**SPECTRUM ANALYSIS USING MATLAB**

|  |
| --- |
| **Name of the student** |
| **SANCHIT KULKARNI** |

**INTRODUCTION**

To fully understand the performance of a device, a signal must also be analyzed in the frequency domain. This is exactly what the spectrum analyzer does. It should be notes, however, that with great advances in digital technology, the distinction has become little fuzzier. Some oscilloscopes can perform vector signal analysis and signal analyzer now have significant amount of time domain measurement capability. Nevertheless, oscilloscopes are optimized for time domain measurement and signal analyzers are the tool of the choice for frequency domain measurements.

In the frequency domain, complex signals (e.g., comprising more than one frequency) are separated into their frequency components and level at each frequency is displayed. Frequency domain measurements has several distinct advantages. For one information not discernible on oscilloscopes becomes readily apparent on spectrum analyzer.

Measuring signals with spectrum analyzer also greatly reduces the amount of noise present in the measurement due to analyzer ability to narrow the measurement bandwidth. Moreover, many of the today’s devices are inherently frequency domain oriented and must be analyzed in frequency domain to ensure there is no interference from neighboring frequencies.

With a frequency domain view of a spectrum it is easy to measure a signal frequency, power, harmonic content, modulation, spurs and noise. With these quantities measured, total harmonic distortion, occupied bandwidth, signal stability, output power, intermodulation distortion, power bandwidth, carrier to noise ratio, and host of other measurements then can be determined using just a spectrum analyzer.

Frequency domain measurements (spectrum analysis) are made with either a fast Fourier transform (FFT) analyzer or a swept-tuned with receiver. The FFT analyzer take a time-domain signal, digitizes it using digital sampling, and then applies the mathematics required to convert it to the frequency domain. The result is displayed as spectrum. With its real time signal analysis capability, the analyzer can capture periodic, random and transient events and can measure phase and magnitude.

**SPECTRUM ANALYSIS USING FFT IN MATLAB**

To begin this discussion spectral analysis, let us begin by considering the question of trying to detect and underlying sinusoidal signal component that is buried in noise.

**Record and Play Audio in MATLAB**

**Record Audio**

To record data from an audio input device (such as a microphone connected to your system) for processing in MATLAB:

1) Create an audio recorder object.

2) Call the record or record blocking method, where record returns immediate control to the calling function or the command prompt even as recording proceeds. Specify the length of the recording in seconds, or end the recording with the stop method. Optionally, call the pause and resume methods. The recording is performed asynchronously, record blocking retains control until recording is compete. Specify the length if the recording in seconds. The recording is performed synchronously.

3) Create a numeric array corresponding to other signal data using the getaudiodata method.

**The following example show how to use the record blocking and record methods**

* Create an audio recorder object named voice1 for recording audio input

voice1= audio recorder//audio recorder creates an 8000hz, 8bit, 1 channel audio recorder object.

* Record your voice for 5 seconds.

disp (‘Start Speaking:’)

recordblocking(voice1, 5);

disp(‘End of recording.’);

* Play back the recording.

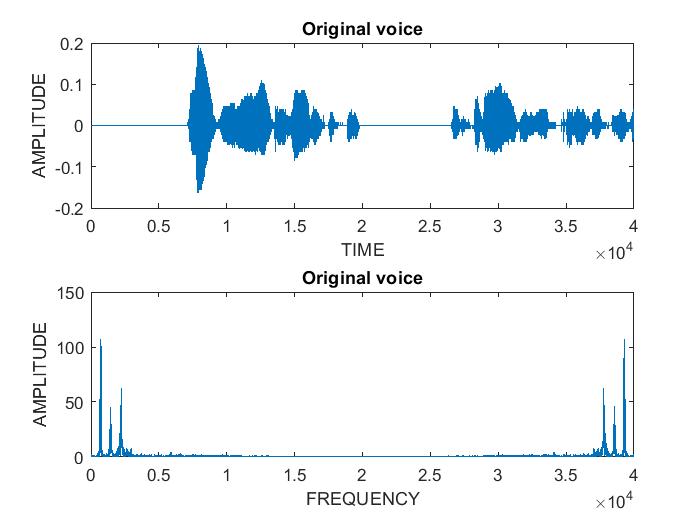
play(voice1);

* Store data in double- precision array, y

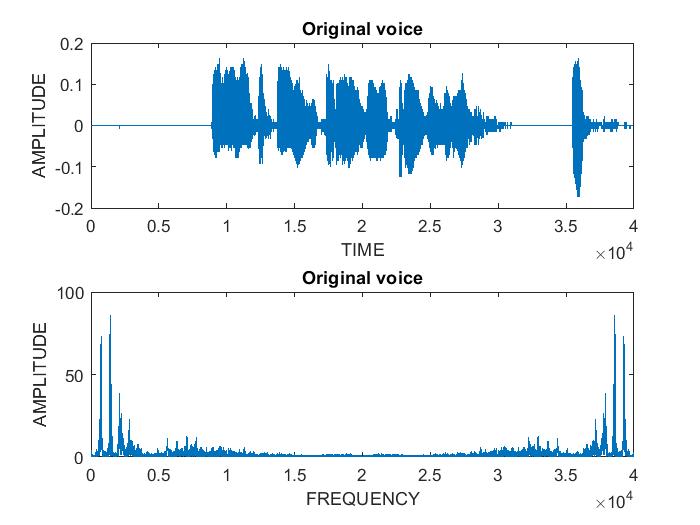
y= getaudiodata (voice1);

* Plot the audio samples
* To plot the given speech signal in frequency domain, apply FFT and plot the same in MATLAB

Audio1:



Audio2:



**APPLYING IIR FILTER**

IIR filters are digital filters with infinite impulse response. Unlike FIR filters, they have the feedback (recursive part of a filter) and are known as recursive digital filters therefore.

For this reason, IIR filters have much better frequency than FIR filters of same order. Unlike FIR filters, their phase characteristics is not linear which cause problem to the systems which need phase linearity. For this reason, it is not preferable to use IIR filters in DSP when the phase is of the essence.

Otherwise, when the linear phase characteristics is not important, the use of IIR filters is an excellent solution.

There is one problem known as potential instability that is typical of IIR filters only. FIR filters do not have such a problem as thy do not have the feedback. For this reason, it is always necessary to check after the design process whether the resulting IIR filter is stable or not.

IIR filters can be designed in different methods. One of the most commonly used is via the reference prototype filter. This method is best for designing all standard types of all filters such as low pass, high pass, bandpass and bandstop filter.

**Implementing IIR (butterworth) filter in MATLAB**

1. **Butterworth low Pass Filter:**

fs = 8000;

[y,fs]= audioread('voice1.wav');

sound(y,fs);

[N,wn]=buttord(0.1,0.3,1,30);

[b,a]=butter(N,wn);

x=filter(b,a,y);

subplot(2,1,1);

plot(x)

xlabel('TIME');

ylabel('AMPLITUDE');

title('Filtered voice(lowpass filter) Time domain');

subplot(2,1,2);

l=fft(x);

plot(abs(l))

xlabel('FREQUENCY');

ylabel('AMPLITUDE');

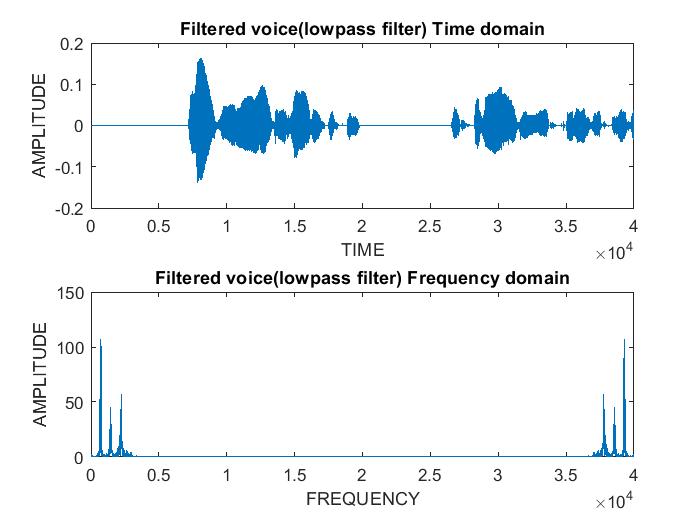
title('Filtered voice(lowpass filter) Frequency domain');

audiowrite('butterworth\_LP.wav', x, fs);

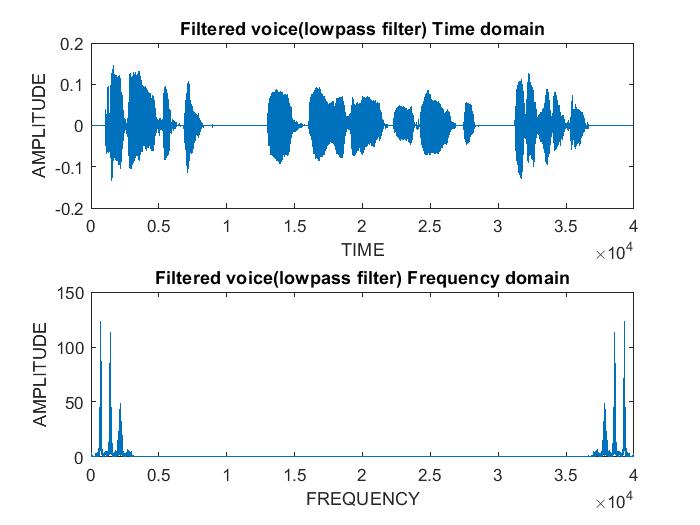
figure(2)

freqz(b,a)

Audio1 output:



Audio 2 output:



1. **Butterworth High Pass Filter:**

fs = 8000;

[y,fs]= audioread('voice1.wav');

sound(y,fs);

[N,wn]=buttord(0.3,0.1,0.4,35);

[b,a]=butter(N,wn,'high');

x=filter(b,a,y);

subplot(2,1,1);

plot(x)

xlabel('TIME');

ylabel('AMPLITUDE');

title('Filtered voice(highpass filter) Time domain');

subplot(2,1,2);

l=fft(x);

plot(abs(l))

xlabel('FREQUENCY');

ylabel('AMPLITUDE');

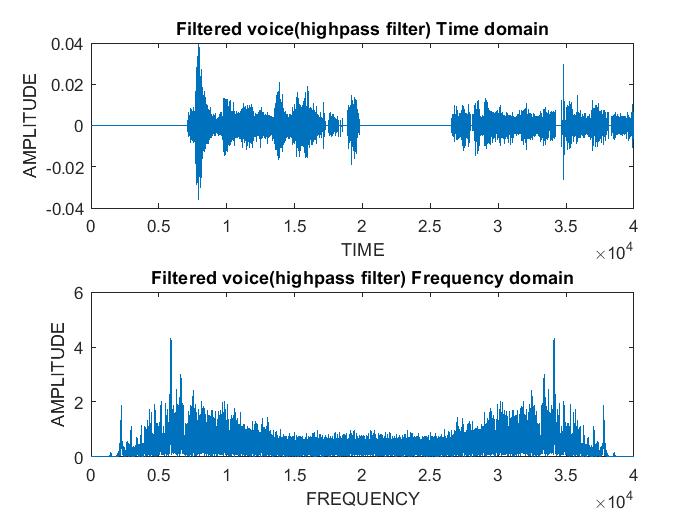
title('Filtered voice(highpass filter) Frequency domain');

audiowrite('butterworth\_HP.wav', x, fs);

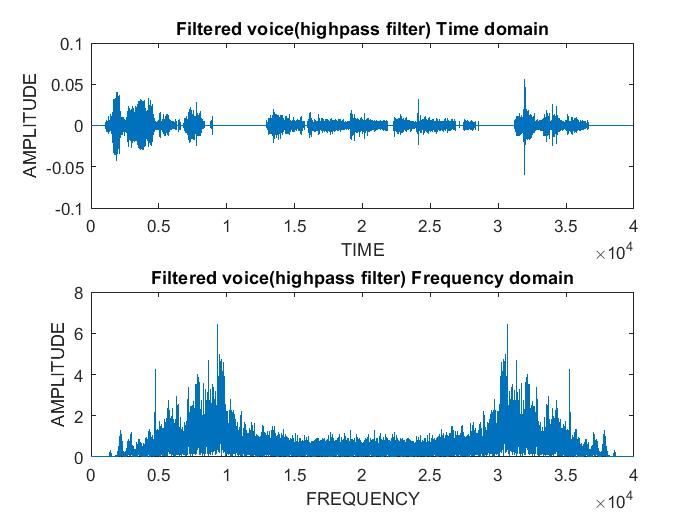
figure(2)

freqz(b,a)

**Audio 1 output:**



**Audio 2 output:**



1. **Butterworth Band Pass Filter:**

fs = 8000;

[y,fs]= audioread('voice1.wav');

sound(y,fs);

[N,wn]=buttord([0.3 0.4],[0.1 0.5],0.1,30);

[b,a]=butter(N,wn);

x=filter(b,a,y);

subplot(2,1,1);

plot(x)

xlabel('TIME');

ylabel('AMPLITUDE');

title('Filtered voice(bandpass filter) Time domain');

subplot(2,1,2);

l=fft(x);

plot(abs(l))

xlabel('FREQUENCY');

ylabel('AMPLITUDE');

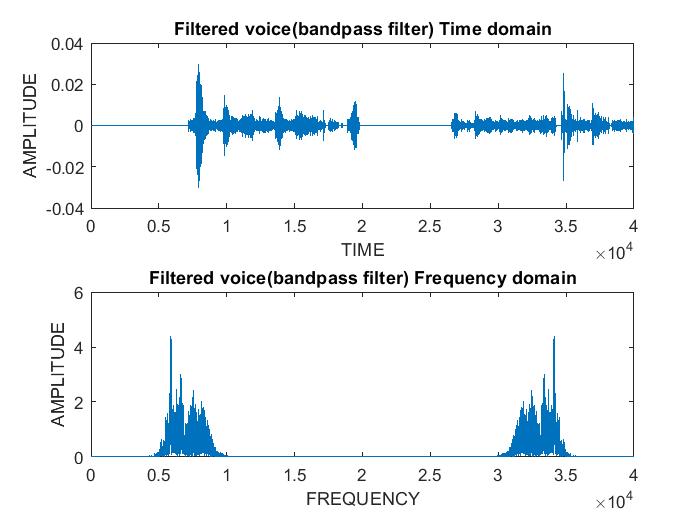
title('Filtered voice(bandpass filter) Frequency domain');

audiowrite('butterworth\_BP.wav', x, fs);

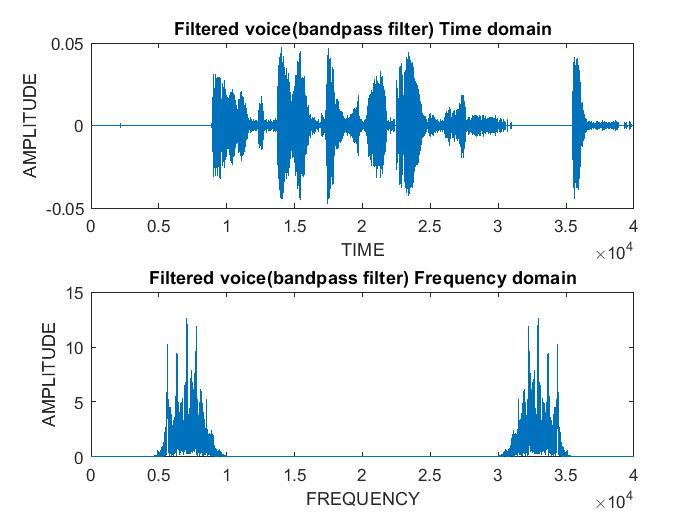
figure(2)

freqz(b,a)

**Audio 1 output:**



**Audio 2 output:**



1. **Butterworth Band Stop Filter:**

fs = 8000;

[y,fs]= audioread('voice1.wav');

sound(y,fs);

subplot(2,1,1);

[N,wn]=buttord([0.1 0.5],[0.2 0.4],0.1,35);

[b,a]=butter(N,wn,'stop');

x=filter(b,a,y);

subplot(2,1,1);

plot(x)

xlabel('TIME');

ylabel('AMPLITUDE');

title('Filtered voice(bandstop filter) Time domain');

subplot(2,1,2);

l=fft(x);

plot(abs(l))

xlabel('FREQUENCY');

ylabel('AMPLITUDE');

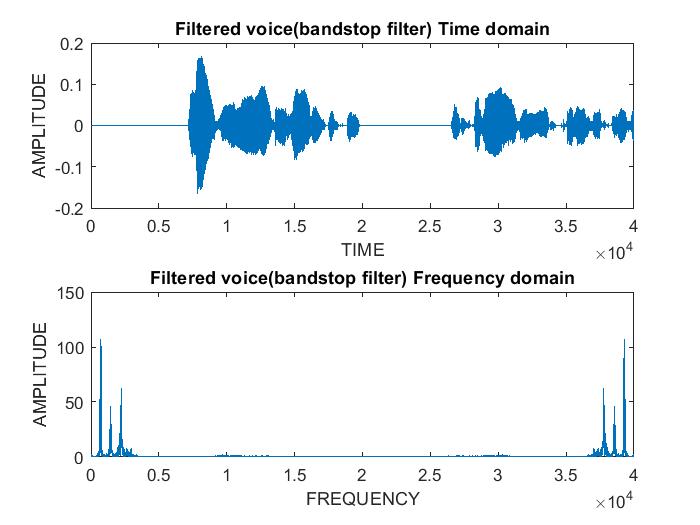
title('Filtered voice(bandstop filter) Frequency domain');

audiowrite('butterworth\_BS.wav', x, fs);

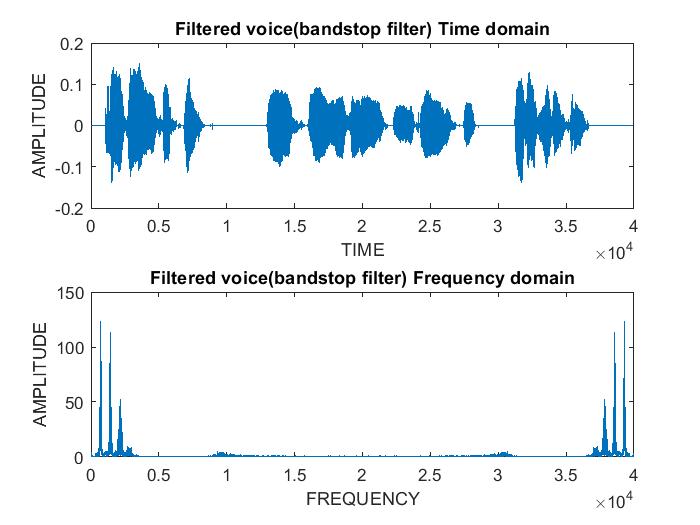
figure(2)

freqz(b,a)

Audio1 output:



Audio2 output:



**APPLYING FIR FILTERS**

In signal processing, a **finite impulse response** (**FIR**) **filter** is a filter whose impulse response (or response to any finite length input) is of *finite* duration, because it settles to zero in finite time. This is in contrast to infinite impulse response (IIR) filters, which may have internal feedback and may continue to respond indefinitely (usually decaying).

The impulse response (that is, the output in response to a [Kronecker delta](https://en.wikipedia.org/wiki/Kronecker_delta) input) of an Nth-order discrete-time FIR filter lasts exactly *N* + 1 samples (from first nonzero element through last nonzero element) before it then settles to zero.

FIR filters can be [discrete-time](https://en.wikipedia.org/wiki/Discrete-time) or [continuous-time](https://en.wikipedia.org/wiki/Discrete_time_and_continuous_time), and [digital](https://en.wikipedia.org/wiki/Digital_data) or [analog](https://en.wikipedia.org/wiki/Analog_circuits).

An FIR filter has a number of useful properties which sometimes make it preferable to an [infinite impulse response](https://en.wikipedia.org/wiki/Infinite_impulse_response) (IIR) filter. FIR filters:

* Require no feedback. This means that any rounding errors are not compounded by summed iterations. The same relative error occurs in each calculation. This also makes implementation simpler.
* Are inherently [stable](https://en.wikipedia.org/wiki/BIBO_stability), since the output is a sum of a finite number of finite multiples of the input values, so can be no greater than {\displaystyle \scriptstyle \sum |b\_{i}|} times the largest value appearing in the input.
* Can easily be designed to be [linear phase](https://en.wikipedia.org/wiki/Linear_phase) by making the coefficient sequence symmetric. This property is sometimes desired for phase-sensitive applications, for example data communications, [seismology](https://en.wikipedia.org/wiki/Seismology), [crossover filters](https://en.wikipedia.org/wiki/Audio_crossover), and [mastering](https://en.wikipedia.org/wiki/Audio_mastering).

The main disadvantage of FIR filters is that considerably more computation power in a general purpose processor is required compared to an IIR filter with similar sharpness or [selectivity](https://en.wikipedia.org/wiki/Selectivity_(electronic)), especially when low frequency (relative to the sample rate) cutoffs are needed. However, many digital signal processors provide specialized hardware features to make FIR filters approximately as efficient as IIR for many applications.

**Implementing FIR filter (windows)in MATLAB**

1. **Hanning window:**

fs = 8000;

[y,fs]= audioread('voice1.wav');

sound(y,fs);

h=fir1(20,0.1,hanning(21));

flhn=filter(h,1,c);

subplot(2,1,1);

plot(x)

xlabel('TIME');

ylabel('AMPLITUDE');

title('Filtered voice(Hanning Window) Time domain');

subplot(2,1,2);

l=fft(x);

plot(abs(l))

xlabel('FREQUENCY');

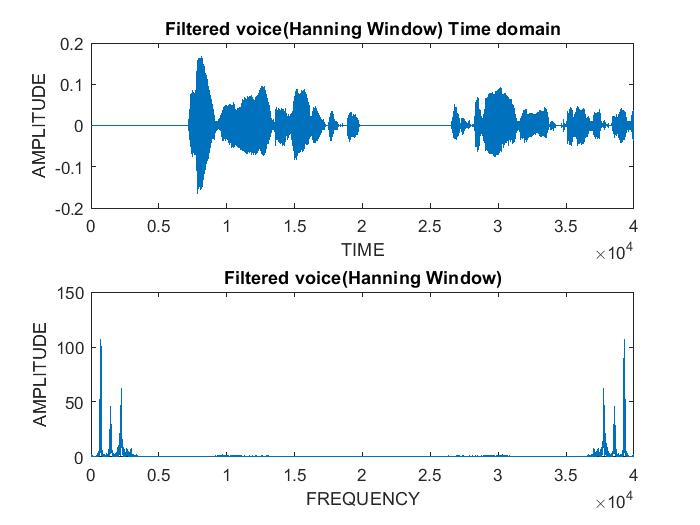
ylabel('AMPLITUDE');

title('Filtered voice(Hanning Window)');

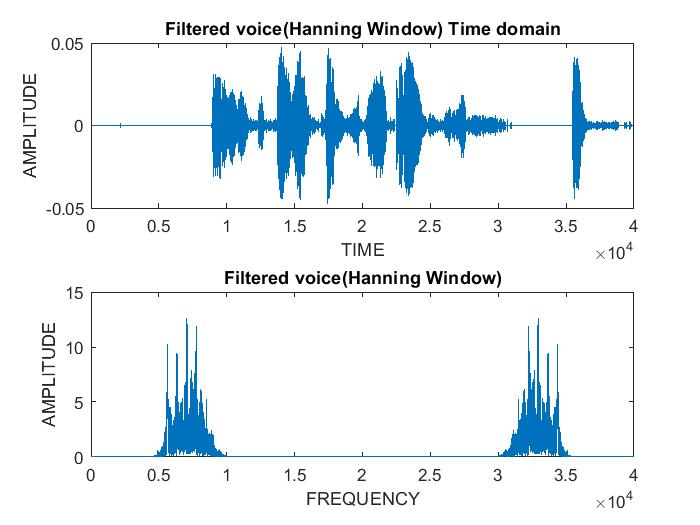
audiowrite('Hanning window.wav', x, fs);

figure(2)

Audio 1 output:



Audio 2 output:



1. **Hanning window:**

fs = 8000;

[y,fs]= audioread('voice1.wav');

sound(y,fs);

h=fir1(20,0.1,hamming(21));

flhn=filter(h,1,c);

subplot(2,1,1);

plot(x)

xlabel('TIME');

ylabel('AMPLITUDE');

title('Filtered voice(Hamming Window) Time domain');

subplot(2,1,2);

l=fft(x);

plot(abs(l))

xlabel('FREQUENCY');

ylabel('AMPLITUDE');

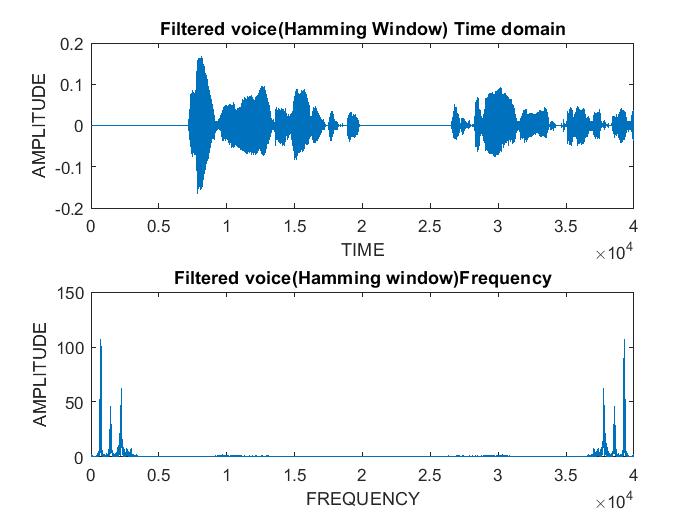
title('Filtered voice(Hamming window)Frequency');

audiowrite('Hamming window.wav', x, fs);

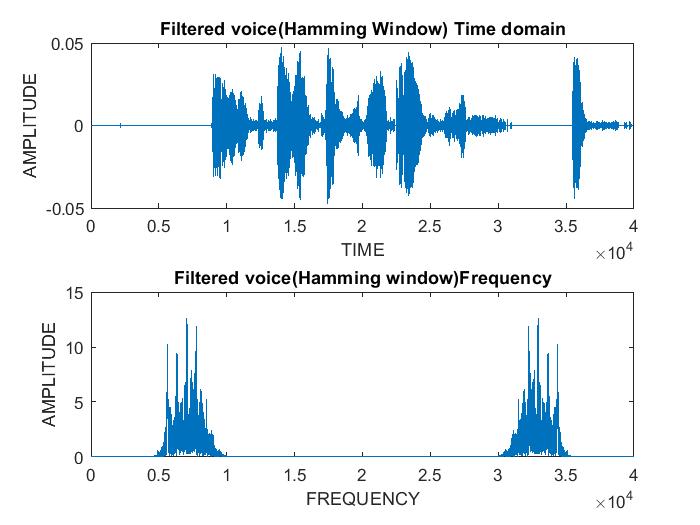
figure(2)

plot(flhn);

**Audio 1 output:**



**Audio 2 output:**



**COMPARING IIR AND FIR FILTERS**

1. IIR is infinite and used for applications where liner characteristics are nor of concerned.

2. FIR filters are finite IIR filters which are required for linear phase characteristics.

3. IIR is better for lower order tapping, where as the FIR filter is used for higher oder tapping

4. FIR filters are preferred over IIR because they are more stable, and ffedback is not involved.

5. IIR filters are recursive and used as an alternate, whereas Fir filters have become too long and cause problems in various applications.