

# 2018 Lab 4 prelab. Introduction to Digital Communications

In this pre-lab you will be introduced to several modes of digital communications. We have written the solutions for you, however, you are more than welcome to download the empty notebook and implement the solutions yourself.

## Analog Modulation

In analog communications we encode continuous valued signals on top of a carrier frequency. Information can be encoded in the amplitude, for example: amplitude modulation (AM) or single sideband (SSB). It can also be encoded in the phase, for example: phase modulation or frequency modulation or encoded in both phase and amplitude. One of the issues with analog modulation is that you can never get the signal you are transmitting exactly recovered. It will always be contaminated by system imperfections, noise and other deviations. At the same time, analog communication offers simple implementations as well as robustness because signals often degrade gracefully and information can still be extracted from them even with severe interferences and low signal to noise ratio.

## Digital Modulation

In contrast, in digital communications we transmit quantized digital data. Because the information is digital, this form of communications offers the advantage of a perfect reproduction of the transmitted signal. Much like analog communications, digital data can be encoded in many different ways using phase, amplitude and frequency. It can be encoded using a binary stream in which '0's and '1's are transmitted in each time interval. It can also be encoded with M-ary symbols, in which higher order symbols (for example 4-ary 0,1,2,3) are transmitted in each time interval.

```
In [1]: # Import functions and libraries
import numpy as np
import matplotlib.pyplot as plt
from numpy import r_, exp, cos, sin, pi, zeros, ones, hanning, sqrt,
log, floor, reshape, mean
from scipy import signal
from numpy.fft import fft
%matplotlib inline
```

```
In [2]: # function to compute average power spectrum
def avgPS( x, N=256, fs=1):
    M = int(len(x)//N)
    x_ = reshape(x[:M*N], (M,N)) * np.hamming(N)[None,:]
    X = np.fft.fftshift(np.fft.fft(x_,axis=1),axes=1)
    return r_[-N/2.0:N/2.0]/N*fs, mean(abs(X**2),axis=0)

def sg_plot( t_range, f_range, y, dbf = 60) :
```

```

eps = 1e-3

# find maximum
y_max = abs(y).max()
# compute 20*log magnitude, scaled to the max
y_log = 20.0 * np.log10( abs( y ) / y_max + eps )

fig=plt.figure(figsize=(15,6))

plt.imshow( np.flipud( 64.0*(y_log + dbf)/dbf ), extent= t_range
+ f_range ,cmap=plt.cm.gray, aspect='auto')
plt.xlabel('Time, s')
plt.ylabel('Frequency, Hz')
plt.tight_layout()

def myspectrogram_hann_ovlp(x, m, fs, fc,dbf = 60):
    # Plot the spectrogram of x.
    # First take the original signal x and split it into blocks of length m
    # This corresponds to using a rectangular window %

    isreal_bool = np.isreal(x).all()

    # pad x up to a multiple of m
    lx = len(x);
    nt = (lx + m - 1) // m
    x = np.append(x,zeros(-lx+nt*m))
    x = x.reshape((m//2,nt*2), order='F')
    x = np.concatenate((x,x),axis=0)
    x = x.reshape((m*nt*2,1),order='F')
    x = x[r_[m//2:len(x),ones(m//2)*(len(x)-1)].astype(int)].reshape
    ((m,nt*2),order='F')

    xmw = x * hanning(m)[: ,None];

    # frequency index
    t_range = [0.0, lx / fs]

    if isreal_bool:
        f_range = [ fc, fs / 2.0 + fc]
        xmf = np.fft.fft(xmw,len(xmw),axis=0)
        sg_plot(t_range, f_range, xmf[0:m//2,:],dbf=dbf)
        print(1)
    else:
        f_range = [-fs / 2.0 + fc, fs / 2.0 + fc]
        xmf = np.fft.fftshift( np.fft.fft( xmw ,len(xmw),axis=0), axis=0 )
        sg_plot(t_range, f_range, xmf,dbf = dbf)

    return t_range, f_range, xmf

```

## Example of Different Digital Modulations:

In the following sections we will consider signals with a symbol-rate of 300 bauds (300b/s) with a carrier frequency of 1800Hz and a sampling rate of 44100. Each bit will correspond to  $44100/300 = 147$  samples. The carrier frequency was chosen such that there are exactly 6 cycles in a bit -- this is useful for the phase-based modulation schemes where we would like to have a continuous phase.

```
In [3]: fs = 44100  # sampling rate
        baud = 300  # symbol rate
        Nbits = 256 # number of bits
        Ns = int(fs//baud) # samples per symbol
        N = int(Nbits * Ns) # number of samples
        f0 = 1800 # center frequency

        Ndisp = 16 ; #number of bits to display
        np.random.seed(seed=1)
        bits = np.random.randn(Nbits,1) > 0
```

### Amplitude-Shift-Keying (ASK) or On-Off Keying (OOK)

ASK/OOK encodes digital data in the amplitude of the carrier,  $M(t) \cos(2\pi f_0 t)$  where  $M(t) = \{0 \text{ or } 1\}$  for binary data. We saw such an example in the ADS-B mode S lab previously. M-ary digital data can also be encoded in the amplitude by dividing the amplitude into more segments. For example  $M(r) = \{0 \text{ or } 0.33 \text{ or } 0.66 \text{ or } 1.0\}$  for a 4-ary symbol.

Here's an example of an OOK signal:

```
In [4]: # Generate bits

# create an impulsive signal
message = np.zeros(N)
message[::Ns] = bits.ravel()

# convolve with a rect
h = np.ones(Ns)
message = signal.fftconvolve(message,h,mode='full')

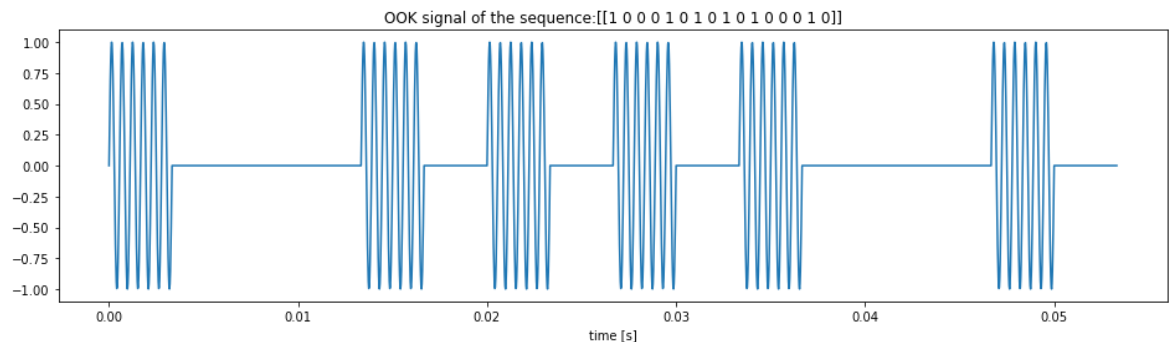
# modulate
t = r_[0.0:len(message)]/fs
OOK = message*sin(2*pi*f0*t)

fig = plt.figure(figsize = (16,4))
plt.plot(t[:Ndisp*Ns],OOK[:Ndisp*Ns])
plt.xlabel('time [s]')
plt.title('OOK signal of the sequence:'+ np.array2string(np.transpose(
np.uint8(bits[:Ndisp]))));
```

/home/sherman-tay/anaconda3/lib/python3.6/site-packages/scipy/signal/signaltools.py:375: FutureWarning: Conversion of the second argument of issubdtype from `complex` to `np.complexfloating` is deprecated. In future, it will be treated as `np.complex128 == np.dtype(complex).type`.

complex\_result = (np.issubdtype(in1.dtype, complex) or  
/home/sherman-tay/anaconda3/lib/python3.6/site-packages/scipy/signal/signaltools.py:376: FutureWarning: Conversion of the second argument of issubdtype from `complex` to `np.complexfloating` is deprecated. In future, it will be treated as `np.complex128 == np.dtype(complex).type`.

np.issubdtype(in2.dtype, complex))



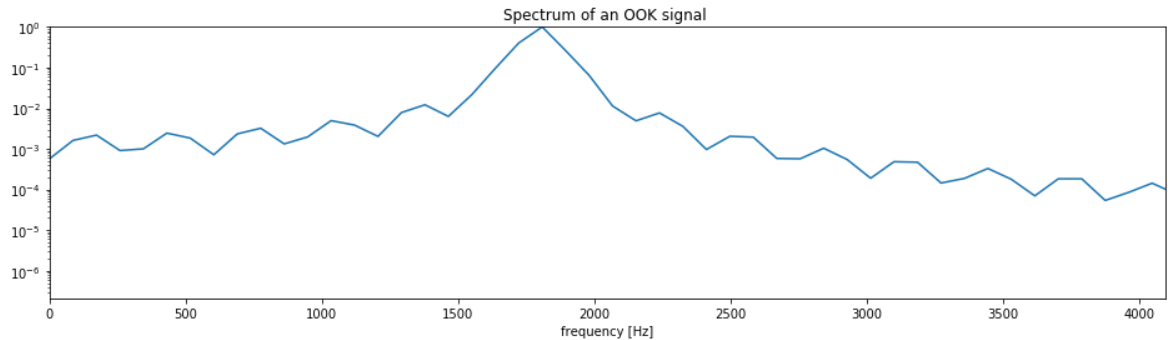
The approach of using OOK/ASK as above is not ideal. We can see this by plotting the spectrum

- Compute the spectrum of the above ook signal using FFT, and plot its magnitude.
- Comment on the spectrum. What is the issue that may arise if one would like to transmit at nearby frequencies?

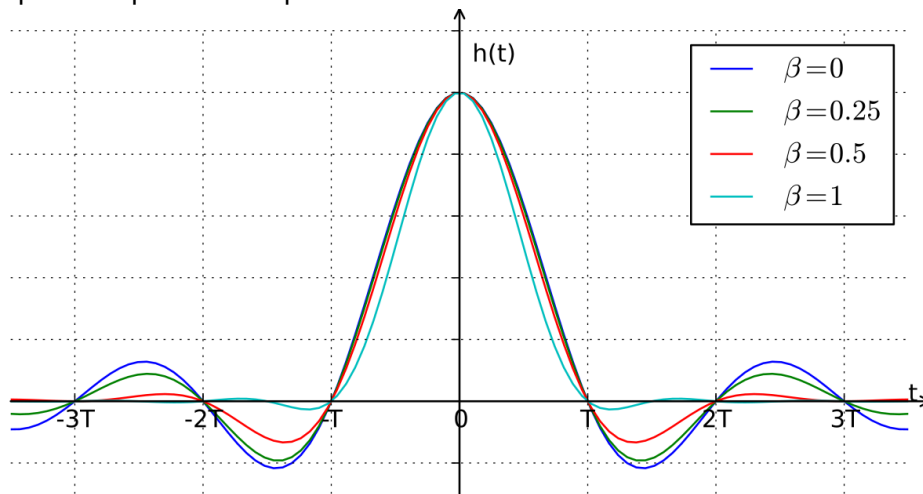
```
In [5]: (f,00K_f) = avgPS(00K,512,fs)

fig = plt.figure(figsize = (16,4))
plt.semilogy(f,abs(00K_f)/max(abs(00K_f)))
plt.xlabel('frequency [Hz]')
plt.title('Spectrum of an 00K signal ')
plt.axis((0,4096,0,1));
```

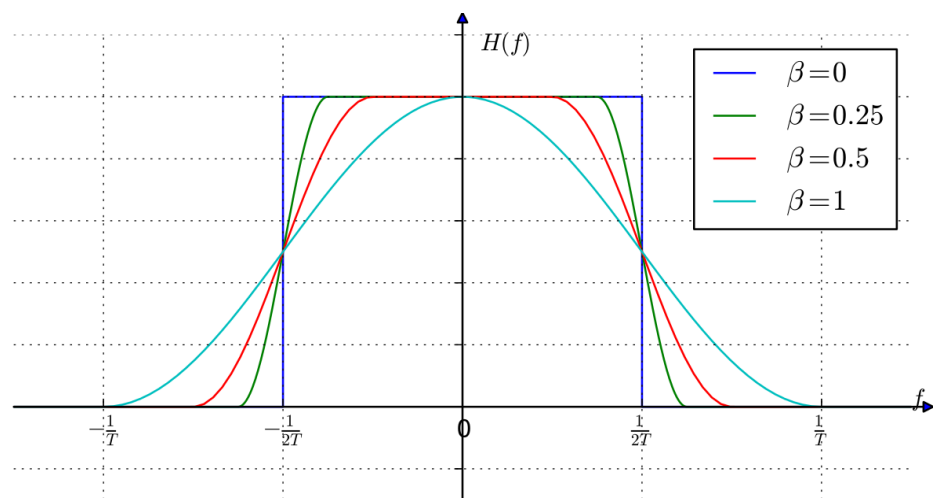
/home/sherman-tay/anaconda3/lib/python3.6/site-packages/matplotlib/axes/\_base.py:3245: UserWarning: Attempted to set non-positive ylimits for log-scale axis; invalid limits will be ignored.  
'Attempted to set non-positive ylimits for log-scale axis; '



Instead of using rectangular pulses, we can use shaped pulses which have more narrowband frequency response. For example, we could use sinc based pulses that would result in bandlimited spectrum. Of course it is not practical to use a sinc, since it is infinite in length. Instead we can use a truncated (windowed) sinc pulses with certain time bandwidth (TBW) product that would have a finite length as well as narrow bandness. While here we use a windowed sinc function as we learned in class, in the communications community the raised cosine filter is often used. A raised cosine is a tapered sinc function (with parameter  $0 \leq \beta \leq 1$ ) in which the impulse response and spectrum looks like\*:



For more information about the raised cosine function look [here](#)



([http://www.commsys.isy.liu.se/TSKS04/lectures/3/MichaelZoltowski\\_SquareRootRaisedCosine.pdf](http://www.commsys.isy.liu.se/TSKS04/lectures/3/MichaelZoltowski_SquareRootRaisedCosine.pdf))

Now, going back to thinking in terms of the time-bandwidth product (TBW), the TBW of a rectangular pulse is 1, so for a symbol length of  $1/300$  seconds we get a bandwidth of 300 Hz (the main lobe full width at half the maximum) but lots of ripple. We could use a TBW = 2 pulse (like a hamming window or higher order pulses.

- Using the random bit sequence chosen above, generate a new sequence with either zero or discrete impulses spaced  $f_s/\text{baud}$  samples apart. For example a sequence of 1010 would have an impulse at position 0, an impulse at position 294 and zeros elsewhere
- Generate a TBW = 4 windowed sinc pulse with zero-crossing occurring every 147 samples.
- Convolve the sinc with the impulse train to generate a sinc OOK/ASK signal.
- modulate the result to 1800Hz
- Plot the signal and its spectrum
- Comment on the separation between on and off.... what happened? Comment on the spectrum.

\*image sources: <http://upload.wikimedia.org/wikipedia/commons/thumb/8/8b/Raised-cosine-impulse.svg/1200px-Raised-cosine-impulse.svg.png> (<http://upload.wikimedia.org/wikipedia/commons/thumb/8/8b/Raised-cosine-impulse.svg.png>)



```

In [6]: message = zeros(N)
        message[:,Ns] = bits.ravel()

        h = signal.firwin(Ns*4,1.0/Ns) # Hamming window

        # Plot original message
        fig = plt.figure(figsize = (16,4))
        t = r_[0.0:len(message)]/fs
        plt.plot(t[:Ndisp*Ns],message[:Ndisp*Ns])
        plt.xlabel('time [s]')
        plt.title('Original sequence:'+ np.array2string(np.transpose(np.uint8
        (bits[:Ndisp]))))

        # Smooth message with window, h
        message = np.convolve(message,h,mode='same')
        fig = plt.figure(figsize = (16,4))
        plt.plot(t[:Ndisp*Ns],message[:Ndisp*Ns])
        plt.xlabel('time [s]')
        plt.title('Smoothed sequence:'+ np.array2string(np.transpose(np.uint8
        (bits[:Ndisp]))))

        00K_hamm = message*sin(2*pi*f0*t)

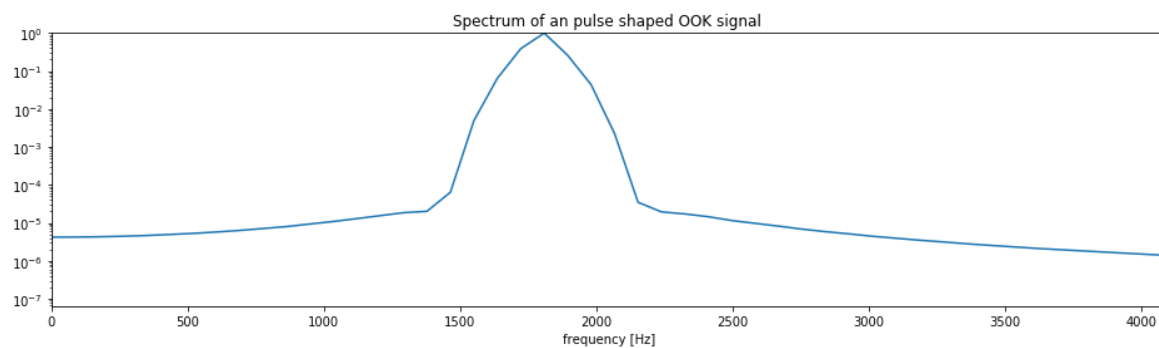
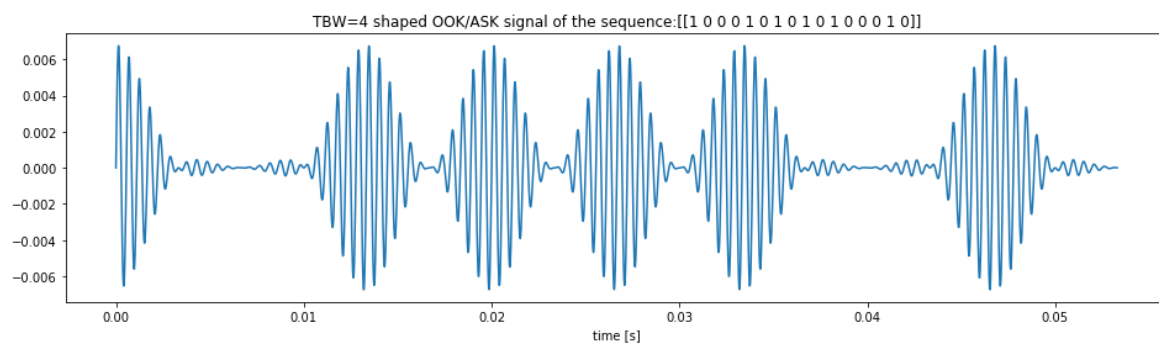
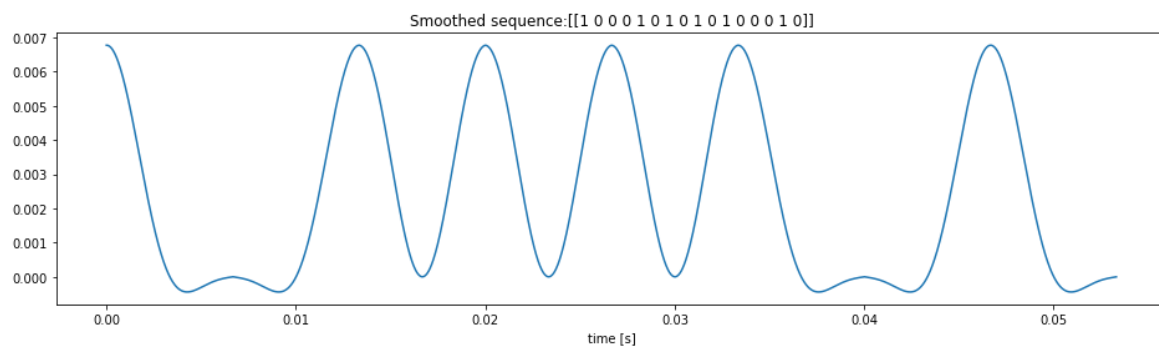
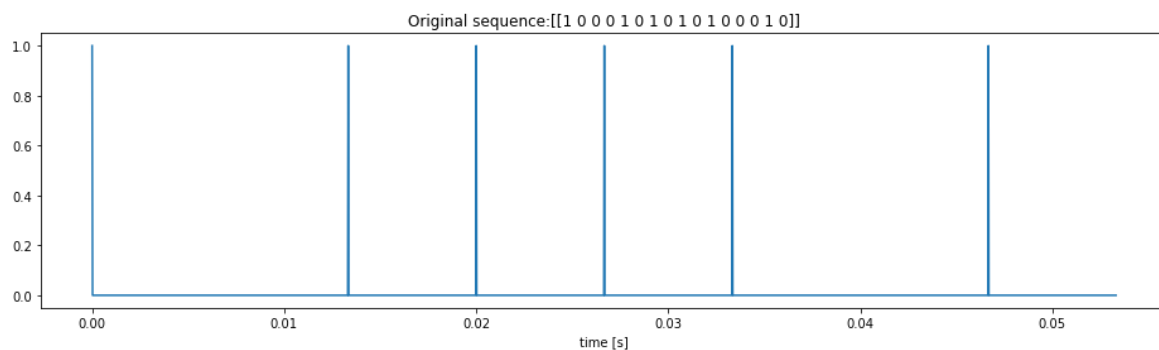
        fig2 = plt.figure(figsize = (16,4))
        plt.plot(t[:Ndisp*Ns],00K_hamm[:Ndisp*Ns])
        plt.xlabel('time [s]')
        plt.title('TBW=4 shaped 00K/ASK signal of the sequence:'+ np.array2st
        ring(np.transpose(np.uint8(bits[:Ndisp]))))

        (f,00K_hf) = avgPS(00K_hamm,512,fs)

        fig3 = plt.figure(figsize = (16,4))
        plt.semilogy(f,abs(00K_hf)/max(abs(00K_hf)))
        plt.xlabel('frequency [Hz]')
        plt.title('Spectrum of an pulse shaped 00K signal ')
        plt.axis((0,4096,0,1));

```

```
/home/sherman-tay/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_base.py:3245: UserWarning: Attempted to set non-positive ylimits for log-scale axis; invalid limits will be ignored.
      'Attempted to set non-positive ylimits for log-scale axis; '
```



## Comment on the separation between on and off.... what happened? Comment on the spectrum. Your answer here:

The separation between on and off is much lower, from being  $\pm 1$  to  $\pm 0.06$  for the OOK/ASK sequence/ The spectrum appears to have lower sideband average power and is much more smooth

### Phase-shift-keying (PSK)

PSK encodes digital data in the phase of the signal. For example, Binary PSK (BPSK) sends a stream of '0's and '1's by alternating the sign of the carrier in each bit period:  $\text{Real} \{e^{j\pi M(t)} \cos(2\pi f_0 t)\}$ , where  $M(t)$  is a signal of 0 or 1 corresponding to '0's and '1's. The RDS subcarrier in FM radio is BPSK encoded. Also, [PSK31](http://en.wikipedia.org/wiki/PSK31) (<http://en.wikipedia.org/wiki/PSK31>) is a very popular text-to-text ham radio protocol for HF that has 31b/s rate, which matches a typical keyboard typing rate. An M-ary symbol can be transmitted by MPSK in which  $M(t)$  are broken to smaller intervals. For example in 4 PSK (QPSK) 0,1,2,4 are encoded in  $M(t) = \{0 \text{ or } 0.5 \text{ or } 1 \text{ or } 1.5\}$ . It could also be encoded in a  $45^\circ$  rotated constellation  $M(t) = \{0.25 \text{ or } 0.75 \text{ or } 1.25 \text{ or } 1.75\}$

Here's an example of a BPSK signal:

```
In [8]: message = zeros(N)
message[:,Ns] = 2*bits.ravel() - 1    #Makes the message +/- 1

fig = plt.figure(figsize = (16,4))
plt.plot(t[:Ns*Ndisp],message[:Ns*Ndisp])
plt.xlabel('time [s]')
plt.title('exp[j\pi M(t)]:'+ np.array2string(np.transpose(np.uint8(bits[:Ndisp]))))

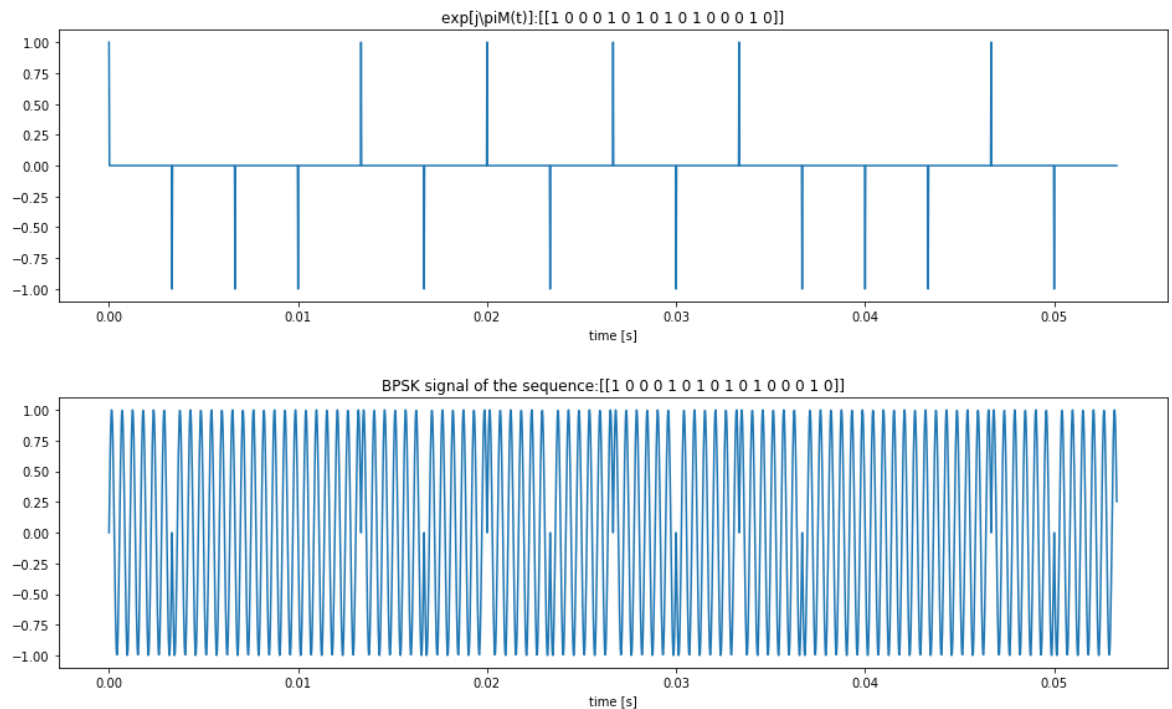
# Convolve
h = np.ones(Ns)
message = signal.fftconvolve(message,h,mode='full')
t = r_[0.0:len(message)]/fs
BPSK = message*sin(2*pi*f0*t)

fig = plt.figure(figsize = (16,4))
plt.plot(t[:Ns*Ndisp],BPSK[:Ns*Ndisp])
plt.xlabel('time [s]')
plt.title('BPSK signal of the sequence:'+ np.array2string(np.transpose(np.uint8(bits[:Ndisp]))))
```

```
/home/sherman-tay/anaconda3/lib/python3.6/site-packages/scipy/signal/
signaltools.py:375: FutureWarning: Conversion of the second argument
of issubdtype from `complex` to `np.complexfloating` is deprecated. I
n future, it will be treated as `np.complex128 == np.dtype(complex).t
ype`.
```

```
complex_result = (np.issubdtype(in1.dtype, complex) or
/home/sherman-tay/anaconda3/lib/python3.6/site-packages/scipy/signal/
signaltools.py:376: FutureWarning: Conversion of the second argument
of issubdtype from `complex` to `np.complexfloating` is deprecated. I
n future, it will be treated as `np.complex128 == np.dtype(complex).t
ype`.
np.issubdtype(in2.dtype, complex))
```

```
Out[8]: Text(0.5,1,'BPSK signal of the sequence:[[1 0 0 0 1 0 1 0 1 0 1 0 0 0
1 0]]')
```



One of the advantages of PSK is that the envelope of the signal is constant. This means that we don't have to use linear amplifiers (also called class A or AB) in order to amplify the signal for transmission. Instead we can use the much more efficient non-linear amplifiers (class C or E). However, the problem is of course that the phase has a discontinuity. This will have an effect on the spectrum.

- Compute the spectrum of the above BPSK signal. Comment on the spectrum The spectrum appears to have high sideband power and does not have a smooth spectrum

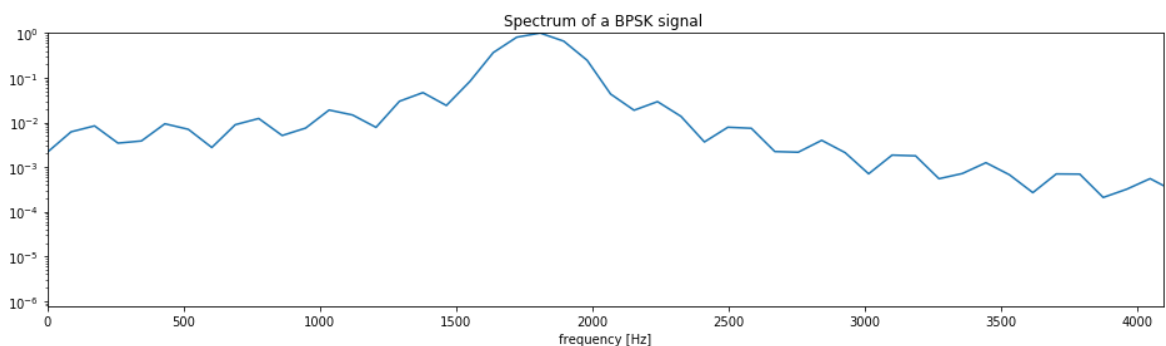
```
In [12]: f = r_[0:len(BPSK)/2.0]/len(BPSK)*fs

(f,BPSK_f) = avgPS(BPSK,512,fs)

fig = plt.figure(figsize = (16,4))
plt.semilogy(f,abs(BPSK_f)/max(BPSK_f))
plt.xlabel('frequency [Hz]')
plt.title('Spectrum of a BPSK signal ')
plt.axis((0,4096,0,1));
```

/home/sherman-tay/anaconda3/lib/python3.6/site-packages/matplotlib/axes/\_base.py:3245: UserWarning: Attempted to set non-positive ylimits for log-scale axis; invalid limits will be ignored.

'Attempted to set non-positive ylimits for log-scale axis; '



We can use the same "trick" as before and shape the pulses with a windowed sinc function.

- Repeat the procedure for the case of BPSK similarly to the shaped ASK/OOK as before. However instead of a zero-one pulse train use a  $\pm 1$  train to get the changes in phase.
- What did we win and what did we lose? Comment on the spectrum the symbol leakage and the envelope of the signal.

```
In [13]: message = zeros(N)
message[:,Ns] = 2*bits.ravel() - 1
t = r_[0.0:len(message)]/fs #Time vector
h = signal.firwin(Ns*4,1.0/Ns) # Hamming window

# Repeat the computation of the BPSK signal, but use the trick from before to shape the pulses.
# Your final signal should be called BPSK_s.
# Your code here:

# Smooth message with window, h
message = np.convolve(message,h,mode='same')

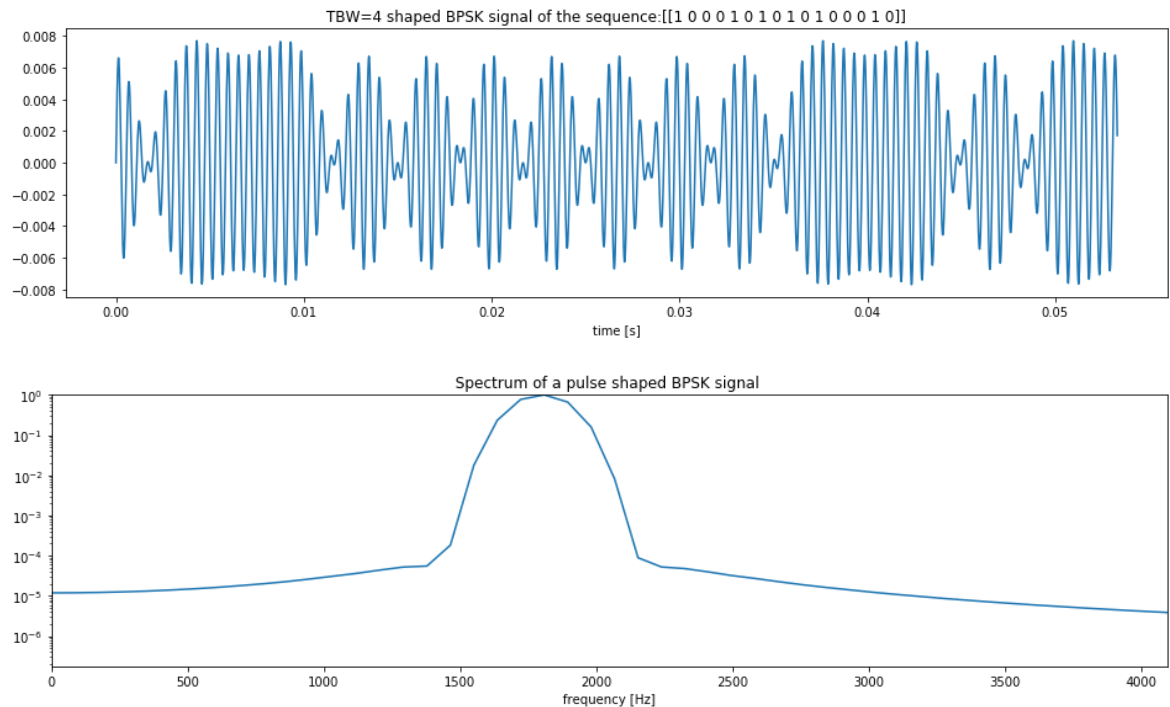
BPSK_s = message*sin(2*pi*f0*t)

fig = plt.figure(figsize = (16,4))
plt.plot(t[:Ndisp*Ns],BPSK_s[:Ndisp*Ns])
plt.xlabel('time [s]')
plt.title('TBW=4 shaped BPSK signal of the sequence:'+ np.array2string(np.transpose(np.uint8(bits[:Ndisp]))))

(f,BPSK_sf) = avgPS(BPSK_s,512,fs)

fig = plt.figure(figsize = (16,4))
plt.semilogy(f,abs(BPSK_sf)/max(BPSK_sf))
plt.xlabel('frequency [Hz]')
plt.title('Spectrum of a pulse shaped BPSK signal ')
plt.axis((0,4096,0,1));
```

```
/home/sherman-tay/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_base.py:3245: UserWarning: Attempted to set non-positive ylimits for log-scale axis; invalid limits will be ignored.  
'Attempted to set non-positive ylimits for log-scale axis; '
```



**What did we win and what did we lose? Comment on the spectrum the symbol leakage and the envelope of the signal. Your answer here:**

The envelope of the signal is no longer constant for PSK. However, the spectrum has a smoother power spectrum. There will be also greater spectral leakage.



## Quadrature-Amplitude-Modulation (QAM)

QAM encodes M-ary digital data in both the amplitude and phase, i.e.,  $\text{Real} \{M(t) \cos(2\pi f_0 t)\}$  where  $M(t)$  have an amplitude and phase that divides the complex plane into M segments. QAM is heavily used in cable TV and modems. You can see constellations of QAM-16 to QAM-256 which break the complex plane to  $4 \times 4$  and  $16 \times 16$  grids.

Here's an image of the constellation of QAM-16:

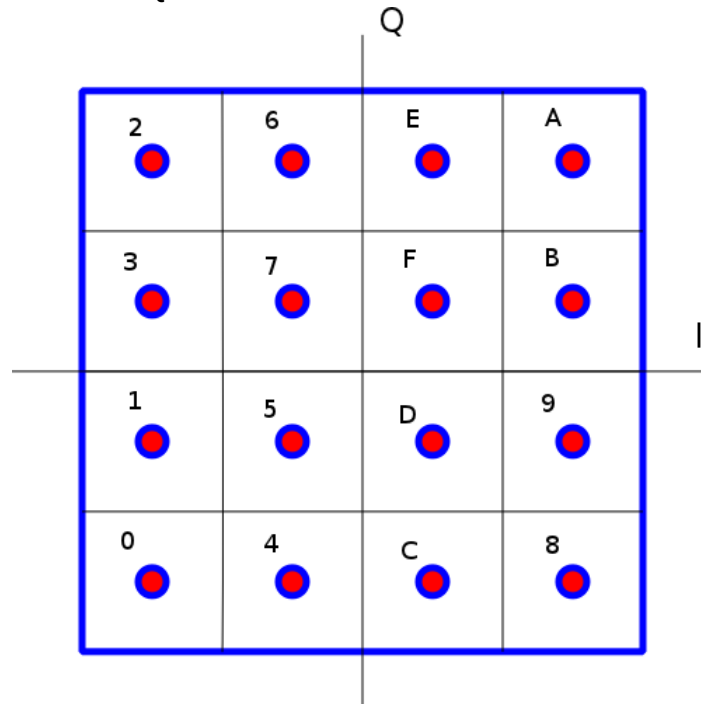


Figure 1: QAM-16 Constellation

```

In [14]: #code = { 2: -2+2j, 6: -1+2j, 14: 1+2j, 10: 2+2j,
#              3: -2+1j, 7: -1-1j, 15: 1+1j, 11: 2+1j,
#              1: -2-1j, 5: -1-1j, 13: 1-1j, 9: 2-1j,
#              0: -2-2j, 4: -1-2j, 12: 1-2j, 8: 2-2j}
Nbits = 16 # number of bits
N = Nbits * Ns
code = np.array((-2-2j, -2-1j, -2+2j, -2+1j, -1-2j, -1-1j, -1+2j, -1+1j, +2-
2j, +2-1j, +2+2j, +2+1j, 1-2j, 1-1j, 1+2j, 1+1j))/2
np.random.seed(seed=1)
bits = np.int16(np.random.rand(Nbits,1)*16)

message = zeros(N, dtype='complex')
message[:,Ns] = code[bits].ravel()

# Convolve
h = np.ones(Ns)
message = signal.fftconvolve(message,h,mode='full')
t = r_[0.0:len(message)]/fs
QAM = np.real(message*exp(1j*2*pi*f0*t))

fig = plt.figure(figsize = (16,4))
plt.plot(t[:Ns*Ndisp],QAM[:Ns*Ndisp])
plt.xlabel('time [s]')
plt.title("QAM=16 of the sequence:"+ np.array2string(np.transpose(bits
[:Ndisp])))

```

```

/home/sherman-tay/anaconda3/lib/python3.6/site-packages/scipy/signal/
signaltools.py:375: FutureWarning: Conversion of the second argument
of issubdtype from `complex` to `np.complexfloating` is deprecated. I
n future, it will be treated as `np.complex128 == np.dtype(complex).t
ype`.

```

```

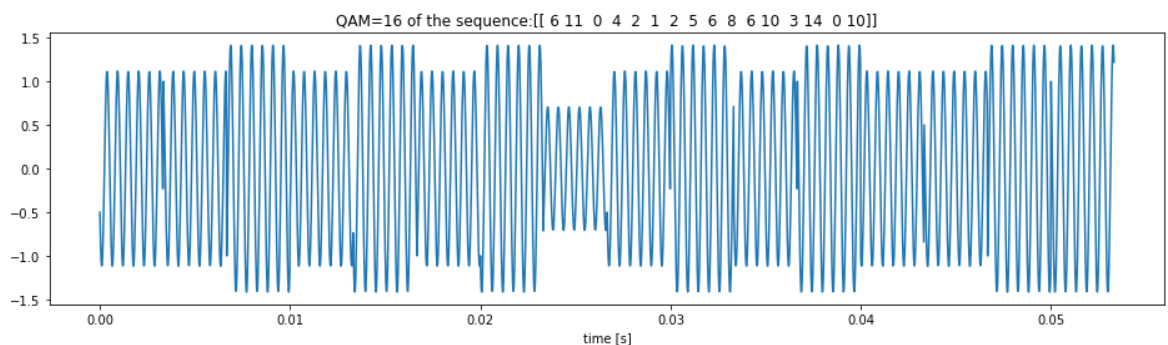
    complex_result = (np.issubdtype(in1.dtype, complex) or

```

```

Out[14]: Text(0.5,1,'QAM=16 of the sequence:[[ 6 11  0  4  2  1  2  5  6  8  6
10  3 14  0 10]]')

```



QAM is a very efficient way to use the amplitude as well as the phase to encode M-ary symbols.

- Compute the spectrum of the above QAM 16 signal.
- Repeat the same for shaped TBW=4 pulses.
- Is the bandwidth much different than the binary sequences? What is the bitrate improvement per bandwidth you get?

```
In [15]: (f,QAM_f) = avgPS(QAM,512,fs)

fig = plt.figure(figsize = (16,4))
plt.semilogy(f,QAM_f/max(QAM_f))
plt.xlabel('frequency [Hz]')
plt.title('Spectrum of a QAM16 signal ')
plt.axis((0,4096,0,1))

message = zeros(N,dtype='complex')
message[::Ns] = code[bits].ravel()

h = signal.firwin(Ns*4,1.0/Ns)
message = signal.fftconvolve(message,h,mode='full')
t = r_[0.0:len(message)]/fs
QAM_s = np.real(message*exp(1j*2*pi*f0*t))

fig = plt.figure(figsize = (16,4))
plt.plot(t[:Ns*Ndisp],QAM_s[:Ns*Ndisp])
plt.xlabel('time [s]')
plt.title('TBW=4 shaped QAM16 signal of the sequence:'+ np.array2string(np.transpose(bits[:Ndisp])))

(f,QAM_sf) = avgPS(QAM_s,512,fs)

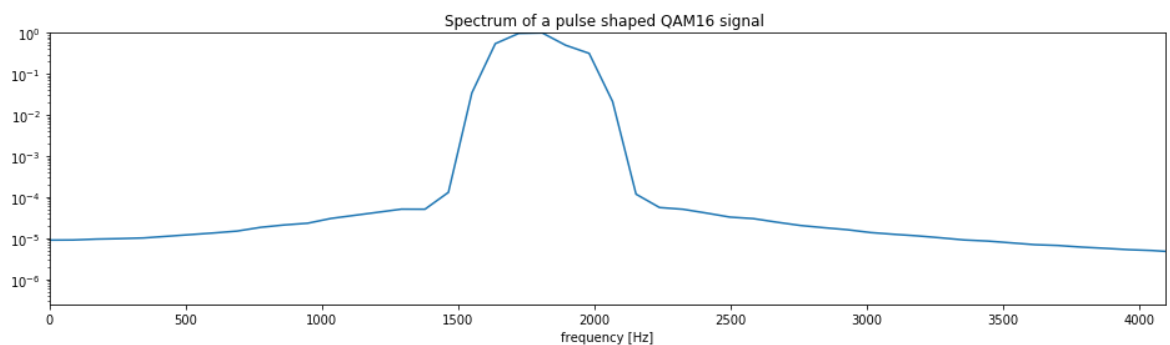
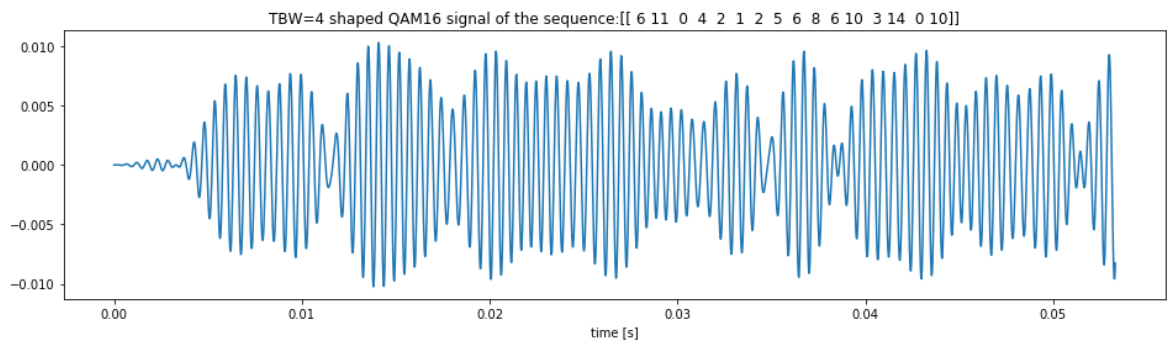
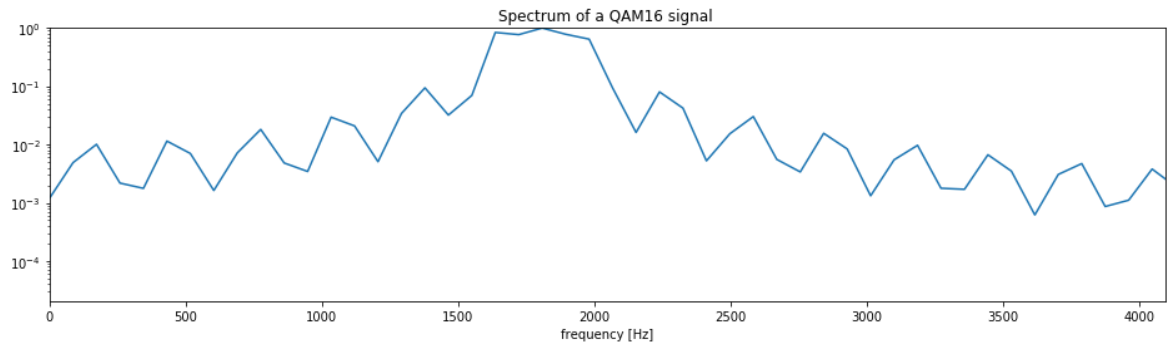
fig = plt.figure(figsize = (16,4))
plt.semilogy(f,QAM_sf/max(QAM_sf))
plt.xlabel('frequency [Hz]')
plt.title('Spectrum of a pulse shaped QAM16 signal ')
plt.axis((0,4096,0,1));
```

```
/home/sherman-tay/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_base.py:3245: UserWarning: Attempted to set non-positive ylimits for log-scale axis; invalid limits will be ignored.
```

```
'Attempted to set non-positive ylimits for log-scale axis; '
```

```
/home/sherman-tay/anaconda3/lib/python3.6/site-packages/scipy/signal/signaltools.py:375: FutureWarning: Conversion of the second argument of issubdtype from `complex` to `np.complexfloating` is deprecated. In future, it will be treated as `np.complex128 == np.dtype(complex).type`.
```

```
complex_result = (np.issubdtype(in1.dtype, complex) or
```



**Is the bandwidth much different than the binary sequences? What is the bitrate improvement per bandwidth you get? Your answers here:**

The bandwidth is not much different from that of the binary sequence as the width of the power spectrum is similar. At constant bandwidth, the bitrate improvement is of the order of 4, as each symbol is able to carry 4 bits.

## Frequency-Shift-Keying (FSK)

FSK encodes digital data in the frequency of the carrier, i.e.,  $\cos(2\pi(f_0 + M(t)\Delta f)t)$ . For binary FSK (BFSK),  $M(t) = \{1 \text{ or } -1\}$ . In M-ary FSK (MFSK) the frequency band can be broken to more segments. Again, much like PSK, FSK has also a constant envelope which makes it ideal for use with efficient non-linear amplifiers.

The choice of frequencies has a huge impact on our ability to demodulate the signal as well as the bandwidth of it. For example, we would like the frequencies to be orthogonal over a bit period so when detected using correlation we will get the maximum difference between the results. We would also like the phase to be continuous for the sidelobes to decay rapidly and not interfere with nearby channels.

For example, the choice of frequencies of 1200Hz and 2400Hz below for a bitrate of 300 baud results in 4 and 8 cycles per bit period respectively. The outcome is a smooth phase and orthogonal frequencies.

```

In [21]: np.random.seed(seed=1)
Nbits = 256
N = Nbits * Ns
bits = np.random.randn(Nbits,1) > 0
M = np.tile(bits*2-1,(1,Ns))
delta_f = 600

# compute phase by integrating frequency
ph = 2*pi*np.cumsum(f0 + M.ravel()*delta_f)/fs
t = r_[0.0:N]/fs
FSK = sin(ph)

fig = plt.figure(figsize = (16,4))
plt.plot(t[:Ns*Ndisp],M.ravel()[:Ns*Ndisp])
plt.xlabel('time [s]')
plt.title('Message:'+ np.array2string(np.transpose(np.uint8(bits[:Ndisp]))))

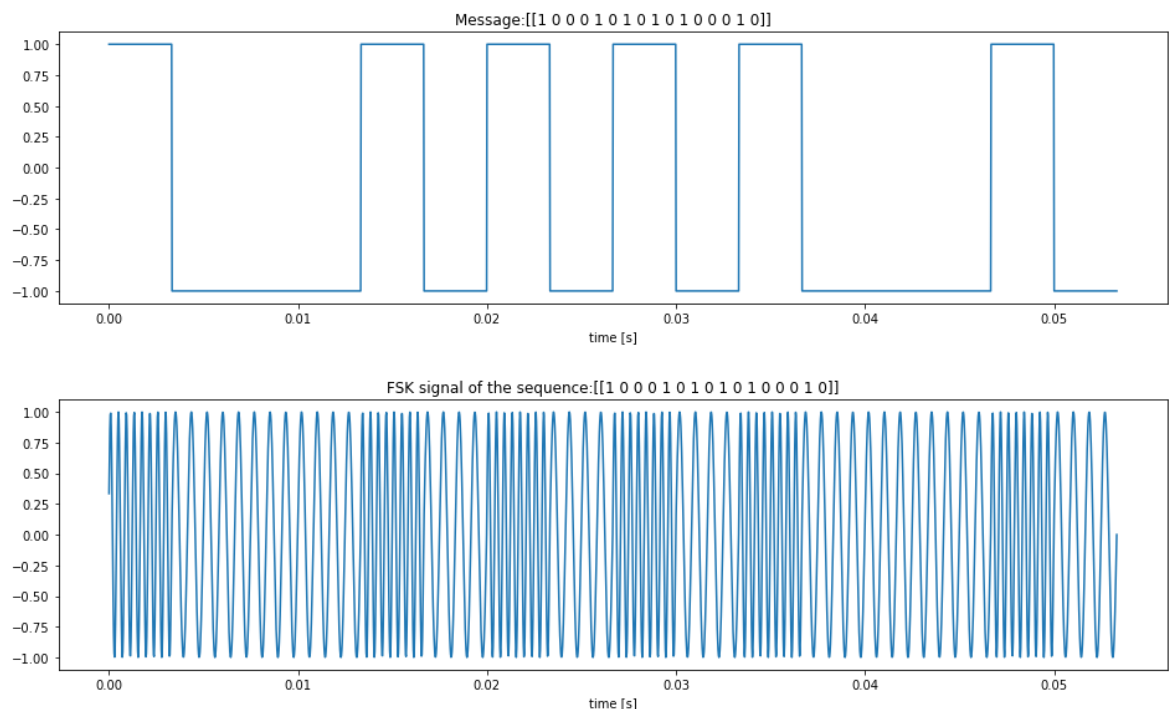
fig = plt.figure(figsize = (16,4))
plt.plot(t[:Ns*Ndisp],FSK[:Ns*Ndisp])
plt.xlabel('time [s]')
plt.title('FSK signal of the sequence:'+ np.array2string(np.transpose(
np.uint8(bits[:Ndisp]))))

```

```

Out[21]: Text(0.5,1,'FSK signal of the sequence:[[1 0 0 0 1 0 1 0 1 0 1 0 0 0
1 0]]')

```



- Compute the spectrum of the above FSK signal. Can you see the frequencies?

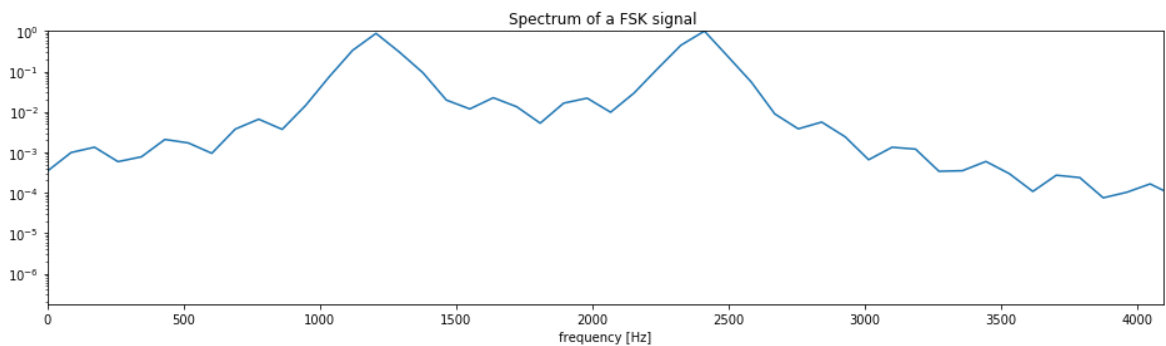
```
In [17]: (f,FSK_f) = avgPS(FSK,512,fs)
```

```
fig = plt.figure(figsize = (16,4))
plt.semilogy(f,FSK_f/max(FSK_f))
plt.xlabel('frequency [Hz]')
plt.title('Spectrum of a FSK signal ')
plt.axis((0,4096,0,1))
```

```
/home/sherman-tay/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_base.py:3245: UserWarning: Attempted to set non-positive ylimits for log-scale axis; invalid limits will be ignored.
```

```
'Attempted to set non-positive ylimits for log-scale axis; '
```

```
Out[17]: (0, 4096, 0, 1)
```



The reason the frequencies are separated well is the low bit-rate. For the same choice of frequencies we can increase the baud (i.e. the bit-rate) and still get continuous phase and orthogonal frequencies. For example: with a 600 baud we will get 2 cycles for 1200Hz and 4 cycles for 2400Hz per bit period. We can continue further for 1200baud which results in 1cycle/2cycle for 1200Hz/2400Hz.

Finally, the highest rate in which we can get orthogonal frequencies is 2400 baud. In that case, we will have 1/2 a cycle at 1200Hz and 1 cycle for 2400Hz. The phase will still be continuous and the frequencies orthogonal. This form of FSK modulation is also called Minimum-Shift-Keying (MSK) and is generalized to the case where the frequency difference is half the bit-rate.

- What would be the choice of frequencies for a 300 baud MSK centered at 1800Hz?
- Generate the MSK-300 modulation of the above signal.
- Plot the signal as well as its spectrum and spectrogram For the spectrogram use 148 window (which is approximately the bit length). Comment on the bandwidth of the signal compared to the BFSK above.

```
In [22]: delta_f = 150
ph = 2*pi*np.cumsum(f0 + M.ravel()*delta_f)/fs
t = r_[0.0:N]/fs
MSK = sin(ph)

fig = plt.figure(figsize = (16,4))
plt.plot(t[:Ns*Ndisp],MSK[:Ns*Ndisp])
plt.xlabel('time [s]')
plt.title('MSK signal of the sequence:'+ np.array2string(np.transpose
(np.uint8(bits[:Ndisp]))))

(f,MSK_f) = avgPS(MSK,512,fs)

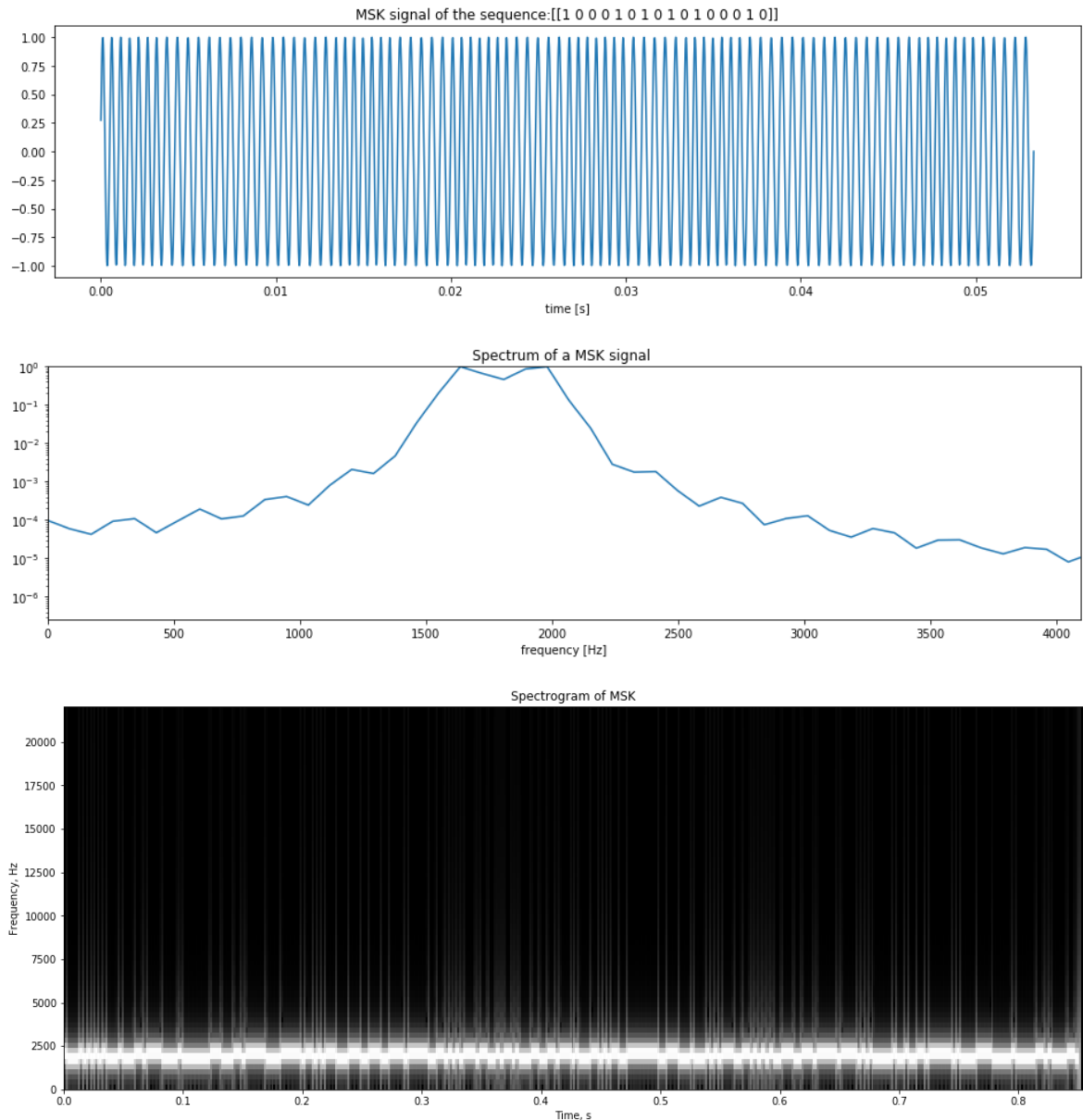
fig = plt.figure(figsize = (16,4))
plt.semilogy(f,MSK_f/max(MSK_f))
plt.xlabel('frequency [Hz]')
plt.title('Spectrum of a MSK signal ')
plt.axis((0,4096,0,1))
tt,ff,xmf = myspectrogram_hann_ovlp(MSK, 148, 44100.0, 0,dbf = 20)
plt.title('Spectrogram of MSK')
```



1

```
/home/sherman-tay/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_base.py:3245: UserWarning: Attempted to set non-positive ylimits for log-scale axis; invalid limits will be ignored.
'Attempted to set non-positive ylimits for log-scale axis; '
```

Out[22]: Text(0.5,1,'Spectrogram of MSK')



**Comment on the bandwidth of the signal compared to the BFSK above:**

The required bandwidth is much less.

MSK is a very efficient modulation with constant envelope and a narrow bandwidth. However its sidelobes are still prohibitly high for application like cellphone networks. The sidelobes can be further reduce by prefiltering the frequency transition by a smoothing filter. A very common filter is a gaussian filter. When a gaussian filter is used, the modulation is called GMSK. This is the modulation that is used for GSM networks. Much like the other modulations, filtering reduces the sidelobes, but increase the inter-symbol interference.

The Gaussian filter used in GMSK is defined as:

$$h(t) = \sqrt{\frac{2\pi}{\log 2}} \cdot BW \exp \left( -\frac{2\pi^2 (BW)^2}{\log 2} t^2 \right),$$

where BW is determined by the time-bandwidth product (TBW) and the duration of the gaussian pulse, which is  $\frac{T_b}{TBW}$  -- where  $T_b$  is the bit time-period.

In the discrete case, where  $f_s$  is the sampling frequency

$$h[n] = \sqrt{\frac{2\pi}{\log 2}} \cdot \frac{BW}{f_s} \exp \left( -\frac{2\pi^2 (BW)^2}{\log 2} \frac{n^2}{f_s^2} \right).$$

- Generate a TBW=0.5 gaussian filter over a 2bit time period. What is the bandwidth?
- Plot the filter.
- Filter the frequency of the MSK signal above using the gaussian filter. Integrate to generate the phase to generate the modulated signal as above.
- Plot the signal and its spectrum. Comment on the result. Does the spectrum have less sidelobes?
- Explain why the spectrum looks like it is and compare to the spectrogram.

```

In [23]: TBW =0.5
          Tb = 294
          B = Tb*TBW
          n = r_[-Tb/2:Tb/2]
          h = sqrt(2*pi/log(2))*B/fs*exp(-2*pi*pi*B*B*n*n/fs/fs/log(2))
          t = r_[0.0:N]/fs

          freq = np.convolve(f0 + M.ravel()*delta_f,h,'same')

          fig = plt.figure(figsize=(16,4))
          plt.plot(t[:Ns*Ndisp],freq[:Ns*Ndisp],t[:Ns*Ndisp],1700*M.ravel()[:Ns
*Ndisp])
          plt.title('Smoothed frequency')
          plt.legend(['Smoothed frequency modulation','message'])

          ph = 2*pi*np.cumsum(freq)/fs

          GMSK = sin(ph)

          t = r_[0.0:len(GMSK)]/fs
          fig = plt.figure(figsize = (16,4))
          plt.plot(t[:Ns*Ndisp],GMSK[:Ns*Ndisp])
          plt.xlabel('time [s]')
          plt.title('GMSK signal of the sequence:'+ np.array2string(np.transpose
(np.uint8(bits[:Ndisp]))))

          (f,GMSK_f) = avgPS(GMSK[Ns:-Ns],512,fs)

          fig = plt.figure(figsize = (16,4))
          plt.semilogy(f,GMSK_f/max(GMSK_f))
          plt.xlabel('frequency [Hz]')
          plt.title('Spectrum of a GMSK signal ')
          plt.axis((0,4096,0,1))

          tt,ff,xmf = myspectrogram_hann_ovlp(GMSK[:Ns*Ndisp], 148, 44100.0, 0,
dbf = 30)
          plt.axis((0, .05, 0, 5000))
          plt.title('Spectrogram of GMSK (first 16 bits)')

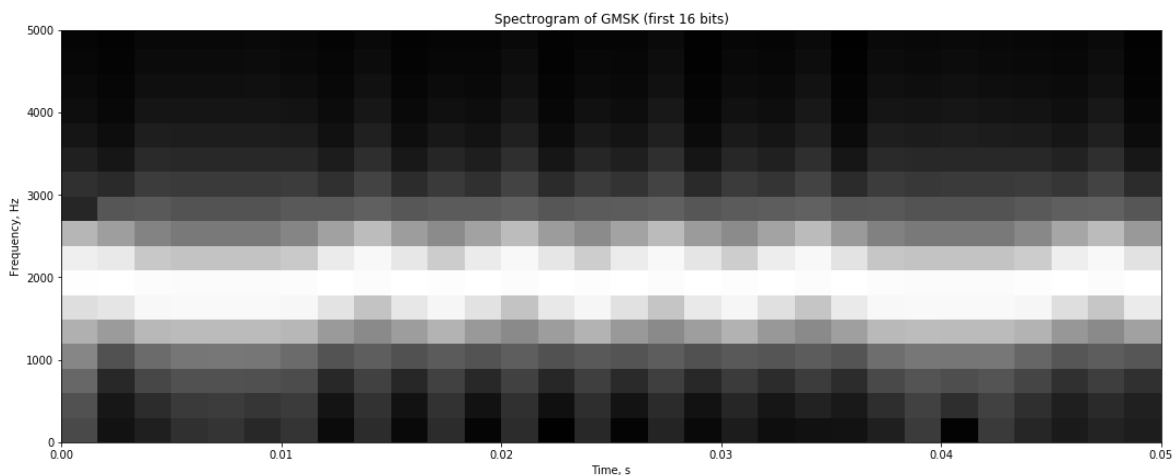
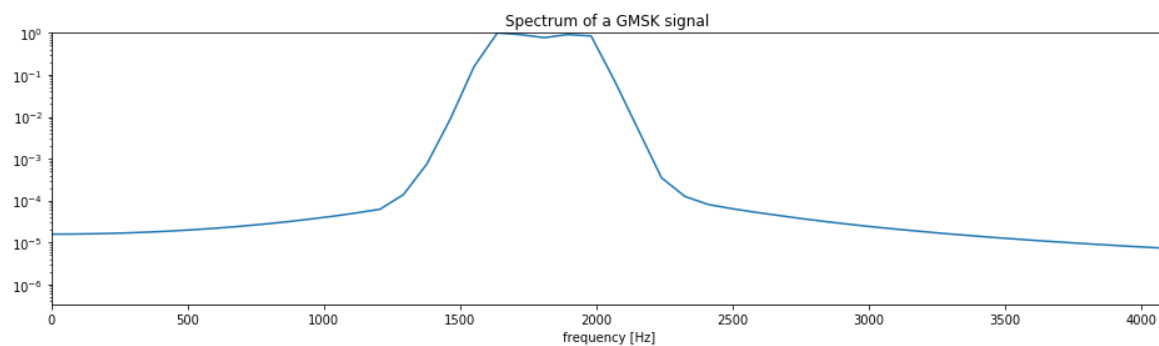
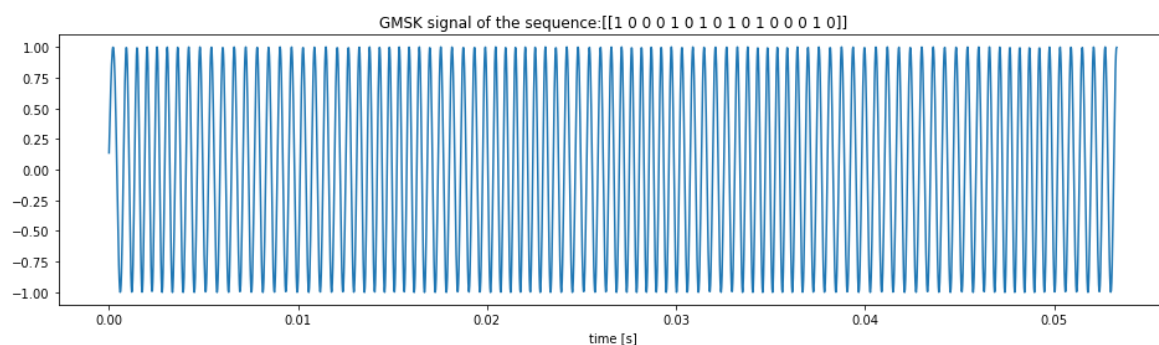
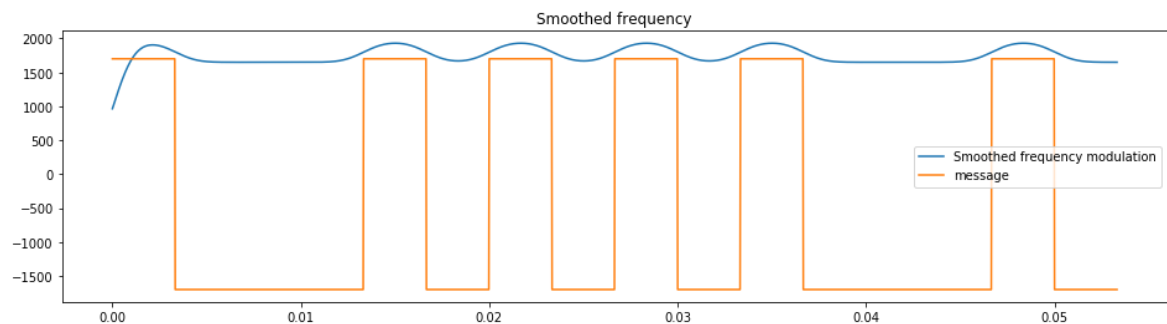
```

1

```
/home/sherman-tay/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_base.py:3245: UserWarning: Attempted to set non-positive ylimits for log-scale axis; invalid limits will be ignored.
```

```
'Attempted to set non-positive ylimits for log-scale axis; '
```

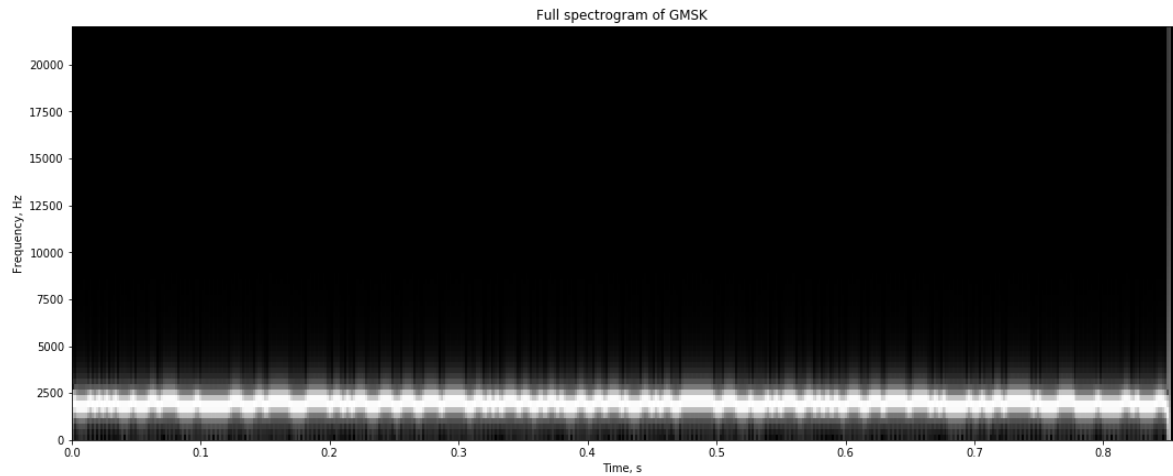
```
Out[23]: Text(0.5,1,'Spectrogram of GMSK (first 16 bits)')
```



```
In [24]: fig = plt.figure(figsize = (16,4))  
         tt,ff,xfm = myspectrogram_hann_ovlp(GMSK, 148, 44100.0, 0,dbf = 30)  
         plt.title('Full spectrogram of GMSK')
```

1

```
Out[24]: Text(0.5,1,'Full spectrogram of GMSK')  
<matplotlib.figure.Figure at 0x7f6ad8397ba8>
```



In [ ]: