汤振礼

Theoretical Background

1.FM:

s(t) = rect($\frac{t-t_0}{T}$)exp(j π K $(t-t_0)^2$)

2.Matched Filtering:

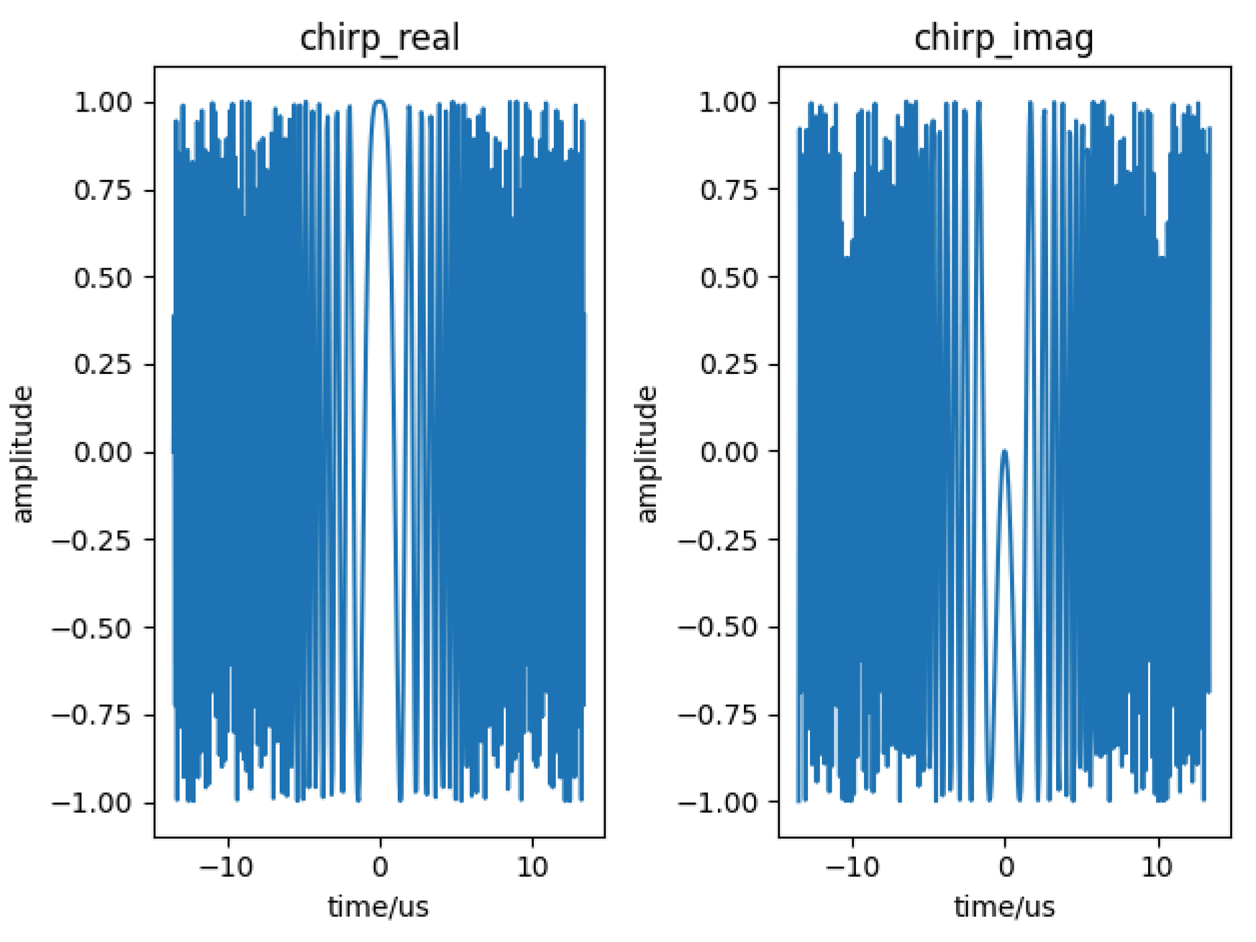
h(t) = rect($\frac{t}{T}$)exp(-j π K t^2)

s_{out}(t) = conv(s(t),h(t)) $\approx Tsinc(KT(t-t_0))$

Experimental results and analysis

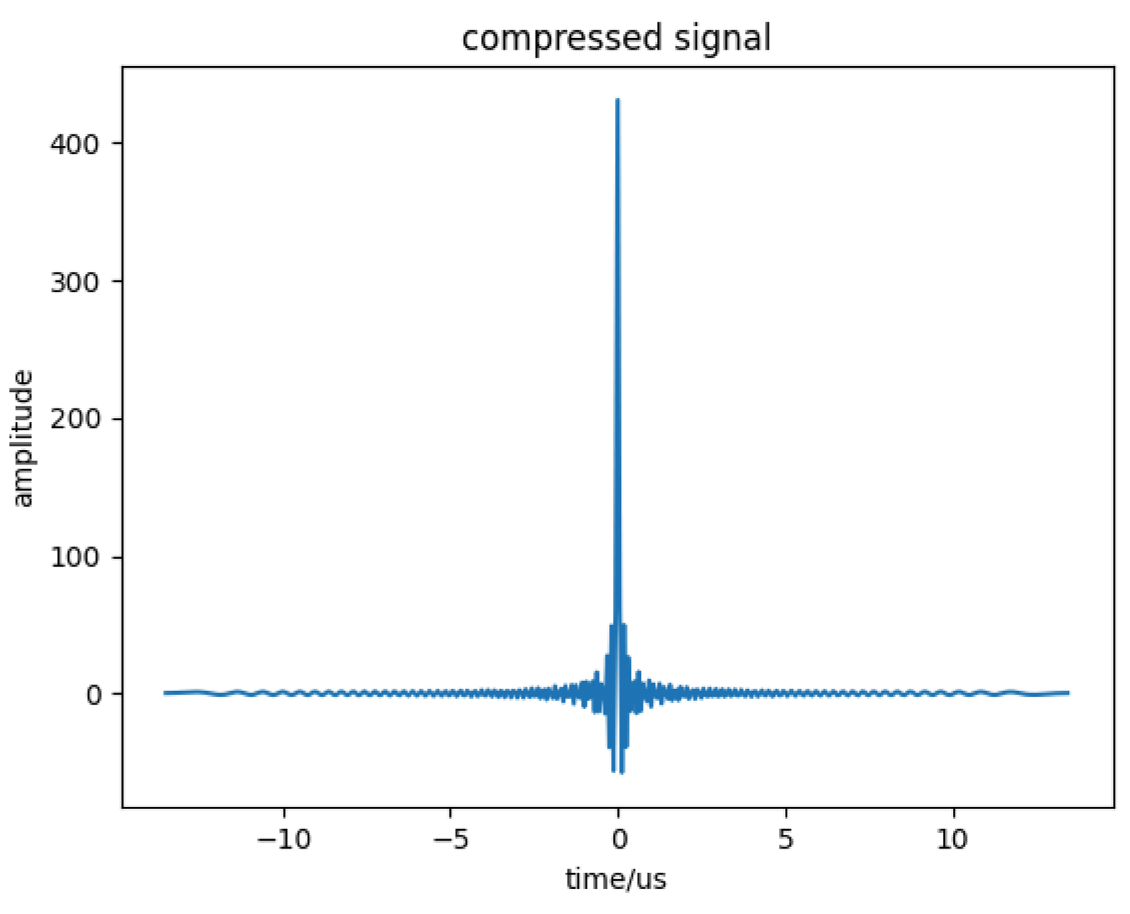
In [169…
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.signal as ss
import scipy.fftpack as sf
```

In [170…
```
chirp_direction = -1
chirp_bandwidth = 1.4e+7
chirp_duration = 2.7e-5
chirp_slope = chirp_direction*chirp_bandwidth/chirp_duration
ADC_sampling_frequency = 1.600000e+07

t_interval = 1/ADC_sampling_frequency
N = int(chirp_duration/t_interval)
t_l = np.linspace(-chirp_duration/2,chirp_duration/2-t_interval,N)
x_chirp = np.exp(1j*np.pi*chirp_slope*(t_l**2))
plt.figure()
plt.subplot(1,2,1)
plt.plot(t_l*1e+6,np.real(x_chirp))
plt.xlabel('time/us')
plt.ylabel('amplitude')
plt.title('chirp_real')
plt.subplot(1,2,2)
plt.plot(t_l*1e+6,np.imag(x_chirp))
plt.xlabel('time/us')
plt.ylabel('amplitude')
plt.title('chirp_imag')
plt.tight_layout()
plt.show()
```

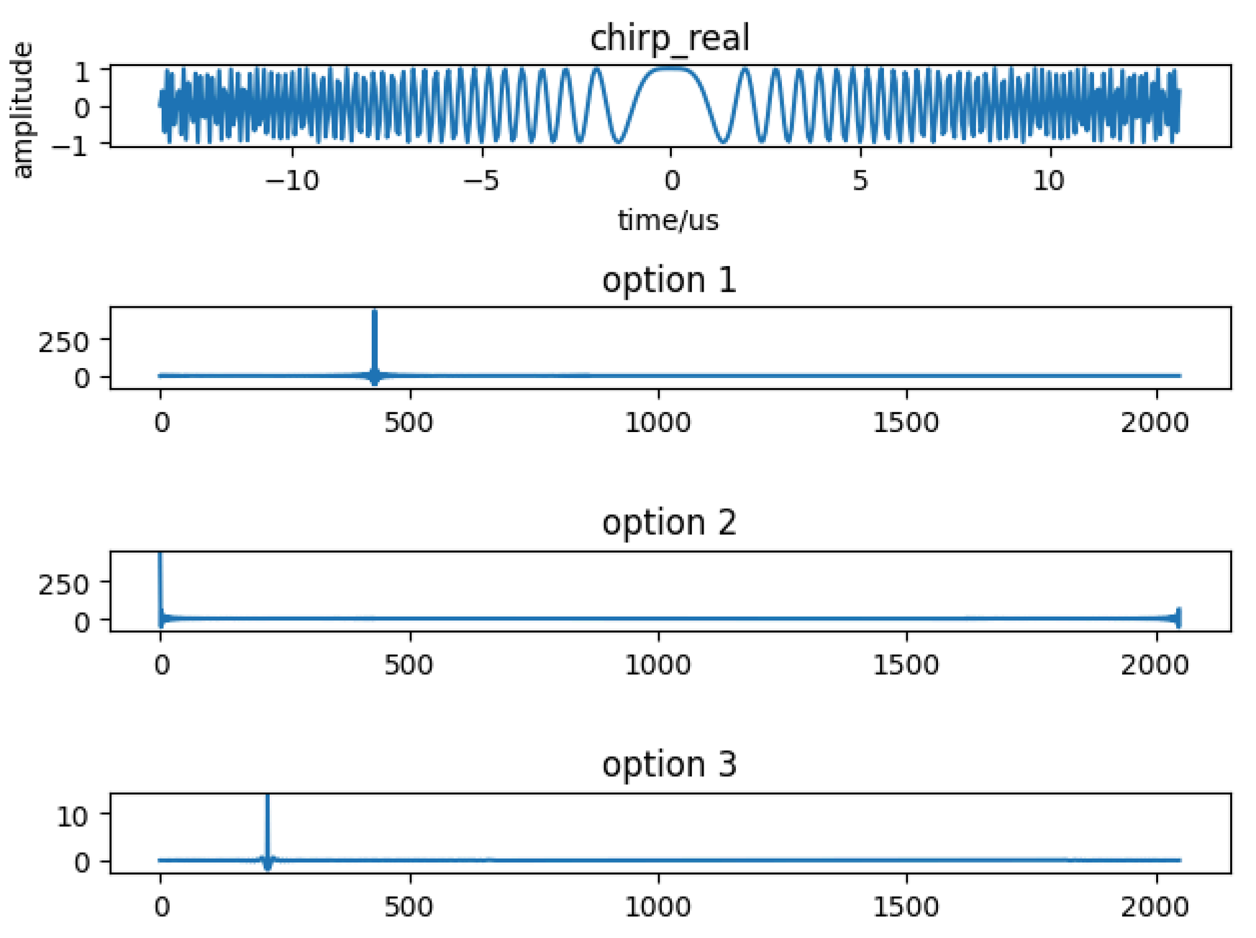


In [171…
```
h_m = np.exp(-1j*np.pi*chirp_slope*(t_l**2))
x_mf = ss.convolve(x_chirp,h_m)
t_0_idx = np.argmax(x_mf)
plt.figure()
plt.plot(t_l*1e+6,np.real(x_mf[t_0_idx-int(N/2):t_0_idx+int(N/2)]))
plt.title('compressed signal')
plt.xlabel('time/us')
plt.ylabel('amplitude')
```

Out[171]:
```
Text(0, 0.5, 'amplitude')
```



In [172…
```
N_fft = 2048
t_0 = 0
x_chirp_shift = np.exp(1j*np.pi*chirp_slope*((t_l-t_0)**2))
x_chirp_fft = sf.fft(x_chirp_shift,N_fft)
x_chirp_fft_shift = np.fft.fftshift(x_chirp_fft)
f_l = np.linspace(-ADC_sampling_frequency/2,ADC_sampling_frequency/2-ADC_sampling_frequency/N_fft,N_fft)
t_n = np.array(range(N_fft))
plt.figure()

plt.subplot(4,1,1)
plt.plot(t_l*1e+6,np.real(x_chirp_shift))
plt.xlabel('time/us')
plt.ylabel('amplitude')
plt.title('chirp_real')

#option 1
plt.subplot(4,1,2)
h_1 = np.conjugate(np.flip(x_chirp_shift))
h_1_fft = sf.fft(h_1,N_fft)
x_1_mf = sf.ifft(x_chirp_fft*h_1_fft)
plt.plot(t_n,np.real(x_1_mf))
plt.title('option 1')

#option 2
plt.subplot(4,1,3)
h_2 = x_chirp_shift
h_2_fft = np.conjugate(sf.fft(h_2,N_fft))
x_2_mf = sf.ifft(x_chirp_fft*h_2_fft)
plt.plot(t_n,np.real(x_2_mf))
plt.title('option 2')

#option 3
plt.subplot(4,1,4)
h_3_fft = np.exp(1j*np.pi/chirp_slope*(f_l**2))
x_3_mf = sf.ifft(x_chirp_fft_shift*h_3_fft)
plt.plot(t_n,np.real(x_3_mf))
plt.title('option 3')

plt.tight_layout()
```



对于序列{x[n],n=0,1,...,L},作N点(N $\geq$ L)FFT加匹配滤波得到输出序列{y[n],n=0,1,...,N-1}:

对于Option 1：先反褶在取共轭，可以推导得到匹配滤波的极大值点位于L-1点

对于Option 2：先FFT再取共轭，可以推导得到匹配滤波的极大值点位于0点

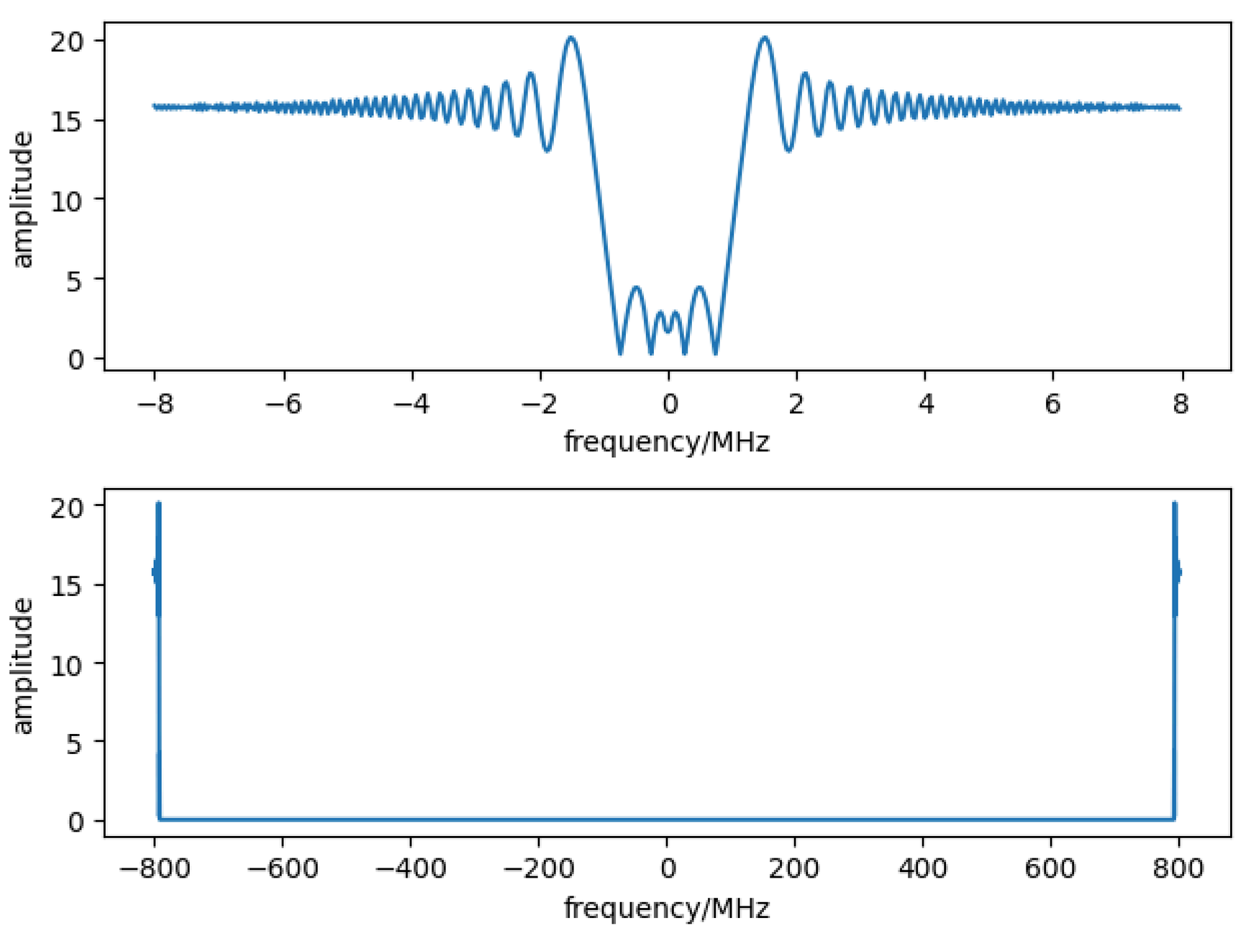
对于Option 3：直接频域匹配滤波，可以推导得到匹配滤波的极大值点位于原信号的零频点

In [173…
```
print('length of chirp:',len(x_chirp))
print('option 1 idx:',np.argmax(np.real(x_1_mf)))
print('option 2 idx:',np.argmax(np.real(x_2_mf)))
print('option 3 idx:',np.argmax(np.real(x_3_mf)))
```

```
length of chirp: 432
option 1 idx: 431
option 2 idx: 0
option 3 idx: 216
```

点目标分析

In [174…
```
#option 3
plt.figure()
x_3_fft_shift = np.fft.fftshift(sf.fft(np.real(x_3_mf[0:len(x_chirp)])))
f_l = np.linspace(-ADC_sampling_frequency/2,ADC_sampling_frequency/2-ADC_sampling_frequency/N,N)
# f_l = np.fft.fftfreq(N,t_interval)
plt.subplot(2,1,1)
plt.plot(f_l*1e-6,np.abs(x_3_fft_shift))
plt.xlabel('frequency/MHz')
plt.ylabel('amplitude')

#100 oversampling
S = 100
x_os_fft = np.zeros(S*N,dtype=np.complex128)
x_os_fft[0:int(N/2)] = x_3_fft_shift[0:int(N/2)]
x_os_fft[S*N-int(N/2):S*N] = x_3_fft_shift[int(N/2):N]
f_l_os = np.linspace(-S*ADC_sampling_frequency/2,S*ADC_sampling_frequency/2-ADC_sampling_frequency/N,S*N)
plt.subplot(2,1,2)
plt.plot(f_l_os*1e-6,np.abs(x_os_fft))
plt.xlabel('frequency/MHz')
plt.ylabel('amplitude')
plt.tight_layout()
```



In [177…
```
x_os = np.real(sf.ifft((x_os_fft)))
x_os = x_os/np.max(np.abs(x_os))
t_interval_os = t_interval/S
t_os = np.linspace(-chirp_duration/2,chirp_duration/2-t_interval_os,S*N)
plt.figure()
plt.subplot(2,1,1)
plt.plot(t_os*1e+6,20*np.log10(np.abs(x_os)))
plt.xlabel('time/us')
plt.ylabel('amplitude')
plt.ylim(-40,0)
plt.xlim(-1,1)
plt.subplot(2,1,2)
plt.plot(t_os*1e+6,(np.angle(x_os)))
plt.xlabel('time/us')
plt.ylabel('angle')
plt.xlim(-1,1)
plt.tight_layout()
t_idx_peak = np.abs(x_os).argmax()
t_idx_3dB = np.abs((np.abs(x_os)**2-0.5)).argmin()
IRW = np.abs(2*(t_os[t_idx_peak]- t_os[t_idx_3dB]))
t_elex_index = ss.argrelextrema(np.abs(x_os),np.greater)
t_elex=np.sort(np.abs(x_os)[t_elex_index])
PSLR = 10*np.log10((t_elex[-2]/t_elex[-1])**2)

#将极值点按主瓣距离排序
t_index = np.array(t_elex_index)[:,(np.argsort(np.abs(np.array(t_elex_index) -t_idx_peak)))].flatten()
t_zero_index = ss.argrelextrema(np.abs(x_os[min(t_index[1],t_index[2]):max(t_index[1],t_index[2])]),np.less)
ISLR = 10*np.log10(np.sum(np.abs(x_os)**2)/np.sum(np.abs(x_os[min(t_index[1],t_index[2])+t_zero_index[0][0]:\
                                                      min(t_index[1],t_index[2])+t_zero_index[0][1]])**2))

print('IRW(us):',IRW*1e+6)
print('PSLR(dB):',PSLR)
print('ISLR(dB)',ISLR)
```

```
IRW(us): 0.06375000000000088
PSLR(dB): -13.086170829714865
ISLR(dB) 0.4931431362047403
```