# Linear Chirp Matched filtering tutorial

We will generate a data realization containing a linear chirp signal and a realization of i.i.d. Normal random variable sequence (White Gaussian Noise). Matched filtering (least squares fitting) will be applied to this data to estimate the signal parameters. Mathematical details can be found in Lecture 2.

## Contents



Number of samples in data, sampling frequency, and sampling times.

```
nSamples = 2048;%samples
fs = 1024;%Hz (Sampling frequency)
timeVec = (0:(nSamples-1))/fs;
```

```
%Length of data in seconds
dataLen = timeVec(end);
```

## Parameters of the linear chirp signal.

```
f0 = 200;%Hz
f1 = 50; %Hz^2
sigLen = 0.5;%sec
initialPhase = pi/3.3;%radians
sigAmplitude = 10;
timeOfArrival = 0.2;%sec
```

Find the samples where the signal should start and end. Store the corresponding time samples.

```
startSample = floor(timeOfArrival*fs);
endSample = floor((timeOfArrival+sigLen)*fs);
sigTimeVec = timeVec(startSample:endSample);
```

## Generate the signal

```
%First allocate an empty array that is as long as the data. Then load the
%signal between start and end samples corresponding to the time of arrival
%and the end of the signal.
sigVec = zeros(1,nSamples);
sigVec(startSample:endSample) = sin(2*pi*(f0*(sigTimeVec-timeOfArrival)+...
                    f1*(sigTimeVec-timeOfArrival).^2)+...
                    initialPhase);
```

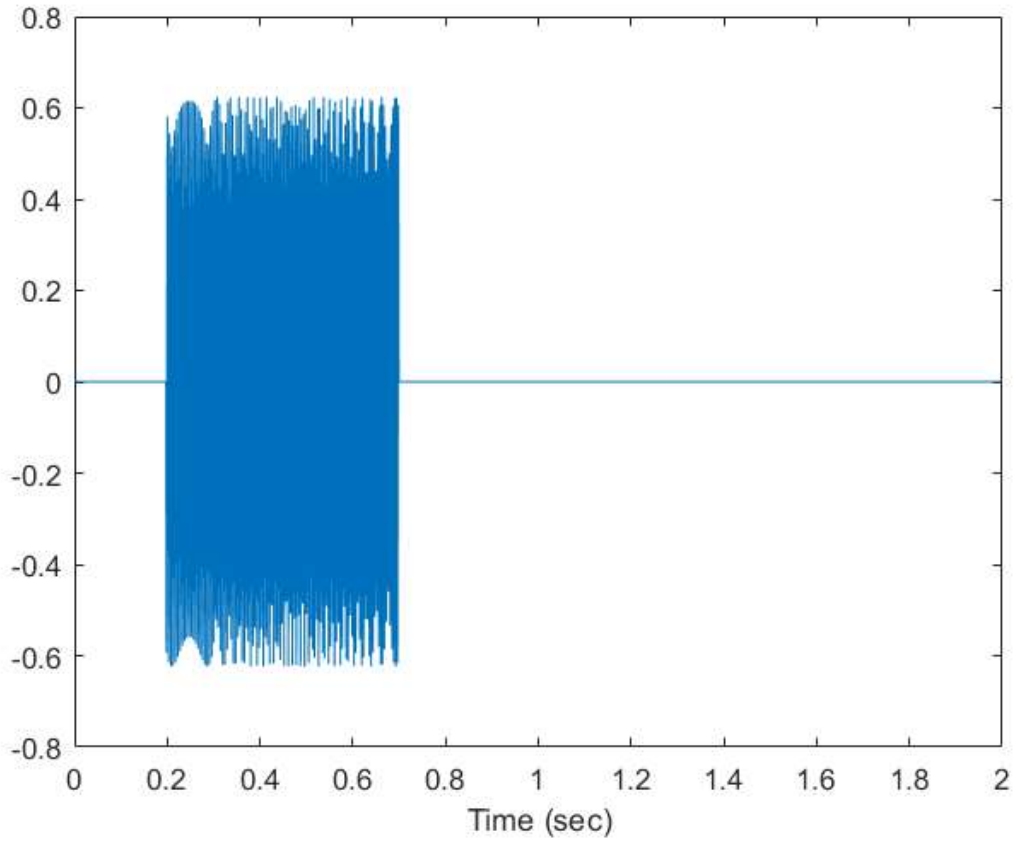
Normalize the signal and multiply with the amplitude parameter.

```
sigVec = sigVec/norm(sigVec);
sigVec = sigAmplitude*sigVec;
```

## Plot the signal in different domains

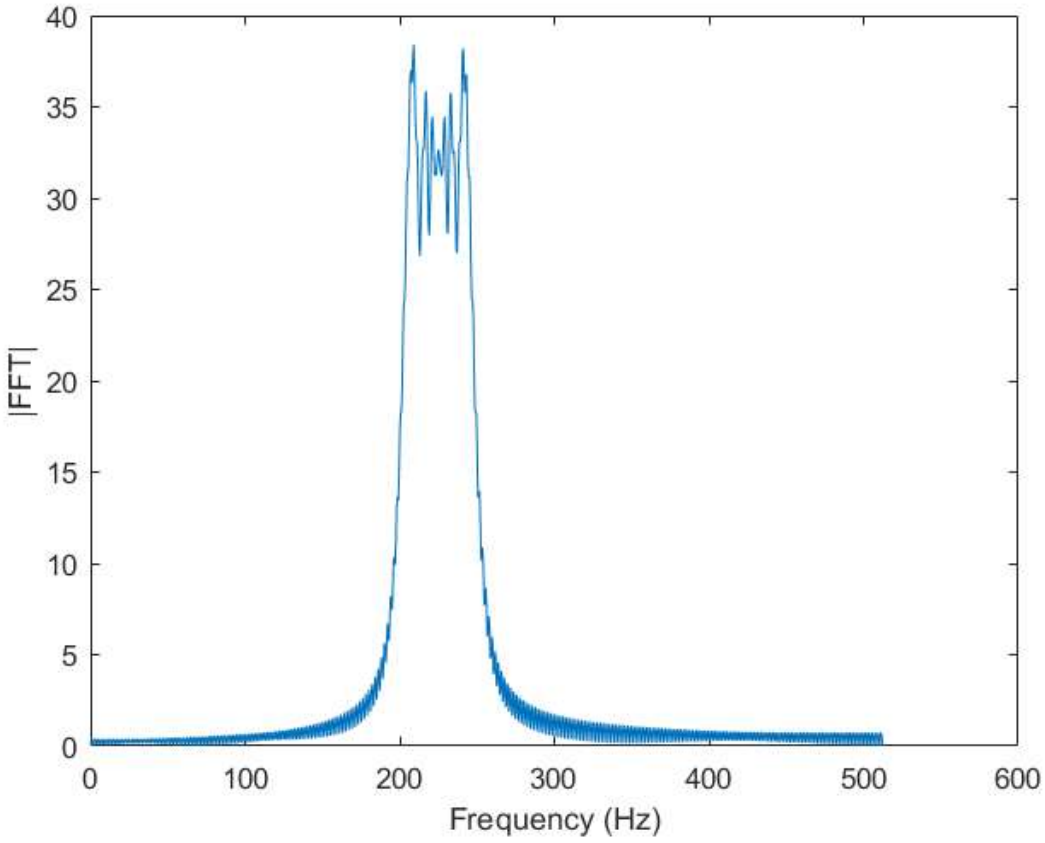
Time domain

```
plot(timeVec,sigVec);
xlabel('Time (sec)');
snapnow;
```



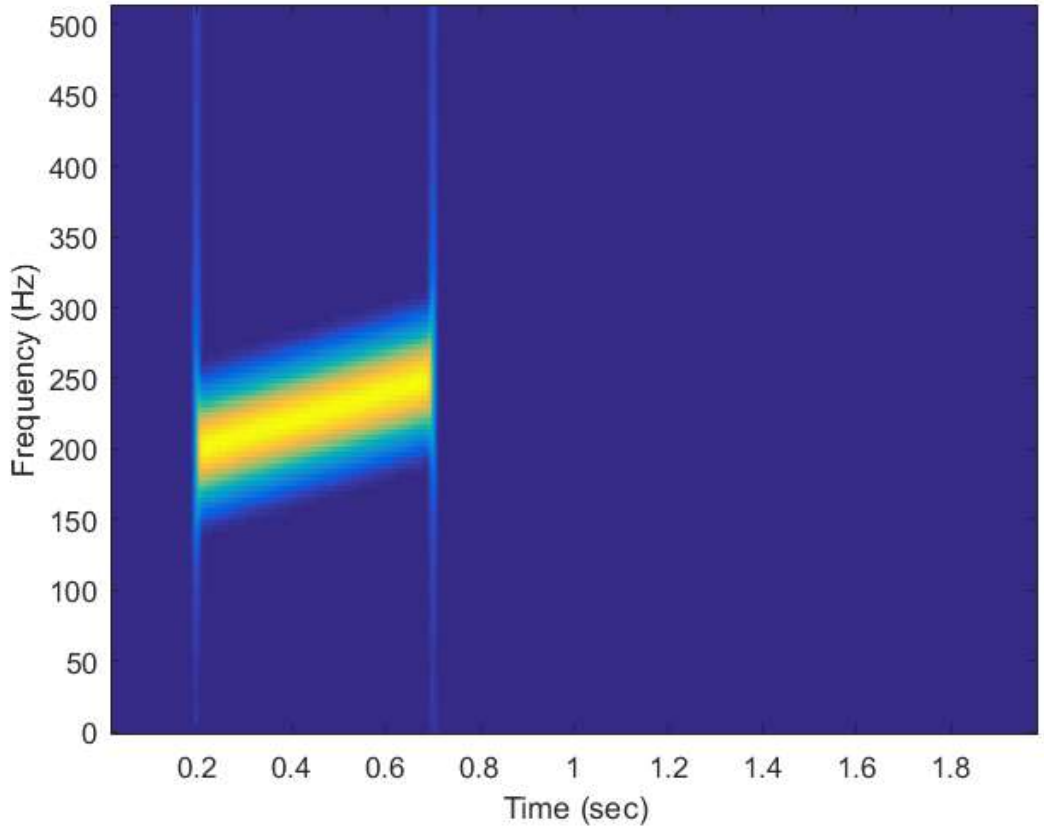

Fourier domain (need to generate the frequency values for the x-axis).

```
nyquistFreqIndx = floor(nSamples/2)+1;
posFreqVec = (0:(nyquistFreqIndx-1))*(1/dataLen);
fftSig = fft(sigVec);
plot(posFreqVec,abs(fftSig(1:nyquistFreqIndx)));
xlabel('Frequency (Hz)'); ylabel('|FFT|');
snapnow;
```




Time-frequency domain (spectrogram)

```
[S,F,T]= spectrogram(sigVec,32,31,[],fs);
imagesc(T,F,abs(S)); axis xy;
xlabel('Time (sec)'); ylabel('Frequency (Hz)');
snapnow;
```




## Generate a data realization

Noise realization

```
noiseVec = randn(1,nSamples);
```

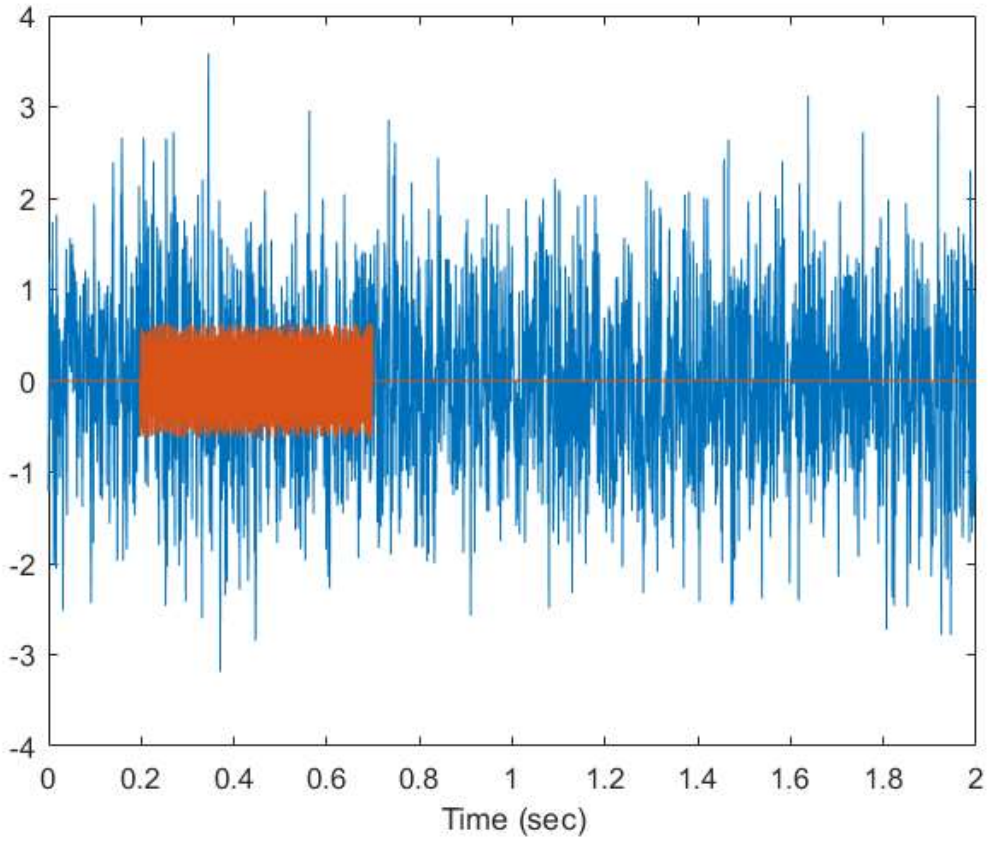
Add signal and noise

```
dataVec = sigVec+noiseVec;
```

## Plot the data in different domains

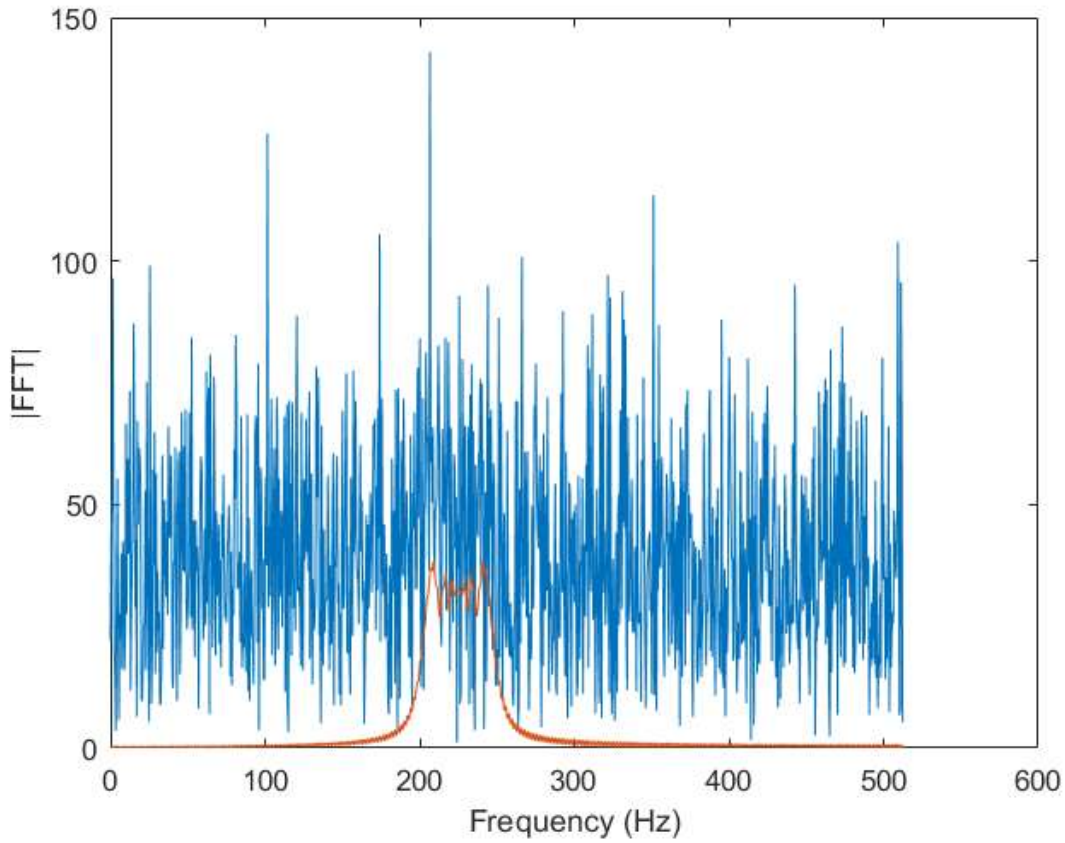
Time domain

```
plot(timeVec,dataVec);
hold on;
plot(timeVec,sigVec);
xlabel('Time (sec)');
hold off;
snapnow;
```



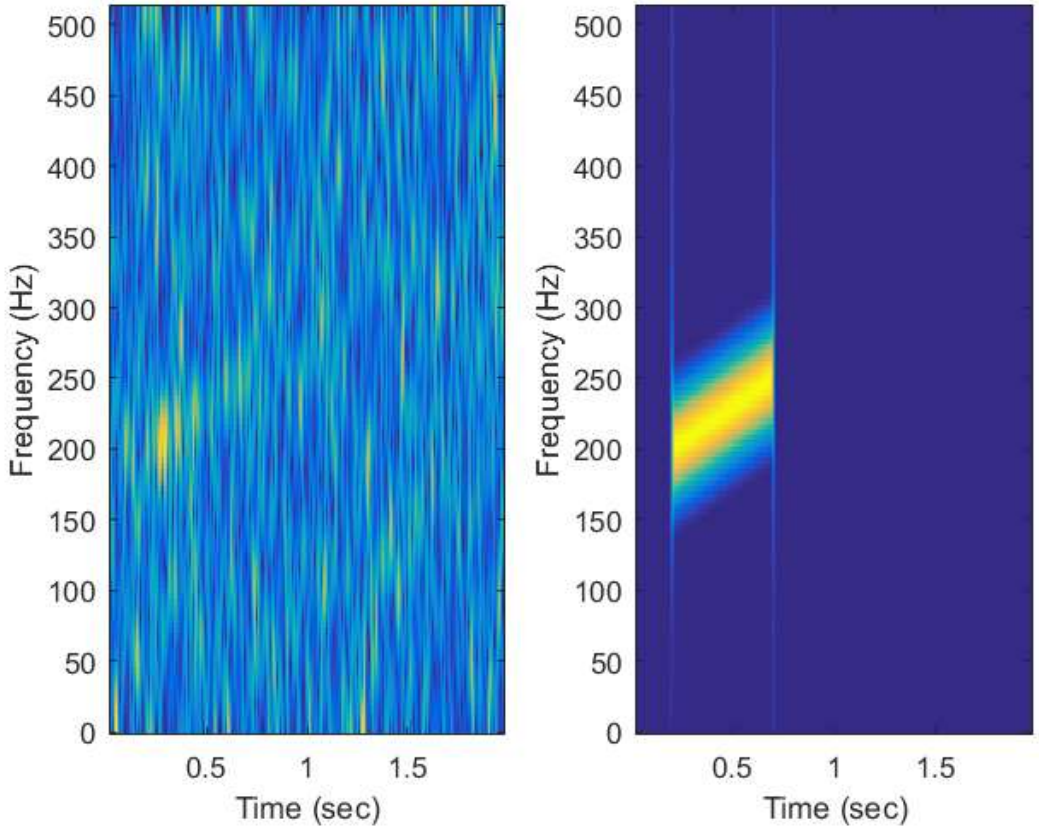

Fourier domain

```
fftData = fft(dataVec);
plot(posFreqVec,abs(fftData(1:nyquistFreqIndx)));
hold on;
plot(posFreqVec,abs(fftSig(1:nyquistFreqIndx)));
xlabel('Frequency (Hz)'); ylabel('|FFT|');
hold off;
snapnow;
```




Time-frequency domain (spectrogram)

```
subplot(1,2,1);
[S,F,T]= spectrogram(dataVec,32,31,[],fs);
imagesc(T,F,abs(S)); axis xy;
xlabel('Time (sec)'); ylabel('Frequency (Hz)');
subplot(1,2,2);
[S,F,T]= spectrogram(sigVec,32,31,[],fs);
imagesc(T,F,abs(S)); axis xy;
xlabel('Time (sec)'); ylabel('Frequency (Hz)');
snapnow;
clf;
```




**Generate quadrature templates**

We will keep the same values of the $\Theta'$ parameters as the signal.

```
sigNSamples = endSample-startSample+1;
templateTimeVec = timeVec(1:sigNSamples);
q0Vec = zeros(1,nSamples);
q0Vec(1:sigNSamples) = sin(2*pi*(f0*(templateTimeVec)+...
                   f1*(templateTimeVec).^2));
q0Vec = q0Vec/norm(q0Vec);
q1Vec = zeros(1,nSamples);
q1Vec(1:sigNSamples) = cos(2*pi*(f0*(templateTimeVec)+...
                   f1*(templateTimeVec).^2));
q1Vec = q1Vec/norm(q1Vec);
```

Fourier transforms of the templates

```
fftq0 = fft(q0Vec);
fftq1 = fft(q1Vec);
```

## Correlate data with each quadrature template

We do the correlation in the Fourier domain.

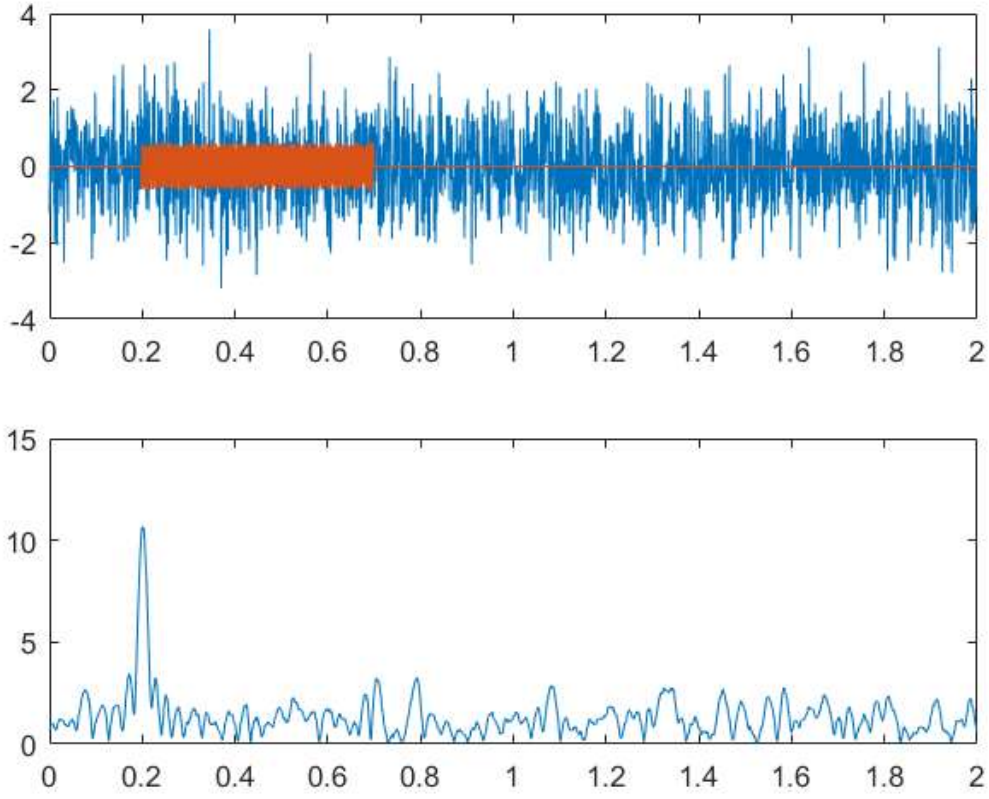
```
q0Corr = real(ifft(fftData.*conj(fftq0)));
q1Corr = real(ifft(fftData.*conj(fftq1)));
```

## Sum of squares of quadrature correlation outputs

```
mfOut = sqrt(q0Corr.^2+q1Corr.^2);
```

## Plot matched filtering output

```
subplot(2,1,1);
plot(timeVec,dataVec);
hold on;
plot(timeVec,sigVec);
subplot(2,1,2);
plot(timeVec,mfOut);
snapnow;
clf;
```


4
2
0
-2
-4
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2


15
10
5
0
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2

*Published with MATLAB® R2016b*