ELEC3305 Sonar Report

Group 10

Name	SID
Yiheng Yang	510231499
Haolin Jin	510085113
Xinyu Huo	520645453
Yiyang Hu	500707469

DDL：May 31

【Guideline from Canvas: When writing your code, you should provide copious comments and explanations in your LiveScript.

In addition to completing the lab code, you should put together a single succinct report (one report per team) that explains what each section of your code is doing. The report should also detail the contribution of each team member. There is a single mark for the team (all members share that mark).】

Part I: Time-domain SONAR

Task I: Generating the Chirp

1. Generate the formula of $f(t)$ and $\phi(t)$:

$$\phi(t) = 2\pi\left(f_0 + \frac{f_1 - f_0}{2T}t^2\right)$$

$$k = \frac{f_1 - f_0}{T}$$

$$f(t) = \frac{d\phi(t)}{2\pi dt} = f_0 + \frac{f_1 - f_0}{T}t$$

$$\phi(t) = 2\pi\int_0^T \left(f_0 + \frac{f_1 - f_0}{T}t\right)dt$$

$$\phi(t) = 2\pi\left[f_0 t + \frac{f_1 - f_0}{2T}t^2\right]_0^T$$

2. Generate time index and phase based on the formular:
   Firstly, we got **time index t** by using **np.r_[0:T:1/fs]**, which T is total time, fs is sample frequency and 1/fs stands for sample period. Then we got **ft = f0 + k*t** which k is gradient.

```
t = np.r_[0:T:1/fs]
k = (f1-f0)/T
ft = f0 + k*t
phi_of_t = 2*np.pi*(k/2)*(t**2) + f0*t
```

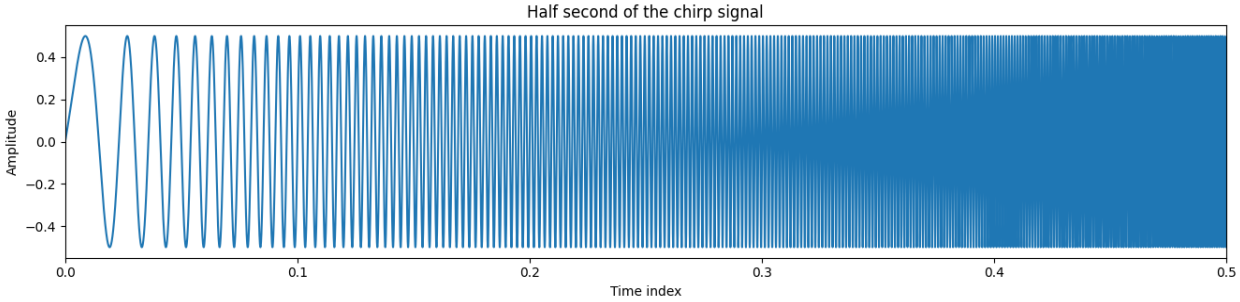
3. Generate chirp function:

$$x(t) = \sin(2\pi\int_0^t f(t')dt') = \sin(2\pi\int_o^t (f_0 + kt')dt') = \sin(2\pi(f_0 + \frac{k}{2}t)t)$$

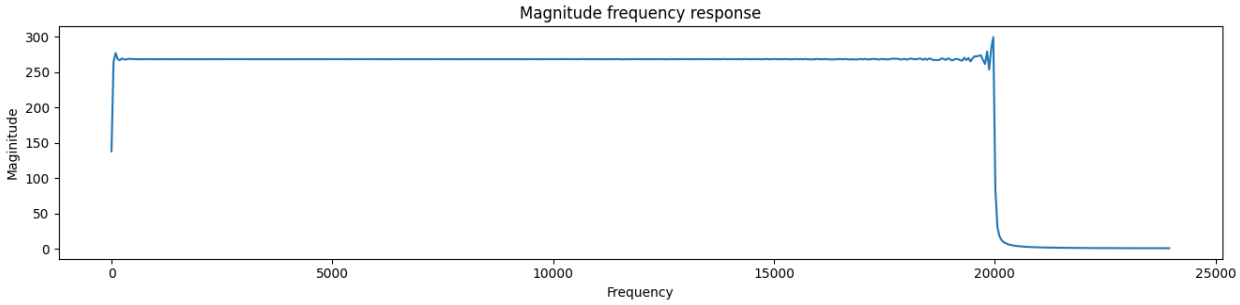
   The chirp function with amplitude of 0.5 by plugging the phase into a sinusoid so we need to multiply 0.5 as the final function.

```
# generate chirp signal
s_chirp = 0.5*np.sin(2*np.pi*(f0+(k/2)*t)*t)
```

4. Plot the first ½ second of the chirp:
   Firstly, we set the aspect ratio:
   Then, we use time index **t** and chirp signal **s_chirp**, set x axis by using **plt.xlim((0, 0.5))** to show the first ½ second.



5. Plot the magnitude frequency response of the sequence from 0 to π:
   Firstly we got an array of discrete frequency points **w** and the complex frequency response value of the corresponding frequency point **h** by using **signal.freqz(s_chirp)**.
   Then calculate the frequency **freq = w*fs/(2*np.pi).**
   Finally, do the plot which x axis is **freq** and y axis is the absolute value of frequency response **abs(h).**



Task II: Playing and Recording the Chirp

- Firstly we record the chirp signal with a sampling rate of 48000 Hz, and store it in the variable “rcv_chirp”.

```
## Play and record chirp at the same time

fs = 48000 # sampling rate = 48000 Hz

rcv_chirp = xciever( s_chirp, fs) # Qin = queue.Queue() Qout =
queue.Queue() / Note: queue instead of Queue for python3
```

- Now we use the “signal.freqz” function to get the angular frequencies and the complex frequency response. Then we divide it by 2π to get the frequency index

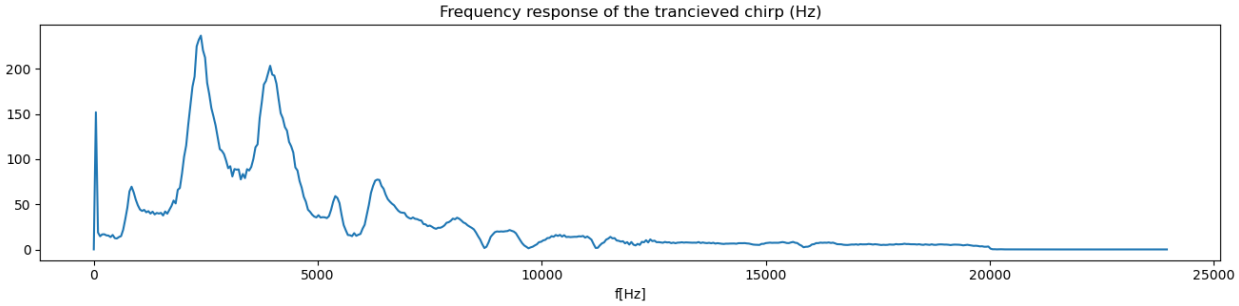
```
# generate frequency response of recorded chirp
w, h = signal.freqz(rcv_chirp)
# generate frequency index
f = w*fs/(2*np.pi)
```

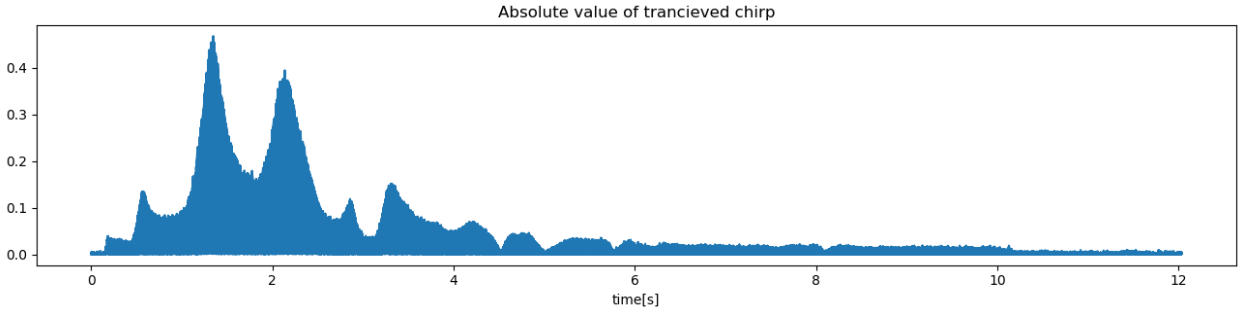
- We firstly plot the frequency response use complex frequency response

```
# generate a time index
t = np.r_[0:(len(rcv_chirp)/fs):(1/fs)]
# free code for your plot:
width, height = figaspect(0.2)
fig1 = figure(figsize=(width,height))
plt.plot(f, abs(h))
plt.title('Frequency response of the trancieved chirp (Hz)')
plt.xlabel('f[Hz]')
```

- Then use the abs() function to plot the absolute value of the chirp signal.

```
plt.plot(t, abs(rcv_chirp))
plt.title('Absolute value of trancieved chirp');
plt.xlabel('time[s]')
```





The above two figures give the frequency response and the absolute value plot of the chirp signal

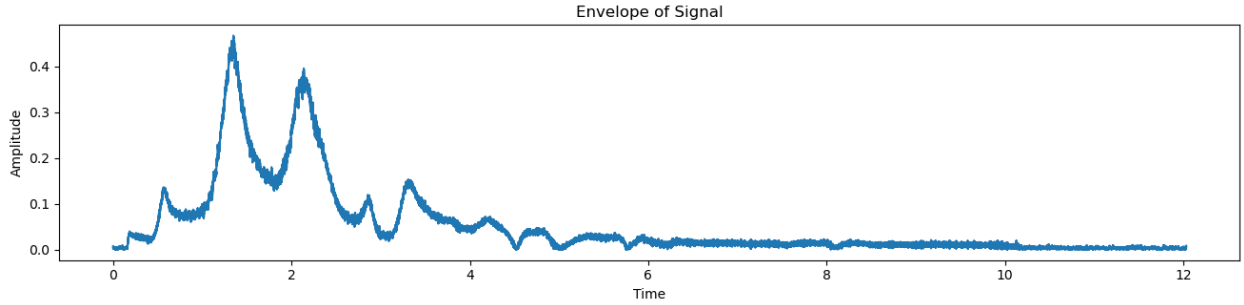
Task III: Envelope detection with Hilbert transform

- In task three we firstly use the “signal.hilbert” function to compute the analytic signal which is our recorded chirp signal. The x-axis is our time domain overall 12 seconds.

```
## Your lovely code here:
x = signal.hilbert(rcv_chirp)
t = np.r_[0:(len(abs(x))/fs):(1/fs)]

plt.plot(t, abs(x))
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.title("Envelope of Signal")
plt.show()
```

- We can see the plotted graph gives the envelope of the signal



Task IV: Auto-correlation Properties of the Chirp

- In the first part we will need to generate the chirp pulse sweep from 17KHz to 18KHz
    - We firstly setup parameters, T is the sample period of 512 samples, k is the rate of change, and use part 1 formula to derive the simultaneous frequnecy

```
# set params
n = 512
fs = 48000
f0 = 17000
f1 = 18000
T = n/fs
t = np.r_[0:T:1/fs]
k = (f1-f0)/T
f_t = f0+k*t
```

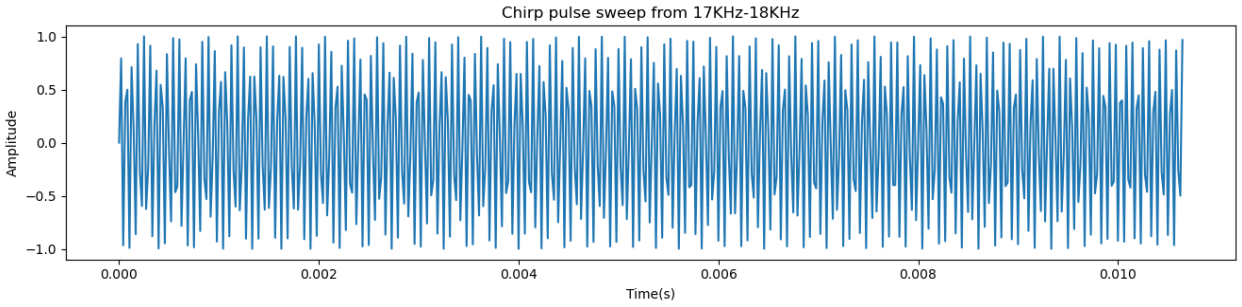
- Now we can use the previous formula to generate the chirp signal and frequency response

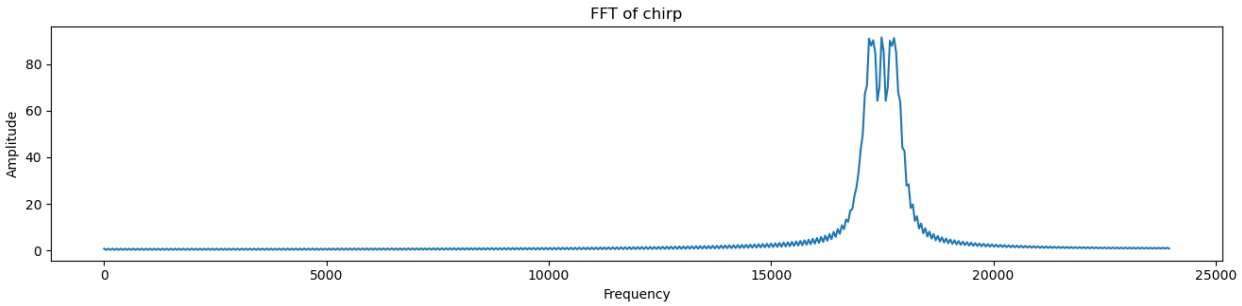
```
chirp = np.sin(2*np.pi*(f0+(k/2)*t)*t)
# generate frequency response of chirp
w, h = signal.freqz(chirp)
f = w*fs/(2*np.pi)
```

- Then we plot two graph, first one is chirp pulse and second one is FFT of chirp

```
width, height = figaspect(0.2)
figure(figsize=(width,height))
plt.plot(t, chirp)
plt.xlabel("Time(s)")
plt.ylabel("Amplitude")
plt.title("Chirp pulse sweep from 17KHz-18KHz")

width, height = figaspect(0.2)
figure(figsize=(width,height))
plt.plot(f, abs(h))
plt.xlabel("Frequency")
plt.ylabel("Amplitude")
plt.title("FFT of chirp")
plt.show()
```

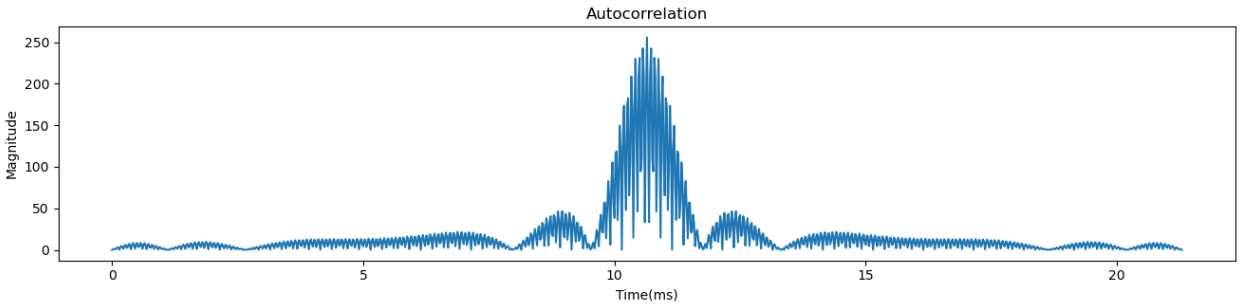




- Now we need to compute the autocorrelation of the chirp signal, we all know the theorem of the autocorrelation in python we flip the signal by x[::-1]

- So the flip variable is the inverse chirp signal, now we can use the “signal.convolve” function to do the convolution process to get the autocorrelation. Because the time domain is milliseconds so we times 1000 on the time, and plot the graph

```
flip = chirp[::-1]
auto_chirp = signal.convolve(chirp, flip)
x = np.r_[0:len(auto_chirp):1]/fs
width, height = plt.figaspect(0.2)
fig = plt.figure(figsize=(width,height))
plt.plot(x*1000, abs(auto_chirp))
plt.xlabel("Time(ms)")
plt.ylabel("Magnitude")
plt.title("Autocorrelation")
```



- Now we do the cross-correlation by using exponential to get another chirp signal:

```
phi_of_t = (f0+(k/2)*t)*t
s_chirp_a = np.exp(1j*phi_of_t*2*np.pi)
cross = signal.fftconvolve(s_chirp_a, flip)
```

- The main lobe (FWHM) can be calculated by the below formula, “tf” is the top half of the main lobe.

```
max_value = np.max(abs(cross))
half_value = max_value/2
tf = np.where(abs(cross) >= half_value)
fwhm = (tf[0][-1] - tf[0][0])/fs *1000
print('FWHM: {:.2f} ms'.format(fwhm))
```

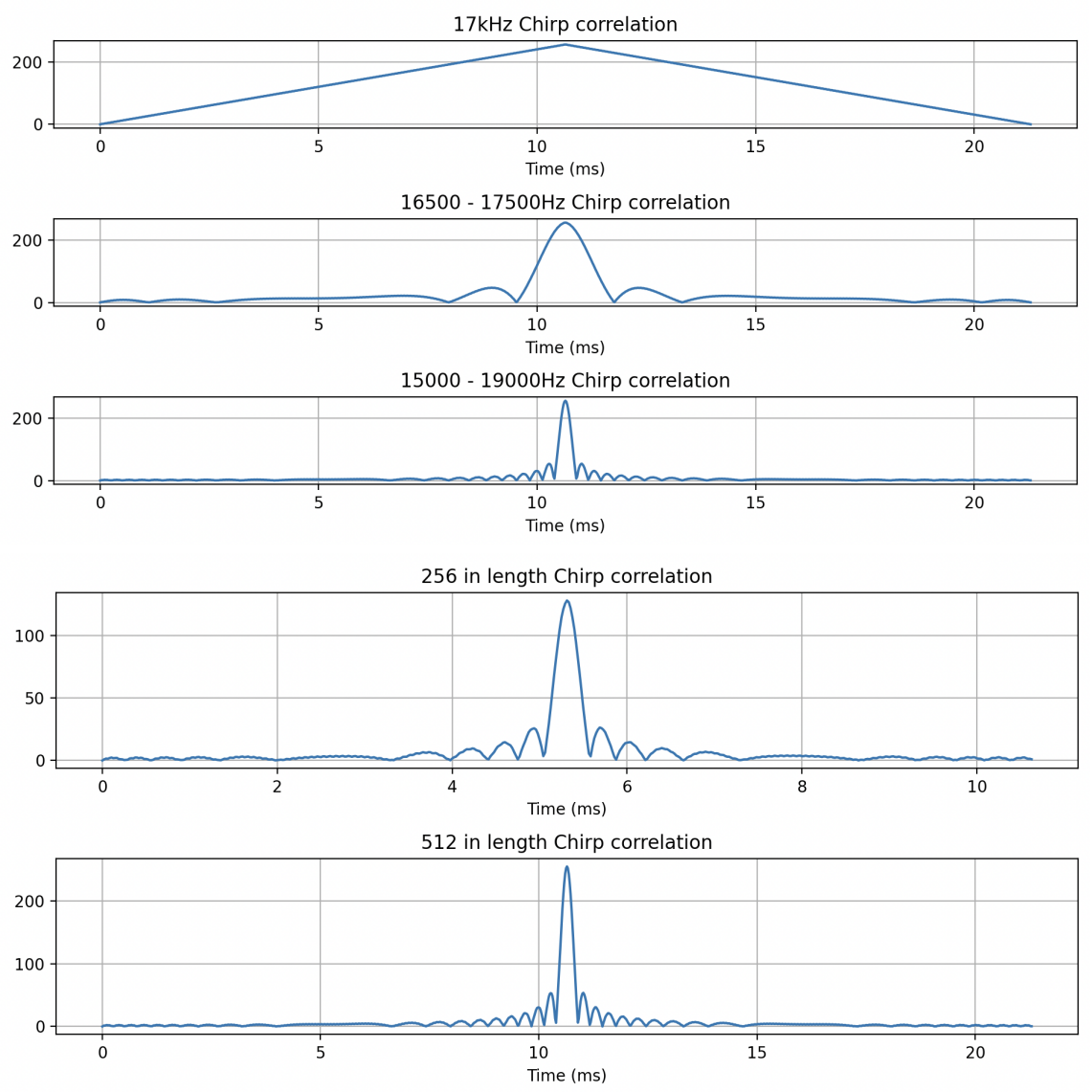
- Then we plot the graph, the FWHM will be calculated and printed out on the terminal

- Now we will look at why the chirp pulse is better for cross-correlation detection than a pure tone.

  In order to reduce the amount of code, we defined two functions:

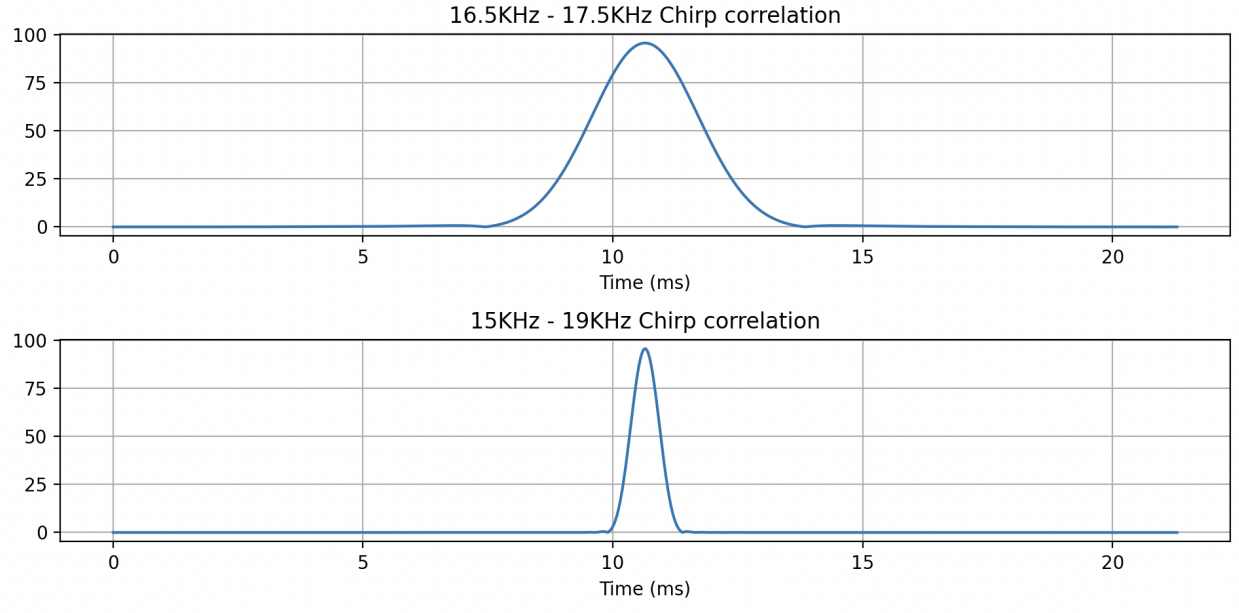
  1. **getCrossAndTime,** used to get the **time index** and **cross-correlation** by repeating the previous step.
  2. **calcFWHM,** used to print out the **FWHM, maximum autocorrelation,** and **compression ratio**.

  We get different results by using these functions and doing the plot.



Dealing with sidelobes

We first redefine the function **getCrossAndTime** by multiplying the **chirp** and its analytic function **s_chirp_a** with a hanning window. Then repeat the previous step to get the results.



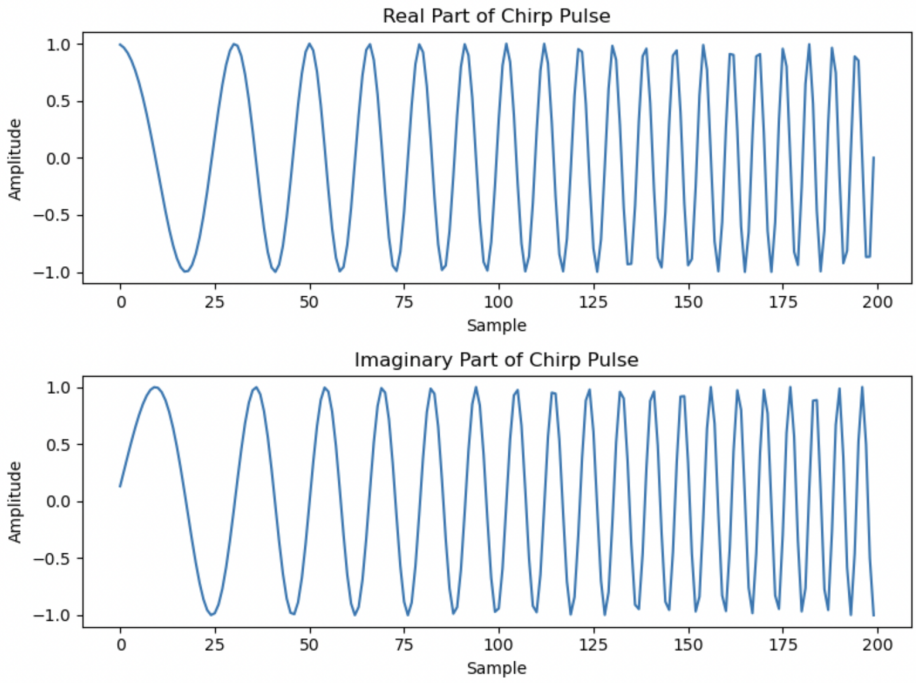
Part II: Time-domain Real-time SONAR

Task I : Generating Chirp Pulse

In this section of the lab, we focus on generating chirp pulses and pulse trains for the sonar system. The chirp pulses allow us to increase the pulse length while maintaining a higher bandwidth, which improves the signal-to-noise ratio and resolution of the sonar.

The first function we implemented is **genChirpPulse(Npulse, f0, f1, fs)**. This function generates an analytic function of a chirp pulse based on the provided parameters. The function first creates a time vector t based on the pulse length and sampling frequency. It then generates the instantaneous frequency of the chirp pulse using **np.linspace** with the starting and ending frequencies. The instantaneous frequency is integrated to obtain the phase of the chirp pulse. Finally, the chirp pulse is generated using Euler's formula **np.exp(1j * phase)**.

To validate the chirp pulse function, we generated a chirp pulse **pulse = genChirpPulse(200, 1000, 8000, 48000)**, and plotted the real part and imaginary part separately as follows.

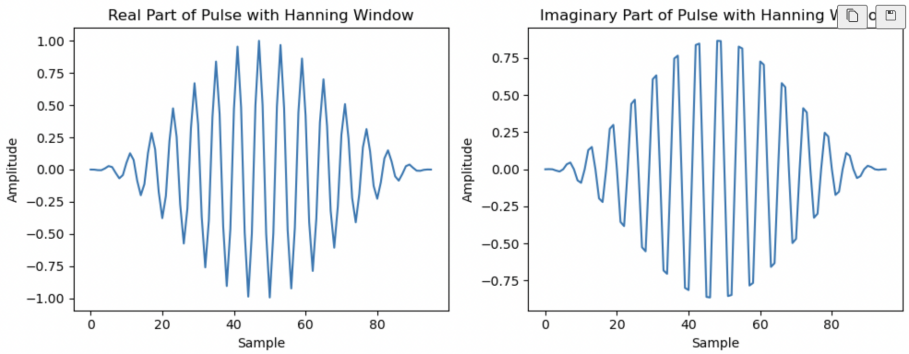


The next function we implemented is **genPulseTrain(pulse, Nrep, Nseg)**. The function first checks if the **Nseg** parameter is larger than the length of the pulse. If it is, it appends zeros at the end of the pulse to make its length equal to **Nseg**. Then, it uses **np.tile** to repeat the pulse **Nrep** times, creating a pulse train of length **Nrep x Nseg**.

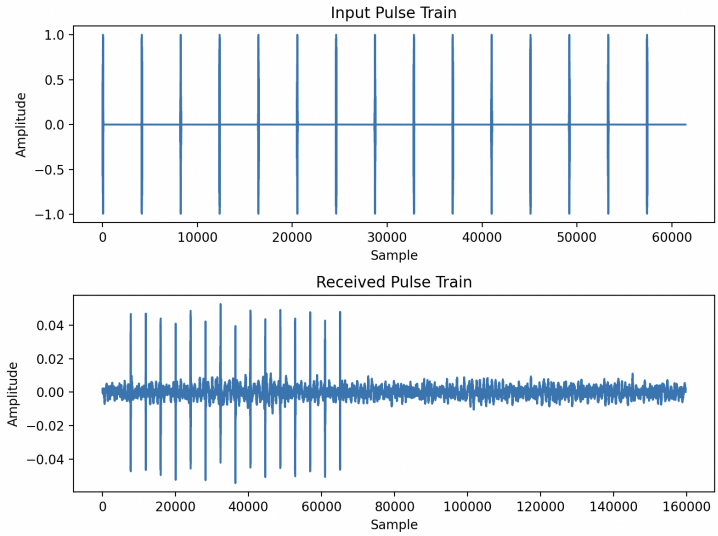
This concludes for Part II, Task I. In this section, we implemented functions to generate chirp pulses and pulse trains, which are fundamental components of the sonar system.

Task I : Echos in with Chirp pulse train

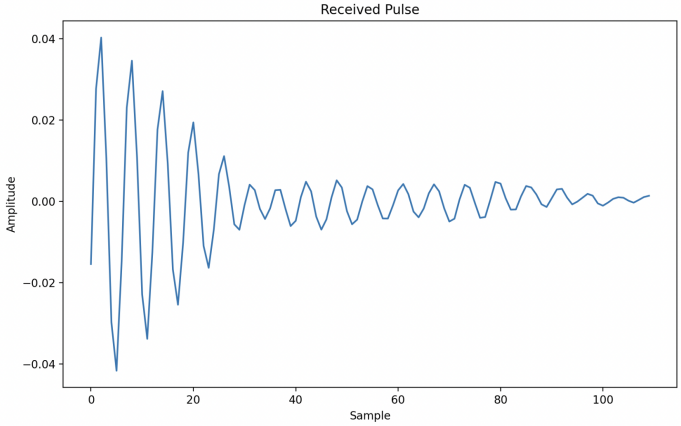
In this task, we will generate a pulse train using chirp pulses and analyze the received pulse train to detect echoes. First, we generate a chirp pulse with the following parameters: fs = 48000, f0 = 8000 , f1 = 8000 , Npulse = 96. We then apply a Hanning window to the pulse to smooth its edges, resulting in a pulse length of 2 ms. We plot the real and imaginary parts of the pulse to visualize its shape.



Next, we use the real part of the pulse to generate a pulse train with Nrep = 15 pulses and Nseg = 4096 samples. We play and record the pulse train, scaling the amplitude of the pulses to half. We then plot the input pulse train and the received pulse train

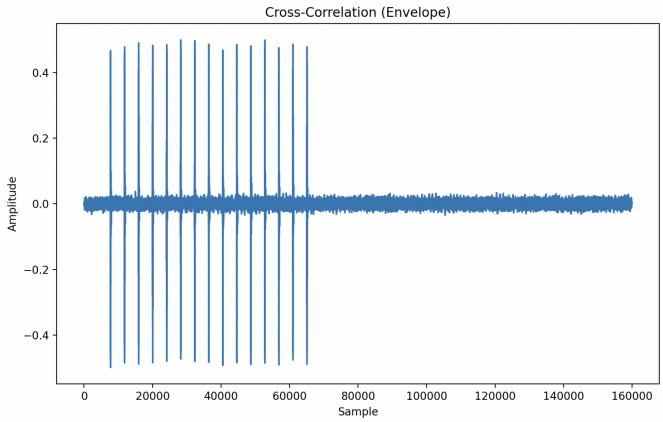


Next, we extract a single pulse from the received pulse train to analyze the echoes. We select the pulse index from the interactive plot to be 15960 and extract at least 2 * Npulse samples before the pulse and 20 * Npulse samples after it.

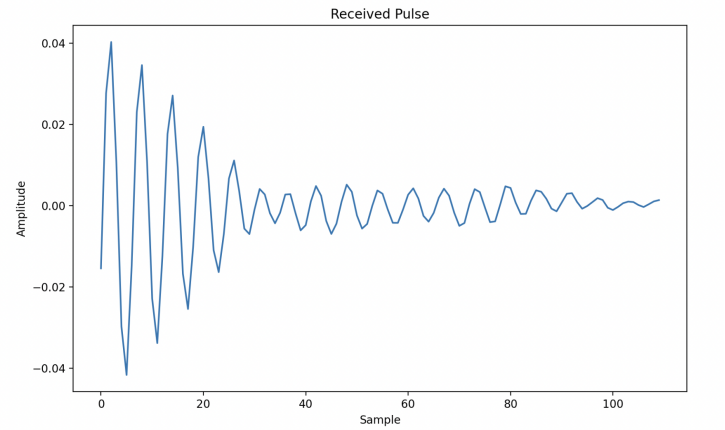


Then, we move to the matched filter part of the project. A function named crossCorr(rcv, pulse_a) was defined to calculate the cross-correlation (matched filter) of the received signal with the analytic function of the pulse. The signal.fftconvolve function was used to perform the cross-correlation operation.

The absolute value of the cross-correlation result was taken to recover its envelope. This envelope was assigned to the variable Xrcv_a. The envelope of the cross-correlation was plotted to visualize the matched filtering result. The plot displayed the amplitude of the cross-correlation over the sample index.



The Npulse variable represents the number of pulses to include before and after the target pulse. The idx variable determines the starting index of the pulse to be extracted from the received pulse train (set to 15960). The extracted pulse is then plotted, showing the amplitude over the sample index.

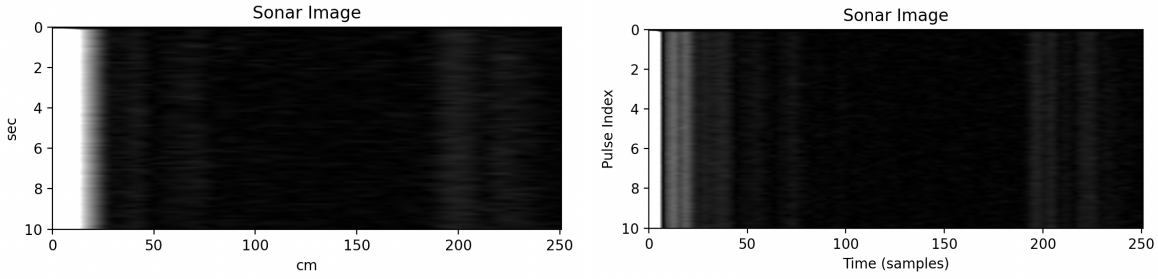


To automate the system and visualize the results, additional components are required. One of the important components is the ability to find the position of the first feedthrough pulse. The steps to implement this component are as follows:

A function named findDelay(Xrcv_a, Nseg) was defined to find the index of the first feedthrough pulse in the matched filtering result. The function uses the np.argmax function to find the index of the maximum value in a segment of the matched filtering result. The Nseg variable represents the length of a segment used to search for the first pulse. The function returns the index of the beginning of the first pulse.

The final step is to convert the time between echoes into actual distance using the speed of sound in air. Two functions, dist2time(dist, temperature) and time2dist(t, temperature), were defined to perform the conversions. The dist2time function converts the distance to the target in centimeters to the time in seconds between the transmitted pulse and its echo. The time2dist function converts the time to the target in seconds to the distance in centimeters between the transmitted pulse and its echo.

In the final part of the report, a function named sortOfASonar is discussed, which utilizes the implemented functions to generate pulses and display the matched filtering of each pulse as intensity in an image. This function allows for the detection of echoes and tracking of the object's distance. The sortOfASonar function generates a pulse train based on the specified parameters, applies matched filtering, and extracts the echoes from the received signal. It then constructs an image by representing the intensity of each pulse's matched filtering result as a horizontal line. The resulting image is displayed.



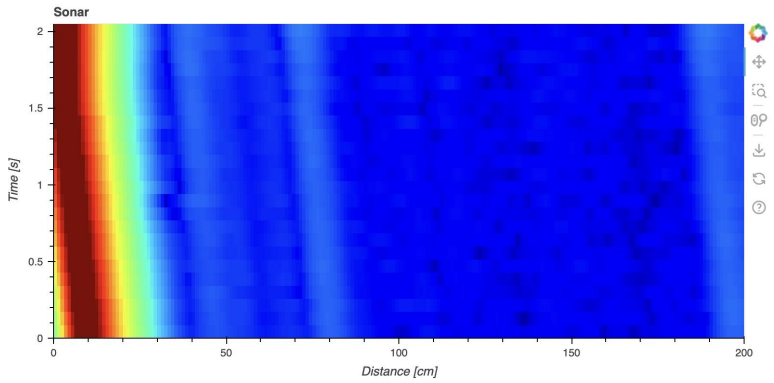
Task III: Play with different parameters with the real-time sonar

1. Import functions and libraries, as well as **rtsonar.py**.
2. Copy the functions genPulseTrain(), genChirpPulse(), crossCorr(), findDelay(), and dist2time(), for the following codes to call.
    - **rtsonar():** Used to create a real-time sonar plot.
    - **genChirpPulse(Npulse, f0, f1, fs):** It generates a chirp pulse.
    - **ggenPulseTrain(pulse, Nrep, Nseg):** This function uses the pulse generated by genChirpPulse, parameter “Nrep” means the number of pulse repetitions, and “Nseg” means the length of each pulse train segment. “Ptrain” will finally be returned as a vector and the size will be “Nrep x Nseg”.
    - **crossCorr(rcv, pulse_a):** Calculate the cross-correlation of the received signal with the analytic function of the pulse.
    - **findDelay(Xrcv, Nseg):** Get the result of the matched filtering and finds the index of the first feedthrough pulse.
    - **dist2time(dist, temperature=21):** Used to convert the time between echoes to actual distance. The distance between the echo and the sound source can be calculated by the echo and its traveling time.
3. **Main function:**
    - **functions = (genChirpPulse, genPulseTrain, crossCorr, findDelay, dist2time) :** In order to make all the functions joined to take effect.
    - **stop_flag = rtsonar(f0, f1, fs, Npulse, Nseg, Nrep, Nplot, maxdist, temperature, functions):** Set a stop flag to prevent the program from running, according to the condition of the rtsonar.py file.
4. We finally run this sonar with changing of parameters. **The initial group of parameters** is as follows:

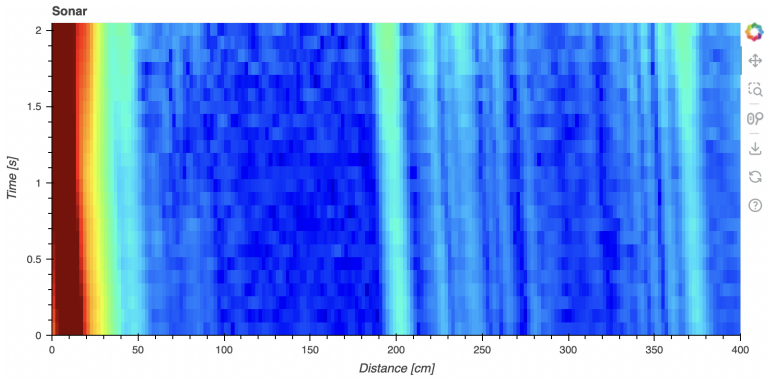
```
fs = 48000 # Sampling frequency
f0 = 6000 # Chirp initial frequency
f1 = 12000 # Chirp ending frequency

Npulse = 500 # Length of Chirp Pulse
Nrep = 24 # Number of repetitions in a pulse train which determines the vertical value of
the picture
Nseg = 2048*2 # Number of samples between pulses which determines maximum arrival time
Nplot = 200 # Number of pixels in the horizontal axis, a higher resolution is slower
maxdist = 200 # Maximum distance value
temperature = 20 # Temperature parameter used for function
```

- **The processing result of these parameters:**



- We slowly moved the computer away from the wall, so we got red lines that were not vertical. The distance between the indicating wall and the computer was approximately 5cm. The read value of the distance would be over 5cm. This is an initial test.
- We used the book to block the computer radio, so the result is that the red curve has no obvious inclination. The slight error that causes the image to bend is due to the shaking of holding the book.

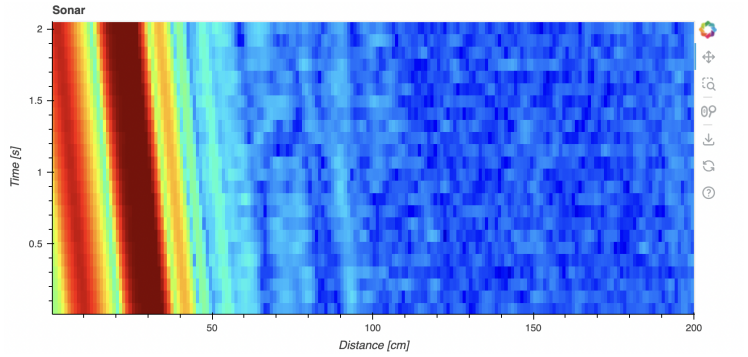


5. **Increased f1**(chirp ending frequency) to 18000Hz.

```
fs = 48000 # Sampling frequency
f0 = 6000 # Chirp initial frequency
f1 = 18000 # Chirp ending frequency

Npulse = 500 # Length of Chirp Pulse
Nrep = 24 # Number of repetitions in a pulse train which determines the vertical value of
the picture
Nseg = 2048*2  # Number of samples between pulses which determines maximum travel time
Nplot = 200 # Number of pixels in the horizontal axis, a higher resolution is slower
maxdist = 200 # Maximum distance value
temperature = 20 # Temperature parameter used for function
```

- **The processing result of these parameters:** According to our understanding, a higher ending frequency is more suitable for detecting small targets at short distances. In this test, all the detections were performed at relatively shorter distances, so no significant advantage was observed. Multiple red lines may relate to multiple objects the computer was facing.

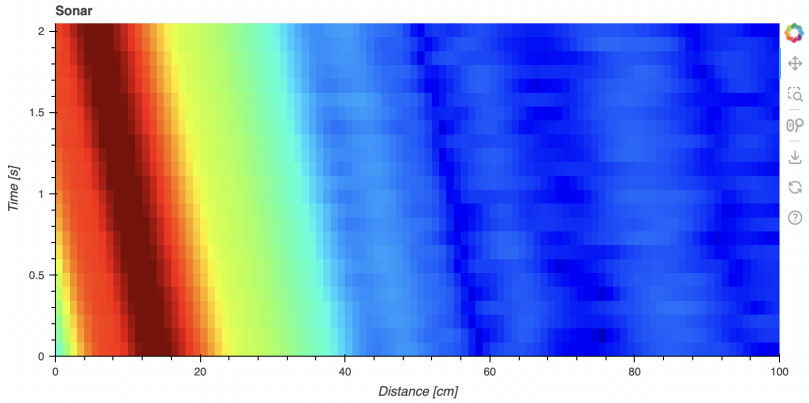


6. Decrease **Nplot** to 100, and we can get a picture with a lower resolution, and the x-axis range is decreased.

```
fs = 48000 # Sampling frequency
f0 = 6000 # Chirp initial frequency
f1 = 12000 # Chirp ending frequency

Npulse = 500 # Length of Chirp Pulse
Nrep = 24 # Number of repetitions in a pulse train which determines the vertical value of
the picture
Nseg = 2048*2 # Number of samples between pulses which determines maximum travel time
Nplot = 100 # Number of pixels in the horizontal axis, a higher resolution is slower
maxdist = 100 # Maximum distance value
temperature = 10  # Temperature parameter used for function
```

- **The processing result of these parameters:** Short horizontal resolution is more suitable for short-distance object detection, especially in our cases. The value of distances will be easier to read.

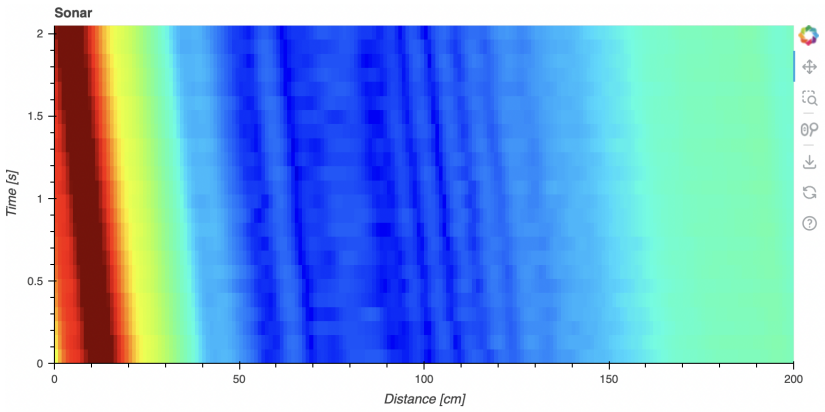


7. The “**Npulse**” parameter was doubled. According to the course note, the longer the pulse is, the worse time resolution we will get, but we will gain a much stronger matched filtering amplitude. Pulse compression with chirp pulses can increase the time resolution, but the bandwidth will be increased alternatively. We decided to increase the “Npulse” in order to gain the reliability of echo signal detection, on the other hand, it can improve the frequency resolution of sonar so that we can read the value precisely.

```
fs = 48000 # Sampling frequency
f0 = 6000 # Chirp initial frequency
f1 = 12000 # Chirp ending frequency

Npulse = 1000 # Length of Chirp Pulse
Nrep = 24 # Number of repetition in a pulse train (determines vertical length of plot )
Nseg = 2048*2 # Number of samples between pulses (determines maximum time-of-arrival)
Nplot = 200 # Number of pixels in plot along horizontal direction (higher resolution is
slower)
maxdist = 200 # Maximum distance in cm
temperature = 20 # Temperature
```

- **The processing result of these parameters:** We still tested the computer away from the wall, for approximately 10cm. The distance value of this result picture would be around 10cm, too. The narrowing of the red line segment also makes it easier to read the distance, and the value we got is closer to the actual distances we measured. In conclusion, the increase of the pulse width amplified the energy of the signal, in this case, may detect targets more accurately.



Group Contribution

- Yiheng Yang: Responsible for the part 1 coding and report.
- Haolin Jin: Responsible for the part 1 coding and report.
- Yiyang Hu: Responsible for part 2 coding and report.
- Xinyu Huo: Responsible for part 2 coding and report.