Application of FIR and IIR Notch Filter to Filter Out An Echo
LAB-2
ECE161B

Samuel Liimatainen

March 29, 2022

Date Performed:	2/22/22 to 3/3/22
Lab TA/'s:	Yiqian Wang
	Prasad Kamath
Instructor:	Professor Truong Nguyen

1 Introduction

The purpose of this lab is to take practice the application of FIR and IIR Notch Filters to filter out and echo in a sound file. An echo was added to an existing sound file, this new signal with an echo will be used to test the results of filtering. A 14th order FIR filter and 2nd order IIR filter were designed and inspected both in MatLab and in C language. The signal with echo was passed through the two filters and the resulting signal was inspected to see if the echo had been removed. It was found that the FIR filter was significantly less effective than the IIR filter.

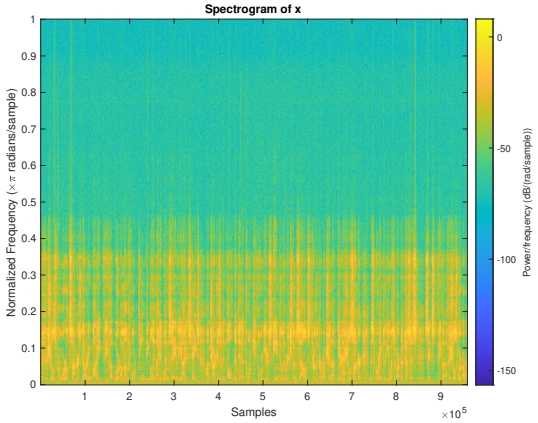
2 Task 1

In this section an echo is added to the speech.wav file. We can hear the added echo in the speech.wav file.

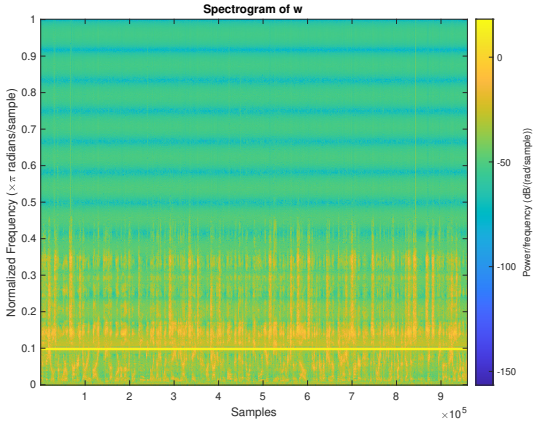
3 Task 2

x = original speech.wav signal.
w = signal with echo in it.
The difference between the two signals is w has a hot bar at 0.1 Normalized Frequency. In the spectrogram of X we can see the signal before the sinusoid was added. In the spectrogram of w we can see the results of adding a sinusoid to the x signal. The 0.1 Normalized Frequency corresponds to 2.4KHz which is the frequency the sinusoid is at. This results in a delta spike at which we see in the spectrogram of w at 2.4KHz.

Spectrogram of speech.wav without echo



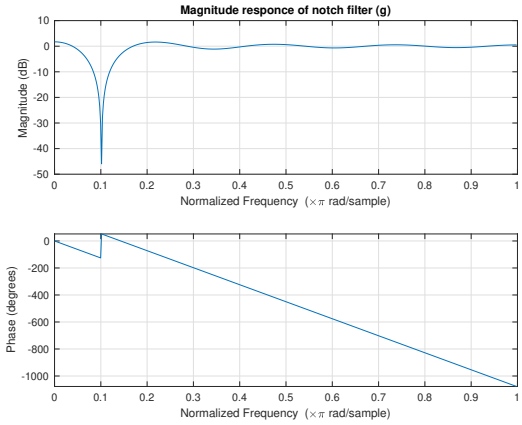
Spectrogram of speech.wav with echo

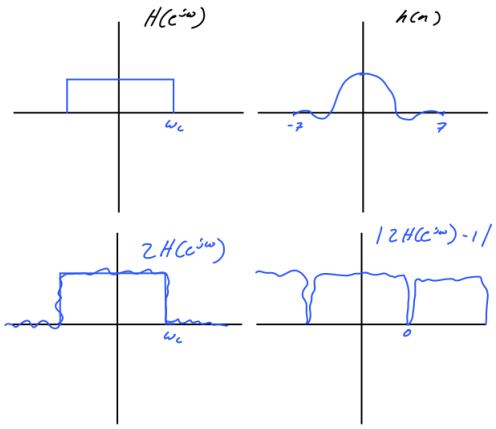


4 Task 3

Description of FIR Notch Filter design approach:
The derivation of the notch filter starts with the idealized filter with cutoff frequency wc in the frequency domain. This results in a sinc function in the time domain from n=-7:7. We can then scale the non-idealized filter in the frequency domain by two, we see ripples after the pass-band. If we absolute value the filter and subtract one we get the desired notch filter. From here we can derive the time domain equation for the notch filter.

Magnitude Responce of FIR filter

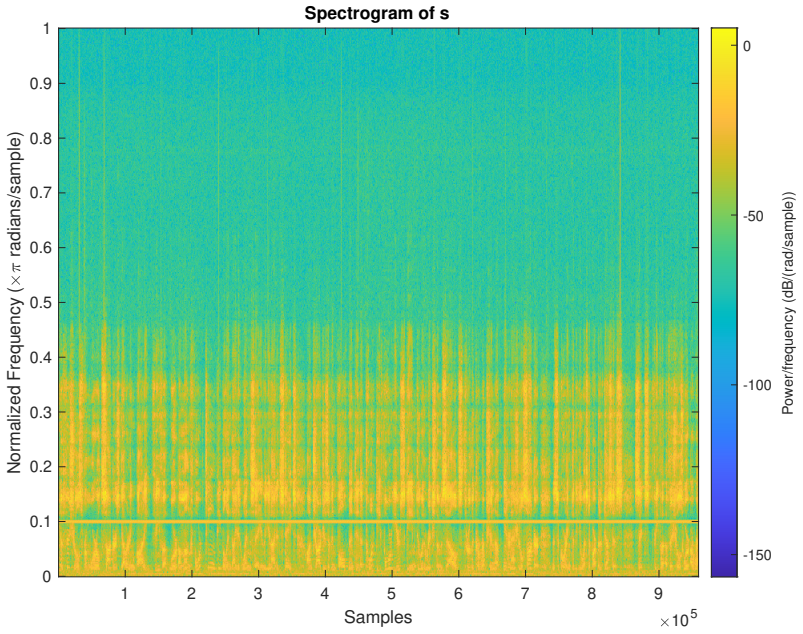




5 Task 4

Here it can be seen that the FIR filter attenuated the signal at 0.1 Normalized frequency (i.e. 2.4KHz frequency). The difference between the filtered and non-filtered signal is the hot bar brightness, which indicates the signal was attenuated here.

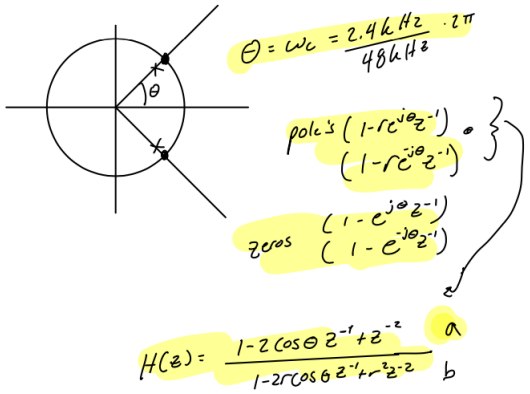
Spectrogram of target signal(s)



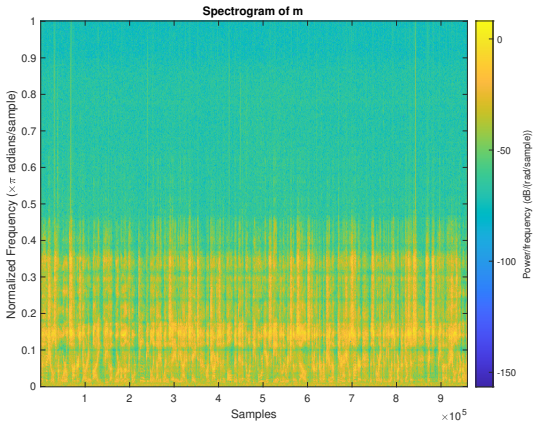
6 Task 5

Description of FIR Notch Filter design approach:
The IIR filter coefficients can be found by first placing the poles and zeros of the filter and then finding the transfer function H(z). From here we can extract the needed coefficients for the 2nd order IIR filter.
Once the echo signal has been filtered by the IIR filter, it can be heard in the resulting signal that the echo has been fully removed. We can also see these results in the spectrogram where the hot bar has been fully attenuated. The spectrogram shows the delta spike at 2.kHz(or 0.1 Normalized Frequency) has been greatly attenuated.

Filter Design Approach



Spectrogram of the target signal(m)



7 Task 6

```
------------------------------TASK 6 C-CODE:-------------------------------------------
#include <stdlib.h>
#include <stdio.h>
#include <float.h>
#include <sndfile.h>
#include <math.h>

#define PI 3.14159265

void iir_filter(float* x,float* a, float* b,float* y);
void fir_filter(float* x,float* h,float* y);


int main(int argc, char *argv[])
{
int ii;

//Require 2 arguments: input file and output file
if(argc < 2)
{
printf("Not enough arguments \n");
return -1;
}

SF_INFO sndInfo;
SNDFILE *sndFile = sf_open(argv[1], SFM_READ, &sndInfo);
if (sndFile == NULL) {
fprintf(stderr, "Error reading source file '%s': %s\n", argv[1], sf_strerror(sndFile));
return 1;
}

SF_INFO sndInfoOut = sndInfo;
sndInfoOut.format = SF_FORMAT_WAV | SF_FORMAT_PCM_16;
sndInfoOut.channels = 1;
sndInfoOut.samplerate = sndInfo.samplerate;
SNDFILE *sndFileOut = sf_open(argv[2], SFM_WRITE, &sndInfoOut);//audi with echo

// Check format - 16bit PCM
if (sndInfo.format != (SF_FORMAT_WAV | SF_FORMAT_PCM_16)) {
fprintf(stderr, "Input should be 16bit Wav\n");
sf_close(sndFile);
return 1;
}
```

```
// Check channels - mono
if (sndInfo.channels != 1) {
fprintf(stderr, "Wrong number of channels\n");
sf_close(sndFile);
return 1;
}

// Your implementation
    // Load data
    float x[960000];//input array
    float y[960000];//output array
    int Ne = 0.5*48000;
    for(ii=0; ii < sndInfo.frames; ii++)
    {
        sf_readf_float(sndFile, &x[ii], 1);

        if(ii - Ne > 0)
        {
            y[ii] = 0.7*x[ii]+0.3*x[ii - Ne];
        }
        else
        {
            y[ii] = 0.7*x[ii];
        }

        //Do something to the variable buffer here

        sf_writef_float(sndFileOut, &y[ii], 1);
    }

sf_close(sndFile);

sf_write_sync(sndFileOut);//sound with echo
sf_close(sndFileOut);

return 1;
}
```

---

8 Task 7

```
------------------------------TASK 7 C-CODE:-------------------------------------------
#include <stdlib.h>
#include <stdio.h>
#include <float.h>
#include <sndfile.h>
#include <math.h>

#define PI 3.14159265


void fir_filter(float* w,float* h,float* y);


int main(int argc, char *argv[])
{
int ii;

//Require 2 arguments: input file and output file
if(argc < 2)
{
printf("Not enough arguments \n");
return -1;
}

SF_INFO sndInfo;
SNDFILE *sndFile = sf_open(argv[1], SFM_READ, &sndInfo);
if (sndFile == NULL) {
fprintf(stderr, "Error reading source file '%s': %s\n", argv[1], sf_strerror(sndFile));
return 1;
}

SF_INFO sndInfoOut = sndInfo;
sndInfoOut.format = SF_FORMAT_WAV | SF_FORMAT_PCM_16;
sndInfoOut.channels = 1;
sndInfoOut.samplerate = sndInfo.samplerate;
SNDFILE *sndFileOut = sf_open(argv[2], SFM_WRITE, &sndInfoOut);//audi with echo

// Check format - 16bit PCM
if (sndInfo.format != (SF_FORMAT_WAV | SF_FORMAT_PCM_16)) {
fprintf(stderr, "Input should be 16bit Wav\n");
sf_close(sndFile);
return 1;
}
```

```
// Check channels - mono
if (sndInfo.channels != 1) {
fprintf(stderr, "Wrong number of channels\n");
sf_close(sndFile);
return 1;
}

// Your implementation
    float w[960000];//input array - signal to be filtered
    float y[960000];//output array - output of filter
    float g[15];
    // make the time domain notch filter
    float wc = (2400/48000)*2*3.14;
    int n[] = {-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7};
    for(int i = 0; i <=14; i++)
    {
        if(n[i] != 0)
        {
            g[i]=2*(sin(wc*n[i])/(3.14*n[i]));
        }
        else
        {
            g[i] = (2*wc/3.14)-1;
        }
    }

    for(ii=0; ii < sndInfo.frames; ii++)
    {
        sf_readf_float(sndFile, &w[ii], 1);
    }

    fir_filter(w,g,y);

    for(ii=0; ii < sndInfo.frames; ii++)
    {
        sf_writef_float(sndFileOut, &y[ii], 1);
    }

sf_close(sndFile);

sf_write_sync(sndFileOut);//sound with echo
sf_close(sndFileOut);



return 1;
```

```
}


void fir_filter(float* w,float* h,float* y) {
    for(int n = 14; n < 960000; n++)
    {
        for(int j = 0; j < 15; j++)
        {
            y[n] = y[n] + h[j]*w[n-j];
        }
    }
}
```

---

9 Task 8

```
------------------------------TASK 8 C-CODE:-------------------------------------------
#include <stdlib.h>
#include <stdio.h>
#include <float.h>
#include <sndfile.h>
#include <math.h>

#define PI 3.14159265

void iir_filter(float* x,float* a, float* b,float* y);


int main(int argc, char *argv[])
{
int ii;

//Require 2 arguments: input file and output file
if(argc < 2)
{
printf("Not enough arguments \n");
return -1;
}

SF_INFO sndInfo;
SNDFILE *sndFile = sf_open(argv[1], SFM_READ, &sndInfo);
if (sndFile == NULL) {
fprintf(stderr, "Error reading source file '%s': %s\n", argv[1], sf_strerror(sndFile));
return 1;
}

SF_INFO sndInfoOut = sndInfo;
sndInfoOut.format = SF_FORMAT_WAV | SF_FORMAT_PCM_16;
sndInfoOut.channels = 1;
sndInfoOut.samplerate = sndInfo.samplerate;
SNDFILE *sndFileOut = sf_open(argv[2], SFM_WRITE, &sndInfoOut);//audi with echo

// Check format - 16bit PCM
if (sndInfo.format != (SF_FORMAT_WAV | SF_FORMAT_PCM_16)) {
fprintf(stderr, "Input should be 16bit Wav\n");
sf_close(sndFile);
return 1;
}

// Check channels - mono
```

```
if (sndInfo.channels != 1) {
fprintf(stderr, "Wrong number of channels\n");
sf_close(sndFile);
return 1;
}

// Your implementation
    float theta = 0.05*2*3.14;
    float r = 0.98;
    float a[] = {1,-2*cos(theta),1};
    float b[] = {1,-2*r*cos(theta),r*r};
    float w[960000];//input array - signal to be filtered
    float y[960000];//output array - output of filter

    for(ii=0; ii < sndInfo.frames; ii++)
    {
        sf_readf_float(sndFile, &w[ii], 1);
        //Do something to the variable buffer here
        //sf_writef_float(sndFileOut, &y[ii], 1);
    }

    iir_filter(w,a,b,y);

    for(ii=0; ii < sndInfo.frames; ii++)
    {
        //sf_readf_float(sndFile, &y[ii], 1);
        //Do something to the variable buffer here
        sf_writef_float(sndFileOut, &y[ii], 1);
    }

sf_close(sndFile);

sf_write_sync(sndFileOut);//sound with echo
sf_close(sndFileOut);

return 1;
}

void iir_filter(float* x,float* a, float* b,float* y) {
    y[0] = (a[0]*x[1])/b[0];
    y[1] = (a[0]*x[2] + a[1]*x[2-1] - b[1]*y[2-1])/b[0];
    for(int n = 2; n < 960000; n++)
    {
        y[n] = (a[0]*x[n] + a[1]*x[n-1] + a[2]*x[n-2] - b[1]*y[n-1] - b[2]*y[n-2]) / b[0];
    }
}
```

---

10 Appendix

```
------------------------------TASK 1 MATLAB CODE:-----------------------------
Fs = 48000;
[x,Fs] = audioread("speech.wav");

%Here I normalize the x signal
maxValue = max(abs(x));
x = x/maxValue;

Ne = 0.5*Fs; %% this is our echo delay
y = 0.7.*x;
y(Ne:end) = y(Ne:end) + 0.3.*x(1:(end-Ne+1));
sound(y,Fs)
------------------------------------------------------------------------------

------------------------------TASK 2 MATLAB CODE:-----------------------------
f1 = 2400;
L = length(x);
v = generate_sigs(f1,Fs,L);
v = v';
w = x + v;

%Spectrogram of x
window = 1000;
spectrogram(x,window,(0.25*window),'yaxis')
title("Spectrogram of x")
print -depsc spectrogramTask2x

%Spectrogram of w
window = 1000;
spectrogram(w,window,(0.25*window),'yaxis')
title("Spectrogram of w")
print -depsc spectrogramTask2w
------------------------------------------------------------------------------

------------------------TASK3 MATLAB CODE:------------------------------------
%Calculate the cuttoff frequency
wc = (2400/48000)*2*pi;
%Create number of samples for time domain filter (h)
%we want a 14 order filter so n = -7:7
n = -7:7;
h = sin(wc.*n)./(pi.*n);
% at n = 0 we get a NaN bc dived by zero so we need to hard code this
h(8) = wc/pi;
% no we manipulate h to make a notch filter
g = 2*h;
```

```
%no subtract the delta to make the notch
g(8)= g(8)-1;
%Look at teh magnitude resp of g, the denom is 1 bc its FIR filter
freqz(g,1)
title("Magnitude responce of notch filter (g)")
print -depsc freqzTask3
--------------------------------------------------------------------------------

------------------------------TASK 4 MATLAB CODE-------------------------------
s = fir_filter(w,g);
%Spectrogram of s
window = 1000;
spectrogram(s,window,(0.25*window),'yaxis')
title("Spectrogram of s")
print -depsc spectrogramTask4s
--------------------------------------------------------------------------------

------------------------------TASK 5 MATLAB CODE-------------------------------
theta = 0.05*2*pi;
r = 0.98;
a = [1,-2*cos(theta),1];
b = [1,-2*r*cos(theta),r*r];
m = iir_filter(w,a,b);
sound(m,Fs);
%Spectrogram of m
window = 1000;
spectrogram(m,window,(0.25*window),'yaxis')
title("Spectrogram of m")
print -depsc spectrogramTask5m
--------------------------------------------------------------------------------

------------------MATLAB FUNCTION CODE: fir_filter(w,h)-----------------------
function fir = fir_filter(w,h)
y = zeros(length(w),1);

    for n = (15):(length(w))
        for i = 1:(15)
            y(n) = y(n) + h(i)*w(n-i+1);
        end
    end
    fir = y;
end
--------------------------------------------------------------------------------

------------------MATLAB FUNCTION CODE: iir_filter(x,a,b)-----------------------
function filtered = iir_filter(x,a,b)
    y = [];
```

```
    %calc the first 2 terms
    y(1) = (a(1)*x(1))/b(1);
    y(2) = (a(1)*x(2) + a(2)*x(2-1) - b(2)*y(2-1))/b(1);

    for n = (3):(length(x))
        y(n) = (a(1)*x(n) + a(2)*x(n-1) + a(3)*x(n-2) - b(2)*y(n-1)
               - b(3)*y(n-2))/ b(1);
    end
    filtered = y;
end
--------------------------------------------------------------------------------

--------------MATLAB FUNCTION CODE: generate_sigs(f,Fs,L)-----------------------
function x = generate_sigs(f,Fs,L)
    t = 0:1/Fs:L/Fs - 1/Fs ;
    x = cos(2*pi*f*t);
end
--------------------------------------------------------------------------------
```