Notes and Reflections

This should be pretty obvious, but this problem set was significantly less meaty than it could have been. It took me ~2.5 hours, and that seems to be normal across the class. It is possible that was intentional, but I personally would have preferred a little more content.

I definitely intend to read through and parse the solutions for this exercise set as a way to get a second run-through of the content because I can solve the problems here, but probably can't yet generate them myself.

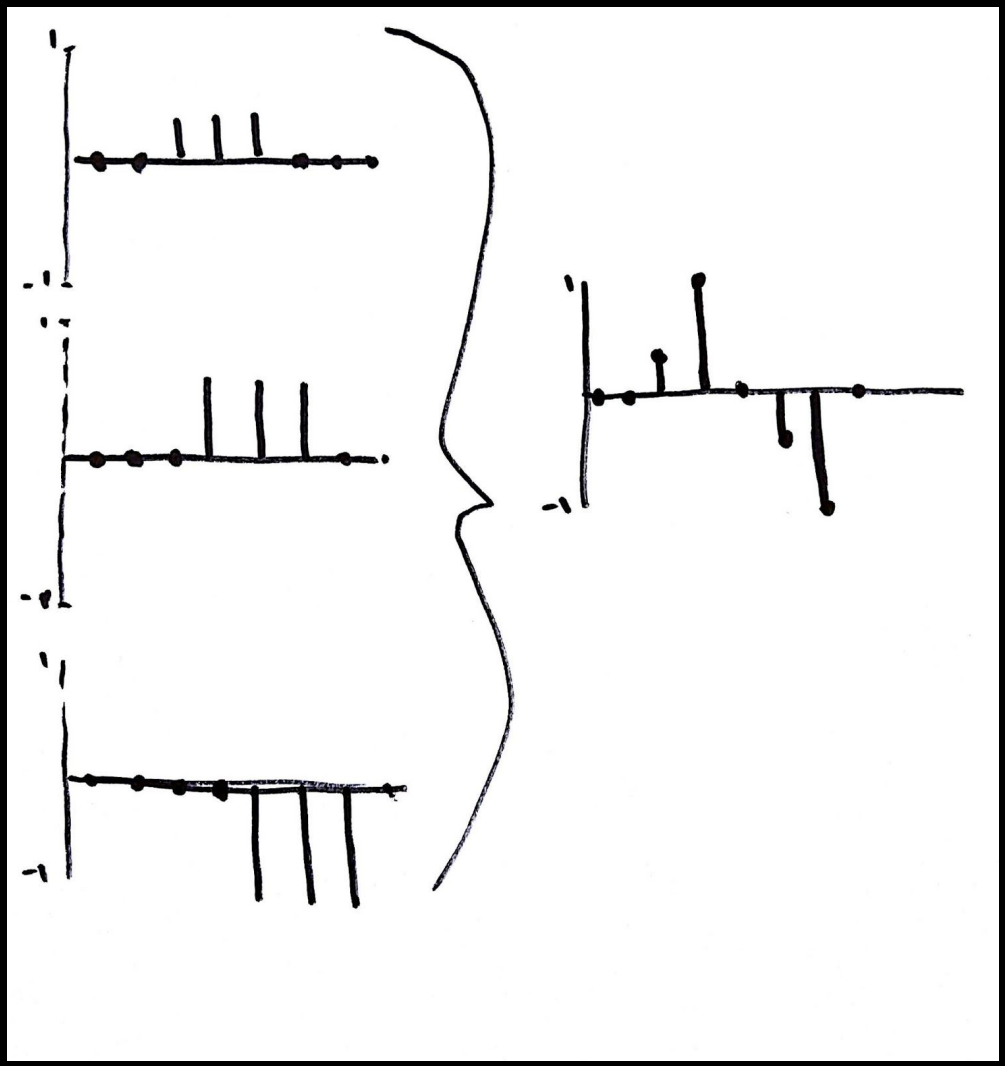
Table of Contents

Solutions begin on next page

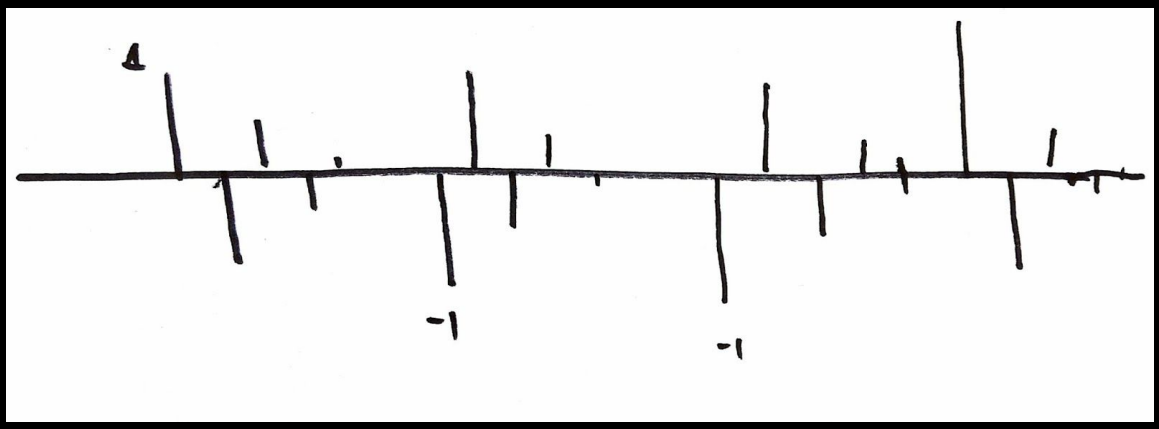
1. Moving average impulse response

The moving average filter has an impulse response of [⅓, ⅓, ⅓, 0, 0, ...]

2. Graphical convolution



3. Simple communication system model



If the time between the data samples is substantially shorter than the duration of h[n], the echos of one sample could cancel out parts of the other signal, completely covering the main signal to the point it is indistinguishable from noise.

It occurs to me that this effect can be avoided if the communication system understands the impulse response of the system extremely precisely, and can thus subtract out its influence. To my knowledge, no communication system does this, and I don't fully know why. Perhaps it is simply a ridiculously precise requirement for analyzing the signal in complicated ways at high frequencies.

4. Guitar in the stairwell

A finger snap produces a single high pressure wave, or impulse. Because other interfering things (like your hand) are relatively close, I'm worried about them causing problems by contributing to the recording, but it is still reasonably close.

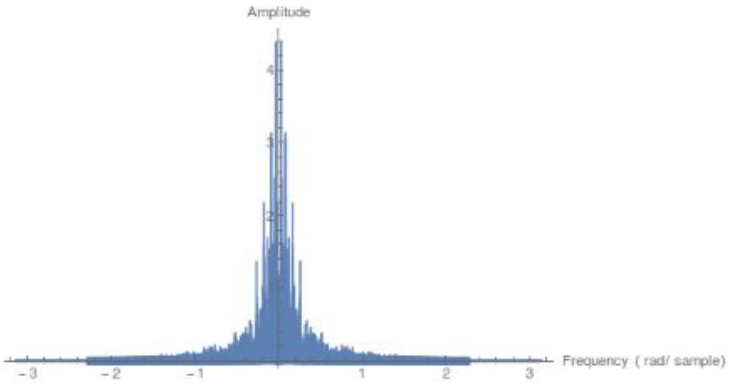
```
In[85]:= With[{data = Import["provided files/guitar_stairwell.mat", "LabeledData"]},
          fs = ("Fs" /. data)[[1, 1]];
          hstairwell = ("h_stairwell" /. data)[[All, 1]];
          x = ("x" /. data)[[All, 1]];
         ]

In[97]:= playSound[x, fs]

In[98]:= playSound[hstairwell, fs]

In[94]:= mutated = ListConvolve[hstairwell, x, {1, -1}, 0];
         mutated /= Max @ Abs @ mutated;

In[96]:= playSound[mutated, fs]
```

The output sound was surprisingly squeaky, which I suppose is simply the effect of small-space echoing.

5. Sinc filtering

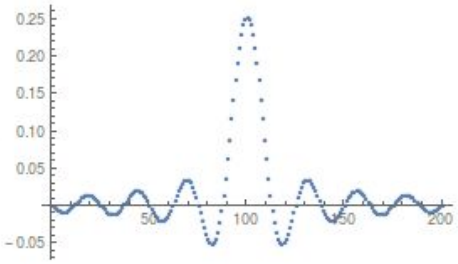
Lowpass

```
In[118]:= plotFFT[handel]
```



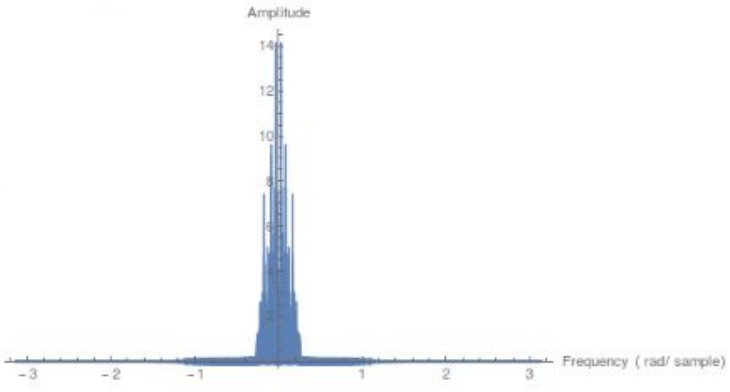
Cutoff frequency is in radians/sample

```
In[121]:= cutoff = Pi / 4;

In[152]:= sinc = N@cutoff / Pi * Sinc[cutoff * Range[-100, 100] / Pi];
          ListPlot[sinc, PlotRange → Full]
```



Convolve the input signal with the generated sinc function

```
In[154]:= filtered = ListConvolve[sinc, handel];

In[155]:= plotFFT[filtered]
```

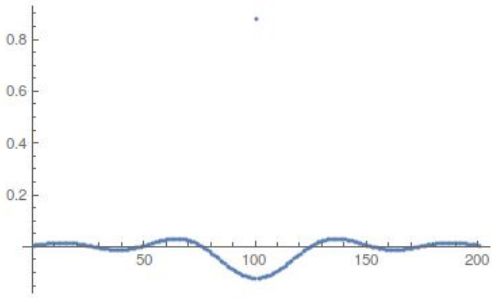


This resolves my concerns about needing to convert to and from frequency domain to accomplish arbitrary filtering operations. Particularly because it is possible to numerically integrate $H(\Omega)$, I feel confident creating an arbitrary filter operation using only a single convolution chip.

High pass

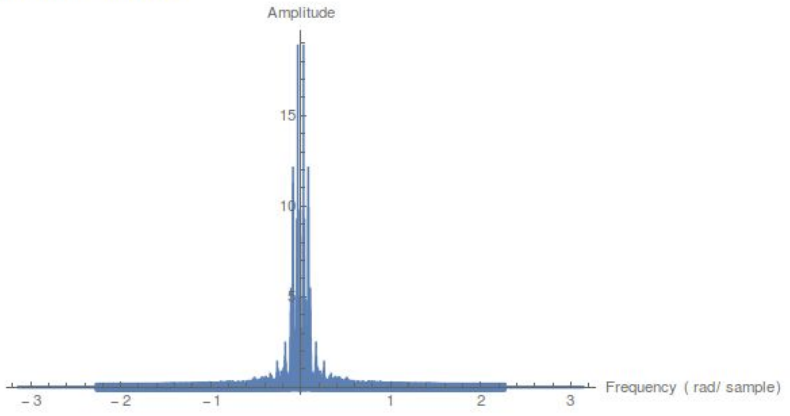
```
In[177]:= cutoff = Pi / 8;
          sinc = N @ cutoff / Pi * Sinc[cutoff * Range[-100, 100] / Pi];

In[191]:= highpassKernel = Join[ConstantArray[0, 100], {1}, ConstantArray[0, 100]] - sinc;
          ListPlot[highpassKernel, PlotRange → Full]
```

Out[192]=



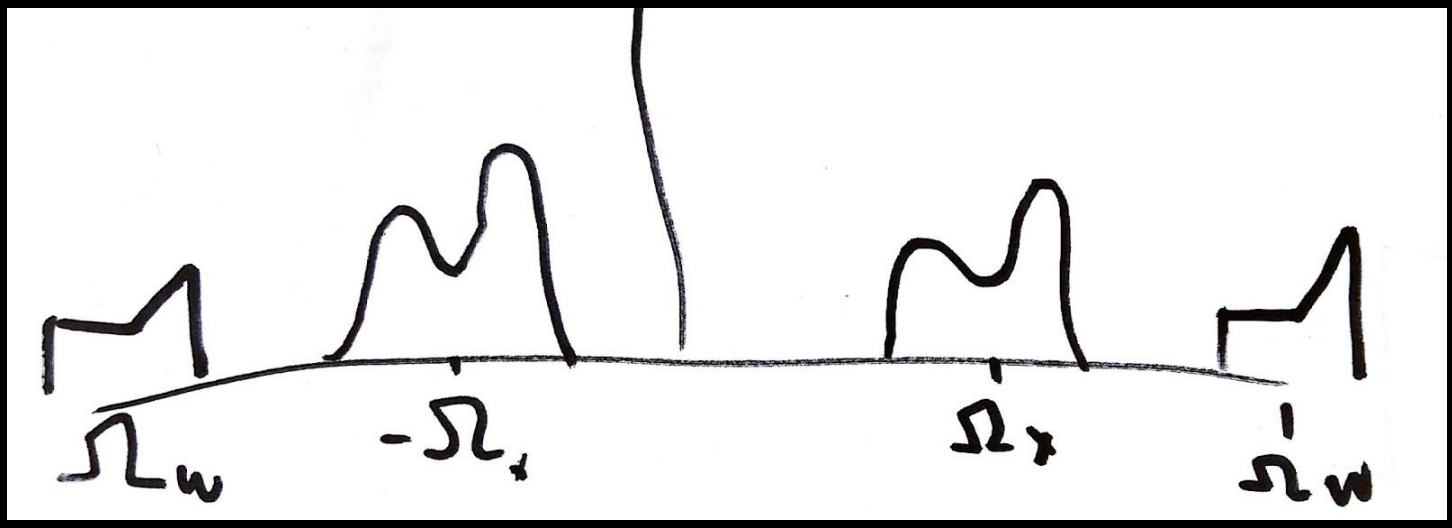

```
In[193]:= highpass = ListConvolve[highpassKernel, handel];

In[194]:= playSound[highpass, fs]

In[190]:= plotFFT @ highpass
```

Out[190]=



This operation should be equivalent to first taking the lowpass-filtered signal, and then subtracting it from the input. Unfortunately, when I implemented this it didn't do nearly as nice and obvious job as the lowpass did, and I don't know what I did wrong. I will look at the answer key to figure it out.

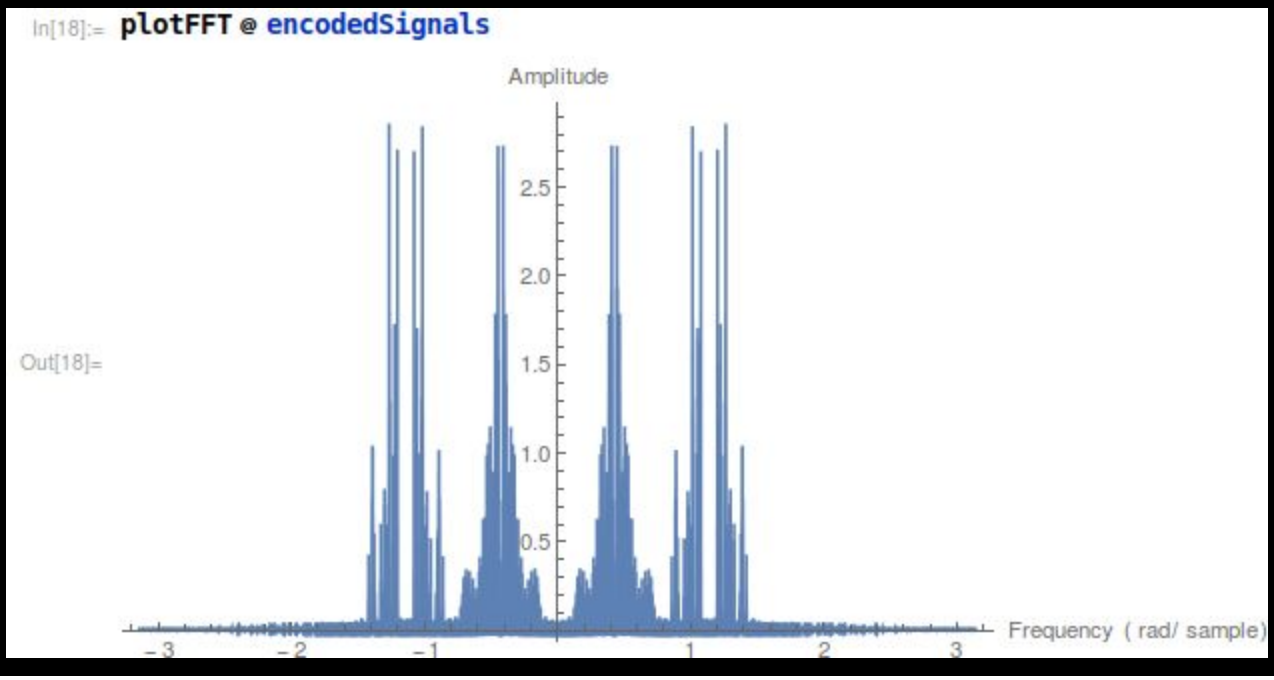
6. Communications overloading



To recover x[n], take the signal and multiply it again by $cos(\Omega_x n)$, then lowpass filter with cutoff frequency Ω_M, then amplify by a factor of 2. y[n] can be recovered the same way, except with the frequency Ω_w used instead.

In[18]:= `plotFFT @ encodedSignals`

Out[18]=



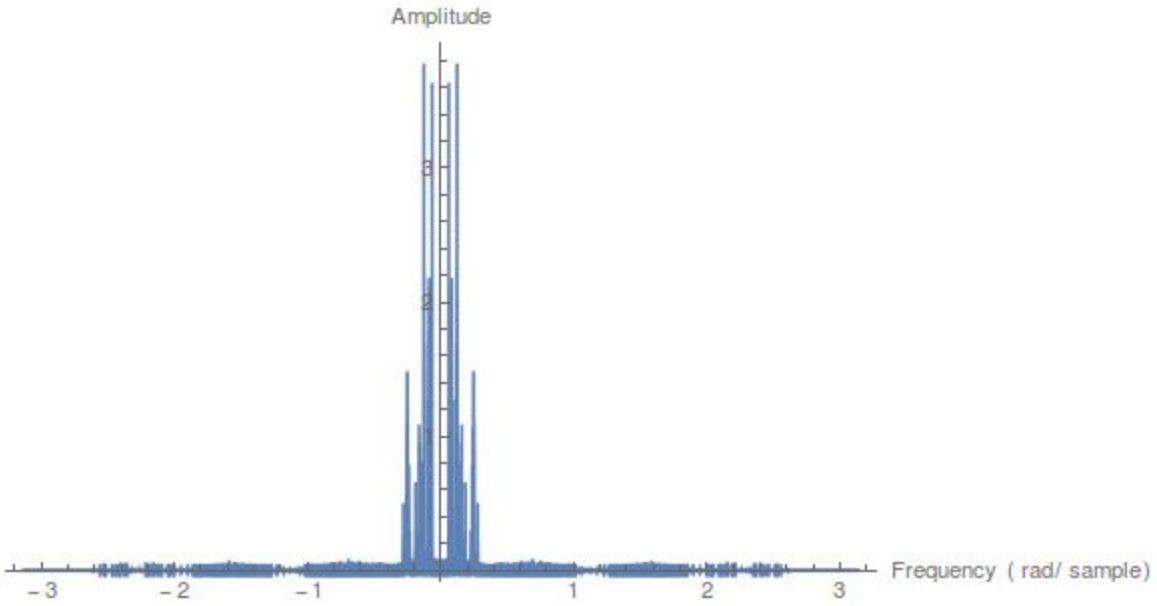

```
lowpass[signal_, freq_] :=
 Module[{sinc},
  sinc = N @ freq / Pi * Sinc[freq * Range[-60, 60] / Pi];
  (*ListPlot[sinc,PlotRange→Full];*)
  ListConvolve[sinc, signal]
 ]
```

```
In[60]:= recoverSignal[data_, freq_] :=
          Module[{shifted, filtered},
           shifted = data * Cos[Pi freq * Range @ Length @ data];
           filtered = lowpass[shifted, 0.3 Pi];
           filtered
          ]
```

Recover x[n]

```
In[73]:= freq = 16000 / fs;
         x = recoverSignal[encodedSignals, freq];

In[84]:= playSound[x, fs];

In[76]:= plotFFT @ x
```

Out[76]=



Recover w[n]

In[77]:=
```
freq = 6000 / fs;
w = recoverSignal[encodedSignals, freq];
```

In[88]:=
```
playSound[w, fs]
```

In[80]:=
```
plotFFT @ w
```

Out[80]=