# Example 1

January 27, 2017

### 0.1 Example

Make a sine wave which at 44100 Hz sampling rate has a frequency of 400 Hz at 1 second duration.

Hence we need 44100 samples, and 400 periods of our sinusoid in this second. Hence we can write our signal in python as:

```
In [1]: %matplotlib inline
        import numpy as np
        from sound import *
        import matplotlib.pyplot as plt

        fs = 44100
        f = 400
        s = np.sin(2 * np.pi * f *np.arange(0, 1., 1./fs))
```

Listen to it:

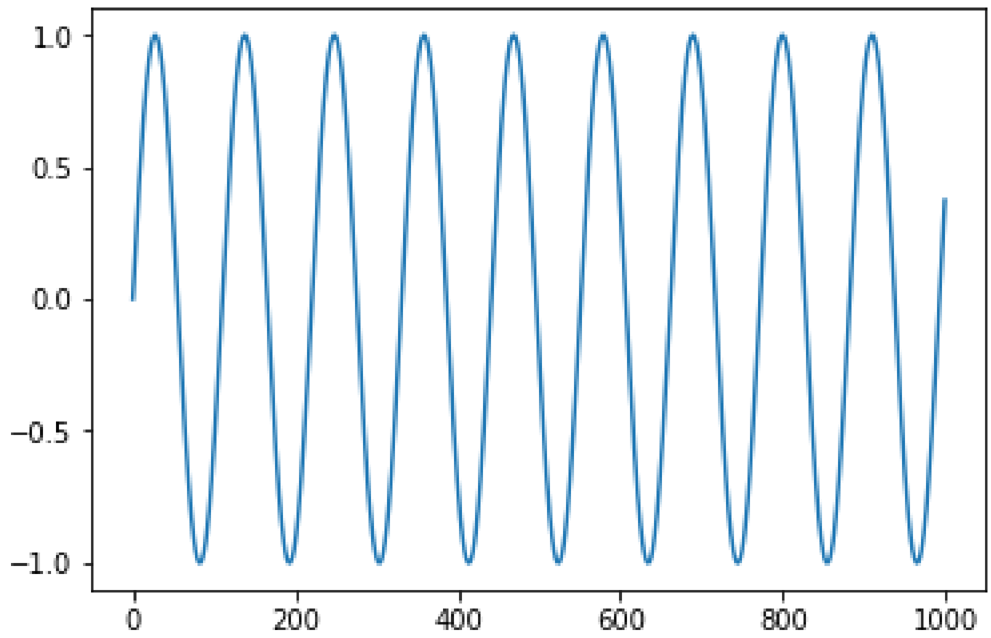
```
In [2]: sound(s*2**15, fs)
```

```
* done
```

Now plot the first 1000 samples:

```
In [3]: plt.plot(s[:1000])
```

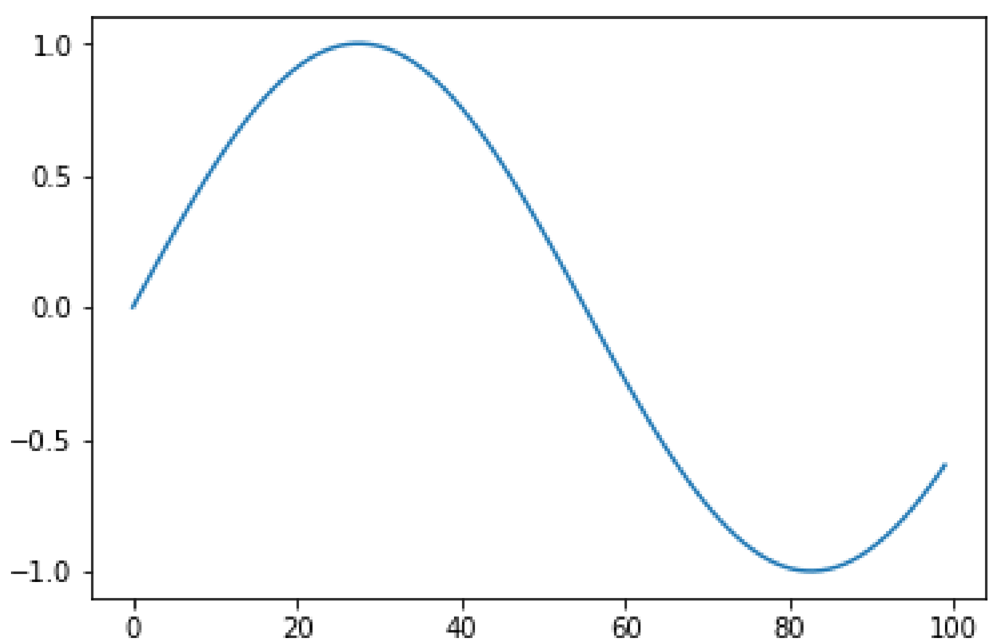
```
Out[3]: [<matplotlib.lines.Line2D at 0x83f1dd0>]
```




Now plot the first 1000 samples:

```
In [4]: plt.plot(s[:100])

Out[4]: [<matplotlib.lines.Line2D at 0x860c830>]
```



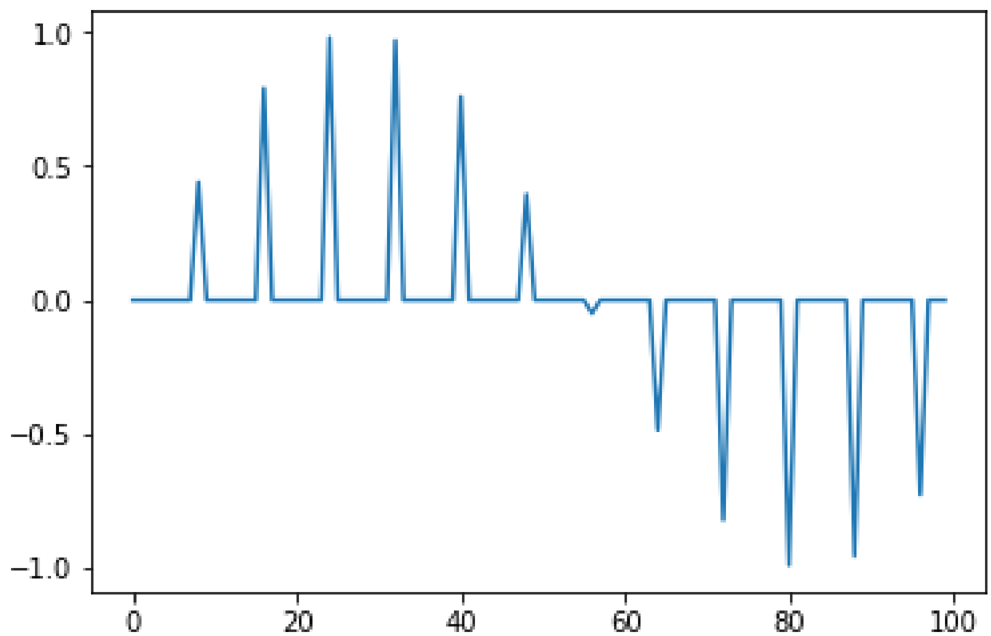
Now we can multiply this sine tone signal with a unit pulse train, with N=8. We use an indexing trick to get the desired result of only keeping every 8th sample and having zeros in between:

```
In [5]: sdu = np.zeros(s.shape)
        sdu[::8] = s[::8]
```

Now plot the result, the first 100 samples:

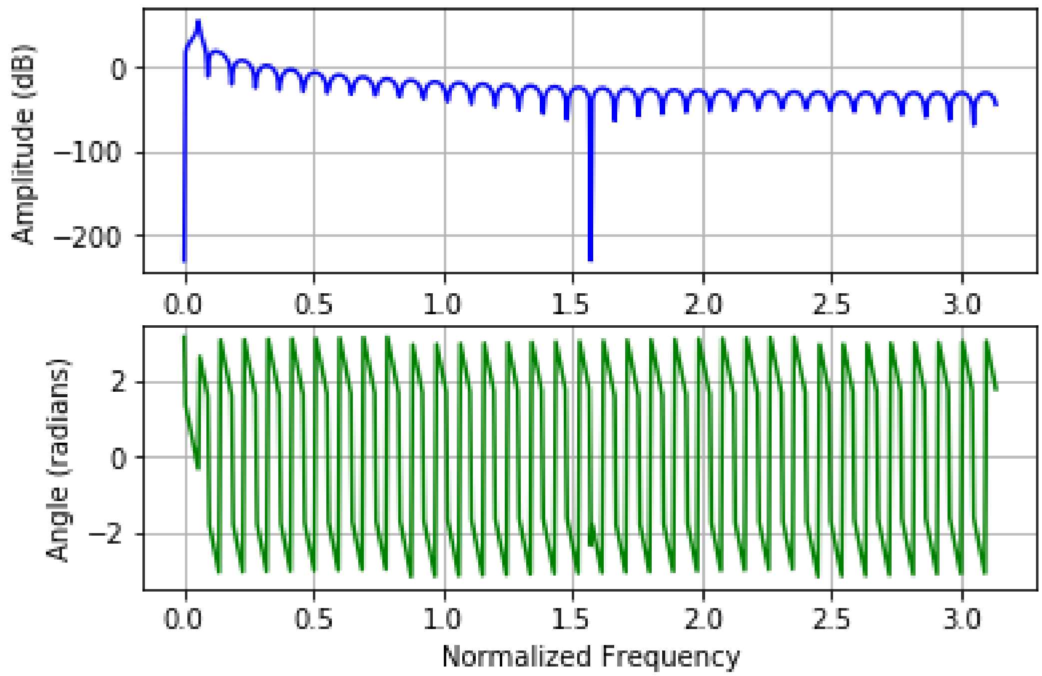
```
In [6]: plt.plot(sdu[:100])

Out[6]: [<matplotlib.lines.Line2D at 0x8763470>]
```



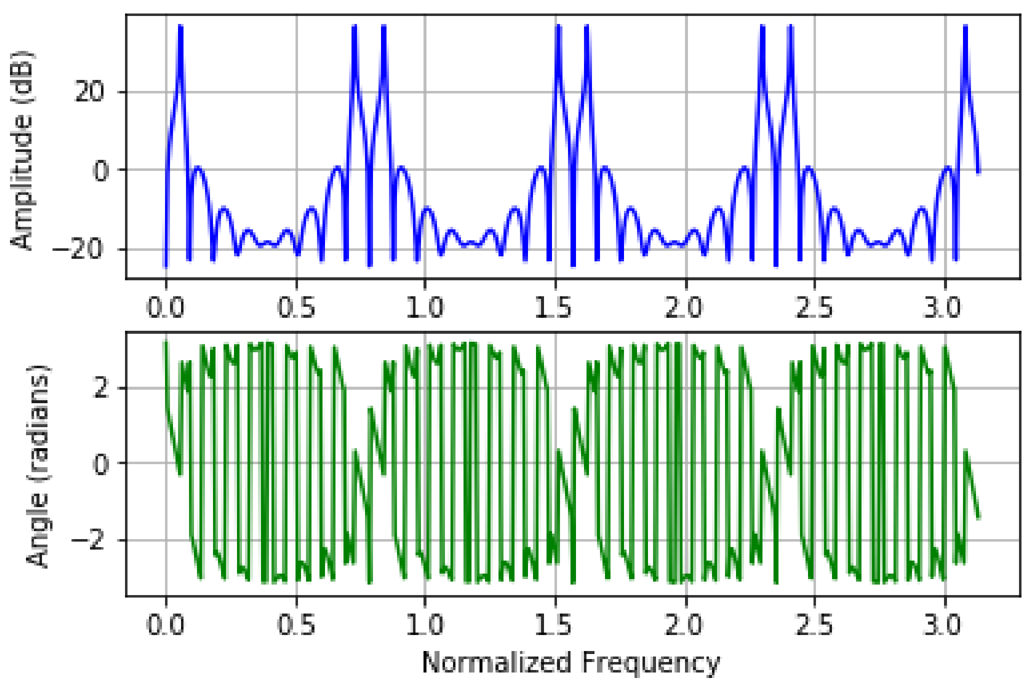

Now take a look at the spectrum of the original signal s:

```
In [7]: from freqz import *
        freqz(s)
```




Now we can compare this to our signal with the zeros, sdu:

```
In [8]: freqz(sdu)
```



Here we can see the original line of our 400 Hz tone, and now also the 7 new aliasing components. Observe that always 2 aliasing components are close together. This is because the original 400 Hz tone also has a spectral peak at the negative frequencies, at -400 Hz, or rather -0.018...

Now also listen to the signal with the zeros:

```
In [9]: sound(sdu*2**15, 44100)
```

```
* done
```

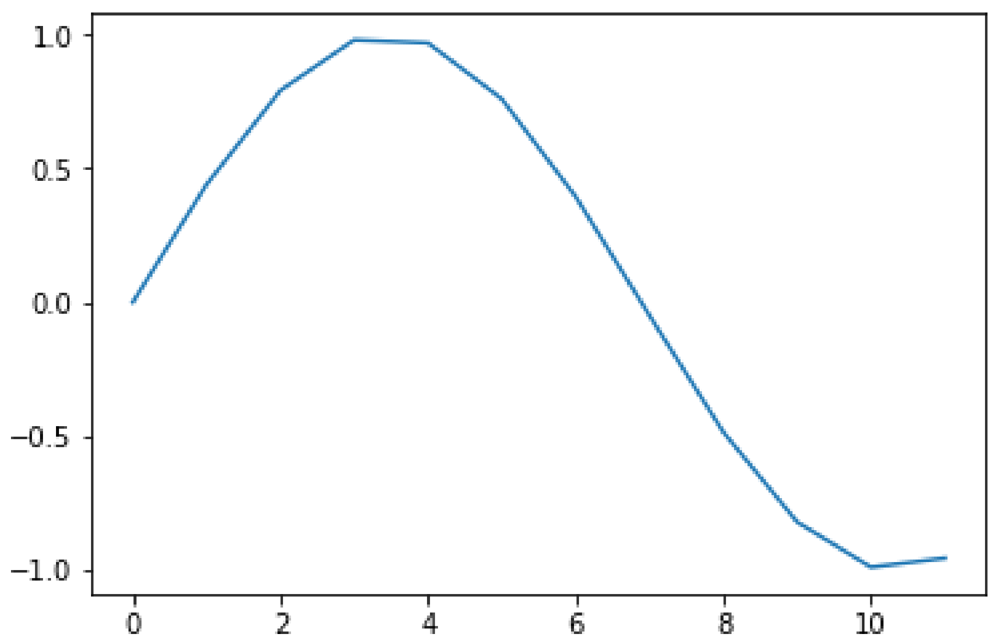
Here you can hear that it sounds quite different from the original, because of the string of aliasing components!

#### 0.1.1 Removing the zeros

The final step of downsampling is now to omit the zeros between the samples, to obtain the **lower sampling rate**. Let's call the signal without the zeros $y(m)$, where the time index $m$ denotes the lower sampling rate (as opposed to $n$ , which denotes the higher sampling rate).

```
In [10]: sd = np.zeros(sdu.shape[0]/8)
         sd = sdu[::8]
         plt.plot(sd[:100/8])
```

```
Out[10]: [<matplotlib.lines.Line2D at 0x126131d0>]
```




Observe that here we only have $100/8 \approx 12$ samples left.