably distorted in the region of 0 Hz i.e. the side-lobes aren't symmetrical around the main-lobes. This effect can be understood by considering each complex exponential 'spike' in isolation. A symmetrical sinc function would be observed if only one complex exponential was present and the distortion observed is due to interference between the two sinc functions which result from the presence both complex exponential 'spikes' in the sinusoidal signal. The code below illustrates this effect by synthesising each of the complex exponentials separately and performing a DFT on each one. Notice how the sinc shape is now symmetrical for both complex exponentials.

```
>> exp_positive = 0.5*exp(j*2*pi*6*n/N);
>> exp_negative = 0.5*exp(-j*2*pi*6*n/N);
>> exp_positive_zpad = exp_positive.*rect_win;
>> exp_negative_zpad = exp_negative.*rect_win;
>> plot(fax, abs(fft(exp_positive_zpad))/N);
>> hold on
>> plot(fax, abs(fft(exp_negative_zpad))/N);
>> xlabel('Frequency (radians per sample)')
>> ylabel('Magnitude');
```



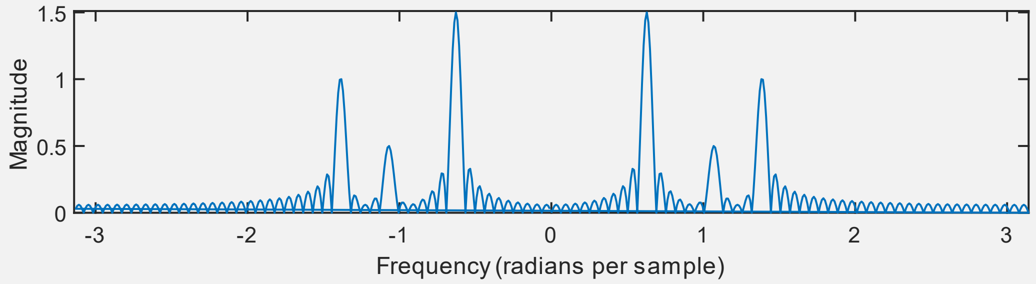
It can be appreciated from the code and plots below that the magnitude of the sum of both complex spectrums (which includes both magnitude and phase information) associated with the two complex exponentials is identical to the magnitude spectrum of the zero-padded sinusoidal signal, whereby interference between sinc functions associated with each individual complex exponential cause the distortions to the sinc shapes mentioned above.

```
>> E_neg = fft(exp_negative_zpad);
>> E_pos = fft(exp_positive_zpad);
>> plot(fax, abs(E_neg+E_pos)/N,'r','LineWidth',2);
>> hold on
>> plot(fax, abs(fft(x_zpad))/N,'k');
>> xlabel('Frequency (radians per sample)')
>> ylabel('Magnitude');
```



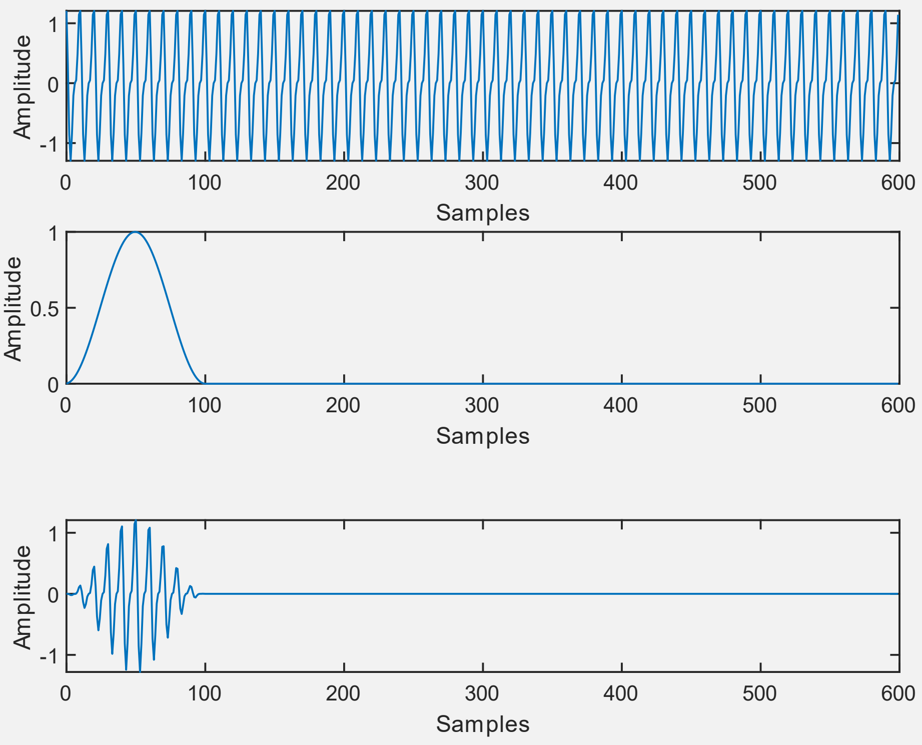
It's worth noting that the sinc shape will appear for any sinusoidal component present in the signal, as illustrated by the plot below, which was generated using the following code. Note that there are three sinusoidal components present in the signal `x_top` and that each sinusoidal component has a different amplitude.

```
>> x_top=3*cos(20*pi*n/N)+cos(34*pi*n/N)+2*cos(44*pi*n/N);
>> x_zpad = x_top.*rect_win;
>> plot(fax, abs(fft(x_zpad))/N);
>> xlabel('Frequency (radians per sample)')
>> ylabel('Magnitude');
```



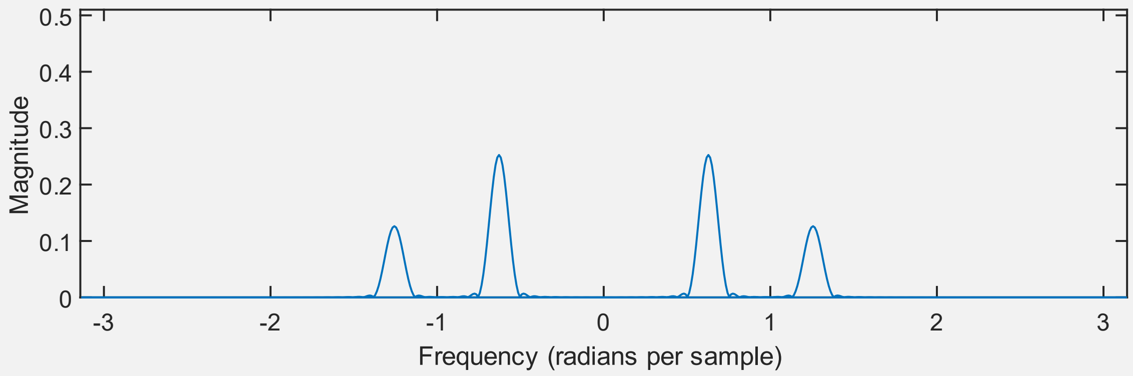
It's also worth noting that the sinc shape is associated with a rectangular window function. We can change the window function to something other than a rectangular window, such as a Hann window function, shown below. Note that the signal `x_top` had been altered to include just two sinusoidal waveforms for this example.

```
>> N = 100;
>> N_zeros = 500;
>> n = 0:N+N_zeros-1;
>> x_top = 0.5*cos(2*pi*20*n/N)+ cos(2*pi*10*n/N+pi/4);
>> hann_win = [hanning(N)' zeros(1,N_zeros)];
>> x_zpad = x_top.*hann_win;
>> subplot(3,1,1); plot(n, x_top);
>> xlabel('Samples');ylabel('Amplitude');
>> subplot(3,1,2); plot(n, hann_win);
>> xlabel('Samples');ylabel('Amplitude');
>> subplot(3,1,3); plot(n, x_zpad);
>> xlabel('Samples');ylabel('Amplitude');
```



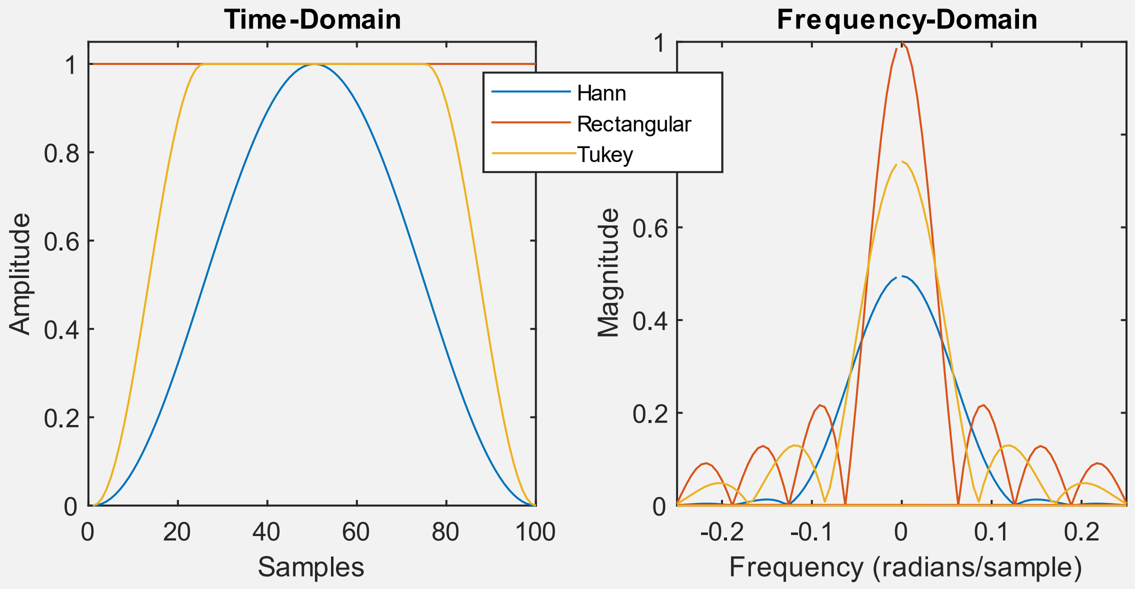
The code above synthesises a signal with two sinusoidal components, of frequency 1.2566 radians/sample and 0.6283 radians/sample) and when we plot the magnitude spectrum, a bell-like shape appears centred at each point along the frequency axis that is associated with these two frequency components i.e. at +0.6283, -0.6283 , +1.2566 and -1.2566 radians per sample, which represent both the positive and negative frequencies of the double-sided spectrum. Take care not to confuse the time-domain shape of the Hann window function with the frequency-domain, as they do appear to be similar at first glance. Closer inspection in the frequency-domain will reveal side-lobe components that are not present in the time-domain.

```
>> X_zpad = fft(x_zpad);
>> fax = [0:length(X_zpad)-1]*2*pi/length(X_zpad);
>> inds = find(fax>pi); fax(inds) = fax(inds)-2*pi;
>> plot(fax, abs(X_zpad)/N);
>> xlabel('Frequency (radians per sample)')
>> ylabel('Magnitude');
```



The plots below show a comparison of three window functions in both the time-domain and frequency-domain. Notice that there is a trade-off between how narrow the main-lobe is against the magnitude of the side-lobes. The main-lobe of the rectangular window is half the width of the hamming window's main-lobe, when measured at the base of the main-lobes. The side-lobe magnitudes of the rectangular window are much larger, however, and these side-lobes can cause significant error's in frequency estimations if frequency components are relatively close to each other.

```
>> N = 100;
>> subplot(1,2,1)
>> plot(hann(N));
>> hold on
>> plot(rectwin(N));
>> plot(tukeywin(N));
>> title('Time-Domain')
>> ylabel('Amplitude'); xlabel('Samples')
>> ylim([0 1.05])
>> legend({'Hann', 'Rectangular','Tukey'})
>> z_pad = zeros(N*10,1);
>> fax = [0:N*11-1]*2*pi/(N*11);
>> inds = find(fax>pi); fax(inds) = fax(inds)-2*pi;
>> subplot(1,2,2);
>> plot(fax, abs(fft([hann(N) ; z_pad]))/N);
>> hold on
>> plot(fax, abs(fft([rectwin(N) ; z_pad]))/N);
>> plot(fax, abs(fft([tukeywin(N) ; z_pad]))/N);
>> title('Frequency-Domain')
>> ylabel('Magnitude');
>> xlabel('Frequency (radians/sample)')
```



The main-lobe width is also dependent upon the length of the signal being analysed and for a rectangular window the main-lobe width, at the base, is given by 4π/N radians per sample, which equates to 2*fs*/N Hz, where N is the length of the window prior to zero-padding and *fs* is the sampling rate. Recall that the DFT bin resolution for a non-zero padded signal is *fs*/N and you should note that the width of window functions are often expressed in terms of this bin width, so you will find that the width of the main-lobe of a rectangular window function is often said to be two bins wide, but this description must be understood in the context of the non-zero-padded signal.

It can also be worth relating this to the Fourier analysis of a pulse waveform (pg 62), whereby the width of the frequency-domain main-lobe is shown to be 2/*P* Hz for a time-domain pulse waveform with a pulse duration of *P* seconds. If the discrete rectangular window function is N samples long, this equates to N/*fs* seconds for a sampling rate of *fs* Hz. Substituting *P* for N/*fs* gives a main-lobe width of 2*fs*/N Hz, which corresponds to our practical observations in plots generated by code.

Why the spectral shape of window functions is frequency shifted

In numerous examples we have seen that after windowing and zero-padding a signal, that the spectral shape associated with the windowing function is effectively shifted to locations on the frequency axis that correspond to the frequency of sinusoidal components present in the signal being analysed. The reason for this can be explained in two ways (at least!). In my video on windowing pzdsp.com/vid31 I use a process known as convolution to explain this phenomenon, but rather than explain convolution I will use a well known trigonometric identity and some coded examples in this text instead. Readers with a background in convolution may prefer the explanation offered in the video.

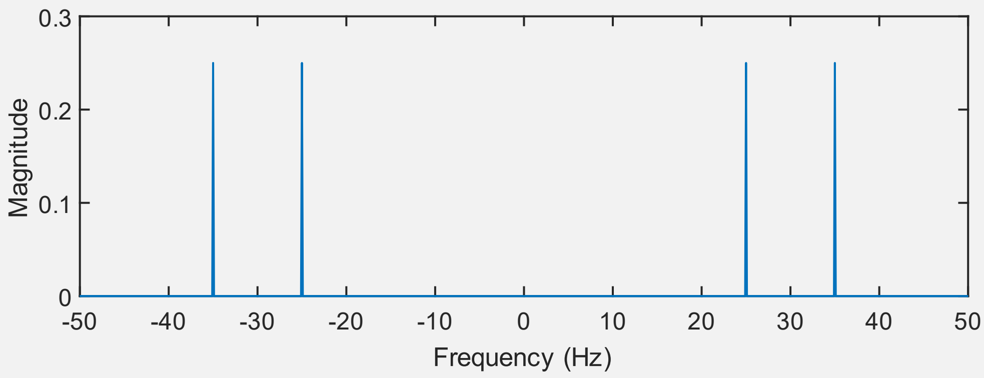
Firstly, it can be shown that

$$2\cos(\mathrm{A})\cos(\mathrm{B}) = \cos(\mathrm{A}+\mathrm{B}) + \cos(\mathrm{A}-\mathrm{B})$$

Therefore, multiplying two sinusoidal waveforms, of different frequency, will produce a waveform that contains two sinusoidal waveforms, as given by the following formula, and demonstrated in the code and plots below in which f_1 and f_2 are 30 Hz and 5 Hz, respectively.

$$\cos(2\pi f_1 t)\cos(2\pi f_2 t) = \frac{\cos(2\pi(f_1+f_2)t) + \cos(2\pi(f_1-f_2)t)}{2}$$

```
>> N = 10000; n = 0:N-1;
>> fs = 1000; t = n/fs;
>> f1 = 30; f2 = 5;
>> x = cos(2*pi*f1*t).*cos(2*pi*f2*t);
>> X = fft(x);
>> fax = [0:length(X)-1]*2*pi/length(X);
>> inds = find(fax>pi); fax(inds) = fax(inds)-2*pi;
>> fax = fax*fs/(2*pi); %convert into Hertz
>> plot(fax, abs(X)/N); xlim([-50 50])
>> xlabel('Frequency (Hz)'); ylabel('Magnitude')
```



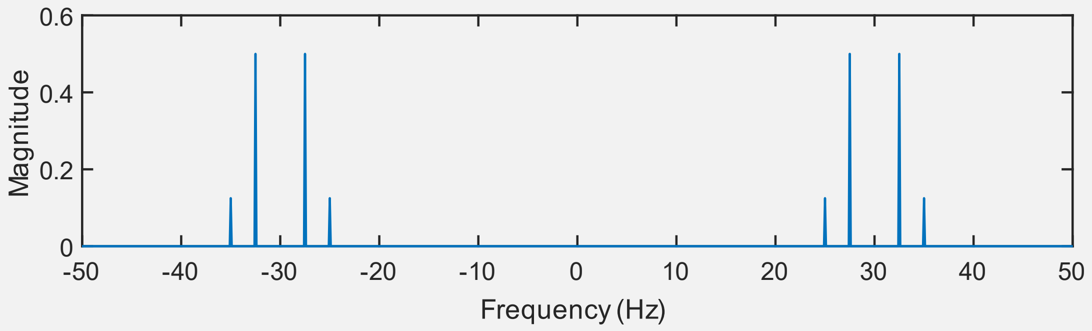
Similarly, it can be shown that

$$\cos(2\pi f_1 t)\,[A_2\cos(2\pi f_2 t) + A_3\cos(2\pi f_3 t)] =$$

$$\frac{A_2\cos(2\pi(f_1+f_2)t) + A_2\cos(2\pi(f_1-f_2)t)}{2} + \frac{A_3\cos(2\pi(f_1+f_3)t) + A_3\cos(2\pi(f_1-f_3)t)}{2}$$

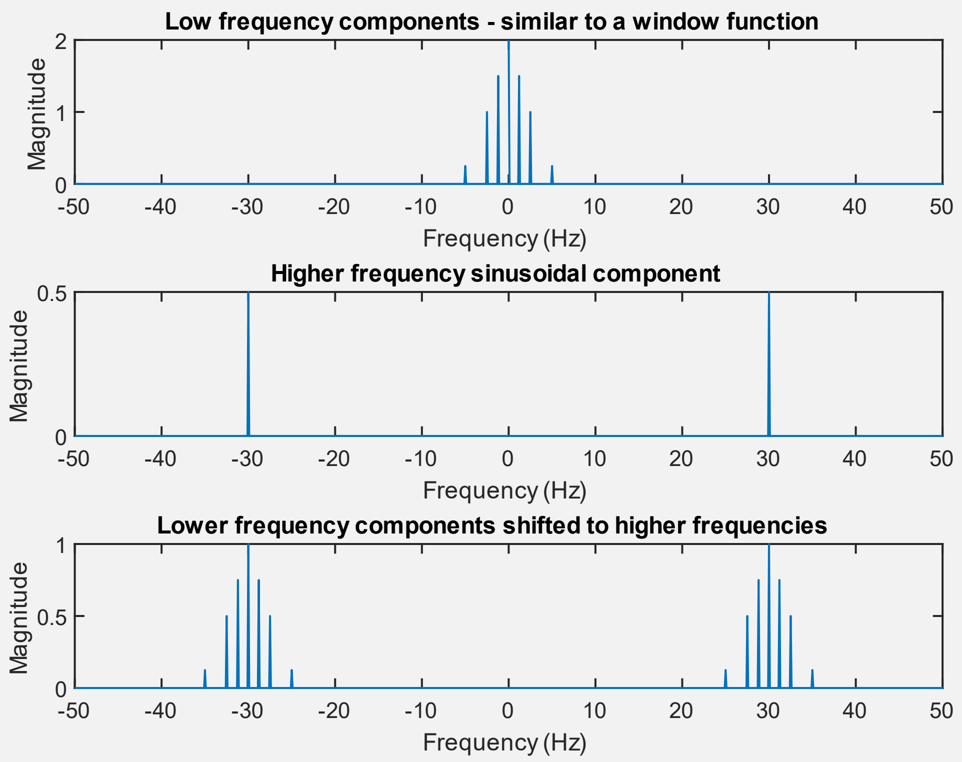
The code below adapts the previous code to include a third sinusoidal waveform. The variable `x_low_freq` is comprised of two relatively low frequency sinusoidal components, which is multiplied by a sinusoidal waveform of higher frequency. The magnitude spectrum shows that both low frequency components are centred around the higher frequency components, as predicted by formula given above.

```
>> N = 10000; n = 0:N-1;
>> fs = 1000; t = n/fs;
>> f1 = 30; f2 = 5; f3 = 2.5;
>> x_low_freq = 0.5*cos(2*pi*f2*t) + 2*cos(2*pi*f3*t);
>> x = cos(2*pi*f1*t).*x_low_freq;
>> X = fft(x);
>> fax = [0:length(X)-1]*2*pi/length(X);
>> inds = find(fax>pi); fax(inds) = fax(inds)-2*pi;
>> fax = fax*fs/(2*pi); %convert into Hertz
>> plot(fax, abs(X)/N); xlim([-50 50])
>> xlabel('Frequency (Hz)'); ylabel('Magnitude')
```



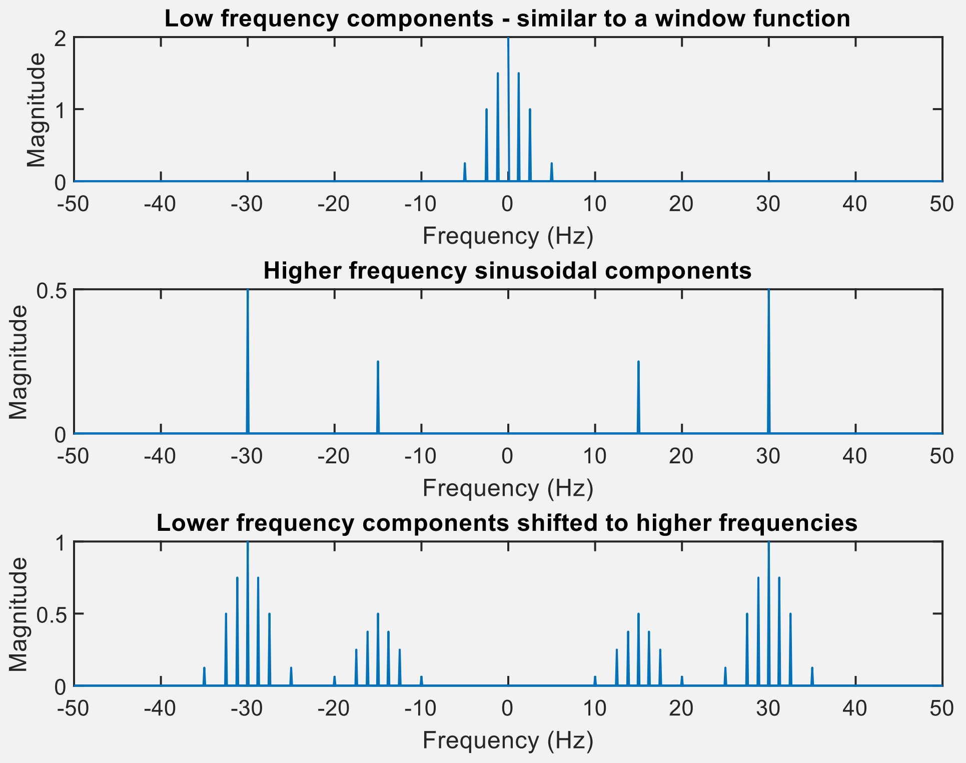
Now let's add in a third low frequency component to the variable `x_low_freq` in addition to a DC component, just to create a salient spectral feature that is somewhat similar to the spectral characteristics of a window function i.e. one with low frequency content. Notice how the spectral shape associated with `x_low_freq` is shifted to frequency locations associated with the higher frequency component `x_high_freq`.

```
>> N = 10000; n = 0:N-1;
>> fs = 1000; t = n/fs;
>> f1 = 30; f2 = 5; f3 = 2.5; f4 = 1.2;
>> x_low_freq = 0.5*cos(2*pi*f2*t) +
2*cos(2*pi*f3*t)+3*cos(2*pi*f4*t)+2;
>> x_high_freq = cos(2*pi*f1*t);
>> x = x_high_freq.*x_low_freq;
>> X = fft(x);
>> fax = [0:length(X)-1]*2*pi/length(X);
>> inds = find(fax>pi); fax(inds) = fax(inds)-2*pi;
>> fax = fax*fs/(2*pi); %convert into Hertz
>> subplot(3,1,1)
>> plot(fax, abs(fft(x_low_freq))/N); xlim([-50 50])
>> xlabel('Frequency (Hz)'); ylabel('Magnitude')
>> subplot(3,1,2)
>> plot(fax, abs(fft(x_high_freq))/N); xlim([-50 50])
>> xlabel('Frequency (Hz)'); ylabel('Magnitude')
>> subplot(3,1,3)
>> plot(fax, abs(X)/N); xlim([-50 50])
>> xlabel('Frequency (Hz)'); ylabel('Magnitude')
```



One more similar example in which `x_high_freq` has two components should help close the loop on this topic. Observe again how the spectral shape associated with the low frequency content is shifted to the frequencies associated with the higher frequency components. Also note that the shape is scaled in proportion to the amplitude of the higher frequency components.

```
>> N = 10000; n = 0:N-1;
>> fs = 1000; t = n/fs;
>> f1 = 30; f2 = 5; f3 = 2.5; f4 = 1.2; f5 = 15;
>> x_low_freq = 0.5*cos(2*pi*f2*t) +
2*cos(2*pi*f3*t)+3*cos(2*pi*f4*t)+2;
>> x_high_freq = cos(2*pi*f1*t)+0.5*cos(2*pi*f5*t);
>> x = x_high_freq.*x_low_freq;
>> X = fft(x);
>> fax = [0:length(X)-1]*2*pi/length(X);
>> inds = find(fax>pi); fax(inds) = fax(inds)-2*pi;
>> fax = fax*fs/(2*pi); %convert into Hertz
>> subplot(3,1,1)
>> plot(fax, abs(fft(x_low_freq))/N); xlim([-50 50])
>> xlabel('Frequency (Hz)'); ylabel('Magnitude')
>> subplot(3,1,2)
>> plot(fax, abs(fft(x_high_freq))/N); xlim([-50 50])
>> xlabel('Frequency (Hz)'); ylabel('Magnitude')
>> subplot(3,1,3)
>> plot(fax, abs(X)/N); xlim([-50 50])
>> xlabel('Frequency (Hz)'); ylabel('Magnitude')
```


Low frequency components - similar to a window function
Higher frequency sinusoidal components
Lower frequency components shifted to higher frequencies
Magnitude
Frequency (Hz)