DSP Project

Audio Equalizer

Name	ID
Sherif Rafik	4635
Mahmoud Salem	4643

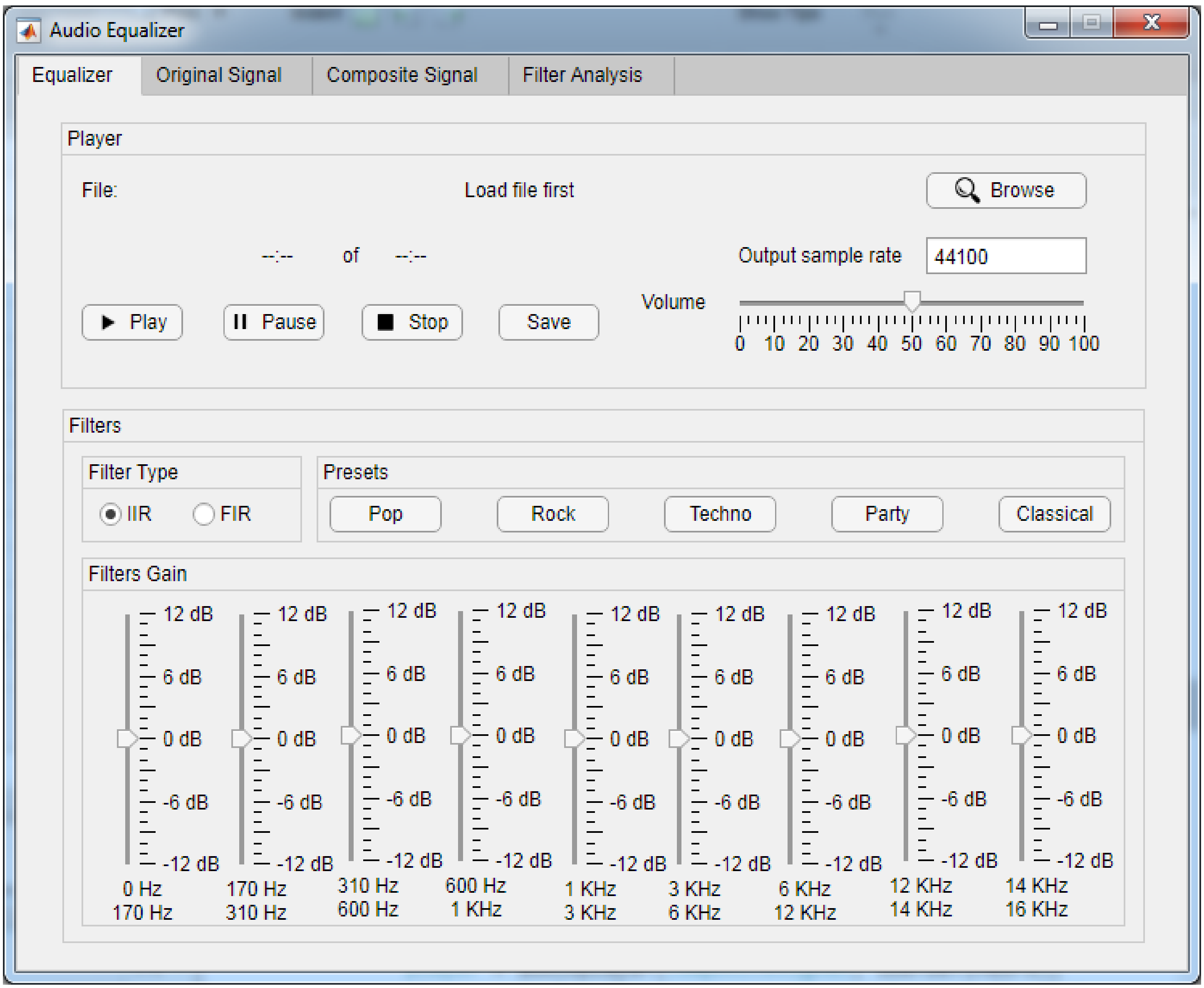
Introduction:

Problem Statement:

It is required to develop audio equalizer using MATLAB. The equalizer function is to vary the gain of each specific band as the user prefers. [i.e. if the user likes base, he will increase the gain of low frequencies].

Inputs to the program:

1. Wave file name.
2. Gain of each of the frequency bands in db.
3. Type of filter used (FIR – IIR)
4. Output Sample Rate



Outputs of the program:

1. Composite signal is played and can be saved
2. Filter analysis results
3. Figures of signals in time and frequency domains

Extras:

- There are 5 built in presets which the user can use on his wav file:
  1. Pop
  2. Rock
  3. Techno
  4. Party
  5. Classical
- User can pause and stop the sound file.

Instructions:

1. Load up an audio file
2. Input the output sample rate
3. Choose the type of the filter (FIR / IIR)
4. Adjust the sliders according to the gain of each filter
5. Play the output sound
6. Save the output sound
7. In filter analysis tab, check the analysis results of the filters

Global variables used:

```
global iirFilter; %boolean: true if iirfilter is used
global firFilter; %boolean: true if firfilter is used
global frequencies; %frequencies array
global numerator;
global denominator;
global slidersValues;
global firOrder;
global iirOrder;
frequencies =[0,170,310,600,1000,3000,6000,12000,14000,16000];
firFilter = false;
iirFilter = true;
numerator = cell(9,1);
denominator = cell(9,1);
slidersValues = [];
firOrder = 30;
iirOrder = 2;
```

Functions used:

1. **uigetfile**: Open file selection dialog box, used for inputting the wav file to the equalizer

```
[file,path] = uigetfile({'*.wav'},'Selec a wav file');
global file_path;
if ~isequal(file,0)
    file_path = fullfile(path,file);
    app.directoryLabel.Text = file;
end
```

2. **resample**: Resample uniform or nonuniform data to new fixed rate, used for resampling the input wav file to the new sampling rate inputted by the user
3. **audioread**: Reads data from an audio file and returns sampled data and sample rate

```
[y, Fs] = audioread(file_path);
userDefinedFs = app.OutputsamplerateEditField.Value;
y = resample(y,userDefinedFs,Fs);
```

4. **butter**: Butterworth filter design, used for designing the IIR filters, the function returns the numerator and denominator of the transfer function of the filter

```
[numerator{1}, denominator{1}] =
butter(iirOrder,frequencies(2)/(Fs/2));
for i = 2 : 9
      [numerator{i},denominator{i}] = butter(iirOrder,[frequencies(i)
      frequencies(i+1)]/(Fs/2));
end
```

5. **fir1**: Window based FIR filter design, used for designing the FIR filters, the function returns the numerator of the transfer function of the filter

```
numerator{1} = fir1(firOrder , frequencies(2)/(Fs/2));
for i = 2 : 9
    numerator{i} = fir1(firOrder, [frequencies(i)
    frequencies(i+1)]/(Fs/2));
end
```

6. **filter**: It filters the input data using a rational transfer function defined by the numerator and the denominator, used in filtering the input wav file

```
if (iirFilter == true)
    for i = 1 : 9
        filteredSound{i} = filter(numerator{i} , denominator{i}, y);
    end
else
   for i = 1 : 9
        filteredSound{i} = filter(numerator{i} , 1, y);
   end
end
```

7. **impz:** Impulse response of a digital filter, the function chooses the number of samples and plots the impulse response, used to calculate the impulse response of the filters whether FIR or IIR

```
if iirFilter == true
    impz(numerator{i},denominator{i});
else
    impz(numerator{i},1);
end
```

8. **stepz:** Step response of a digital filter, the function chooses the number of samples and plots the step response, used to calculate the step response of the filters whether FIR or IIR

```
if iirFilter == true
    stepz(numerator{i},denominator{i});
else
    stepz(numerator{i},1);
end
```

9. **freqz:** Frequency response of a digital filter, used to calculate the phase and gain of each filter whether its FIR or IIR

```
if iirFilter == true
    stepz(numerator{i},denominator{i});
else
    stepz(numerator{i},1);
end
```

10. **tf:** It returns a transfer function model

```
transferFunction = tf(numerator{i},denominator{i});
```

11. **pzmap:** Pole-zero plot of a dynamic system, used to plot the poles and zeros of each filter whether FIR or IIR

```
if iirFilter == true
    transferFunction = tf(numerator{i},denominator{i});
    pzmap(transferFunction);
else
    transferFunction = tf(numerator{i},1);
    pzmap(transferFunction);
end
```

12. **fft:** Computes the discrete Fourier transform (DFR) using the fast Fourier transform algorithm
13. **fftshift:** It rearranges the flourier transform by shifting the zero-frequency component to the center of the array

```
plot(app.UIAxes_2,frequency,abs(fftshift(fft(compositeSignal))));
plot(app.UIAxes_4,frequency,abs(fftshift(fft(y))));
```

14. **audioplayer:** Use an audioplayer object to play audio, the object contains properties that enable additional flexibility during playback. For example, you can pause or resume.

```
player = audioplayer(compositeSignal, userDefinedFs);
```

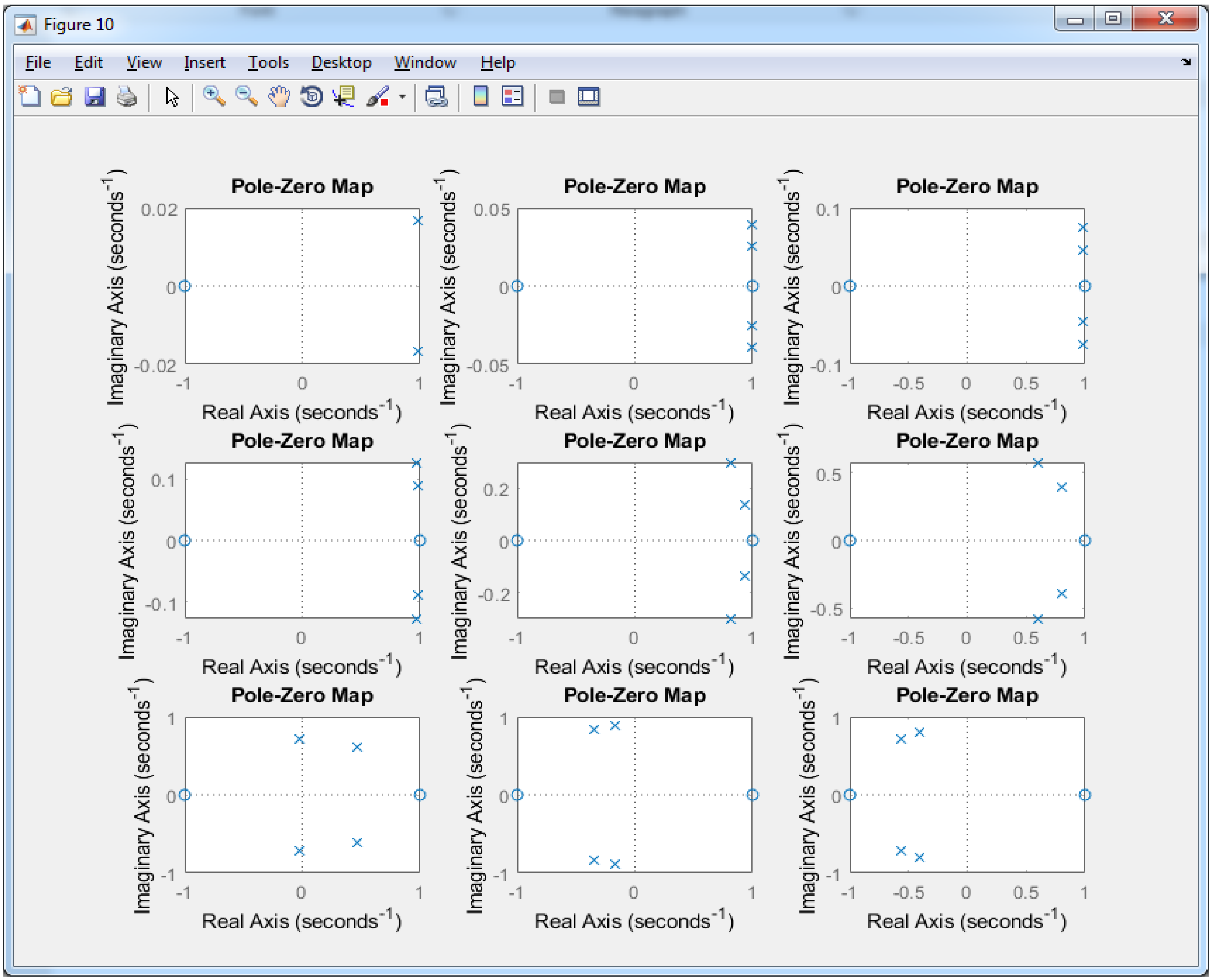
15. **audiowrite:** Writes a matrix of audio data, with a specific sample rate.

```
global compositeSignal;
userDefinedFs = app.OutputsamplerateEditField.Value;
audiowrite('output.wav',compositeSignal,userDefinedFs);
```

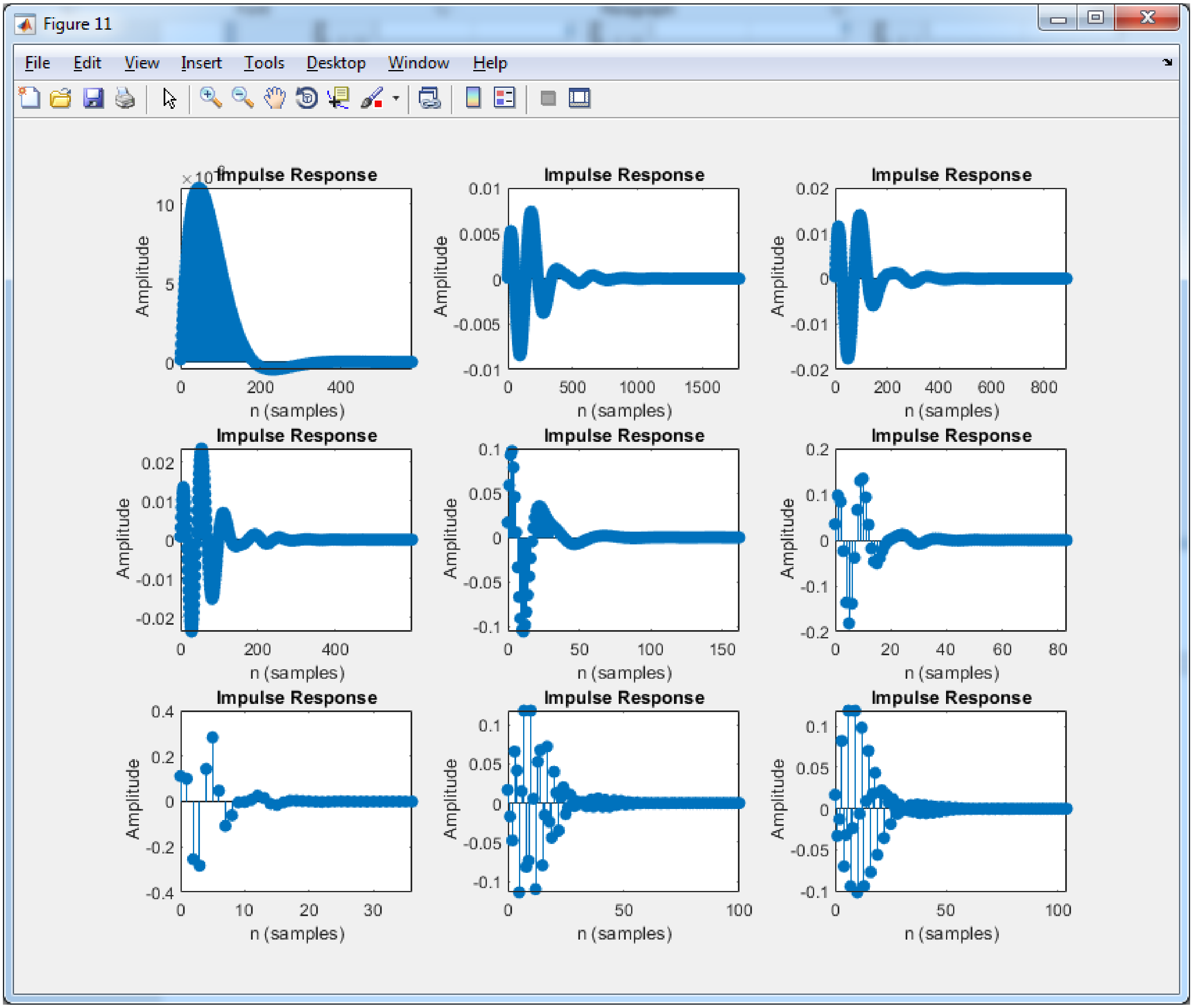
Filter analysis Results:

IIR filter:

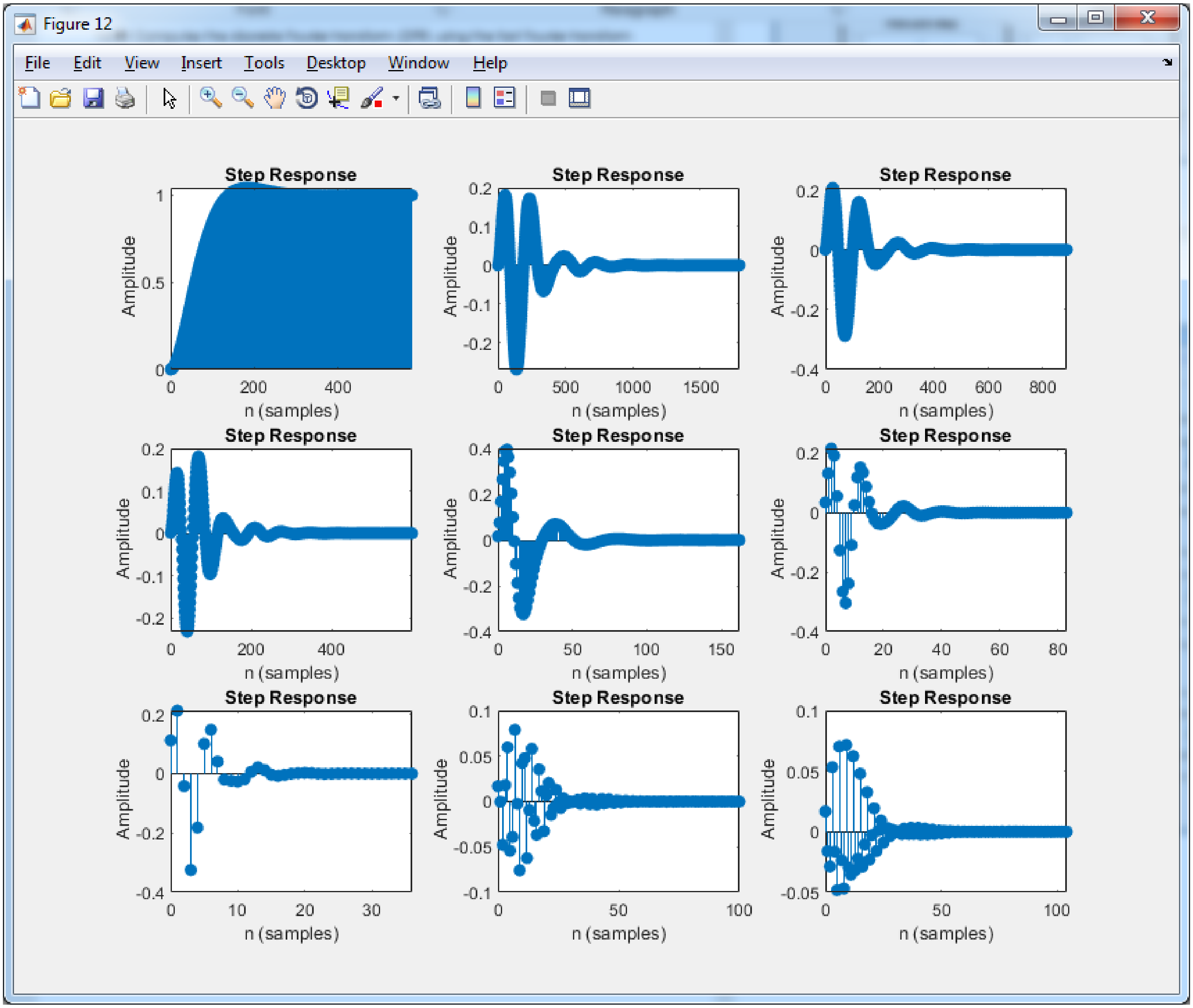
1. **Poles and Zeros of each filter:**



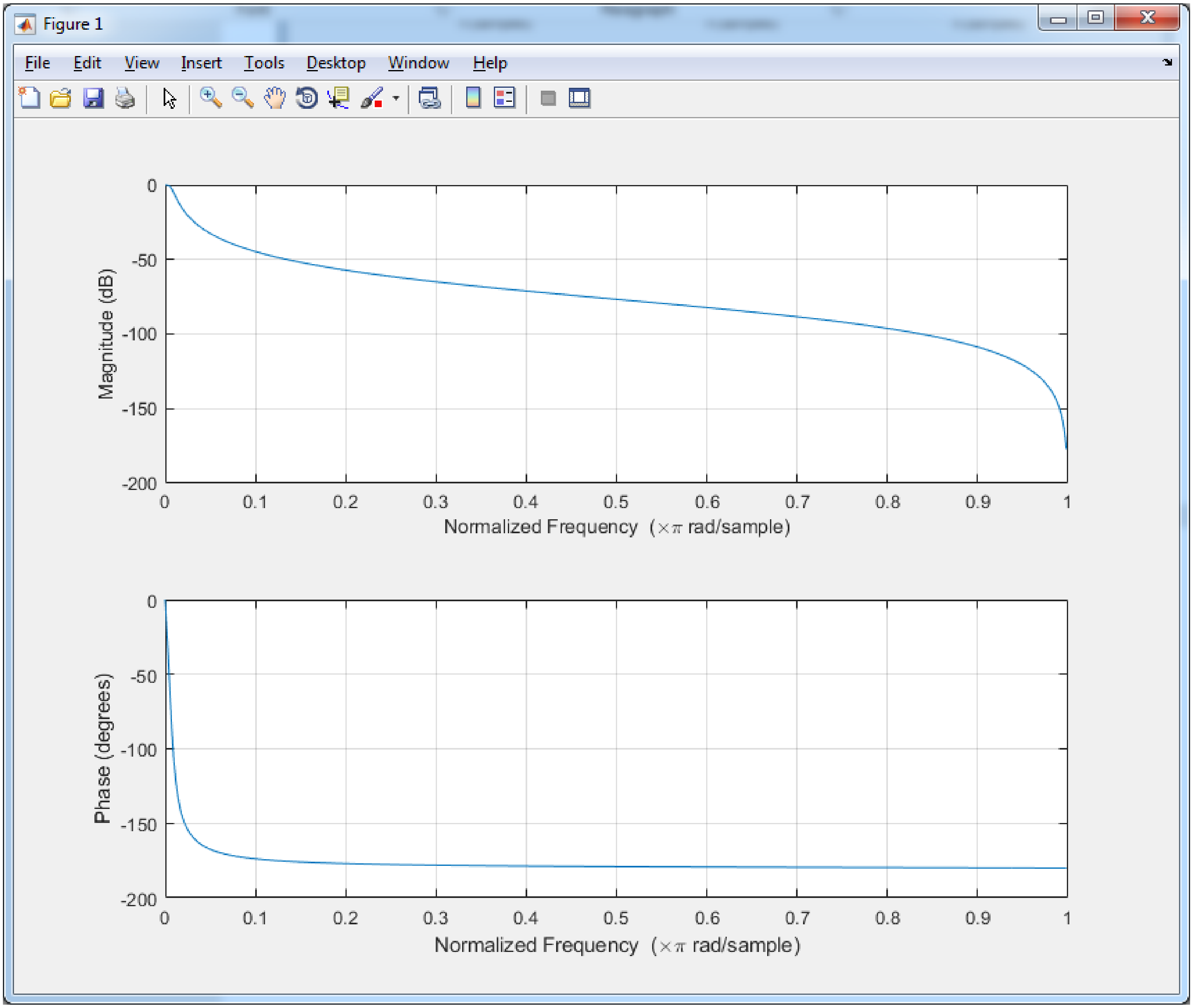
2. Impulse response of each filter:



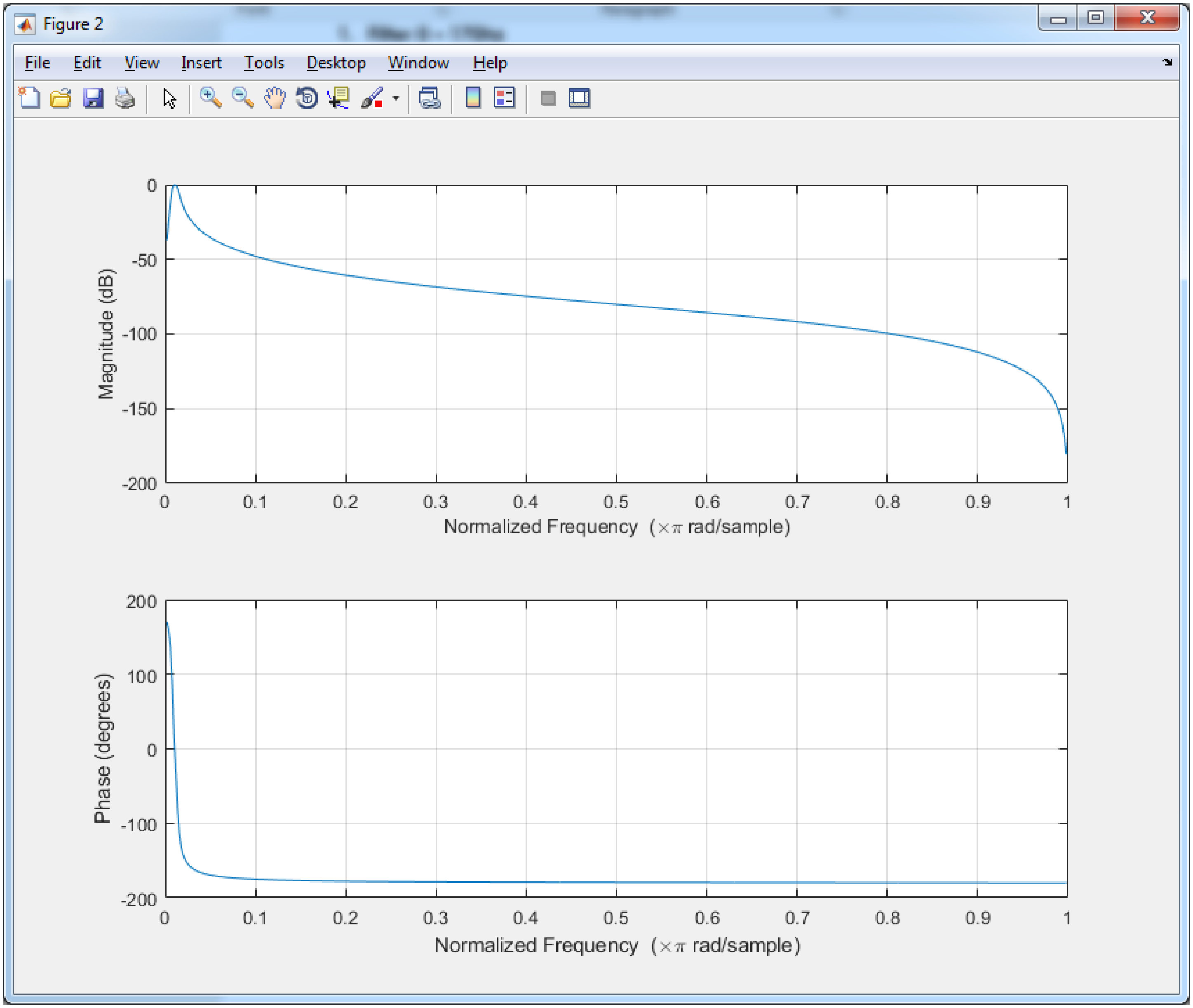
3. Step response of each filter:



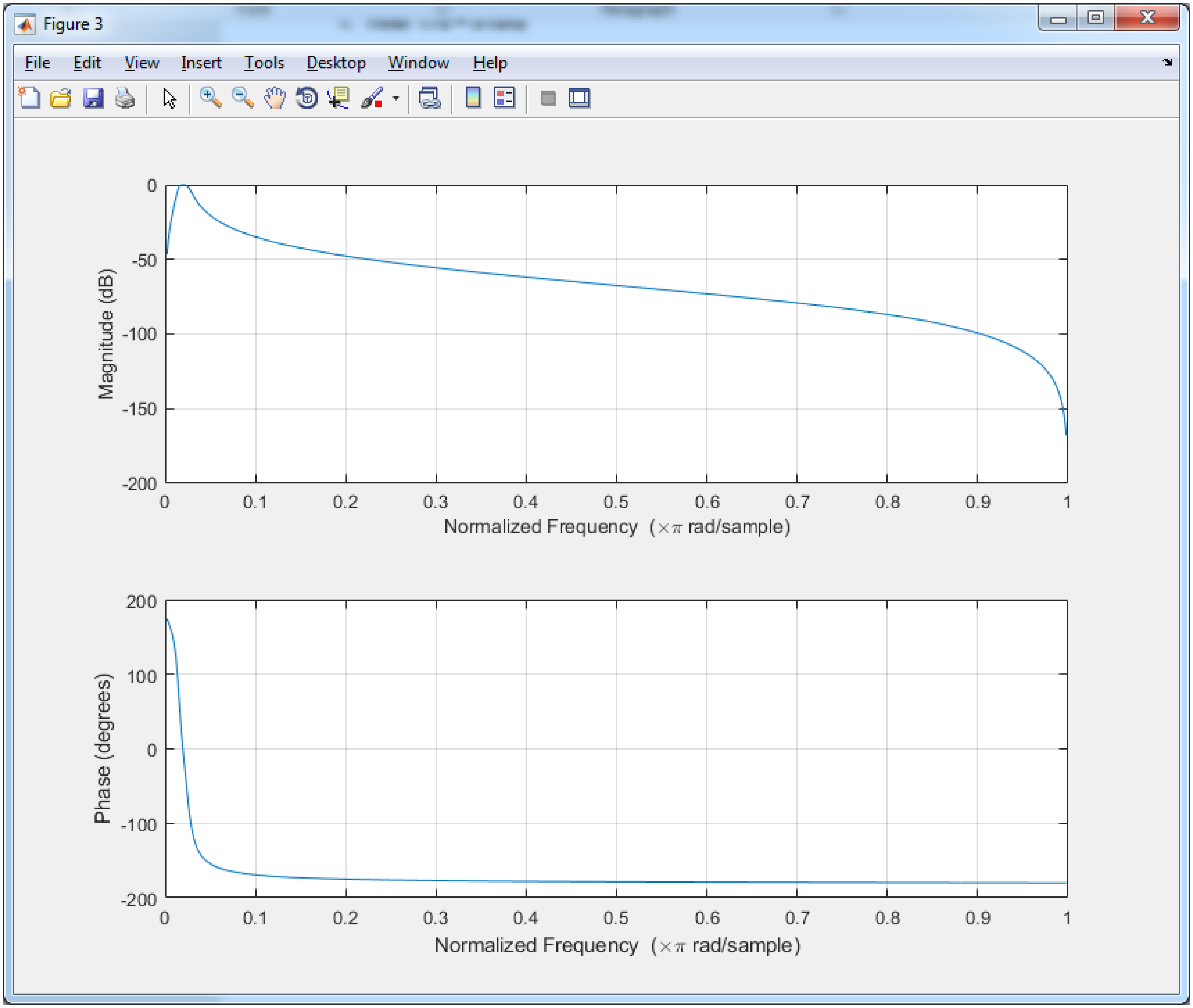
4. **Phase and gain:**
   1. **Filter 0 – 170Hz**



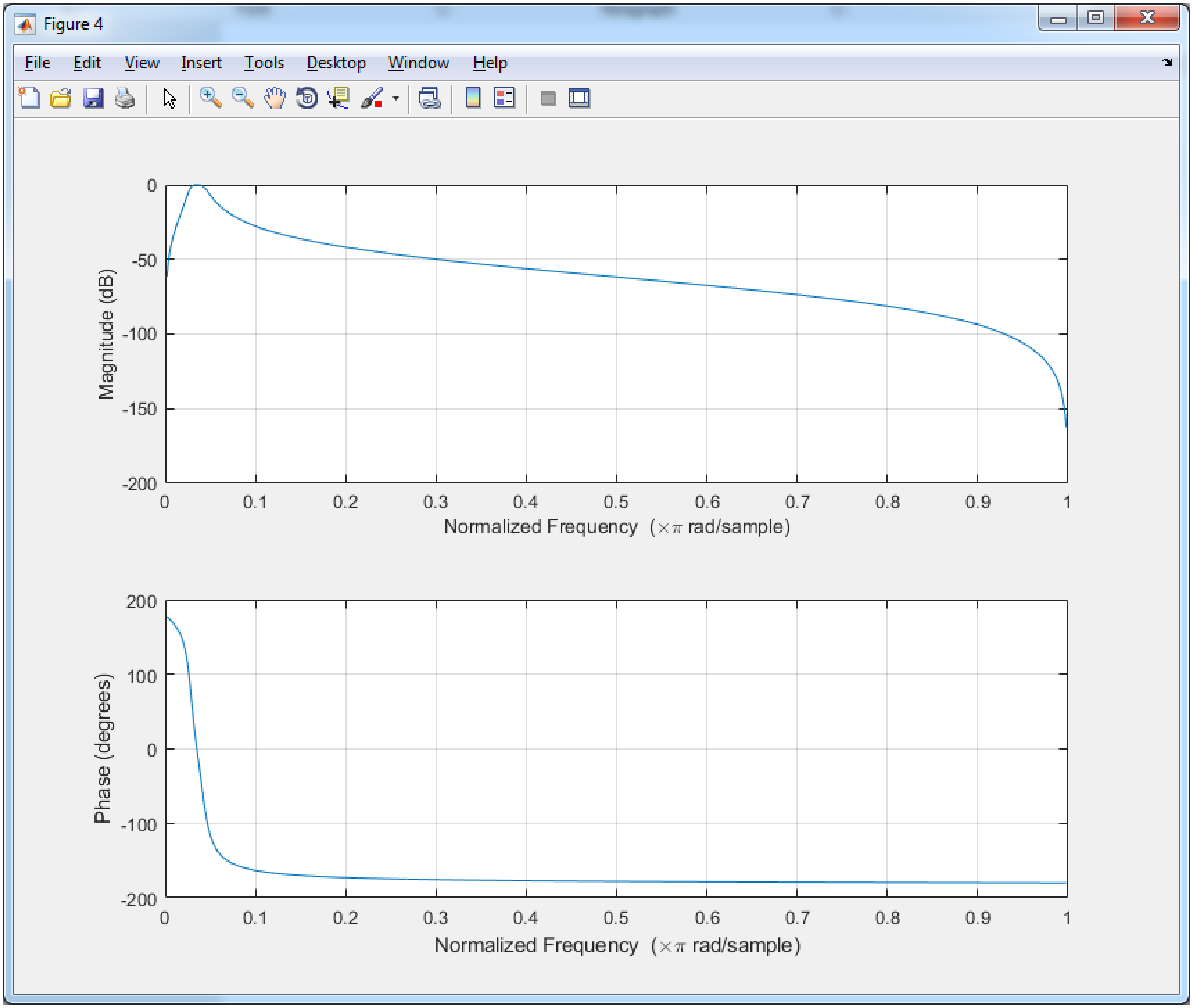
2. Filter 170 – 310Hz



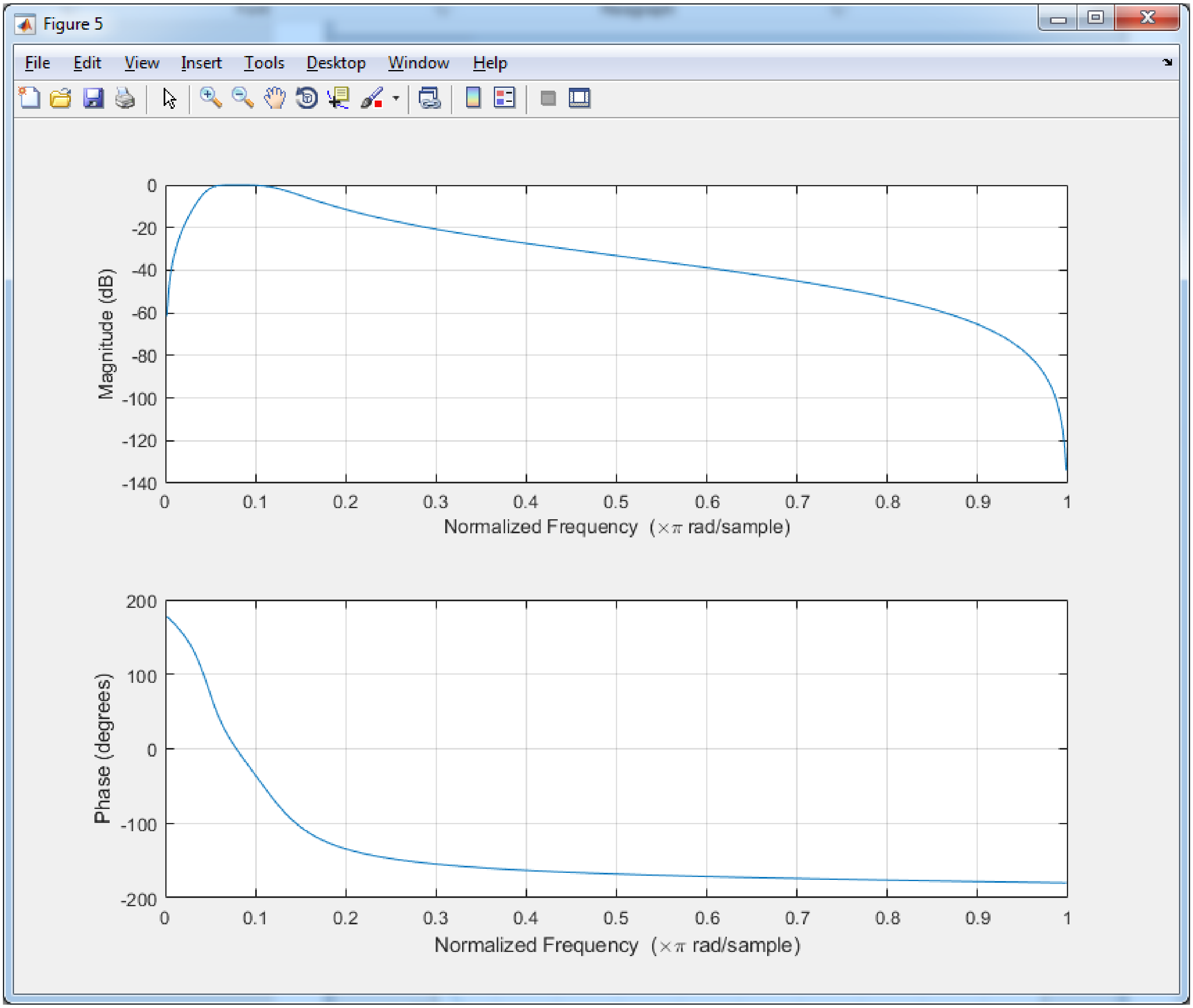
3. Filter 310 – 600Hz



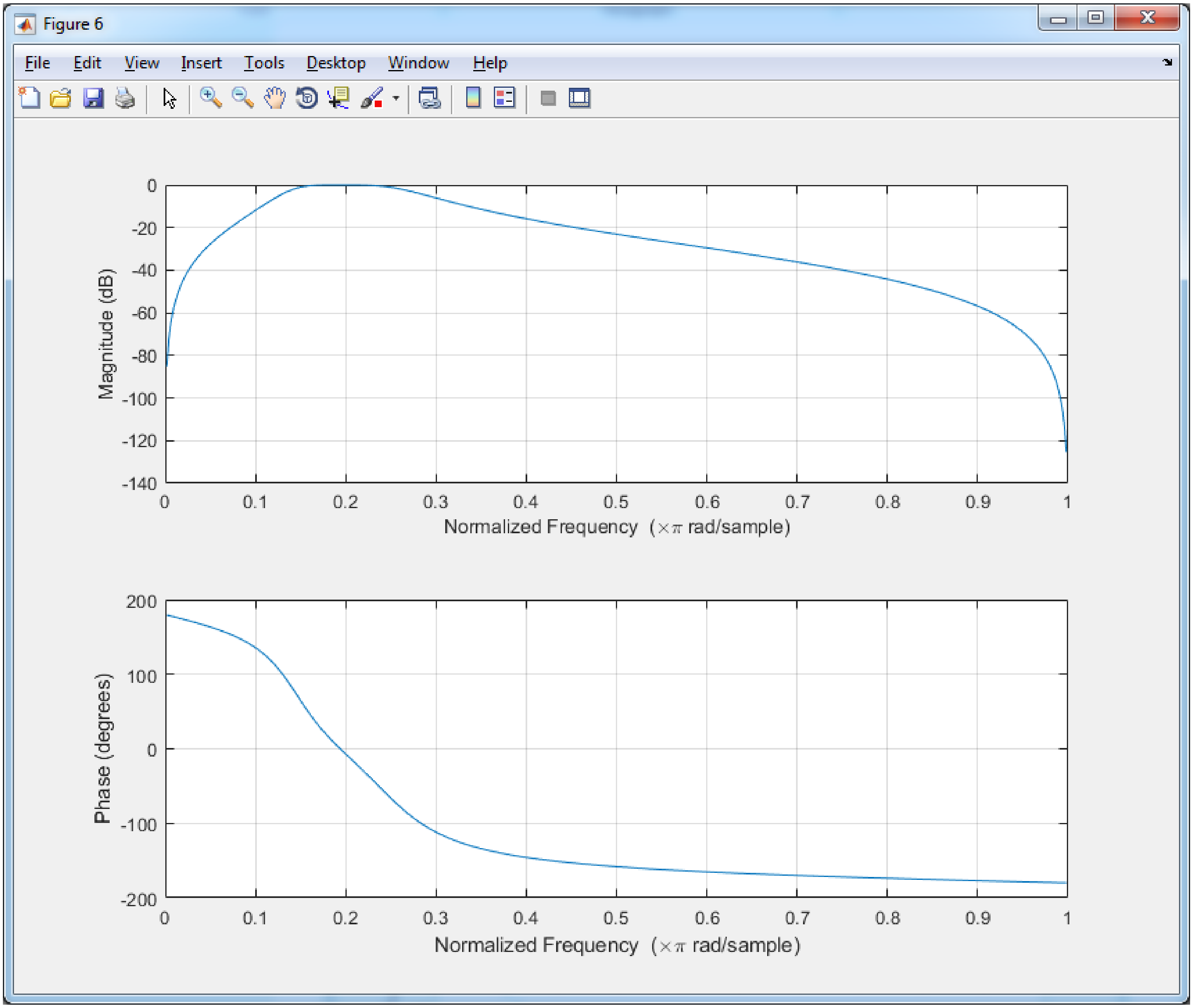
4. Filter 600 – 1000Hz



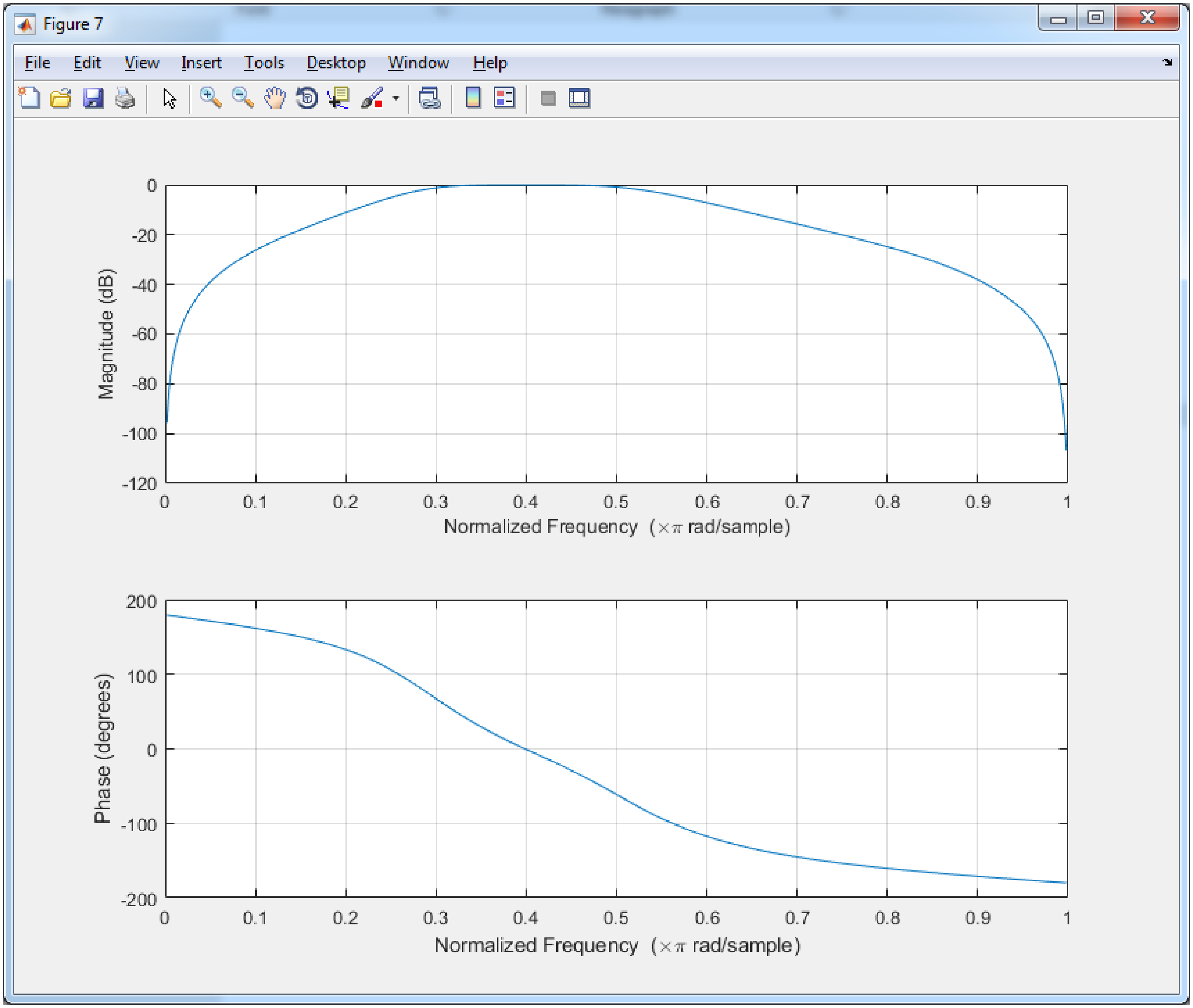
5. Filter 1 – 3KHz



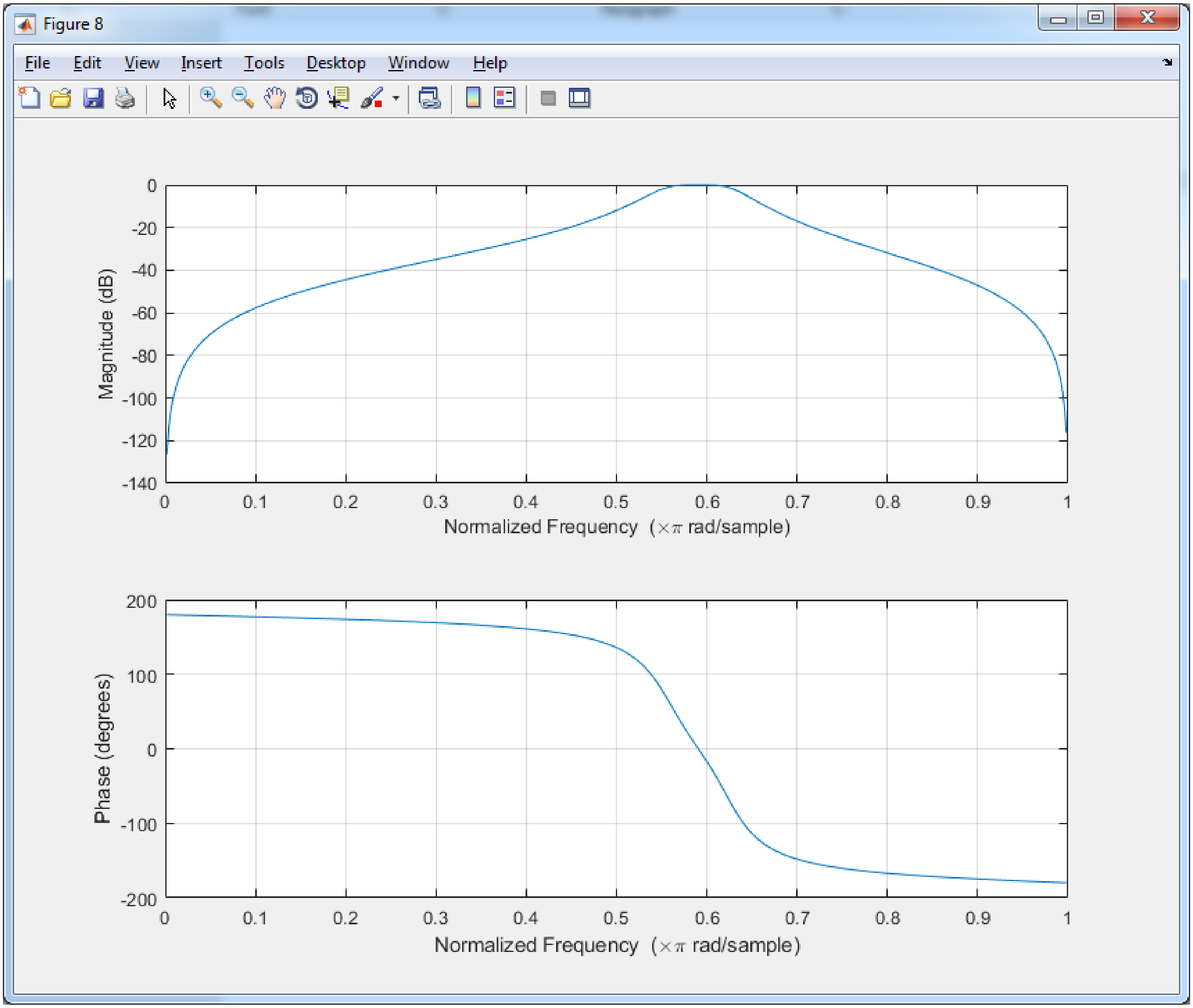
6. Filter 3 – 6KHz



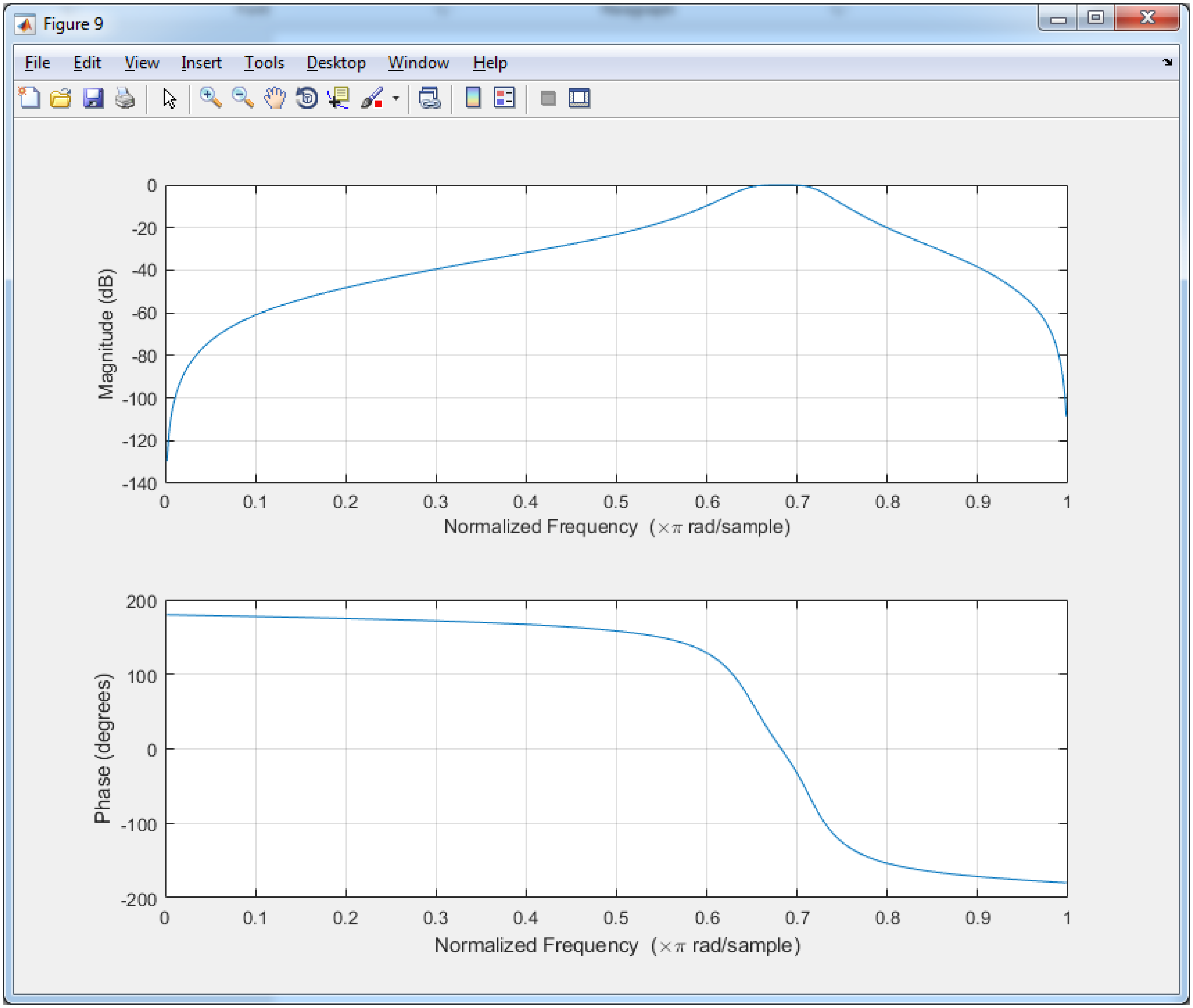
7. Filter 6 – 12KHz



8. Filter 12 – 14KHz

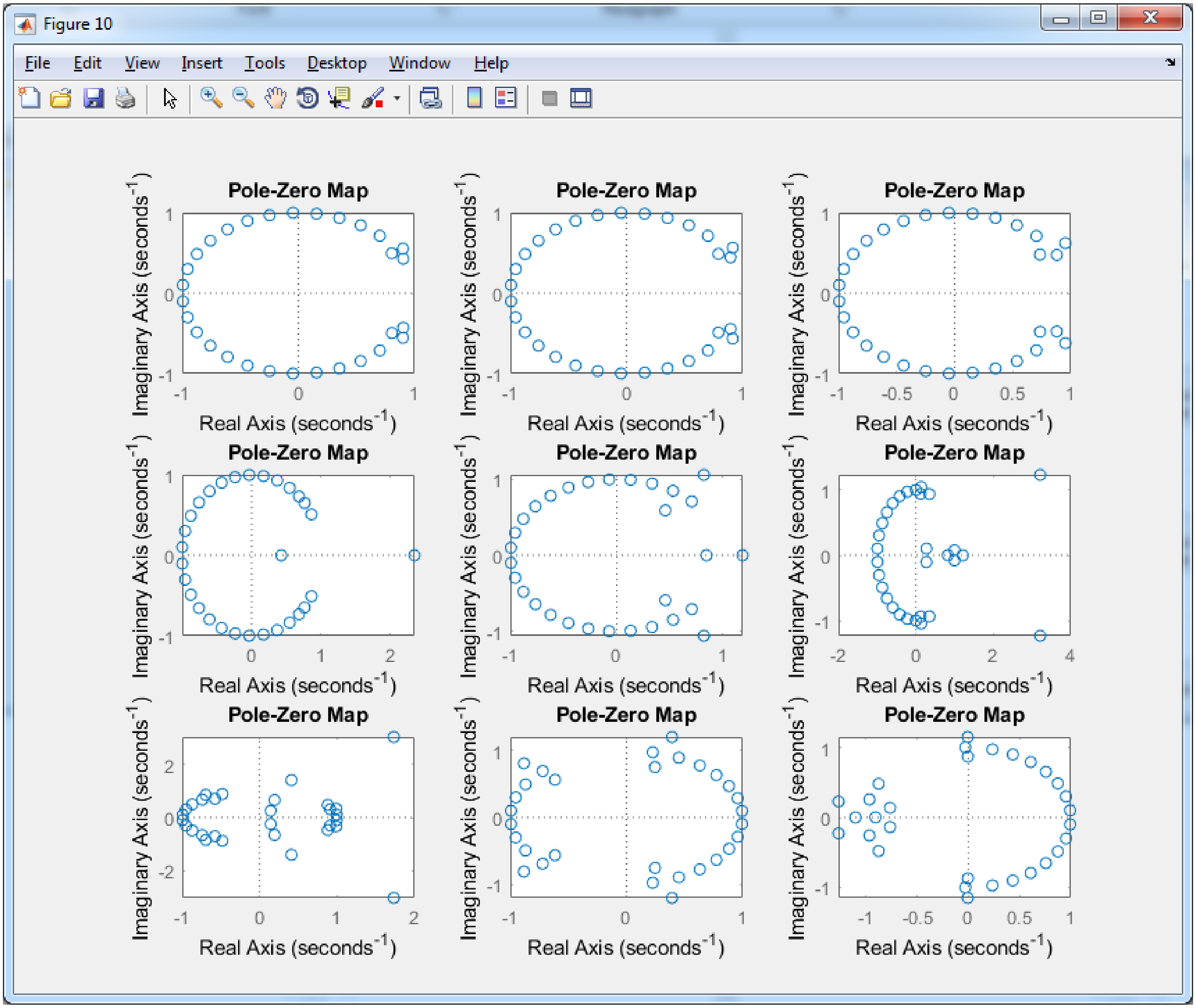


9. Filter 14 – 16KHz

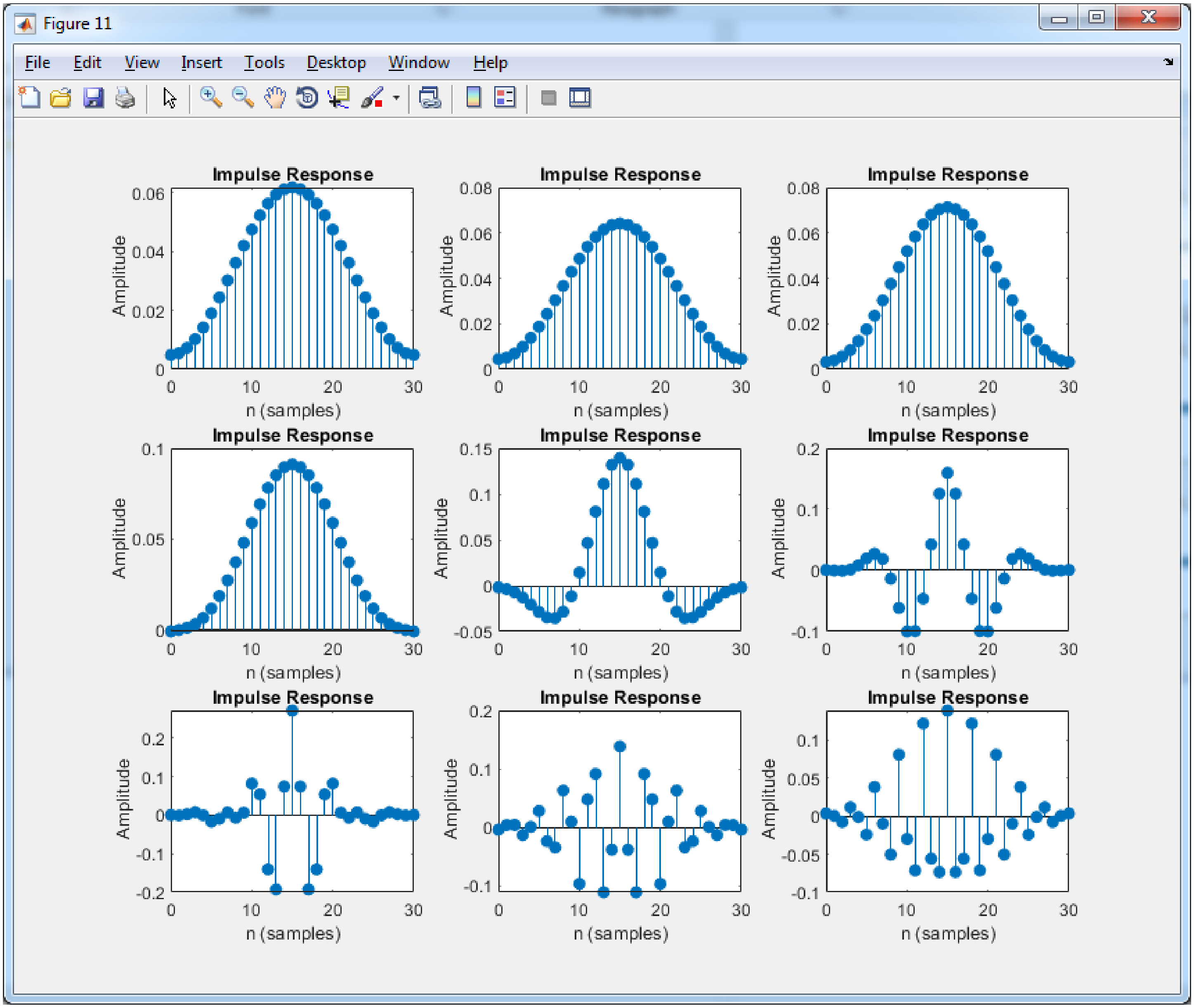


FIR filter:

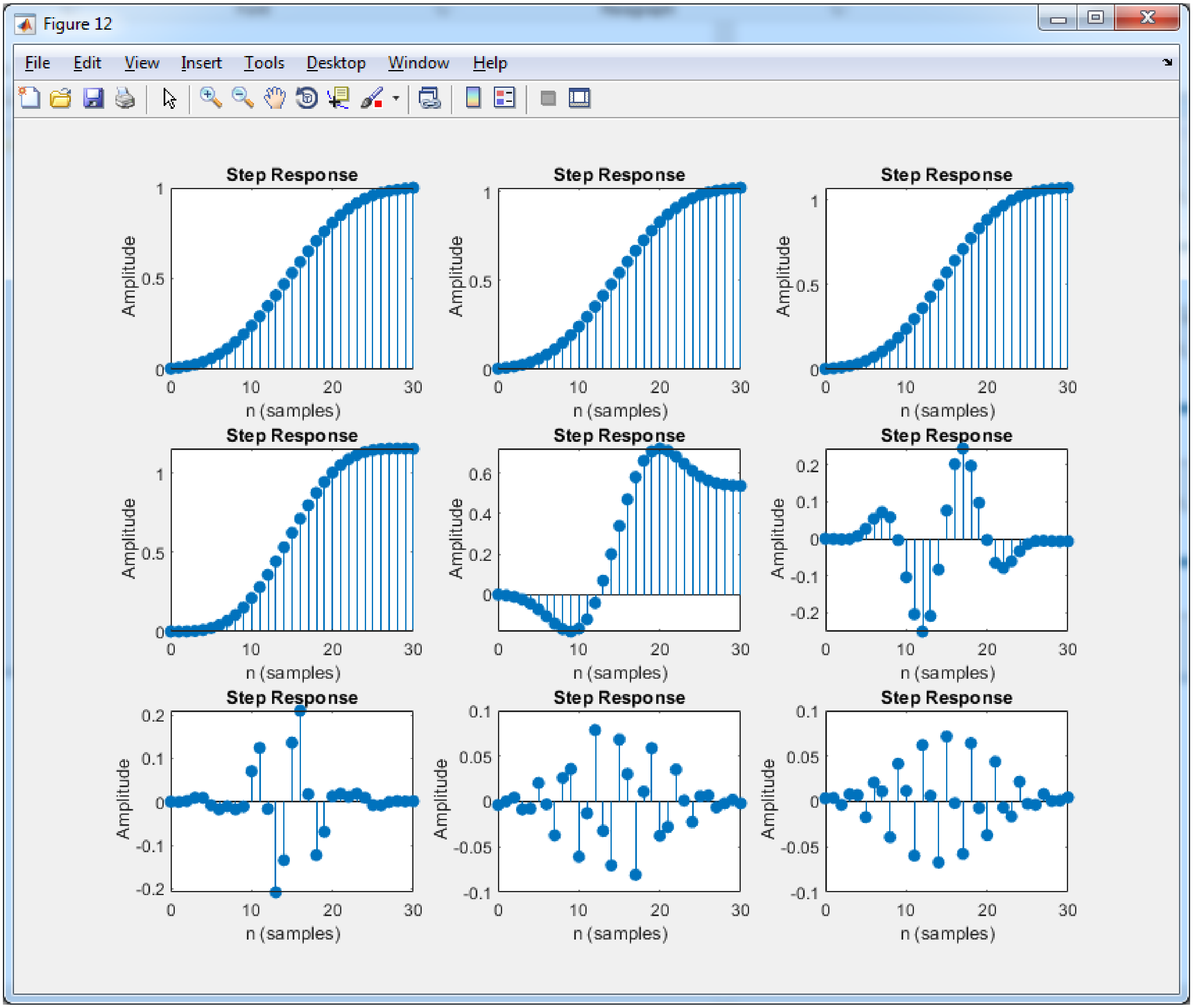
1. Poles and Zeros of each filter:



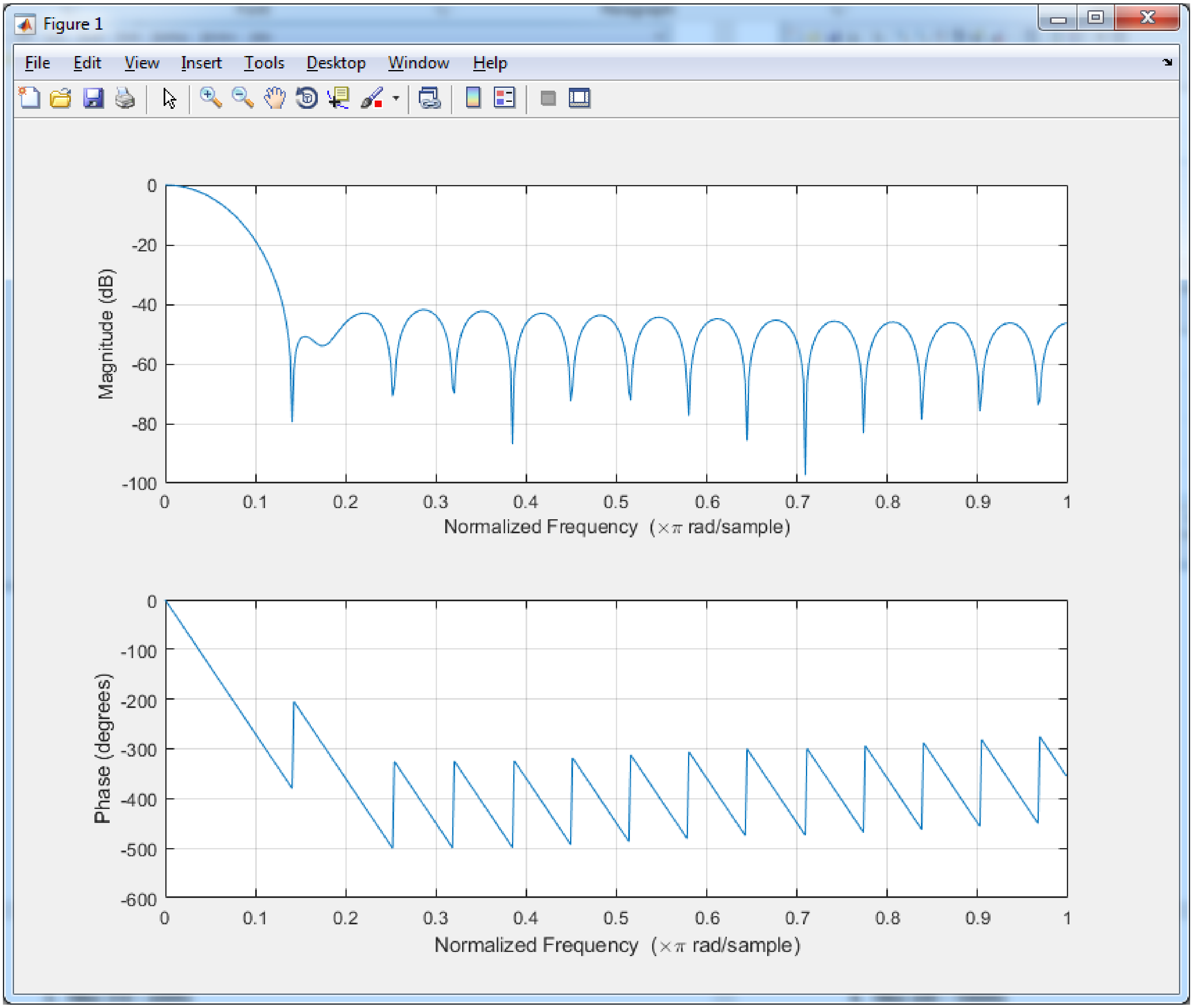
2. Impulse response of each filter:



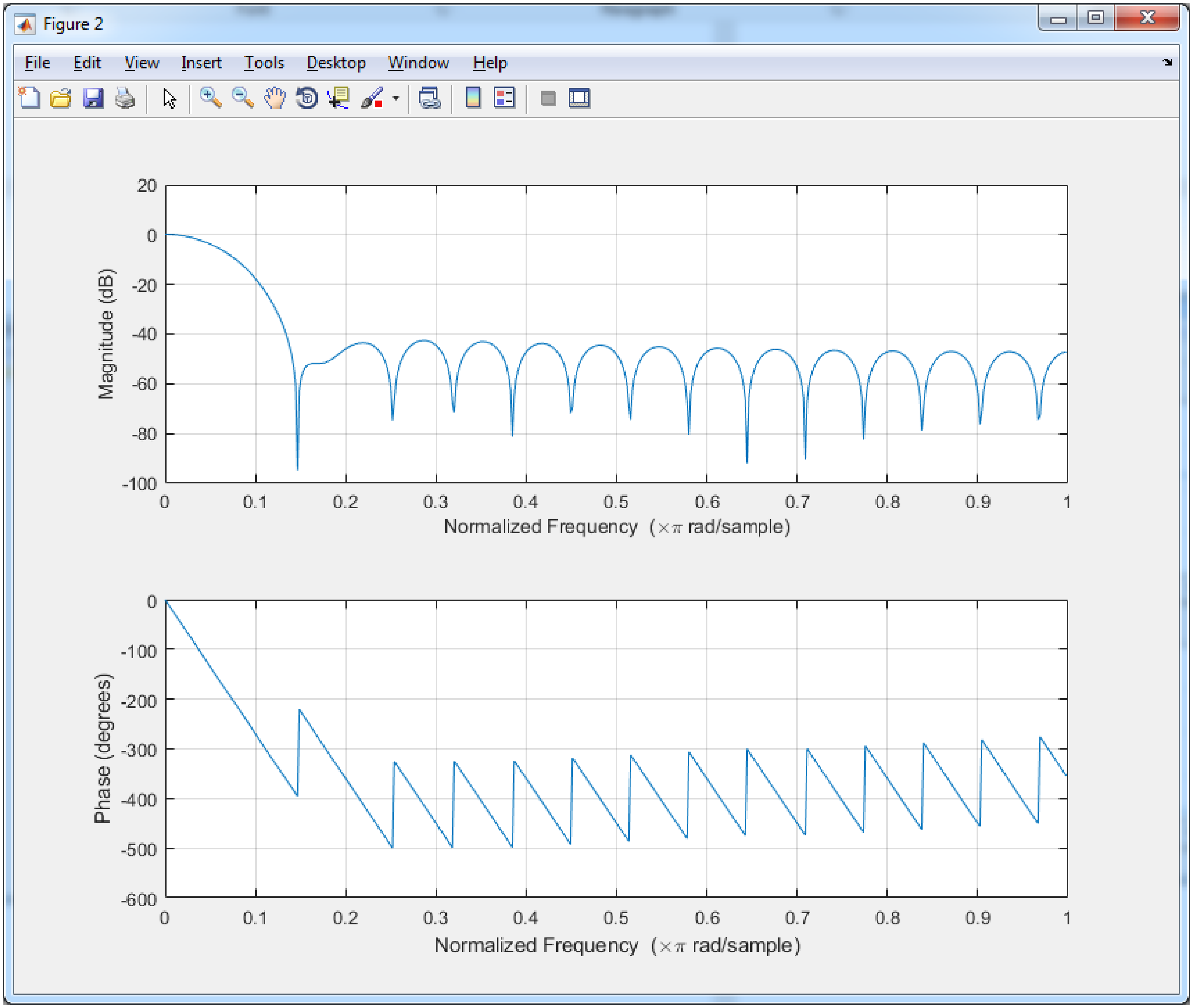
3. Step response of each filter:



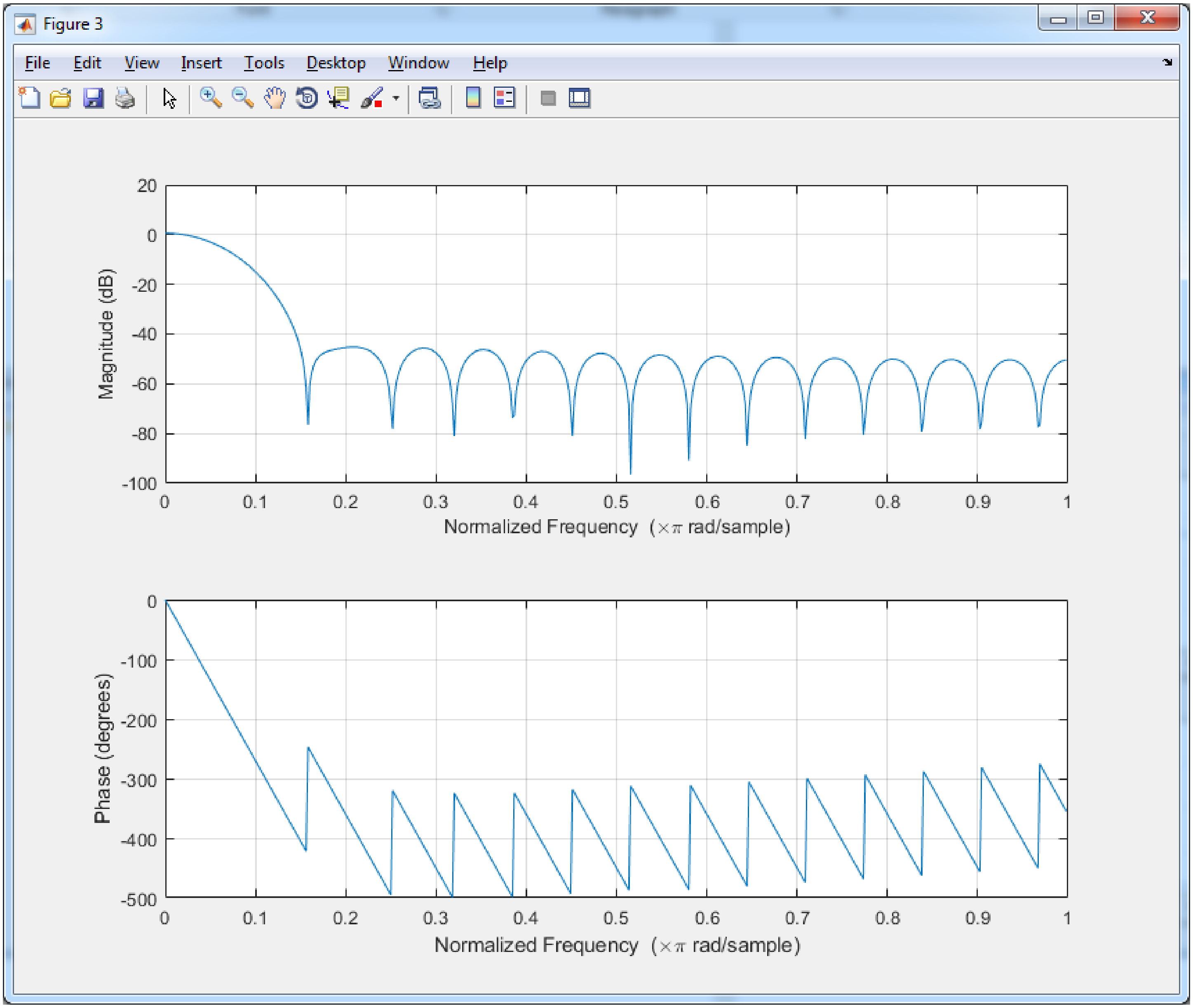
4. **Phase and gain:**
    1. **Filter 0 – 170Hz**



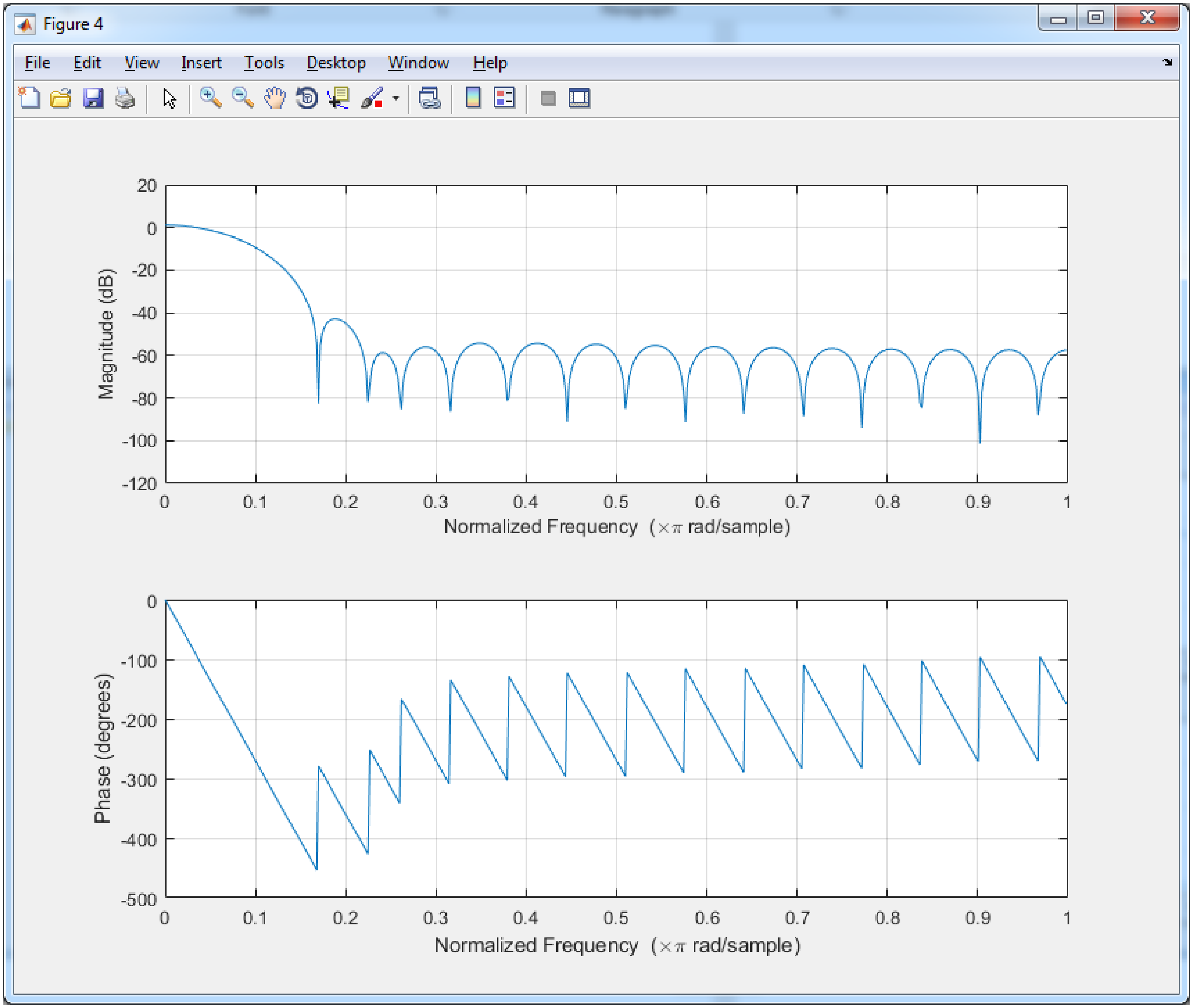
2. Filter 170 – 310Hz



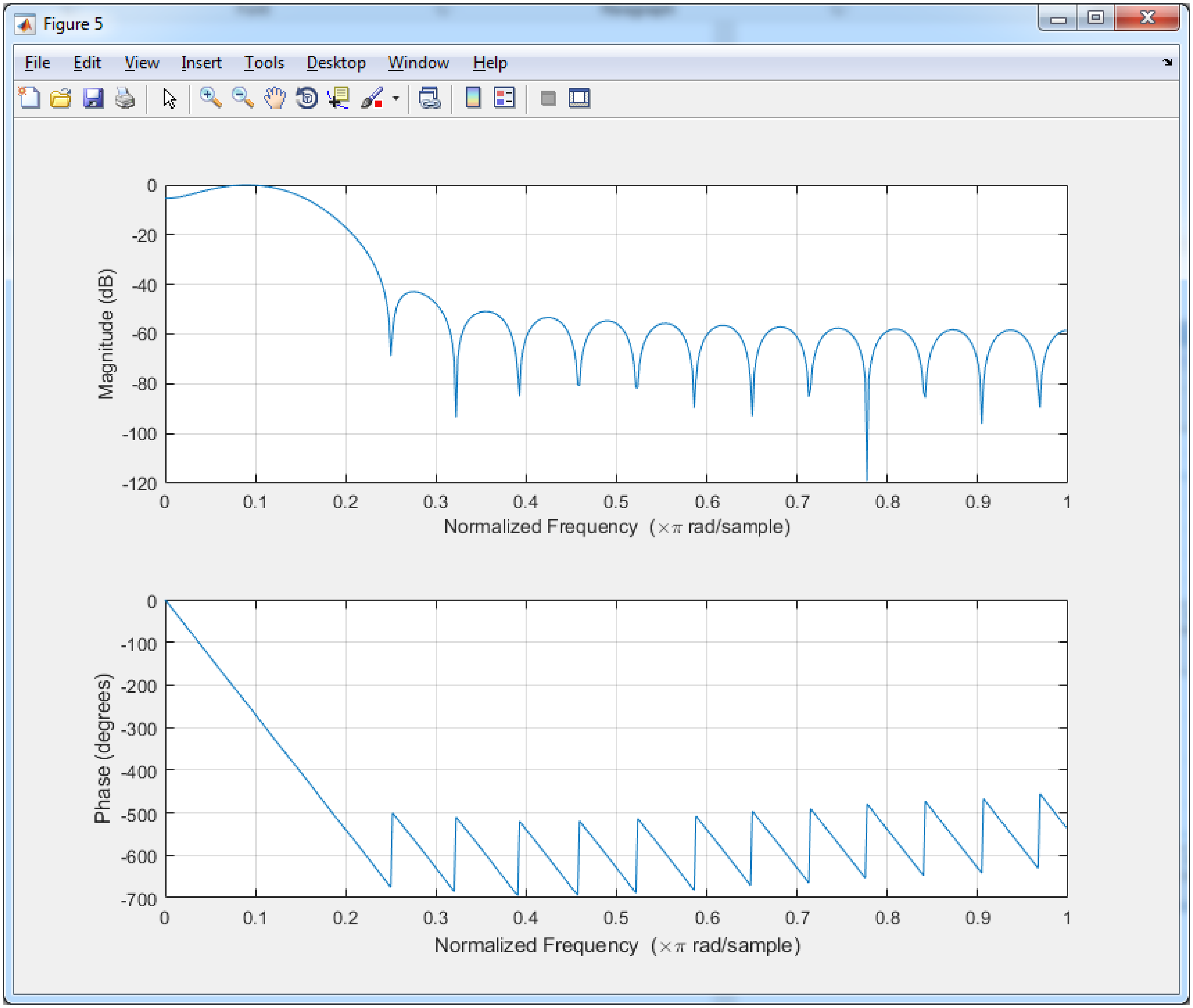
3. Filter 310 – 600Hz



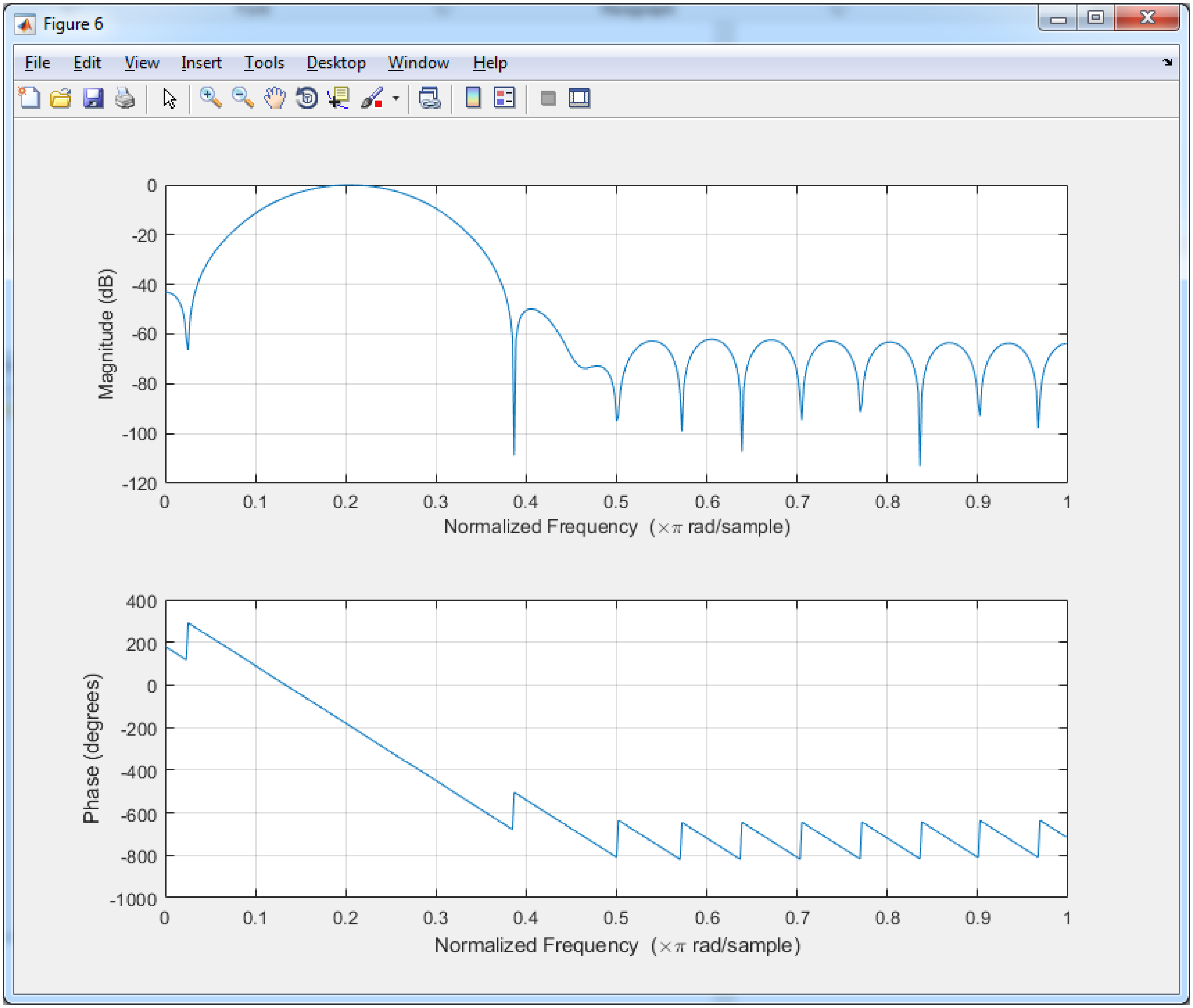
4. Filter 600 – 1000Hz



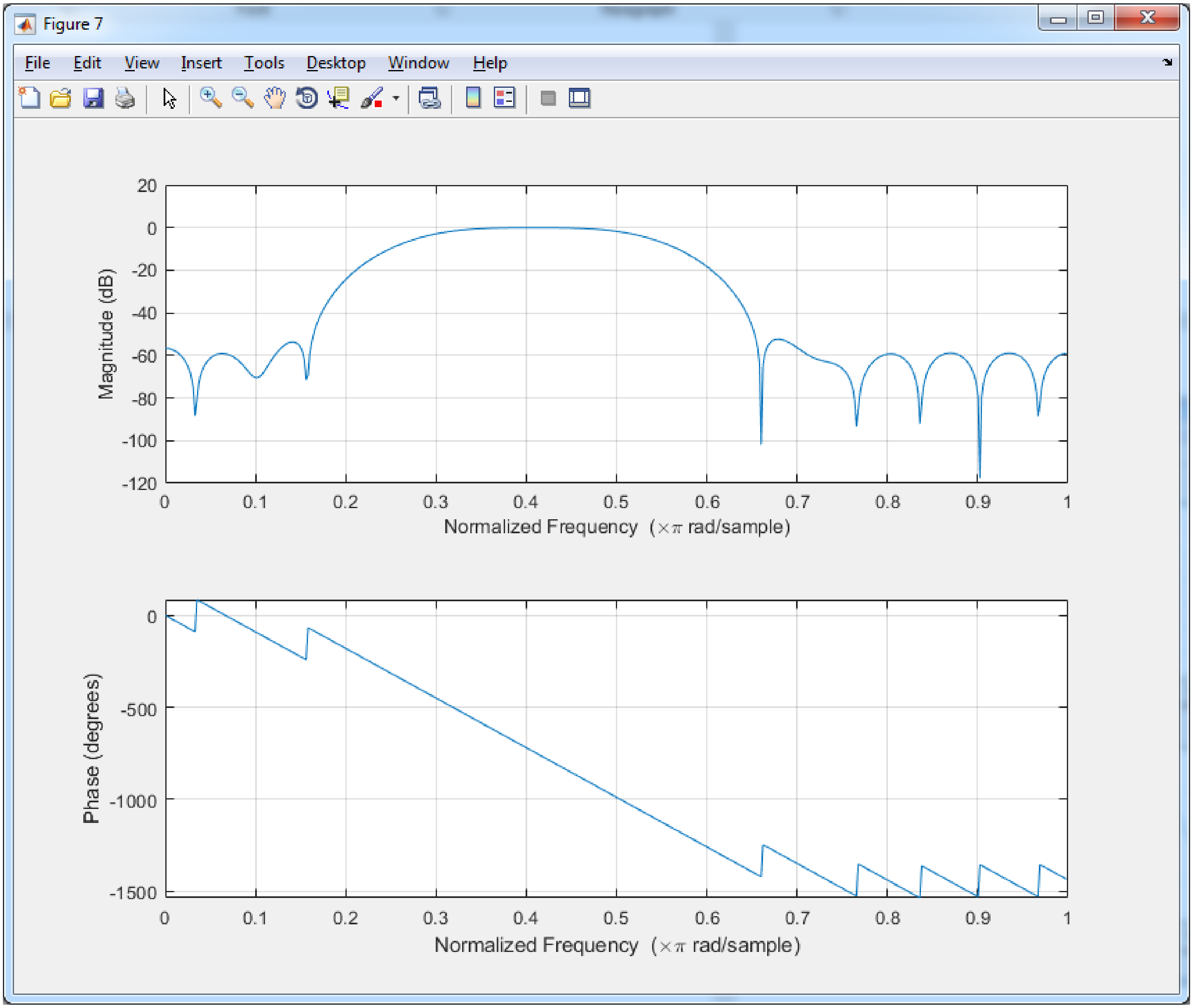
5. Filter 1 – 3KHz



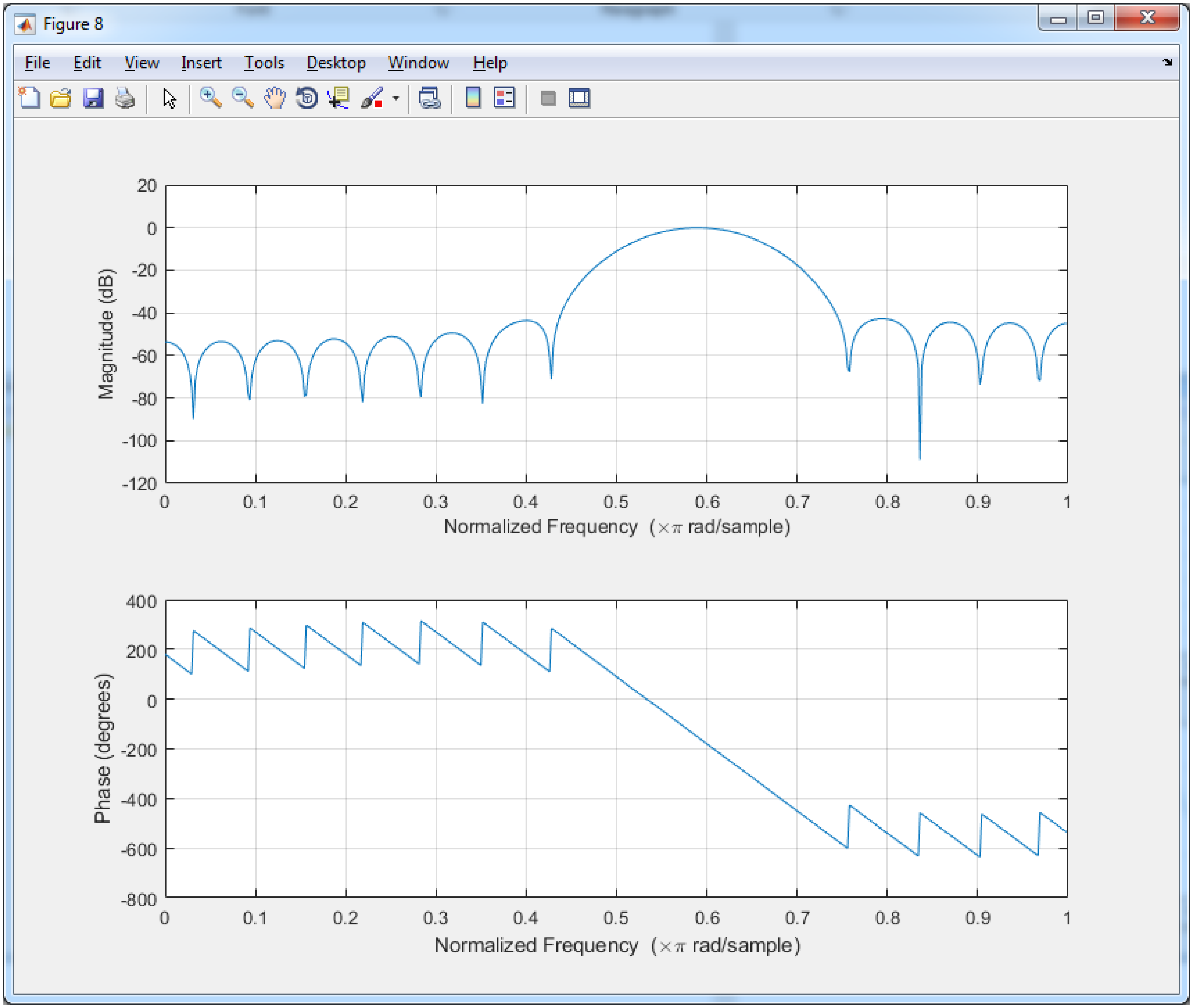
6. Filter 3 – 6KHz



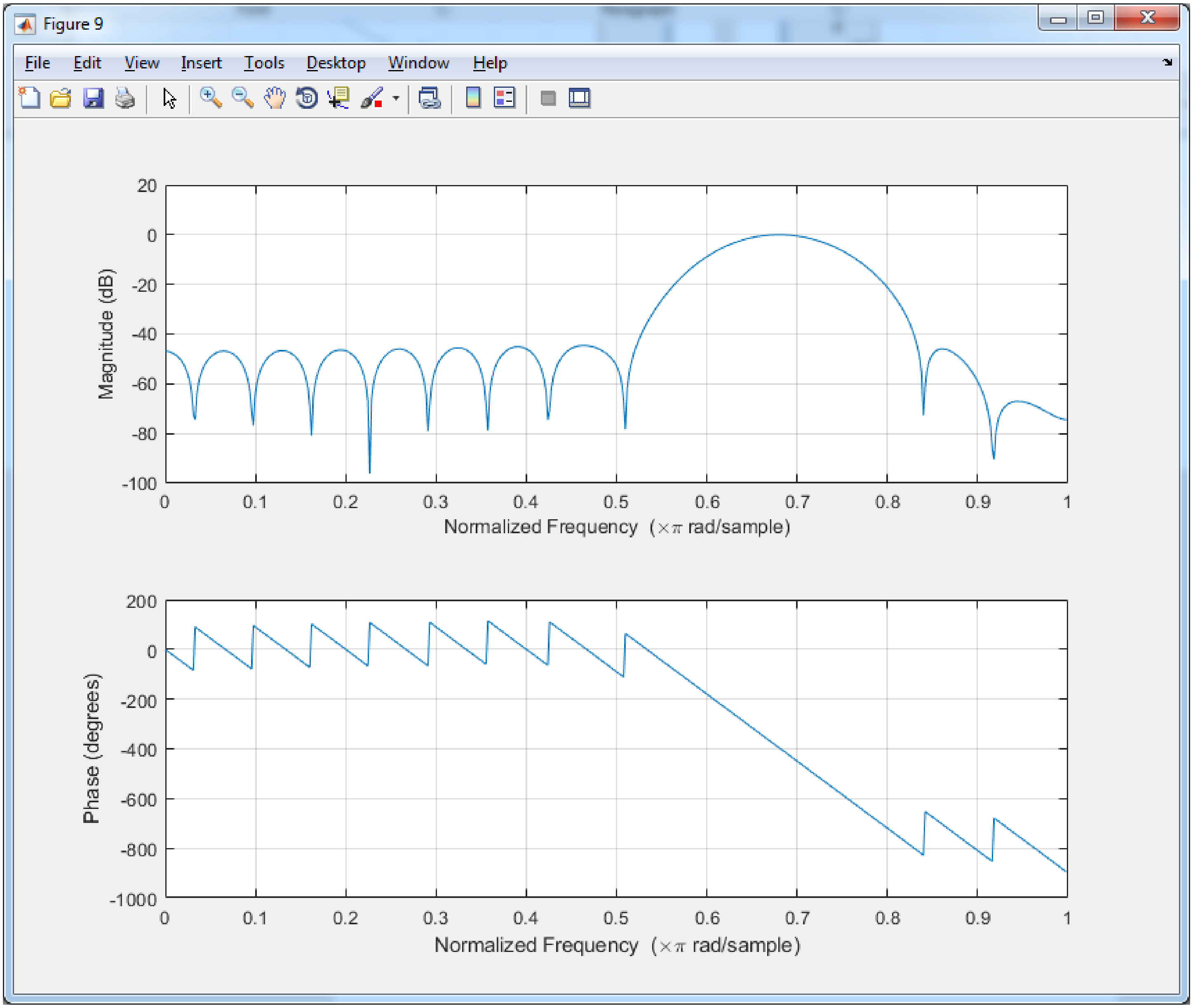
7. Filter 6 – 12KHz



8. Filter 12 – 14KHz

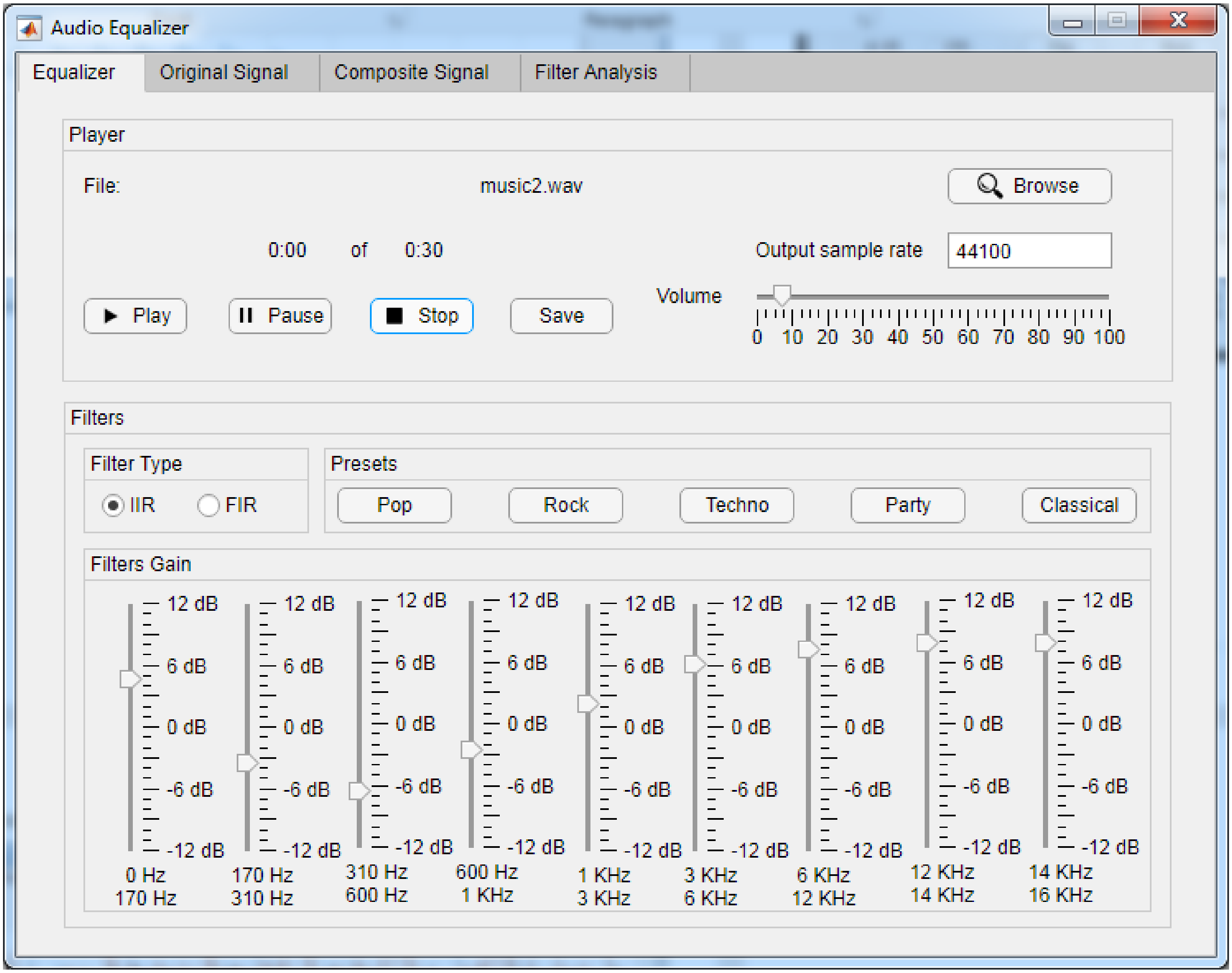


9. Filter 14 – 16KHz

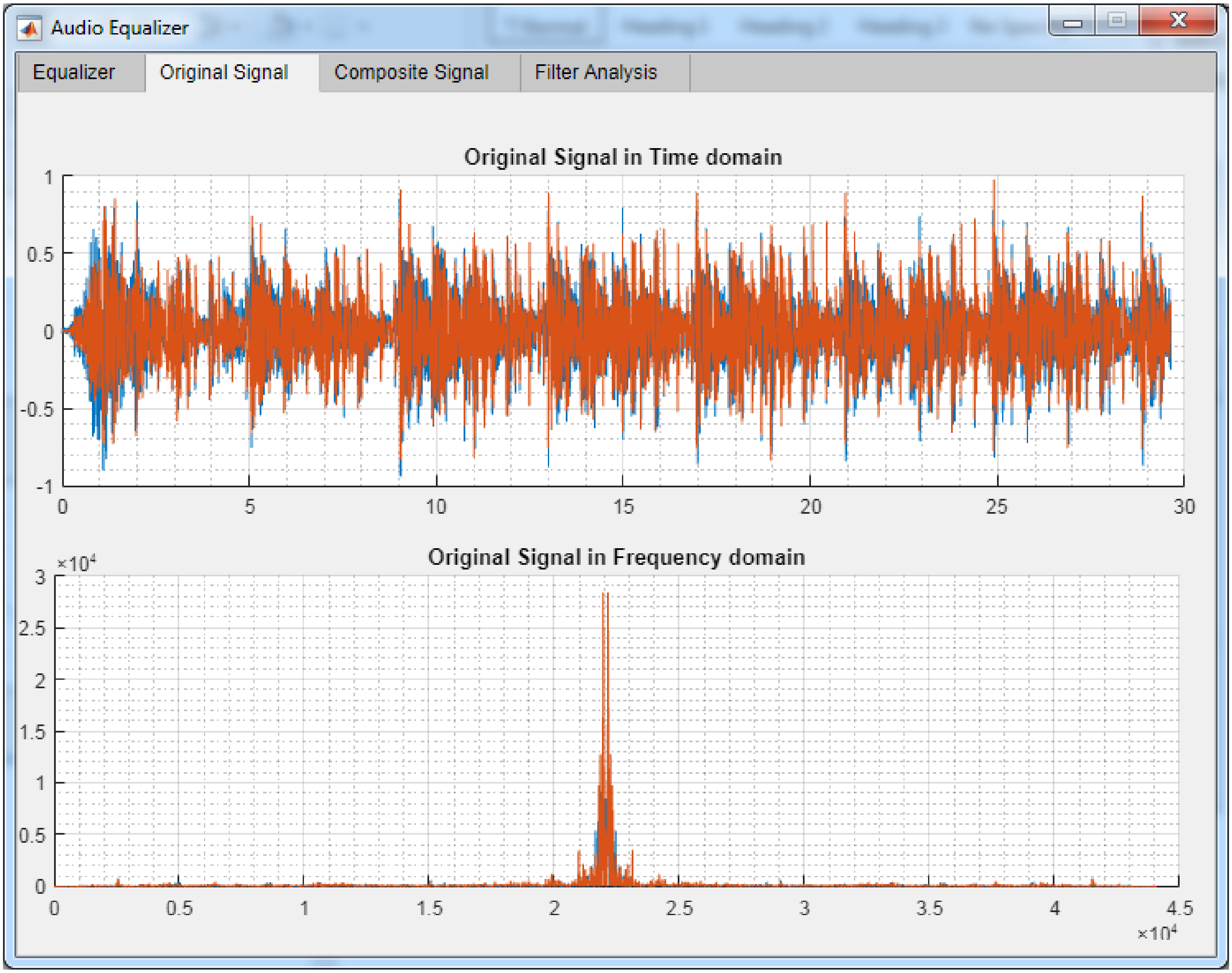


Sample runs:

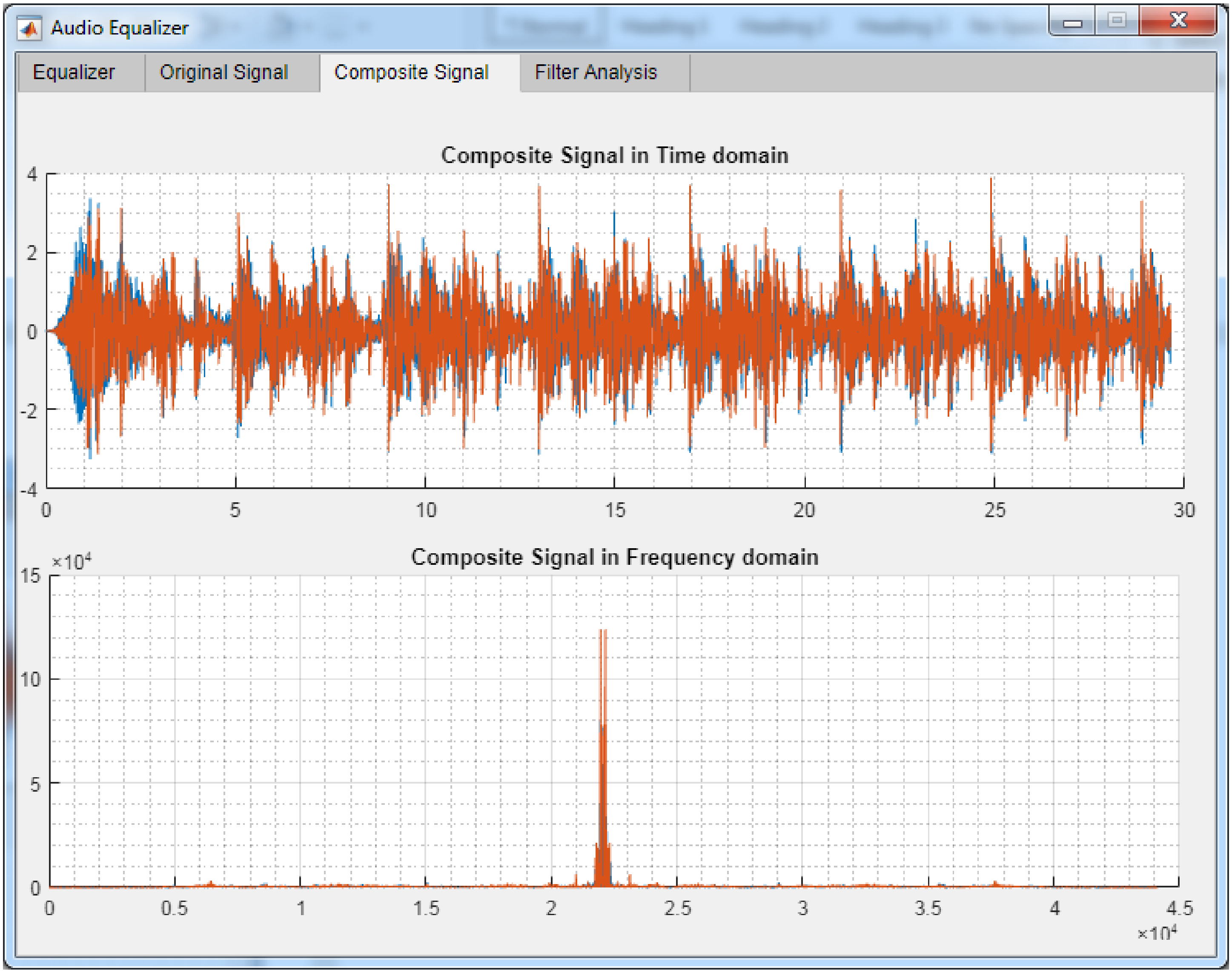
1. Sample run 1:
    - Output sample rate = 44100
    - IIR Filter



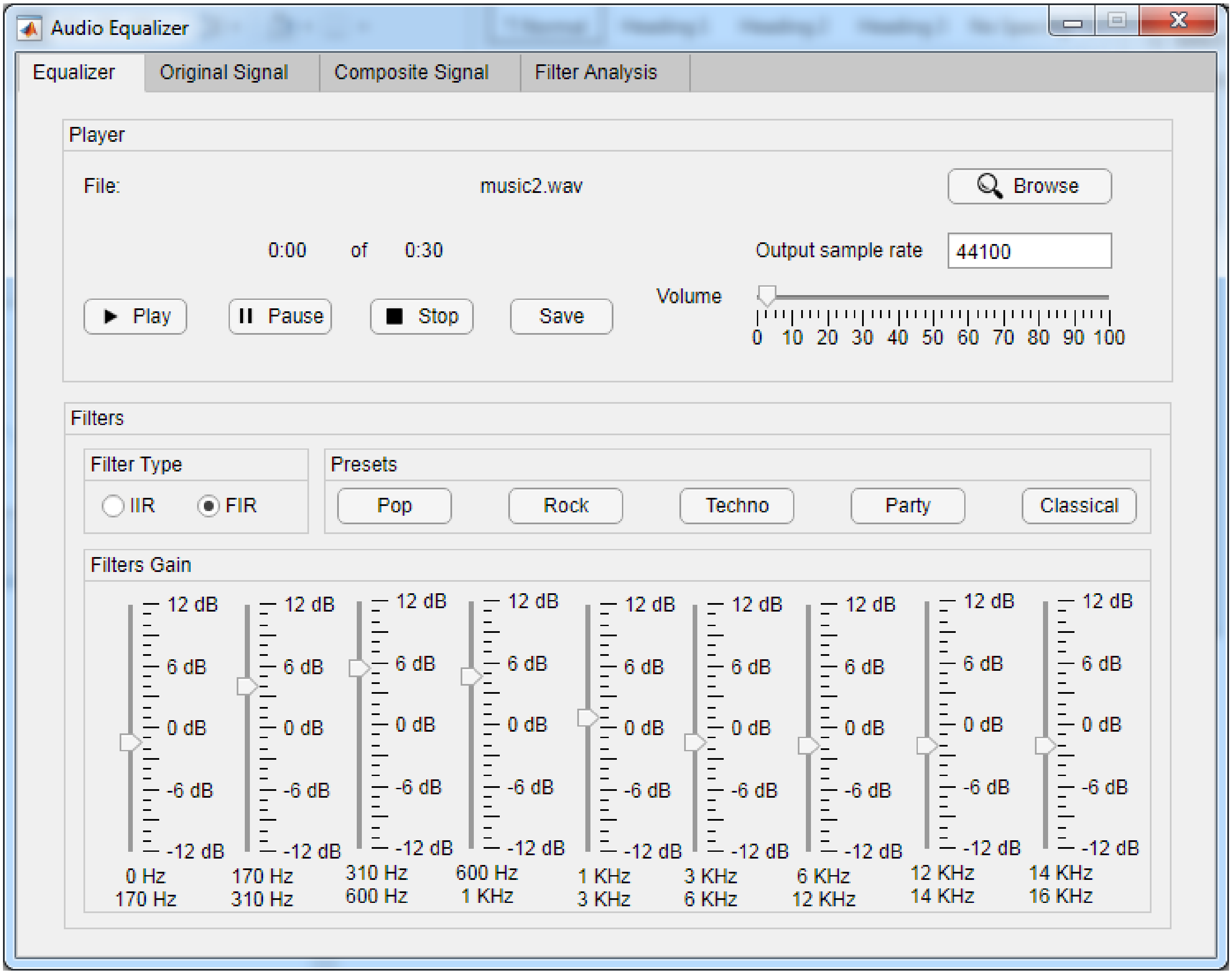
- Original Signal



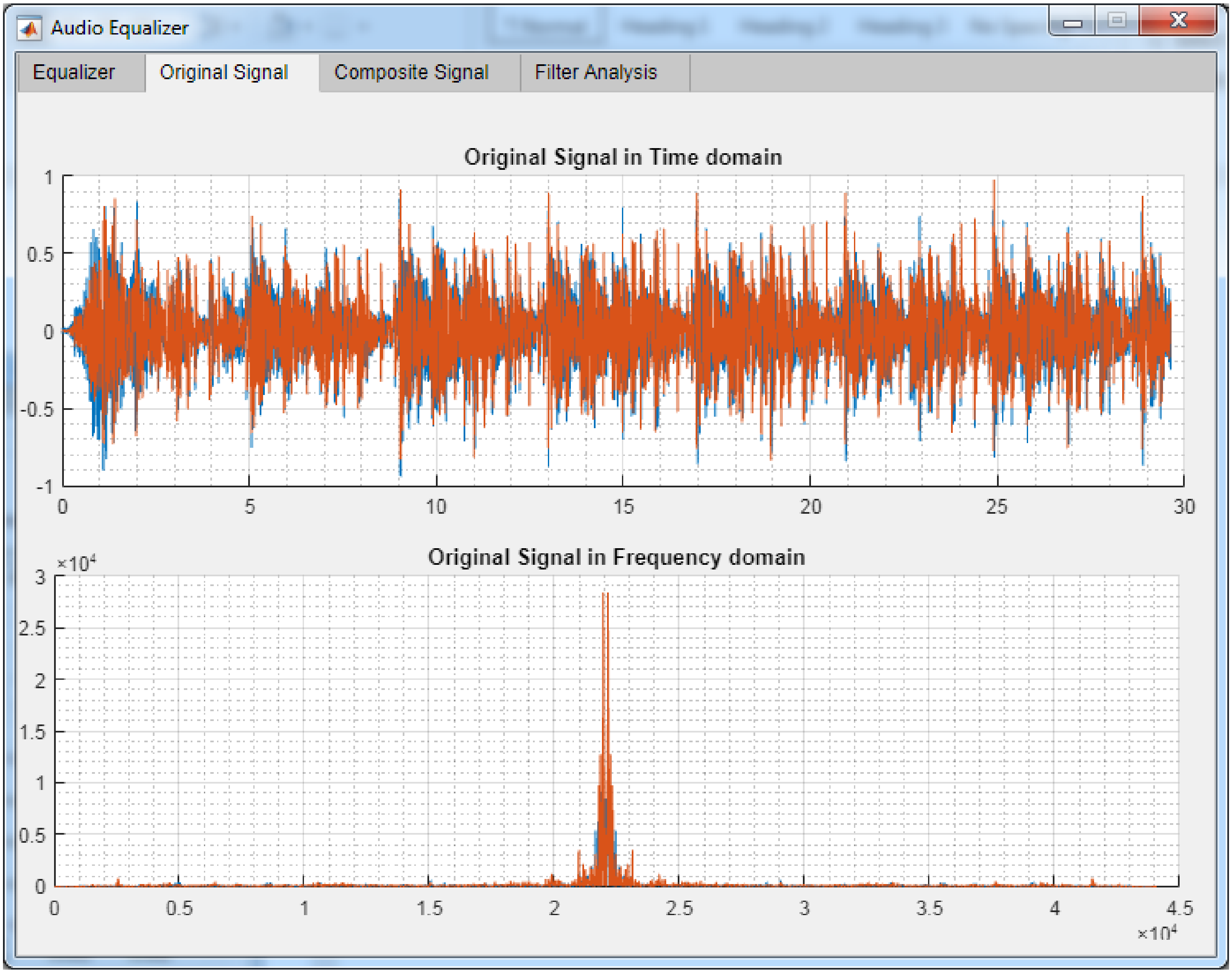
- Composite Signal



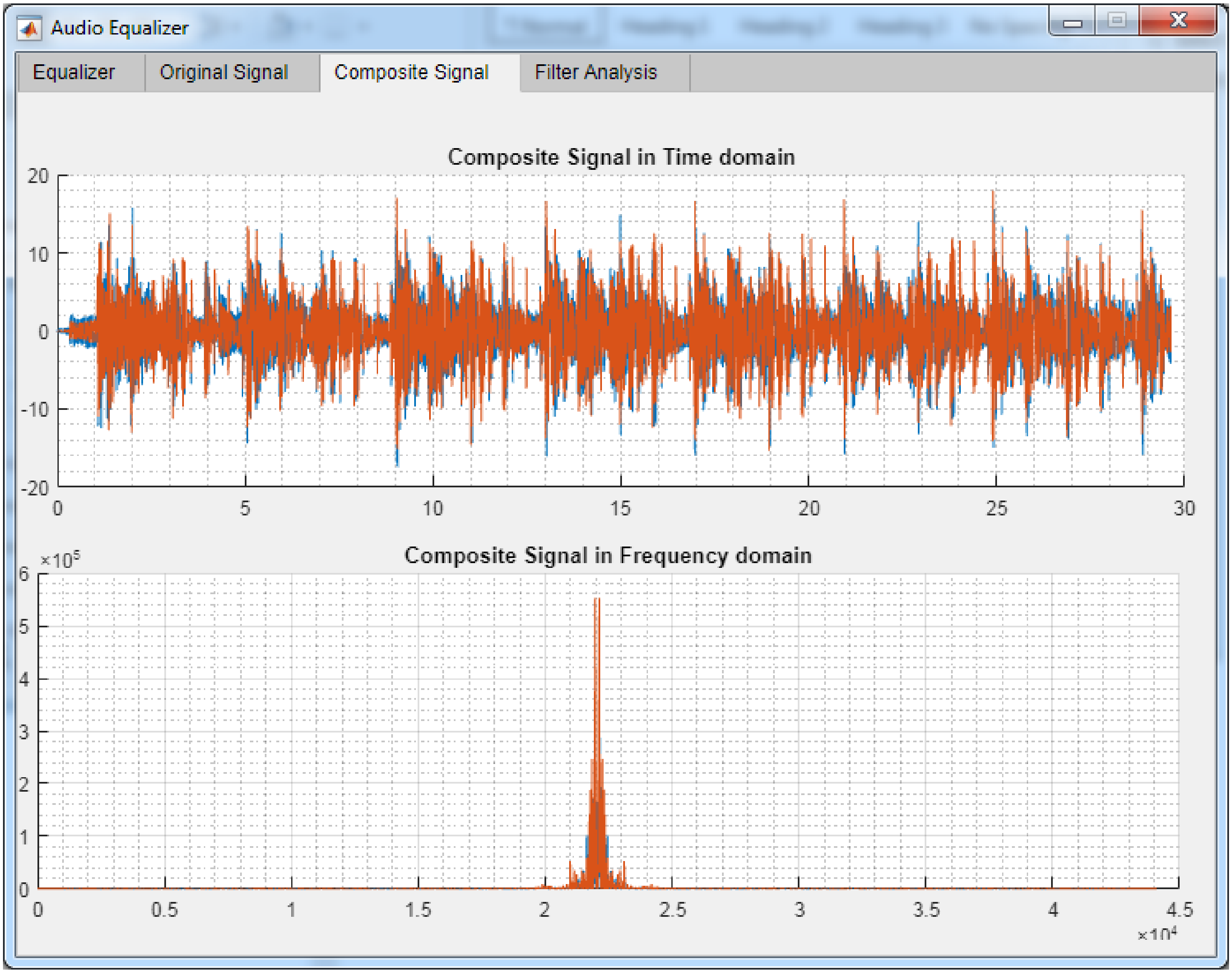
2. Sample run 2:
   - Output sample rate = 44100
   - FIR Filter



- Original signal:

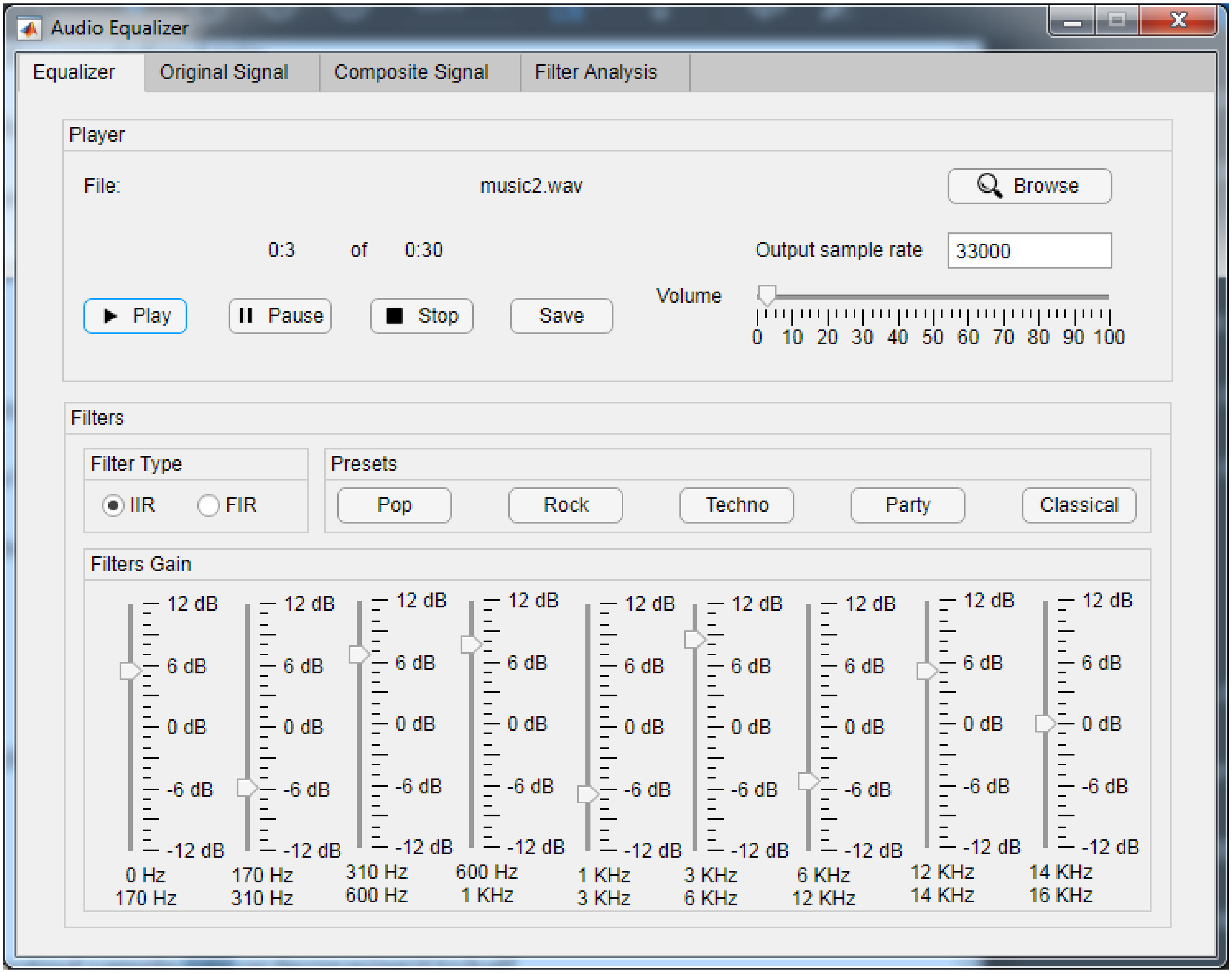


- Composite Signal

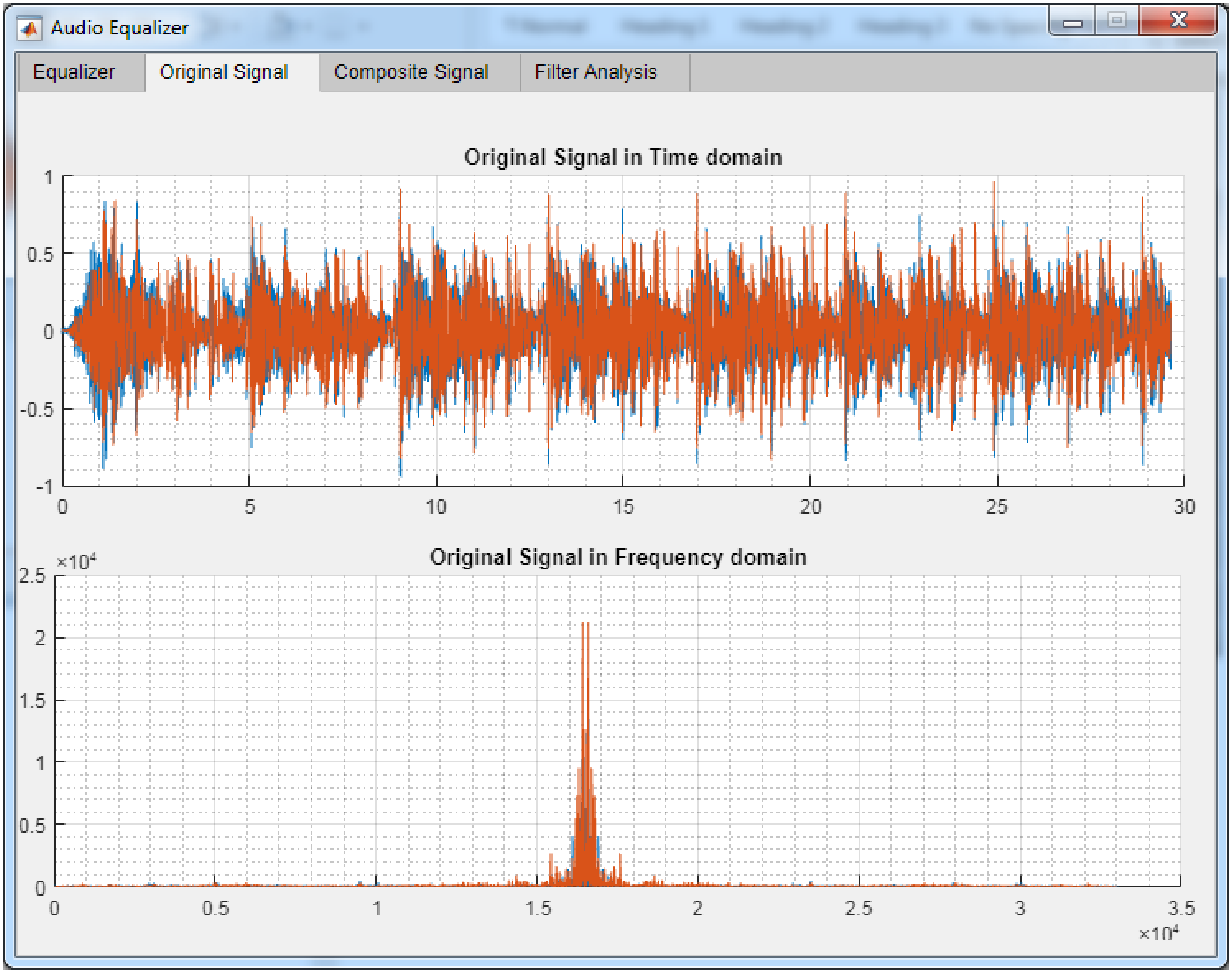


- Output signal in case if doubling output sample rate or decreasing it to half

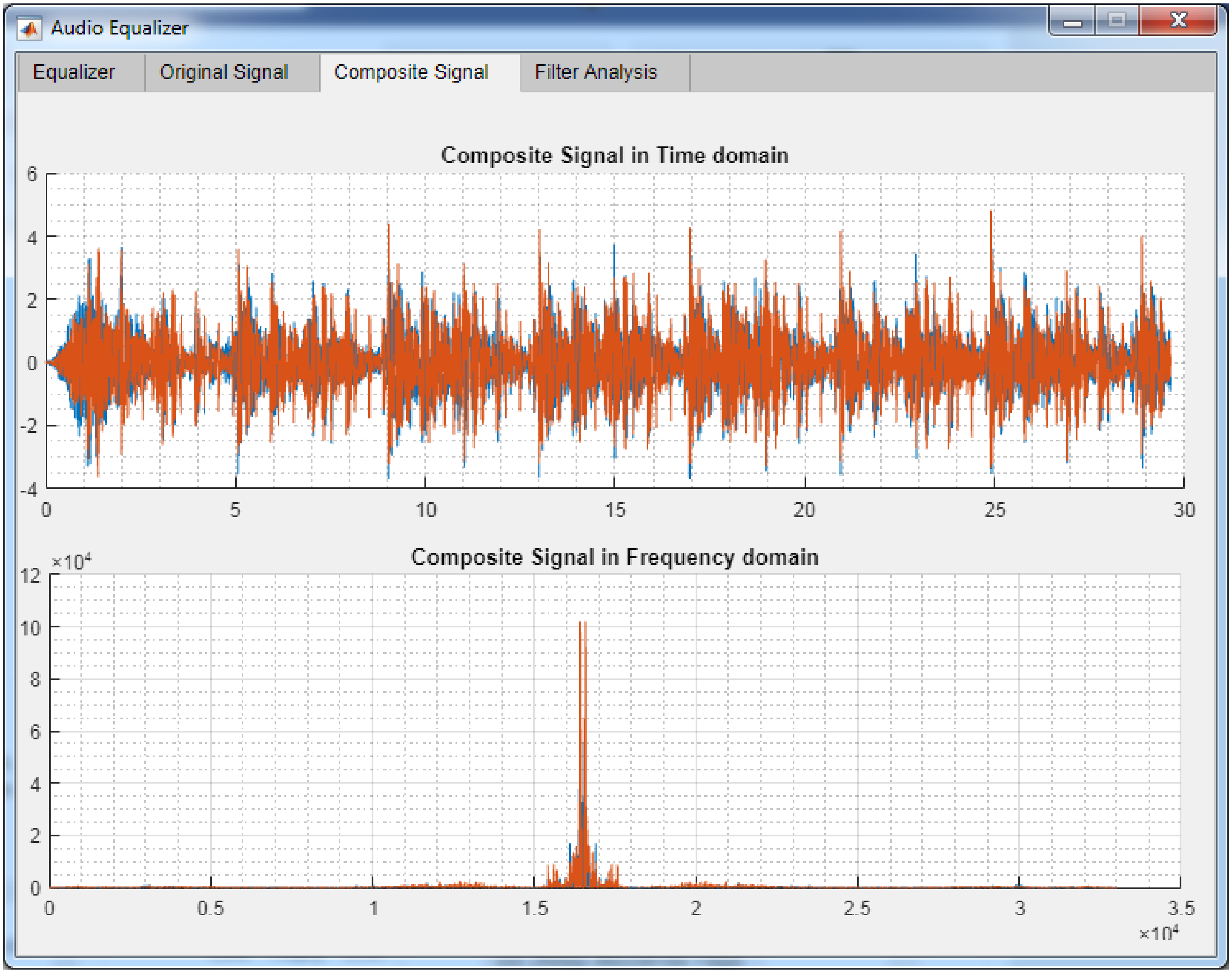
3. Sample run 3:
    - Output sample rate = 33000
    - IIR Filter



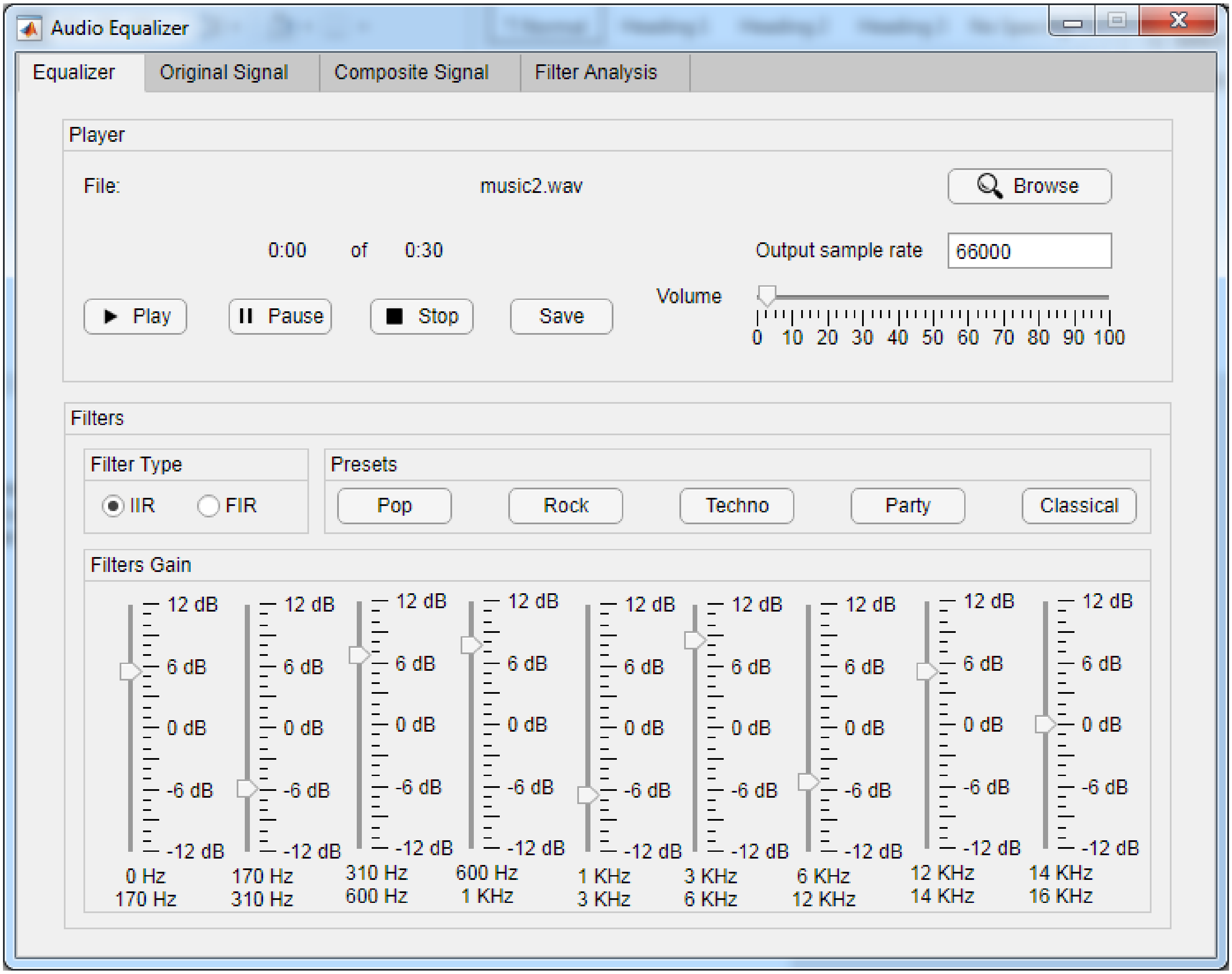
- Original signal



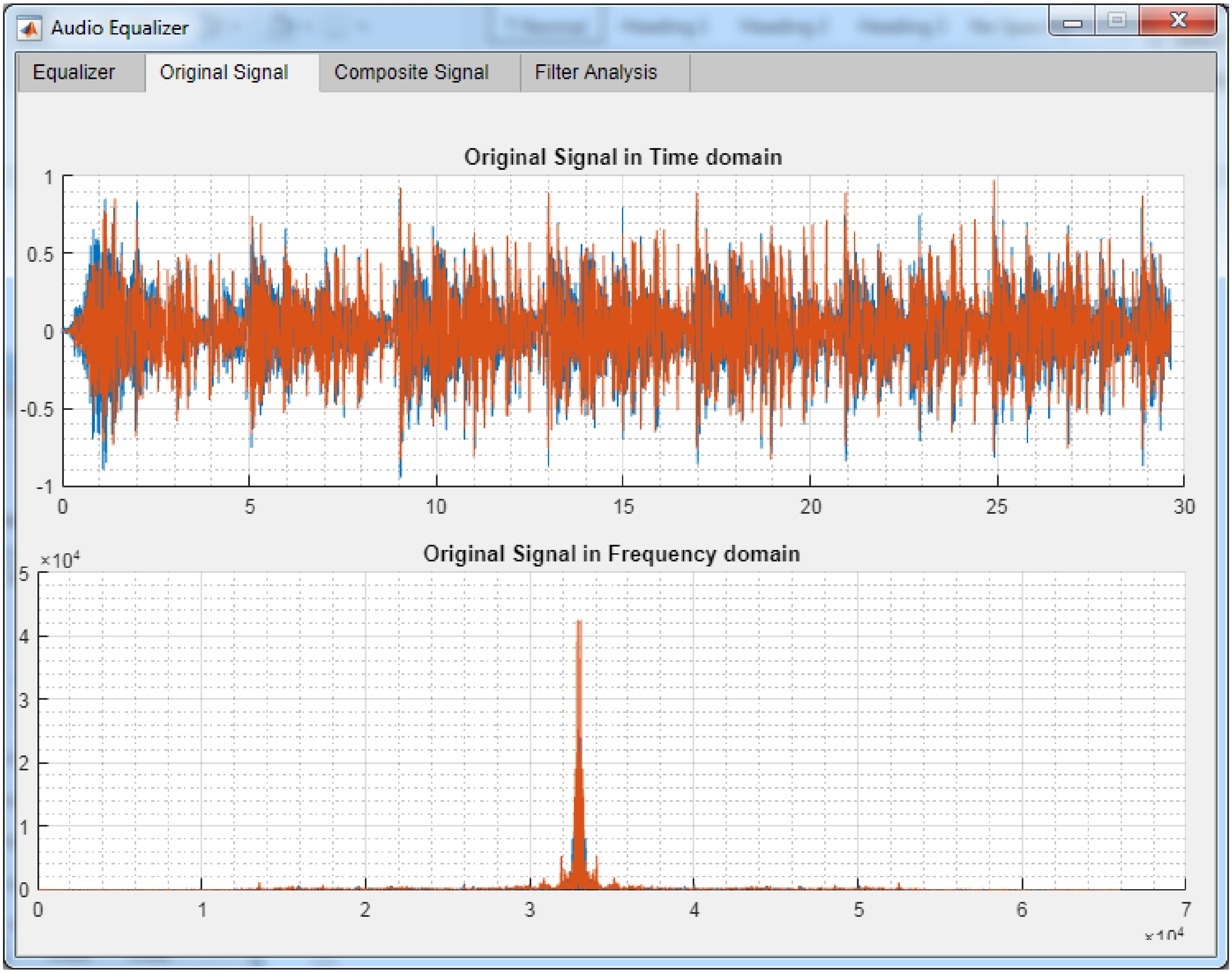
- Composite signal



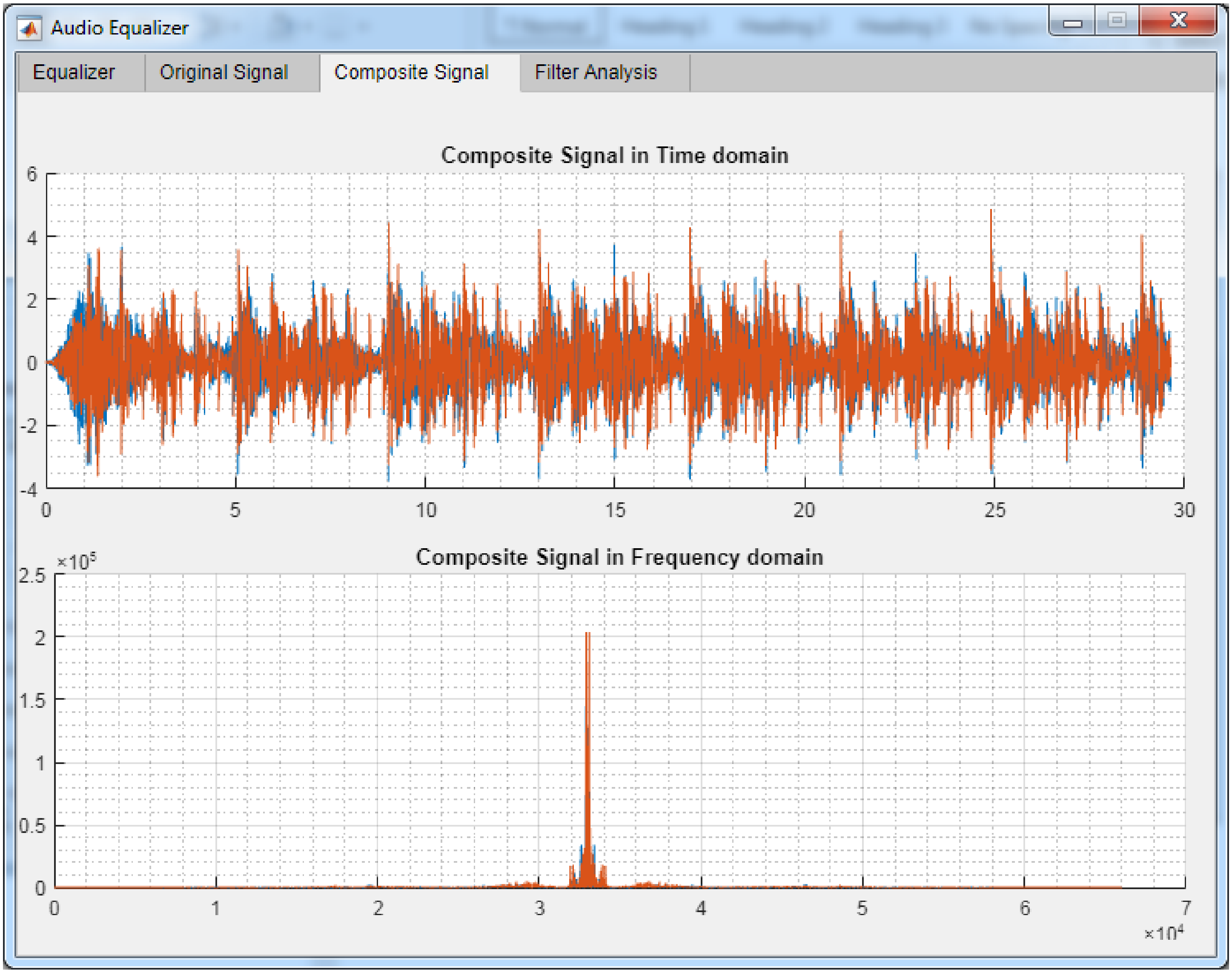
4. Sample run 4:
   - Output sample rate = 66000
   - IIR Filter



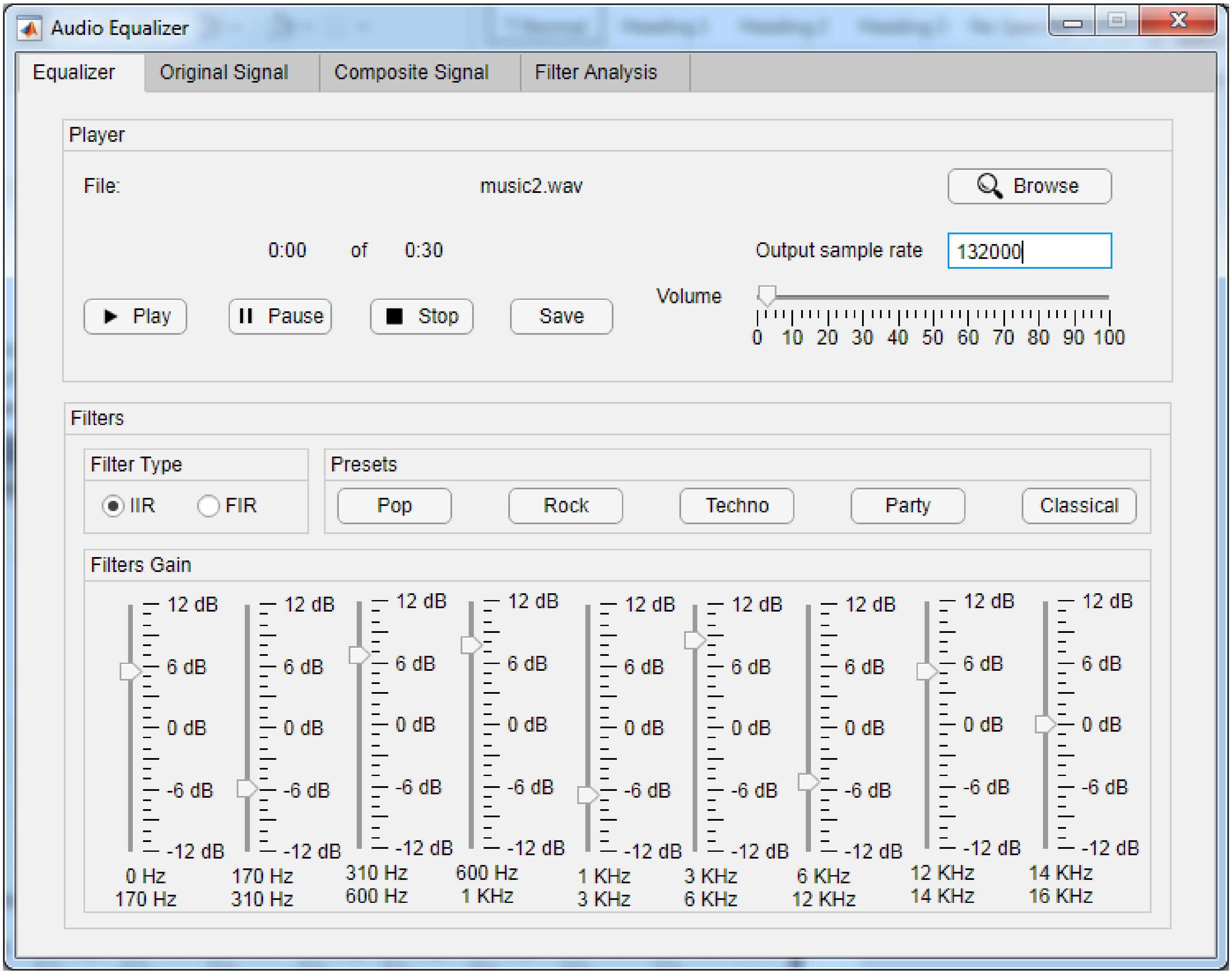
- Original signal



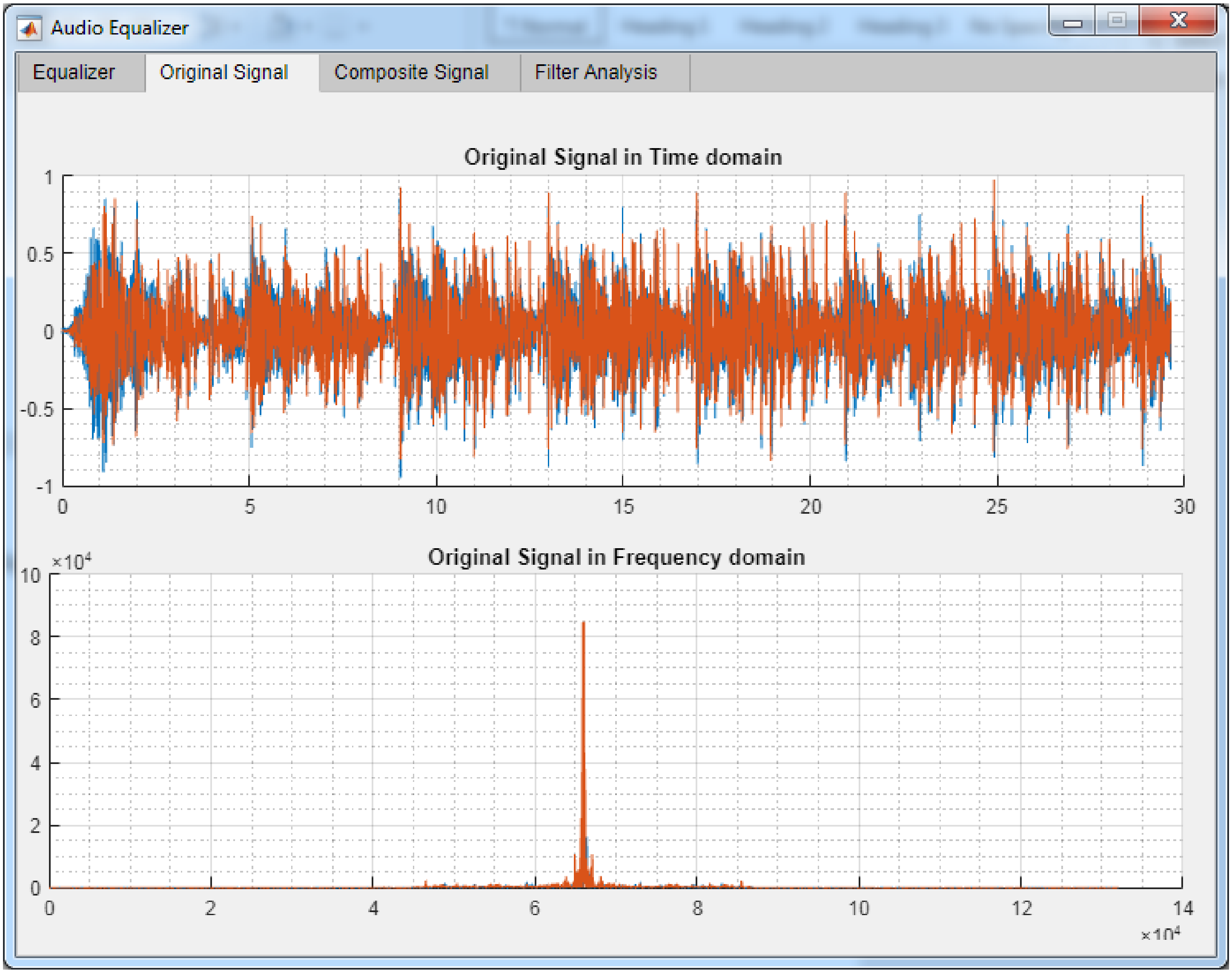
- Composite signal



5. Sample run 5:
    - Output sample rate = 132000
    - IIR Filter



- Original signal



- Composite signal