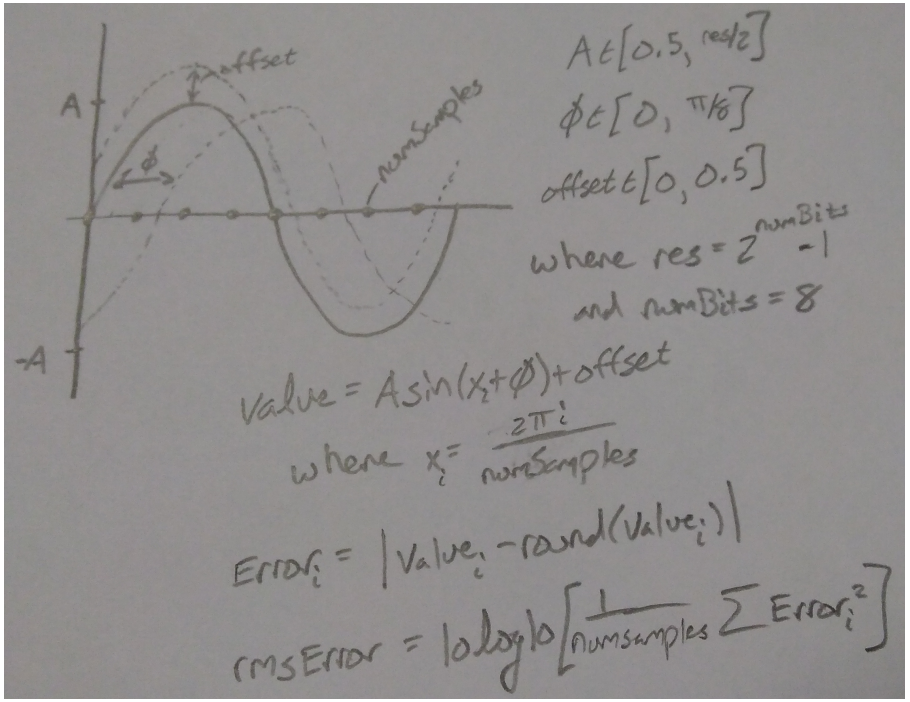
Quantization Error

Josh Wilkins
6/12/2017

In [20]:
```
# Imports↔
```

Sampling a signal causes rounding errors, produced by the difference in the actual value of the signal to the measurable value. The degree of this error is directly proportional to the number of bits (resolution) used to measure the signal's value and the number of points used to sample the signal. Besides increasing the resolution, the quantization error can also be reduced by altering the shape of the signal. Depicted below is a representation of this process:



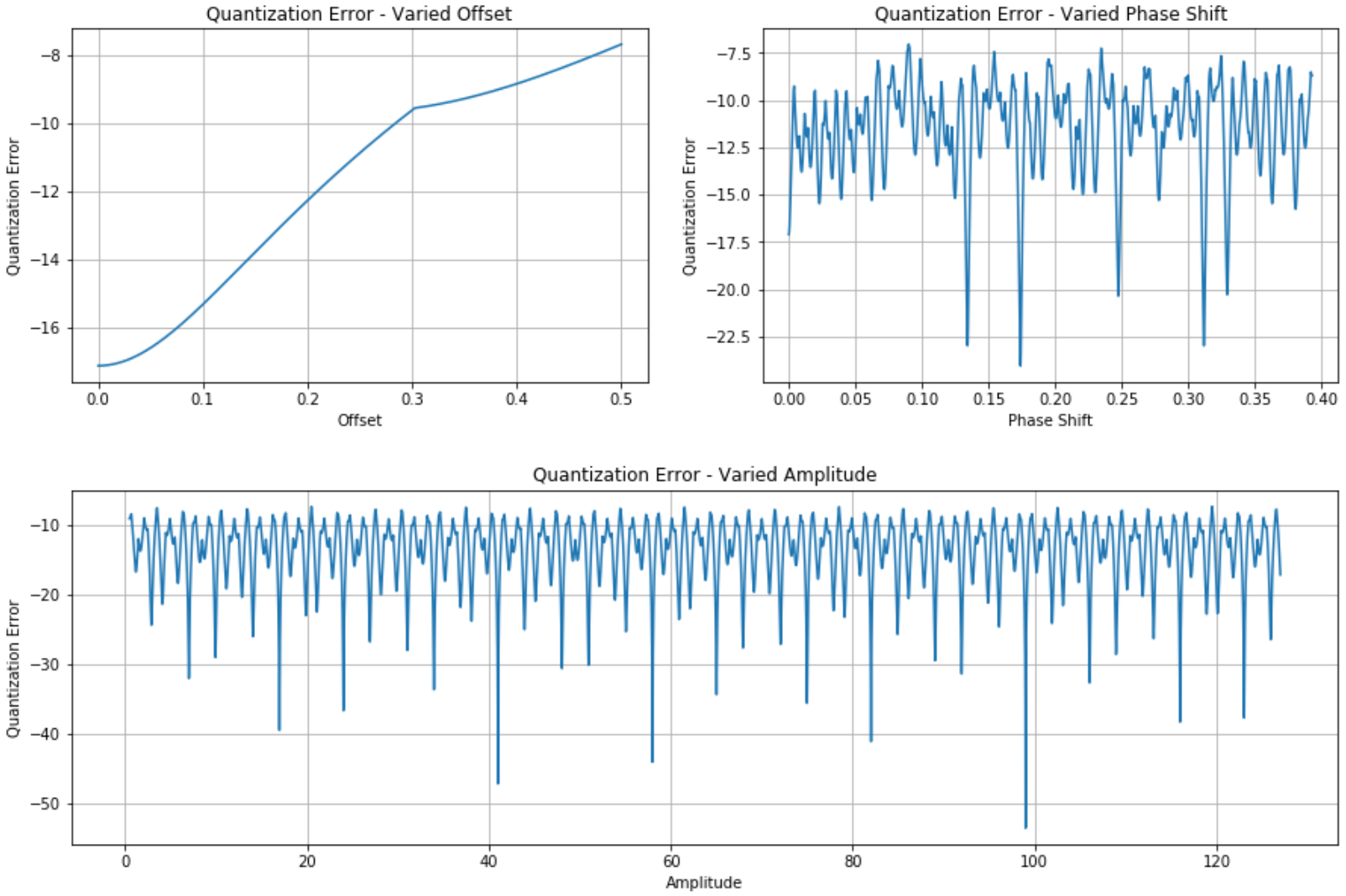
In [2]:
```
# Constants↔
```

In [3]:
```
def calcError(amp=resolution, offset=0, shift=0, numSteps=8):↔
```

Quantization Error

The three main adjustment parameters here are offset, phase shift, and amplitude. To get a general sense of the effect these parameters have on the quantization error, the three plots below were made, each holding two of the parameters constant while varying the third parameter.

In [24]:
```
# Quantization Error↔
```

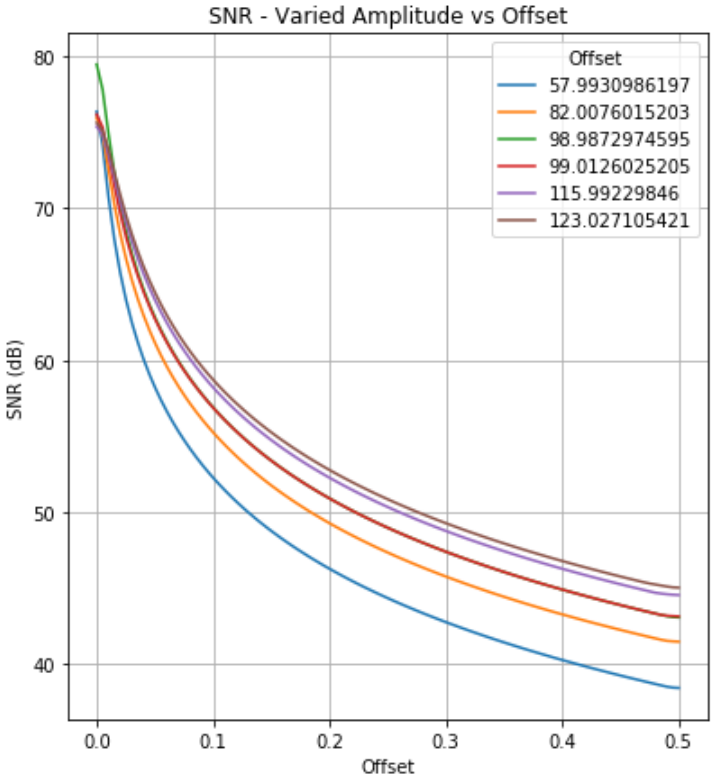


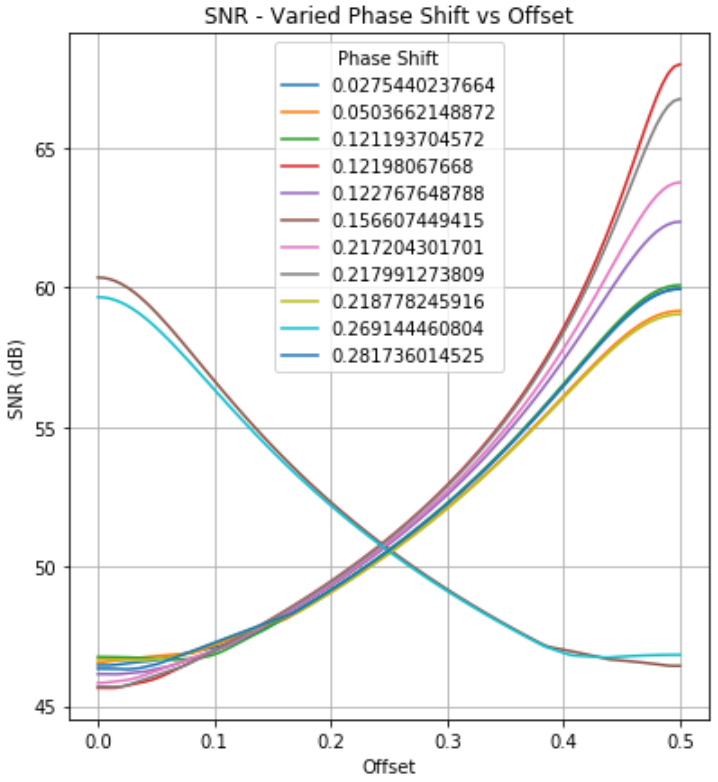
There is a clear relationship between offset and quantization error, but the other two parameters are probably pseudo-random.

Amplitude, Offset, and Phase Shift

Going a little bit further here, the following 6 plots illustrate the effect of altering these parameters with respect to each other. Meaning one of the parameters is held constant and the other two parameters are swept, then plotted against each other. Only the best resulting SNR plots are kept and graphed.

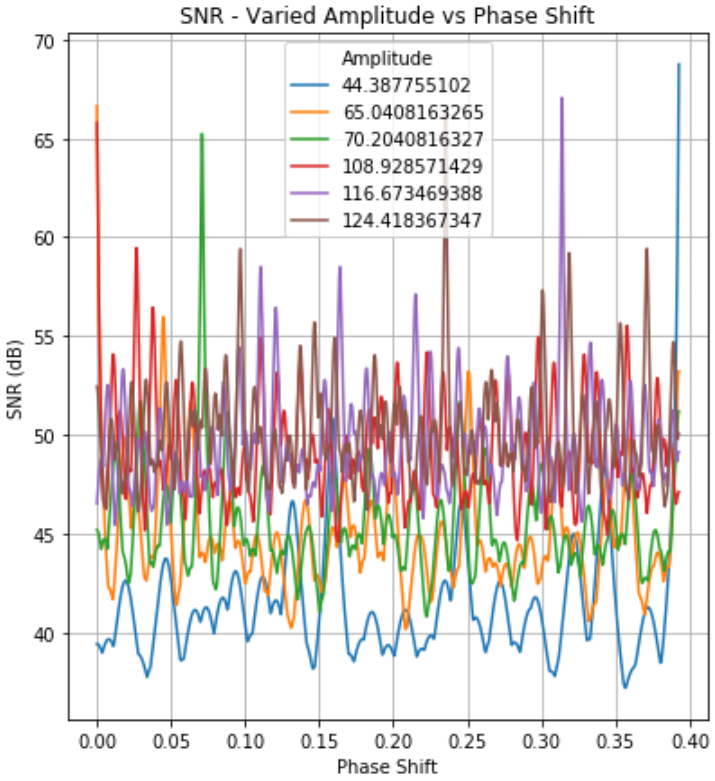
In [5]:
```
# Determining the best value for the offset↔
```

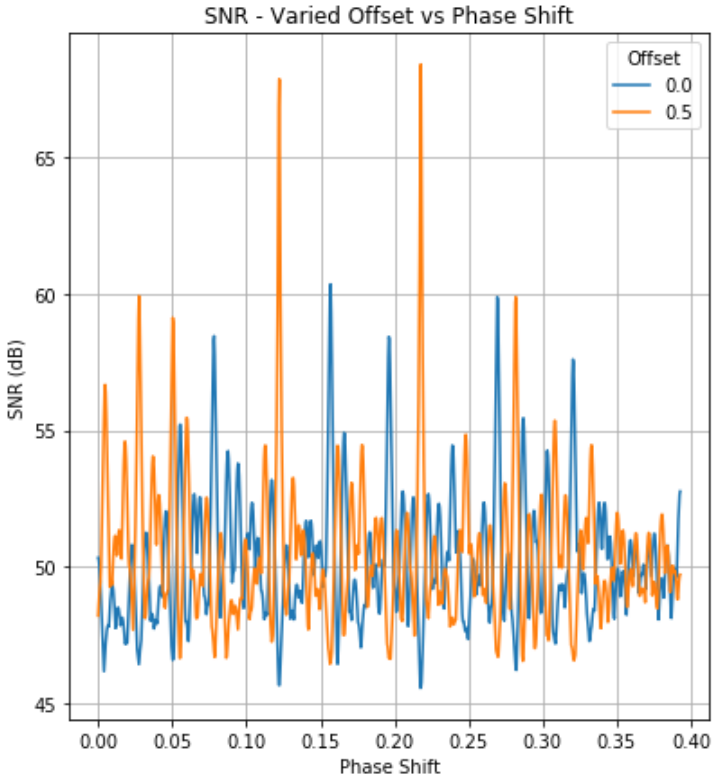




The first figure indicates that an offset of 0 produces the best results (not including phase shift)
The second figure shows that either an offset of 0 or 0.5 is good, but the deviation in the first suggests an offset of 0 is always better.

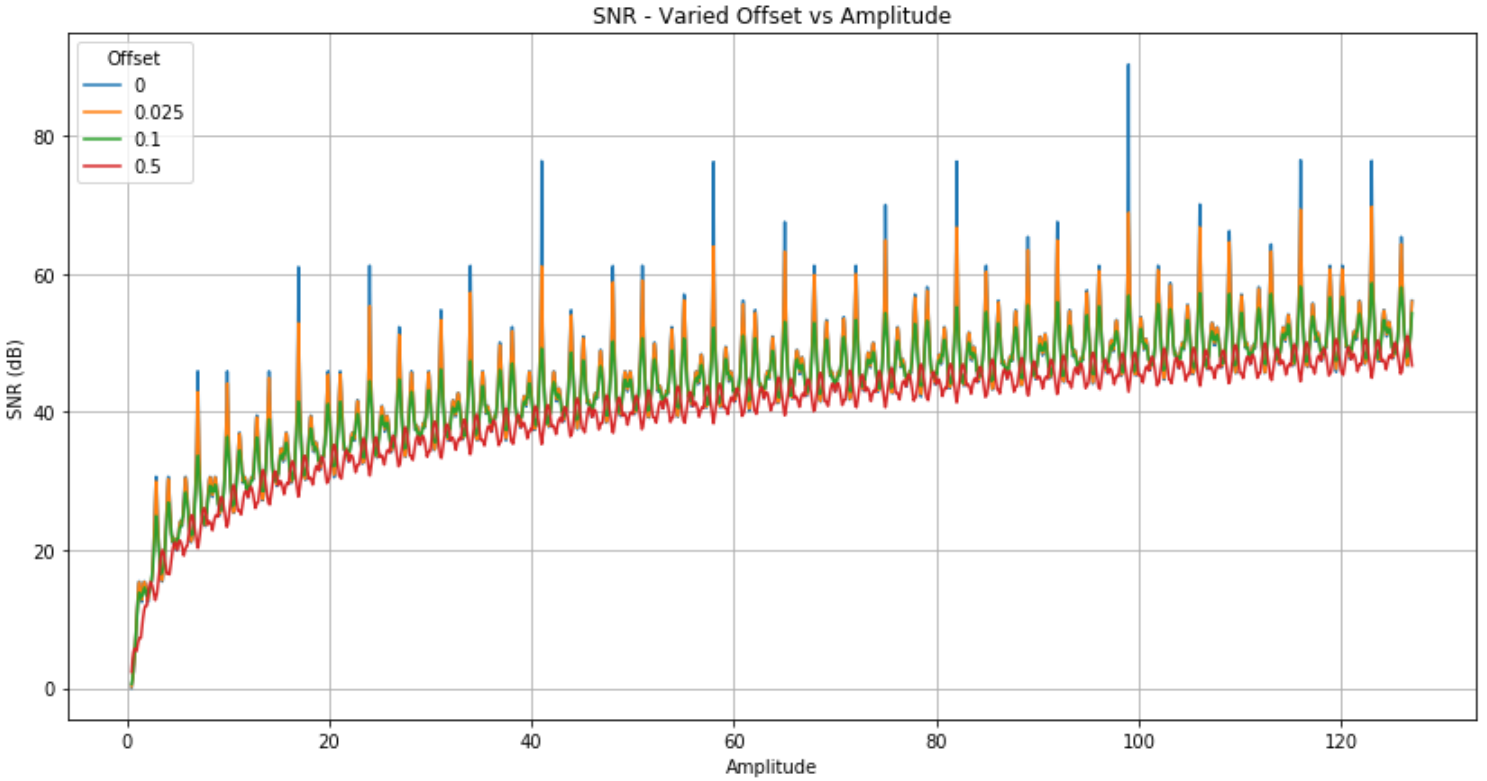
In [6]:
```
# Determining the best value for the phase shift↔
```





The first plot shows that, in general, the quantization error decreases with an amplitude increase, but the peak SNR of each amplitude is seemingly random. The second plot resulted in the validation that an offset of 0 or 0.5 produces the best results.

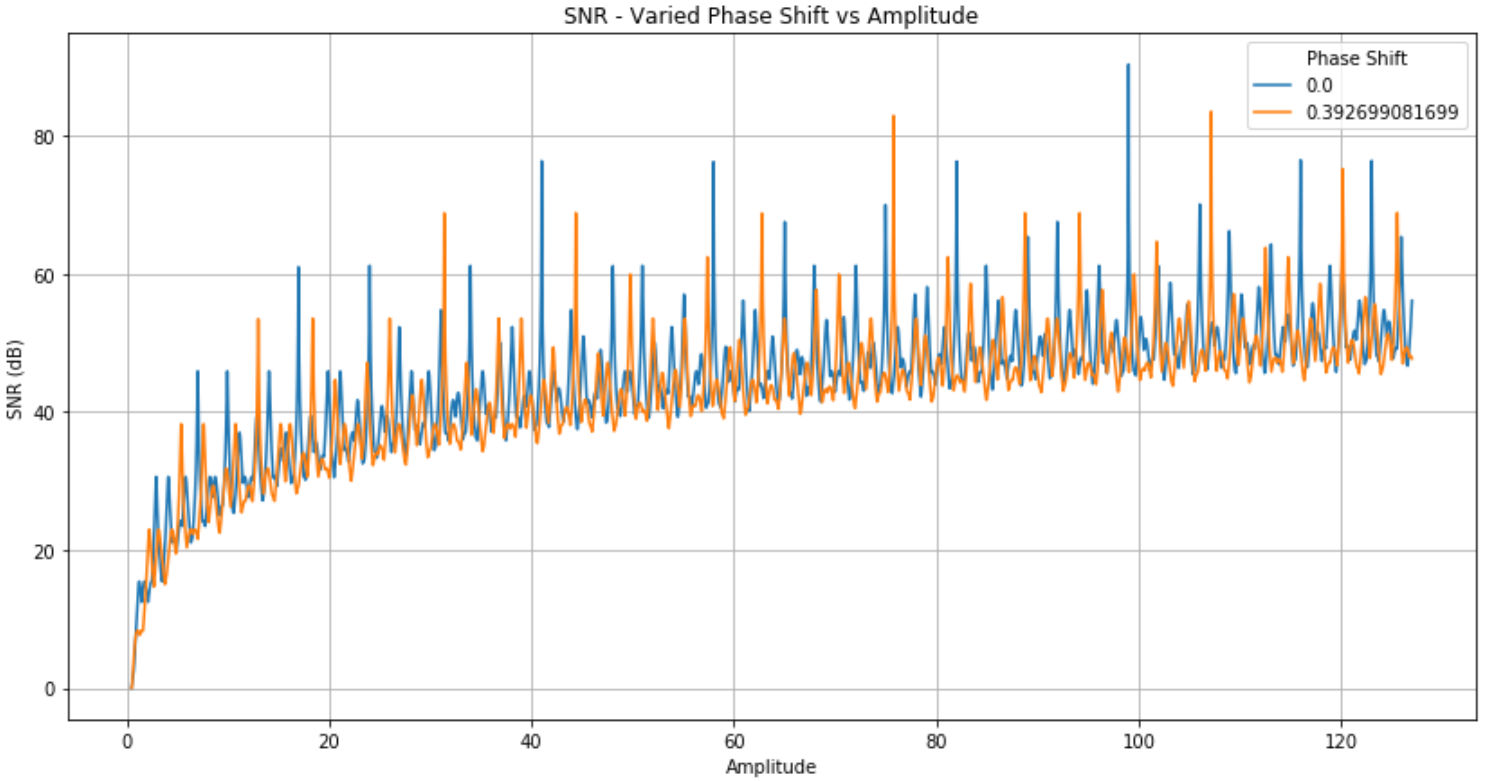
In [7]:
```
# Determining the best value for the Amplitude↔
```



The figure above shows that, as the offset approaches 0, the SNR peaks get larger. The deviation in these peaks is much greater then that of the other varied offset plots, illustrating that an offset of 0 is the best (not 0.5).

Furthermore, these peaks occur at the same amplitude, regardless of the offset. If you look close enough, you will see a pattern in the peaks; Every fourth major spike is substantially larger than the three before it. This suggests that an amplitude of ~99 is the best peak you can find, not considering phase shift.

In [8]:
```
# Phase Shift vs Amplitude↔
```

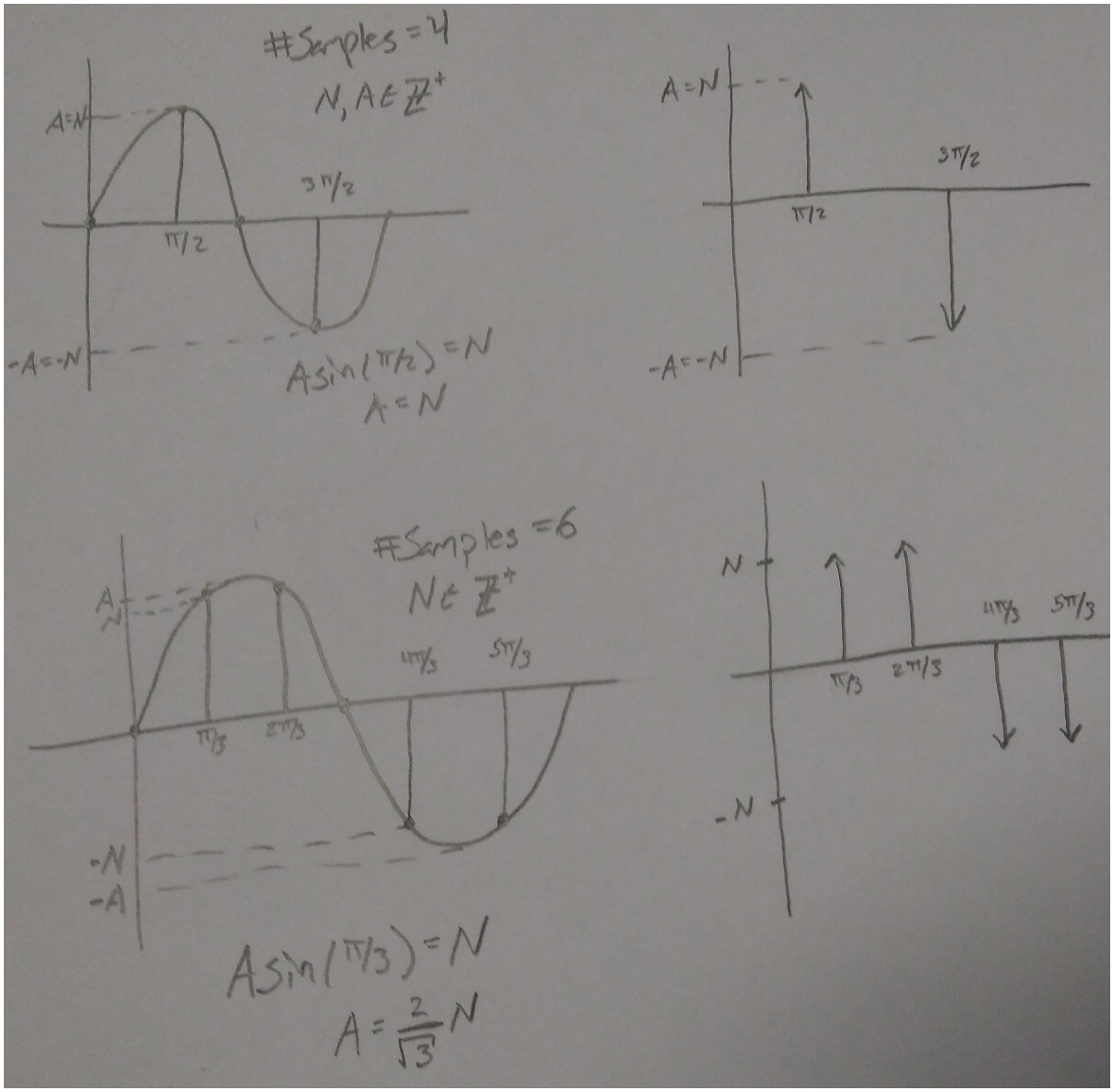


From the above figure, the same pattern is seen. Interestingly enough, a phase offset of 0 resulted in this pattern shifted to maximize the peak in SNR.

A couple of other things to consider are the number of bits used to represent the signal (bit resolution) and the number of sampling points to use. Since the varied offset vs amplitude plot showed the most variance in SNR value, it will be used as the benchmark to decide on the best number of sample points and number of bits to use.

Number of Samples

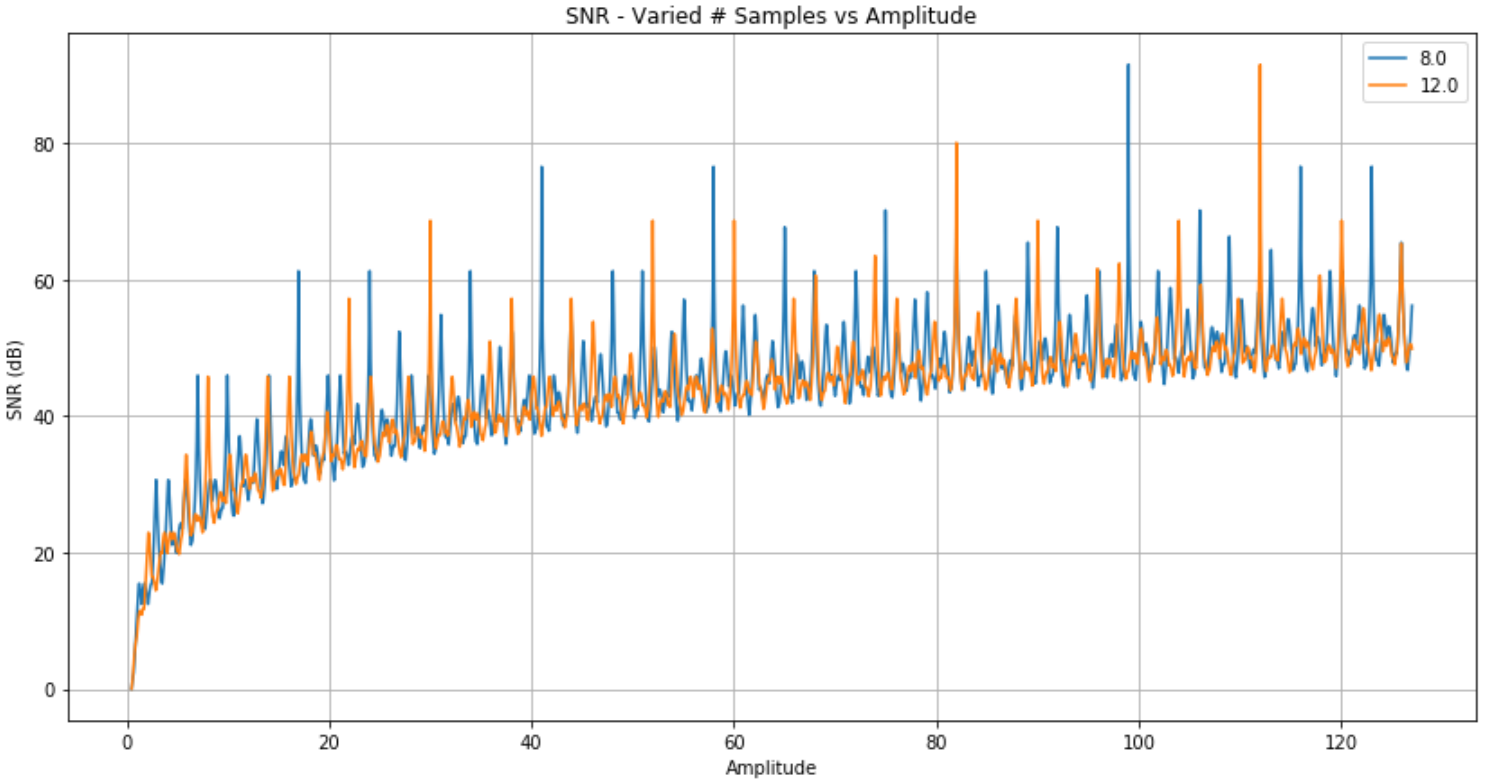
Some special cases are shown below in which the values of the signals at the sample points all land exactly on integers for given values of the amplitude. Sampling the signal at 4 points results in the values being integers whenever the amplitude is an integer. For 6 samples, the values are integers whenever the amplitude is a multiple of $\frac{2}{\sqrt{3}}$



The plot below illustrates the best results when increasing the number of samples up to 15 samples. Only the plots with SNR peaks above 90 dB were kept.

In [9]:
```
# Determining the best number of sample points↔
```

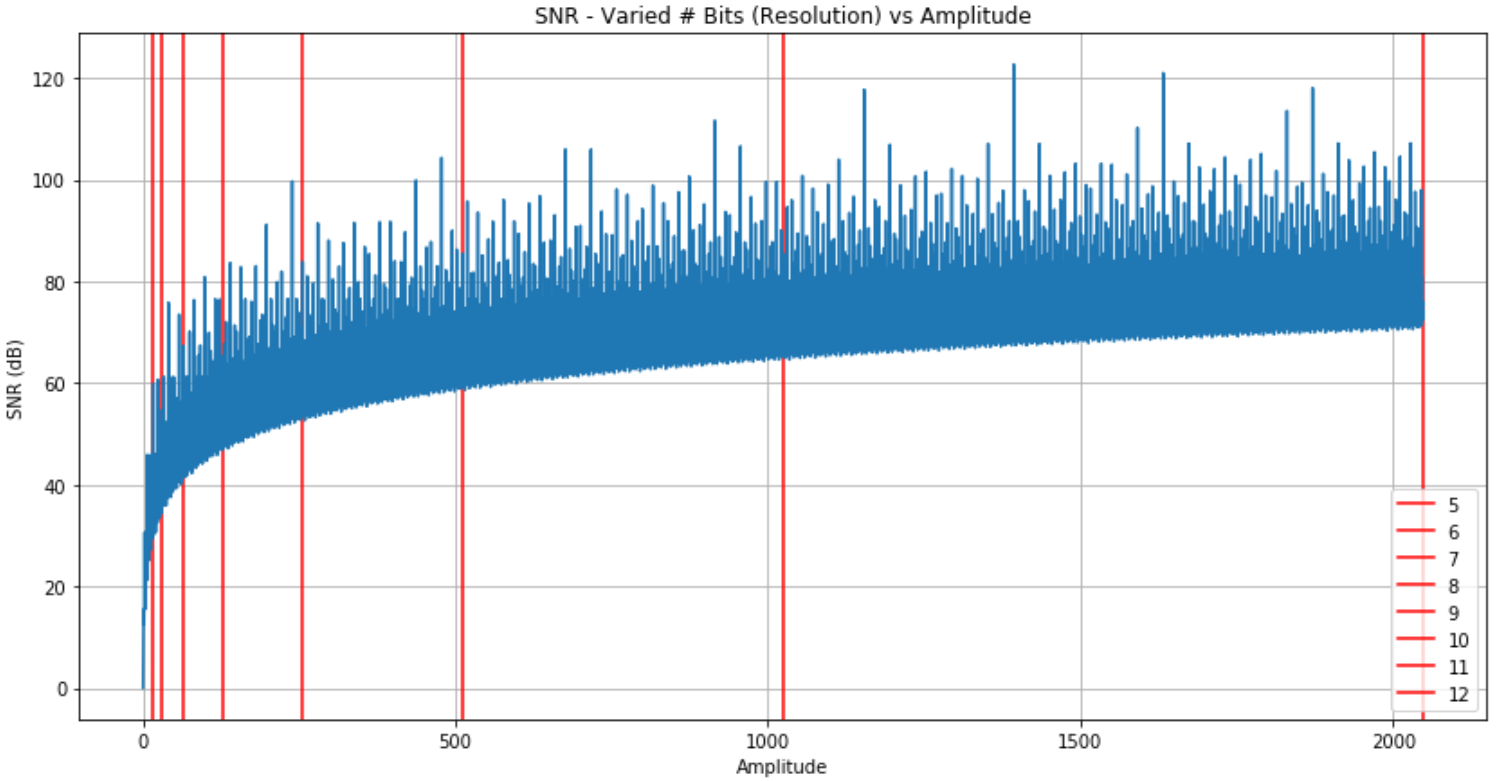
```
C:\ProgramData\Anaconda2\lib\site-packages\ipykernel_launcher.py:11: DeprecationWarning: object of type <type 'numpy.float64'> cannot be safely interpreted as an integer.
  # This is added back by InteractiveShellApp.init_path()
```



Resolution Bits

The next thing to consider is the number of bits used to represent the signal (amplitude resolution). The vertical red lines mark the end of the bit.

In [10]:
```
# Determining the best number of bits (resolution)↔
```

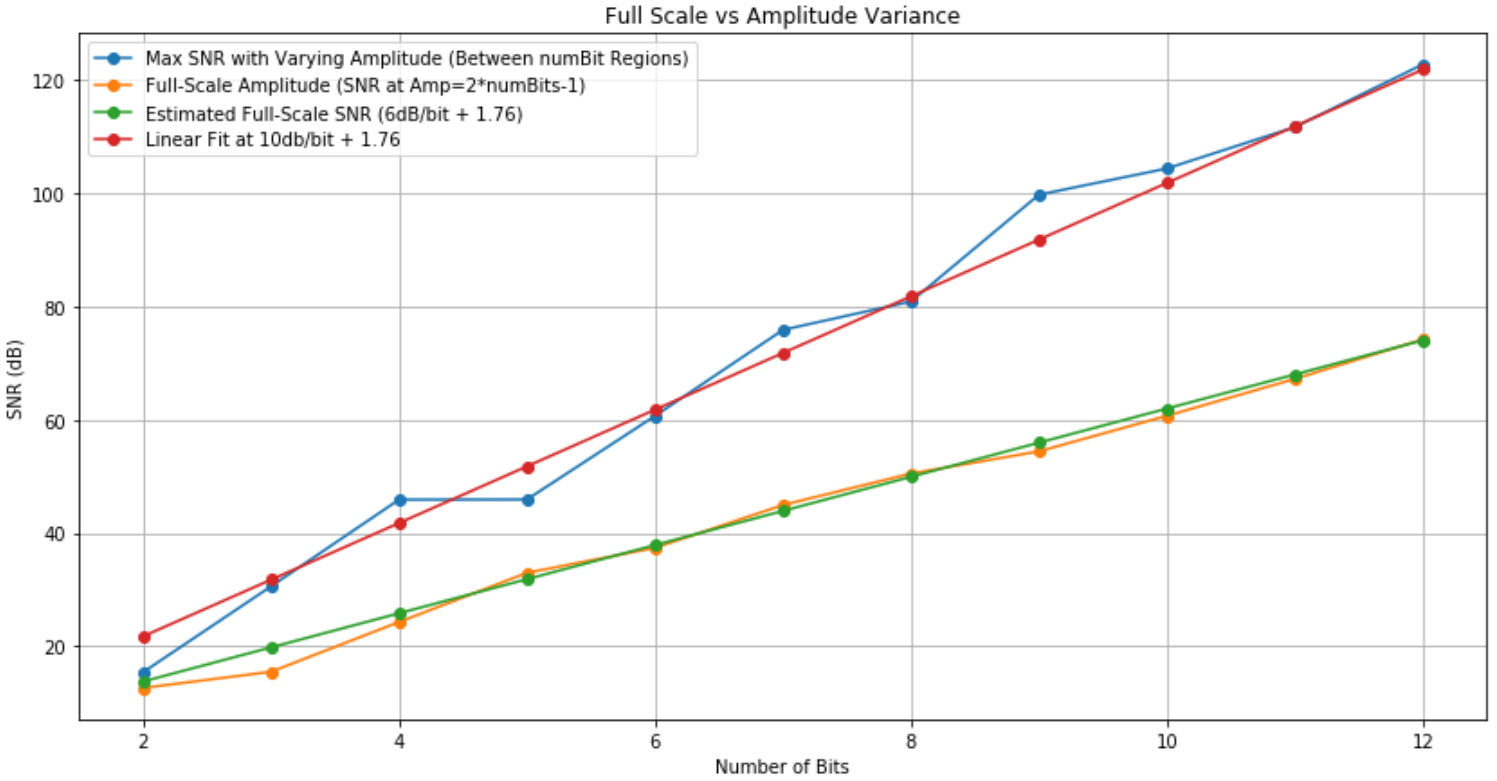


Note that it is sufficient to show only one plot, since each would be identical when varying the resolution, at least up until its max amplitude value of resolution/2.

Results

Quantization error is generally approximated by 6 dB/bit + 1.76 dB. This assumes the entire bit range is used. By not utilizing the entire range, the SNR is improved to about 10 dB/bit + 1.76 dB.

In [11]:
```
# Full Scale Amplitude vs Custom Amplitude SNR↔
```

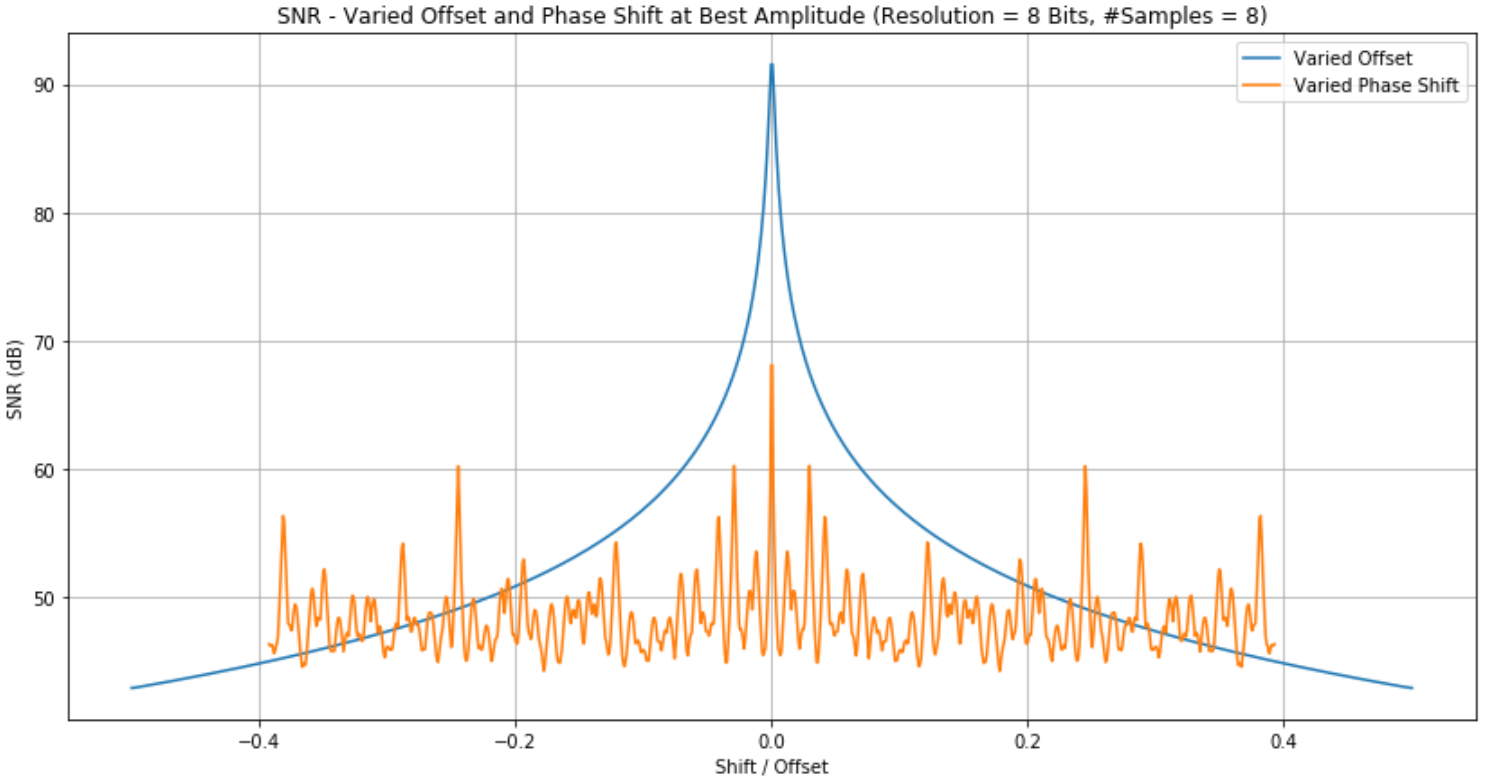


Being able to use a custom value for the amplitude greatly increases the SNR with respect to the quantization error. The linear fit of the points indicates an increase of 4 dB/bit! Unfortunately, using a LUT on a DDS for example, would use the full range of the number of bits it has.

Verifying Results

Using the results from above, it is clear that the amplitude parameter most greatly affects the resulting SNR. Thus, setting the amplitude to the highest peak at ~99, we can then verify that an offset of 0 and phase shift of 0 will produce the best resulting SNR.

In [12]:
```
# Varied Offset and Phase Shift at Best Amplitude↔
```

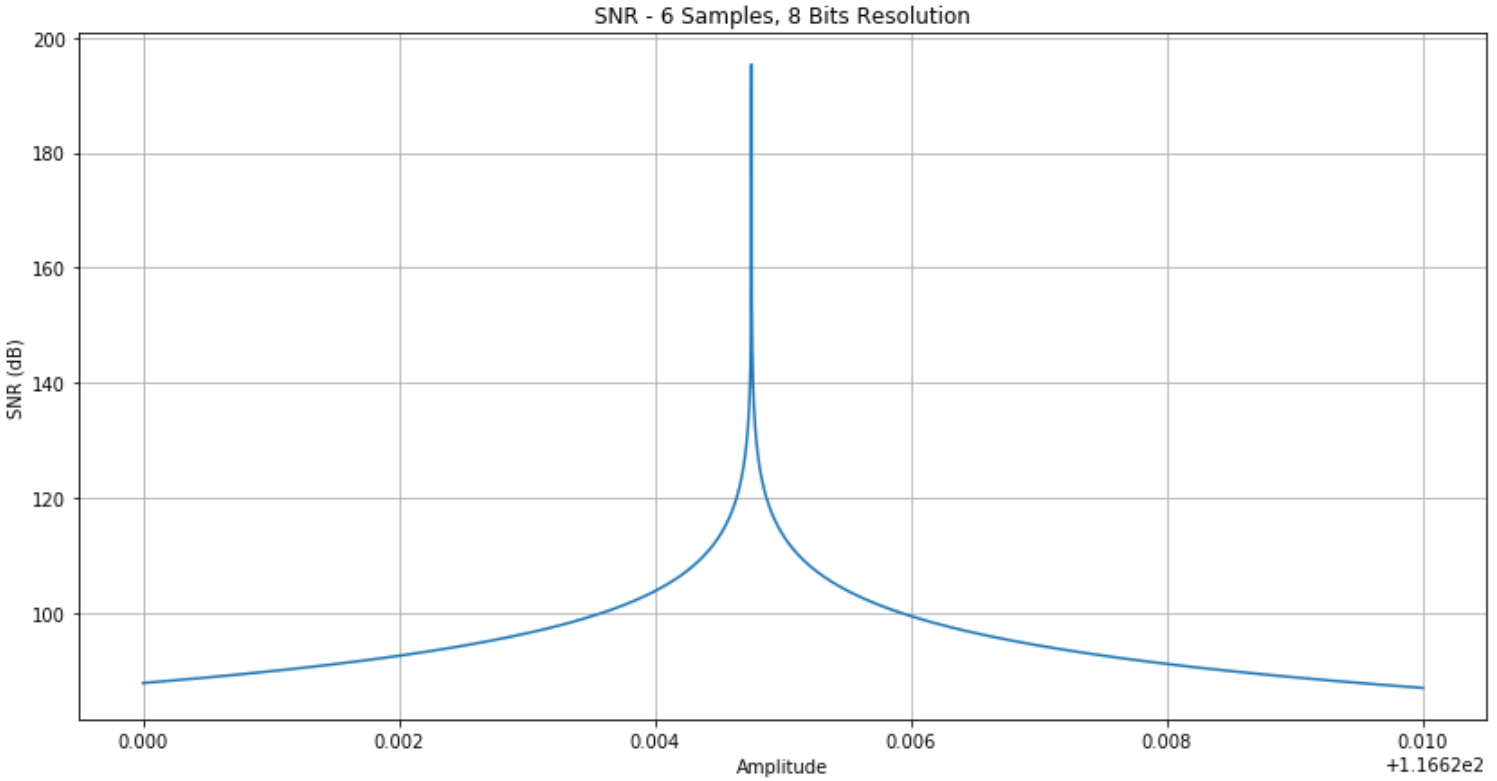


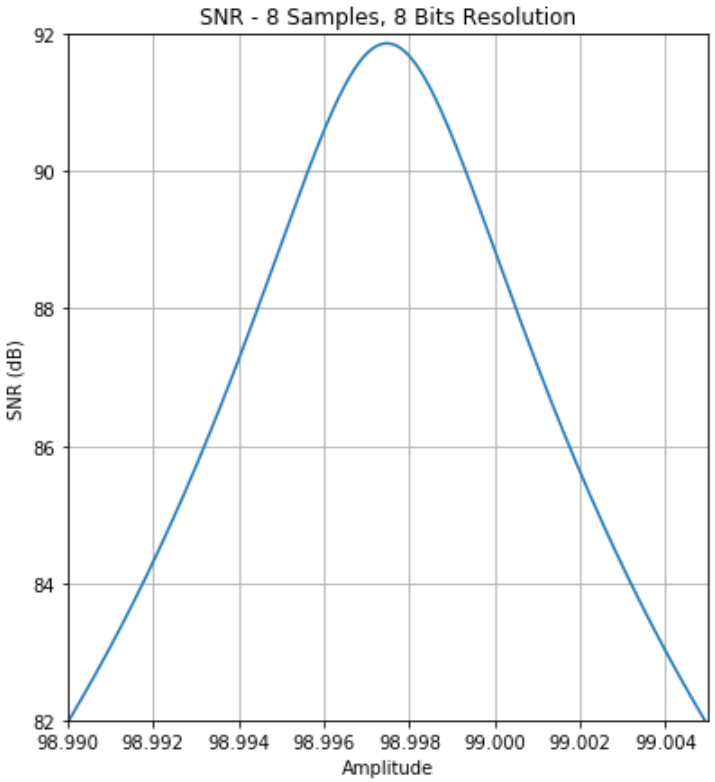
The above plot illustrates that a change in offset or a change in phase shift, both whose values are 0, will result in a decrease in SNR. It should be observed that the SNR is much more sensitive to the change in phase shift than it is a change in offset.

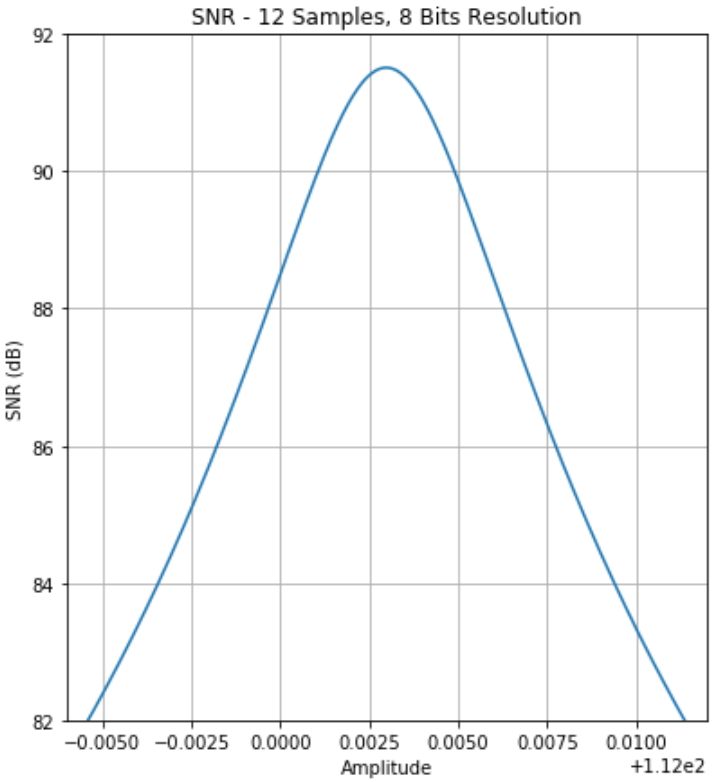
Magnified Regions

To illustrate the sensitivity of the SNR at its highest peak, a magnified plot of the region of interest is shown below. Again, with 6 samples, the SNR would approach infinity with the amplitude at multiples of $\frac{2}{\sqrt{3}}$

In [13]:
```
# Magnified Regions of Best SNR↔
```







The sensitivity of the amplitude in the last two plots are about the same; A change in amplitude of ±0.008 results in the SNR dropping by about 10dB.

Digital Representation

Sampling speed could present a restriction in implementation on an FPGA. An alternative would be using high speed logic gates, if the resulting circuit is relatively simple enough.

Using 8 bits of resolution and 8 sampling points, the rounded values of the signal (shown in the table below) are used to determine if a digital form of sampling is reasonable. ABC represent the sample point and the output is the resulting 8 bit binary value representing the value at the sample point.

In [14]:
```
# Rounded Signal Values at the numBits sample points:↔
```

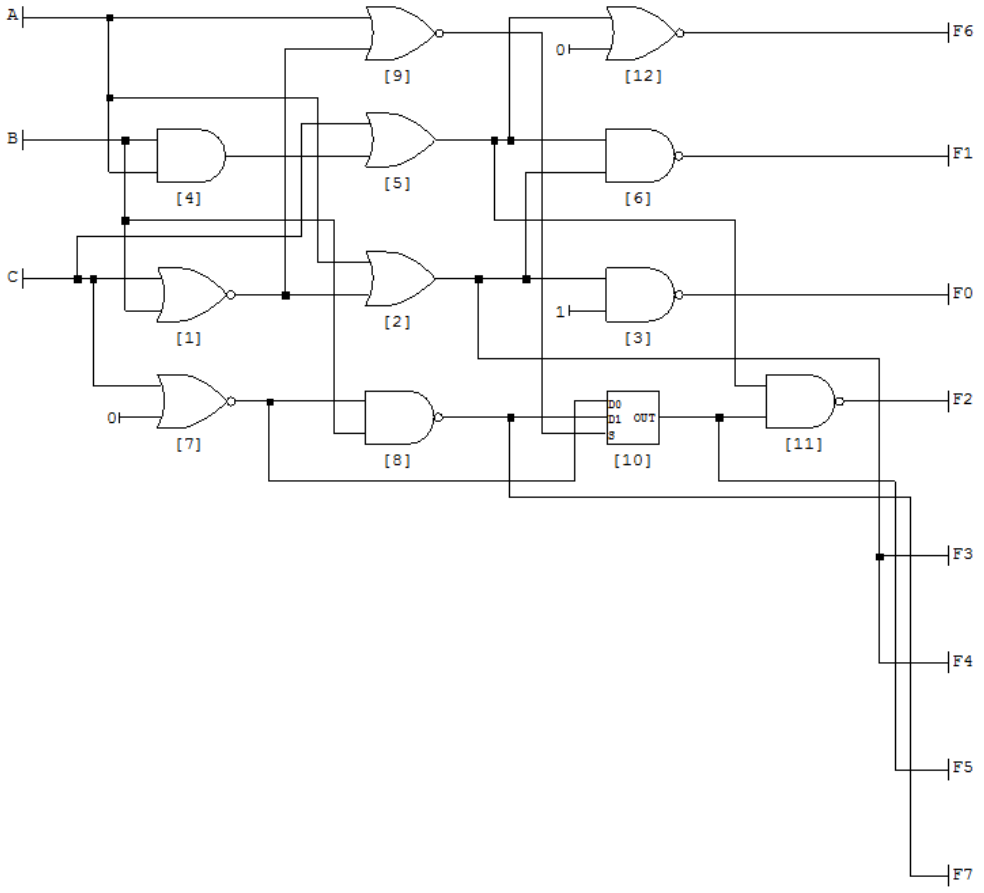
```
ABC | abcdefgh
----+---------
000 | 01111111
001 | 11000101
010 | 11100010
011 | 11000101
100 | 01111111
101 | 00111001
110 | 00011100
111 | 00111001
```

Using the Quine–McCluskey algorithm, the resulting set of digital logic could be used to represent the sampled signal.

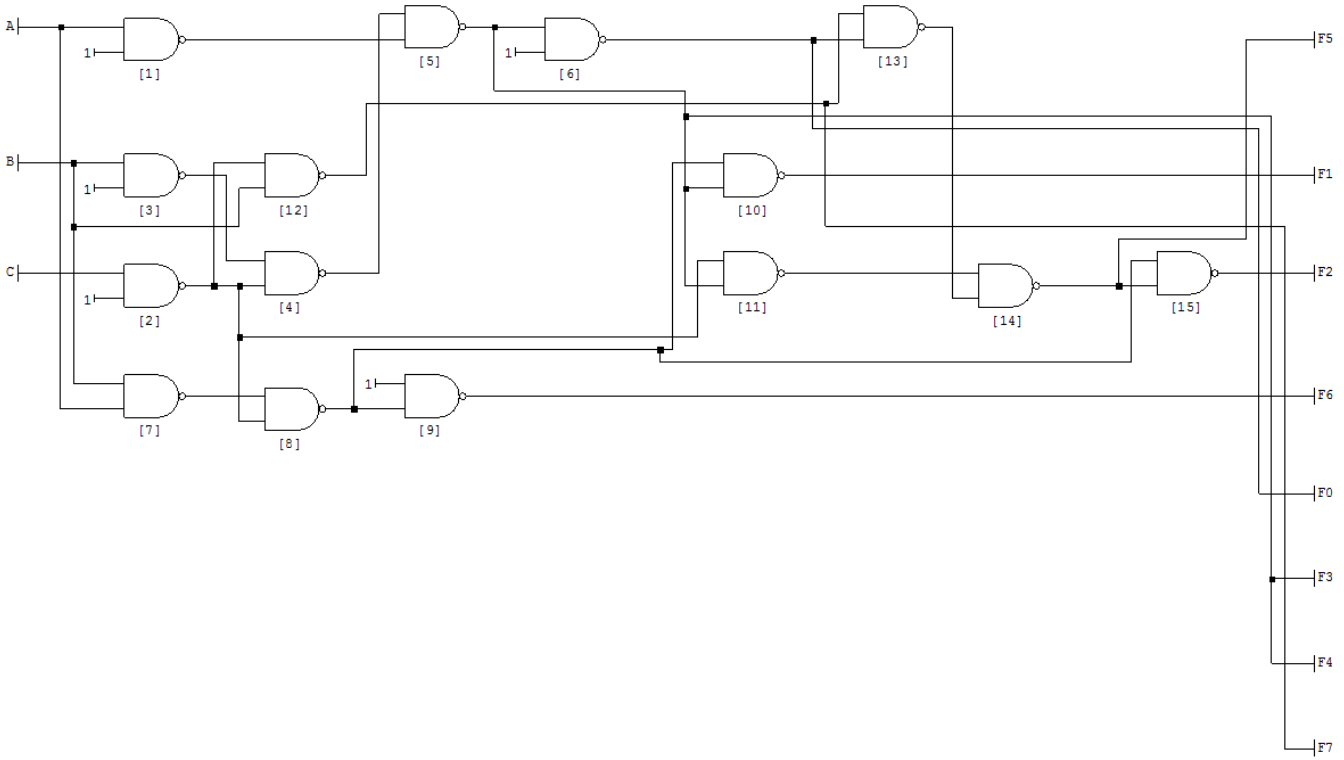
In [15]:
```
# Resulting digital logic to represent the rounded signal values↔
```

```
a = A'B + A'C
b = A' + B'C'
c = AC + A'C' + AB'
d = A + B'C'
e = A + B'C'
f = AC' + A'C + A'B'
g = A'C' + B'C'
h = B' + C
```

And by utilizing a freeware program called 'Logic Friday', the digital logic results from above was verified and the minimum amount of logic gates found as shown below



NAND only logic results in only 15 gates


A
B
C
1
[1]
[2]
[3]
[4]
[5]
[6]
[7]
[8]
[9]
[10]
[11]
[12]
[13]
[14]
[15]
F5
F1
F2
F6
F0
F3
F4
F7