Louis Rosenblum
EEE509 - ASU
06/16/2020

Project 2 Report

The Frequency Domain Adaptive Filter:

An adaptive noise-cancelling filter was developed for this project. This filter adapts its coefficients in the frequency domain based on data collected from the filter input. One main advantage to this method over time-domain adaptive filters is that the low computational complexity of the FFT can be exploited. In addition to this high order filters (long delays) aren't always necessary for FDAF.

(2006) Frequency-Domain Adaptive Filters. In: Acoustic MIMO Signal Processing. Signals and Communication Technology. Springer, Berlin, Heidelberg

Block Adaptive Filters and Frequency Domain Adaptive Filters . Tampere University of Technology, CS. http://www.cs.tut.fi/~tabus/course/ASP/SGN2206LectureNew6.pdf

The signal shown below was used as the input signal.
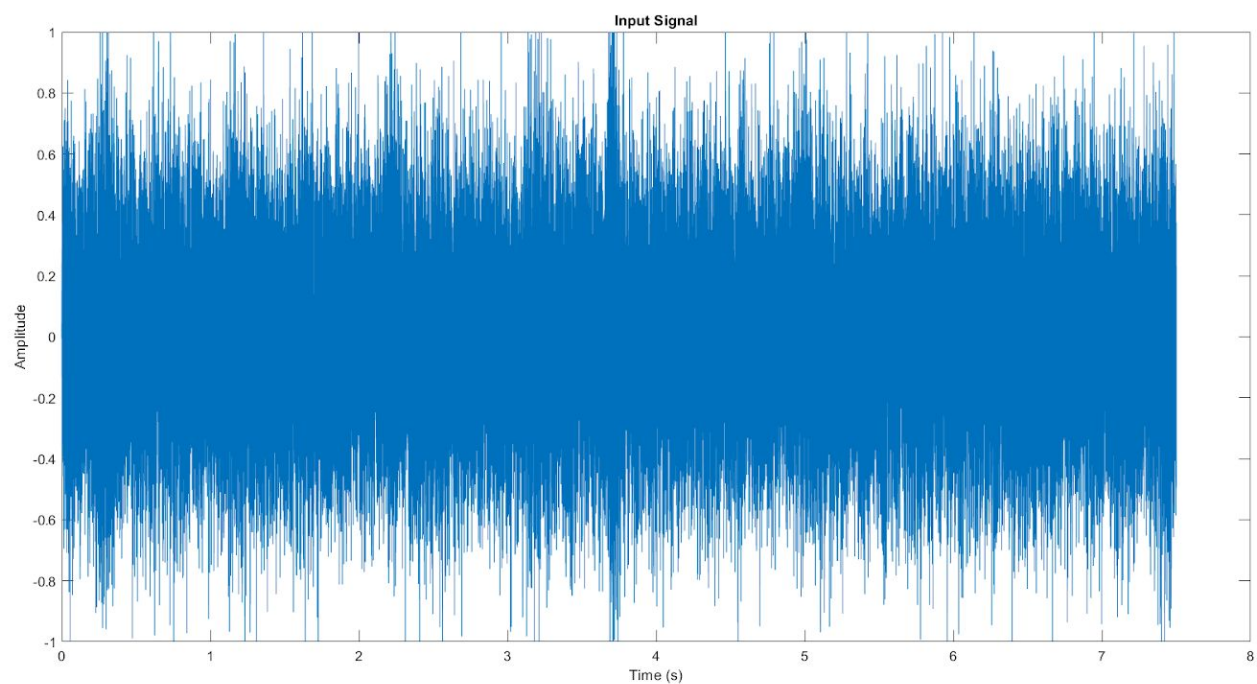
The signal to noise ratio of the input signal before filtering is measured at -21.13.

```
snr1 =

    -21.1297

fx >> |
```
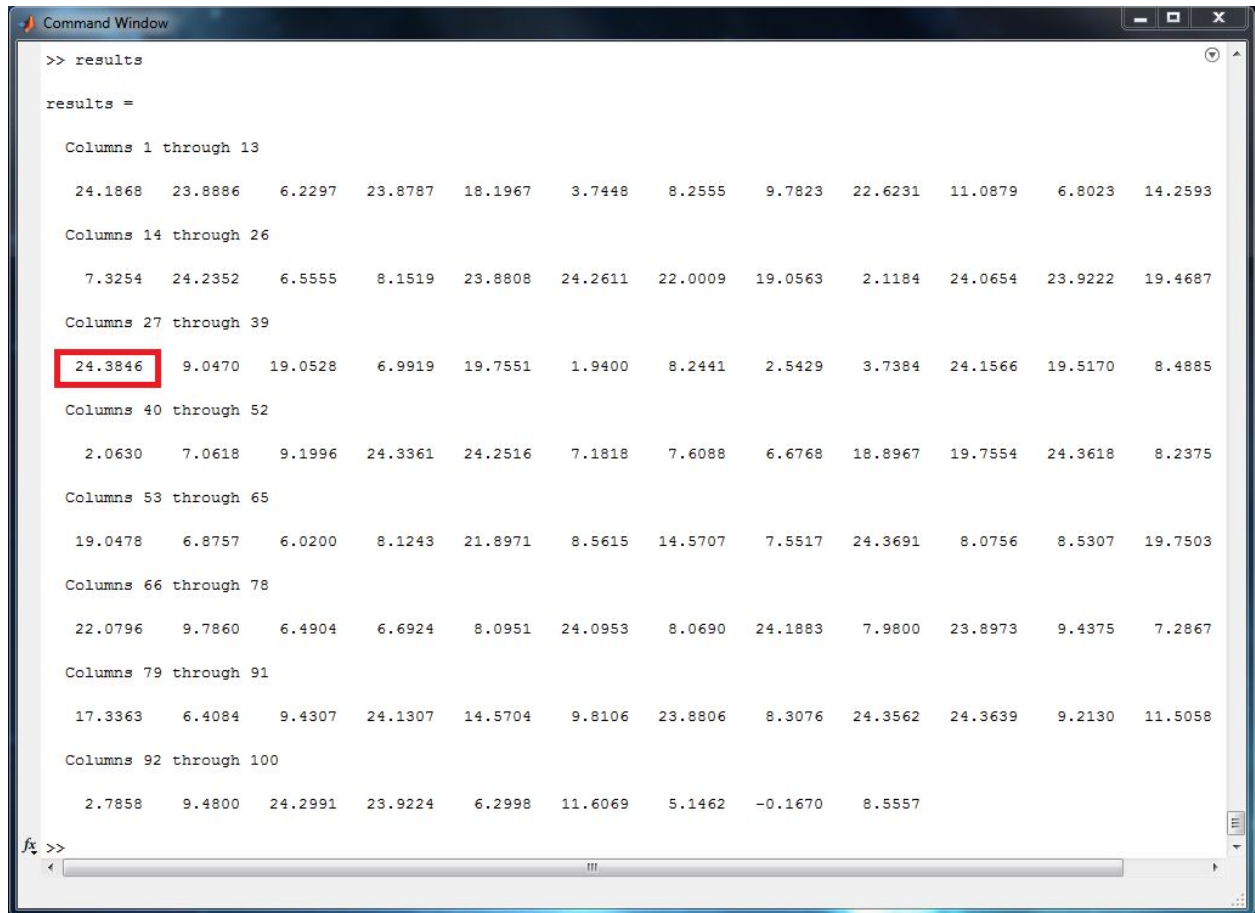
MATLAB Program:

(See appendix for full code)

A MATLAB program was developed to simulate the adaptive digital filter. The program contains an initialization state, file I/O, signal deconstruction into frames, parallel processing, the filter instantiation, and plots/results.
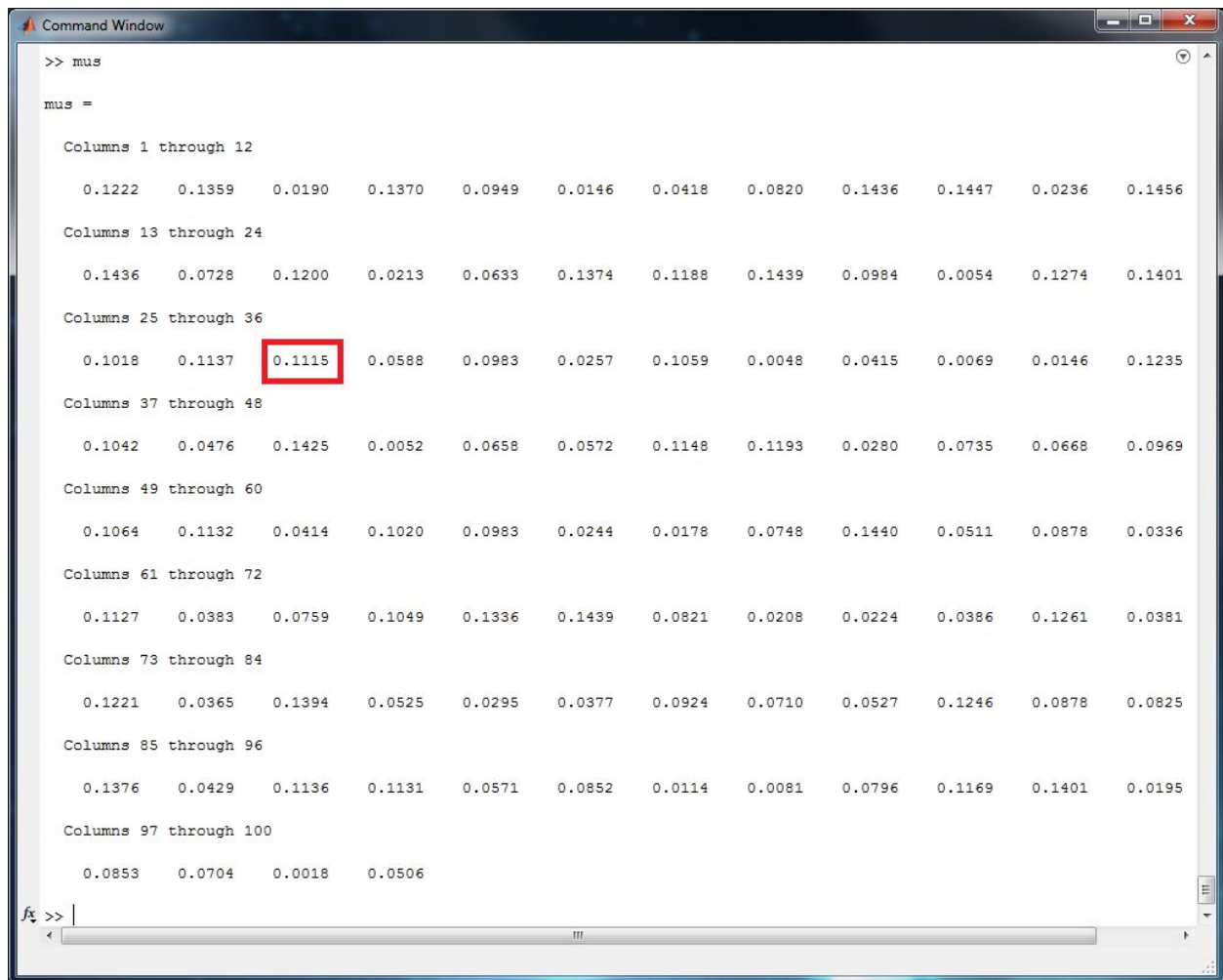
Another script was also created in order to determine the optimal mu value for each N-point FFT.

Determining mu:

The optimal mu was determined used MATLAB's random number generator with an upper limit of mu = 0.15. Values reaching not far above 0.15 would cause instability in the filter. 100 different mus between 0 and 0.15 were tested for each N-point fft case. Below the results from N = 64 can be seen.

```
Command Window                                                    _ ☐ X

>> results

results =

 Columns 1 through 13

  24.1868   23.8886    6.2297   23.8787   18.1967    3.7448    8.2555    9.7823   22.6231   11.0879    6.8023   14.2593

 Columns 14 through 26

   7.3254   24.2352    6.5555    8.1519   23.8808   24.2611   22.0009   19.0563    2.1184   24.0654   23.9222   19.4687

 Columns 27 through 39

  24.3846    9.0470   19.0528    6.9919   19.7551    1.9400    8.2441    2.5429    3.7384   24.1566   19.5170    8.4885

 Columns 40 through 52

   2.0630    7.0618    9.1996   24.3361   24.2516    7.1818    7.6088    6.6768   18.8967   19.7554   24.3618    8.2375

 Columns 53 through 65

  19.0478    6.8757    6.0200    8.1243   21.8971    8.5615   14.5707    7.5517   24.3691    8.0756    8.5307   19.7503

 Columns 66 through 78

  22.0796    9.7860    6.4904    6.6924    8.0951   24.0953    8.0690   24.1883    7.9800   23.8973    9.4375    7.2867

 Columns 79 through 91

  17.3363    6.4084    9.4307   24.1307   14.5704    9.8106   23.8806    8.3076   24.3562   24.3639    9.2130   11.5058

 Columns 92 through 100

   2.7858    9.4800   24.2991   23.9224    6.2998   11.6069    5.1462   -0.1670    8.5557

fx >>
```
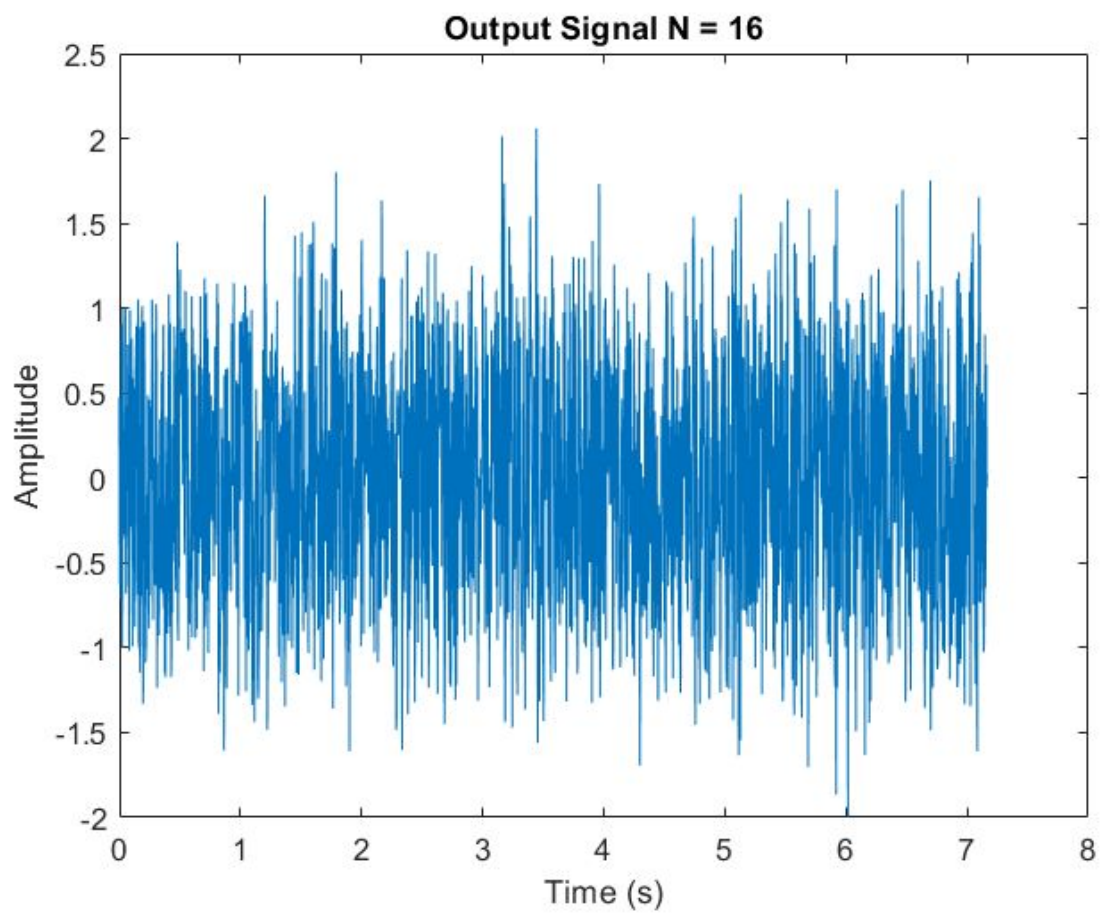
SNR Difference between output and input at each mu
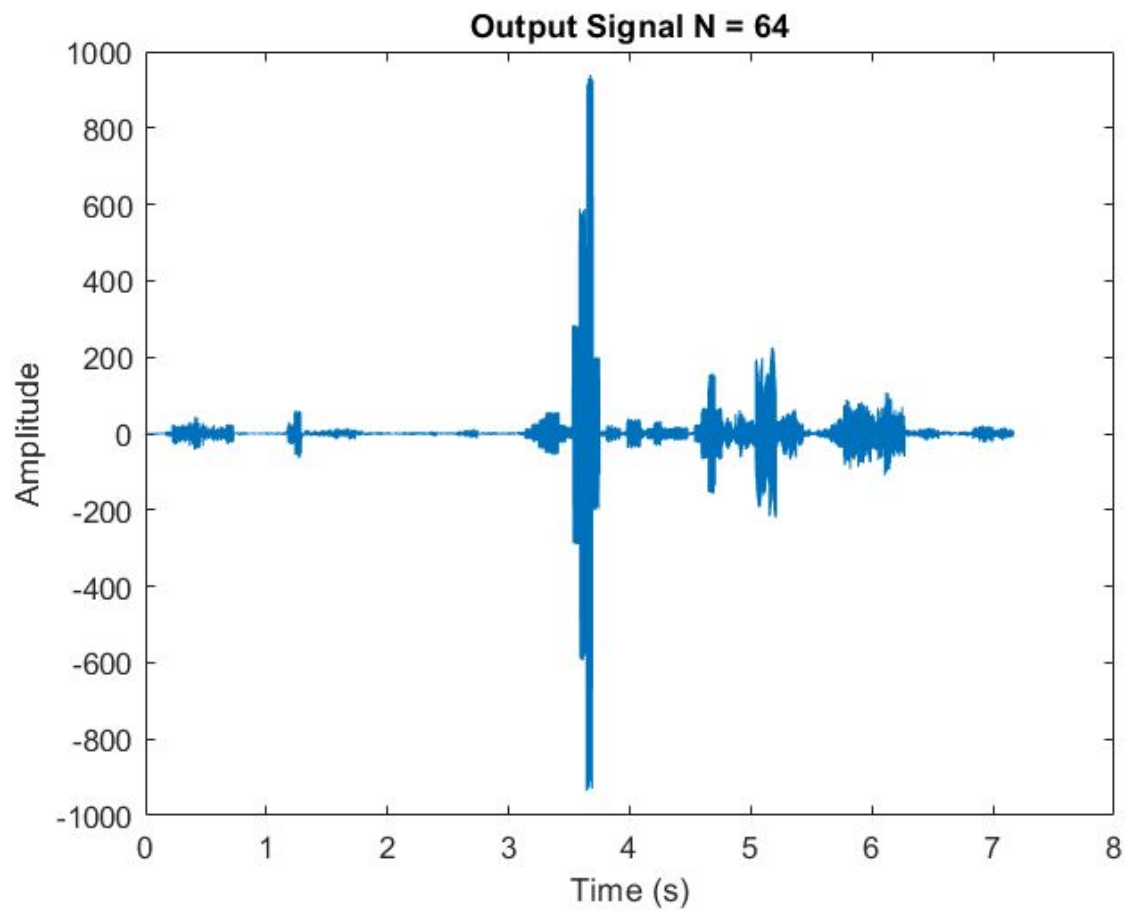
Test vector of mu values

Results

For each N-point FFT's optimal mu value the output signal and the difference in SNR between the input signal and output signal can be seen for each of the cases below:
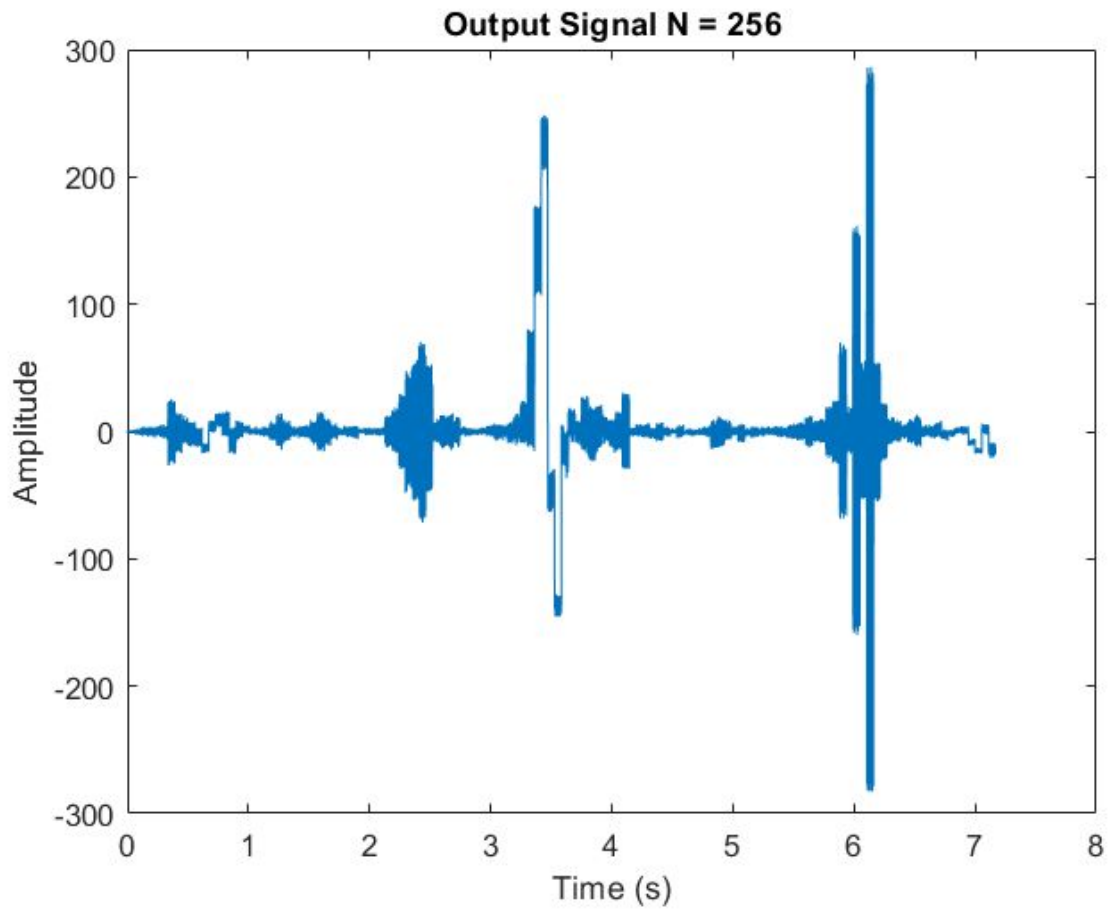
Output Signal N = 4
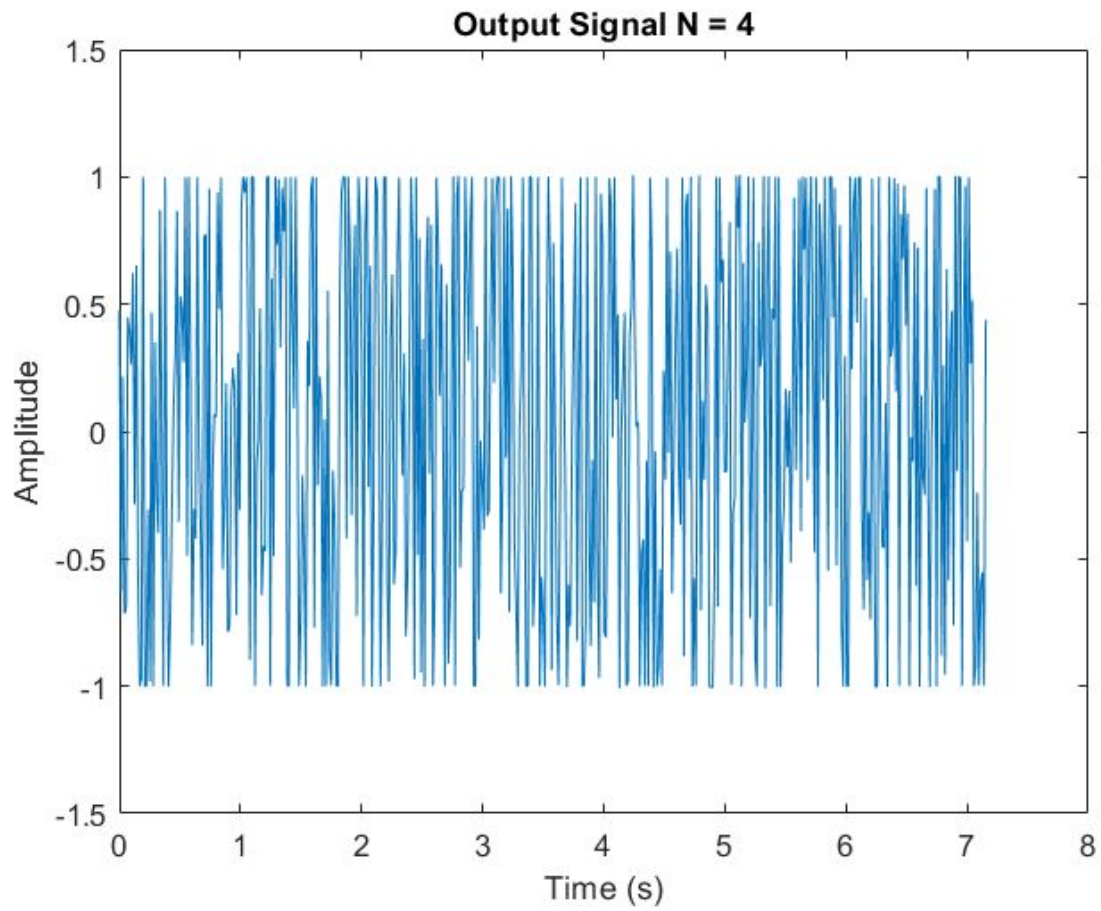

```
snr1 =

  -21.1297


snr2 =

  -9.6178

>> diff

diff =

   11.5120
```

Output Signal N = 16

```
snr1 =

  -21.1297


snr2 =

  -11.9589


diff =

    9.1709
```

Output Signal N = 64

snr1 =

   -21.1297

snr2 =

    3.2545

diff =

    24.3842

Output Signal N = 256

snr1 =

  -21.1297

snr2 =

   8.3085

diff =

  29.4383

Full simulation results:

N-point FFT	SNR Improvement (dB)	mu	nFrames
4	11.51	0.0007	128
16	9.17	0.0636	128
64	24.38	0.1115	128
256	29.44	0.0235	128

Error Convergence Charts:

N = 4

N = 16:

Error convergence plot

N = 64:

Error convergence plot
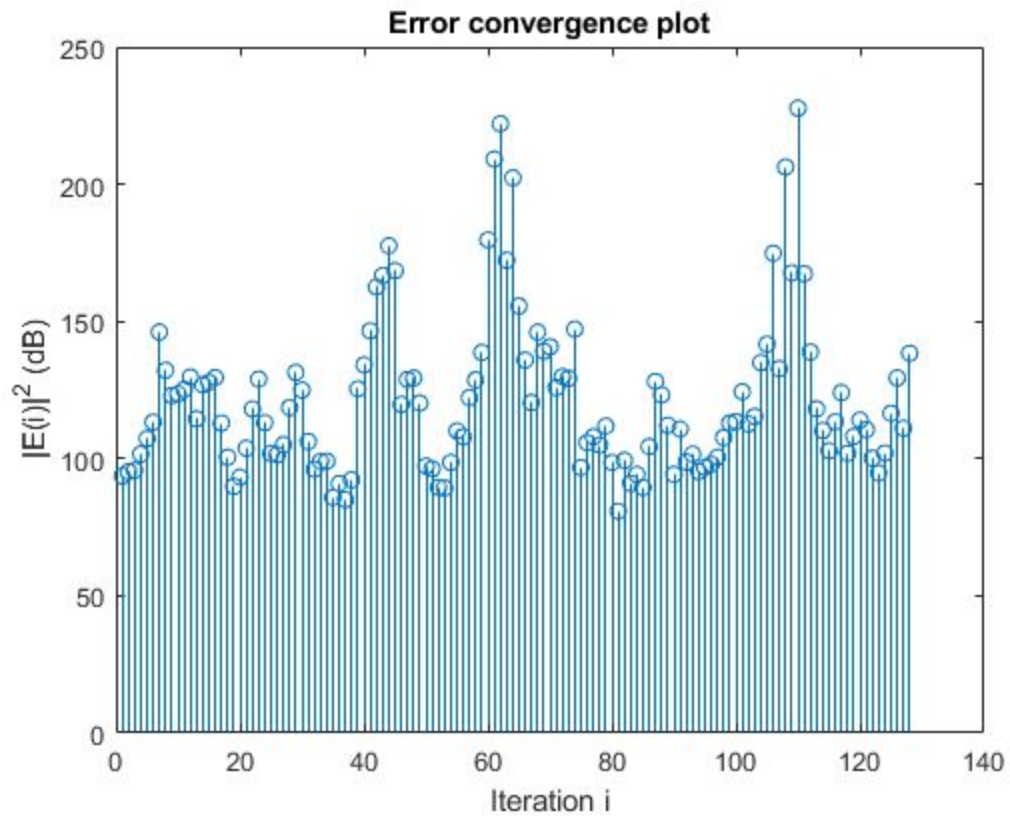
N = 128:

Error convergence plot
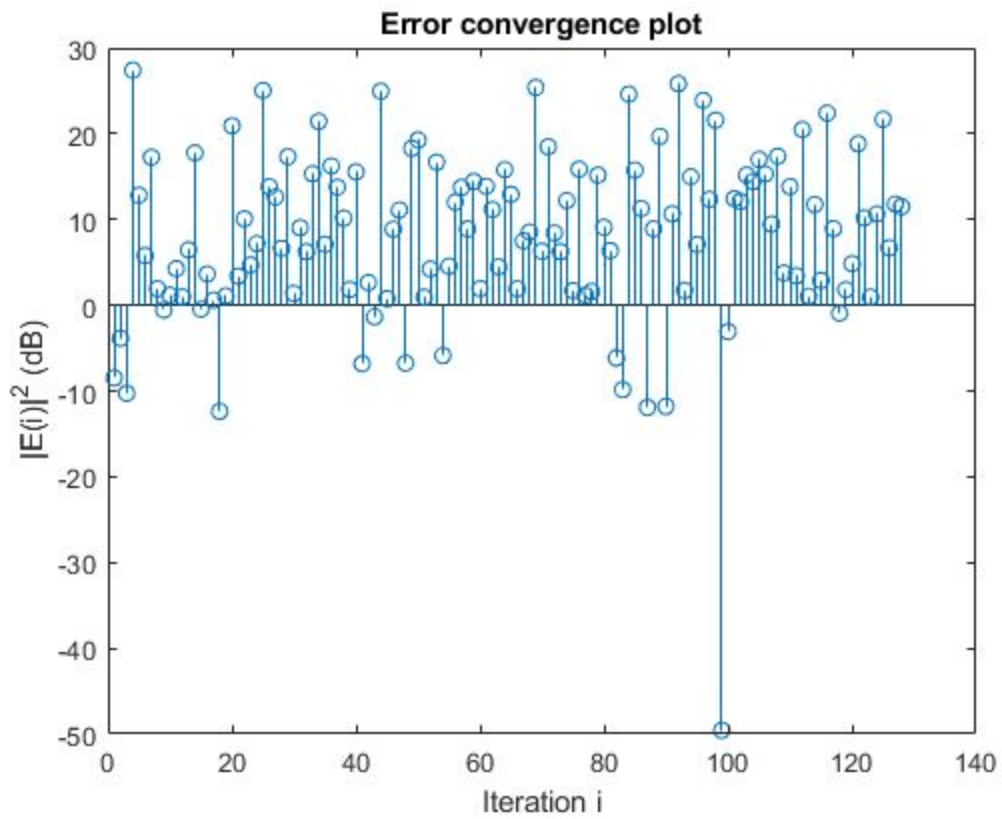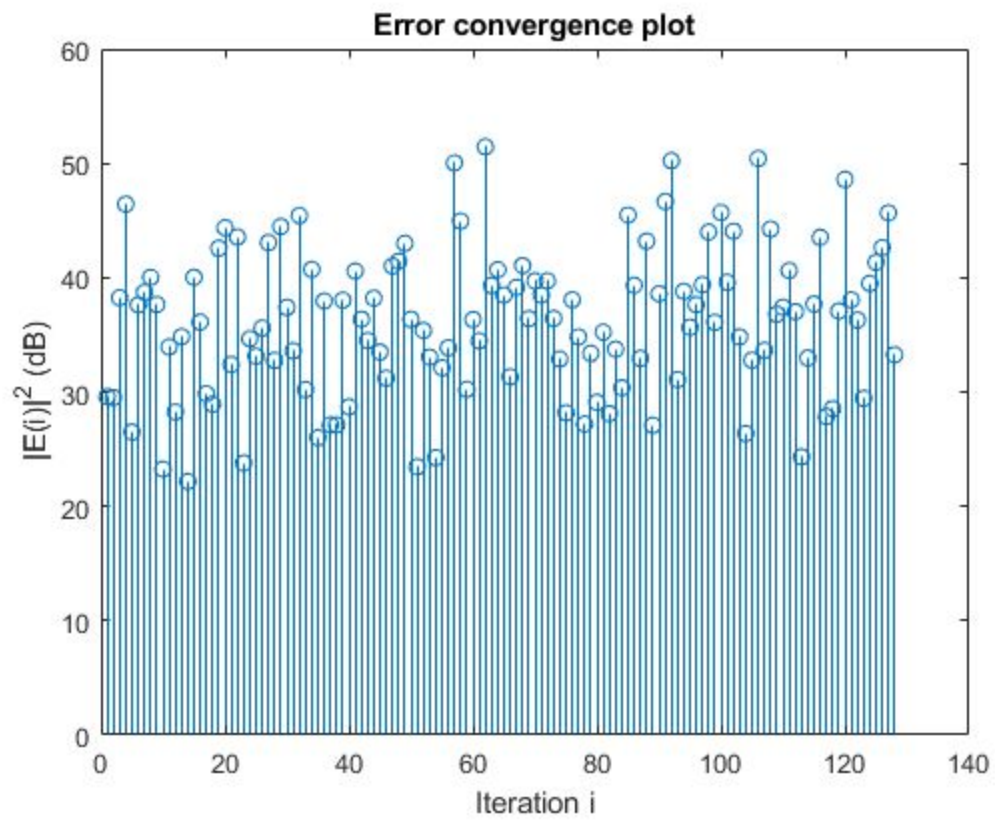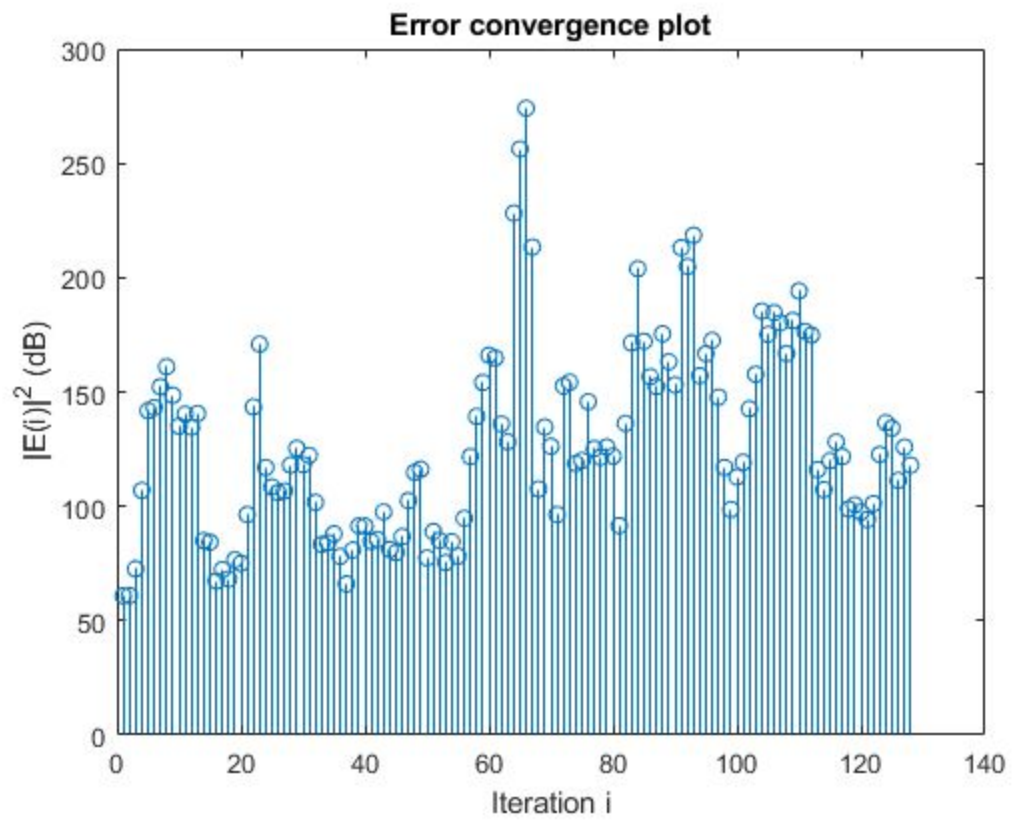
N = 256:
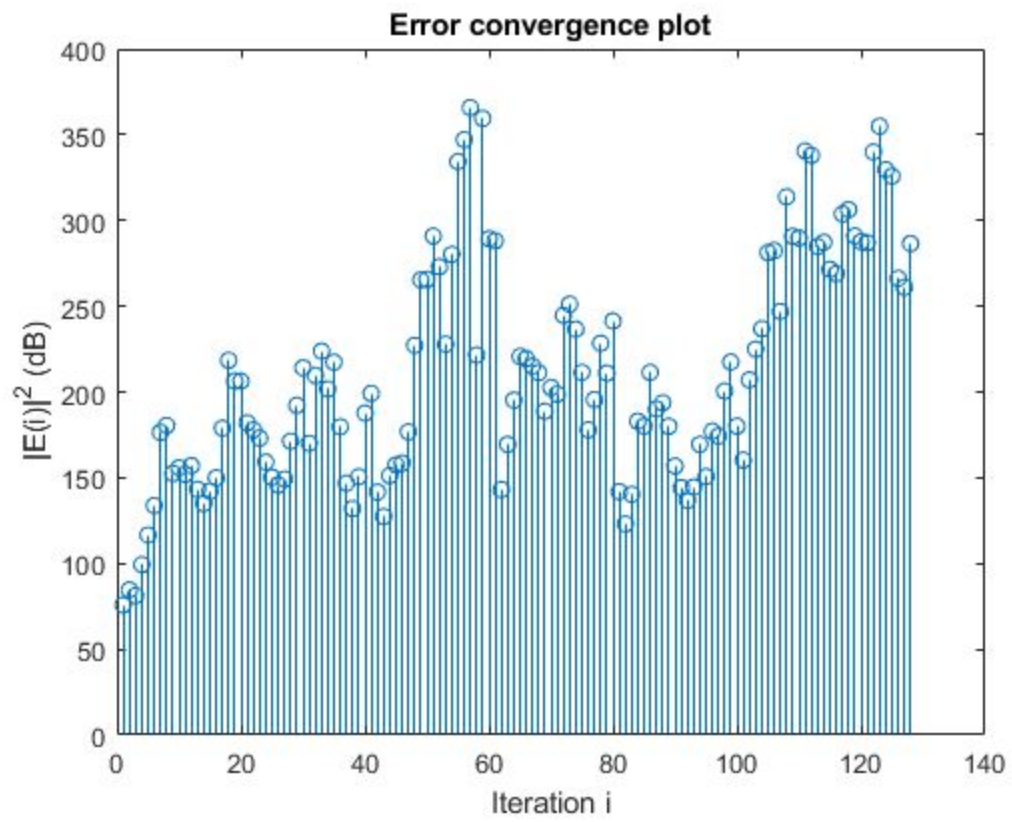
Error convergence plot
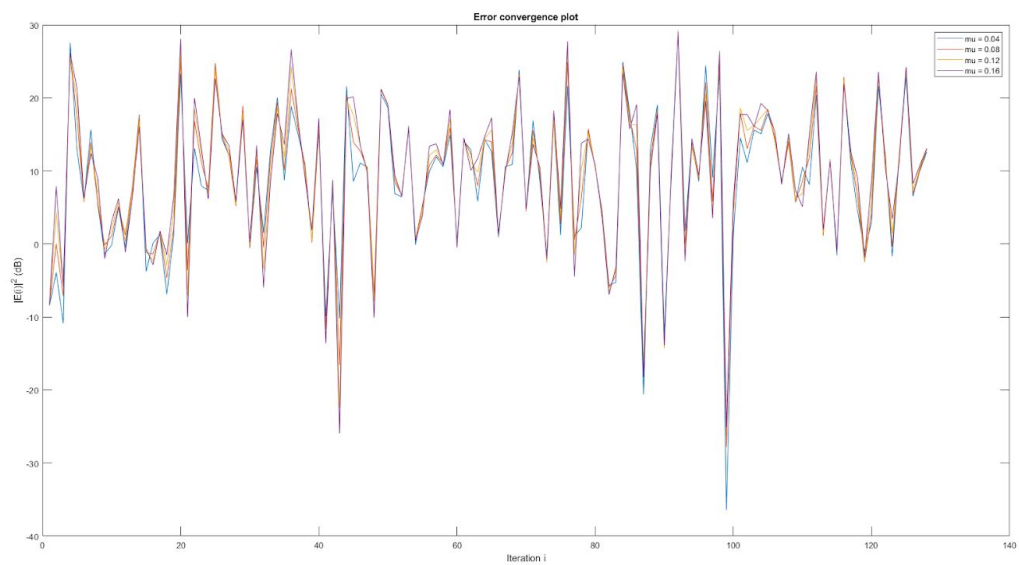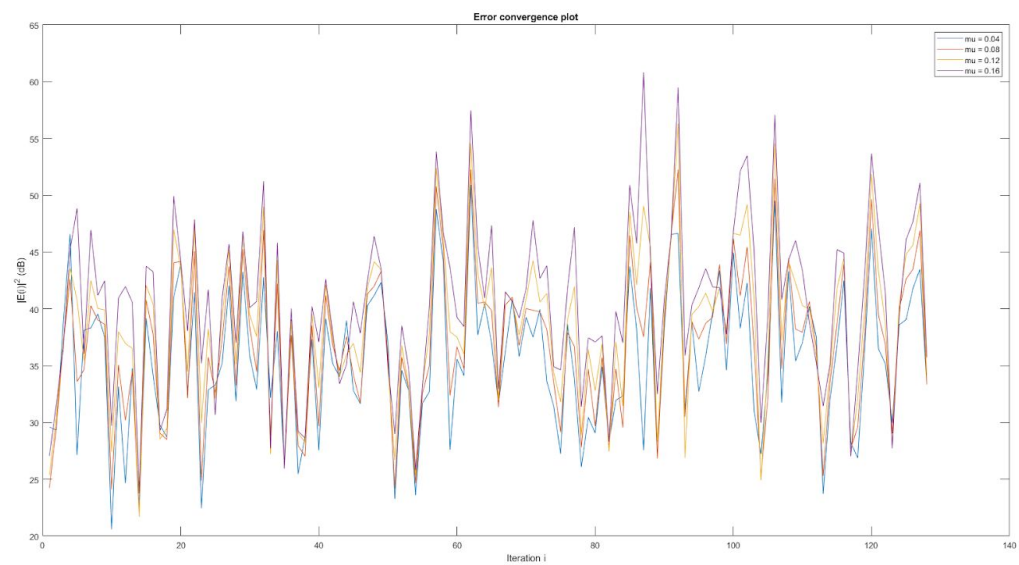

The following error convergence plots were generated at different mu values for each N-point FFT case. At N = 256 the filter becomes unstable by the time mu reaches a value of 0.16.
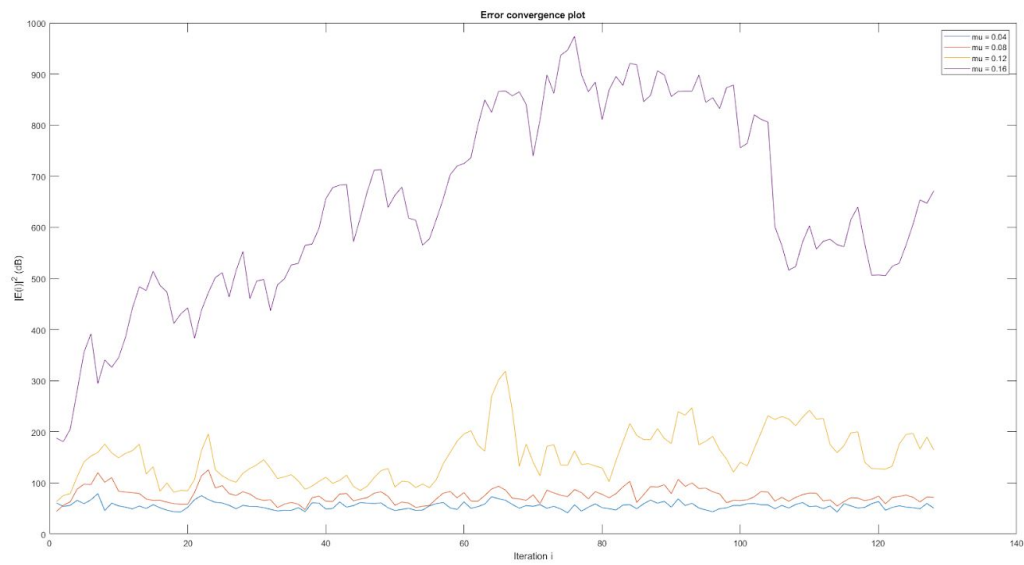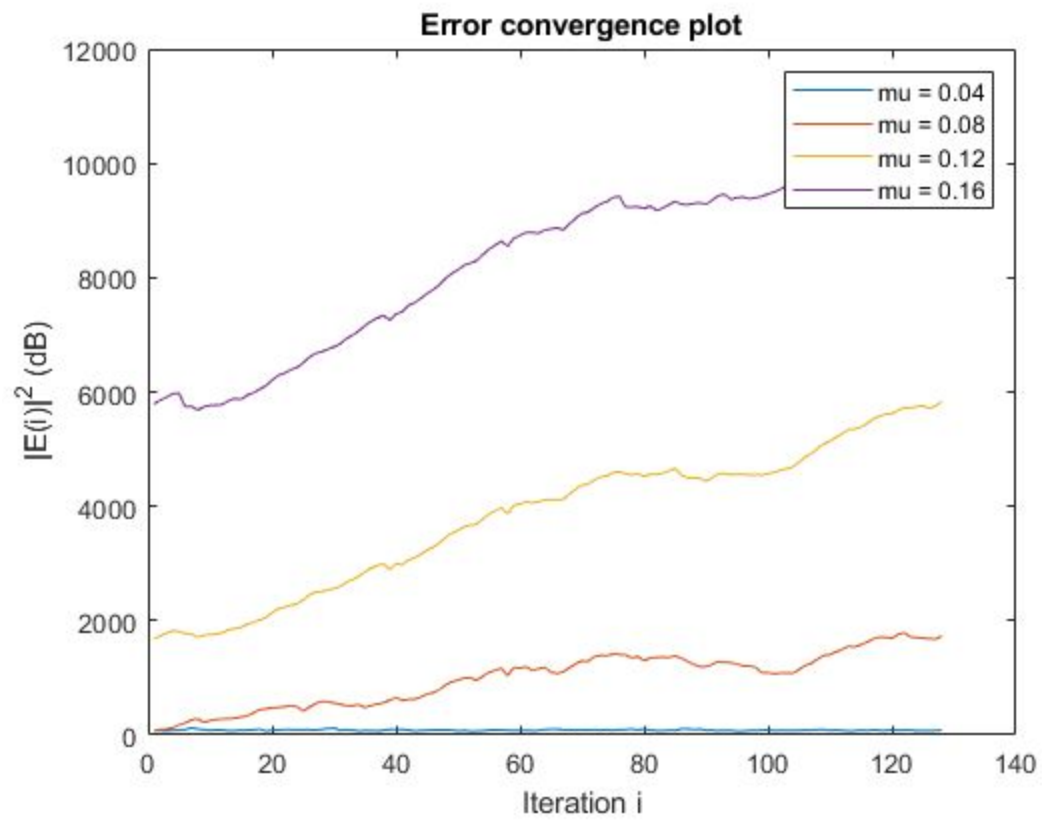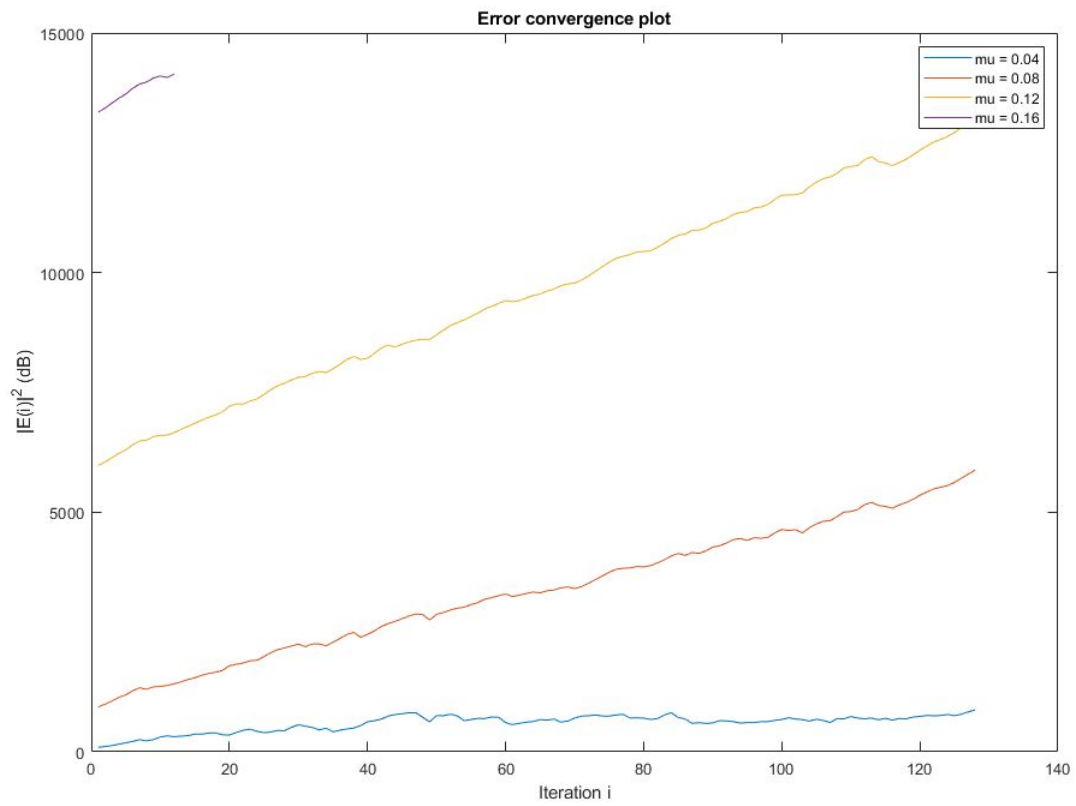
N = 4:

N = 16:

N = 64:

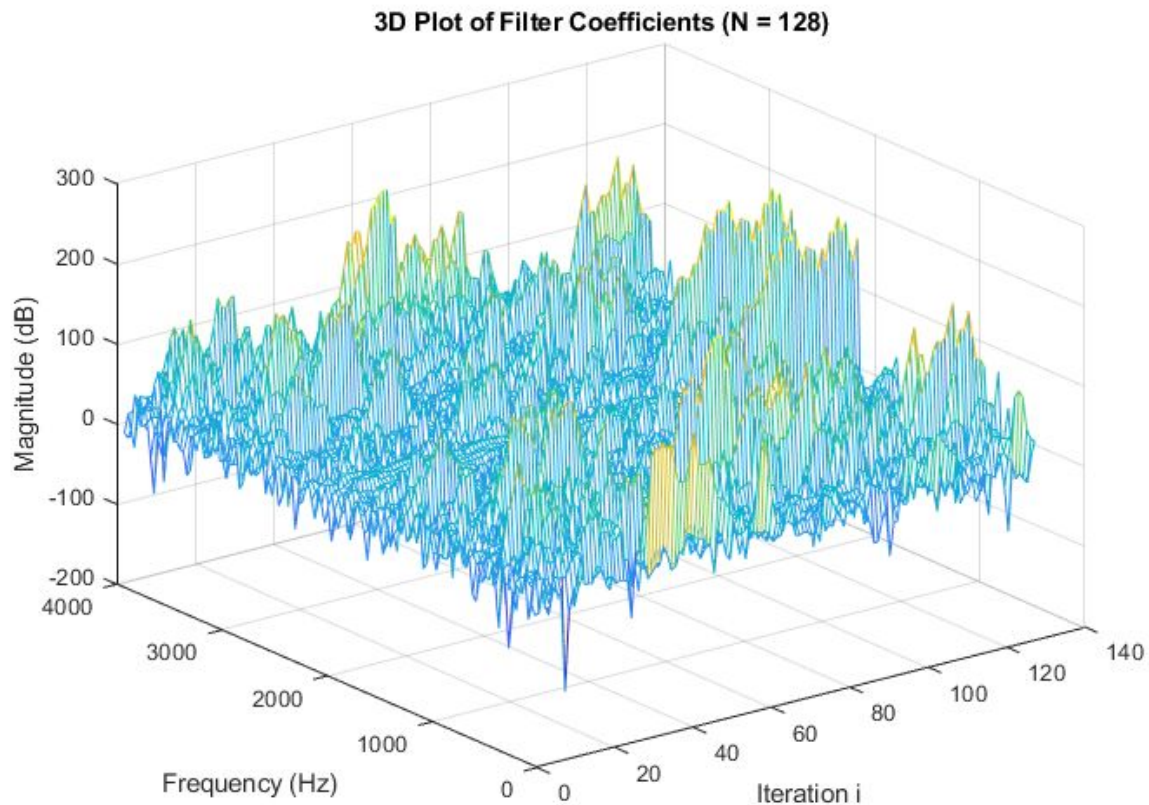Error convergence plot

N = 128:

Error convergence plot

N = 256:

Error convergence plot

The following 3D plot shows how the filter coefficients change at different frequencies across each adaptive iteration:

3D Plot of Filter Coefficients (N = 128)

Remarks

In the MATLAB program the variable B holds the filter coefficients. B(k) is a two-dimensional slice of the 3D mesh above taken at any frequency value (k).

The N-point FFT increases the computational complexity of the algorithm as N increases. When N is increased the spectral resolution improves but the order of the filter is held constant (some terms may be zero).

A larger value of mu increases the flexibility of the filter to react to each incoming data frame. However as the filter is made more flexible (more willing to adapt to each subsequent frame rather than resisting change) the stability of the filter comes into question. In general values of mu below mu = 0.15 lead to stable filters. However at a large value of N (such as N = 256) a smaller value of mu was required to avoid stability issues.

The filter coefficients after each subsequent frame were dependent on the FFT of each input signal frame. This frame-style signal segmentation allowed for parallel processing of data and improved computation time.

The computation time for the program is rather fast. I was able to run about 100 simulations in a manner of 30 seconds when determining optimal mu coefficients.

Appendix:

Main Code:

```matlab
% Louis Rosenblum
% Project 2
% EEE 509 - ASU
% 06/10/2020

%% Initialization

%clear all
close all

cd 'C:\Users\Louis\Desktop\DSP\Project 2'

%% Set runtime constants

N = 128;
nFrames = 128;

if (N == 4)
    mu = 0.0007;
elseif (N == 16)
    mu = 0.0636;
elseif (N == 64)
    mu = 0.1115;
elseif (N == 128)
    mu = 0.058;
else (N == 256)
    mu = 0.0235;
end

%% Read audio from file

[mic1, Fs1] = audioread('mic1_2019.wav');
[mic2, Fs2] = audioread('mic2_2019.wav');

% Calculate starting SNR
snr1 = snr(mic1)

%% Create frame data structures
```

```matlab
[mic1_frames, len1] = createFrames(mic1,nFrames);
[mic2_frames, len2] = createFrames(mic2,nFrames);

%% Perform N-point fft

mic1_fft = fft(mic1_frames,N);
mic2_fft = fft(mic2_frames,N);

%% Main adaptive filter algorithm

Es = [];
Bs = [];
es = [];
B = zeros(N,1);

k = 1:nFrames;
figure();

for i = 1:nFrames
    % Calculate for current frame
    diaganol = diag(mic1_fft(:,i));
    E = mic2_fft(:,i) - diaganol * B;
    e = ifft(E);

    Es = [Es E];
    Bs = [Bs B];
    es = [es e];

    % Prepare for next frame
    B = B + 2*mu * diaganol'*E;

    p = 10*log(1/N * E'*E);
    error = [error p]
end

title("Error convergence plot")

f = ones(1,nFrames);

for i = 1:nFrames
    f(i) = Fs1/2 / 128 *i;
end

figure()
mesh(k,f,20*log(abs(Bs)))
title("3D Plot of Filter Coefficients (N = 128)")
```

```matlab
xlabel("Iteration i")
ylabel("Frequency (Hz)")
zlabel("Magnitude (dB)")


%% Revert frames to vector

final_signal = createVector(es);

% Calculate ending SNR
snr2 = snr(final_signal)

diff = snr2 - snr1

%% Create plots

% Plot original signal

t = 0:1/Fs1:(length(mic1)-1)/Fs1;
t2 = 0:1/Fs1:(length(final_signal)-1)/Fs1;

figure();
plot(t,mic1);
title("Input Signal")
xlabel("Time (s)")
ylabel("Amplitude")

% Plot filtered signal
figure();
plot(t2,final_signal);
title("Output Signal")
xlabel("Time (s)")
ylabel("Amplitude")
%% Function definitions

function [data, len] = createFrames(audio,nFrames)
    len = length(audio);
        frameSize = ceil(len/nFrames);
    total = frameSize*nFrames;
    z = total - len;
    pad = [audio;zeros(z,1)];
    data = reshape(pad,frameSize,nFrames);
end

function data = createVector(audio)
    [m,n] = size(audio);
    data = [];
```

```
    for i = 1:n
        data = [data rot90(audio(:,i))];
    end
end
```

Optimal-mu Calculator:

```
results = []
mus = []

for i = 1:100
    mu = rand(1,1) * 0.15;
    project2;
    results = [results diff];
    mus = [mus mu];
end

[M,I] = max(results);

Max_diff = M

Max_mu = mus(I)
```