Notes and Reflections

I'm trying a black background this time, just for fun. It should highlight inserted images more, which is both a good and a bad thing. As always, reflection is blue.

This Bset made me stop and look back at the explanations a lot, which implies either:
1) I was repeating the concepts, and thus becoming comfortable with them
OR
2) I haven't learned the concepts, as evidenced by the fact that I needed to look back in the packet to get the relevant formulas, rather than reaching to memory.

I'm leaning toward the first explanation, but time will tell.

I probably spent about the right amount of time on this bset, which conflicts with my semesterly goal of spending too little time on things to know what that feels like. As such, I should probably get more liberal with skipping or half-doing problems when I can improve my $\frac{learning}{time}$ ratio. For example, I should *really* stop typesetting my equations ;)

Solutions begin on next page

1. Moving Average

$$y[n] = \frac{1}{3}(x[n+1] + x[n] + x[n-1]).$$

Because the DTFT (for which I'll use the operator D) is additive,

$$Y(\Omega) = D(y[n]) = 1/3(D(x[n+1]) + D(x[n]) + D(x[n-1])$$

Using the second property,

$$D(x[n-1]) = e^{-j\Omega}D(x[n]) = e^{-j\Omega}X(\Omega)$$

and

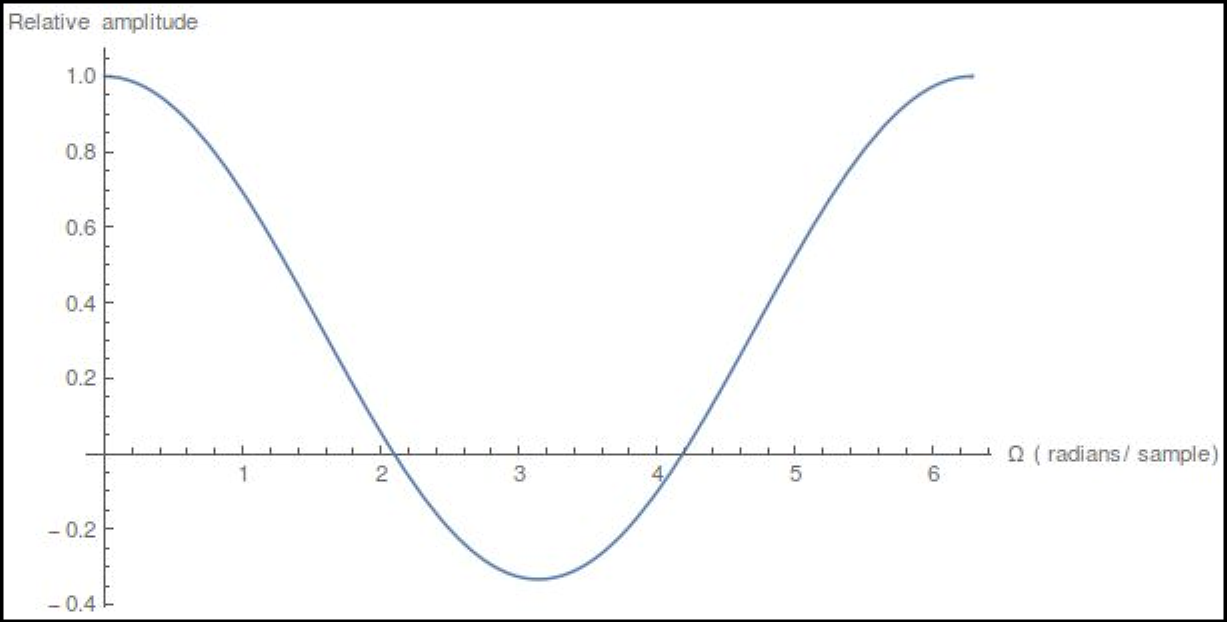
$$D(x[n+1]) = e^{j\Omega}D(x[n]) = e^{j\Omega}X(\Omega)$$

Combining these results,

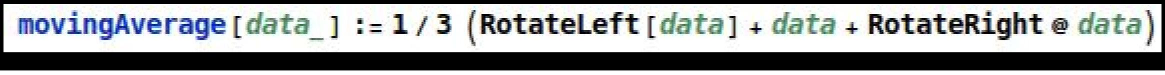
$$D(x[n-1]) + D(x[n+1]) = X(\Omega) * (e^{j\Omega} + e^{-j\Omega}) = X(\Omega) * 2\ cos(\Omega)$$

$$Y(\Omega) = \tfrac{1}{3}X(\Omega)(2\ cos(\Omega) + 1) \blacksquare$$

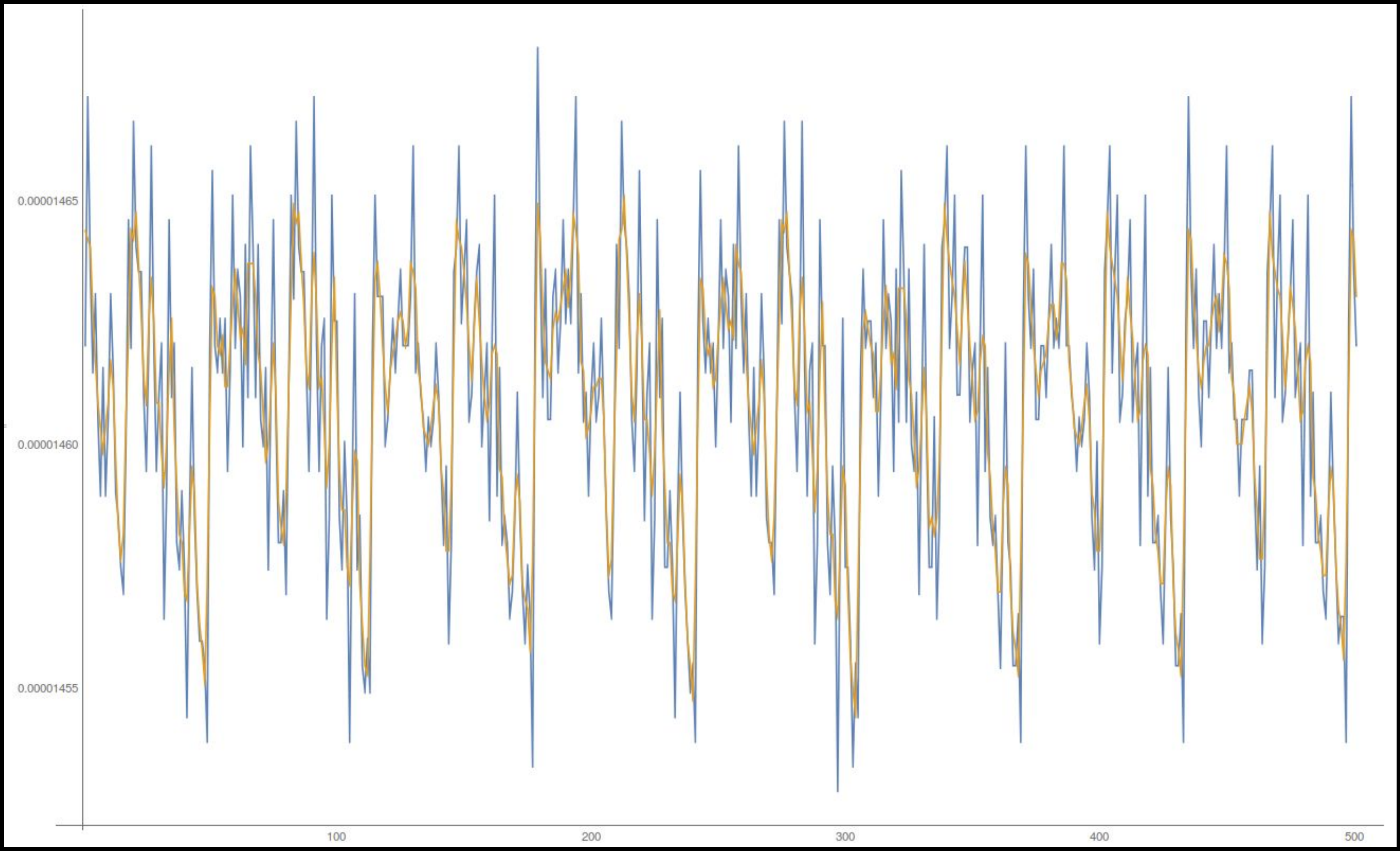
This implies that the frequency content of y is very similar to x for low frequencies (those near 0 rad/sample). For high frequencies approaching 1 rad/sample, the amplitude in y drops, even going negative.



c) The difference between the original sound and the filtered one is almost impossible to hear. This isn’t particularly surprising, as the difference doesn’t become pronounced until frequencies of ~4000 hz, substantially higher than the major frequency components of the signal.

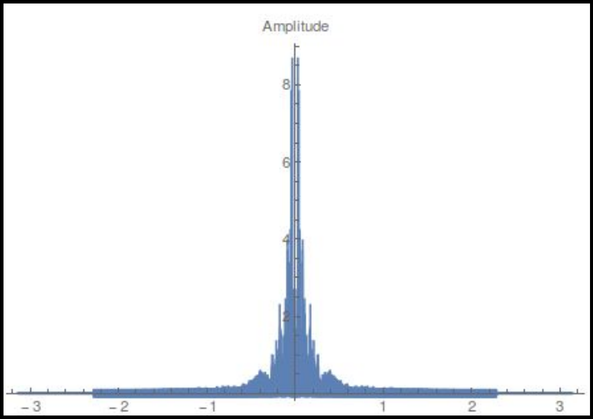


```
movingAverage[data_] := 1 / 3 (RotateLeft[data] + data + RotateRight @ data)
```

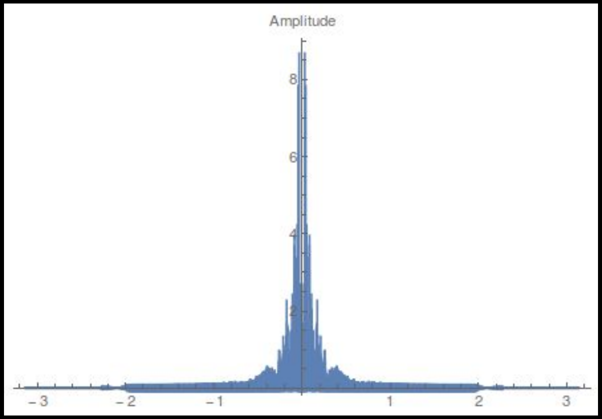


The effect would be more audible if the sample rate was lower because more of the audible frequencies would be affected.

Before



After

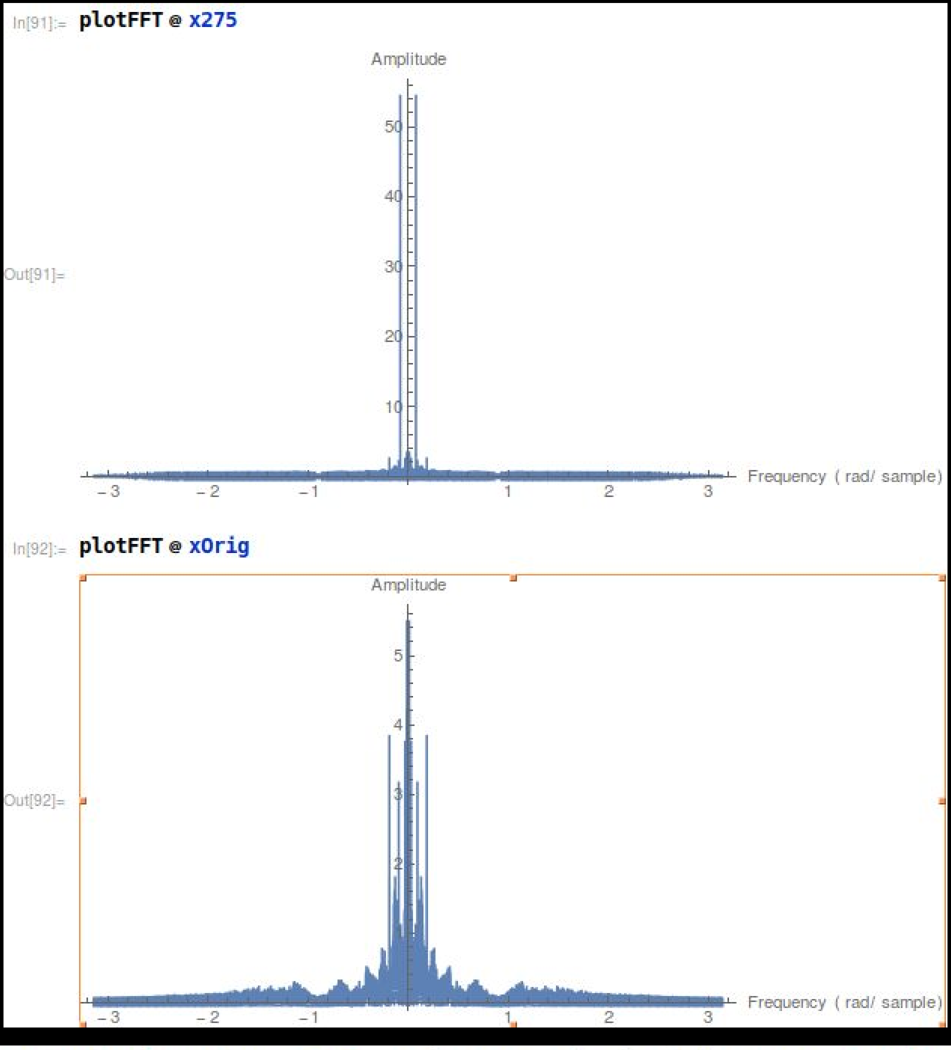


Putting these on the same axes would show that the signal after has less pronounced components around 2 rads/sec.

2. Removing Disturbances

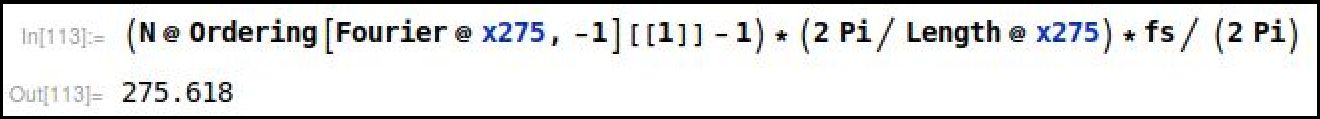
Eeeeee! I'm nerding out right now, I love this audio file! Thank you QEA teaching team, you're the best! (Although... a file without a built-in background hum would have been nice. I assume you know how to remove it.)

Frequency analysis of starting signal



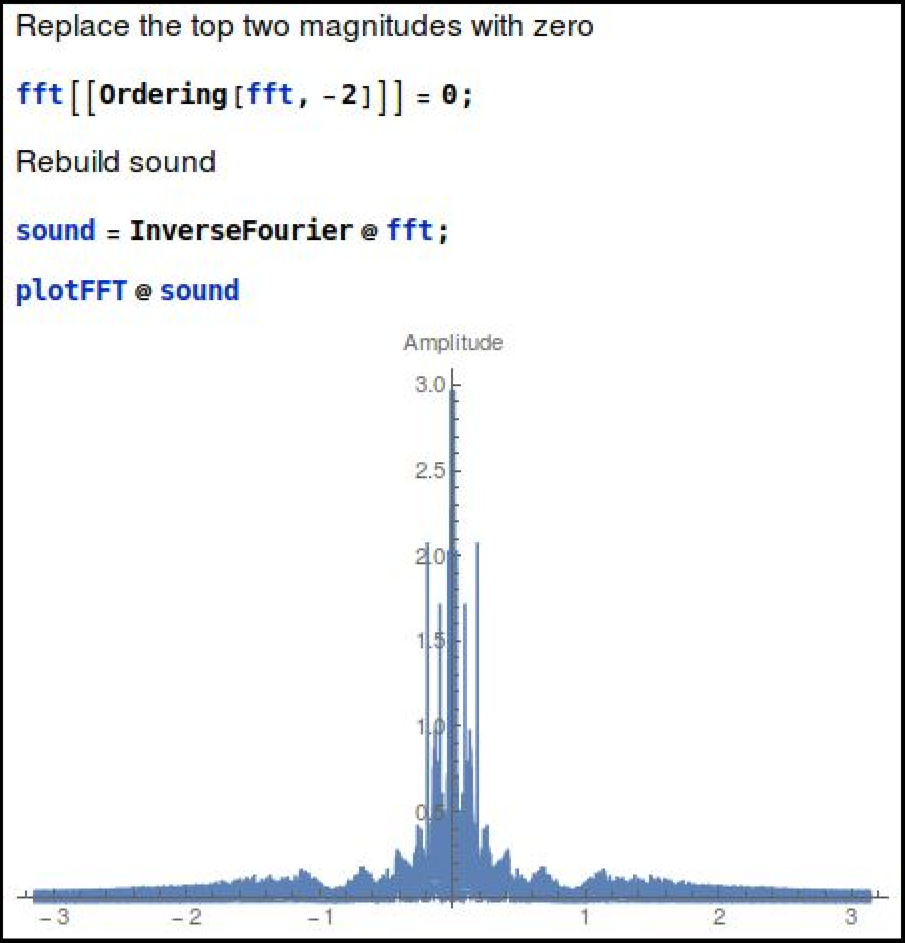
```
In[91]:= plotFFT @ x275
Out[91]=
```

```
In[92]:= plotFFT @ xOrig
Out[92]=
```



```
In[113]:= (N @ Ordering[Fourier @ x275, -1][[1]] - 1) * (2 Pi / Length @ x275) * fs / (2 Pi)
Out[113]= 275.618
```

Attempt 1.1: Remove the dominant frequencies



Replace the top two magnitudes with zero

```
fft[[Ordering[fft, -2]]] = 0;
```

Rebuild sound

```
sound = InverseFourier @ fft;
plotFFT @ sound
```

Observations: This procedure eliminates all/most of the noise but is hacky.

Attempt 1.2: Notch filter

```
In[32]:= notchFilter[data_, freq_, q_] :=
          Module[{a, b, p, qq, r, y},
           a = 2 q Cos @ freq;
           b = -q ^ 2;
           p = 1;
           qq = -2 Cos @ freq;
           r = 1;
           y = ConstantArray[0, Length @ data];
           Do[
            y[[n]] = a y[[n - 1]] + b y[[n - 2]] + p data[[n]] + qq data[[n - 1]] + r data[[n - 2]],
            {n, 3, Length @ data}];
           y
          ]
```

Pass the x275 through a notch filter

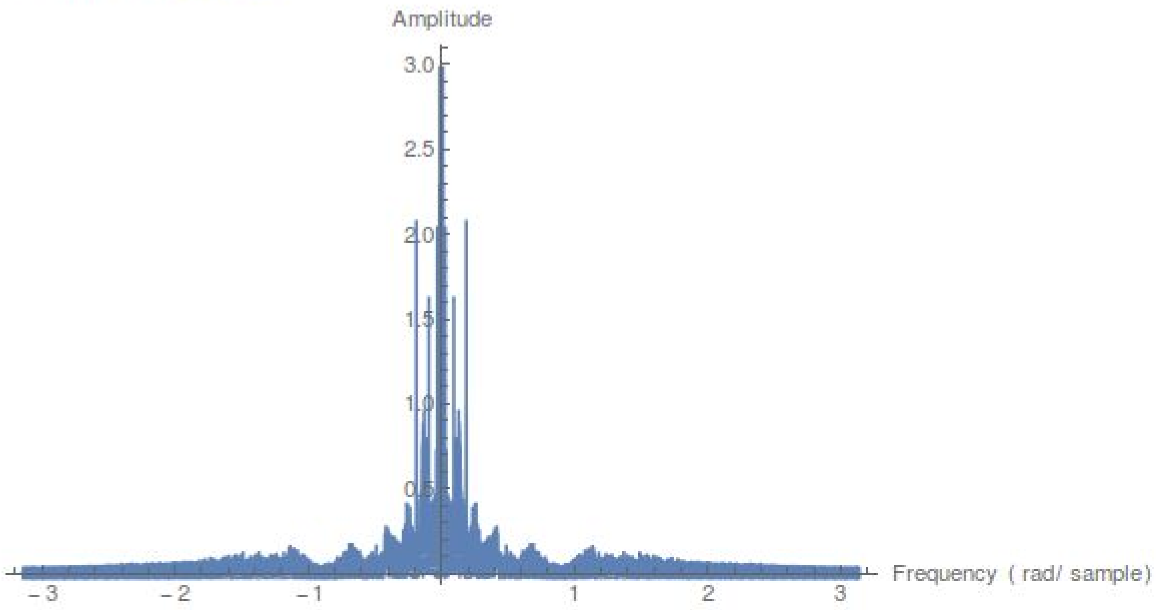
```
In[27]:= freq = x275freq * 2 Pi / fs

Out[27]= 0.0785379

In[63]:= filtered = notchFilter[x275, freq, .999];

In[57]:= plotFFT @ filtered

Out[57]=
```



```
In[66]:= playSound[filtered, fs]
```

Results: yep, it works.

Attempt 2.1: Remove from nonaligned signal

```
In[107]:= fft = Fourier @ x261;
```

Replace the top bunch of magnitudes with zero

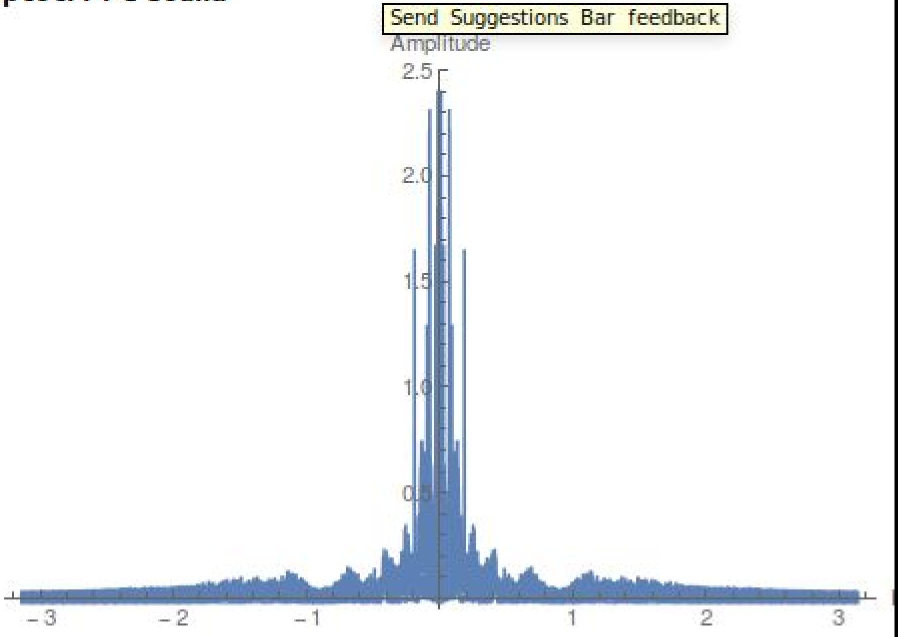
```
In[108]:= fft[[Ordering[Abs @ fft, -30]]] = 0;
```

Rebuild sound

```
In[109]:= sound = InverseFourier @ fft;

In[110]:= plotFFT @ sound
```

Out[110]=



Compare the results

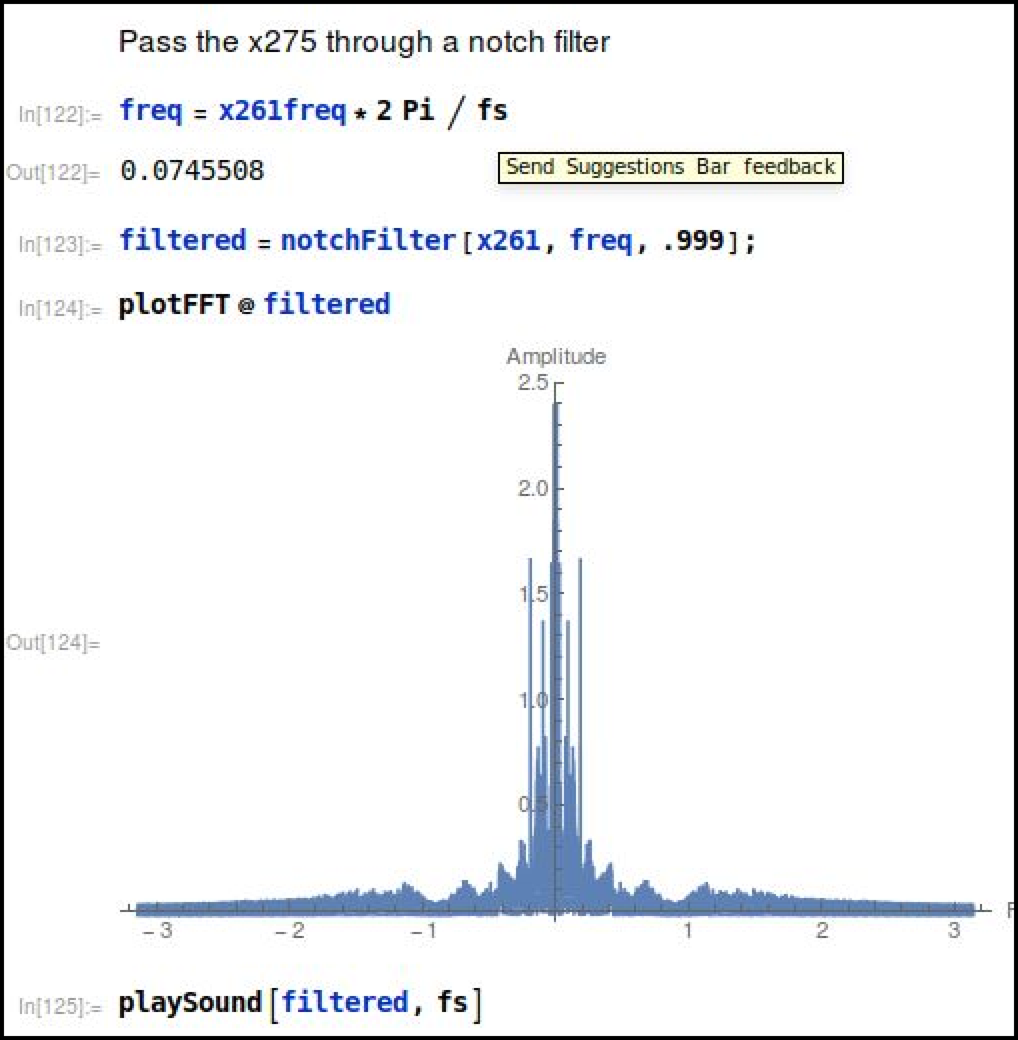
```
In[119]:= playSound[Re @ x261, fs]

In[117]:= playSound[Re @ sound, fs]

In[118]:= playSound[Re @ xOrig, fs]
```

Note that in this case, I needed to remove the 30 strongest frequency components before the offending noise (mostly) disappeared. This created nasty effects elsewhere in the signal, though not intolerably.

Attempt 2.2: Notch filter in nonaligned signal



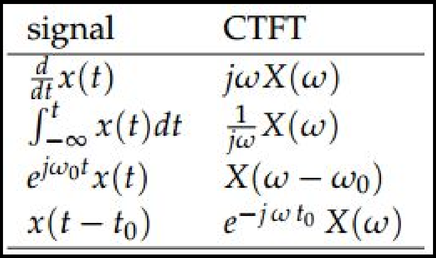
Pass the x275 through a notch filter

```
In[122]:= freq = x261freq * 2 Pi / fs
Out[122]= 0.0745508
In[123]:= filtered = notchFilter[x261, freq, .999];
In[124]:= plotFFT @ filtered
Out[124]=
In[125]:= playSound[filtered, fs]
```

This worked better than Attempt 2.1 at preserving the original signal, but most importantly, it did so with absolutely no alterations to the code from before. This demonstrates it as a more generalizable and useful technique.

OK, the results of this technique were pretty cool, but I still don't feel like it was well-motivated as a thing to try. In particular, I don't yet have an "exhaustive list of all the filters I could ever want" stored in my brain, so in some ways the "directly bash around with the DFT" approach seems more generalizable.

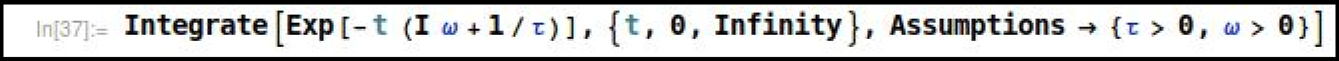
3. Continuous Time Fourier Transform



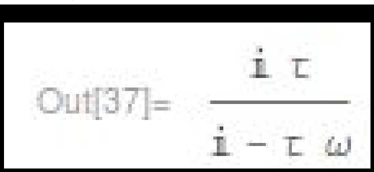
signal	CTFT
$\frac{d}{dt}x(t)$	$j\omega X(\omega)$
$\int_{-\infty}^{t} x(t)dt$	$\frac{1}{j\omega}X(\omega)$
$e^{j\omega_0 t}x(t)$	$X(\omega-\omega_0)$
$x(t-t_0)$	$e^{-j\omega t_0}X(\omega)$

a. Half-sinusoid

$$X(\omega) = \int_0^\infty e^{-t(j\omega+1/\tau)}dt$$



```
In[37]:= Integrate[Exp[-t (I ω + 1 / τ)], {t, 0, Infinity}, Assumptions → {τ > 0, ω > 0}]
```



```
Out[37]= i τ / (i - τ ω)
```

b. AM modulation

$$y(t) = \frac{1}{2}e^{j\omega_0 t}x(t) + \frac{1}{2}e^{-j\omega t}x(t)$$

$$Y(\omega) = \frac{1}{2}\left(X(\omega-\omega_0) + X(\omega+\omega_0)\right)$$

c. Differential Equation handling

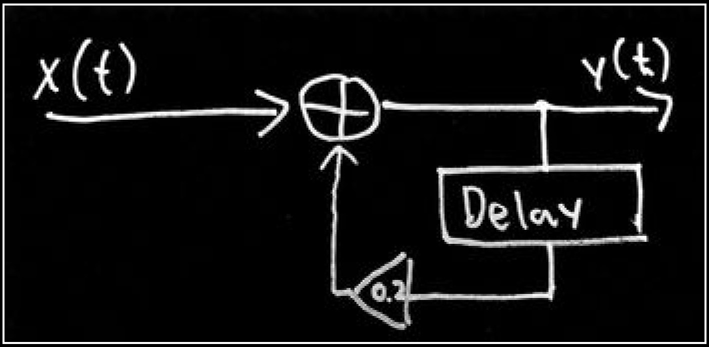
$$y(t) = \frac{d}{dt}x(t) + 3x(t)$$

$$Y(\omega) = \underset{\uparrow}{j}\omega X(t) + 3X(t)$$

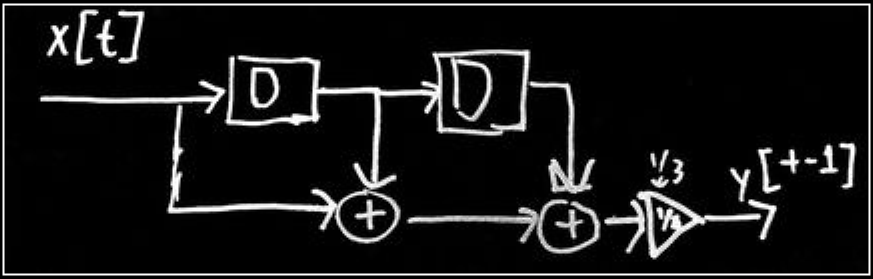
90° phase shift

This is where the beauty of representing phases in complex space becomes extremely obvious, the 90° rotation performed by multiplying by j is beautiful. Although... it wouldn't be that bad with real sines and cosines.

4. Adding an Echo



5. Moving Average block diagram



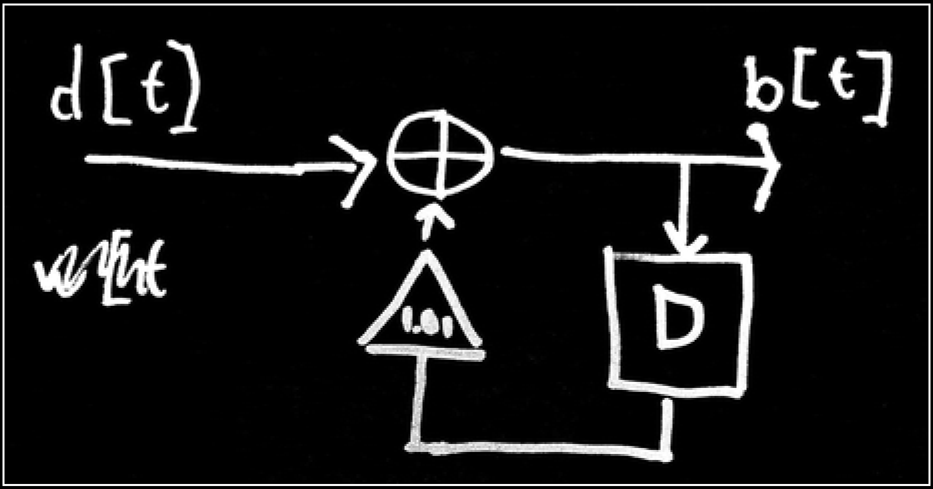
6. Noise and Offset handling

The relevant noise/offset is simply added to the input signal before the rest of the block diagram. Figure 5 is moderately sensitive to DC offset input, increasing the output signal proportionally to the input offset. Figure 4 is very sensitive to noise, with small amplitude noises producing arbitrarily large swings in output signal if they happen at high enough frequencies.

7. Bank account

Where b[t] is the balance output
and d[t] is the deposited amount

$$b[t] \ = 1.01\ b[t-1] + d[t]$$



Why does this problem describe the steps as "creating" a model using mathematical notation, then "implementing" it with a block diagram? Normally, I would assign those steps the other way, but that is dependant on block diagrams being non-computable representations.

8. Circuit analysis

$$V_R(t) = R \cdot i(t)$$

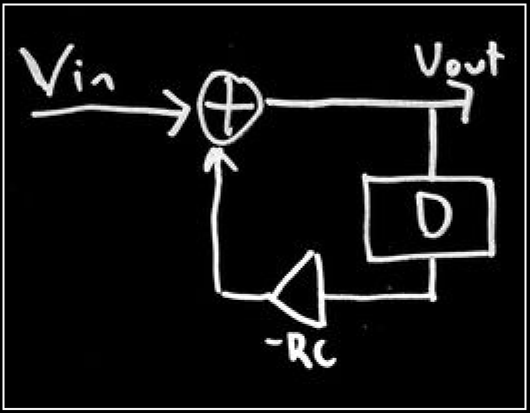
$$i(t) = \frac{V_R(t)}{R} = \frac{V_{in}(t) - V_{out}(t)}{R}$$

$$V_{out}(t) = \frac{1}{C}\int_{-\infty}^{t} \frac{V_{in}(\tau) - V_{out}(\tau)}{R}\, d\tau$$

$$V_{out}(t) = \frac{1}{RC}\int_{-\infty}^{t} V_{in} - V_{out}\, d\tau$$

$$\frac{d}{dt}V_{out}(t) = \frac{1}{RC}\left(V_{in}(t) - V_{out}(t)\right)$$

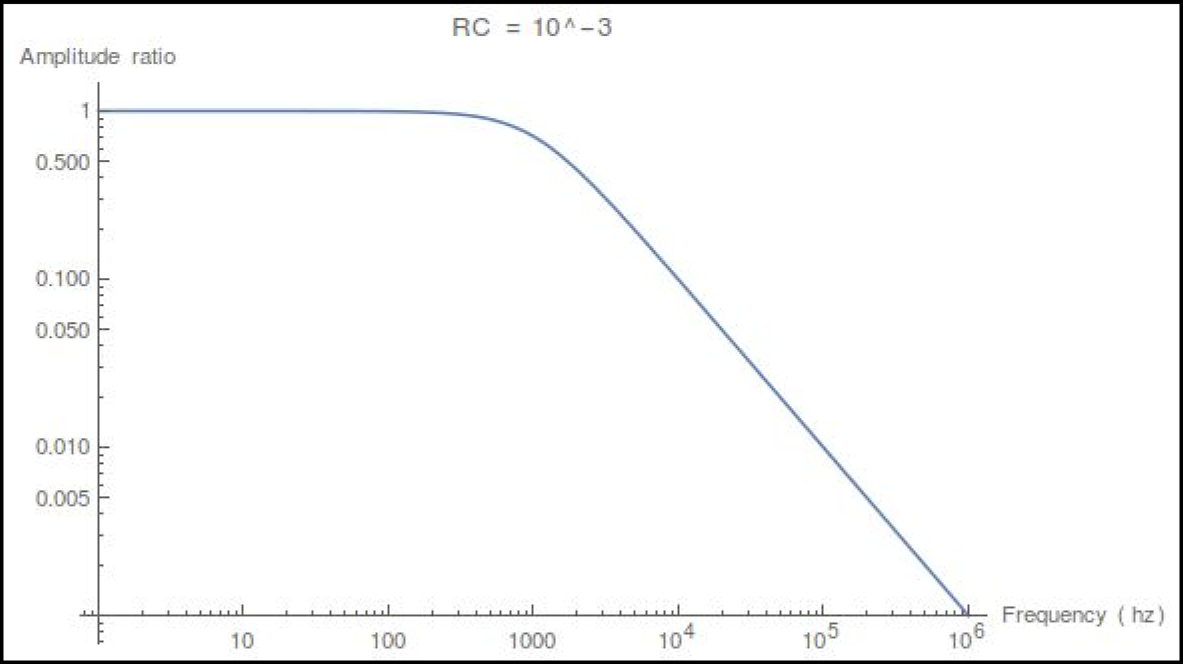
$$V_{in}(t) = V_{out}(t) + RC\, V_{out}'(t)$$



$$V_{in}(w) = V_{out}(w) + RCjwV_{out}(w)$$

$$V_{in}(w) = V_{out}(w)(1 + RCjw)$$

$$\frac{V_{out}}{V_{in}}(w) = \frac{1}{1 + RCjw}$$



This is probably the same analysis we went through in iSIM, but it makes a lot more sense this time. As expected, BodePlot[] confirms that the frequency shift is as expected.