ECE 437 Digital Signal Processing
Dusan Veselinovic
Computer Project

Marcus Favila
07 Dec 2019

Abstract

This project implemented the use of infinite impulse response (IIR) and finite impulse response (FIR) filters on an input signal to produce a modified output signal in relation to the input signal. Additionally, analysis and comparison on the filters created were analyzed. An IIR filter was implemented from the linear and time-invariant filter equation. Three FIR filters were implemented using variations of the overlap-add method with varying block sizes and the time-domain filtering method. Analysis of each filter was performed, and visual and auditory representations of Linear Time-invariant systems and impulse responses were demonstrated. Characteristics such as benefits and drawbacks to implementing IIR filters and varying FIR filter methods were also analyzed.

Introduction

The input signal used in this project was an audio sample corrupted with feedback at a certain frequency. A corresponding transfer function was used to attenuate the frequency corrupting the signal. The IIR filter was based upon the coefficients of the transfer function, relating to the impulse response of the system. The FIR filter was based upon the coefficients of the transfer function as well, but were modified such that the characteristically different than the transfer function used in the IIR Filter. Analysis of each filter was conducted in MATLAB to calculate the output signal samples from the input signal.

Methods

To generate the filtered output signal, the transfer function (*H(z)*) was represented as the two arrays, b and a, such that b are the coefficients relating to the numerator of *H(z)*, and a are the coefficients of the denominator of H(z). Where b(n) is the coefficient of z^{-n-1} term and likewise for a(n). Thus the output signal y(n) can be represented as $y(n) = \sum_{k=1}^{M+1} b_k\, x(n-k+1) - \sum_{j=2}^{N+1} a_j\, y(n-j+1)$ where all values of n < 1 for x(n) and y(n) are 0 such that the input signal is at rest at n < 1. In this instance N represents the order of the numerator of H(z) and M represents the order of the numerator and N represents the order of the denominator of H(z).The IIR filter was designed to calculate y(n) for every input sample, one sample at a time to model real-time FIR filters.

To create a more stable filter instead of the fixed-point IIR filter, several FIR filters were used to generate an output signal. To mimic the IIR filter as closely as possible, a high order FIR filter was used, such that the order of the numerator was 256. Using the Discrete Fourier transform (DFT) and Inverse Discrete Fourier transform (IDFT), the output signal y(n) can be calculated as $y(n) = \sum_{l=-\infty}^{\infty} y(n-lN)$ where N represents the block-size used in the overlap-add method. The transfer function of the FIR filters was dervied from MATLAB functions such that H(z) was a 256th order bandpass filter, with a passband between $0.04\pi$and 0.08π. The first FIR filter used a block size of 512 or 2^9 samples. The second filter would be tested with the MATLAB 1-D digital filter using the same H(z) coefficients as the first FIR filter. The third FIR filter used a block size of 262144 or 2^{18}samples.

Results

A section of input signal can be seen in Figure 1, pertaining to samples 32500 through 42500, to allow for visual clarity. This represents the signal with feedback, corrupted by a high frequency as the signal is observably dense throughout. In comparison with the filtered signal, as shown in Figure 2, of the same section after filtering. It is clear that the corrupting frequency has been attenuated. Listening to the filtered output signal also demonstrates this. By examining the zero-pole relationship and frequency response in Figure 3, the zeros and pole of the transfer function correlate to the order of the numerator and denominator of H(z) as well as displaying the magnitude of all frequencies of the signal, including the attenuation of the corrupting frequency. The zeros were determined to be approximately 0.9808 + j0.1951, 0.9808 - j0.1951, 0.9808 + j0.1951, and 0.9808 - j0.1951.The poles of this function were determined to be approximately 0.9792 + j0.1948, 0.9792 - j0.1948, 0.9777 + j0.1945, and 0.9777 - j0.1945.The corrupting frequency was determined to be approximately 450Hz.

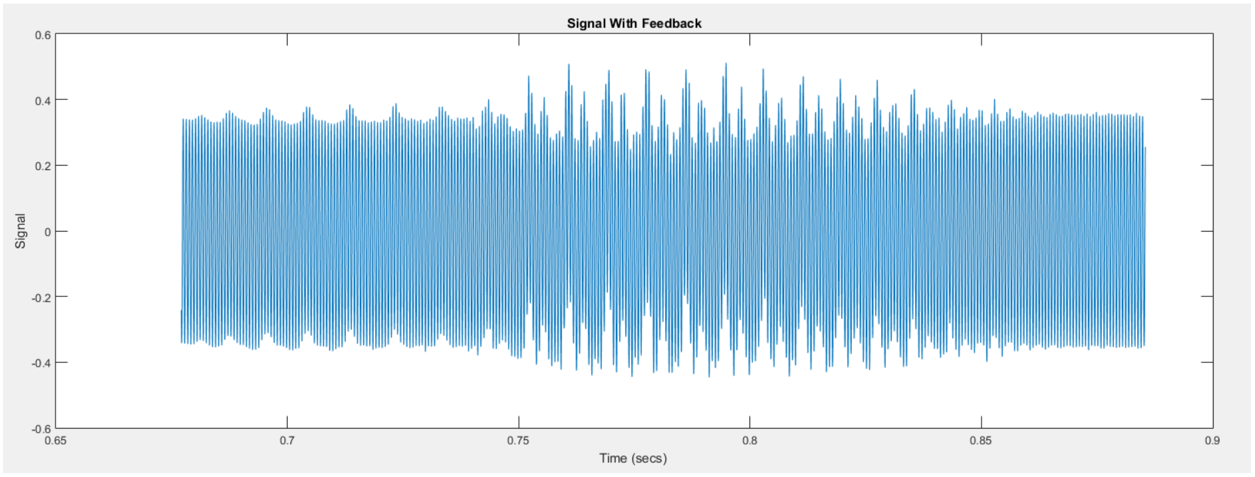

Figure 1: Original Signal with Feedback: Selection shown from ~0.67 secs to ~0.89 secs. Signal input at x(n) where 32500 < n <42500.

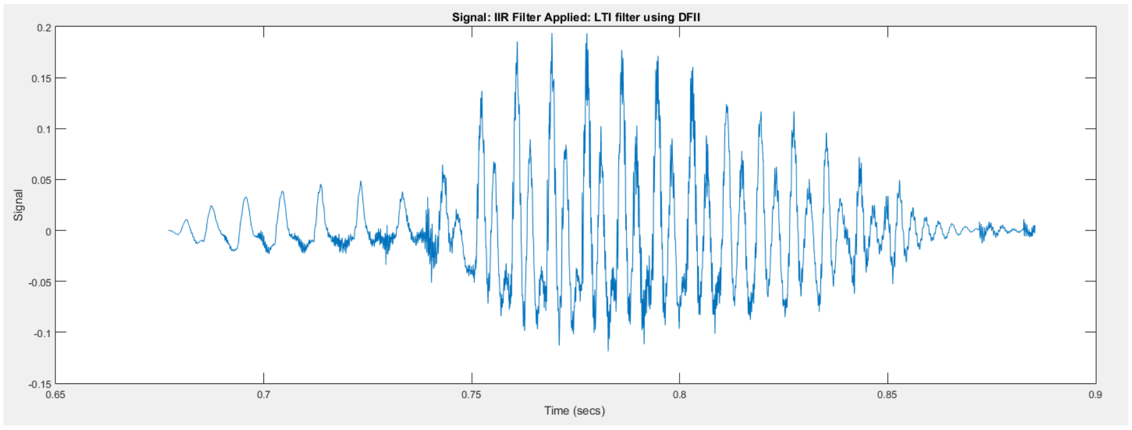

Figure 2: IIR filter applied. Selection shown from ~0.67 secs to ~0.89 secs. Signal input at x(n) where 32500 < n <42500.

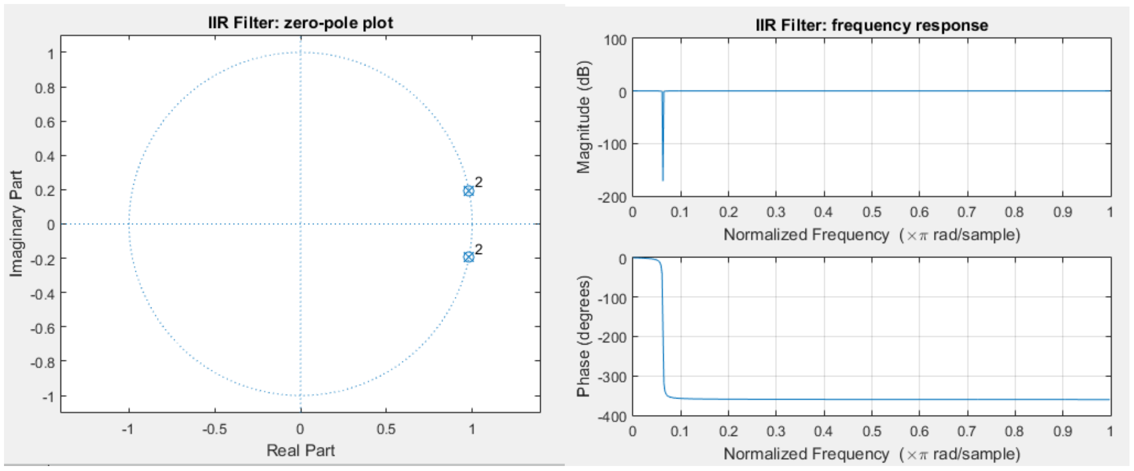

Figure 3: zero-pole plot (left) and frequency response (right) plot of the transfer function used for the IIR filter.

Using the same section of the signal to display as shown in Figure 4, the first FIR filter also minimizes the corrupting frequency. A similar result is shown in Figure 5 and Figure 6, of the 1-D digital filter and the modified block-size filter of the third FIR filter, respectively. Listening to the audio samples produced by each filter shows the differences in each that may not be as visually observable. The FIR filter shown in Figure 4 had audible artifacts and presence of the corrupting frequency demonstrated that this filter had the least attenuation of the approximate 450Hz frequency. The filters shown in Figure 5 and Figure 6 were comparable and very similar visually and audibly. A clear visual difference amongst the three can be seen at the beginning of the signal section shown as the magnitude and polarity of amplitudes in each of these signals. Observing the zero-pole relationship and frequency response of the 256-order transfer function used in these three FIR filters is shown in Figure 7. The poles are located within minute proximity to 0. The zeros are located around the unit circle, both inside and outside the region of convergence (ROC).

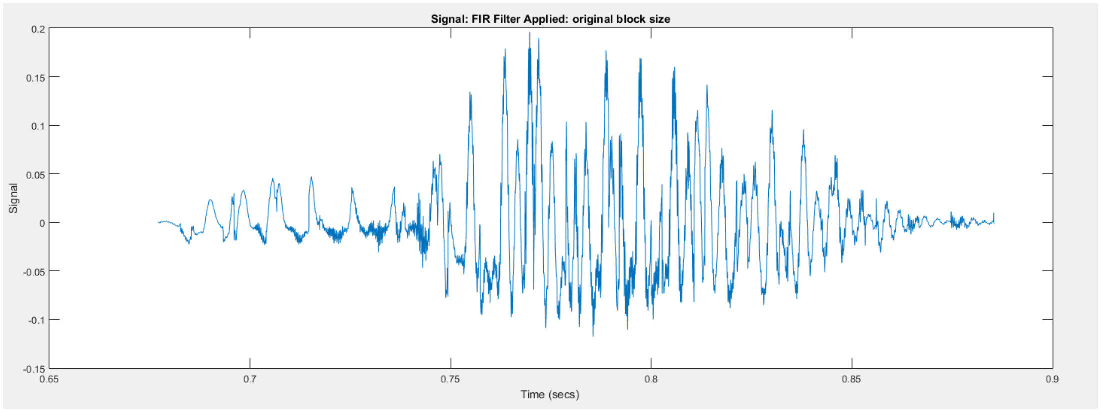

Figure 4: FIR Filter applied: overlap-add method with block size 512 points. Selection shown from ~0.67 secs to ~0.89 secs. Signal input at x(n) where 32500 < n <42500.

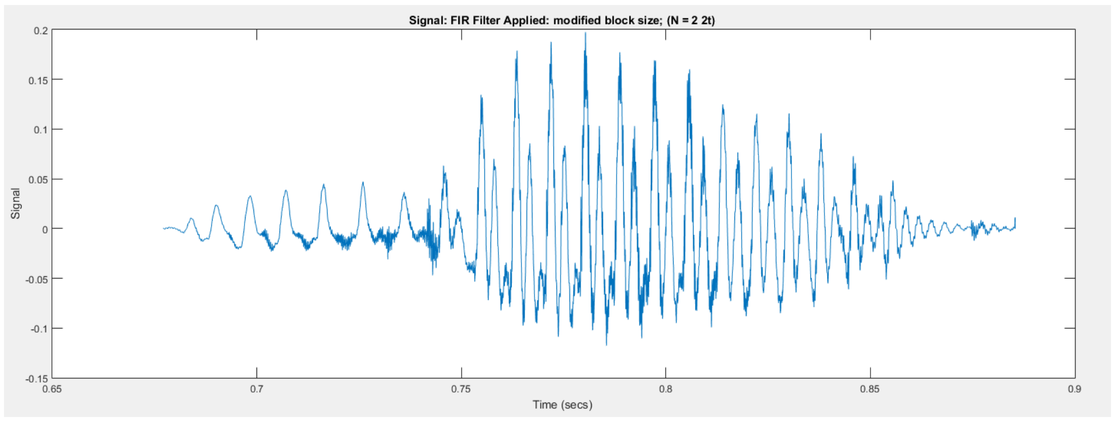

Figure 5: FIR Filter applied: overlap-add method with block size 262144 points. Selection shown from ~0.67 secs to ~0.89 secs. Signal input at x(n) where 32500 < n <42500.

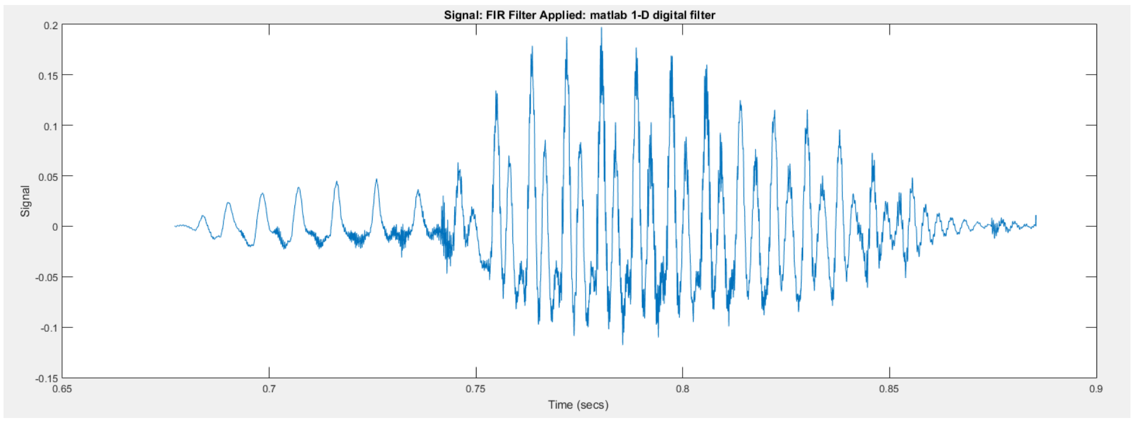

Figure 6: FIR Filter applied: time-domain. Selection shown from ~0.67 secs to ~0.89 secs. Signal input at x(n) where 32500 < n <42500.

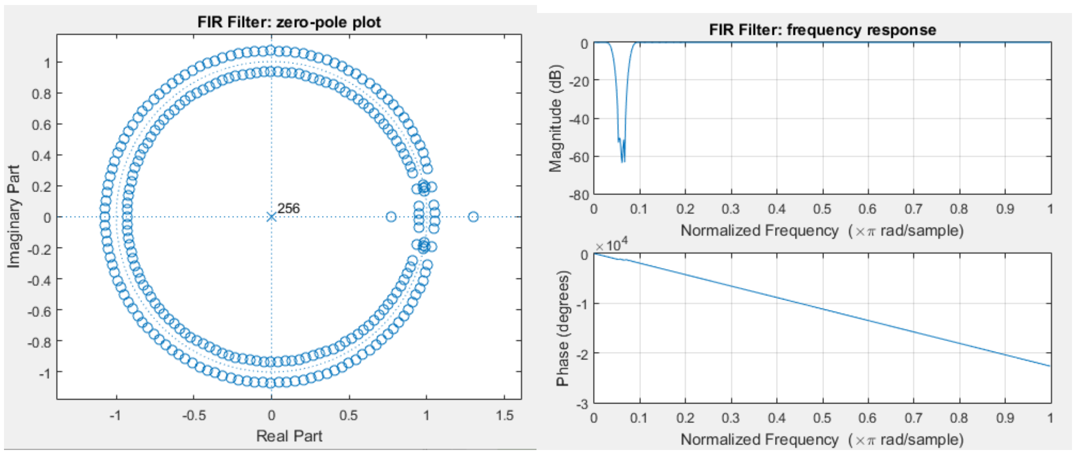

Figure 7: zero-pole plot (left) and frequency response (right) plot of the transfer function used for the FIR filters.

Discussion

The roots identified in the IIR Filter corresponded to the 4 zeros and 4 poles as defined by the order of the numerator and denominator of H(z). This filter is a band-stop filter, as shown in Figure 3 through both the zero-pole plot and magnitude of the frequency response. The frequency that this filter attenuated was approximately 450Hz and frequencies near this value. This was shown by the magnitude plot of the frequency response. At approximately 450Hz the magnitude was most negative, demonstrating attenuation. It did not amplify any frequencies and allowed all other frequencies to pass. The magnitudes at these frequencies are located at 0dB, such that they were not modified. The maximum attenuation of the filter was approximately -180dB. If the filter were to be applied to a discretized sinusoid with analog frequency of 10kHz, the signal would receive no attenuation or gain as the filter allows 10kHz frequencies to pass.

The issues with the first FIR filter with block size of 512 samples is that the block sizes are very small, resulting in artifacts and an undesirable attenuation level. Observation of the DFT/IDFT based output with the time-domain filtered output show the artifacts left in the signal by the DFT/IDFT based output. These can be seen in comparing Figure 4 and Figure 5 towards the beginning of the signal section for most observable difference. The amplitudes of some samples in the DFT/IDFT based output are also lesser than that of some samples in the time-domain filtered output. These artifacts exist because of the conversion from the linear time-domain to the periodic frequency domain. This causes samples in the signal to overlap into adjacent periods of the signal samples. Thus, circular convolution tends to cause these aliasing type artifacts. FIR filters are generally preferred over IIR filters to reduce complications with stability issues and to prevent phase shifts over the frequency response. Comparing the FIR filter to the IIR filter, there are less poles and zeros in the IIR filter than the FIR filter. Additionally, the FIR filter demonstrates a linear phase while an IIR filter results in an exponential phase. The location of the poles in the DFT/IDFT are more stable as they are closer to the Real Axis than the IIR filter and such that the impulse response approaches zero as *n* approaches *infinity*. The DFT/IDFT based implementation was considered instead of the FIR filtering in the time-domain because the DFT/IDFT is more efficient the larger and higher order the transfer function is. The problems with the first FIR filter used on the input signal is that the block sizes were too small, such that because of the periodicity of the frequency domain, it causes more aliasing and artifact production than a larger block size as used in the third FIR filter. It is possible to zero-pad the filter/input signal to prevent these artifacts as less overlap would occur, and aliasing artifacts would be reduced. If the FIR filter were to have a length of 257 samples, then the transfer function would be one order higher, and the block size for the DFT/IDFT based output would be smaller and contain less samples per block. Increasing the block size of N allows for less overlap and less aliasing artifacts to be present in the output signal. However, the larger the block size, the larger the computational time complexity of calculating the output becomes. The overlap-add method used in the DFT/IDFT based filtering methods truncate or section the input signal in to equal sized blocks, such that the convolution with the input signal and transfer function becomes more efficient to determine. The time-invariance of the system allows the time-shift that represents each block to be combined in the final determination of the output signal. The block-size used in the is project was 262144 samples per block.

Appendix:

A.

Printout of whole Filtered Signal through IIR filter.

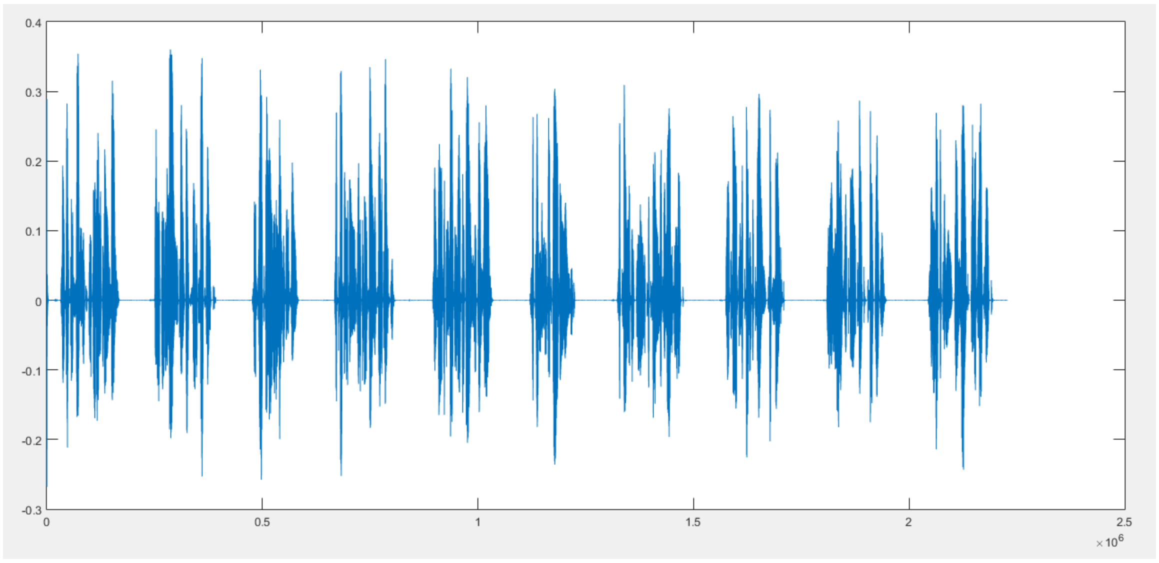

Figure 8. FilteredSignal_Part1.m

B.

MATLAB Code:

iirFilt.m

```
function y_n = iirFilt(x_n)

%Marcus Favila
%ECE 437 Computer Project
%07 Dec 2019

% 'clear iifFilt' should be entered in Command Window before running
% Project_Part1.m
% if not cleared, peristent variables will cause calculations to be
% inaccurate resulting in an unrepresentative output signal for the
% filter

global a b M N wmax
%These values are used to store the previous values for the calculation
%of y(n)
persistent w;
persistent X;
persistent Y;
persistent count;
%addend refers to the term inside the summation at some k or j
B_addend = 0;
A_addend = 0;
%term refers to the sum of all terms in the summation for range of k or j
B_term = 0;
A_term = 0;
```

```
%count is analogous to n in y(n) or x(n)
%count is set to 1 on first call of function: persistent variables should
%be cleared with every run of a script that calls the iirFilt function
if isempty(count)
       count = 1;
end

if isempty(w)
       w=zeros(1,N+1);
end
if isempty(X)
       X=zeros(1,N+1);
end
if isempty(Y)
       Y=zeros(1,N+1);
end
X(count) = x_n;

%Calculation of the x(n) summation
for k = 1:M+1
       if (count-k) < 1
       B_addend = 0;
       else
       B_addend = b(k)*X(count-k+1);
       end
       B_term = B_term + B_addend;
end
%disp(B_term)
%Calculation of the y(n) summation
for j = 2:N+1
       if (count-j) < 1
       A_addend = 0;
       else
       A_addend = a(j)*Y(count-j+1);
       end
       A_term = A_term + A_addend;
end
%disp(A_term)

%calulates the final y(n) value
y_n = B_term - A_term;

Y(count) = y_n;
%disp(count)
count = count + 1;
%pause(2);
End
```

Project_Part1.m (Unmodified)

```
%Project - part 1 : DO NOT MODIFY
%clear variables
clear

global a b M N

M = 4; %order of numerator
N = 4; %order of denominator

%filter coefficients
b= [0.995304813340108, -3.904721241753850, 5.820302744670216, -3.904721241753850,
0.995304813340108];
a =[1.000000000000000, -3.913921547130277, 5.820292921365054, -3.895520936377423,
0.990619449985378];

%read input audio
[XpF,Fs] = audioread('Signal_plus_feedback.wav');

%plot input audio
figure(1);
plot(XpF);
%play input audio
%sound(XpF,Fs);

%initialize output to all zeros
%y=zeros(1,length(XpF));

for i=1:length(XpF),
       %read input signal sample-by-sample
       x_n = XpF(i);
       % call the iirFilt function to produce the output sample
       y(i)=iirFilt(x_n);
end

%plot the output audio
figure(2);
plot(y);

%play the output audio
%sound(y,Fs);

%save the output audio to wave-file
audiowrite('FilteredSignal_Part1.wav',y,Fs);
```

Project_Part1_Questions.m

```
%Marcus Favila
%ECE 437 Computer Project
%07 Dec 2019

%Project Part 1: Questions

global a b M N wmax
[XpF,Fs] = audioread('FilteredSignal_Part1.wav');
for i = 1:length(XpF)
       time(i) = i/Fs;
end

figure('Name','IIR Filter: frequency response','NumberTitle','off');
freqz(b,a)
title('IIR Filter: frequency response')

figure('Name','IIR Filter: frequency response: logarithmic scale','NumberTitle','off');
[h,w] = freqz(b,a,length(XpF),'whole',Fs);
semilogx(w/pi,20*log10(abs(h)));
ax = gca;
ax.YLim = [-180, 20];
ax.XLim = [0,200000];
title('IIR Filter: frequency response')
xlabel('Frequency (Hz)')
ylabel('Magnitude (dB)')

figure('Name','IIR Filter: zero-pole plot','NumberTitle','off');
[vz,vp]=zplane(b,a);
title('IIR Filter: zero-pole plot')
zeros = vz
poles = vp
```

Project_Part2.m

```
%Marcus Favila
%ECE 437 Computer Project
%07 Dec 2019
%PART2

% clear all variables
clear
clear iirFilt;
time = [];
xmin  = 32500;
xmax = 42500;

%read audio file
[XpF,Fs] = audioread('Signal_plus_feedback.wav');
for i = 1:length(XpF)
       time(i) = i/Fs;
end

%plot audio file
```

```
figure('Name','Signal with Feedback','NumberTitle','off');
plot(time(1,xmin:xmax),XpF(xmin:xmax,1));
title('Signal With Feedback')
xlabel('Time (secs) ')
ylabel('Signal')

%play audio
%sound(XpF,Fs);
%________________________________________________________________________

%First FIR Filter: DFT/IDFT values provided

L=256;
%design FIR filter of order L (i.e. length L+1)
b_fir=fir1(L,[0.04 0.08],'stop');

%use block size N that is the next power of 2 compared to size of b_fir
% e.g. 256, 512, 1024, 2048 ...
t = nextpow2(length(b_fir));
N = 2^t;

% make FIR length equal to block-size, i.e.
h=zeros(N,1);
%copy coefficients
h(1:length(b_fir))=b_fir;
H=fft(h); % H is N-point fft

num_Blocks = length(XpF)/N;
%initialize output vector
y=[];
for block = 1:num_Blocks,
       %get block of input audio data
       % (NOTE: in a real-time implementation, you would get only 1
       % block of data at a time to work with)
       x = XpF((block-1)*N+1:block*N);

       X=fft(x);
       % multiply N-point ffts
       Y=X.*H;

       %get the N-point ifft of the output
       y_fft=ifft(Y);

       % append block to the output
       y=[y;y_fft];
       block
end

%plot final output
figure('Name','Signal: FIR Filter Applied: {original block size}','NumberTitle','off');
plot(time(1,xmin:xmax),y(xmin:xmax,1));
title('Signal: FIR Filter Applied: {original block size}')
xlabel('Time (secs) ')
```

```
ylabel('Signal')
%play final output
%sound(y,Fs);

%write the output to a wave file
audiowrite('FilteredSignal_Part2a_DFT_IDFT_originalblock.wav',y,Fs);

%__________________________________________________________________________

%Second Filter: time-domain filtered output
y2=[];
b2 = b_fir;
a2=[1];
y2 = filter(b2,a2,XpF);

%plot the output audio
figure('Name','Signal: FIR Filter Applied: {matlab 1-D digital filter}','NumberTitle','off');
plot(time(1,xmin:xmax),y2(xmin:xmax));
title('Signal: FIR Filter Applied: {matlab 1-D digital filter}')
xlabel('Time (secs) ')
ylabel('Signal')
%play the output audio
%sound(y2,Fs);

%save the output audio to wave-file
audiowrite('FilteredSignal_Part2b_FIR_filter_timedomain.wav',y2,Fs);


%___________________________________________________________________________

%Third Filter: Modified N: DFT/IDFT based filtering
%Chosen value of N: 262144 or 2^18

%use block size N that is the next power of 2 compared to size of b_fir
% e.g. 256, 512, 1024, 2048 ...
t = nextpow2(length(b_fir));
N3 = 2^(2*t);

% make FIR length equal to block-size, i.e.
h=zeros(N3,1);
%copy coefficients
h(1:length(b_fir))=b_fir;
H=fft(h); % H is N-point fft

num_Blocks3 = length(XpF)/N3;
%initialize output vector
y3=[];
for block3 = 1:num_Blocks3,
       %get block of input audio data
       % (NOTE: in a real-time implementation, you would get only 1
       % block of data at a time to work with)
       x = XpF((block3-1)*N3+1:block3*N3);
```

```
        X=fft(x);
        % multiply N-point ffts
        Y=X.*H;

        %get the N-point ifft of the output
        y_fft=ifft(Y);

        % append block to the output
        y3=[y3;y_fft];
        block3
end

%plot final output
figure('Name','Signal: FIR Filter Applied: {modified block size; (N = 2 2t)
}','NumberTitle','off');
plot(time(1,xmin:xmax),y(xmin:xmax,1));
title('Signal: FIR Filter Applied: {modified block size; (N = 2 2t) }')
xlabel('Time (secs) ')
ylabel('Signal')
%play final output
%sound(y,Fs);

%write the output to a wave file
audiowrite('FilteredSignal_Part2c_DFT_IDFT_modifiedblock.wav',y,Fs);
%________________________________________________________________________
%{

% Altering the size of N such that the number of blocks used results in
% memory errors as array becomes too large for program to compute
% This section is omitted to reflect that

%use block size N that is the next power of 2 compared to size of b_fir
% e.g. 256, 512, 1024, 2048 ...
t = nextpow2(length(b_fir));
N3 = 2^(3*t);

% make FIR length equal to block-size, i.e.
h=zeros(N3,1);
%copy coefficients
h(1:length(b_fir))=b_fir;
H=fft(h); % H is N-point fft

num_Blocks3 = length(XpF)/N3;
%initialize output vector
y3=[];
for block3 = 1:num_Blocks3,
        %get block of input audio data
        % (NOTE: in a real-time implementation, you would get only 1
        % block of data at a time to work with)
        x3 = XpF((block3-1)*N3+1:block3*N3);

        X=fft(x3);
        % multiply N-point ffts
        Y=X.*H;

        %get the N-point ifft of the output
        y_fft=ifft(Y);

        % append block to the output
        y3=[y;y_fft];
        block3
end

%plot final output
figure(5);
plot(y3(xmin:xmax,1));
title('modified block size overlap-add (N = 2 3t) Method')
%play final output
%sound(y,Fs);

%write the output to a wave file
audiowrite('FilteredSignal_Part2e.wav',y,Fs);
%}
%___________________________________________________________________

%Same function as Part1
%IIR Filter for comparison with FIR filters
for i=1:length(XpF),
        %read input signal sample-by-sample
        x_n = XpF(i);
        % call the iirFilt function to produce the output sample
        y_iir(i)=iirFilt(x_n);
end

%plot the output audio
figure('Name','Signal: IIR Filter Applied: LTI filter using DFII','NumberTitle','off');
plot(time(1,xmin:xmax),y_iir(1,xmin:xmax));
title('Signal: IIR Filter Applied: LTI filter using DFII');
xlabel('Time (secs) ')
ylabel('Signal')
audiowrite('FilteredSignal_Part2d_IIR_Filter.wav',y,Fs);
```

Project_Part2_Questions.m

```
%Marcus Favila
%ECE 437 Computer Project
%07 Dec 2019

%Project Part 2: Questions
L = 256;
b_fir=fir1(L,[0.04 0.08],'stop');
a_fir=[1];

[XpF,Fs] = audioread('FilteredSignal_Part1.wav');
for i = 1:length(XpF)
       time(i) = i/Fs;
end

figure('Name','FIR Filter: frequency response','NumberTitle','off');
freqz(b_fir);
title('FIR Filter: frequency response')

figure('Name','FIR Filter: frequency response: logarithmic scale','NumberTitle','off');
[h,w] = freqz(b_fir,a_fir,length(XpF),'whole',262144);
semilogx(w/pi,20*log10(abs(h)));
ax = gca;
ax.YLim = [-180, 20];
ax.XLim = [0,20000];
title('FIR Filter: frequency response')
xlabel('Frequency (Hz)')
ylabel('Magnitude (dB)')

figure('Name','FIR Filter: zero-pole plot','NumberTitle','off');
[vz,vp]=zplane(b_fir,a_fir);
title('FIR Filter: zero-pole plot')
zeros = vz
poles = vp
```