Tutorial 3

Topic One - Filtering Windows

This section will hopefully give a little bit more of an intuition about filtering windows and what they are used for. The most important thing you take away from this is a general understanding of how windowing functions work, and why they are used. The section on filtering windows is on page 302 in Lathi.

There are two main things that filtering windows can do. These are in the areas of “Spectral Analysis” and “Filter Design”. I’ll be quickly covering both of these in this tutorial.

Spectral Analysis

So without further ado, lets start with a signal.

A Basic Signal

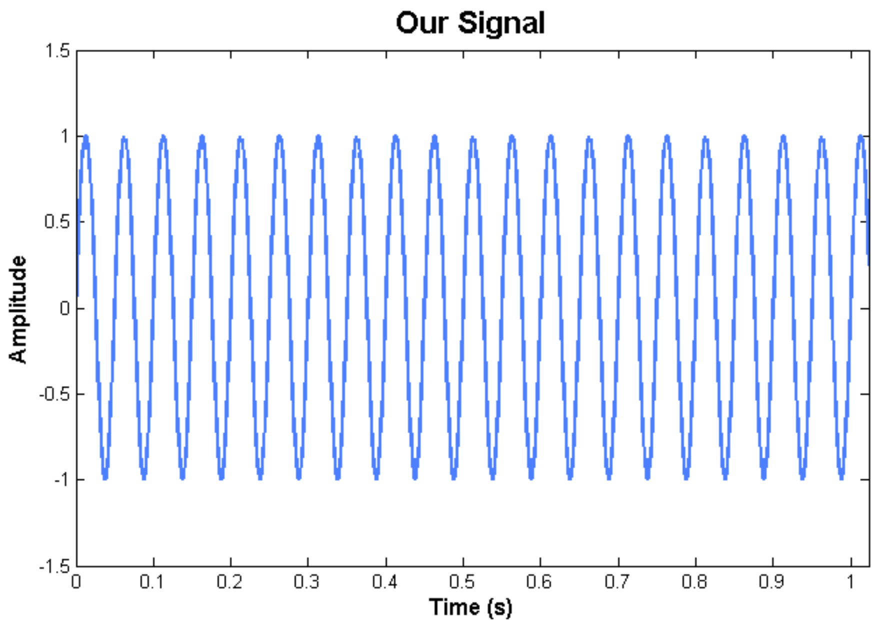

Figure 1: A classic sinusoidal signal.

This is a pure sinusoid at 20Hz, nothing special about it at all. Lets take the FFT of this to see what it looks like:

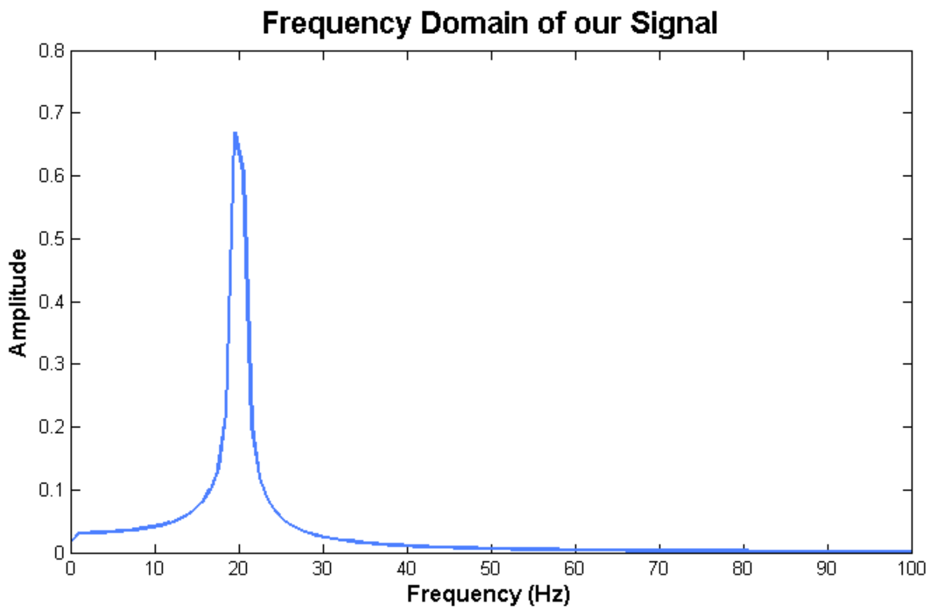

Figure 2: The FFT of our classic 20Hz sinusoid.

As we expected, there is one crisp peak[1] at the 20Hz mark, and everything else is pretty low. It's not zero, but it's pretty low. To show this, lets have a look at a semi-log graph of the same thing:

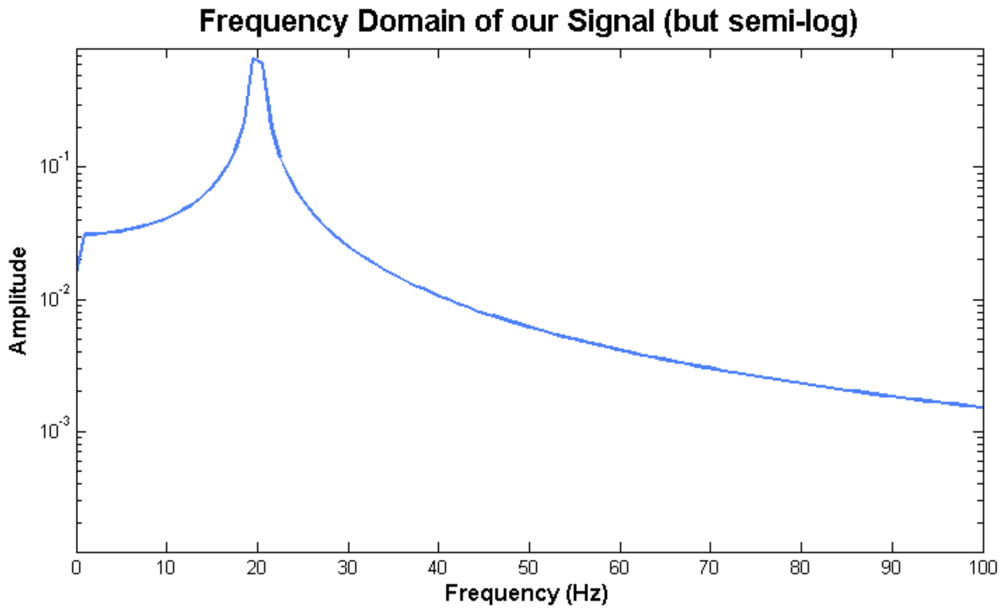

Figure 3: The FFT of our classic 20Hz sinusoid, this time shown with a semi-log graph.

[1]The reason it's not 'super' crisp is because the actual frequency of the sin wave is somewhere between the two closest points, so both of them share the highest magnitude. To get rid of this, I could increase the amount of samples in the signal, and thus the quality of the FFT, or I could pick the numbers perfectly to make the peak of the FFT exactly on a specific frequency probed by the FFT algorithm.

Now imagine we want to only sample a small amount of this total signal. So we truncate a section of it. This is what our new signal will look like:

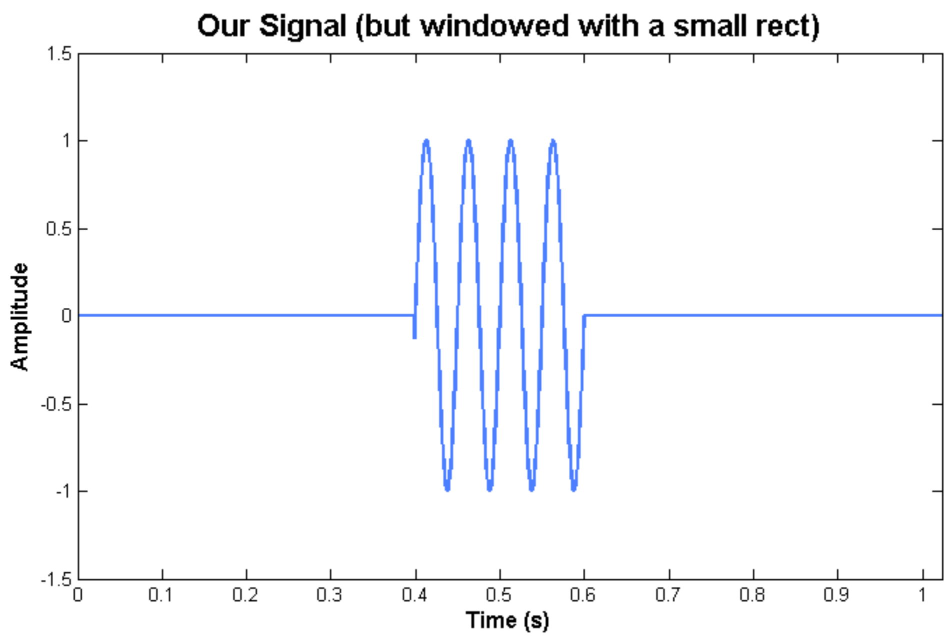

Figure 4: Our classic 20Hz sinusoid, but with a rectangle window from 0.4s to 0.6s.

Practically, this is done frequently because often we want to find the FFT of a small part of a larger signal. Music is a good example here, where the whole song is the signal and we want to find the frequency of only a section of it. This is those animations are created which move in time with the music, frequently seen on YouTube. This is also how Shazam works[2] - just taking repeated FFT's of sections of song, and checking it against a database. MP3 encoding also uses FFT's with window functions.[3]

[2]In case you haven't heard of it, Shazam is that app which you can record ten seconds of a song, and it tells you exactly what song is playing.

[3]MP3's actually run FFT's on chunks of song around 256 to 1024 samples long and store the coefficients of the FFT rather than sampled values for compression. Coincidentally the windowed samples I'm taking are 200 samples long, so the sizes here are realistic too!

Figure 5: The animation of the coloured section around the circle moves with the music, and different parts move depending on the frequency of the music. This was created by taking repeated FFT's of windowed sections of the songs spectrum.

Anyhow, now lets take the FFT of this windowed sinusoid and see what the damage is.

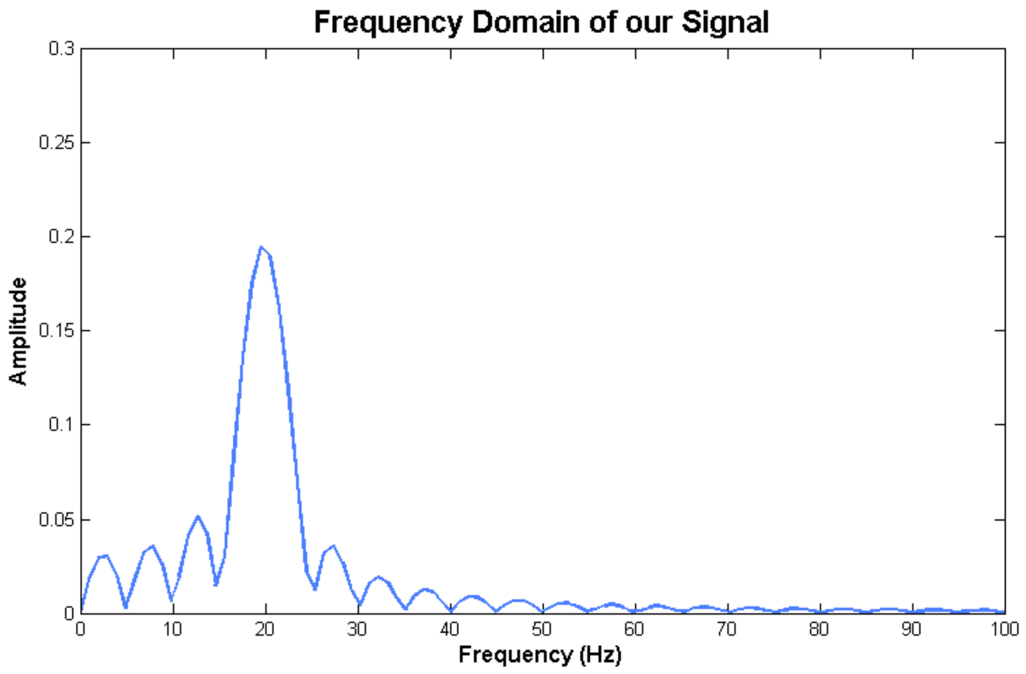

Figure 6: The FFT of the windowed sinusoid.

As expected, this windowing has added 'spectral leakage' to the original frequency response. Polluting the original response with side lobes. Lets have a look at this with the semi-log graph to compare it to the last FFT.

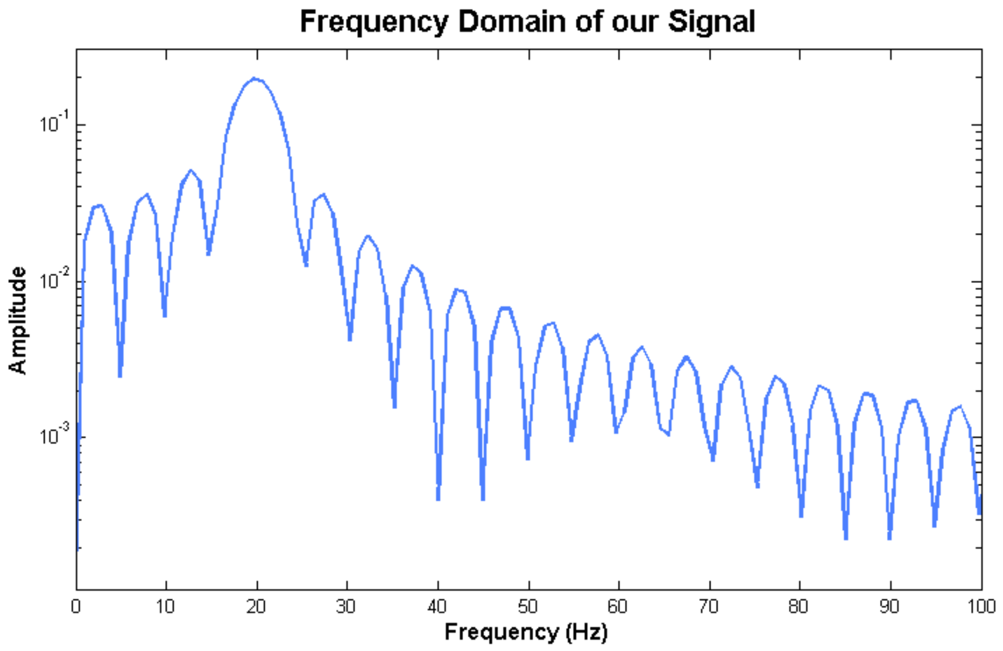

Figure 7: The FFT of the windowed sinusoid, this time on a semi-log plot.

So we've established that simply cutting the signal at a specific point is going to add a bunch of spectral leakage, so can we do better? Well yes we can! This is what windowing is for! So let's get that signal we used the rect window on before, and now use a Hamming window.

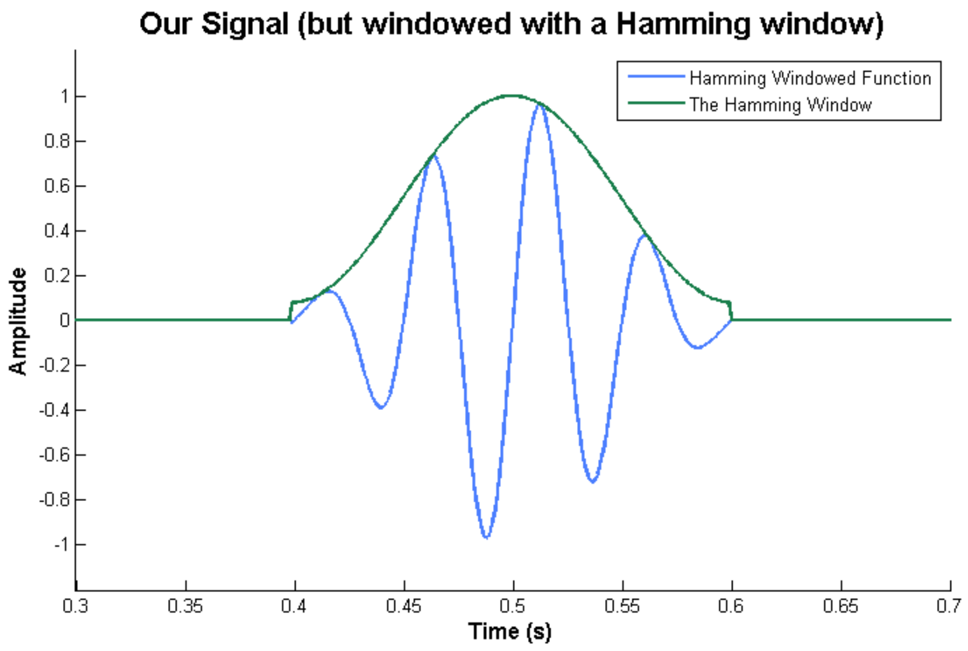

Figure 8: Now we've applied a Hamming filter to the windowed portion of our original signal.

Lets take this into the frequency domain and see if we can tell the difference.

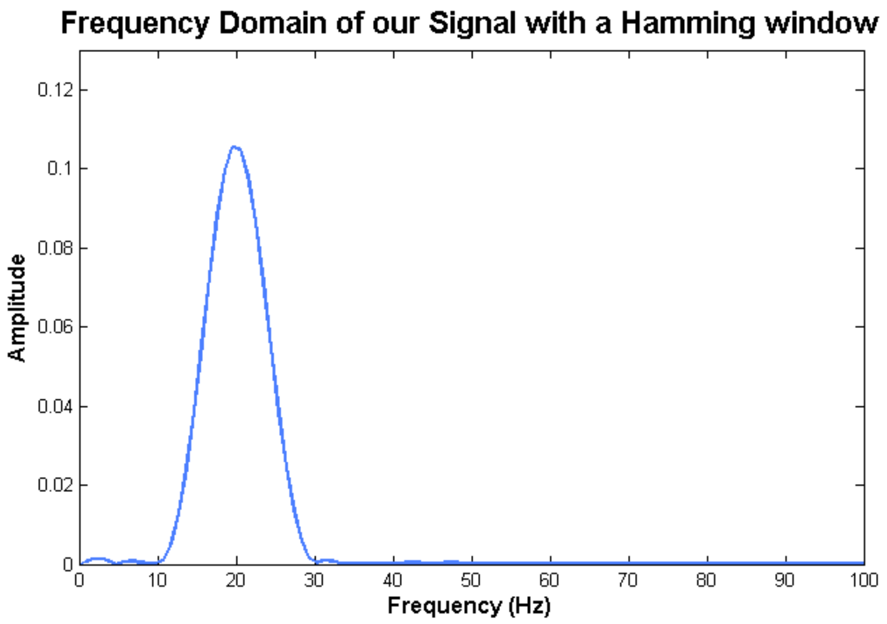

Figure 9: The frequency domain of the Hamming filtered signal.

And oh boy isn't that a huge improvement when compared to Figure 6. This is shown pretty well again with the semi-log graph of the same thing.

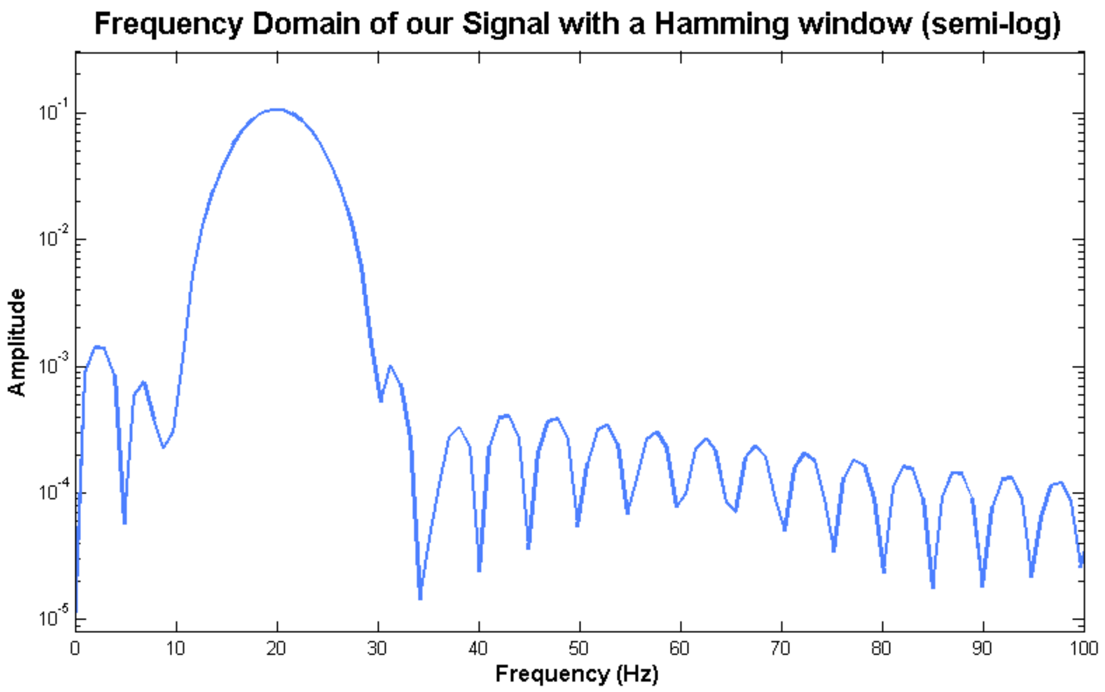

Figure 10: The frequency domain of the Hamming filtered signal (This time, with semi-log).

So you'll notice that the windowing function has definitely changed the signal. It has unfortunately widened the main lobe, but has resulted in a much steeper roll off after the main lobe. This may be a characteristic you're looking for when you're analysing your signal.

A more Complicated Signal

So this time, let's do the same thing, but with two superimposed sinusoids. Here we have a strong sine wave at 20Hz, and a weaker one at 29 Hz.

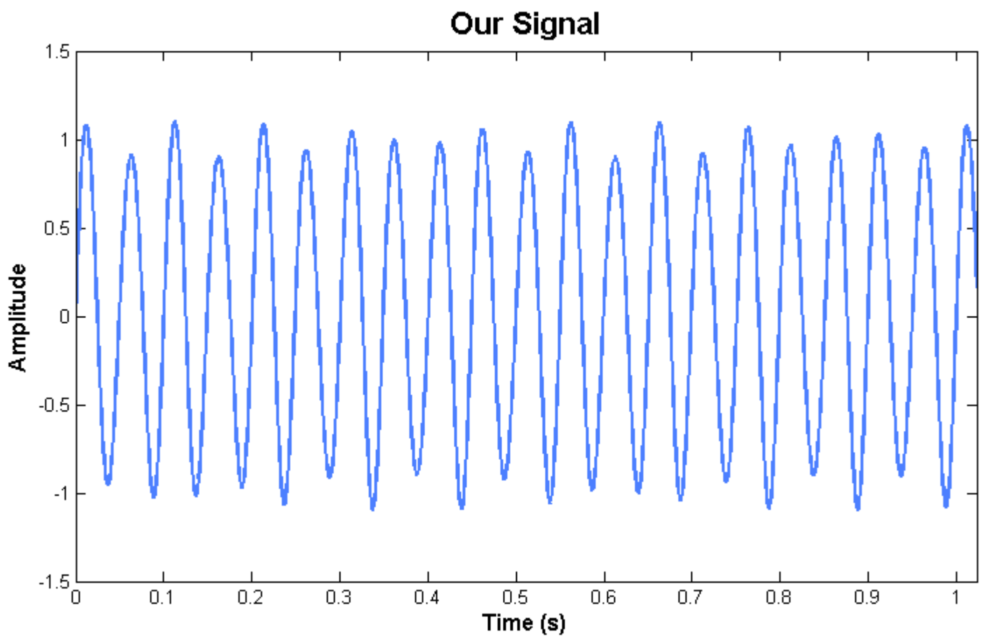

Figure 11: Our new and more complicated signal.

Lets look at the fourier transform in normal scale and semilog scale here again for reference.

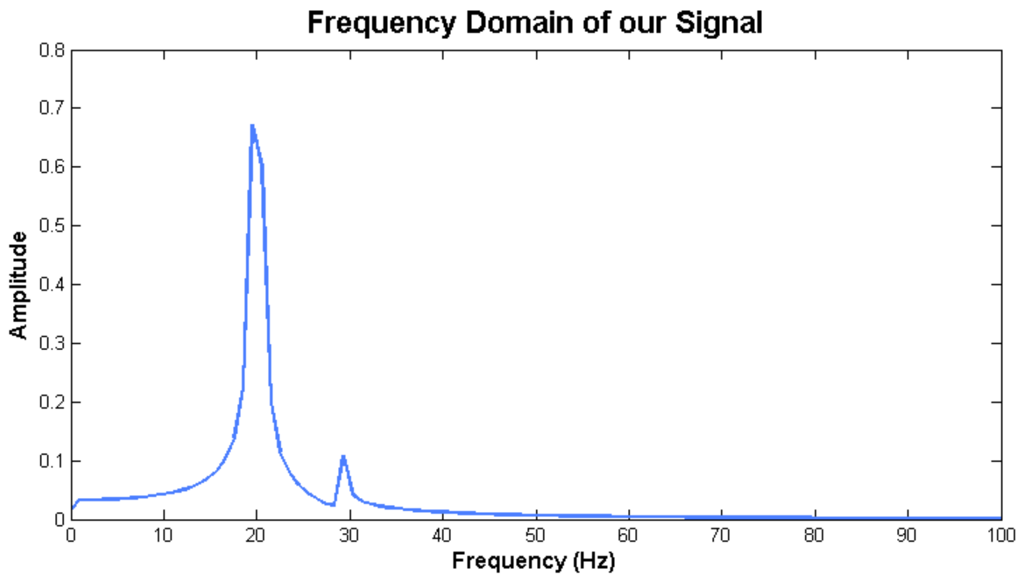

Figure 12: The frequency domain of our new signal.

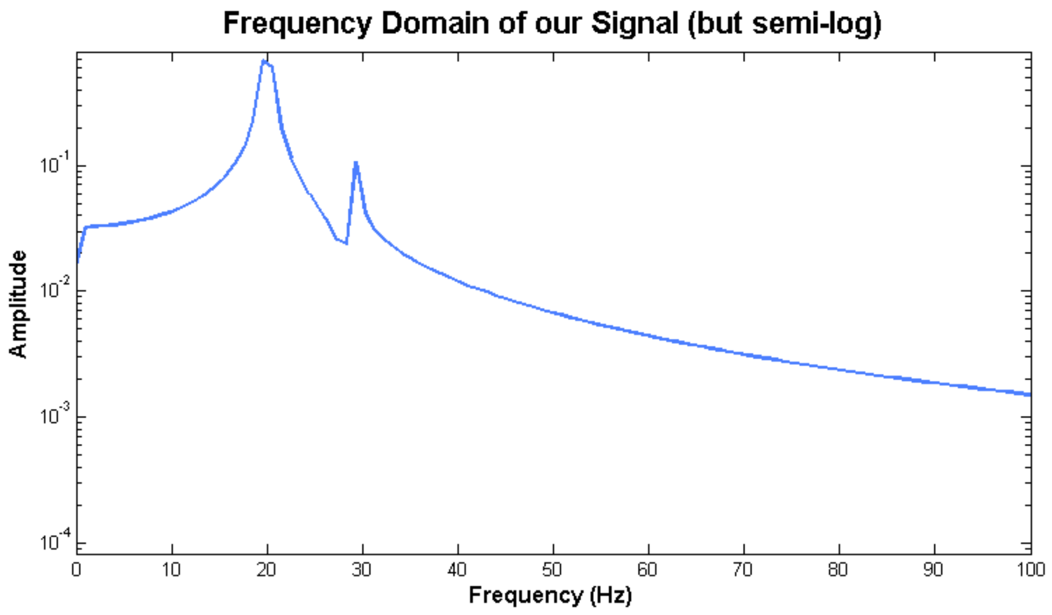

Figure 13: The frequency domain of our new signal (but in semi-log).

So here, we can tell there are obviously two sinusoids at different frequencies, and that's it. It's a very clean graph. So now lets take a window just as we did before.

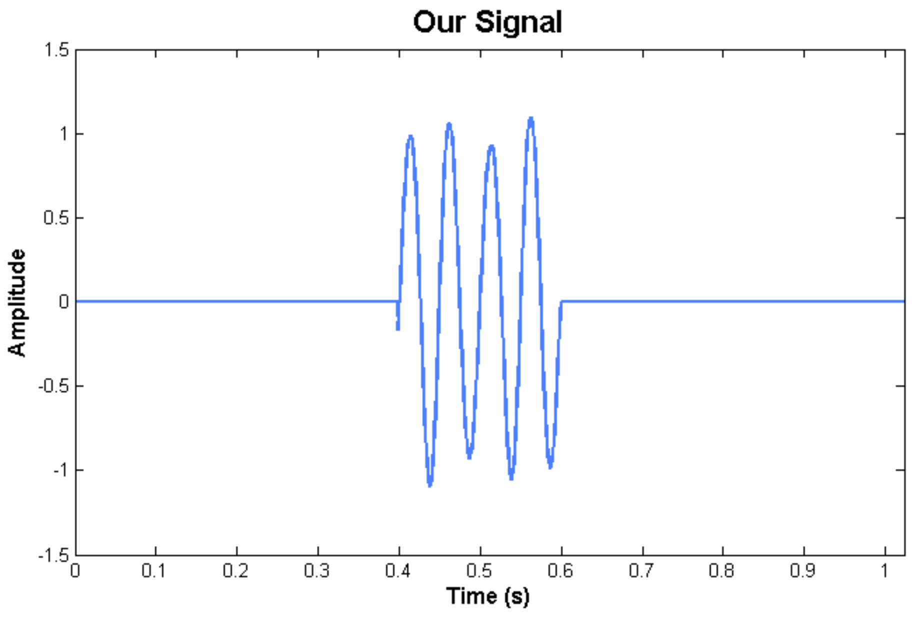

Figure 14: Our windowed signal.

So this here is the small window we are taking from our original sample. As you'd imagine this will introduce lots of spectral leakage into the frequency domain, lets have a look.

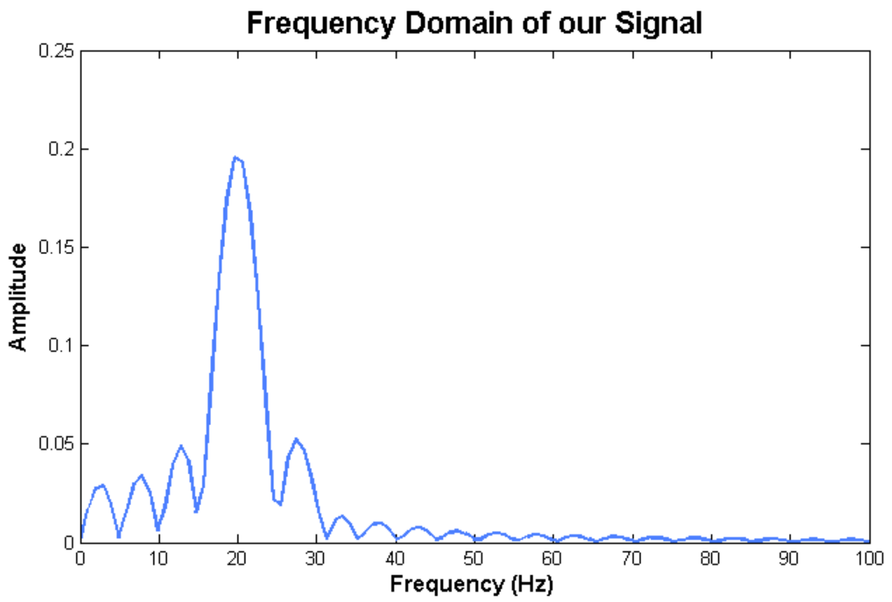

Figure 15: The frequency domain of our windowed signal.

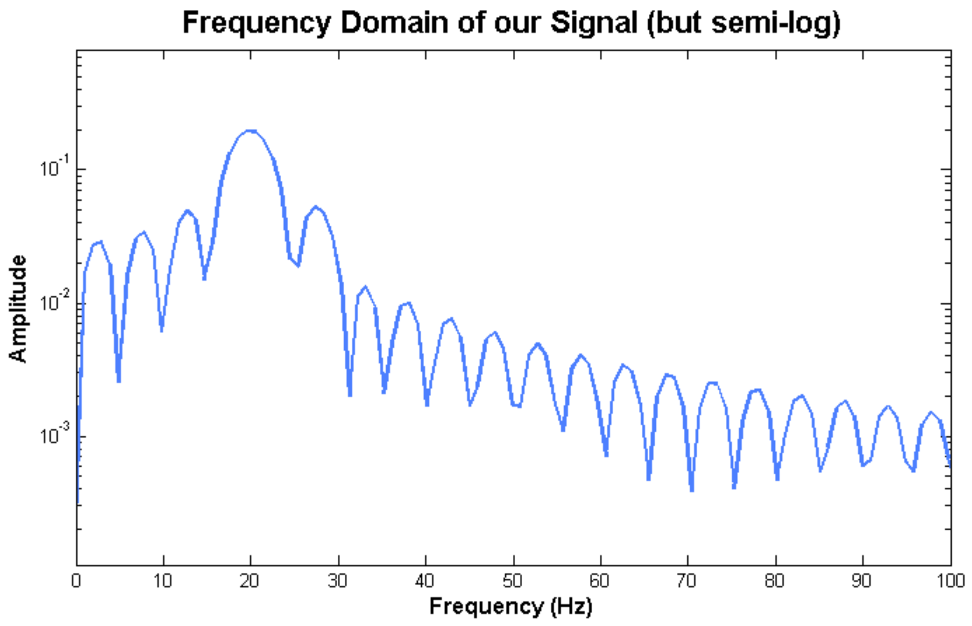

Figure 16: The frequency domain of our windowed signal (but in semi-log).

As you can see, the second peak is almost unrecognisable, and without the prior knowledge that there is a peak at 29Hz, you'd probably look at these graphs and just simply see the peak at 20Hz. This is because the second peak is hidden under one of the side lobes of the main peak! Is there anything we can do in this scenario? Why yes, of course! It might seem a little counter intuitive that slightly distorting the windowed signal will result in a cleaner view in the frequency domain, but that's exactly what windowing does! Lets slap another Hamming filtering window over the original signal:

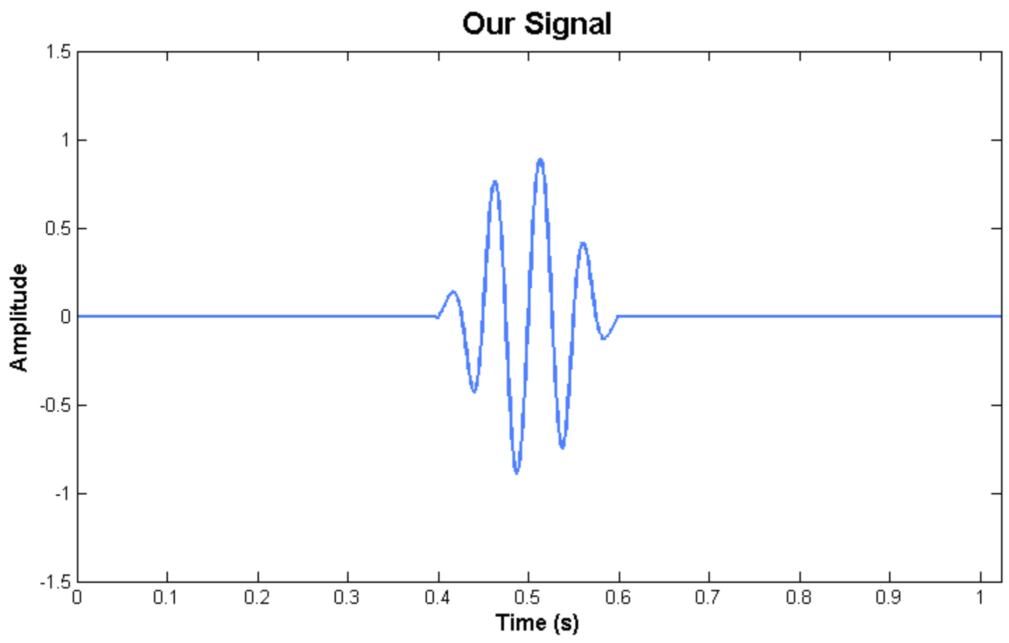

Figure 17: Our Hamming windowed signal (but in semi-log).

And lets have a look at this in the frequency domain and see if we can see two peaks again:

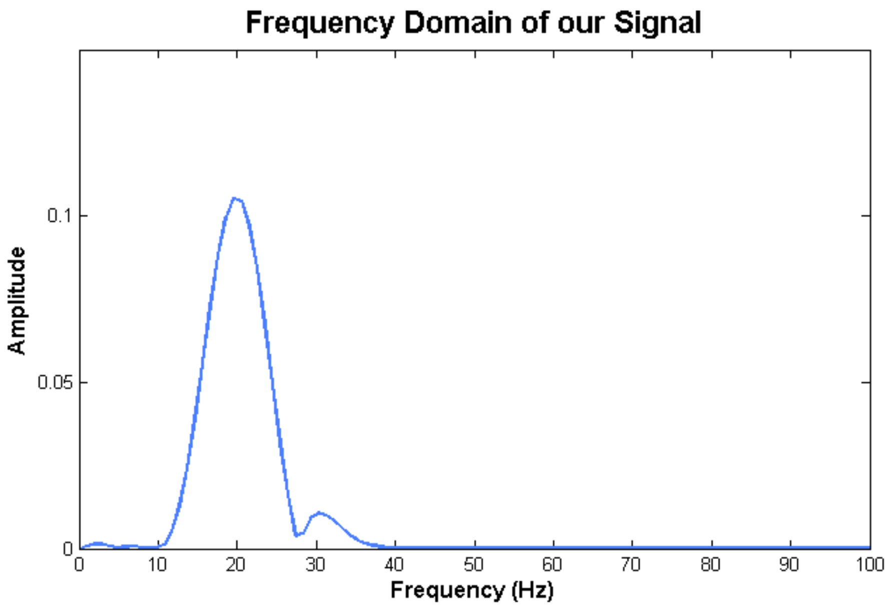

Figure 18: The frequency domain of our Hamming windowed signal.

Wow, we can definitely see the second peak nice and clearly! This is the benefit of windowing!

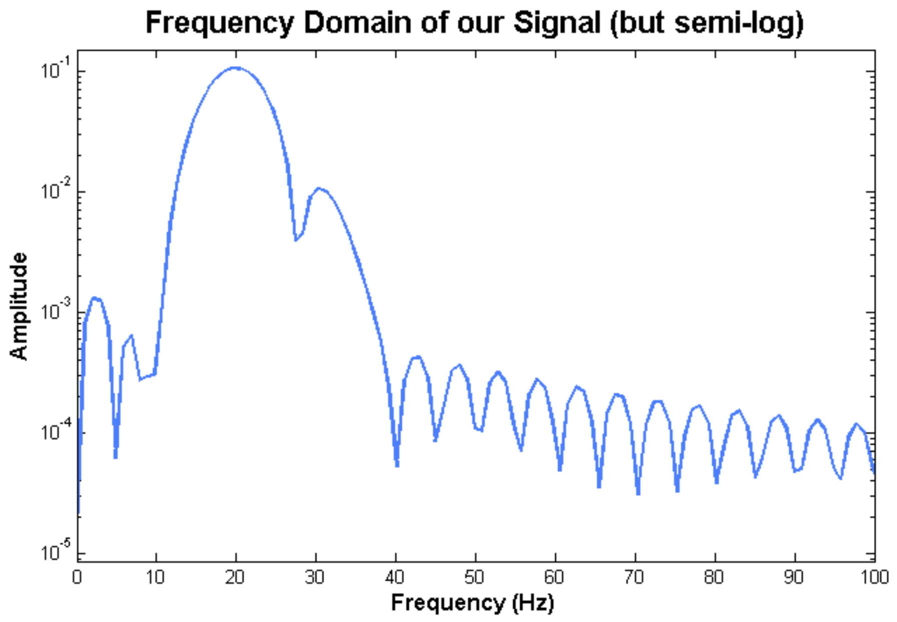

Figure 19: The frequency domain of our Hamming windowed signal (but in semi-log).

Again, you can see the peak much more clearly now.

If you want to recreate these images yourself, I've pasted the Matlab code onto the bottom of this document.

Filter Design

The other main use for windowing functions is in filter design, and designing FIR filters.

So imagine that you want to create a 100% perfect low pass filter. Well lets have a look at this:

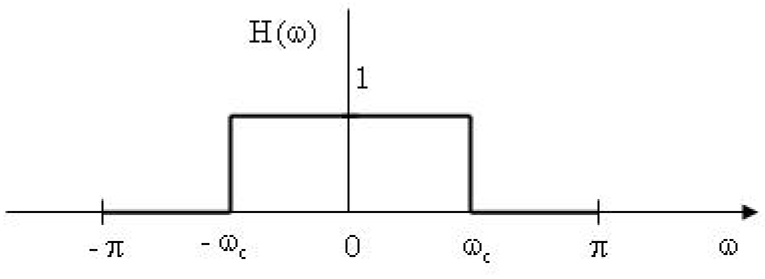

Figure 20: The perfect low-pass filter.

As you can see, it is a perfect rectangle function in the frequency domain. Now, if we remember that a multiplication in the frequency domain is equivalent to a convolution in the time domain, carrying out perfect filtering is as simple as convolving our signal with the FFT of this rect function. However, as I'm sure you know by this point, a Fourier transform of a rect function is the sinc function which looks like this:

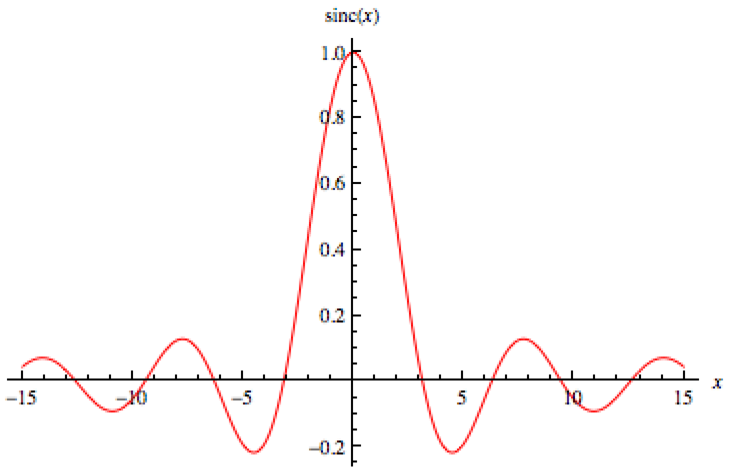

Figure 21: The sinc function.

The problem with the sinc function is that it is continuous and infinity long. This means that if you want to convolve it with your signal, you need an infinity long signal, to sample an infinite amount of points on your signal and sinc function, and the convolve all infinite points with the infinite sinc function. So it turns out that this method is actually impossible in practice.[4] One way around this is to simply truncate the sinc function at a certain distance from the centre, and to use that. I mean, it does go to zero, so that should be fine right? Let's use Matlab's function **wvtool** here which stands for: "Window Visualisation Tool". We'll make a centred sinc function, and visualize it.

```
x = -5:0.1:5;
```

[4]It can be done with continuous functions if you know the equations for them, but then you're solving a math proof by hand, not potentially using this as a real time filtering method.

```
f = sinc(2*x);
wvtool(f)
```

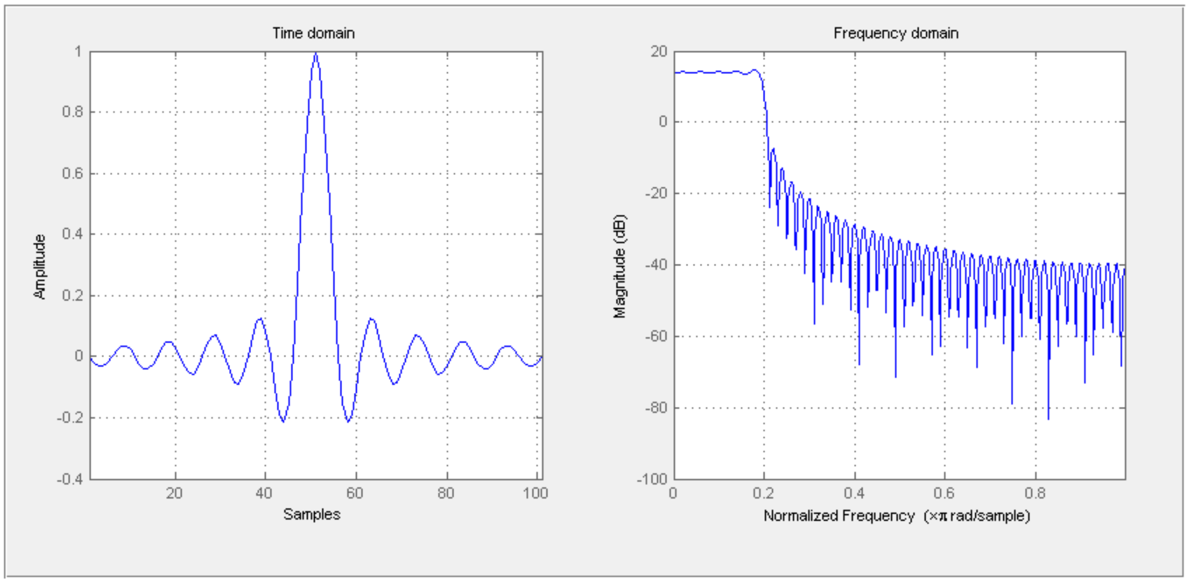

Figure 22: The frequency response by using our truncated sinc function as our filter.

Keep in mind here that the quality of the filter is impacted by the number of samples. Using the following code, we get:

```
x = -2:0.1:2;
f = sinc(2*x);
wvtool(f)
```

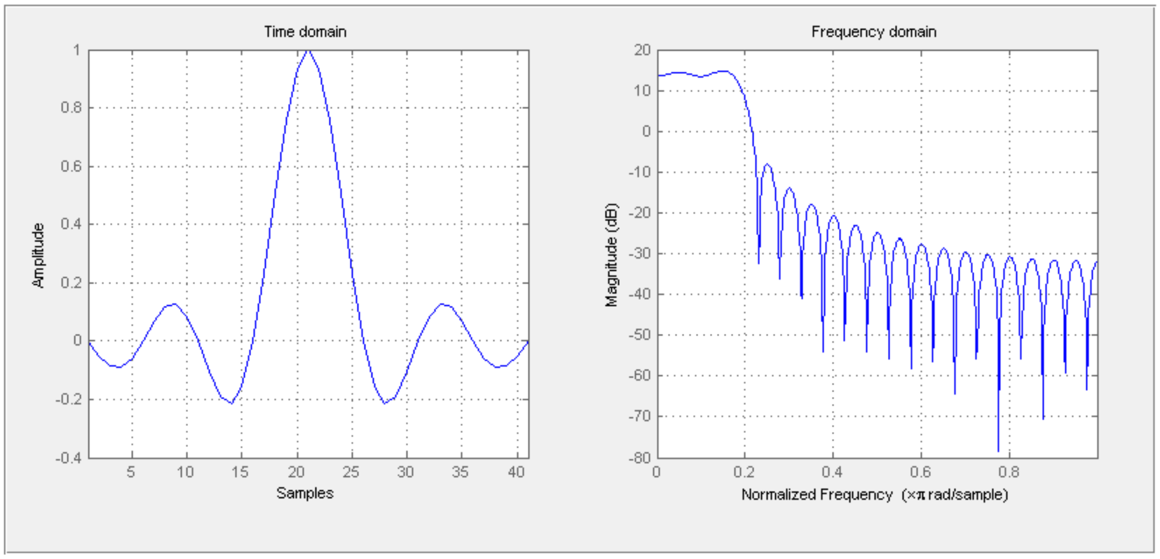

Figure 23: The frequency response by using a truncated sinc with 41 samples instead of 101.

That's because using the 101 sample sinc function creates a 100th order filter, and the 41 sample sinc function creates a 40th order filter. The larger the order, the better the filter. This is expected. However, the larger order comes at the price of more processing.

The cutoff frequency of the filter can be changed by changing the frequency of the sinc function:

```
x = -5:0.1:5;
a = sinc(1*x);
b = sinc(5*x);
wvtool(a, b)
```

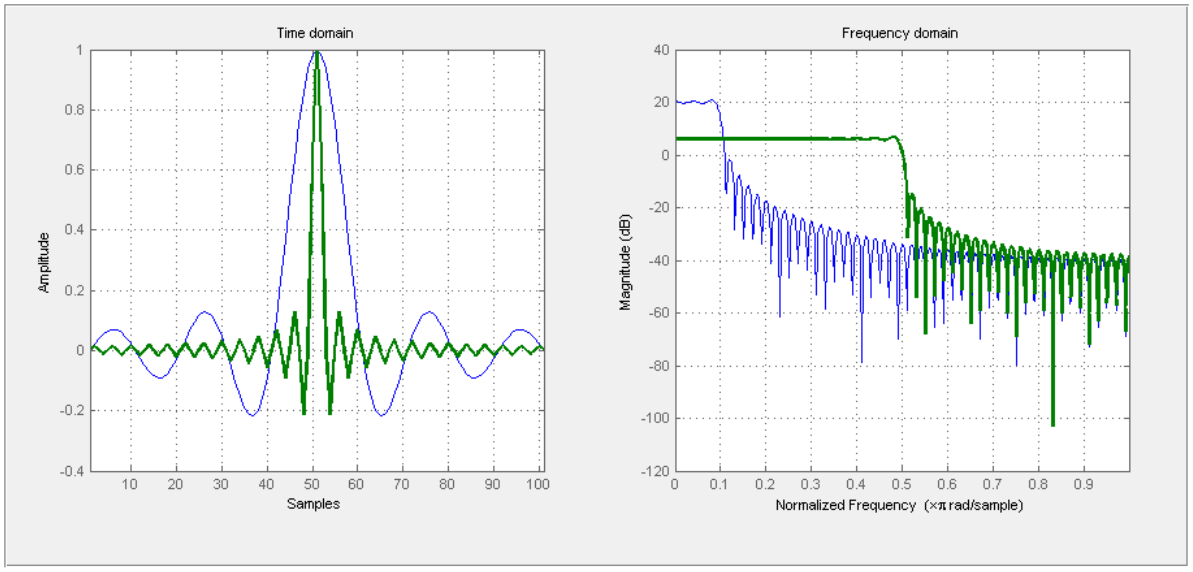

Figure 24: Two different frequency 101 sample sinc function create two filters with cut-off frequencies of 0.1 and 0.5 pi rads/sample respectively.

Anyhow, so this is one way of filtering. The reason a bunch of spectral leakage gets introduced is because we are cutting off part of the sinc function. Given this new constraint, that we must truncate our sinc, can we optimise what we have to remove some of this spectral leakage? Well it turns out that yes we can! This is how windowing functions are used in filtering! Look carefully through the following code to see what I'm doing.

```
x = -5:0.1:5;
a = sinc(1*x);
b = a;
b = b .* transpose(hamming(101));
wvtool(a, b)
```

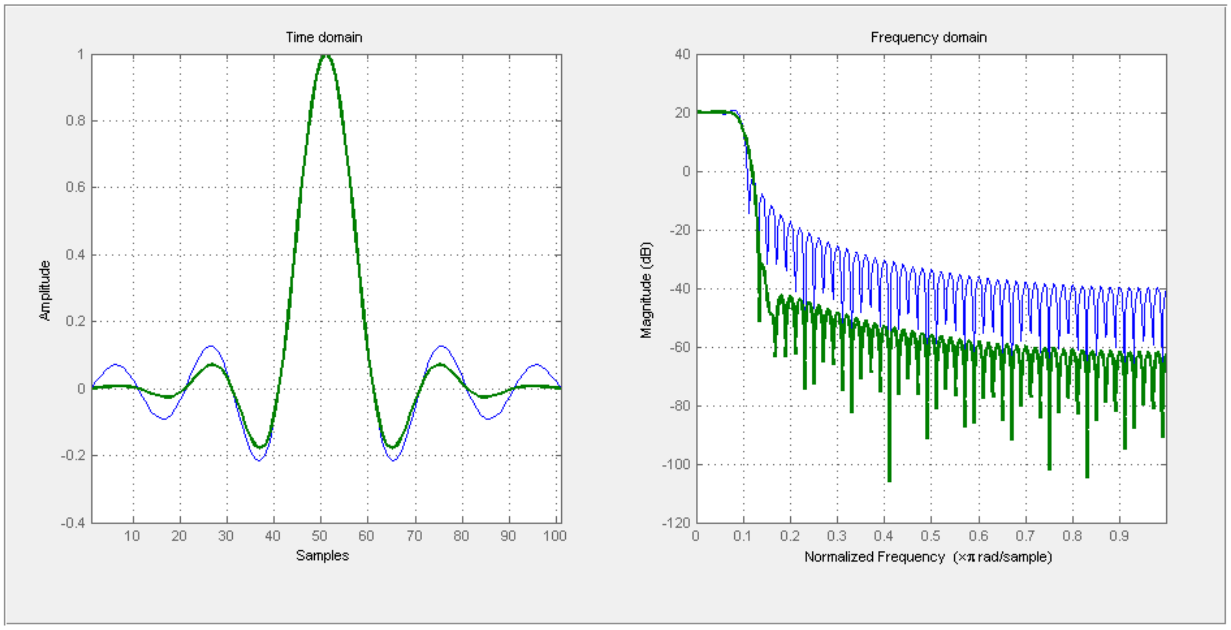

Figure 25: The blue line is the original truncated sinc, and the green is the windowed sinc.

So here we have one sinc function left as is, and another multiplied by the hamming window function. Look at the huge difference this makes in the performance of the filter! This means that if you're trying to filter a signal using a FIR filter, you should always use a windowing function, because a correctly chosen windowing function can drastically increase the performance of the filter without adding any more orders or delays

Problem set 2

Triangle Rule

The triangle rule is another method of approximating a continuous laplace transform with a discrete z-transform

Inertial Measurement Units and control- Gyroscope/Accelerometer noise

This is all about question 5. In question 5, the noise is unspecified. However, generally (IE stating this assumption will be quite valid), the noise of an accelerometer/gyroscope can be modelled by an additive Gaussian noise source. So the output from the Accelerometer is

$$X(t) = A(t) + \mathcal{N}(\sigma, t)$$

. The key to understanding what we can do about the Gaussian noise is to understand its spectrum.[5]

The power spectral density of white Gaussian noise is constant[6]. IE, magnitude of each bin in the Fourier transform, when averaged, will be equal to a constant value. While we don't know the dynamics of the Daedalus drone, It is a fair assumption that the power spectral density of its motion will be band limited, with a maximum frequency in the single to early double digits.

What is a good strategy for a narrowband signal corrupted by a broadband signal?

Is the averaging filter a good approach?

What is the bandwidth of the averaging filter?

Assuming we build a filter, how would we go about finding the optimal bandwidth?

How does phase delay manifest here? is this going to be of significance to the system?

A forward backward filter design has zero phase delay, is it a good choice?

Given all of the above, discuss between the class the best approach to beating the noise

[5]The Gyroscope is the rotational equivalent, and the analysis is basically identical
[6]`https://en.wikipedia.org/wiki/White_noise`

Appendix

Code for Spectral Analysis

Original Sinusoid

```
%Creating a basic sinusoidal function:
x = 0:0.001:1.023; %Create a vector of length 1024. (Power of 2)
f = sin(20*2*pi*x); %Create a sinusoid on this function.
plot(x, f,'linewidth',2,'color',[0.3,0.5,1]) %Plot it.

%Fancy plotting settings
axis([0,1.023,-1.5,1.5]);
xlabel('Time (s)','FontSize',12,'FontWeight','bold');
ylabel('Amplitude','FontSize',12,'FontWeight','bold');
title('Our Signal','FontSize',16,'FontWeight','bold');
%% Doing an FFT on our function and presenting the results
L = 1023; %The length of our vector.
Fs = 1000; %Set the frequency of the sampling. This is 1000Hz.
y = fft(f); %Take the fft of our function.
p2 = abs(y/L); %Take the absolute value of the Re+Im values returned.
p1 = p2(1:L/2+1); %Make the first half of p1 equal to p2.
p1(2:end-1) = 2*p1(2:end-1); %Fold the second half of p2 back onto the first half of p1.
x_axis = Fs*(0:(L/2))/L; %Scaling our new 'x' axis so we can plot in Hz.
plot(x_axis, p1, 'linewidth',2,'color',[0.3,0.5,1]) %Our FFT.

%Fancy plotting settings
axis([0,100,0,0.8])
xlabel('Frequency (Hz)','FontSize',12,'FontWeight','bold');
ylabel('Amplitude','FontSize',12,'FontWeight','bold');
title('Frequency Domain of our Signal','FontSize',16,'FontWeight','bold');
%% Presenting the same results on a semi-log graph
semilogy(x_axis, p1, 'linewidth',2,'color',[0.3,0.5,1])

%Fancy plotting settings
axis([0,100,0,0.8])
xlabel('Frequency (Hz)','FontSize',12,'FontWeight','bold');
ylabel('Amplitude','FontSize',12,'FontWeight','bold');
title('Frequency Domain of our Signal (but semi-log)','FontSize',16,'FontWeight','bold');
```

Original Sinusoid with Rect Window

```
%% Windowing our signal with a Rectangular window and plotting it
x = 0:0.001:1.023; %Create a vector of length 1024. (Power of 2)
f = sin(20*2*pi*x); %Create a sinusoid on this function.
for element = 1:numel(f)
if element < 400
f(element) = 0;
end
if element > 600
f(element) = 0;
end
end
```

```
plot(x, f,'linewidth',2,'color',[0.3,0.5,1]) %Plot it.

%Fancy plotting settings
axis([0,1.023,-1.5,1.5]);
xlabel('Time (s)','FontSize',12,'FontWeight','bold');
ylabel('Amplitude','FontSize',12,'FontWeight','bold');
title('Our Signal (but windowed with a small rect)','FontSize',16,'FontWeight','bold');
%% Taking the FFT of the rect window and plotting it
L = 1023; %The length of our vector.
Fs = 1000; %Set the frequency of the sampling. This is 1000Hz.
y = fft(f); %Take the fft of our function.
p2 = abs(y/L); %Take the absolute value of the Re+Im values returned.
p1 = p2(1:L/2+1); %Make the first half of p1 equal to p2.
p1(2:end-1) = 2*p1(2:end-1); %Fold the second half of p2 back onto the first half of p1.
x_axis = Fs*(0:(L/2))/L; %Scaling our new 'x' axis so we can plot in ms.
plot(x_axis, p1, 'linewidth',2,'color',[0.3,0.5,1]) %Our FFT.

%Fancy plotting settings
axis([0,100,0,0.3])
xlabel('Frequency (Hz)','FontSize',12,'FontWeight','bold');
ylabel('Amplitude','FontSize',12,'FontWeight','bold');
title('Frequency Domain of our Signal','FontSize',16,'FontWeight','bold');
%% Plotting it again but on semilog
semilogy(x_axis, p1, 'linewidth',2,'color',[0.3,0.5,1])

%Fancy plotting settings
axis([0,100,0,0.3])
xlabel('Frequency (Hz)','FontSize',12,'FontWeight','bold');
ylabel('Amplitude','FontSize',12,'FontWeight','bold');
title('Frequency Domain of our Signal','FontSize',16,'FontWeight','bold');
```

Original Sinusoid with Hamming Window

```
%% This time we are applying a hamming window to the signal
x = 0:0.001:1.023; %Create a vector of length 1024. (Power of 2)
f = sin(20*2*pi*x); %Create a sinusoid on this function.
for element = 1:numel(f)
if element < 400
f(element) = 0;
end
if element > 600
f(element) = 0;
end
end
ham = transpose(hamming(202));
for element = 1:numel(f)
if (element >= 400) && (element <= 600)
f(element) = f(element) * ham(element - 399); %Multiply f by the hamming window.
end
end
ham2 = x * 0;
for element = 1:numel(ham2)
```

```
if (element >= 400) && (element <= 600)
ham2(element) = ham(element - 399);
end
end

hold on;
plot(x, f,'linewidth',2,'color',[0.3,0.5,1]) %Plot it.
plot(x, ham2, 'linewidth',2,'color',[0.1,0.5,0.3])
hold off;

%Fancy plotting settings
axis([0.3,0.7,-1.2,1.2]);
legend('Hamming Windowed Function', 'The Hamming Window');
xlabel('Time (s)','FontSize',12,'FontWeight','bold');
ylabel('Amplitude','FontSize',12,'FontWeight','bold');
title('Our Signal (but windowed with a Hamming window)','FontSize',16,'FontWeight','bold');
%% Plotting the FFT of the hamming windowed function
L = 1023; %The length of our vector.
Fs = 1000; %Set the frequency of the sampling. This is 1000Hz.
y = fft(f); %Take the fft of our function.
p2 = abs(y/L); %Take the absolute value of the Re+Im values returned.
p1 = p2(1:L/2+1); %Make the first half of p1 equal to p2.
p1(2:end-1) = 2*p1(2:end-1); %Fold the second half of p2 back onto the first half of p1.
x_axis = Fs*(0:(L/2))/L; %Scaling our new 'x' axis so we can plot in ms.
plot(x_axis, p1, 'linewidth',2,'color',[0.3,0.5,1]) %Our FFT.

%Fancy plotting settings
axis([0,100,0,0.13])
xlabel('Frequency (Hz)','FontSize',12,'FontWeight','bold');
ylabel('Amplitude','FontSize',12,'FontWeight','bold');
title('Frequency Domain of our Signal with a Hamming window','FontSize',16,'FontWeight','b
%% On a semi-log graph this time
semilogy(x_axis, p1, 'linewidth',2,'color',[0.3,0.5,1]) %Our FFT.

%Fancy plotting settings
axis([0,100,0,0.3])
xlabel('Frequency (Hz)','FontSize',12,'FontWeight','bold');
ylabel('Amplitude','FontSize',12,'FontWeight','bold');
title('Frequency Domain of our Signal with a Hamming window (semi-log)','FontSize',16,'Fon
```

Double Sinusoid

```
%% Now lets do this all again, but with two signals to show why this is useful with a more
%Creating a basic sinusoidal function:
x = 0:0.001:1.023; %Create a vector of length 1024. (Power of 2)
f = sin(20*2*pi*x) + 0.1*sin(29*2*pi*x); %Create a sinusoid on this function.
plot(x, f,'linewidth',2,'color',[0.3,0.5,1]) %Plot it.

%Fancy plotting settings
axis([0,1.023,-1.5,1.5]);
xlabel('Time (s)','FontSize',12,'FontWeight','bold');
ylabel('Amplitude','FontSize',12,'FontWeight','bold');
```

```
title('Our Signal','FontSize',16,'FontWeight','bold');

%% Doing an FFT on our function and presenting the results
L = 1023; %The length of our vector.
Fs = 1000; %Set the frequency of the sampling. This is 1000Hz.
y = fft(f); %Take the fft of our function.
p2 = abs(y/L); %Take the absolute value of the Re+Im values returned.
p1 = p2(1:L/2+1); %Make the first half of p1 equal to p2.
p1(2:end-1) = 2*p1(2:end-1); %Fold the second half of p2 back onto the first half of p1.
x_axis = Fs*(0:(L/2))/L; %Scaling our new 'x' axis so we can plot in ms.
plot(x_axis, p1, 'linewidth',2,'color',[0.3,0.5,1]) %Our FFT.

%Fancy plotting settings
axis([0,100,0,0.8])
xlabel('Frequency (Hz)','FontSize',12,'FontWeight','bold');
ylabel('Amplitude','FontSize',12,'FontWeight','bold');
title('Frequency Domain of our Signal','FontSize',16,'FontWeight','bold');
%% Presenting the same results on a semi-log graph
semilogy(x_axis, p1, 'linewidth',2,'color',[0.3,0.5,1])

%Fancy plotting settings
axis([0,100,0,0.8])
xlabel('Frequency (Hz)','FontSize',12,'FontWeight','bold');
ylabel('Amplitude','FontSize',12,'FontWeight','bold');
title('Frequency Domain of our Signal (but semi-log)','FontSize',16,'FontWeight','bold');
```

Double Sinusoid with Rect Window

```
%%
%Creating a basic sinusoidal function:
x = 0:0.001:1.023; %Create a vector of length 1024. (Power of 2)
f = sin(20*2*pi*x) + 0.1*sin(29*2*pi*x); %Create a sinusoid on this function.
for element = 1:numel(f)
if element < 400
f(element) = 0;
end
if element > 600
f(element) = 0;
end
end
plot(x, f,'linewidth',2,'color',[0.3,0.5,1]) %Plot it.

%Fancy plotting settings
axis([0,1.023,-1.5,1.5]);
xlabel('Time (s)','FontSize',12,'FontWeight','bold');
ylabel('Amplitude','FontSize',12,'FontWeight','bold');
title('Our Signal','FontSize',16,'FontWeight','bold');

%% Doing an FFT on our function and presenting the results
L = 1023; %The length of our vector.
Fs = 1000; %Set the frequency of the sampling. This is 1000Hz.
y = fft(f); %Take the fft of our function.
```

```
p2 = abs(y/L); %Take the absolute value of the Re+Im values returned.
p1 = p2(1:L/2+1); %Make the first half of p1 equal to p2.
p1(2:end-1) = 2*p1(2:end-1); %Fold the second half of p2 back onto the first half of p1.
x_axis = Fs*(0:(L/2))/L; %Scaling our new 'x' axis so we can plot in ms.
plot(x_axis, p1, 'linewidth',2,'color',[0.3,0.5,1]) %Our FFT.

%Fancy plotting settings
axis([0,100,0,0.25])
xlabel('Frequency (Hz)','FontSize',12,'FontWeight','bold');
ylabel('Amplitude','FontSize',12,'FontWeight','bold');
title('Frequency Domain of our Signal','FontSize',16,'FontWeight','bold');
%% Presenting the same results on a semi-log graph
semilogy(x_axis, p1, 'linewidth',2,'color',[0.3,0.5,1])

%Fancy plotting settings
axis([0,100,0,0.8])
xlabel('Frequency (Hz)','FontSize',12,'FontWeight','bold');
ylabel('Amplitude','FontSize',12,'FontWeight','bold');
title('Frequency Domain of our Signal (but semi-log)','FontSize',16,'FontWeight','bold');
```

Double Sinusoid with Hamming Window

```
%Creating a basic sinusoidal function:
x = 0:0.001:1.023; %Create a vector of length 1024. (Power of 2)
f = sin(20*2*pi*x) + 0.1*sin(29*2*pi*x); %Create a sinusoid on this function.
for element = 1:numel(f)
if element < 400
f(element) = 0;
end
if element > 600
f(element) = 0;
end
end
ham = transpose(hamming(202));
for element = 1:numel(f)
if (element >= 400) && (element <= 600)
f(element) = f(element) * ham(element - 399); %Multiply f by the hamming window.
end
end
plot(x, f,'linewidth',2,'color',[0.3,0.5,1]) %Plot it.

%Fancy plotting settings
axis([0,1.023,-1.5,1.5]);
xlabel('Time (s)','FontSize',12,'FontWeight','bold');
ylabel('Amplitude','FontSize',12,'FontWeight','bold');
title('Our Signal','FontSize',16,'FontWeight','bold');

%% Doing an FFT on our function and presenting the results
L = 1023; %The length of our vector.
Fs = 1000; %Set the frequency of the sampling. This is 1000Hz.
y = fft(f); %Take the fft of our function.
p2 = abs(y/L); %Take the absolute value of the Re+Im values returned.
```

```
p1 = p2(1:L/2+1); %Make the first half of p1 equal to p2.
p1(2:end-1) = 2*p1(2:end-1); %Fold the second half of p2 back onto the first half of p1.
x_axis = Fs*(0:(L/2))/L; %Scaling our new 'x' axis so we can plot in ms.
plot(x_axis, p1, 'linewidth',2,'color',[0.3,0.5,1]) %Our FFT.

%Fancy plotting settings
axis([0,100,0,0.15])
xlabel('Frequency (Hz)','FontSize',12,'FontWeight','bold');
ylabel('Amplitude','FontSize',12,'FontWeight','bold');
title('Frequency Domain of our Signal','FontSize',16,'FontWeight','bold');
%% Presenting the same results on a semi-log graph
semilogy(x_axis, p1, 'linewidth',2,'color',[0.3,0.5,1])

%Fancy plotting settings
axis([0,100,0,0.15])
xlabel('Frequency (Hz)','FontSize',12,'FontWeight','bold');
ylabel('Amplitude','FontSize',12,'FontWeight','bold');
title('Frequency Domain of our Signal (but semi-log)','FontSize',16,'FontWeight','bold');
```