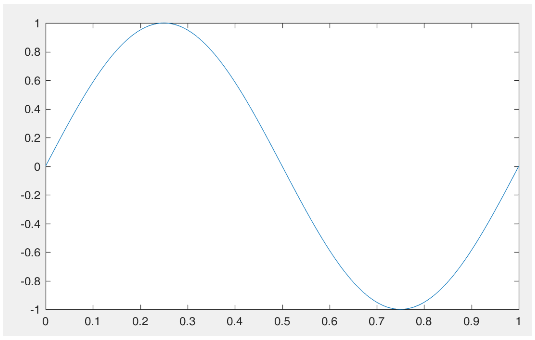
Creating signals as arrays

By creating a time array, and then passing that vector to a function such as `sin()`, we can apply that function to each elemnt in our time array. For example:

```
time = 0:.001:1;
sig = sin(2*pi*time);
plot(time,sig)
```

In this case, we made time be from 0 to 1, in increments of 0.001. We then made a sine wave on this interval, and plotted it. The plot shows



Note that while this looks like a continuous sine wave, since it is in a computer it is discrete. However, since the rate is so high, it approximates continuous time well. Lets make another sine wave, over a longer interval, that we can play on our computer speakers.

```
time = (0:8192) / 8192;
tone1 = sin(2*pi*400*time);
sound(tone1,8192);
```

We constructed a new time vector, that represents time in seconds, sampled at 8192Hz. This is chosen as that is a standard way to represent sound on a computer. `tone1` is a sin wave at 400Hz, sampled at 8192Hz. the `sound` function will play the signal over the computer speakers.

Fourier Transforms

Using the last example, we can now take a look at the spectrum of a sin wave.

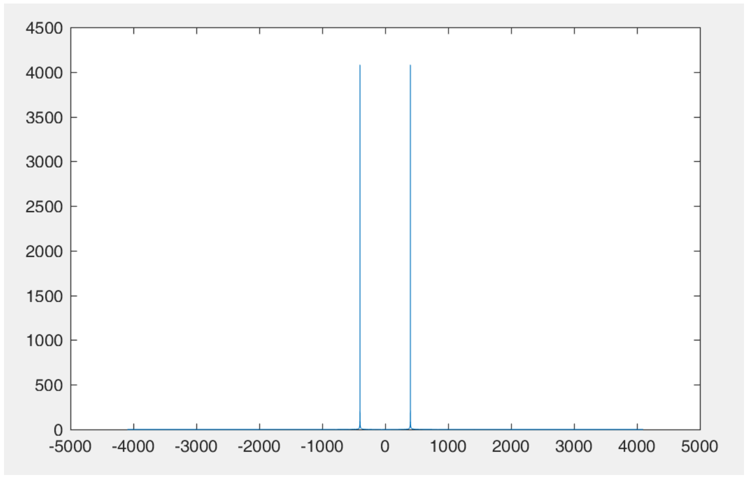
```
spec = fft(tone1);
spec_centered = fftshift(spec);
freqs = -8192/2:8192/2;
plot(freqs,abs(spec_centered));
```

`fft` computes the Discrete Fourier Transform on our signal `tone`. However, since our sampling rate is high, this approximates a Continuous Time Fourier Transform. It returns an array that is the same length as the input.

`fftshift` is an additional function that is used to shift the array returned by `fft` such that the negative frequencies are on the left, the positive frequencies are on the right, and 0 is in the middle.

`freqs` is simply an array that gives the frequencies in Hz corresponding to the elements of `spec_centered`.

Lastly, we call `abs` on `spec_centered` so that we plot the magnitude. The plot we obtain is:

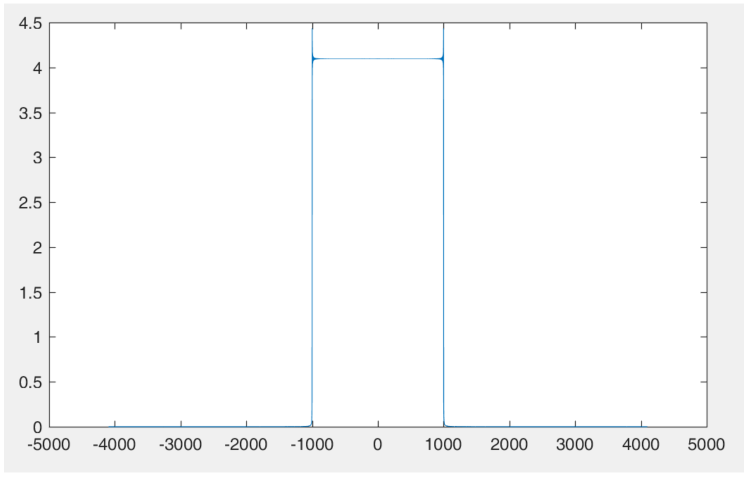


Here we see two "impulses" at -400Hz and 400Hz, as we should for a sine wave at 400Hz. Note that they do not actually go to infinity, but this is approximate as everything is actually being done in discrete time.
Lets do another example, with sinc, instead.

```
time = (-4096:4096) / 8192;
sig = sinc(2*1000*time);
spec = fft(sig);
spec_centered = fftshift(spec);
freqs = -8192/2:8192/2;
plot(freqs,abs(spec_centered));
```

Note that this time we start `time` at a negative value, so that time 0 is in the center. Also, we chose a frequency of 1000Hz to be the cutoff. There is no π this time, due to MATLAB's definition of sinc.

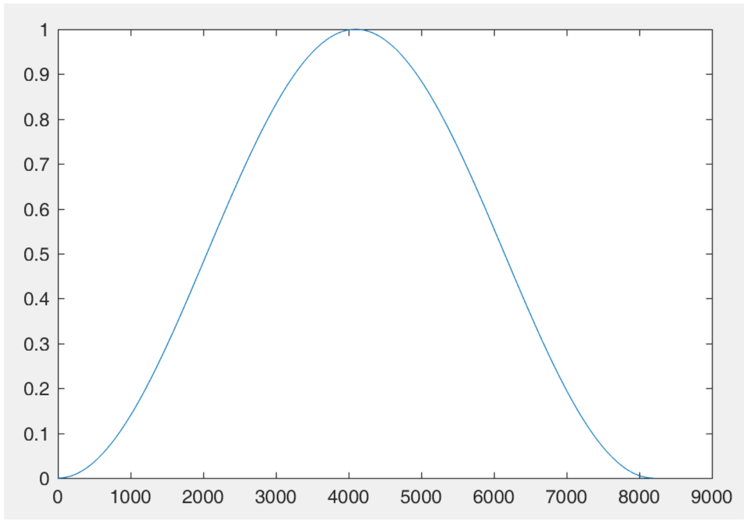
The resulting plot is:



Window Functions

In this course we often use window functions to reduce things like gibbs phenomena in our signals.
The `hann(N)` function returns an `N`-length array of the hann window, shown below.

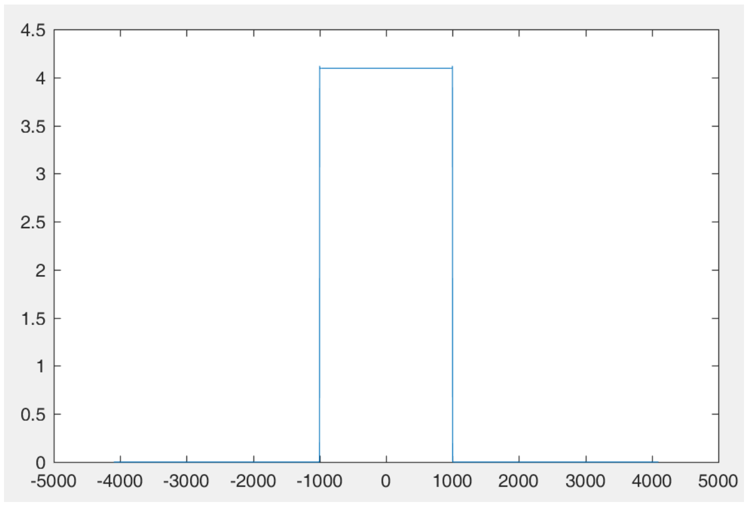
```
h = hann(8193)';
plot(h)
```



Note that we used the `'` opperator to take the transpose of the array, as `hann(N)` returns an $N \times 1$ array, and we want $1 \times N$.

Here is a simple example building off of the previous sinc example, but with a hanning window used.

```
time = (-4096:4096) / 8192;
sig = sinc(2*1000*time);
spec = fft(sig .*h );
spec_centered = fftshift(spec);
freqs = -8192/2:8192/2;
plot(freqs,abs(spec_centered));
```


4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
-5000
-4000
-3000
-2000
-1000
0
1000
2000
3000
4000
5000