Ge193 Imaging radar and application

Homework 4: Range-Doppler / Unfocussed processing

In [ ]:
```
## Import modules

import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from tqdm import tqdm

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

In [ ]:

```
## Define functions

def readuint8(file, nsamp, nlines):
    with open(file, 'rb') as fn:
        load_arr = np.frombuffer(fn.read(), dtype=np.uint8)
        load_arr = load_arr.reshape((nlines, nsamp))
    return np.array(load_arr)


def plot_img(data, nhdr=0, title='Data', scale=1, vlim=[None,None], origin='upper', aspect=
"equal", interpolation='none', savetif=None):
    if scale > 1:
        clabel = 'Value * {} [-]'.format(scale)
    else:
        clabel = 'Value [-]'

    # Adjust the data part for better visualization
    val = np.array(data)
    val[:,nhdr:] = scale * val[:,nhdr:]

    # plot the 2D image
    plt.figure(figsize=[14,14])
    im   = plt.imshow(val, cmap='gray', interpolation=interpolation, vmin=vlim[0], vmax=vlim
[1], origin=origin, aspect=aspect)
    cbar = plt.colorbar(im, shrink=0.3, pad=0.02)
    cbar.set_label(clabel, rotation=270, labelpad=30)
    plt.title(title)
    plt.xlabel('Range [bins]')
    plt.ylabel('Azimuth [lines]')
    if savetif is not None:
        plt.savefig('{}'.format(savetif), format = 'tif')
    plt.show()


def makechirp(N, slope, tau, fs, fc=0, start=0, phi0=0):
    """ Make a reference chirp pulse
    N:     Num of points of the whole pulse    [#]
    slope: slope of the chirp                  [Hz/s]
    tau:   chirp length                        [s]
    fs:    sample rate                         [Hz]
    fc:    central carrier freq                [Hz]
    start: starting sample # of the chirp      [#]
    """
    dt    = 1/fs                                        # sampling time interval
 [s]
    npts  = tau * fs                                    # num of points of the pure chirp
 [#]
    t     = dt * np.arange(-npts/2, npts/2)             # time axis of the pure chirp
 [s]
    phase = np.pi*slope*(t**2) + 2*np.pi*fc*t + phi0    # chirp phase                   [r
ad]
    chirp = np.exp(1j*phase)                            # chirp                         (c
mplx)
    chirp = np.pad(chirp, (start,N-len(chirp)-start))   # pad zeros at tail and beginning (c
mplx)
    t_arr = dt * np.arange(0, N)                        # time axis of the whole pulse
 [s]
    #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 domai
n
    ref_fft = np.fft.fft(ref)                           # transform the reference chirp to f
```

```
req domain
    spec    = sig_fft * np.conjugate(ref_fft)           # cross-correlation gives the spectr
um
    comp    = np.fft.ifft(spec)                         # inverse transform it back to time
 domain
    return comp, spec


def plot_freq(freq, val, title, x='Frequency', y='20*log10(|spectrum|), [dB]', xlim=[None,No
ne], ylim=[None,None], unit='MHz', shift=False):
    x += ' [{}]'.format(unit)
    if unit == 'MHz':
        u = 1e-6
    elif unit == 'Hz':
        u = 1
    if shift:
        val = np.fft.fftshift(val)
    plt.figure(figsize=[14,4])
    plt.plot(freq*u, val)
    plt.title(title)
    plt.xlim(min(freq)*u, max(freq)*u)
    plt.xlim(xlim[0], xlim[1])
    plt.ylim(ylim[0], ylim[1])
    plt.xlabel(x)
    plt.ylabel(y)
    plt.show()


def plot_time(t, val, title, x=r'Time', y='amplitude [-]', xlim=[None,None], ylim=[None,None
], unit='micros', shift=False):
    if unit == 'micros':
        x += r' [$\mu$ s]'
        u = 1e6
    elif unit == 's':
        x += r' [s]'
        u = 1
    if shift:
        val = np.fft.fftshift(val)
    plt.figure(figsize=[14,4])
    plt.plot(t*u, val)
    plt.title(title)
    plt.xlim(min(t)*u, max(t)*u)
    plt.xlim(xlim[0], xlim[1])
    plt.ylim(ylim[0], ylim[1])
    plt.xlabel(x)
    plt.ylabel(y)
    plt.show()


def magdB(val):
    mag = 20 * np.log10(np.abs(val) + 1e-30)
    return mag


def doppler_phase_shift(R, PRF):
    Naz, Nr = R.shape
    psRg = np.zeros(Nr, dtype=np.complex128)
    for i in np.arange(2,Naz):
        psRg += R[i,:] * np.conjugate(R[i-1,:])
    # compute phase shift at each range bin
    phi = np.arctan(np.imag(psRg)/np.real(psRg))
    phi_avg = np.mean(phi)
    fd = PRF * phi_avg/(2*np.pi)
    return fd, phi_avg

def doppler_cent_squint(fd, v, theta, wavelength):
    sine_squint = fd * wavelength * 0.5 * v / np.sin(theta)
    squint = np.arcsin(sine_squint)
    return squint
```

Consider a radar with the following parameters:

Range modulation:

- Chirp slope: $4.189166 \cdot 10^{11}\ Hz/s$
- Pulse length: $37.12\ \mu s$
- Sample rate fs: $18.96\ MHz$

Other parameters:

- PRF: $1679.9\ Hz$
- Range r_0: $830000\ m$
- I,Q average values: 15.5
- Platform velocity: $7550\ m/s$
- Wavelength: $0.0566\ m$
- Antenna length: $10\ m$
- Earth radius: $6378\ km$
- Look angle: 23°

Problem 1:

(a) How many valid range bins are found in the range compressed data, assuming as before the range record length is 10,218 bytes of which 412 are header bytes?

(b) What is the minimum fft size for range processing?

Valid range bins:

$$n_{valid} = n_{echo} - n_{ref} + 1 = \frac{10218 - 412}{2} - \tau \times f_s + 1$$

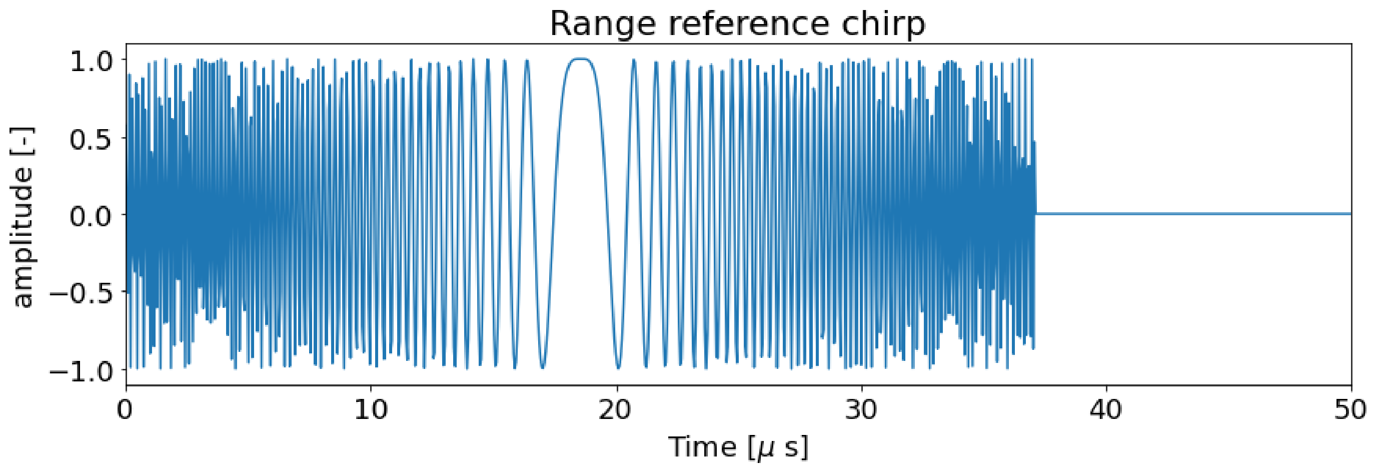
$$= 4903 - (37.12 \cdot 10^{-6} \times 18.96 \cdot 10^{6}) + 1$$
$$= 4903 - 703.8 + 1 = 4200.2$$

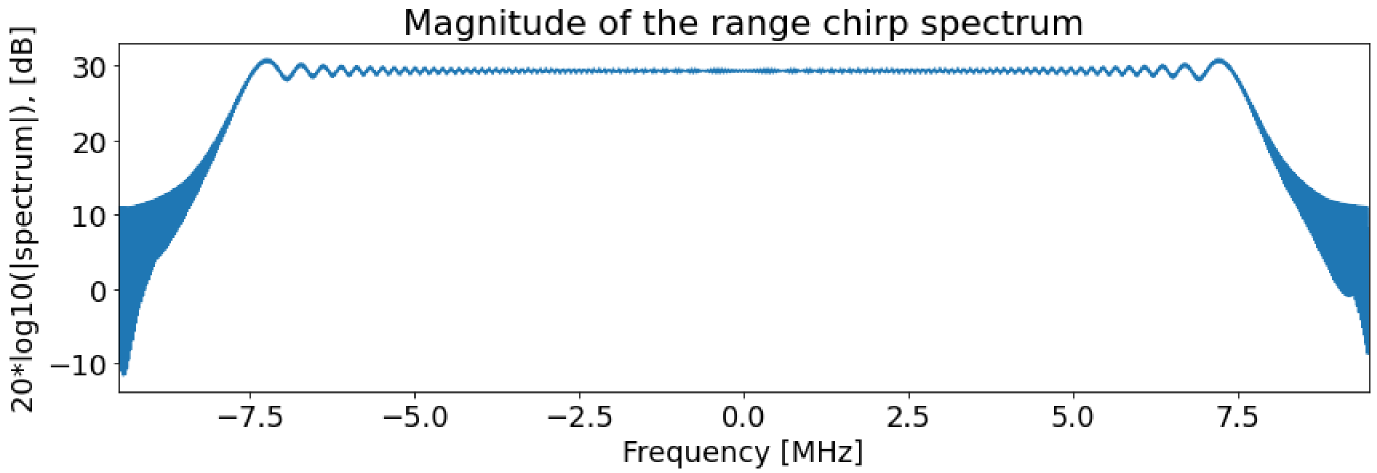
Round it so that n_{valid} = 4200

Minimum FFT size for range processing:

The minimum fft size for range processing is the same as the length of the original echo, which is 4903 samples. If we want this to be a power of 2, I can zero pad up to 8192 samples, keeping in mind that I should only keep the valid data after compressions (4200). With our small data set, zero-padding should not be necessary. So we use 4903 as the FFT size in range processing.

```
In [ ]: ## Problem 1 parameters:

        # ERS data info
        ERS    = './ersdata.hw3'
        nhdr   = 412
        nsamp  = 10218
        nlines = 10100

        # ERS range modulation
        N      = int((nsamp-nhdr)/2)   # number of complex samples [-]    (N=4903)
        slope  = 4.189166e11           # slope                     [Hz/s]
        tau    = 37.12e-6              # pulse length              [s]
        bw     = slope * tau           # bandwidth                 [Hz]
        fs     = 18.96e6               # sample rate               [Hz]
        fc     = 0                     # center Frequency          [Hz]

        # Other parameters
        c      = 2.99792458e8          # speed of light            [m/s]
        PRF    = 1679.9                # pulse repetition freq     [Hz] (the number of pulses of a r
        epeating signal in a second)
        r0     = 830000                # platform range            [m]
        iq_avg = 15.5                  # I/Q system average values [-]
        vx     = 7550                  # platform velocity         [m/s]
        wl     = 0.0566                # wavelength                [m]
        l      = 10                    # antenna length            [m]
        Re     = 6378 * 1e3            # Earth's radius            [m]
        lk     = 23                    # Look angle                [deg]


        # create chirp
        chirp, t = makechirp(N, slope, tau, fs, fc=fc)  # time axis: total extent N/fs centered at f
        c*slope

        # transform it to a spectrum
        chirp_fft = np.fft.fft(chirp)
        freqRg    = np.linspace(fc-fs/2, fc+fs/2, N)    # freq axis: total extent fs centered at fc

        # calculate the dB spectrum of the signal
        chirp_mag = magdB(chirp_fft)

        # plot the chirp
        plot_time(t, np.real(chirp), 'Range reference chirp', xlim=[0, 50])

        # plot the chirp spectrum
        plot_freq(freqRg, chirp_mag, 'Magnitude of the range chirp spectrum', shift=True)
```





(c) What is the effective spacecraft velocity v_{eff}?

(d) What is the range bin spacing in meters? What is the range resolution? Give both slant range and approximate ground range results.

Effective velocity:

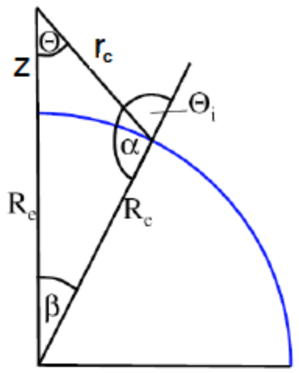


Figure 1: Spherical earth geometry.

First get the range to the center of the swath: (Δr is the range bin spacing, defined as $\frac{c}{2\times f_s}$)

$$r_c = r_0 + \Delta r \times \frac{n_{valid}}{2}$$

Then using the geometry to get the platform altitude:

$$R_e^2 = R_e^2 + (R_e^2 + z)^2 - 2r_c(R_e + z)cos(\theta)$$
$$z = r_c cos(\theta) - R_e + \sqrt{R_e^2 - r_c^2 sin^2(\theta)}$$

Finally, the effective velocity based on v, z, and R_e

$$v_{eff} = v \times \sqrt{\frac{r_e}{r_e + z}}$$
$$= 7550 \times \frac{6378 \cdot 10^3}{6378 \cdot 10^3 + z}$$

Slant and ground range spacing

$$\Delta r = \frac{c}{2f_s}$$
$$\Delta r_{ground} = \frac{c}{2f_s sin(\theta_i)}$$

Slant and ground range resolution

$$\delta r = \frac{c}{2BW}$$
$$\delta r = \frac{c}{2BW sin(\theta_i)}$$

where θ_i is the central incidence angle at the center of the swath,

$$\frac{sin\beta}{r_c} = \frac{sin\theta}{R_e} \rightarrow \beta = sin^{-1}(\frac{r_c}{R_e} sin\theta)$$
$$\theta_i = \theta + \beta$$

```
In [ ]: # Number of valid range bins
        n_valid = int(np.round(N - tau * fs + 1))
        print('Valid range bins = {}'.format(n_valid))

        # Find the range pointing to the center of the swath, rc
        delta_r = c/2/fs
        rc = r0 + delta_r*n_valid/2
        print('Central range = {:.2f} m'.format(rc))

        # Find platform altitude
        z = rc*np.cos(np.deg2rad(lk)) - Re + np.sqrt(Re**2 - rc**2*np.sin(np.deg2rad(lk))**2)
        print('Platform altitude = {:.2f} m'.format(z))

        # Effective spacecraft velocity
        veff = vx * np.sqrt(Re/(Re+z))
        print('Effective velocity = {:.2f} m/s'.format(veff))

        # incidence angle at the center of the swath
        beta = np.rad2deg(np.arcsin(rc*np.sin(np.deg2rad(lk))/Re))
        inc  = lk + beta
        print('Central incidence angle = {:.2f} deg'.format(inc))

        # Range spacing
        delta_gr = delta_r/np.sin(np.deg2rad(inc))
        print('Slant range spacing: {:.2f} m/pixel'.format(delta_r))
        print('Ground range spacing: {:.2f} m/pixel'.format(delta_gr))

        # Range resolution
        dr  = c/bw/2
        dgr = dr/np.sin(np.deg2rad(inc))
        print('Slant range resolution: {:.2f} m'.format(dr))
        print('Ground range resolution: {:.2f} m'.format(dgr))
```

```
Valid range bins = 4200
Central range = 846602.43 m
Platform altitude = 770717.58 m
Effective velocity = 7131.41 m/s
Central incidence angle = 25.97 deg
Slant range spacing: 7.91 m/pixel
Ground range spacing: 18.05 m/pixel
Slant range resolution: 9.64 m
Ground range resolution: 22.01 m
```

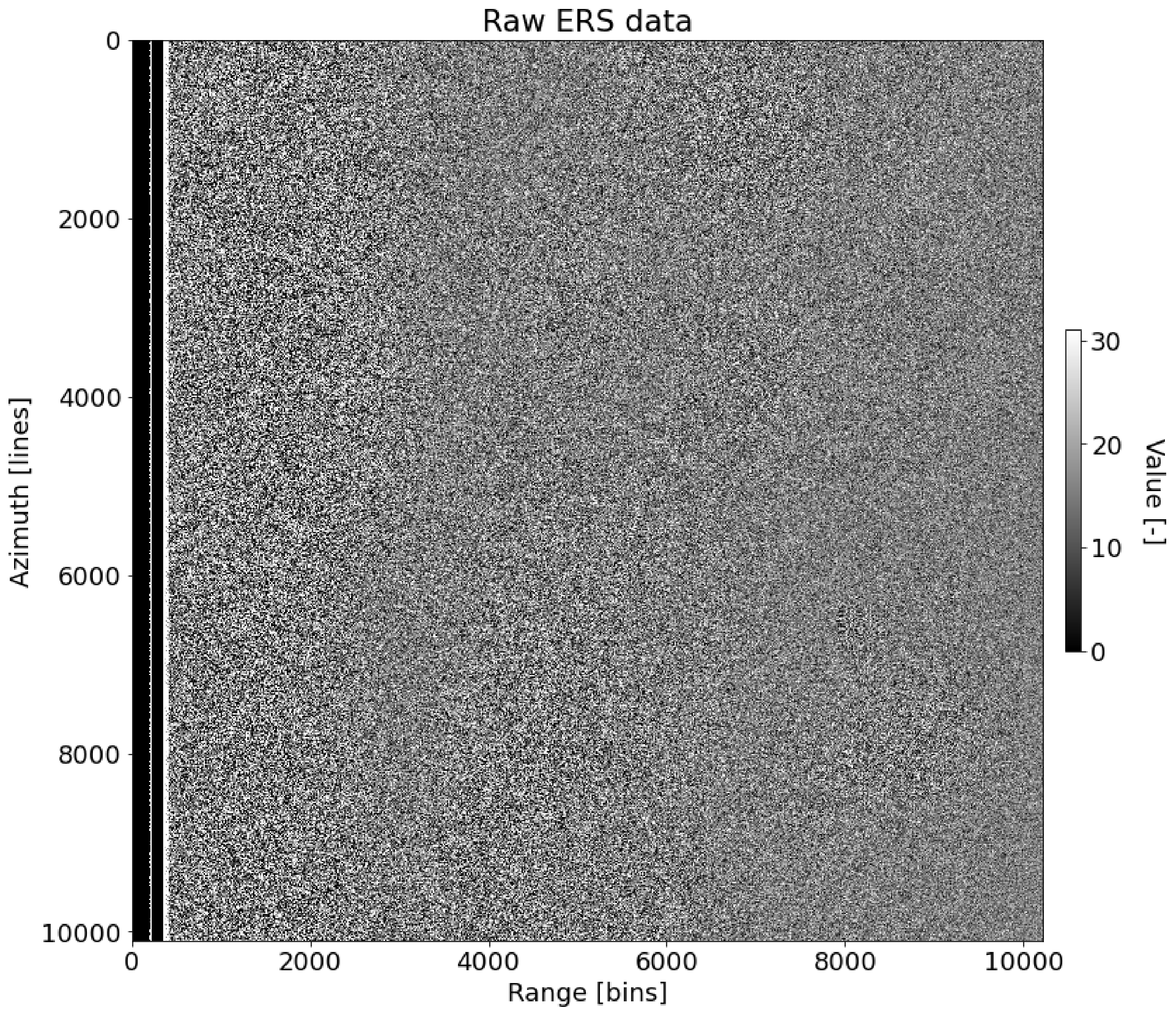
Problem 2 Download the raw data file ersdata.hw3 that we used last time. As before, the quantization level is 5 bits with average I and Q values of 15.5.

Problem 2 (a):

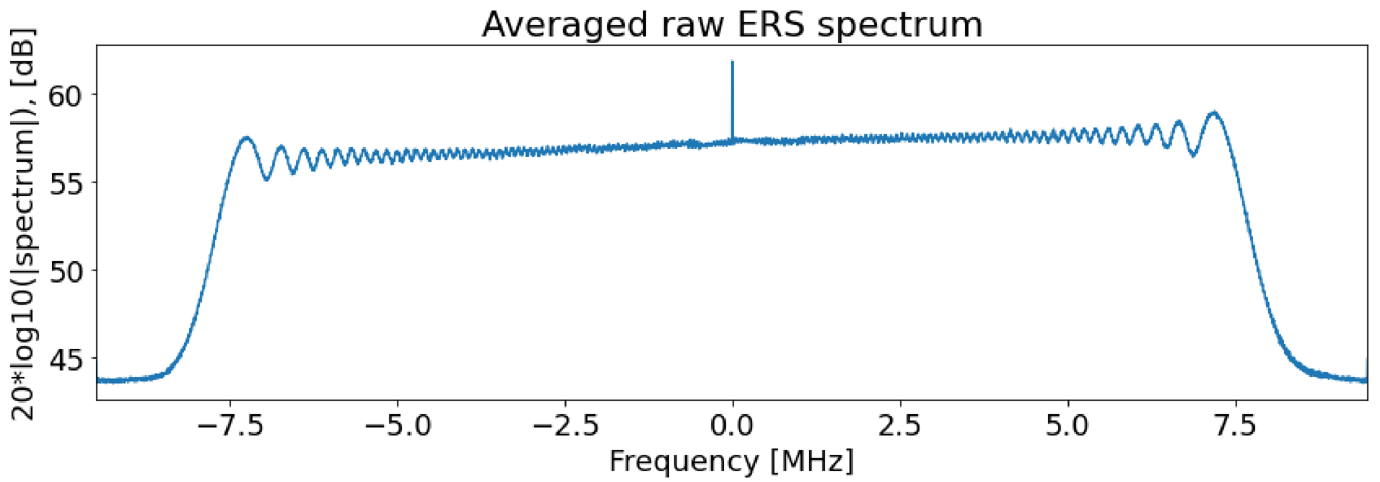
Determine the approximate Doppler centroid of the data.

```
In [ ]: ## Let's read ERS data from the binary file

        # read ERS data
        data = readuint8(ERS, nsamp, nlines)

        # plot ERS data: ERS is digitized in to 5-bit (0~31) in a 8-bit system
        plot_img(data, nhdr, title='Raw ERS data', scale=1, vlim=[0,31])

        # convert ERS data to complex floats
        sig      = data[:, nhdr:]
        sig_odd  = sig[:,  ::2] - iq_avg    # real part
        sig_even = sig[:, 1::2] - iq_avg    # imaginary part
        sig      = sig_odd + 1j*sig_even    # combine to complex

        # 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 oth
        er)
        sig_avgRg_f = np.mean(np.abs(sig_fft), axis=0)

        # get the magnitude and plot
        sig_avgRg_f_mag = magdB(sig_avgRg_f)

        # plot ERS range spectrum
        plot_freq(freqRg, sig_avgRg_f_mag, 'Averaged raw ERS spectrum', shift=True)


        ## Compress each range line and create a "range-compressed" image

        # do matched filter for each azimuth line
        sig_comp_t, sig_comp_f = matched_filter(sig, chirp)

        # get only the valid range bins
        sig_comp_t_valid = sig_comp_t[:, :n_valid]

        # get the azimuth spectrum of each range bin
        #sig_comp_f_valid = sig_comp_f[:, :n_valid]        # this is range spectrum of each azimuth
         line, not what we want
        sig_az_fft = np.fft.fft(sig_comp_t_valid, axis=0)   # this is what we want (contains Doppler
        information)


        # plot the compressed img
        plot_img(np.abs(sig_comp_t_valid), title='Magnitude of range compressed image', savetif='./h
        w3_rangecompressedERS.tiff')

        # average the compressed data
        sig_comp_avgRg_f = np.mean(np.abs(sig_comp_f), axis=0)  # average range spectrum over azimut
        h lines (looks like reference chirp)
        sig_comp_avgAz_f = np.mean(np.abs(sig_az_fft), axis=1)  # average azimuth spectrum over rang
        e bins (Doppler effects)

        # get the magnitude and plot
        sig_comp_avgRg_f_mag = magdB(sig_comp_avgRg_f)
        sig_comp_avgAz_f_mag = magdB(sig_comp_avgAz_f)
        freqAz = np.linspace(-PRF/2, PRF/2, nlines)

        # plot ERS spectrum
        plot_freq(freqRg, sig_comp_avgRg_f_mag, 'ERS compressed average range spectrum', shift=True)
        plot_freq(freqAz, sig_comp_avgAz_f_mag, 'ERS compressed average azimuth spectrum', unit='Hz'
        , shift=True)

        # find the spectrum peak, that is the Doppler centroid
```

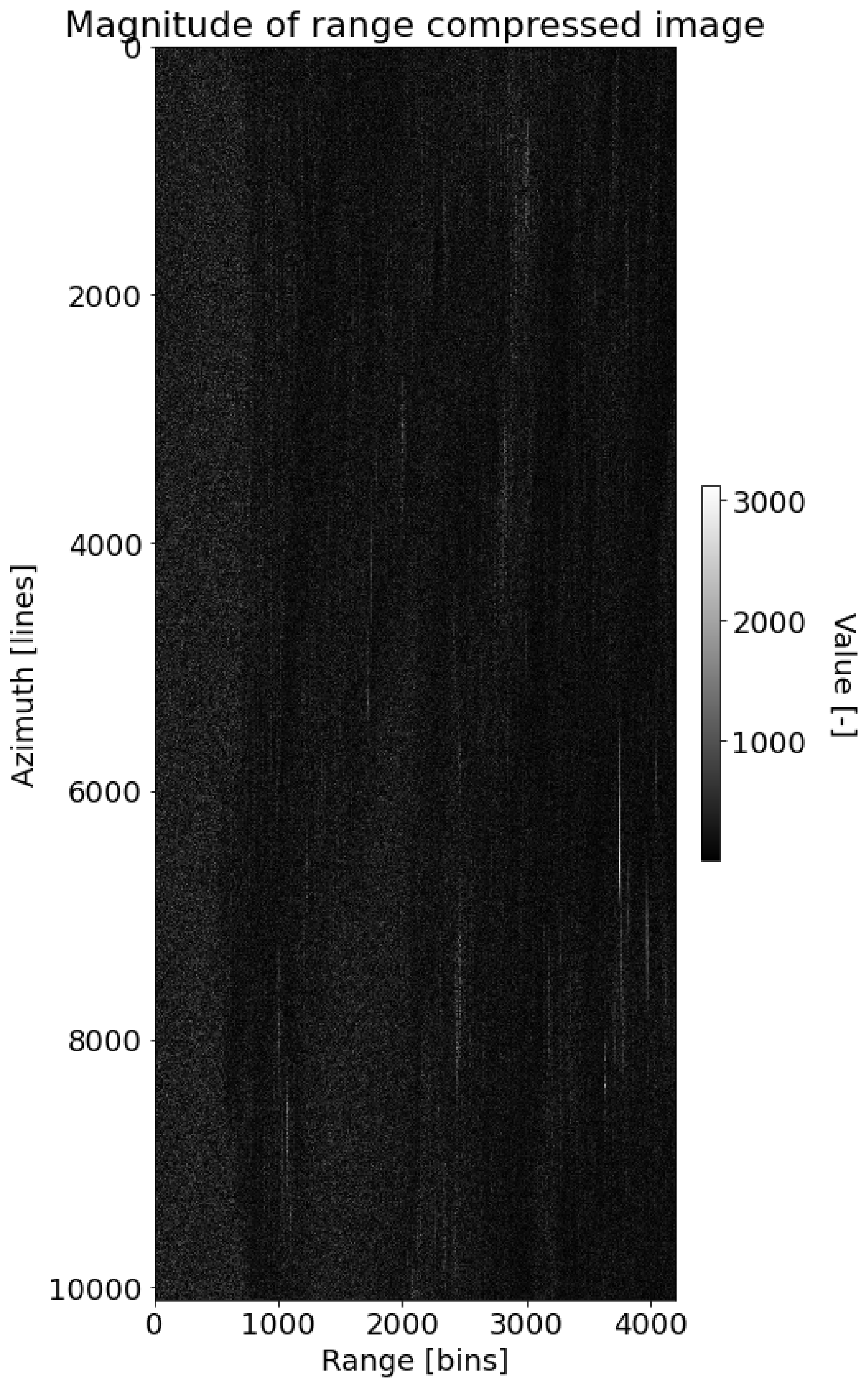
```
idx = np.fft.fftshift(sig_comp_avgAz_f_mag) < 95
fd = freqAz[np.argmax(np.fft.fftshift(sig_comp_avgAz_f_mag)[idx])]
print('Doppler centroid (numerically determined) = {:.3f} MHz'.format(fd))

# find the doppler freq using phase change method
fd, phi_avg = doppler_phase_shift(sig_comp_t_valid, PRF)
print('Doppler centroid (phase shift method) = {:.3f} MHz'.format(fd))
```

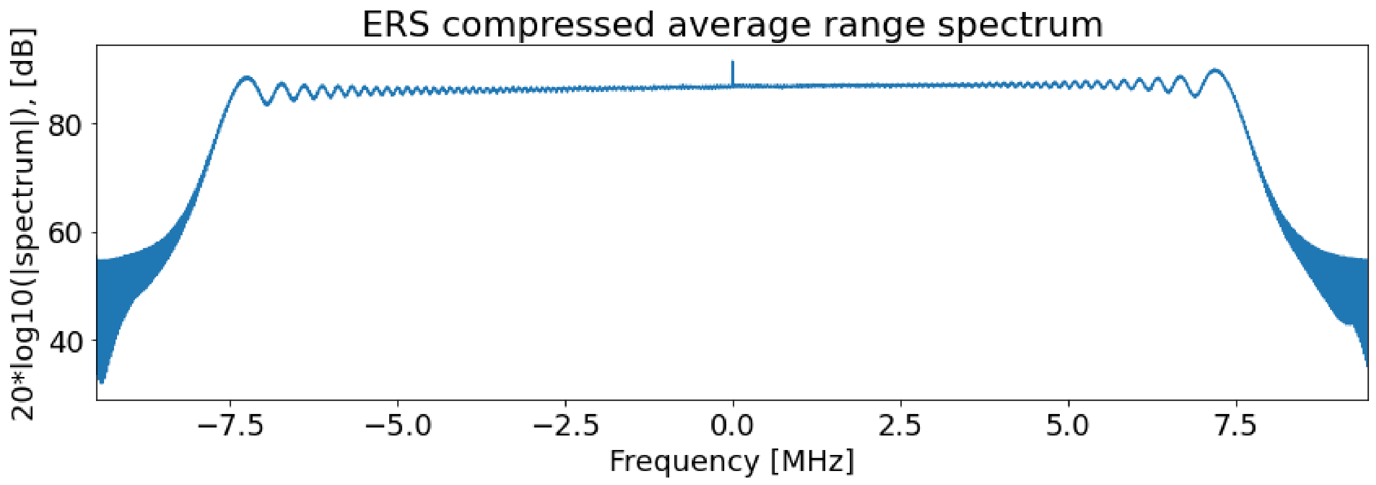


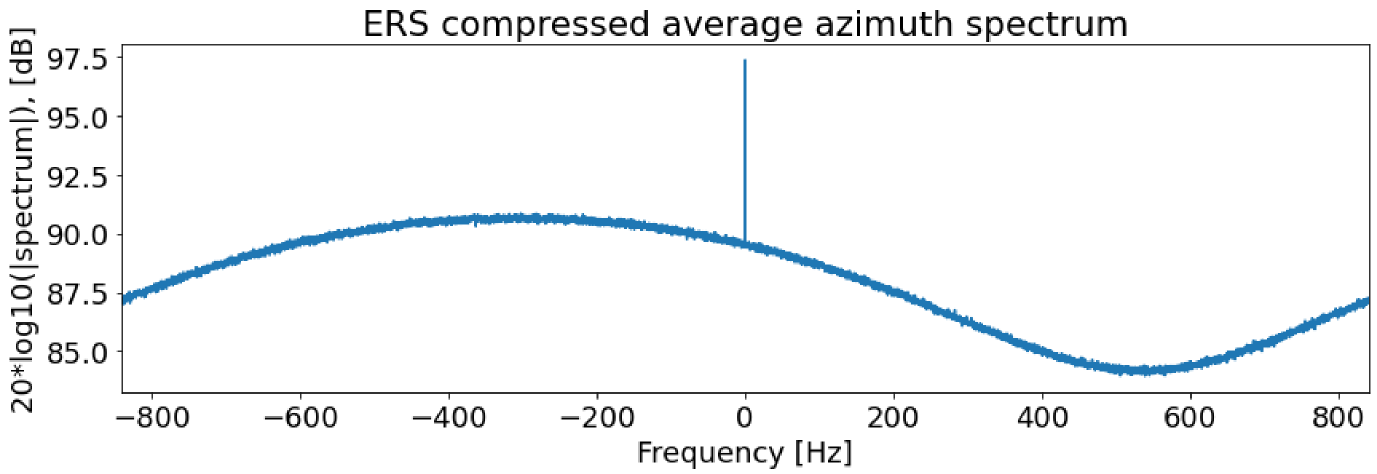
Raw ERS data
Azimuth [lines]
Range [bins]
Value [-]
0
2000
4000
6000
8000
10000
0
10
20
30



Averaged raw ERS spectrum
20*log10(|spectrum|), [dB]
Frequency [MHz]
45
50
55
60
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5


Magnitude of range compressed image
Azimuth [lines]
0
2000
4000
6000
8000
10000
Range [bins]
0
1000
2000
3000
4000
Value [-]
1000
2000
3000





```
Doppler centroid (numerically determined) = -302.162 MHz
Doppler centroid (phase shift method) = -299.959 MHz
```

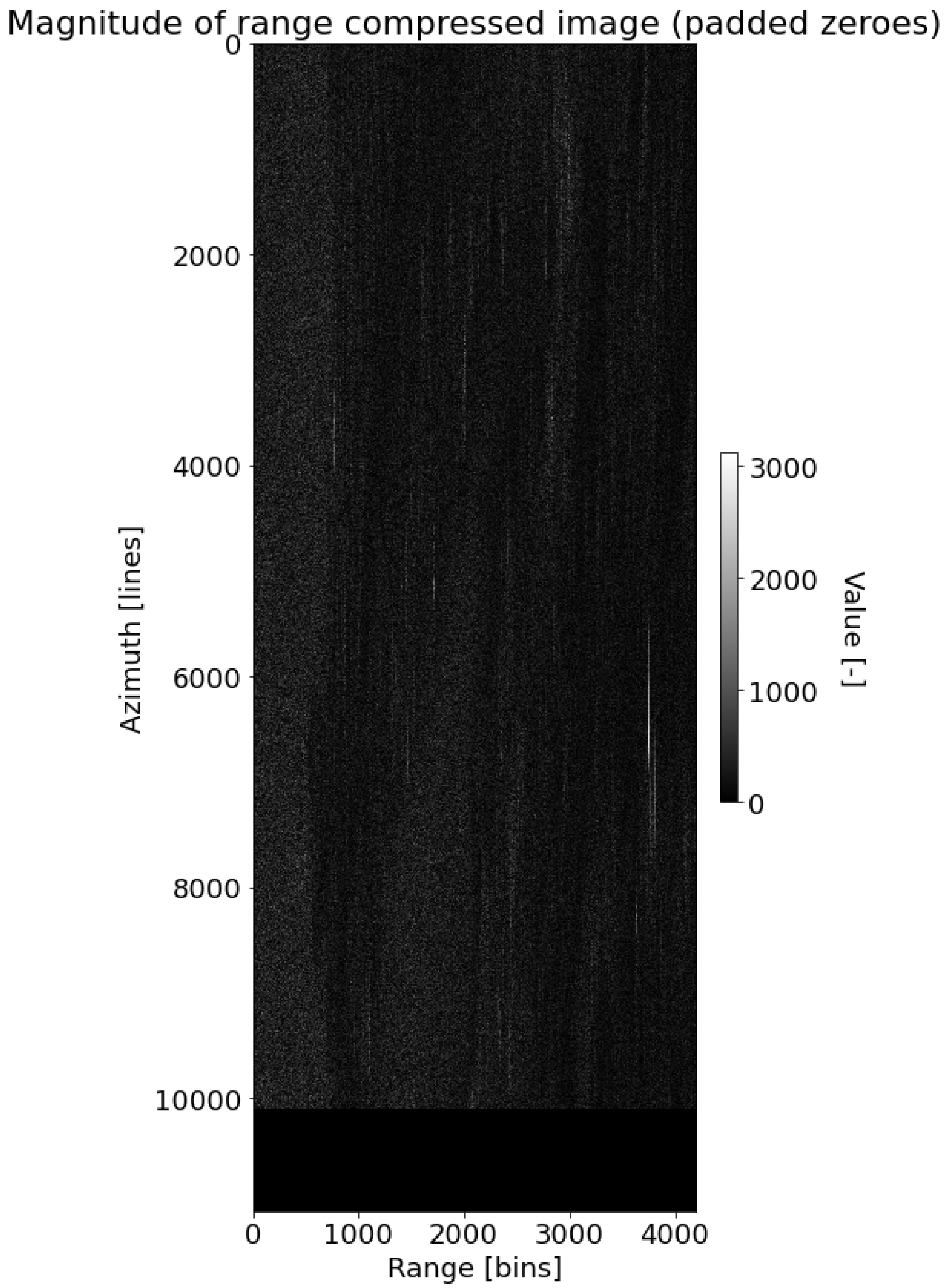
Problem 2 (b)

Write a multipatch SAR processor for this data set. Set the azimuth beamwidth to 80% of the full illuminated beamwidth to keep the reference function below 1024 points. Process as many patches as you can from the data set given.

Each patch has 2048 azimuth lines, as specified in the problem. To process as many patches as possible, we need the offset between patches to be as small as possible. Since the offset is the number of the valid pixels in the azimuth direction. To minimize the shift, we want to maximize the azimuth reference length. This happens at the farthest possible range $r_{dc,max}$, which will have the maximum illumination time $\tau_{az,max}$, and therefore the largest number of azimuth chirp samples.

```
In [ ]: # Azimuth beamwidth ground extent for a 10-m antenna (s)
        L_az = r0 * wl / l    # ground extent of azimuth beamwidth [m]
        t_cyc = L_az / vx     # cycle time between iluminating this patch and the next patch
        print('Azimuth beamwidth of a {:.1f}-m antenna = {:.3f} [m]'.format(l, L_az))
        print('Repeat cycle time for a {:.1f}-m antenna = {:.3f} [s]'.format(l, t_cyc))

        # Number of lines to process in one patch (num_pulse)
        num_az_echo = int(2048)       # given by the problem set
        print('Num of pulses = {:d} [#]'.format(num_az_echo))

        # Range at the far end of the swath
        r_max = r0 + (n_valid-1)*delta_r
        print('Range maximum = {:.2f} m'.format(r_max))

        # Range to the Doppler centroid at far end
        r_dc_max = np.sqrt(r_max**2 + (fd*wl*r_max/2/veff)**2)
        print('Range maximum to Doppler centroid = {:.2f} m'.format(r_dc_max))

        # The time required for a point at the far end to traverse 80% of the beamwidth
        tau_az_max = 0.8 * r_dc_max * wl / veff / l
        print('Azimuth pulse length maximum = {:.2f} s'.format(tau_az_max))

        # Number of points in the azimuth reference chirp (longest azimuth chirp)
        num_az_ref = int(tau_az_max * PRF)
        print('Azimuth chirp length to process max number of patches: {} points'.format(num_az_ref))

        # Offset between nearby patches
        n_az_valid = num_az_echo - num_az_ref
        print('Patch azimuth offset: {}'.format(n_az_valid))


        # Azimuth spacing
        #delta_az = veff / PRF
        delta_az = (vx/PRF) * (Re/(Re+z))
        print('Azimuth spacing = {:.2f} m/pixel'.format(delta_az))

        # Azimuth resolution
        d_az = l/2/0.8
        print('Azimuth Resolution = {:.3f} [m]'.format(d_az))

        # Num of looks: the ratio between them
        azlook = int(np.round(delta_gr/delta_az))
        print('Number of looks needed: {:.3f}'.format(azlook))

        # number of lines per patch with looks
        Naveaz = np.floor(n_az_valid/azlook)

        ## Number of patches
        num_patch = int(np.round(nlines/n_az_valid))  # number of patches we will process
        Naz       = int(np.round(n_az_valid * num_patch))
        print('Number of patches we will process: {}'.format(num_patch))
        print('Number of azimuth pixels in the multi-looked image = {} * {} = {}'.format(n_az_valid,
        num_patch, Naz))


        # num of lines in the final image with looks
        numazave = int(Naveaz * num_patch)

        # initialize the final image with looks
        finalimage_ml = np.zeros([numazave, n_valid])
        print('New image output size (multilook): {}'.format(finalimage_ml.shape))

        # Pad range compressed image with zeros - prevent out of bound errors
        print('Original image size: {}'.format(sig_comp_t_valid.shape))
        data_all = np.pad(sig_comp_t_valid, [(0, int(np.round(n_az_valid * (num_patch-1)))+num_az_ec
ho-sig_comp_t_valid.shape[0]), (0,n_valid-sig_comp_t_valid.shape[1])])
#data_all = np.pad(sig_comp_t_valid, [(0, Naz-sig_comp_t_valid.shape[0]), (0,n_valid-sig_com
p_t_valid.shape[1])])

print('Original image size (padded zeros): {}'.format(data_all.shape))
plot_img(np.abs(data_all), title='Magnitude of range compressed image (padded zeroes)')
```

```
Azimuth beamwidth of a 10.0-m antenna = 4697.800 [m]
Repeat cycle time for a 10.0-m antenna = 0.622 [s]
Num of pulses = 2048 [#]
Range maximum = 863196.95 m
Range maximum to Doppler centroid = 863197.57 m
Azimuth pulse length maximum = 0.55 s
Azimuth chirp length to process max number of patches: 920 points
Patch azimuth offset: 1128
Azimuth spacing = 4.01 m/pixel
Azimuth Resolution = 6.250 [m]
Number of looks needed: 5.000
Number of patches we will process: 9
Number of azimuth pixels in the multi-looked image = 1128 * 9 = 10152
New image output size (multilook): (2025, 4200)
Original image size: (10100, 4200)
Original image size (padded zeros): (11072, 4200)
```



Magnitude of range compressed image (padded zeroes)
Azimuth [lines]
Range [bins]
Value [-]
0
2000
4000
6000
8000
10000
1000
3000

```
In [ ]: # initializae the final output image without looks
        data_new = np.zeros([Naz, n_valid], dtype=np.complex128)
        print('New image output size: {}'.format(data_new.shape))

        info = dict()
        info['Fr']     = []
        info['R_dc']   = []
        info['tau_az'] = []

        # Run the Azimuth Processing code and take looks in azimuth
        for k in tqdm(np.arange(num_patch)):
            #print('patch {} of {} | data {}-{} | new data {}-{} '.format(k+1,num_patch,k*n_az_vali
        d,k*n_az_valid+num_az_echo, k*n_az_valid, (k+1)*n_az_valid))
            for i in np.arange(n_valid):
                # range at each range bin
                ri       = r0 + (i * delta_r)
                xi       = fd * wl * ri / 2 / veff        # assuming constant Doppler centroid
                r_dc_i   = np.sqrt(ri**2 + xi**2)
                fr_i     = -2*(veff**2)/r_dc_i/wl
                tau_az_i = 0.8 * r_dc_i * wl / veff / l

                info['Fr'].append(fr_i)
                info['R_dc'].append(r_dc_i)
                info['tau_az'].append(tau_az_i)

                # create azimuth reference chirp
                ref_az   = makechirp(N=num_az_echo, slope=fr_i, tau=tau_az_i, fs=PRF, fc=fd)[0]
                signal   = data_all[k*n_az_valid:k*n_az_valid+num_az_echo, i]
                newsig   = matched_filter(signal, ref_az)[0]

                data_new[k*n_az_valid:(k+1)*n_az_valid, i] = newsig[:n_az_valid]

            # multilook in azimuth
            for j in np.arange(int(Naveaz)):
                startline = k     * n_az_valid
                endline   = (k+1) * n_az_valid
                finalimage_ml[j+k*int(Naveaz), :] = np.mean(np.abs(data_new[startline+j*azlook : sta
        rtline+(j+1)*azlook,:]), axis=0)
```

```
  0%|          | 0/9 [00:00<?, ?it/s]

New image output size: (10152, 4200)

100%|██████████| 9/9 [00:08<00:00,  1.09it/s]
```

```
In [ ]: for key in info:
            print('{}:\t{}\tto\t{}'.format(key, np.min(info[key]), np.max(info[key])))
```

```
Fr:     -2165.1373023667484     to      -2081.8701348400255
R_dc:   830000.5880221013       to      863197.5664760073
tau_az: 0.5269988642509154      to      0.5480768853923952
```

The offset between patches, which will be the number of valid azimuth samples in each patch, is 2048-920 = 1128. So, for each patch, we will use 2048 azimuth samples to correlate with the azimuth reference chirp, and keep only 1128 of the output pixels. Since there are 10100 azimuth lines, the total number of patches we will have is $\lceil 10100/1128 \rceil$ = 9 patches. We will then have 1128*9 = 10152 azimuth lines in the final, non-multilooked image.

For each range bin, we need to compute the range to the center of the bin, the chirp rate, and the azimuth reference chirp pulse length. The ranges and pulse lengths can be computed using the same equations above, except instead of using the maximum range, rmax, we use the range bin index to calculate ri, where we replace nvalid with i, the range bin index. Then in the following equations, use the new range results for each bin. The chirp rate at each range bin, i, is then:

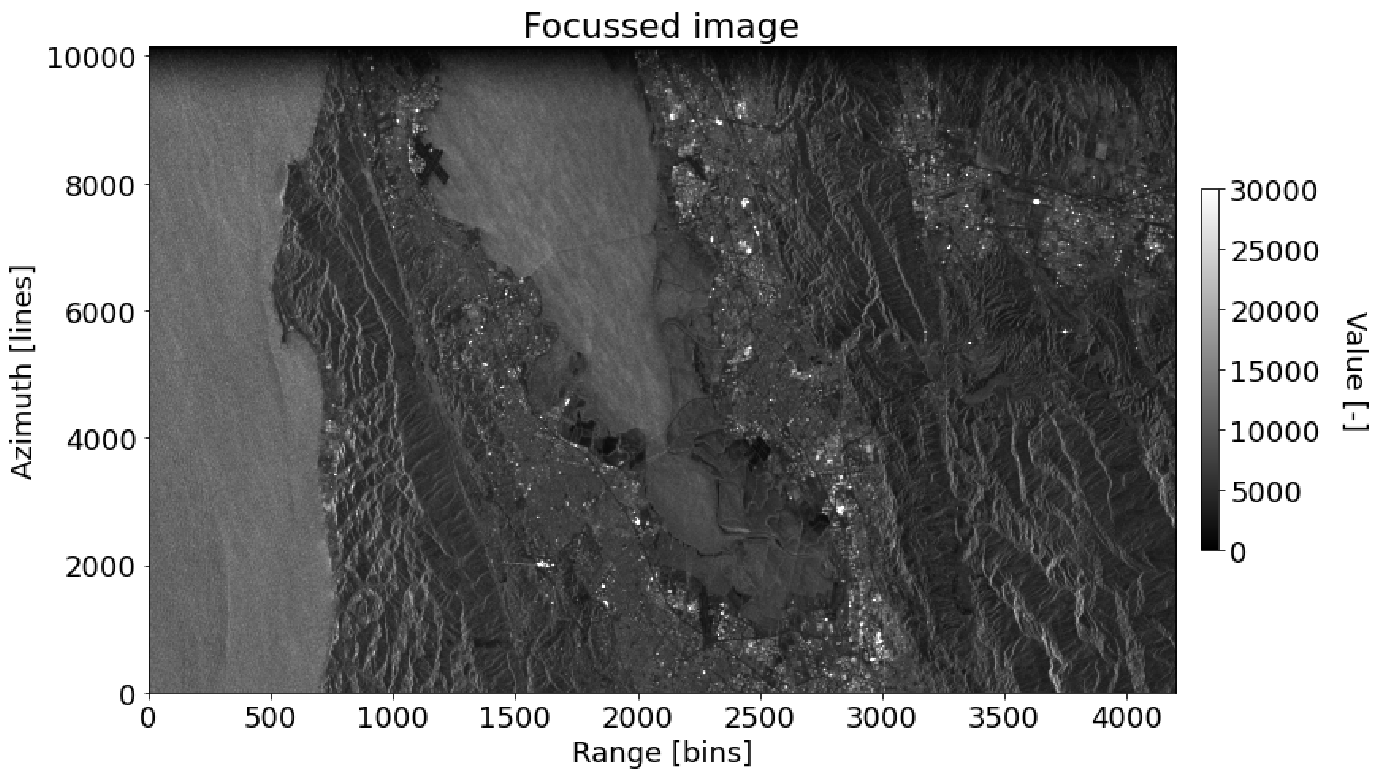
$$f_{R,i} = -\frac{2v_{eff}^2}{\lambda r_{dc,i}}$$

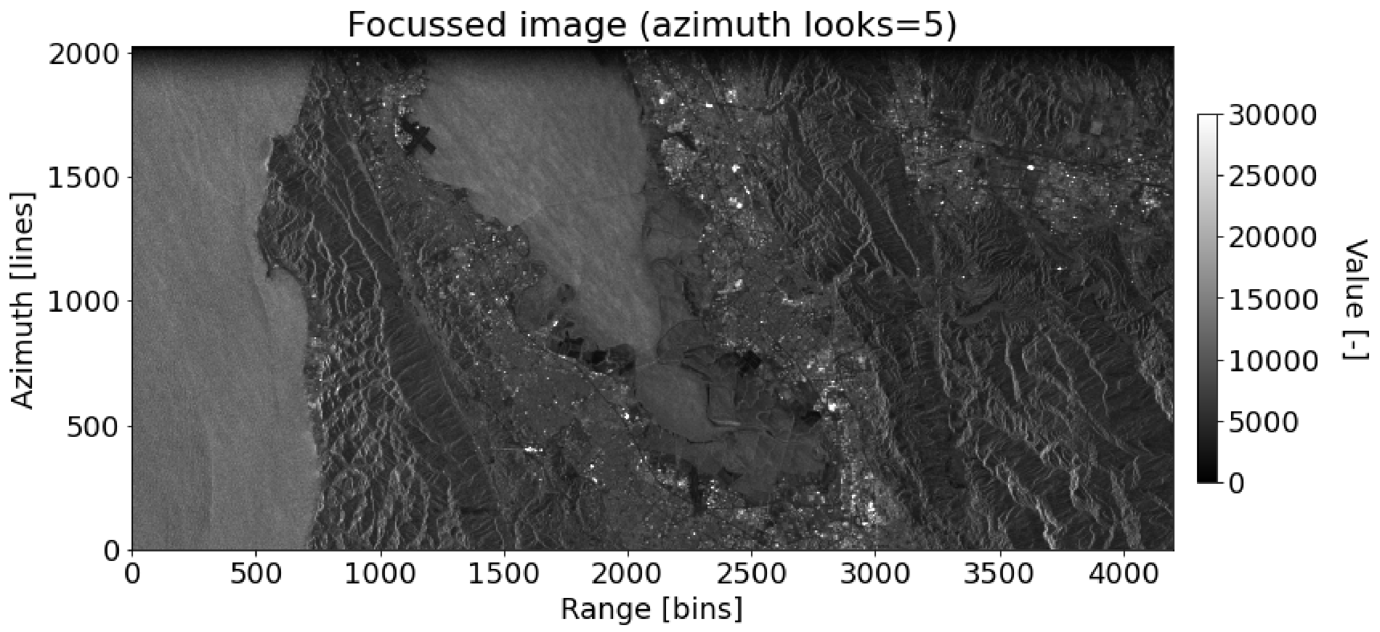
$$r_{dc,i} = r_0 + (i-1) * \Delta r$$

Then, the reference function for azimuth compressions is a chirp with slope given by the chirp rate $f_{R,i}$, duration $\tau_{az,i}$, sample rate PRF, and center frequency given by the Doppler centroid f_{dc}. We apply azimuth compression in the same way by performing the correlation in the frequency domain. We then save the magnitude of the compressed signal in the processed image, keeping only the valid bins, and combine the patches correctly.

```
In [ ]: print('Image size', data_new.shape)

        plot_img(np.abs(data_new), title='Focussed image', vlim=[0,3e4], origin='lower', aspect=0.26
        , interpolation='antialiased', savetif='focus.tiff')

        plot_img(np.abs(finalimage_ml), title='Focussed image (azimuth looks={})'.format(azlook), vl
        im=[0, 3e4], origin='lower', interpolation='antialiased', savetif='focus_look.tiff')
```

```
Image size (10152, 4200)
```





Problem 2 (c)

What is the azimuth resolution for your 80% bandwidth processor? What is the azimuth point spacing on the ground?

The azimuth ground pixel spacing is (deriving the expression from spherical earth geometry):

$$\Delta az_g = \frac{v}{PRF}(\frac{R_e}{z + R_e}) = (\frac{7550\ m/s}{1679.9\ Hz})(\frac{6378\ km}{6378\ km + 707.71758\ km}) = 4.01\ pixels$$

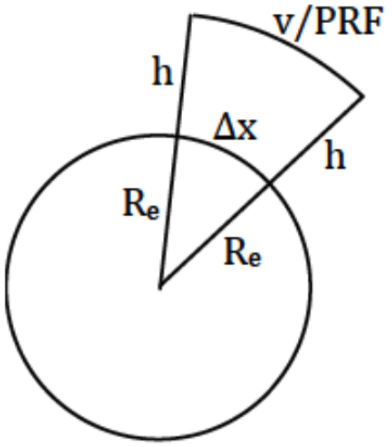


Figure 2: Spherical earth geometry to derive expression for azimuth ground pixel spacing.

```
In [ ]: ## We have described these above. Now redo it again:

        d_az = 1/2/0.8
        print('Resolution in azimuth direction for 80% of the beamwidth = {:.2f} m'.format(d_az))

        delta_az = (vx/PRF) * (Re/(Re+z))
        print('Ground spacing in azimuth direction = {:.2f} m'.format(delta_az))
```

```
Resolution in azimuth direction for 80% of the beamwidth = 6.25 m
Ground spacing in azimuth direction = 4.01 m
```

Problem 2 (d)

Calculate looks to obtain approximately square pixels on the ground, and display the image. Submit electronically.

```
In [ ]: # Num of looks: the ratio between them (the final image has plotted above)
        azlook = int(np.round(delta_gr/delta_az))
        print('Number of looks needed: {:.3f}'.format(azlook))
```

```
Number of looks needed: 5.000
```

```
In [ ]:
```