Homework 2 (ge193 Imaging radar and applications)

In [ ]:
```
print('This is a chirp and compress coding example.')

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

matplotlib.rcParams.update({'font.size': 18})
plt.rcParams.update({'font.family':'Verdana'})
```

```
This is a chirp and compress coding example.
```

How to make a reference chirp signal

To construct a chirp waveform, use the following relation

$$\phi(t) = \pi \cdot slope \cdot t^2 + 2\pi f_c t$$

where

- $\phi(t)$: chirp phase
- $slope$: slope of the chirp
- t: time axis
- f_c: central frequency

Then, the chirp waveform can be expressed as:

$$r(t) = exp^{(-j \times \phi(t))}$$

- $r(t)$: the chirp waveform
- j: the imaginary number

Define some functions for future usage

In [ ]:
```
# make a function that creates a chirp
    # N:     Number of points of the pulse
    # slope: slope of the chirp
    # tau:   chirp length
    # fs:    sample rate
    # fc:    central carrier freq
    # start: starting sample # of the chirp
def makechirp(N, slope, tau, fs, fc=0, phi0=0, start=0):
    dt    = 1/fs                                        # sampling time interval
    npts  = tau * fs                                    # num of points of the chirp
    t     = dt * np.arange(-npts/2, npts/2)             # time axis of the chirp
    phase = np.pi*slope*(t**2) + 2*np.pi*fc*t + phi0    # chirp phase
    chirp = np.exp(1j*phase)                            # chirp
    chirp = np.pad(chirp, (start,N-len(chirp)-start))   # pad zeros at tail
    t_arr = dt * np.arange(0, N)                        # time axis of the whole pulse
    print('Chirp starts from {} samples, {} mu s'.format(start, 1e6*(start/fs)))
    return chirp, t_arr


def matched_filter(sig, ref):
    sig_fft = np.fft.fft(sig)                           # transform the signal to freq domain
```

```
    ref_fft = np.fft.fft(ref)                           # transform the reference chirp to freq domai
    spec    = sig_fft * np.conjugate(ref_fft)           # cross-correlation gives the spectrum
    comp    = np.fft.ifft(spec)                         # inverse transform it back to time domain
    return comp, spec


def quick_evaluate_compress(signal, N, slope, tau, fs, fc=0, explot=False, liplot=False):
    """
    Inputs:
        signal     signal to be compressed
        N          number of points of the reference chirp (including padded zeros)
        slope      frequency slope of the chirp [Hz/s]
        tau        pulse length of the chirp [s]
        fs         sampling rate [Hz]
        fc         central freq of the chirp [Hz]
        explot    whether to make extra plots [True/False]

    Output:
        a plot
    """
    chirp, t = makechirp(N, slope, tau, fs)

    BW = slope * tau

    comp, spec = matched_filter(signal, chirp)
    freq       = np.linspace(-fs/2, fs/2, N)    # Frequency axis


    # See the lobes at zoom-in
    comp_mag = np.fft.fftshift(20*np.log10(np.abs(comp) + 1e-30))
    spec_mag = np.fft.fftshift(20*np.log10(np.abs(spec) + 1e-30))

    peaks, _ = find_peaks(comp_mag)

    # Peak ratio = (main_lobe) - (first_side_lobe)
    idx = np.argsort(comp_mag[peaks])[::-1][:2]
    ratio = comp_mag[peaks][idx][0] - comp_mag[peaks][idx][1]


    if explot:
        plt.figure(figsize=[14,4])
        plt.plot(freq*1e-6, spec_mag, label='{:.2e} Hz/s'.format(slope))
        plt.title('magnitude of the impulse response spectrum')
        plt.ylabel('Magnitude [dB]')
        plt.xlim((fc-2*BW)*1e-6, (fc+2*BW)*1e-6)
        plt.legend(loc='upper right')
        plt.xlabel(r'Freq [MHz]')
        plt.show()

    if liplot:
        plt.figure(figsize=[14,4])
        plt.plot(t*1e6, np.abs(comp), label='{:.2e} Hz/s'.format(slope))
        plt.title('magnitude of the impulse response (time domain)')
        plt.ylabel('Magnitude')
        #plt.xlim()
        plt.legend(loc='upper right')
        plt.xlabel(r'Time [$\mu$s]')
        plt.show()
    else:
        t = (t-np.mean(t))*1e6
        plt.figure(figsize=[14,4])
        plt.plot(t, comp_mag, label='{:.2e} Hz/s'.format(slope))
        plt.plot(t[peaks], comp_mag[peaks], 'o')
        plt.title('dB magnitude of impulse response, peak ratio = {:.2f} dB'.format(ratio))
        plt.ylabel('Compressed signal, dB')
        plt.ylim(0,60)
        plt.xlim(-0.5,0.5)
        plt.legend(loc='upper right')
        plt.xlabel(r'Time [$\mu$s]')
        plt.show()
```

Problem 1: chirp compression

In the computer, create a data record consisting of 2048 complex samples, which contains a chirp signal with the following parameters:

- Slope (s) = $10^{12}\ Hz\ s^{-1}$
- Pulse length (τ) = 10^{-5} s
- Sample rate (f_s) = $10^{8}\ Hz$

Position the chirp at the beginning of the array, zero-filling the unused portion.

Compression:

Compression is also called matched filtering. This is just by applying the cross-correlation of two waveforms. We apply cross-correlation ($\star$) of the reference chirp $r(t)$ on the signal $s(t)$. The idea is to utilize the properties of cross-correlation to find a similar pattern (your reference chirp) in the incoming signal and amplify it.

$$i(t) = s(t) \star r(t) = r(t) \star s(t)$$

where

- $i(t)$: the compressed signal (or the impulse response) in time domain
- $r(t)$: the reference chirp in time domain
- $s(t)$: the incoming signal that need to be compressed in time domain

In frequency domain, the cross-correlation is done in multiplication ($*$):

$$I(f) = S(f) * R(f) = R(f) * S(f)$$

where

- $I(f)$: the compressed signal (or the impulse response) in freq domain
- $R(f)$: the reference chirp in freq domain
- $S(f)$: the incoming signal that need to be compressed in freq domain

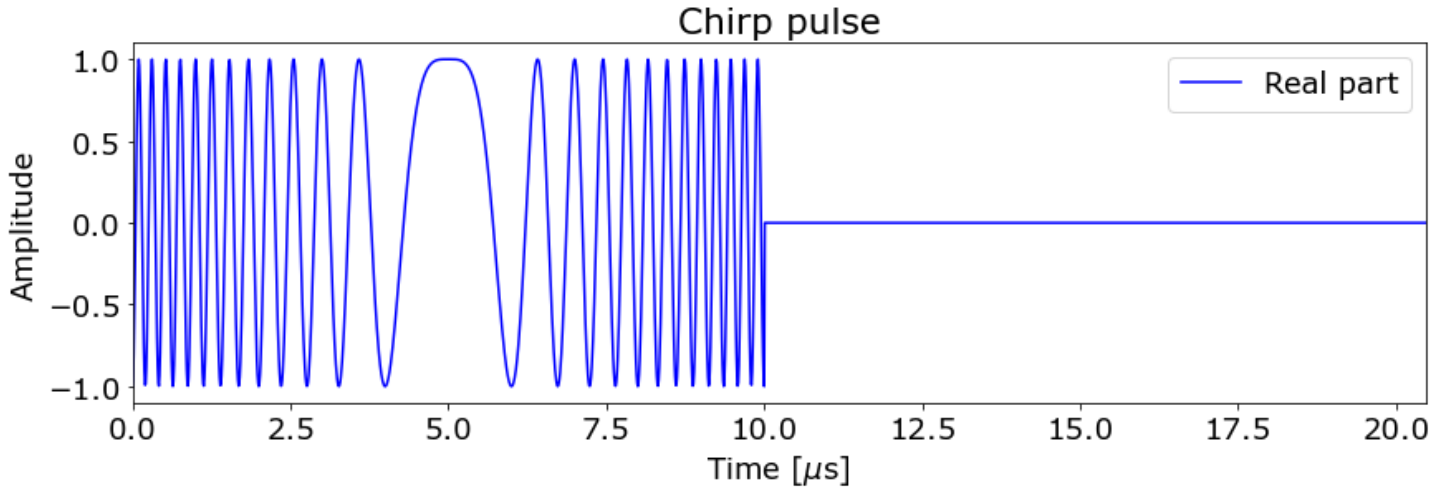
Pratically in coding cross-correlation, we would prefer to do Fourier transforms on the two waveforms first, then multiply them in frequency domain.

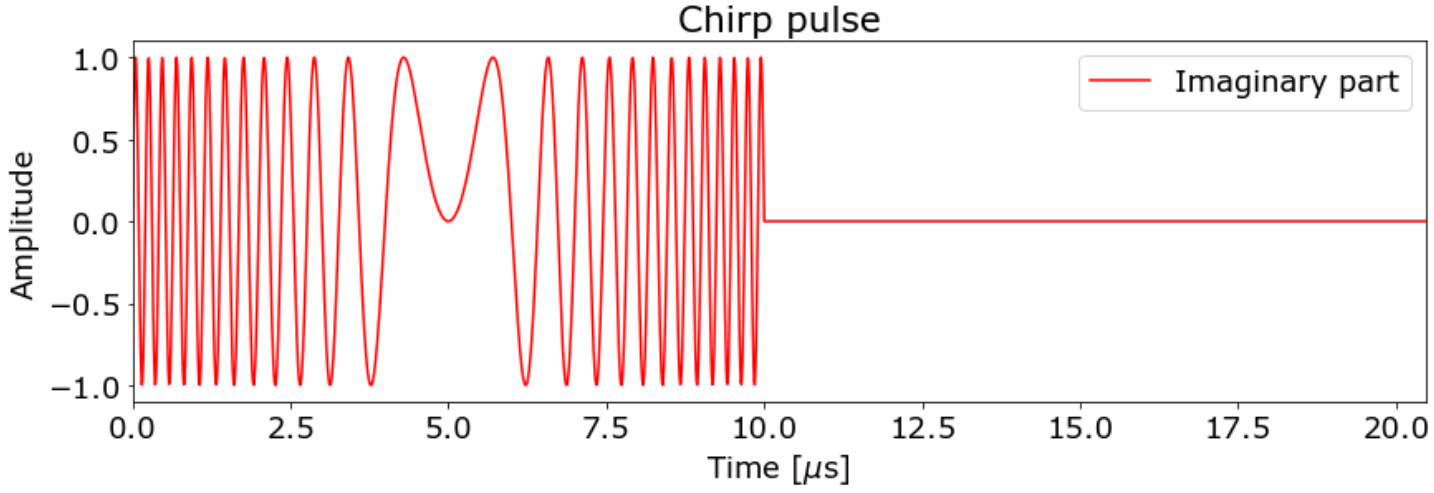
In [ ]:

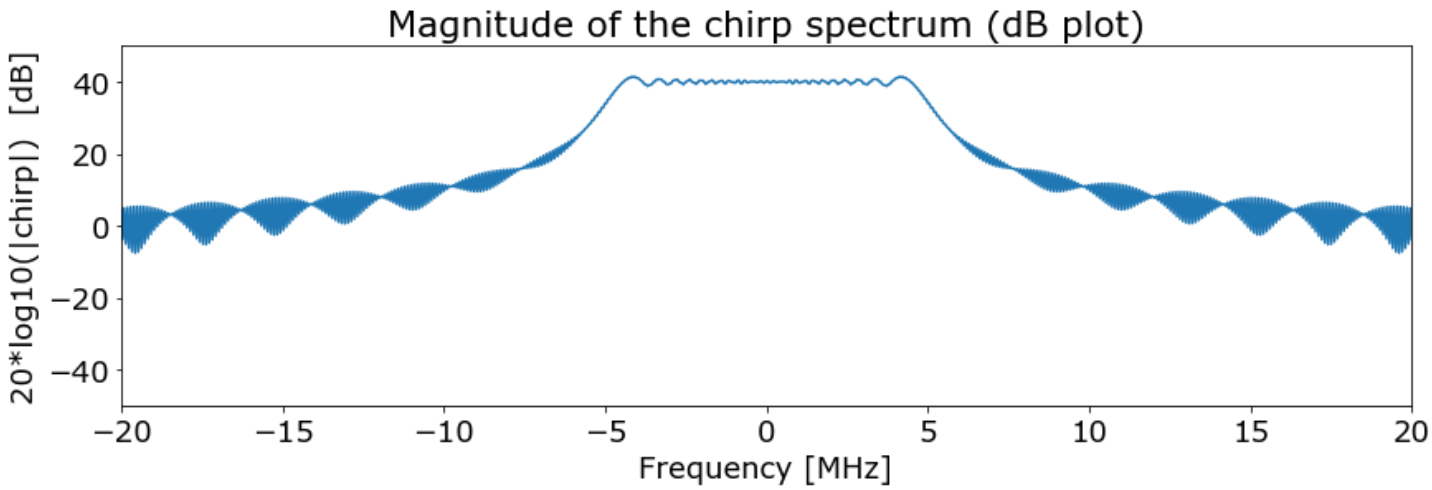
```
## Problem 1 (a): Make a dB plot of the chirp spectrum

# Set the chirp parameters
N     = 2048        # number of samples (include padded zeros)
slope = 1e12        # slope, Hz/s
tau   = 1e-5        # pulse length, s
fs    = 1e8         # sampling freq, Hz
fc    = 0           # center freq, Hz
BW    = slope * tau # bandwidth [Hz]

# create the chirp
chirp, t = makechirp(N, slope, tau, fs)

# spectrum of the chirp
chirp_fft = np.fft.fft(chirp)


# plot the chirp in time domain
plt.figure(figsize=[14,4])
```

```
plt.plot(t*1e6, np.real(chirp), c='b', label='Real part')
plt.legend(loc='upper right')
plt.xlim(0, np.max(t*1e6 ))
plt.title('Chirp pulse')
plt.xlabel(r'Time [$\mu$s]')
plt.ylabel('Amplitude')
plt.show()

plt.figure(figsize=[14,4])
plt.plot(t*1e6, np.imag(chirp), c='r', label='Imaginary part')
plt.legend(loc='upper right')
plt.xlim(0, np.max(t*1e6 ))
plt.title('Chirp pulse')
plt.xlabel(r'Time [$\mu$s]')
plt.ylabel('Amplitude')
plt.show()



# calculate the dB spectrum of the chirp
chirp_mag = 20*np.log10(np.abs(chirp_fft))
freq      = np.linspace(-fs/2, fs/2, N)    # Frequency axis

plt.figure(figsize=[14,4])
plt.plot(freq*1e-6, np.fft.fftshift(chirp_mag))
plt.title('Magnitude of the chirp spectrum (dB plot)')
plt.xlim((fc-2*BW)*1e-6, (fc+2*BW)*1e-6)
plt.ylim(-50,50)
plt.xlabel('Frequency [MHz]')
plt.ylabel('20*log10(|chirp|)  [dB]')
plt.show()
```

```
Chirp starts from 0 samples, 0.0 mu s
```







Note:

The theoretical bandwidth is BW = $s\tau$ = 10^{12} Hz/s * $10^{-5}s$ = $10^{7}Hz$ = 10MHz. From the plot, we can see that the bandwidth is close to 10MHz, since the width of the peak of the spectrum is from -5MHz to 5MHz, or 10MHz width.

Problem 1 (b):

Now we will compress the chirp by itself. Apply the cross-correlation by itself is also called "auto-correlation". Pratically in coding cross- or auto-correlation, we would prefer to do Fourier transforms on the two waveforms first, then multiply them in frequency domain.

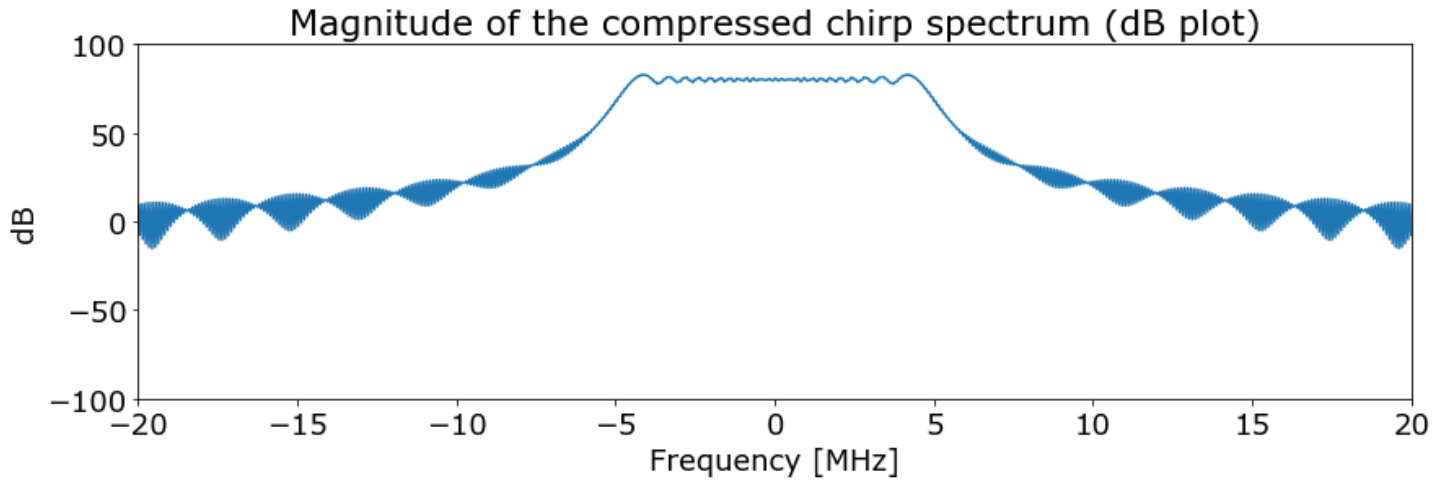
In time domain: $i(t) = r(t) \star r(t)$

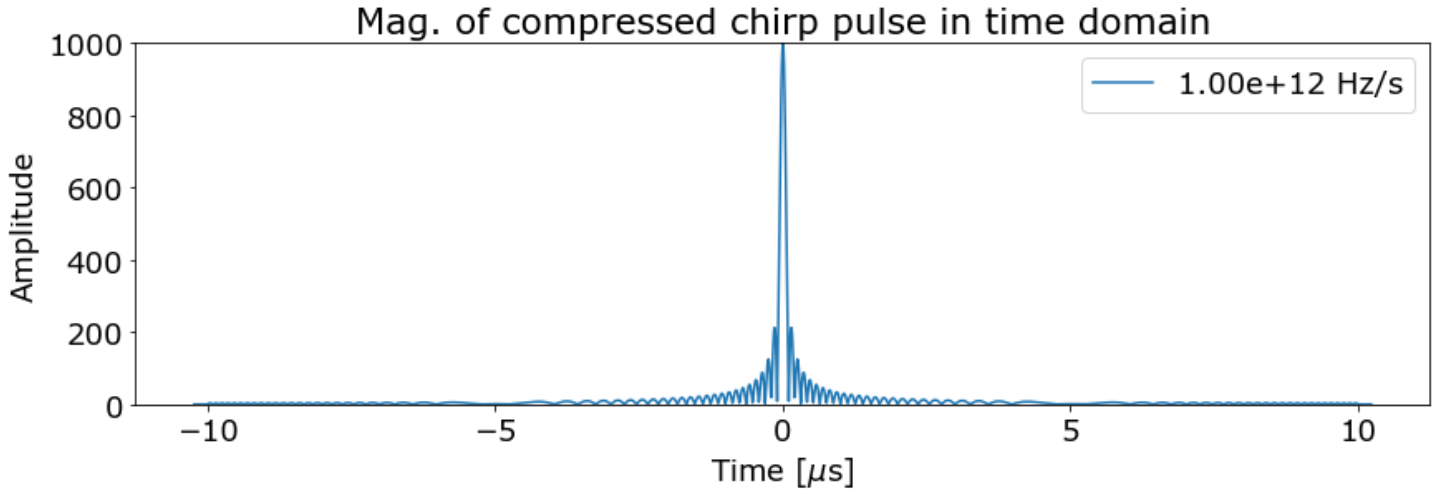
In frequency domain (do this): $I(f) = R(f) * R(f)$

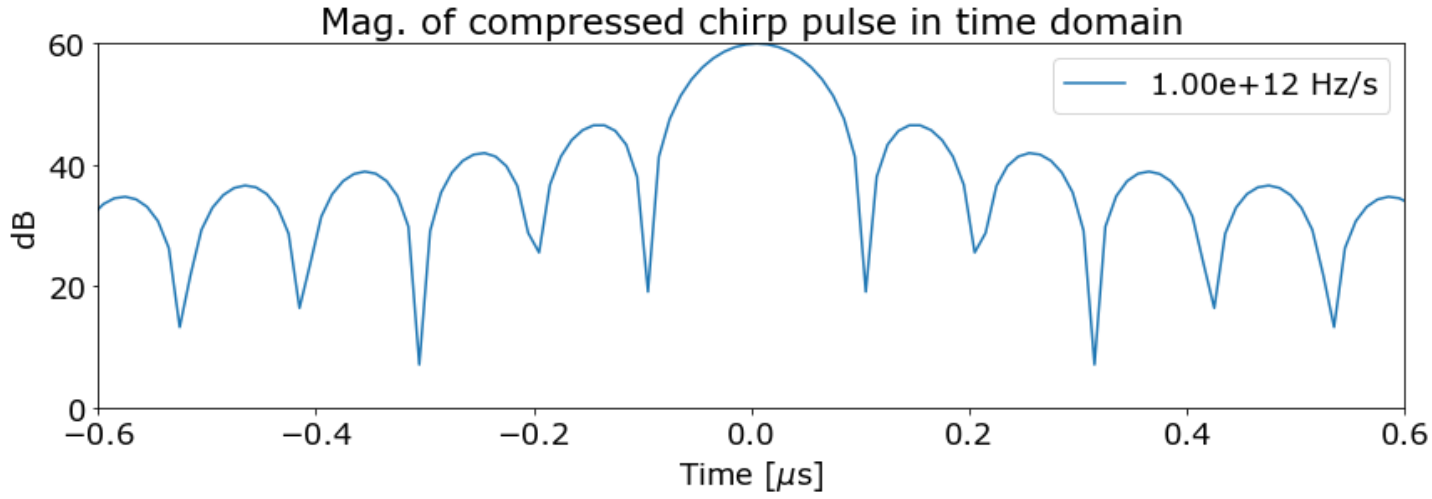
In [ ]:

```
## Problem 1 (b): Compress the chirp with a perfect reference signal

# Compress the chirp using matched filter
chirp_fft_compress = chirp_fft * np.conjugate(chirp_fft)


# calculate the dB spectrum of the signal
chirp_compress_mag = 20*np.log10(np.abs(chirp_fft_compress))

plt.figure(figsize=[14,4])
plt.plot(freq*1e-6, np.fft.fftshift(chirp_compress_mag))
plt.title('Magnitude of the compressed chirp spectrum (dB plot)')
plt.xlim((fc-2*BW)*1e-6, (fc+2*BW)*1e-6)
plt.ylim(-100,100)
plt.xlabel('Frequency [MHz]')
plt.ylabel('dB')
plt.show()


# compressed chirp in time domain
chirp_compress = np.fft.ifft(chirp_fft_compress)

plt.figure(figsize=[14,4])
plt.plot((t-np.mean(t))*1e6, np.fft.fftshift(np.abs(chirp_compress)), label='{:.2e} Hz/s'.format(slop
plt.legend(loc='upper right')
plt.title('Mag. of compressed chirp pulse in time domain')
plt.xlabel(r'Time [$\mu$s]')
plt.ylabel('Amplitude')
plt.ylim(0,1000)
plt.show()


# See the lobes at zoom-in
chirp_compress_mag_time = np.fft.fftshift(20*np.log10(np.abs(chirp_compress) + 1e-30))
```

```
plt.figure(figsize=[14,4])
plt.plot((t-np.mean(t))*1e6, chirp_compress_mag_time, label='{:.2e} Hz/s'.format(slope))
plt.legend(loc='upper right')
plt.title('Mag. of compressed chirp pulse in time domain')
plt.xlabel(r'Time [$\mu$s]')
plt.ylabel('dB')
plt.xlim(-0.6,0.6)
plt.ylim(0,np.max(chirp_compress_mag_time))
plt.show()
```





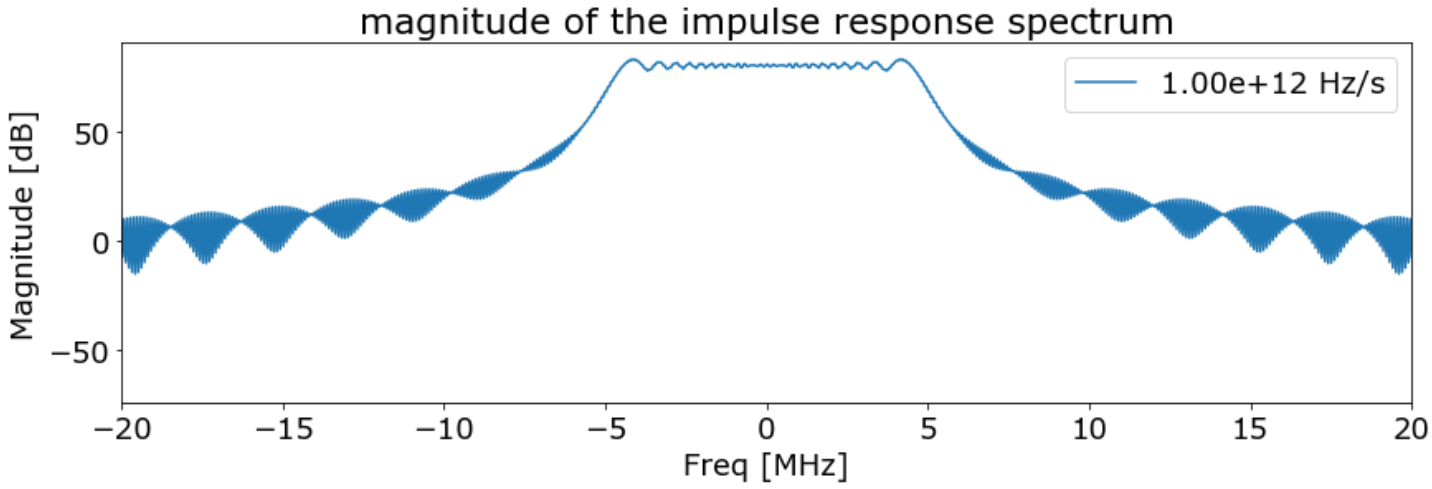


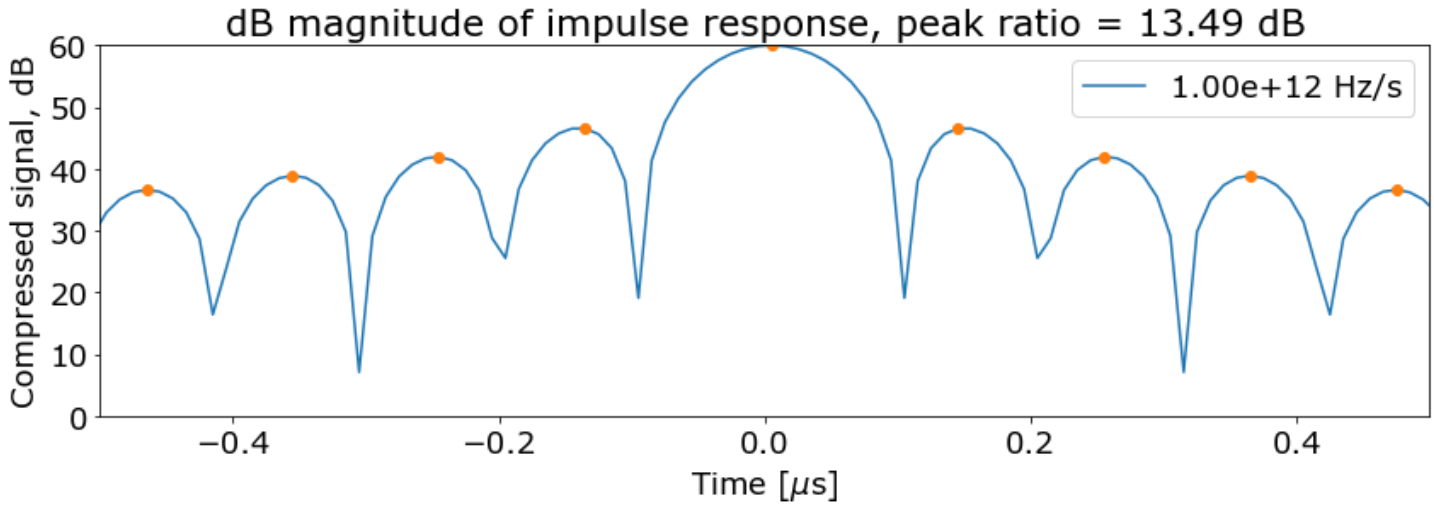
In [ ]:

```
## Problem 1 (c): What is the performance if the reference chirp is not perfectly like the signal

# a perfect ref (like in (b))
slope = 1.00e12    # slope, Hz/s
quick_evaluate_compress(chirp, N, slope, tau, fs, explot=True)
```

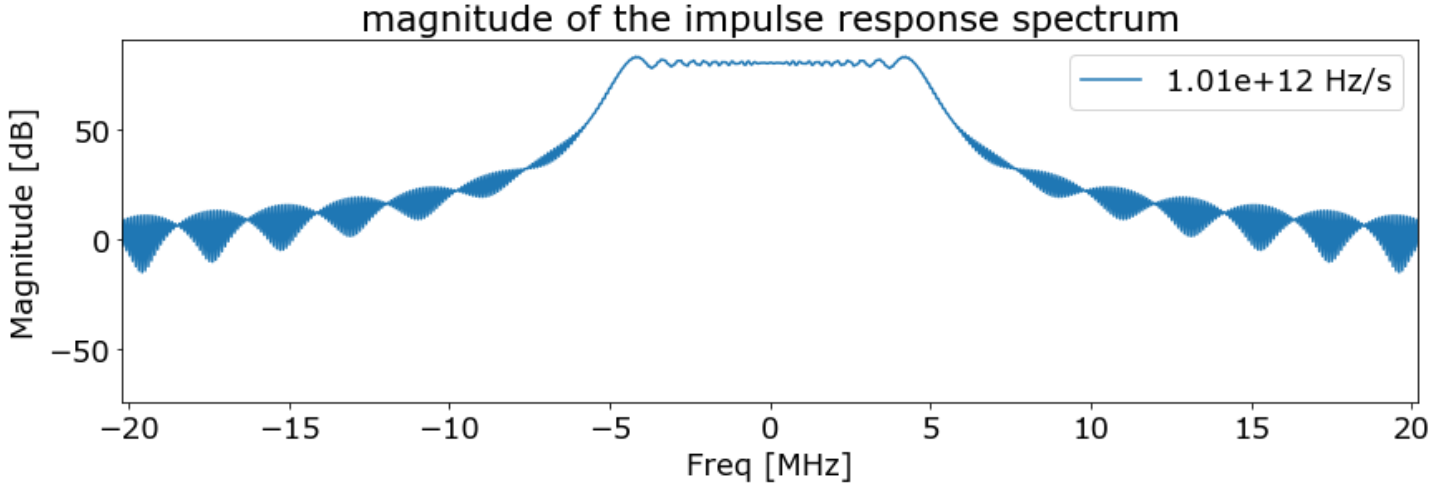
```
# ref slope is off by 1% compared to the signal
slope = 1.01e12    # slope, Hz/s
quick_evaluate_compress(chirp, N, slope, tau, fs, explot=True)

# ref slope is off by 3% compared to the signal
slope = 1.03e12    # slope, Hz/s
quick_evaluate_compress(chirp, N, slope, tau, fs, explot=True)
```

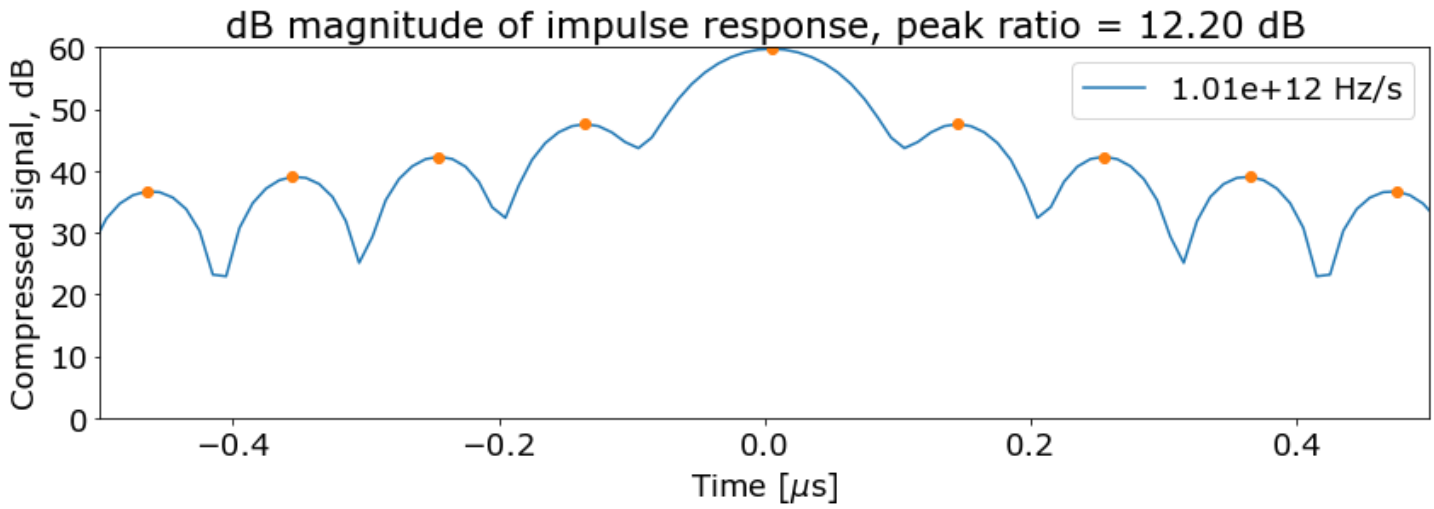
```
Chirp starts from 0 samples, 0.0 mu s
```



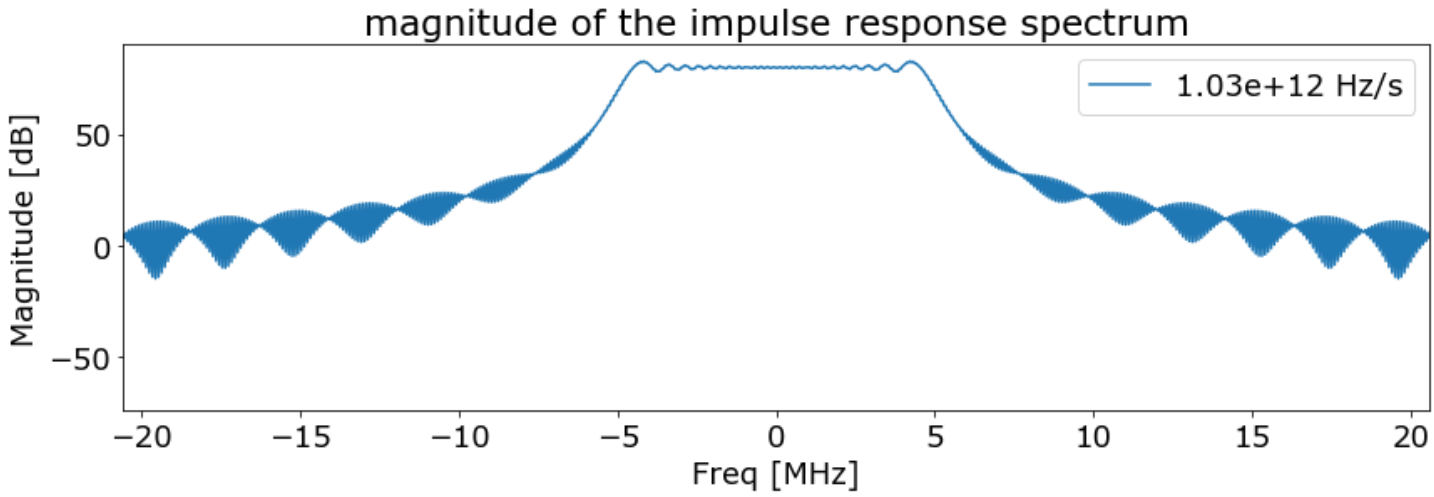


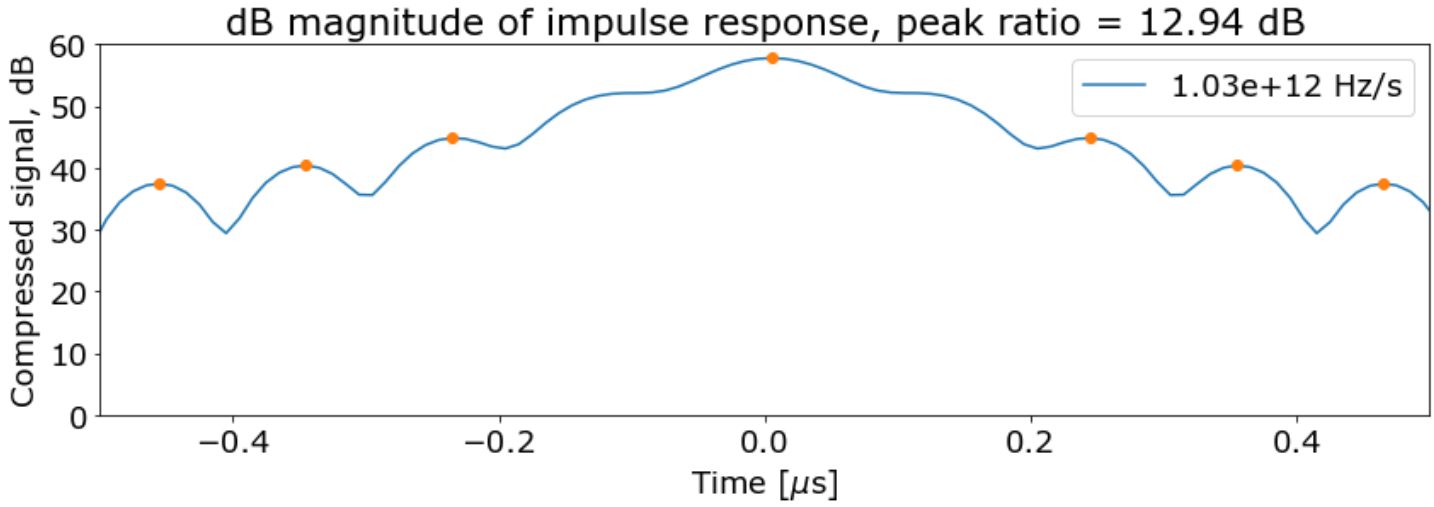

```
Chirp starts from 0 samples, 0.0 mu s
```






```
Chirp starts from 0 samples, 0.0 mu s
```





If the referenced chirp is perfect, the peak ratio is about 13.5 dB.

If the referenced chirp slope is off by 1%, the peak ratio reduced to 12.2 db.

If the referenced chirp slope is off by 3%, the peak ratio reduced to ~5 db. Notice that the peak of the first sidelobeis not detected. So the number in the title (12.94 dB) in the third plot is the wrong number.

Problem 2: Separating multiple chirps with radar compressing

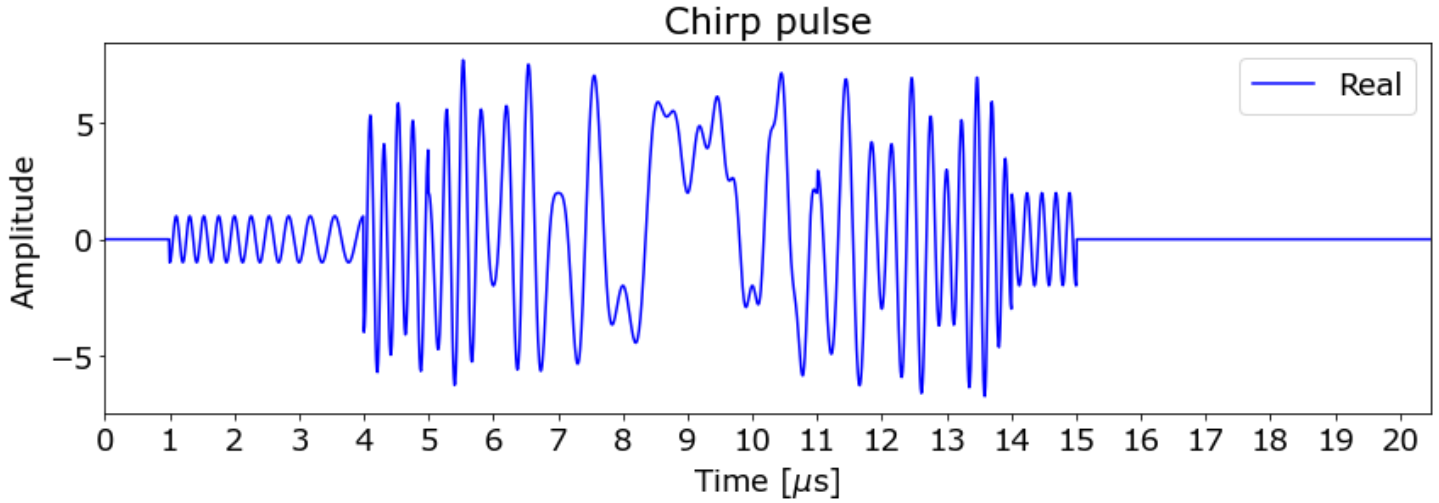
Using the chirp parameters given in (1) above, create a signal record consisting of the sum of 3 chirps, one beginning at location 100 in the array with amplitude 1, the second beginning at location 400 with amplitude 5, and the third beginning

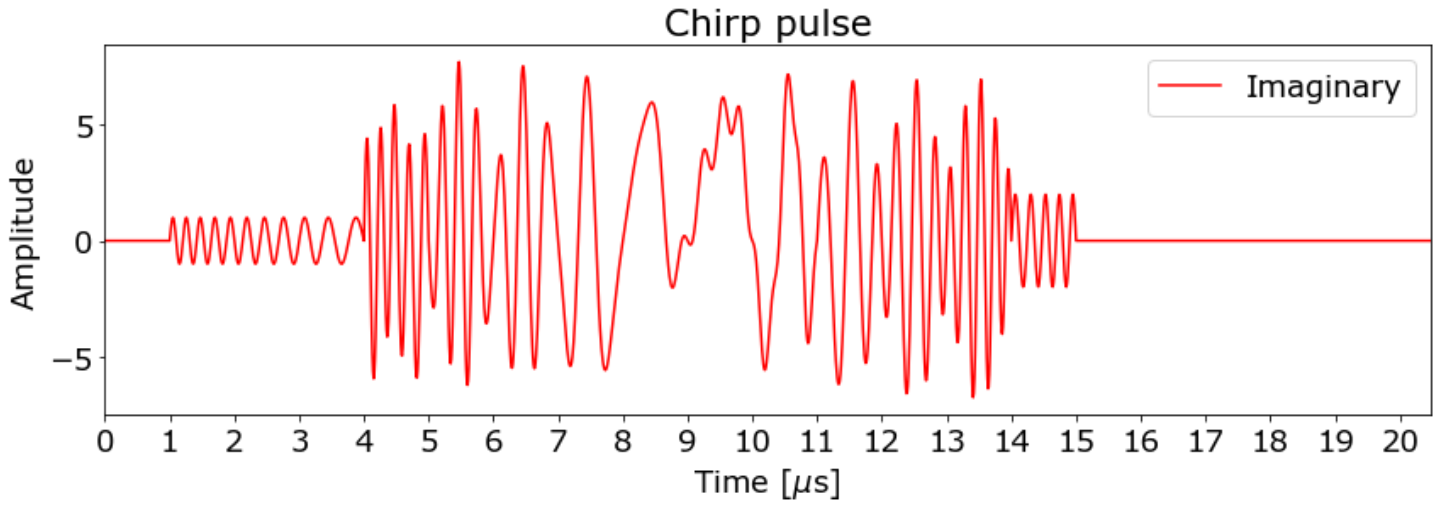
at location 500 with amplitude 2.

In [ ]:

```
## Problem 2 (a): Create a composite chrip and plot it

slope = 1.00e12    # slope, Hz/s

chp1, t = makechirp(N, slope, tau, fs, start=100)
chp2, t = makechirp(N, slope, tau, fs, start=400)
chp3, t = makechirp(N, slope, tau, fs, start=500)

chp_all = chp1 + 5*chp2 + 2*chp3

plt.figure(figsize=[14,4])
plt.plot(t*1e6, np.real(chp_all), c='b', label='Real')
plt.legend(loc='upper right')
plt.xlim(0, np.max(t*1e6))
plt.xticks(np.arange(0, np.max(t*1e6)))
plt.title('Chirp pulse')
plt.xlabel(r'Time [$\mu$s]')
plt.ylabel('Amplitude')
plt.show()

plt.figure(figsize=[14,4])
plt.plot(t*1e6, np.imag(chp_all), c='r', label='Imaginary')
plt.legend(loc='upper right')
plt.xlim(0, np.max(t*1e6))
plt.xticks(np.arange(0, np.max(t*1e6)))
plt.title('Chirp pulse')
plt.xlabel(r'Time [$\mu$s]')
plt.ylabel('Amplitude')
plt.show()
```

```
Chirp starts from 100 samples, 1.0 mu s
Chirp starts from 400 samples, 4.0 mu s
Chirp starts from 500 samples, 5.0 mu s
```



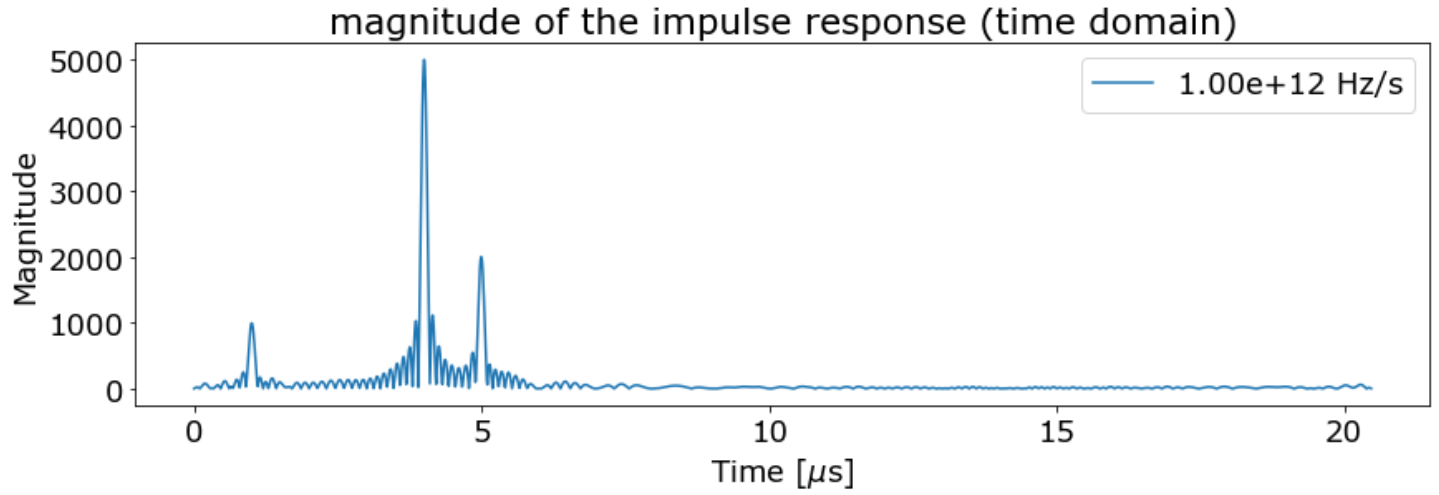


We can definitely tell that we do not have a standard, single chirp, but it would be very difficult to distinguish between the individual signals in the time domain because the signals overlap. Chirps starting at location 100, 400, and 500 will interfere with each other.

In [ ]:

```
## Problem 2 (b): Try compress that composite chirp with a perfect reference

slope = 1.00e12    # slope, Hz/s
quick_evaluate_compress(chp_all, N, slope, tau, fs, liplot=True)
```

```
Chirp starts from 0 samples, 0.0 mu s
```



After applying the correlation with a perfect reference chirp, we can distinguish the individual chirps easily. There are three peaks, each corresponds to the chirp at 1, 4, and 5 μ s, which are the chirps starting at 100, 400, and 500 samples. The relative amplitude of the peak also tells us about the relative amplitude of the chirps. That is:

first chrip : second chirp : third chirp = $1:5:4$

Problem 3: Play with the actual ERS data

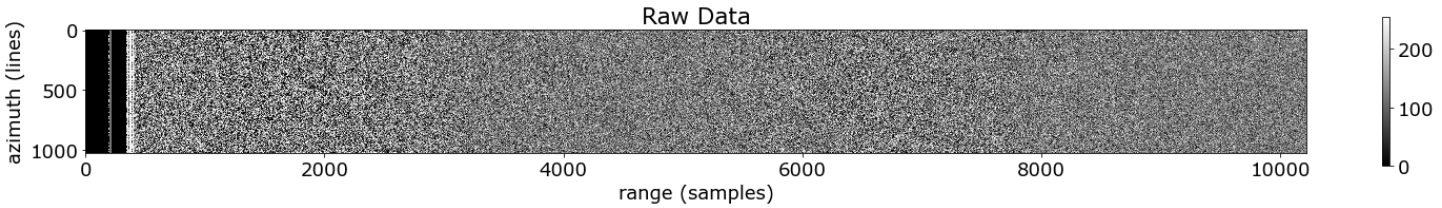
Download to your computer a data file containing actual data from the ERS satellite. The data file is available on the class web site under the Homework tab, for Homework 2. The file name is “ersdata”.

We will need to use some python modules to read the binary file `ersdata` . I found `np.frombuffer` works, so I use that here. You may use other ways.

In [ ]:

```
## Problem 3 (a): Display the byte file on the screen

# define some line numbers of the file
```

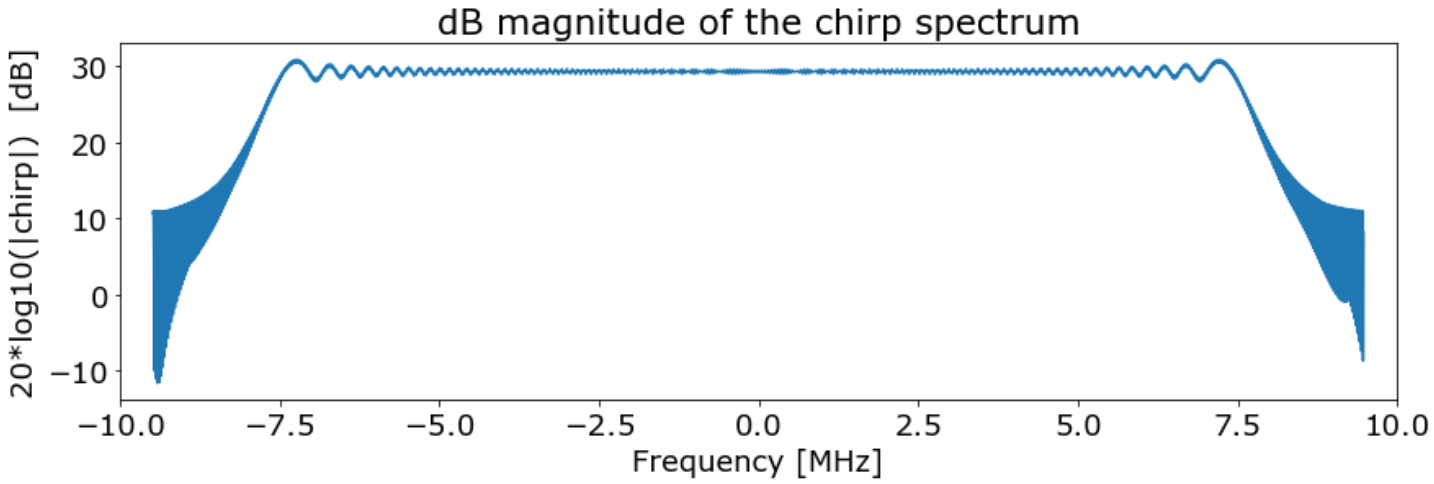
```
nhdr   = 412
nsamp  = 10218
nlines = 1024

# Read the file
ERS_file = './ersdata'
with open(ERS_file, 'rb') as fh:
    load_arr = np.frombuffer(fh.read(), dtype=np.uint8).reshape((nlines, nsamp))


# Adjust the data by multiplying by 8, so it is discritized coarsely for easy visualization
data = np.array(load_arr)
data[:,nhdr:] = 8 * data[:,nhdr:]

# plot the byte file
plt.figure(figsize=[26,26])
im = plt.imshow(data, cmap='gray', interpolation='none')
plt.colorbar(im, shrink=0.1)
plt.title('Raw Data')
plt.xlabel('range (samples)')
plt.ylabel('azimuth (lines)')
plt.show()
```



In [ ]:

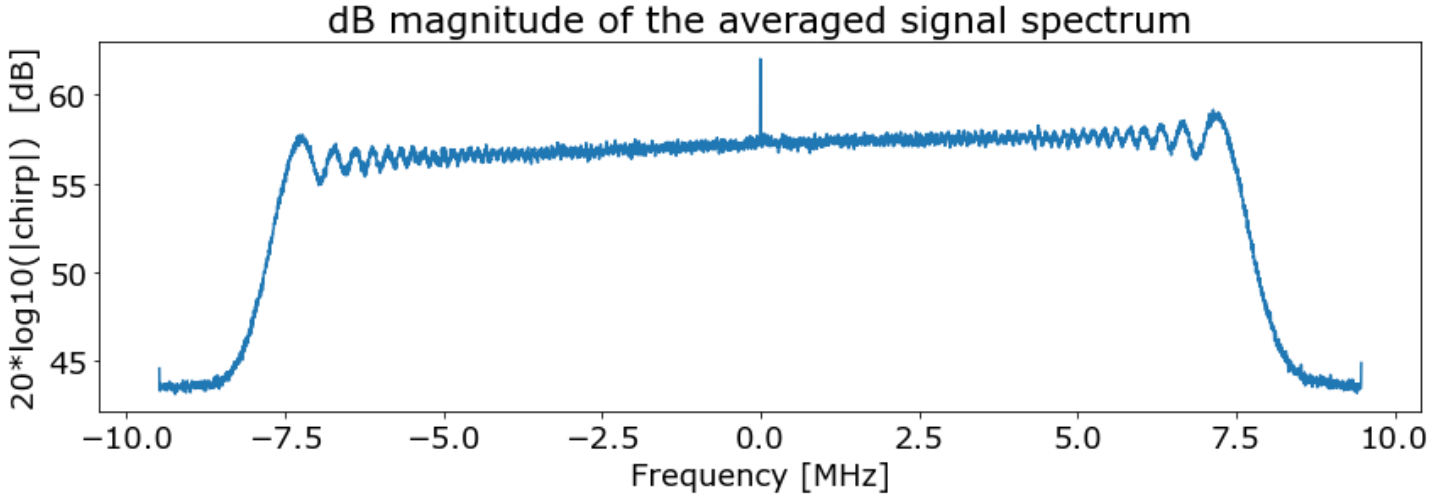
```
## Problem 3 (b): Create a ERS chirp

# given chirp parameters
N     = int((nsamp-nhdr)/2)   # number of complex samples
slope = 4.189166e11           # slope in [Hz/s]
tau   = 37.12e-6              # pulse length [s]
fs    = 18.96e6               # sample rate [Hz]
fc    = 0                     # center Frequency

# create chirp
chirp, t = makechirp(N, slope, tau, fs, fc=fc)

# Transform it to a spectrum
chirp_fft = np.fft.fft(chirp)

# calculate the dB spectrum of the signal
chirp_mag = 20*np.log10(np.abs(chirp_fft))
freq      = np.arange(-fs/2,fs/2,fs/N) # Frequency axis

# plot the spectrum
plt.figure(figsize=[14,4])
plt.plot(freq*1e-6, np.fft.fftshift(chirp_mag))
plt.title('dB magnitude of the chirp spectrum')
plt.xlim(-10,10)
plt.xlabel('Frequency [MHz]')
plt.ylabel('20*log10(|chirp|)  [dB]')
plt.show()
```

```
Chirp starts from 0 samples, 0.0 mu s
```



In [ ]:

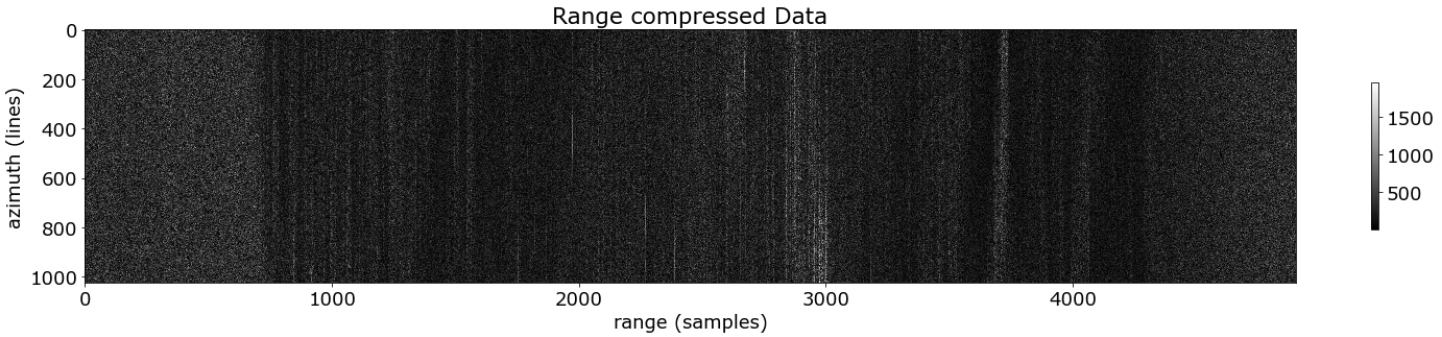
```
## Problem 3 (c): ERA data average across the azimuth axis

# ERS I/Q data has 5 bit (from 0 to 2^5-1 = 31), thus the theoretical mean is 15.5
ers_mean = 15.5


# read the complex data from data array
sig      = np.array(load_arr)[:, nhdr:]
sig_odd  = sig[:,  ::2] - ers_mean    # real part
sig_even = sig[:, 1::2] - ers_mean    # imaginary part
sig      = sig_odd + 1j*sig_even

# transform
sig_fft = np.fft.fft(sig, axis=1)

# averaged over the azimuth direction
# (average the abs value of the spectrum, so that complex numbers will not cancel each other)
sig_avg = np.mean(np.abs(sig_fft), axis=0)

# get the magnitude and plot
sig_mag = 20*np.log10(np.abs(sig_avg))

plt.figure(figsize=[14,4])
plt.plot(freq*1e-6, np.fft.fftshift(sig_mag))
plt.title('dB magnitude of the averaged signal spectrum')
#plt.xlim(-10,10)
plt.xlabel('Frequency [MHz]')
plt.ylabel('20*log10(|chirp|)  [dB]')
plt.show()
```



The magnitude of the ERS spectrum averaged over the azimuth direction looks very similar to that of the spectrum of a chirp. They both have a plateau from -7.5 MHz to +7.5 MHz.

In [ ]:

```
## Problem 3 (d): Compress each range line and create a “range-compressed” image

# do matched filter all at once. (you can also do it row-by-row, then stack each processed row to for
comp, spec = matched_filter(sig, chirp)

# display the result
plt.figure(figsize=[26,26])
im = plt.imshow(np.abs(comp), cmap='gray', interpolation='none')
plt.colorbar(im, shrink=0.1)
plt.title('Range compressed Data')
plt.xlabel('range (samples)')
plt.ylabel('azimuth (lines)')
plt.savefig('./hw2_rangeCompressed_ERS.tiff', format = 'tif')
plt.show()
```



4. I/Q and offset video processing

Consider a chirp signal with the following parameters:

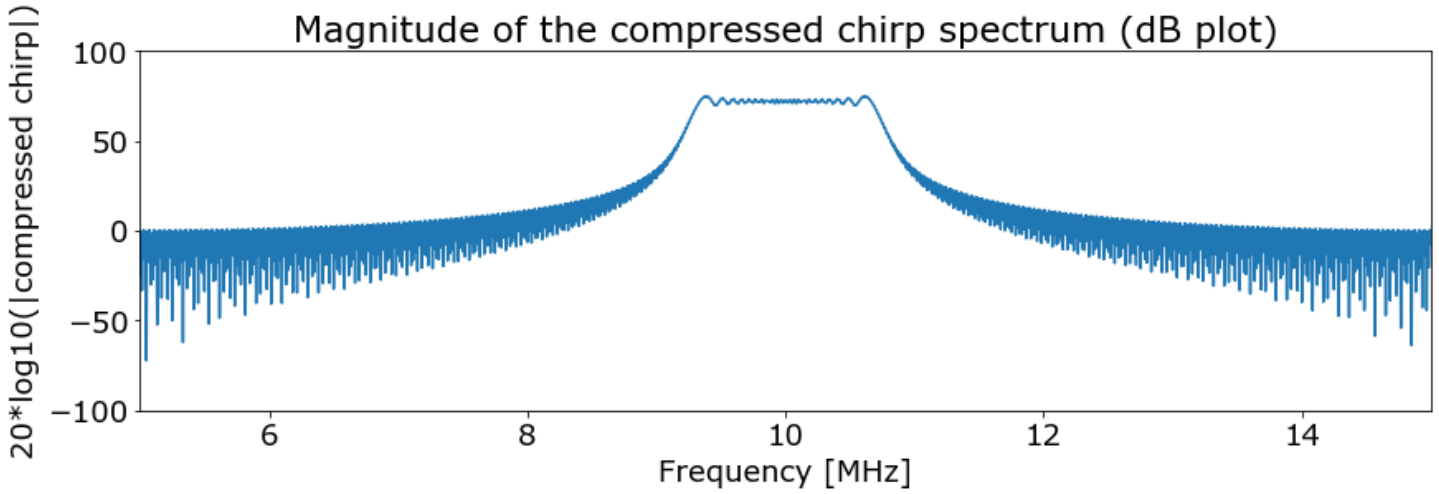
- Slope (s) = $10^{11}\ Hz\, s^{-1}$
- Pulse length (τ) = 30 μs = 3×10^{-5} s
- Sample rate (fs) = 20 MHz = $2 \times 10^{8}\ Hz$
- center freq (fc) = 10 MHz = $10^{8}\ Hz$

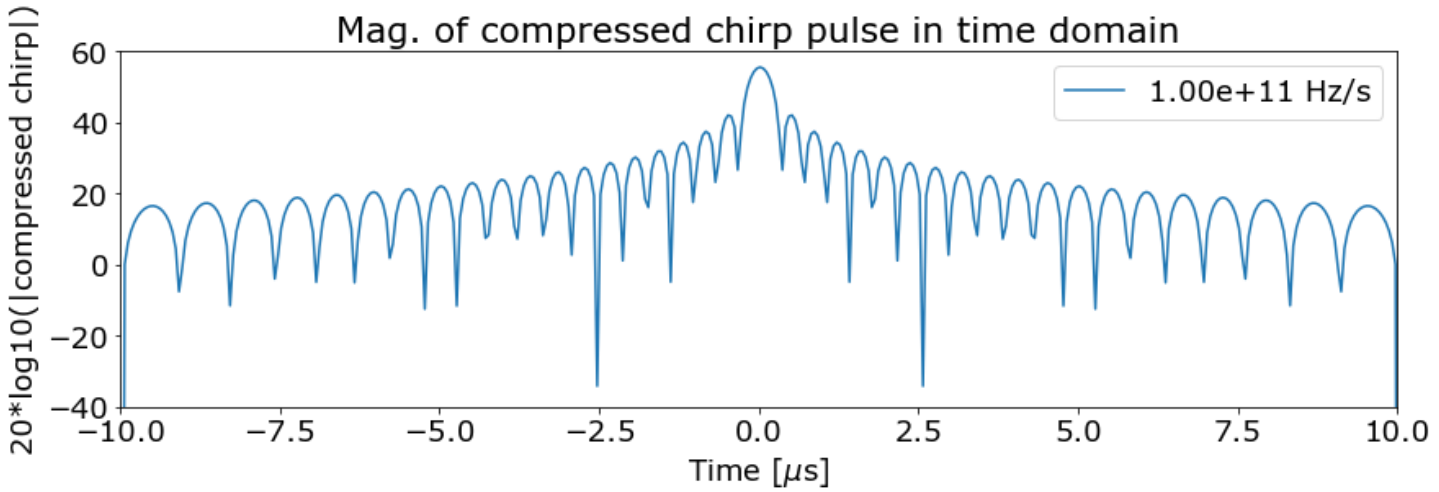
In [ ]:

```
## Problem 4 (a): Assume an I/Q system and that the chirp is centered at the carrier fc. Compress the

# define parameters
N     = 2048     # number of samples (include padded zeros)
slope = 1e11     # slope, Hz/s
tau   = 30e-6    # pulse length, s
fs    = 20e6     # sampling freq, Hz
fc    = 10e6     # center freq, Hz

# Make a reference chirp and a signal which looks identical to the chirp
ref, t = makechirp(N, slope, tau, fs, fc)
sig, t = makechirp(N, slope, tau, fs, fc)

# compress the signal
comp, spec = matched_filter(sig, ref)

freq = np.linspace(fc-fc/2,fc+fc/2,N) # Frequency axis

# calculate the dB spectrum of the signal
spec_mag = 20*np.log10(np.abs(spec))

# Spectrum
plt.figure(figsize=[14,4])
plt.plot(freq*1e-6, spec_mag)
plt.title('Magnitude of the compressed chirp spectrum (dB plot)')
plt.xlim(5,15)
plt.ylim(-100,100)
plt.xlabel('Frequency [MHz]')
plt.ylabel('20*log10(|compressed chirp|)')
```

```
plt.show()


# Compressed magnitude in time domain
comp_mag = np.fft.fftshift(20*np.log10(np.abs(comp) + 1e-30))

plt.figure(figsize=[14,4])
plt.plot((t-np.mean(t))*1e6, comp_mag, label='{:.2e} Hz/s'.format(slope))
plt.legend(loc='upper right')
plt.title('Mag. of compressed chirp pulse in time domain')
plt.xlabel(r'Time [$\mu$s]')
plt.ylabel('20*log10(|compressed chirp|)')
plt.xlim(-10,10)
plt.ylim(-40, 60)
plt.show()
```

```
Chirp starts from 0 samples, 0.0 mu s
Chirp starts from 0 samples, 0.0 mu s
```





problem 4 (b)

According to the chirp parameters, the theoretical bandwidth can be computed:

$BW = s\tau = (10^{11})$ Hz/s $\times (30 \times 10^{-6})$ s $= 3 \times 10^{6}$ Hz $= 3$ MHz

The minimum chirp frequency is:

$f_{min} = f_c - \frac{BW}{2} = 10$ MHz $- \frac{3MHz}{2} = 8.5$ MHz

The maximum chirp frequency is:

$f_{max} = f_c + \frac{BW}{2} = 10$ MHz $+ \frac{3MHz}{2} = 11.5$ MHz

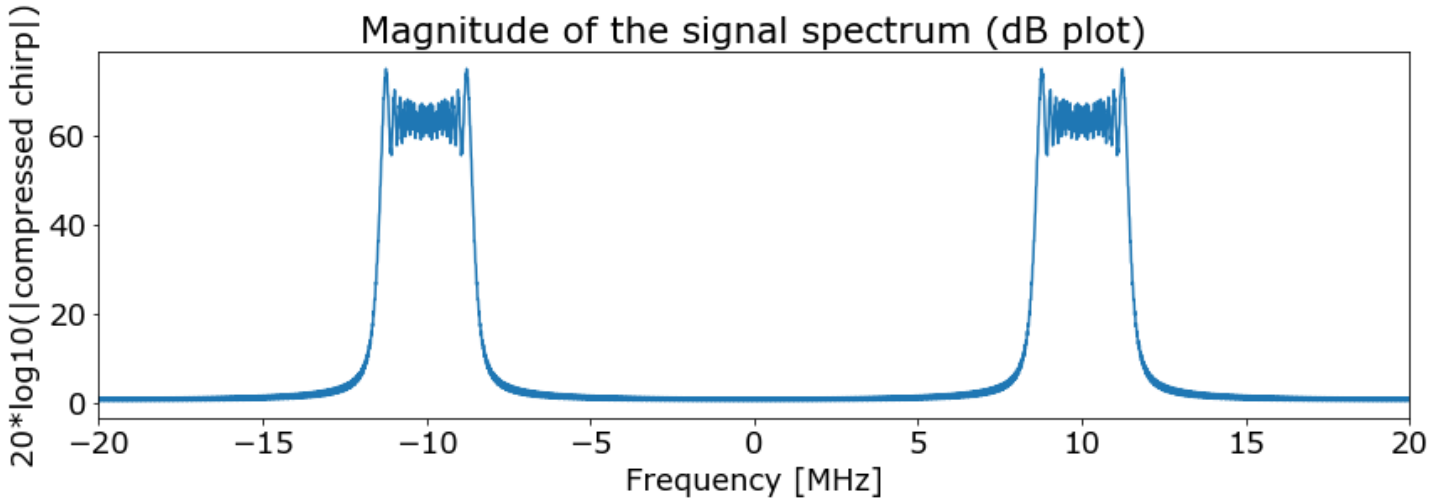
The impulse response of the offset video system should be identical to the impulse response for the I/Q system. Since we sampled at $2 \times fs$, we have conjugate symmetry, and can simply take the positive frequencies of the spectrum for the signal compression.

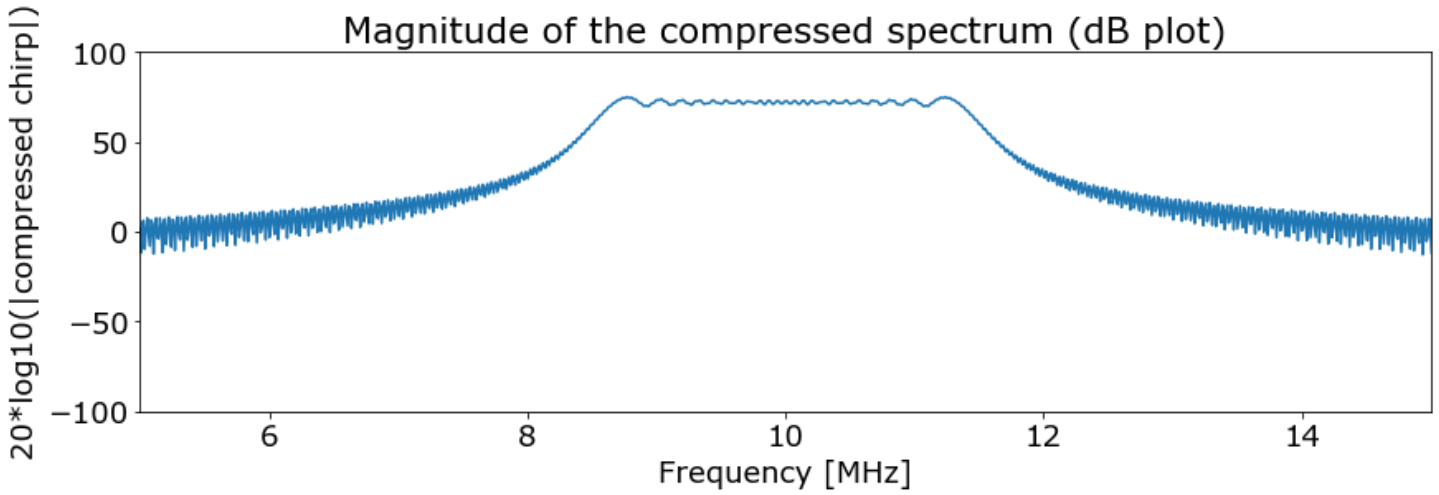
In [ ]:

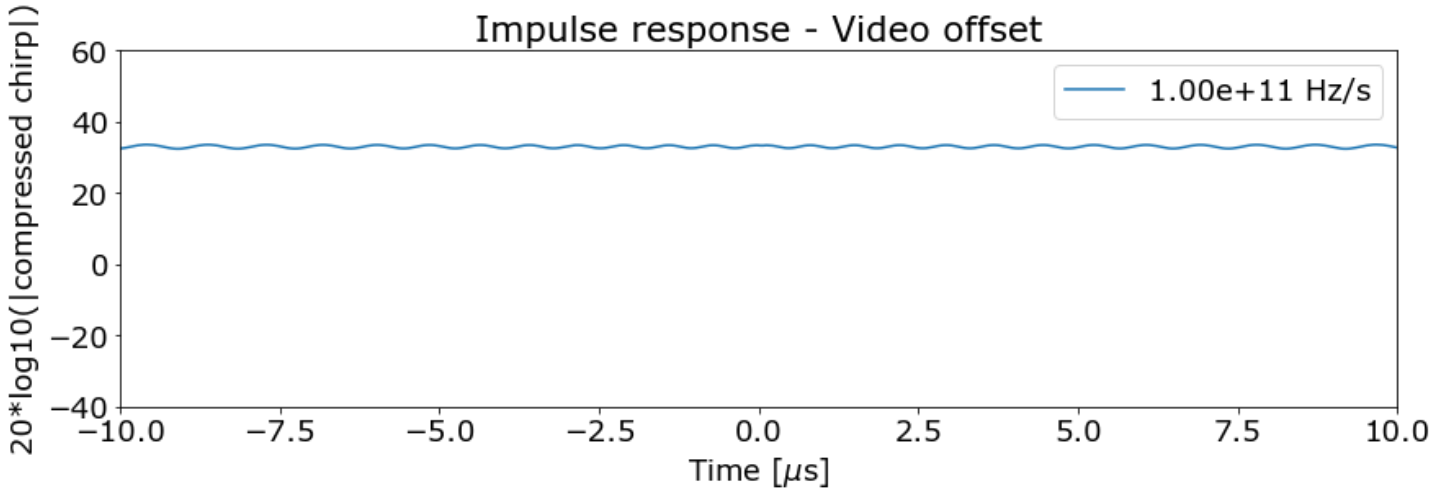
```
## Problem 4 (b): Assume we are using an offset video system.

# Make a ref chirp
N     = 2048     # number of samples (include padded zeros)
slope = 1e11     # slope, Hz/s
tau   = 30e-6    # pulse length, s
fs    = 20e6     # sampling freq, Hz
fc    = 10e6     # center freq, Hz
ref, t1  = makechirp(N, slope, tau, fs, fc)
ref_fft = np.fft.fft(ref)

# Make a signal that is received via an offset video system
N     = 2048  * 2   # number of samples (double the length since we will sample twice often)
slope = 1e11        # slope, Hz/s
tau   = 30e-6       # pulse length, s
fs    = 20e6  * 2   # sampling freq, Hz (sample the signal twice often because we will take only the
fc    = 10e6        # center freq, Hz
sig, t2 = makechirp(N, slope, tau, fs, fc)

# offset video algorithm
sig_fft      = np.fft.fft(np.real(sig))  # retreive only the real part of complex signal array
sig_fft_side = sig_fft[int(N/2):]        # use only the positive frequency side of the spectrum

# Do matched filter (compress the signal using the chirp)
spec         = sig_fft_side * np.conjugate(ref_fft)
comp         = np.fft.fftshift(np.fft.ifft(spec))
spec_mag     = 20*np.log10(np.abs(spec) + 1e-30)
comp_mag     = 20*np.log10(np.abs(comp) + 1e-30)

freq  = np.linspace(-fc-fc,fc+fc,N) # Frequency axis for the ref chirp
freq2 = freq[int(N/2):]             # Frequency axis for the video offset signal (take only positive


# Chirp spectrum
plt.figure(figsize=[14,4])
plt.plot(freq*1e-6, np.abs(sig_fft))
plt.title('Magnitude of the signal spectrum (dB plot)')
plt.xlim(-20,20)
#plt.ylim(-100,100)
plt.xlabel('Frequency [MHz]')
plt.ylabel('20*log10(|compressed chirp|)')
plt.show()


# Compressed spectrum
plt.figure(figsize=[14,4])
plt.plot(freq2*1e-6, spec_mag)
plt.title('Magnitude of the compressed spectrum (dB plot)')
plt.xlim(5,15)
plt.ylim(-100,100)
plt.xlabel('Frequency [MHz]')
plt.ylabel('20*log10(|compressed chirp|)')
plt.show()


# Compressed magnitude in time domain
plt.figure(figsize=[14,4])
plt.plot((t1-np.mean(t1))*1e6, comp_mag, label='{:.2e} Hz/s'.format(slope))
plt.legend(loc='upper right')
plt.title('Impulse response - Video offset')
plt.xlabel(r'Time [$\mu$s]')
plt.ylabel('20*log10(|compressed chirp|)')
plt.xlim(-10,10)
```

```
plt.ylim(-40, 60)
plt.show()
```

```
Chirp starts from 0 samples, 0.0 mu s
Chirp starts from 0 samples, 0.0 mu s
```







Note:

- The signal spectrum is two-sided and symmetric because we only take the real part of it.
- Check the magnitude of the compressed signal, the bandwidth starts from ~8.5 MHz to ~11.5 MHz, centered at the $f_c \sim 10$ MHz . This is what we expected.
- Finally, the inpulse response here looks just like the impulse response for the I/Q system. But it has slight numerical differences to the I/Q system at very small numbers.

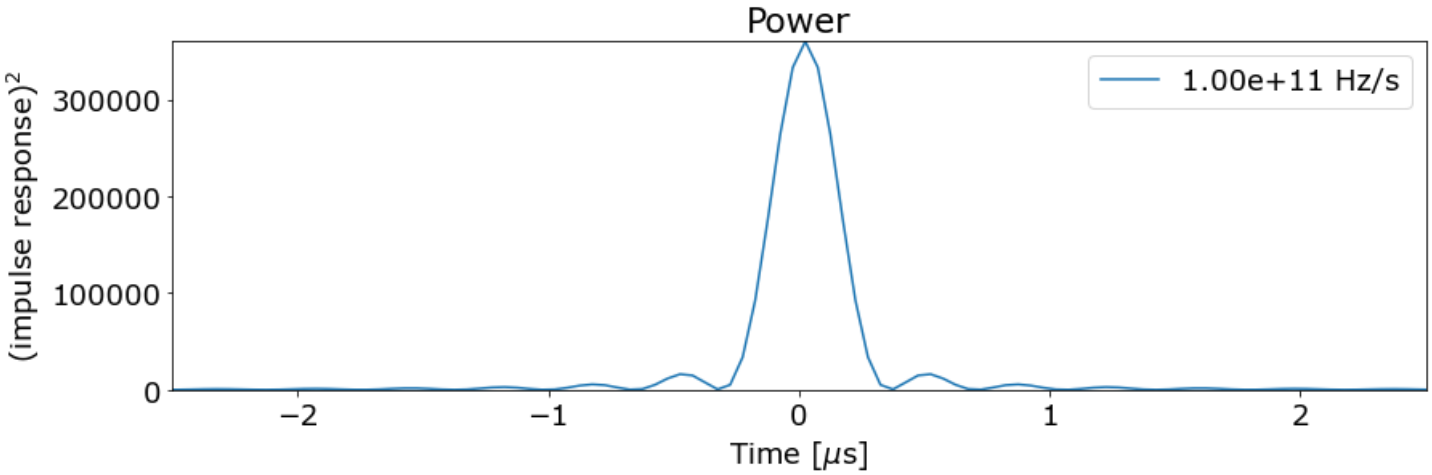
Problem 5: Sidelobe filtering

Problem 5 (a): What are the peak and integrated sidelobe levels for the chirp in problem 4 above?

In [ ]:

```
## Problem 5 (a): PSLR and ISLR

# Chirp parameters
N     = 2048     # number of samples (include padded zeros)
slope = 1e11     # slope, Hz/s
tau   = 30e-6    # pulse length, s
fs    = 20e6     # sampling freq, Hz
fc    = 10e6     # center freq, Hz

# make ref chirp
ref, t  = makechirp(N, slope, tau, fs, fc)
freq    = np.arange(-fs/2,fs/2,fs/N) # Frequency axis

# impulse response (compress the chirp itself)
impres, spec = matched_filter(ref, ref)
impres_pow   = np.fft.fftshift(np.abs(impres)**2)
impres_mag   = np.fft.fftshift(20*np.log10(np.abs(impres) + 1e-30))
spec_mag     = 20*np.log10(np.abs(spec)   + 1e-30)


# Peak ratio = (main_lobe) - (first_side_lobe)
peaks, _ = find_peaks(impres_mag)
idx = np.argsort(impres_mag[peaks])[::-1]
ratio = impres_mag[peaks[idx[0]]]-impres_mag[peaks[idx[1]]]

print('Main lobe / sidelobe ratio = {}'.format(ratio))

plt.figure(figsize=[14,4])
plt.plot((t-np.mean(t))*1e6, impres_pow, label='{:.2e} Hz/s'.format(slope))
plt.legend(loc='upper right')
plt.title('Power')
plt.xlabel(r'Time [$\mu$s]')
plt.ylabel(r'(impulse response)$^2$')
plt.xlim(-2.5,2.5)
plt.ylim(0,np.max(impres_pow))
plt.show()
```

```
Chirp starts from 0 samples, 0.0 mu s
Main lobe / sidelobe ratio = 13.491550390857725
```



Properties of the impulse response:

Peak sidelobe ratio (PSLR): (this is just the mainlob/sidelobe ratio expressed in dB)

$$PLSR = \frac{P_{1st\_sidelobe}}{P_{mainlobe}}$$

P is the power of the impulse response, P = (impuse response)^2

. .

Integrated sidelobe ratio (ISLR):

$$ILSR = \frac{E_{all\_sidelobes}}{E_{mainlobe}}$$

E is the energy (or the integrated power across a window), which is the integrated area below the power curve of impulse response. E = Σ[(impuse response)^2]

The energy of the mainlob is the integrated area around the mainlobe (in between the first null points). The energy of all the sidelobes are the integrated area outside the first null point. Since the impulse response is symmetric with respect to the peak, then I only need to calculate this ratior for one side of the curve.

In [ ]:

```
## Define a functino that calculates the properties

def impres_property(impres_pow, vocal=False):
    """
    Computes ISLR and PSLR of a given impulse response (power)
    It requires that the impresрw is symmetric about the peak (do the np.fft.fftshift(impres_pow) bef
    """
    n = len(impres_pow)

    # find position of the main peak
    peak_id = np.argmax(impres_pow)

    # find position of the second peak (1st sidelobe)
    peaks, _ = find_peaks(impres_pow)
    side_id  = peaks[np.where(peaks>peak_id)[0][0]]
    #side_id  = peaks[np.argsort(impres_pow[peaks])[::-1][1]]

    # compute peak-firstsidelobe ratio (PSLR)
    PSLR = impres_pow[side_id] / impres_pow[peak_id]
    PSLR = 10*np.log10(PSLR)

    # find first null position
    for i in np.arange(peak_id, n):
        if (impres_pow[i]<impres_pow[i-1]) and (impres_pow[i]<impres_pow[i+1]):
            null_id = int(i)
            break

    # compute intergrated sidelobe ratio (ISLR)
    ISLR = np.sum(impres_pow[null_id:]) / np.sum(impres_pow[peak_id:null_id])
    ISLR = 10*np.log10(ISLR)

    if vocal:
        print('Peak sidelobe ratio = {:.2f} dB'.format(PSLR))
        print('Integrated sidelobe level ratio = {:.2f} dB'.format(ISLR))
        print('     peak         (#{}) = {}'.format(peak_id, np.sum(impres_pow[peak_id])))
        print('     1st sidelobe (#{}) = {}'.format(side_id, np.sum(impres_pow[side_id])))
        print('     null         (#{}) = {}'.format(null_id, np.sum(impres_pow[null_id])))

    return PSLR, ISLR, (peak_id, null_id, side_id)
```

In [ ]:

```
## Report the properties of the impulse response (in time domain)

PSLR, ISLR, _ = impres_property(impres_pow, vocal=True)
```

```
Peak sidelobe ratio = -13.49 dB
Integrated sidelobe level ratio = -10.44 dB
```

```
peak         (#1024) = 360000.0
1st sidelobe (#1034) = 16111.926115971164
null         (#1031) = 463.15370863888194
```

Note:

- The mainlobe is 13.49 dB higher than the first sidelobe (a factor of 22.33 in a linear scale)
- The integrated sidelobe energy is only 10% of the mainlobe

Weight the chirp to increase mainlobe feature:

It is shown that, by weighting the reference chirp before compressing the signal, we can increase the mainlobe feature in the impulse response. We can weight the chirp in time domain by convolving it with three delta functins (0.5 of the previours lobe, 0.5 of the next lobe, and 1 of itself), this is called Hann weighting function (see handout 11):

$$sinc(x) * [\frac{1}{2}\delta(x-1) + \frac{1}{2}\delta(x) + \frac{1}{2}\delta(x+1)]$$

where x is the location of each lobe on the time axis. $(x-1)$ correspondes to the previous lobe; $(x+1)$ corresponds to the next lobe.

In the frequency domain, it is equivalent to:

$$rect(f) \cdot [0.5 + 0.5 \cdot cos(f)]$$

More generally, we can use various weighting:

$$rect(f) \cdot [w + (1-w)cos(f)]$$

where $rect(f)$ is the spectrum of your original reference chirp. So f here denotes the frequency range (axis) of that chirp. We define the cosine curve as a function of frquency in that same range.

When $w = 1$, there is no weights (orignal reference chirp).

In [ ]:

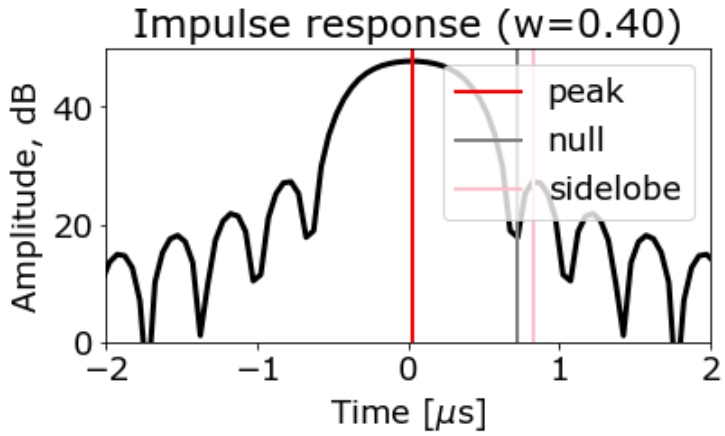
```
## Problem 5 (b) Now try weighting the chirp with different weights


# First let me define this function for weghting the chirp
def weight_chirp(N, slope, tau, fs, fc, Ws, chirp_plot=False, vocal=False, plot_reso=False):
    """
    INPUT:
    - N, slope, tau, fs, fc:
                  these are the param to create a default reference chirp (like we did before)
    - Ws:         is a list of different weights you would try
    - chirp_plot: whether to plot the weighted chirp power (in time domain)
    - volcal:     report some values
    """
    # make ref chirp
    ref, t  = makechirp(N, slope, tau, fs, fc=fc)
    freq    = np.linspace(-fs/2, fs/2, N)

    # fixed bandwidth
    bw = slope * tau

    # initialize the property arrays
    PSLR_arr = []       # array of PSLR
    ISLR_arr = []       # array of ISLR
    RESO_arr = []       # array of the width of the mainlobe (resolution)

    for w in Ws:
        # weighting function
        wf = w + (1-w)*np.cos((2*np.pi) * freq/bw)

        # weighted chirp in freq domain
```

```
        ref_fft   = np.fft.fft(ref)
        ref_fft_w = wf * ref_fft

        # get impulse response spectrum
        spec = ref_fft * np.conjugate(ref_fft_w)

        # transform it back to time domain
        impres_w     = np.fft.fftshift(np.fft.ifft(spec))

        # get the power
        impres_pow_w = np.abs(impres_w)**2

        # compute the properties
        PSLR, ISLR, idx = impres_property(impres_pow_w, vocal=vocal)
        PSLR_arr.append(PSLR)
        ISLR_arr.append(ISLR)

        pk_id = idx[0]
        nl_id = idx[1]
        sl_id = idx[2]

        # compute the mainlobe width
        ts = (t-np.mean(t))*1e6
        resol = 2 * np.abs(ts[nl_id]-ts[pk_id])
        RESO_arr.append(resol)

        # display the compressed chirp (impulse reponse) in time
        if chirp_plot:
            plt.figure(figsize=[6,3])
            plt.plot(ts, 10*np.log10(impres_pow_w), c='k', lw=3)
            plt.axvline(x=ts[pk_id], c='r',    lw=2, label='peak')
            plt.axvline(x=ts[nl_id], c='gray', lw=2, label='null')
            plt.axvline(x=ts[sl_id], c='pink', lw=2, label='sidelobe')
            plt.legend(loc='upper right', markerscale=2)
            plt.title('Impulse response (w={:.2f})'.format(w))
            plt.xlabel(r'Time [$\mu$s]')
            plt.ylabel('Amplitude, dB')
            plt.xlim(-2,2)
            plt.ylim(0,50)
            plt.show()

    plt.figure(figsize=[10,6])
    plt.plot(Ws, PSLR_arr, c='b', lw=3, label='PSLR')
    plt.plot(Ws, ISLR_arr, c='r', lw=3, label='ISLR')
    if len(Ws) < 100:
        plt.scatter(Ws, PSLR_arr, marker='o', s=40, c='b', linewidths=2)
        plt.scatter(Ws, ISLR_arr, marker='x', s=40, c='r', linewidths=2)
    plt.legend()
    plt.xlabel(r'Weights, $w$ [-]')
    plt.ylabel('Power [dB]')
    plt.show()

    if plot_reso:
        plt.figure(figsize=[10,6])
        plt.plot(Ws, RESO_arr, c='gold', lw=3, label='Mainlobe width')
        if len(Ws) < 100:
            plt.scatter(Ws, RESO_arr, marker='o', s=40, c='gold', linewidths=3)
        plt.legend()
        plt.xlabel(r'Weights, $w$ [-]')
        plt.ylabel(r'Mainlobe width [$\mu$ s]')
        plt.show()

    w_min_PSLR = Ws[np.argmin(PSLR_arr)]
    w_min_ISLR = Ws[np.argmin(ISLR_arr)]
    min_PSLR = np.min(PSLR_arr)
    min_ISLR = np.min(ISLR_arr)

    print('PSLR minimized at {:.3f}, by w = {:.3f}'.format(min_PSLR, w_min_PSLR))
    print('ISLR minimized at {:.3f}, by w = {:.3f}'.format(min_ISLR, w_min_ISLR))
```

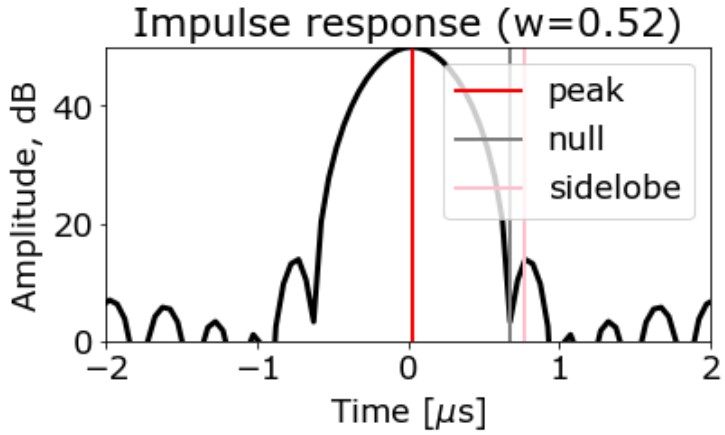
In [ ]:

```
# Chirp parameters
N     = 2048     # number of samples (include padded zeros)
slope = 1e11     # slope, Hz/s
tau   = 30e-6    # pulse length, s
fs    = 20e6     # sampling freq, Hz
fc    = 10e6     # center freq, Hz

# set a range of weights
Ws = np.linspace(0.4, 1, 6)

# Run that function, check the shape of the chirp power
weight_chirp(N, slope, tau, fs, fc, Ws, chirp_plot=True, vocal=True, plot_reso=True)
```

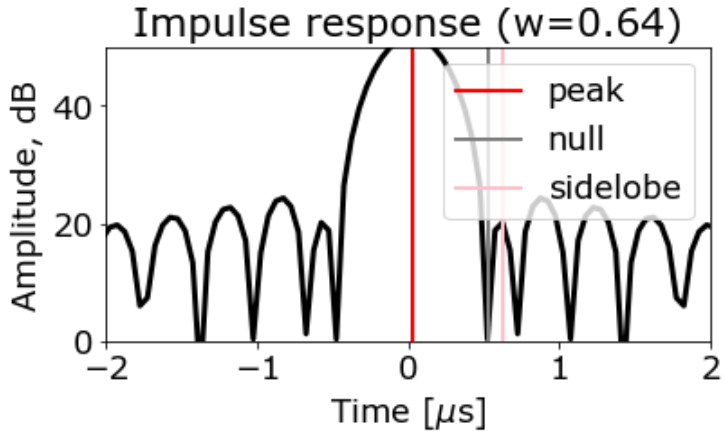
```
Chirp starts from 0 samples, 0.0 mu s
Peak sidelobe ratio = -20.52 dB
Integrated sidelobe level ratio = -18.98 dB
     peak         (#1024) = 59457.11876507216
     1st sidelobe (#1040) = 527.6259297220194
     null         (#1038) = 60.123409501440456
```



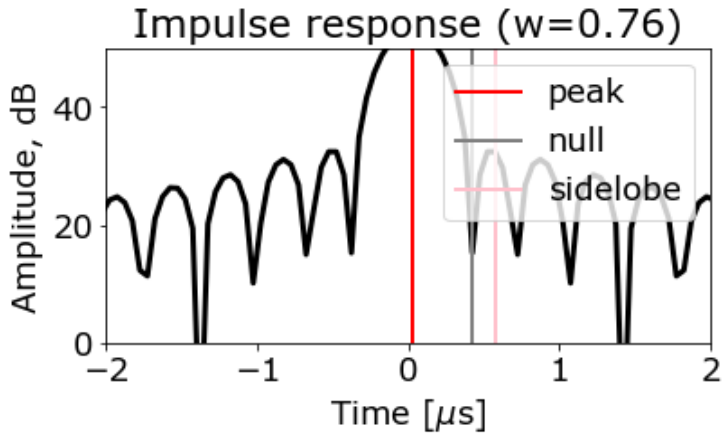

```
Peak sidelobe ratio = -36.11 dB
Integrated sidelobe level ratio = -21.75 dB
     peak         (#1024) = 99269.51048295233
     1st sidelobe (#1039) = 24.318205821222065
     null         (#1037) = 2.128902661665342
```



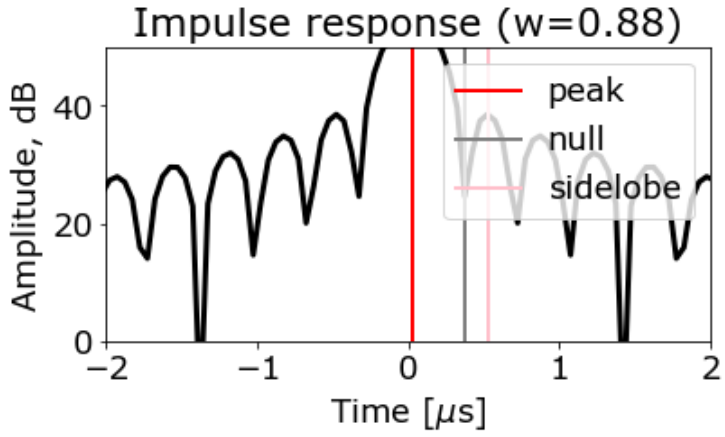

```
Peak sidelobe ratio = -31.61 dB
Integrated sidelobe level ratio = -19.14 dB
     peak         (#1024) = 149229.99446538519
     1st sidelobe (#1036) = 102.97772966095678
     null         (#1034) = 1.0801148314959717
```




```
Peak sidelobe ratio = -20.81 dB
Integrated sidelobe level ratio = -15.42 dB
     peak          (#1024) = 209338.5707123708
     1st sidelobe (#1035) = 1738.5386773915566
     null          (#1032) = 33.4851108070392
```

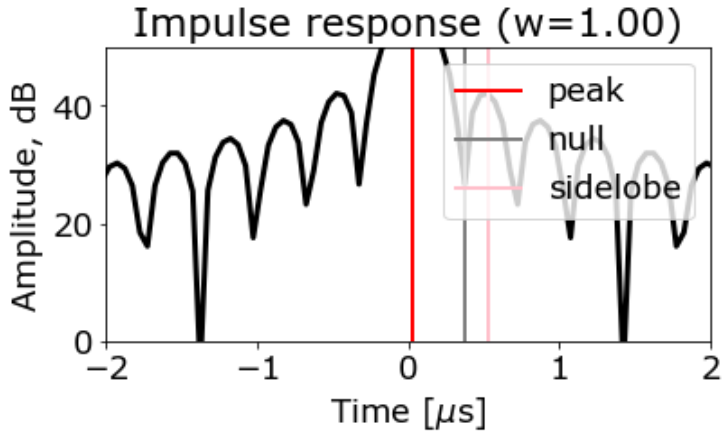


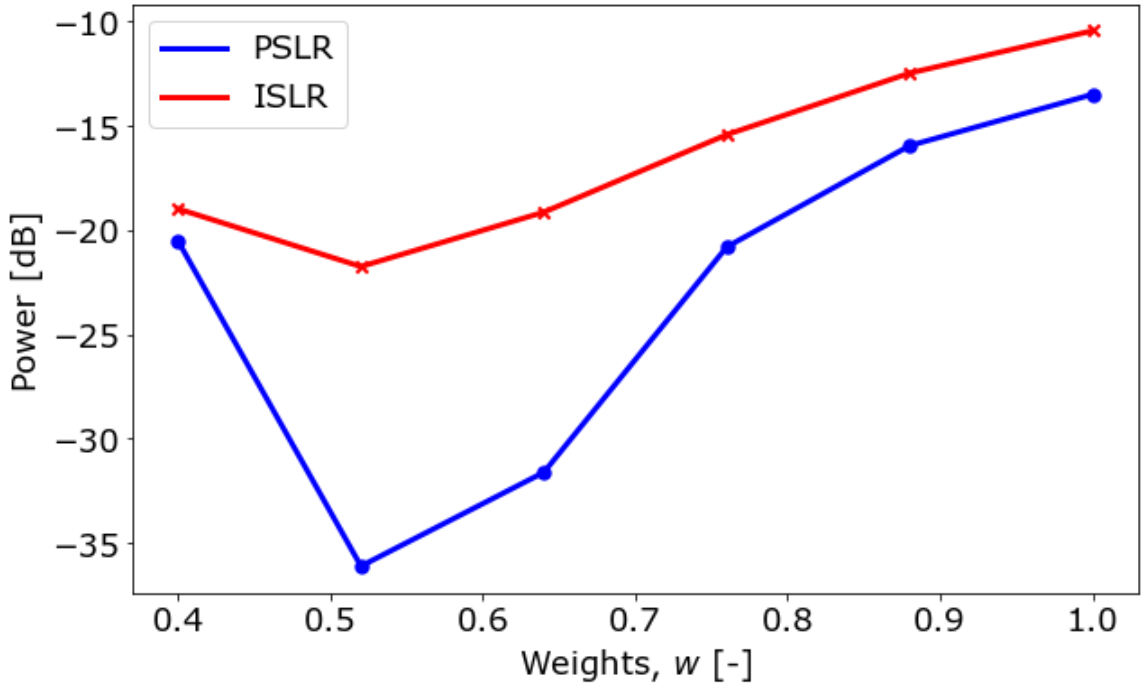

```
Peak sidelobe ratio = -15.95 dB
Integrated sidelobe level ratio = -12.47 dB
     peak          (#1024) = 279595.239223909
     1st sidelobe (#1034) = 7104.458645541188
     null          (#1031) = 288.1262688651125
```

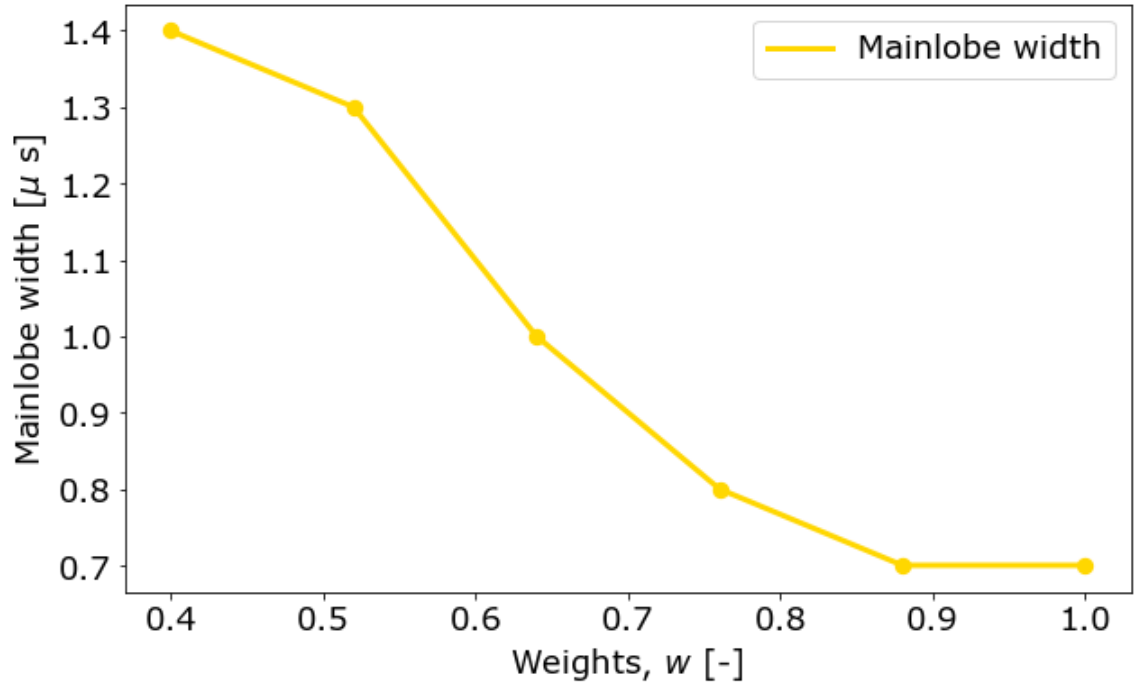



```
Peak sidelobe ratio = -13.49 dB
Integrated sidelobe level ratio = -10.44 dB
     peak          (#1024) = 360000.0
     1st sidelobe (#1034) = 16111.926115971164
     null          (#1031) = 463.15370863888194
```

```
<ipython-input-17-dac803f26b34>:60: RuntimeWarning: divide by zero encountered in log10
  plt.plot(ts, 10*np.log10(impres_pow_w), c='k', lw=3)
```





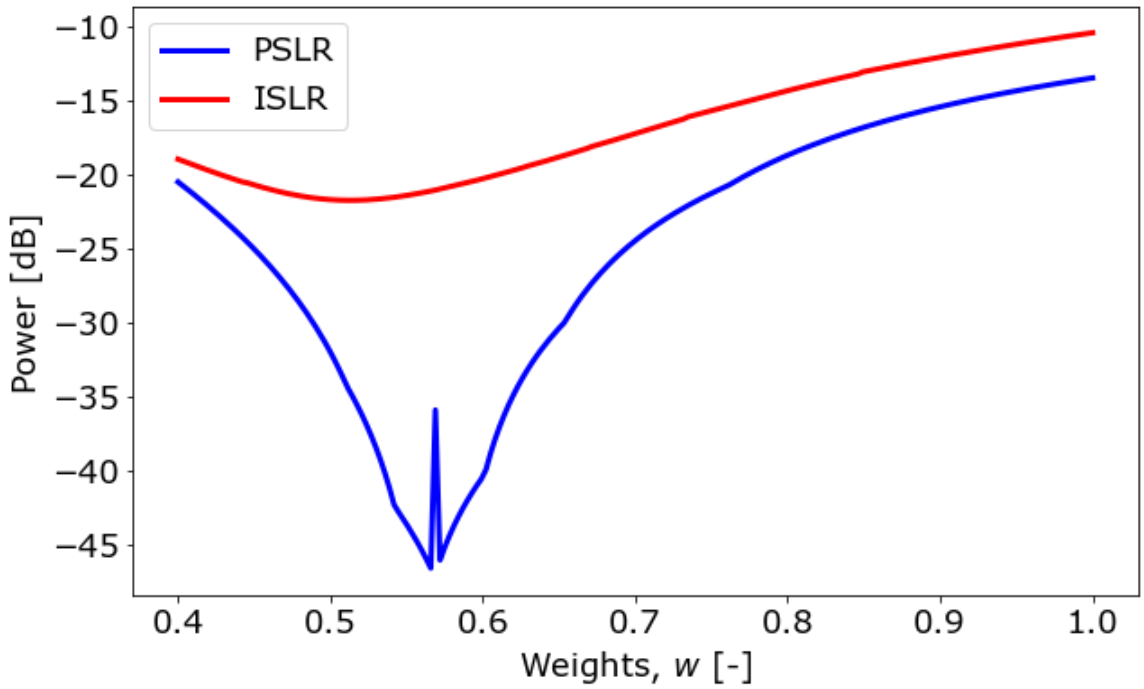



```
PSLR minimized at -36.109, by w = 0.520
ISLR minimized at -21.746, by w = 0.520
```

Notice that by doing the weighting or filtering, the width of the mainlobe will increase. Thus the resolution decreases.

In [ ]:
```
# Run this at a finer range of weights

Ws = np.linspace(0.4, 1, 200)
weight_chirp(N, slope, tau, fs, fc, Ws, chirp_plot=False, vocal=False)
```

```
Chirp starts from 0 samples, 0.0 mu s
```




```
PSLR minimized at -46.598, by w = 0.566
ISLR minimized at -21.758, by w = 0.515
```

- PSLR is minimized at -46.435, by w = 0.566
- ISLR is minimized at -21.756, by w = 0.512

The PSLR is minimized at $w \sim 0.57$, which is roughly consistent with the equation for the Hamming window that best surpresses the amplitude of the first sidelobe.

The ISLR is minimized at $w \sim 0.51$, which is roughly consistent with the equation for the Hann window that minimizes total sidelobe energy.

There is some numerical issue around the minimum of the PSLR, where the first sidelobe partially merges into the mainlobe, while the sidelobe peak remains. Thus the PSLR looks anamolous at that specific w.

In [ ]: