3. Modulation

Demo: https://youtu.be/YMWrkLl6x1o

Adding noise

We can add random noise to our signal using the random.normal function. For example we can add random noise with a magnitude of 0.1 with:

```
noise = 0.1*np.random.normal(0, 1, len(xs))
ys = ys + noise
```

Now integrate noise with the sawtooth waveform:

```
import matplotlib.pyplot as plot
import numpy as np
from scipy import signal
xs = np.arange(0.0, 2, 0.01)
ys= signal.sawtooth(2 * np.pi * 5 * xs,width=0.5)

noise = 0.1*np.random.normal(0, 1, len(xs))
ys = ys + noise

plot.xlabel('time')
plot.ylabel('amplitude')
plot.plot(xs,ys)
plot.show()
```

Observe the output and the effect that varying the amplitude of the noise has on the signal. Plot the waveforms of varying noise amplitude:

Show that for a noise amplitude of 0.2 and a sine wave, the resulting plots are the same is in Figure 1. Vary the levels of the noise from 0 to 1 and observe the effect on the frequency response.

How might a filter be produced to recover the signal from the noise:

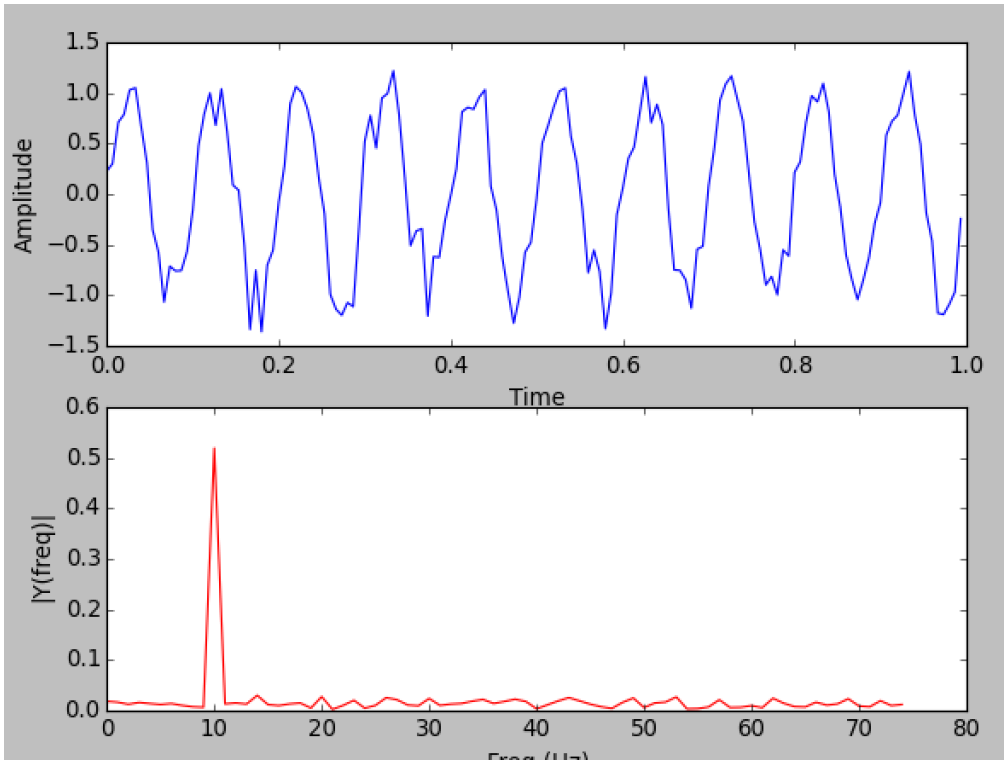

Figure 1: Waveform with noise

Check your answer here: http://asecuritysite.com/calculators/plot01

Amplitude modulation

Now we will implement amplitude modulation (AM), and has the form of:

$$f(t) = E(\sin(2\pi f_c t) + m.\sin(2\pi f_s t)))$$

where f_c is the carrier frequency, f_s is the signal frequency and m is the modulation index. In the following code we have a carrier frequency of 10Hz and a signal frequency of 1Hz. The amplitude of the signal is 0.3 of the amplitude of the carrier frequency:

```
import matplotlib.pyplot as plot
import numpy as np
from scipy import signal
t = np.arange(0.0, 2, 0.01)
freq=10
freqs=1

carrier= (1+m*np.sin(2 * np.pi * freqs * t))* np.sin(2 * np.pi
* freq * t)

plot.xlabel('time')
plot.ylabel('amplitude')
plot.plot(t,ys)
plot.show()
```

Plot the wave:

Now plot the resulting waveform when the amplitude of the carrier is 0.1, 0.5 and 1:

Now perform an FFT on the signal and how that the frequency components relate to the signals contained in the AM waveform:

```
import matplotlib.pyplot as plt
import numpy as np
from scipy import signal
Fs = 150.0;  # sampling rate
Ts = 1.0/Fs; # sampling interval

t = np.arange(0,2,Ts)

freq=10
freqs=1

carrier= np.sin(2 * np.pi * freq * t)
```

```
signal =0.3*np.sin(2 * np.pi * freqs * t)

y = carrier + signal
n = len(y) # length of the signal
k = np.arange(n)
T = n/Fs
frq = k/T # two sides frequency range
frq = frq[range(n/2)] # one side frequency range
Y = np.fft.fft(y)/n # fft computing and normalization
Y = Y[range(n/2)]

fig,myplot = plt.subplots(2, 1)
myplot[0].plot(t,y)
myplot[0].set_xlabel('Time')
myplot[0].set_ylabel('Amplitude')

myplot[1].plot(frq,abs(Y),'r') # plotting the spectrum
myplot[1].set_xlabel('Freq (Hz)')
myplot[1].set_ylabel('|Y(freq)|')

plt.show()
```

Observe the waveform when the amplitude of the signal is 20% larger than the carrier amplitude:

What effect does it have on the frequency content of the waveform:

What problems are likely if the signal is larger than the carrier amplitude:

Frequency modulation (FM)

With FM we have the waveform of:

$$f(t) = E\sin(2\pi f_c t + m.\sin(2\pi f_s t))$$

where f_c is the carrier frequency, f_s is the signal frequency and *m* is the modulation index. This can be integrated into the code with (note: we have moved the FFT code into a function):

```
import matplotlib.pyplot as plt
import numpy as np
from scipy import signal

Fs = 150.0;  # sampling rate
Ts = 1.0/Fs; # sampling interval

def calc_fft(y):
        n = len(y) # length of the signal
        k = np.arange(n)
        T = n/Fs
        frq = k/T # two sides frequency range
        frq = frq[range(n/2)] # one side frequency range
        Y = np.fft.fft(y)/n # fft computing and normalization
        Y = Y[range(n/2)]
        return(Y,frq)

t = np.arange(0,2,Ts)
freqc=10
freqs=2
m=0.4

y= np.sin(2 * np.pi * freqc * t + m*np.sin(2 * np.pi * freqs *
t))

Y,frq = calc_fft(y)

fig,myplot = plt.subplots(2, 1)
myplot[0].plot(t,y)
myplot[0].set_xlabel('Time')
myplot[0].set_ylabel('Amplitude')

myplot[1].plot(frq,abs(Y),'r') # plotting the spectrum
myplot[1].set_xlabel('Freq (Hz)')
myplot[1].set_ylabel('|Y(freq)|')

plt.show()
```

What is the value of the carrier frequency:

What is the value of the signal frequency:

If you vary the signal frequency from 1 Hz to 8 Hz, what do you observe from the frequency spectrum:

If you vary the modulation index from 0 to 1, what do you observe from the frequency spectrum:

Amplitude Shift Keying (ASK)

With ASK we can transmit binary values with one or more frequency amplitudes. In the simplest case we can use a single frequency to identify a '1', and no frequency for a '0'. In this case we have a bit pattern of "10110" and use a frequency of 10 Hz to represent a '1':

```
import matplotlib.pyplot as plt
import numpy as np
from scipy import signal

Fs = 150.0;  # sampling rate
Ts = 1.0/Fs; # sampling interval

def calc_fft(y):
        n = len(y) # length of the signal
        k = np.arange(n)
        T = n/Fs
        frq = k/T # two sides frequency range
        frq = frq[range(n/2)] # one side frequency range
        Y = np.fft.fft(y)/n # fft computing and normalization
        Y = Y[range(n/2)]
        return(Y,frq)

t = np.arange(0,2,Ts)
freqc=10
```

```
bit_arr = np.array([1, 0, 1, 1, 0])
samples_per_bit = 2*Fs/bit_arr.size
dd = np.repeat(bit_arr, samples_per_bit)

y= dd*np.sin(2 * np.pi * freqc * t)

Y,frq = calc_fft(y)

… previously defined code
```

From this show that the output is in the form:

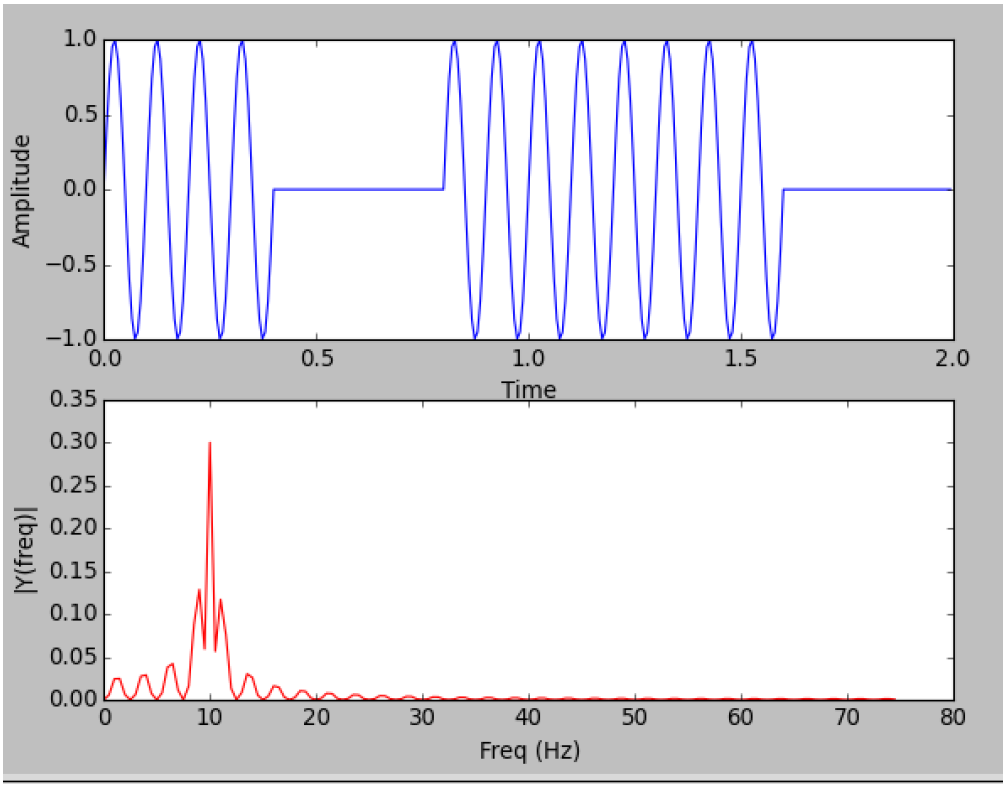

Figure 2: Waveform w

With the following bit patterns, observe the differences in the frequency spectrum:

11111

01010

If we are only transmitting one frequency, with different amplitudes, why do we see other frequencies in the spectrum:

We can also send more than one bit at a time. For example if we want to send two bits at a time we can have four amplitudes (00 – 0, 01 – 0.33, 10 – 0.67 and 11 – 1). Now modify the program to send '11 10 00 01 11':

```
bit_arr = np.array([1,0.67,0,0.33,1])
```

Plot the waveform and the spectrum:

Frequency Shift Keying (FSK)

https://youtu.be/YMWrkLl6x1oWith FSK we can transmit binary digits using one or more frequencies. For example if we have a single binary digit we can transmit with two frequencies, such as 5Hz and 15 Hz. In this case we send a binary value of "11010":

```
t = np.arange(0,2,Ts)
freqc=10

bit_arr = np.array([5,5,-5,5,-5])
samples_per_bit = 2*Fs/bit_arr.size
dd = np.repeat(bit_arr, samples_per_bit)

y= np.sin(2 * np.pi * (freqc + dd) * t)
```

Plot the waveform and the spectrum:

At which frequencies do we have a peak values:

Why do we have extra frequencies in the spectrum if we only transmit with two frequencies:

Now implement FSK which can send two bits at a time. Outline the additional code used for a transmitted binary value of '11 10 00 01 11':

How does the spectrum change with the addition of four frequencies:

Phase shift keying (PSK)

With PSK we change the phase of the transmitted frequency to identify the transmitted bit. In the following we have two bit shifts (180° and 0°):

```
t = np.arange(0,2,Ts)
freqc=10

bit_arr = np.array([180,180,0,180,0])
samples_per_bit = 2*Fs/bit_arr.size
dd = np.repeat(bit_arr, samples_per_bit)

y= np.sin(2 * np.pi * (freqc) * t+(np.pi*dd/180))
```

Plot the waveform and the spectrum:

Now implement PSK which can send two bits at a time. Outline the additional code used for a transmitted binary value of '11 10 00 01 11':

How does the spectrum change with the addition of four phases:

M-ary modulation

With *M*-ary modulation a change in amplitude, phase or frequency represents one of *M* possible signals. It is possible to have *M*-ary FSK, *M*-ary PSK and *M*-ary ASK modulation schemes. This is where the baud rate differs from the bit rate. The bit rate is the true measure of the rate of the line, whereas the baud rate only indicates the signalling element rate, which might be a half or a quarter of the bit rate.

Figure 3 shows the 16 combinations of phase and amplitude with 12 different phase shifts and four different amplitudes. Each transmission is known as a symbol, thus each transmitted symbol contains 4 bits.

Implement a Python program to transmit using the method outlined in Figure 3, and show a demonstration for a transmission of "0001 1010 1111 01010 0001".

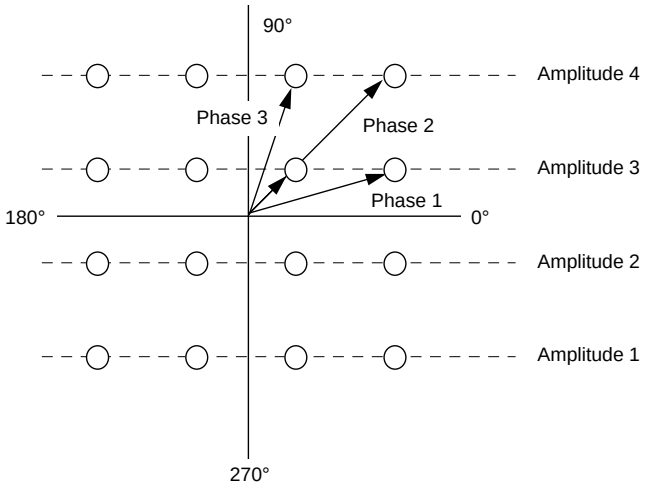

Figure 3:

Project

A. Implement a program using the Python program at:

http://asecuritysite.com/calculators/plot01

If you name your file "myplot1.py" then you can run with:

```
python mypoy1.py myfile.svg 0.1 sawtooth 10
```

which outputs to a file "myfile.svg", and has 0.1 noise amplitude, and is a 10Hz saw-tooth wave.

B. Implement a program using the Python program at:

http://asecuritysite.com/calculators/plot02

If you name your file "myplot2.py" then you can run with:

```
python mypoy.py myfile2.svg 0.1 am 10 5
```

which outputs to a file "myfile.svg", and has 0.1 is m, and we have AM with a carrier frequency of 10Hz and a signal frequency of 5Hz.

C. Implement a program using the Python program at:

http://asecuritysite.com/calculators/plot03

If you name your file "myplot3.py" then you can run with:

```
python myploy.py myfile2.svg ask 10
```

which outputs to a file “myfile.svg”, and uses ASK with a frequency of 10Hz.