EE102B Pring 2018-19 Pauly
Signal Processing and Linear Systems II

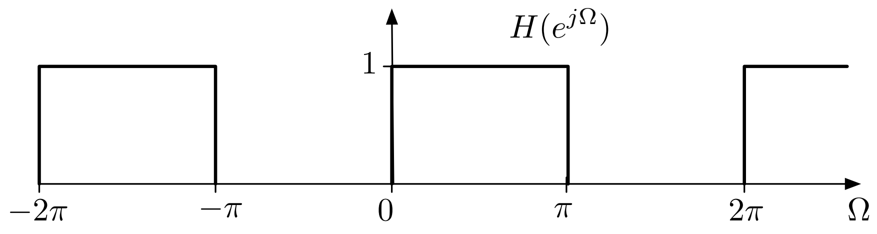
**Problem Set #1**

Problem Set Due: Friday, April 19

1. *Single Sideband Filters*

   As we talked about in class, the spectrum of a real signal has redundant information. The spectrum of the signal for negative frequencies is the conjugate of the spectrum for positive frequencies. We only need to transmit one of these, since we can reconstruct the other half at the receiver. This is called single sideband modulation (SSB). This widely used in amateur radio and other communication systems, and was due to Prof. Mike Villard of Stanford EE.

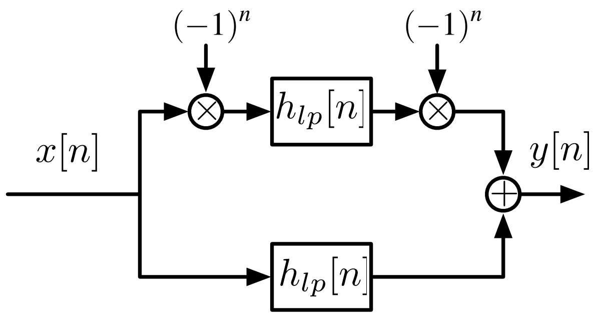
   There are many ways to create the single sideband signal. One is to do it digitally with a filter that looks like this



   Find the impulse response of this filter $h[n]$. Use the same approach as we used in class for highpass filter design.

2. *Discrete Time System Frequency Response*

   Determine the frequency response $H(e^{j\Omega})$ of the following system



   The lowpass filter $h_{lp}[n]$ has the following impulse response

$$h_{lp}[n] = \frac{1}{4}\text{sinc}\left(\frac{n}{4}\right)$$

   Plot the frequency response $H(e^{j\Omega})$.

3. *Discrete Time Differentiators*

The course notes includes a derivation of the impulse response of an ideal discrete time differentiator. It turns out that you mostly solved this problem already in HW 1, Problem 3, where you found the filter that would reconstruct the derivative of a signal from it's samples. In that problem the sampling period was $T = 1$ sample/second, so that $\omega = \Omega$.

What you showed was that

$$H(j\omega) = j\omega\Pi(\frac{\omega}{2\pi})$$

and the impulse response is

$$h(t) = \mathrm{sinc}'(t)$$

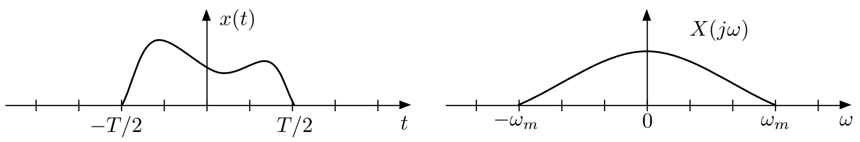
(a) Show that the ideal discrete time differentiator is

$$h[n] = \mathrm{sinc}'(n)$$

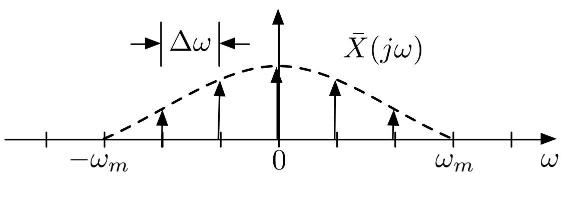
(b) Use the definition of $\mathrm{sinc}(t)$ to compute $\mathrm{sinc}'(t)$, and evaluate it at the integers $t = n$. It should simplify considerably. Show that you get the same expression as in the course notes. Ignore constant scale factors.

4. *Spectral Sampling*

So far, we have looked at sampling signals in time. In some applications, such as MRI, data is sampled in the spectral domain. We have a signal $x(t)$ with a spectrum $X(j\omega)$,



The duration of the signal is $T$, and the spectrum is band limited to $\pm\omega_m$. The sampled spectrum $\bar{X}(j\omega)$ looks like



where $\Delta\omega$ is the spacing of the samples of the spectrum.

a) Write an expression for $\bar{X}(j\omega)$.

b) Find the signal $\bar{x}(t)$ corresponding to the sampled spectrum.

c) Sketch the signal $\bar{x}(t)$.

d) Find $\Delta\omega$ such that $x(t)$ can be recovered exactly.

e) How many spectral samples $N_s$ are required?

f) How many samples $N_t$ would be required if the signal is sampled in time at the Nyquist rate?

## Laboratory

The purpose of this lab is to design a bandpass filter. It is very common when acquiring data, that there will be signals you are interested in at one frequency, and interfering signals at other frequencies. You will need to design filters that pass your signals, and suppress the interference.

**Task 1:** *Generate the C Major Chord Waveform*

For our lab, we will assume that we have a C major chord that consists of three frequencies. The goal is to extract on the middle frequency. Feel free to pick another more interesting chord if you'd like, but try to extract one of the notes in the middle of the chord.

Starting at middle C, the frequencies (in Hz) of the three notes are

```
>> fc = 261.63;
>> fe = 329.63;
>> fg = 392.00;
```

This is for a tempered scale. There are slight variations for other tunings. This is beyond the scope of the class.

The signal for a C major chord sampled at 8192 samples/second is

```
>> t = [1:8192]/8192;
>> cm = 0.25*(cos(2*pi*fc*t) + cos(2*pi*fe*t) + cos(2*pi*fg*t));
```

where the factor of 0.25 ensures that we are within the +/- 1 range required of `sound()`. Play this back, and see if it sounds like you expect.

```
>> sound(cm,8192);
```

To test to make sure you have the right signal, compute it's spectrum

```
>> cmf = fftshift(fft(fftshift(cm)));
```

The reason for the `fftshift()` operations is that `fft()` assumes that frequencies start at zero, and go to the sampling frequency at the end of the array. The effect of `fftshift()` is to put zero frequency in the middle of the array, which is easier to make sense of visually. Since we have 8192 samples, sampled at 8192 Hz, the total length of the signal is T=1 second, and the resolution of the spectrum is 1/T = 1 Hz. Plot the middle +/- 1000 Hz of the spectrum with

```
>> ft = [-1000:1000];
>> plot(ft, abs(cmf(4097+ft)))
```

since 4097 is the middle of the array, which is now zero frequency. Submit this plot. Are the peaks where you expect? Zoom in to make sure. The next part is much harder if they aren't in the right place!

Note that the peaks have different shapes, and have different amplitudes. We will explain why this happens later in the course.

**Task 2:** *Filtering*

The next task is to design a digital bandpass filter that passes the ″E″ frequency, and suppresses the ″C″ and ″G″. There are many possible solutions, and we'll give you credit for anything that works. A carefully crafted rect(n) will work, but is a fragile solution. Better solutions first design a lowpass filter, and then modulate it up to the right bandpass frequency. Things to think about are where the band edges of the ideal lowpass filter are, where is the center frequency you modulate to for the bandpass, and particularly the width of the transition bands.

Compute your filter that passes ″E″ as `he` and plot it. Plot the spectrum with

```
>> hera = abs(fftshift(fft(he,8192)));
>> plot(ft, hera(4097+ft));
```

This pads the filter to 8192 samples, so it compares directly to the signal spectrum. Zoom in to make sure that this filter passes the ″E″ frequency, and suppresses the others.

Filter the C major chord signal `cm`,

```
>> cme = conv(cm,he,'same');
```

and plot the resulting spectrum as above to show you have suppressed the other two notes. Listen to it

```
>> cme = cme/max(abs(cme));
>> sound(cme,8192)
```

to see if you hear just the single pure tone of the ″E″. The first step normalizes the signal to +/-1 so that sound `sound()` does the right thing, and doesn't clip the signal.