# EE 542

# Applications
# In
# Digital Signal Processing

## Computer Assignment #4

Zexi Liu

May 5th, 2007

Electrical and Computer Engineering

Temple University

# 1. Introduction

*Wiener Filters*. We are going to explore the world of adaptive filters by looking at the classical adaptive filter, the *Wiener filter*. The task we are assigned is to clean a speech signal, which is corrupted by an unknown channel. For this purpose, we are given two signals, d[n] and u[n], corresponding to the desired (clean) signal and the distorted signal. The usual protocol is to determine wiener filters on short ‘frames’ of speech data, i.e., on data windows of duration 32 milliseconds.

We can use the command *buffer ( );* to create frames and determine the optimal filter weights for each frame. We will need to determine a good number for M, the number of filter coefficients. Is there a way to determine the order automatically?

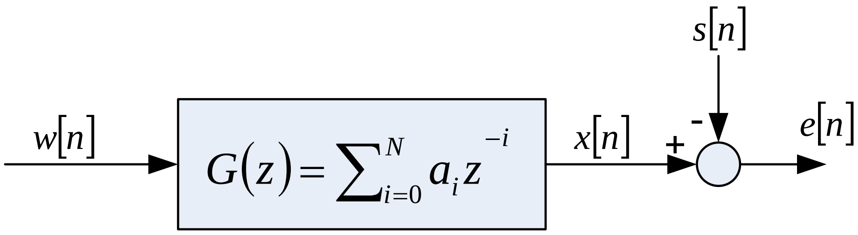


Figure 1.1

Figure 1.1 shows the block diagram view of the FIR Wiener filter for discrete series. An input signal w[n] is convolved with the Wiener filter g[n] and the result is compared to a reference signal s[n] to obtain the filtering error e[n].

# 2. Simulation & Results

### 2.1 Simulation version # 1 (Matlab Simulink)

In this version, a LMS adaptive filter (Figure 2.1) was used. First of all, the following MatLab code will load the data into workplace and create two *.wav files which is going to be used in the simulation.

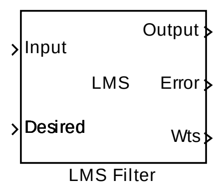


Figure 2.1

```
load wiener_project.mat;
wavwrite(d, fs, 32, 'd.wav');
wavwrite(u, fs, 32, 'u.wav');
```

The input signal of the LMS adaptive filter will be “u.wav” (distorted signal). The desired input of the LMS filter will be “d.wav” (desired signal).

The output signals of the LMS filter are “Output”, “Error” and “Wts”. “Wts” is the filter weights, the filter coefficients. Figure 2.2 shows the Simulink diagram.

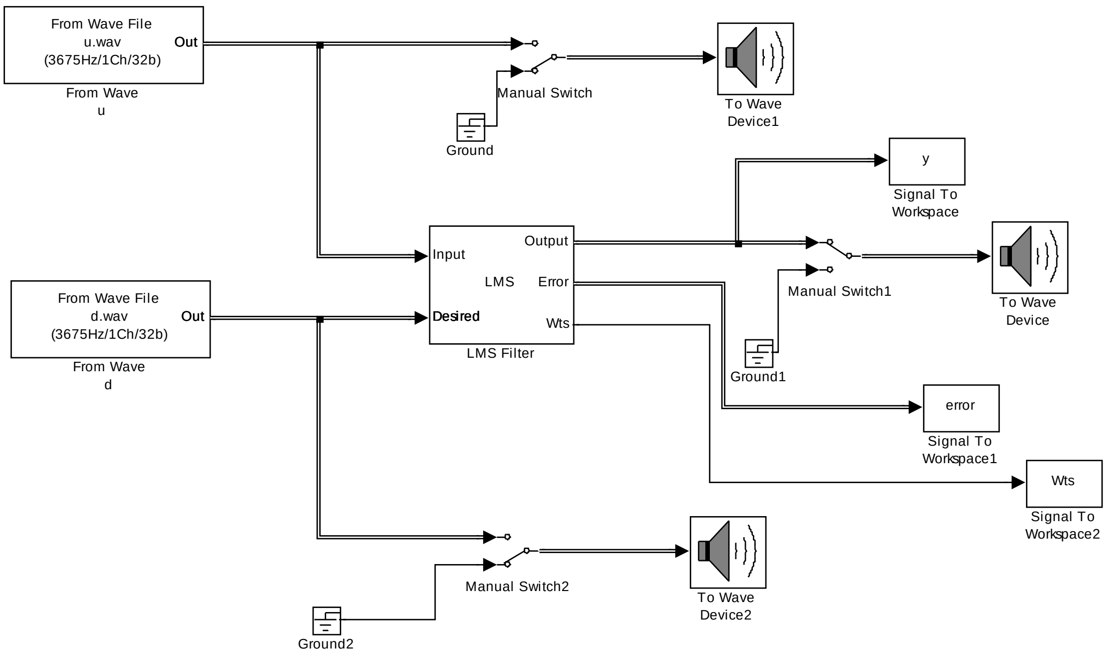


Figure 2.2

The input signals (Input and Desired) are divided into small frames, each of them contains only 117 points (32 milliseconds). (Figure 2.3)

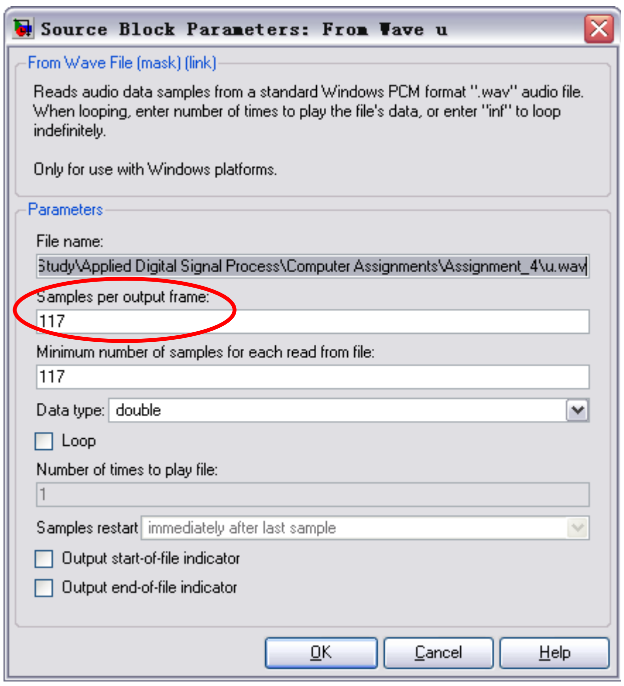


Figure 2.3

There are different sets of coefficients for each frame. As a result, there are 26 sets of coefficients.

The signal to noise ratio, SNR, will be calculated based on the output *y*, input *d* and *u*.

$$SNR = 10\log\left(\frac{Energy(d)}{Energy(y-d)}\right)$$

$$SNR = 10\log\left(\frac{Energy(d)}{Energy(u-d)}\right)$$

Appendix B is the program for calculating SNR.

Table 2-1 shows the relation between SNR and the filter length. The filter length can be set by double click the LMS block. In Figure 2.4, The LMS block will use 32$^{nd}$ order filter on every frame, but with different coefficients.

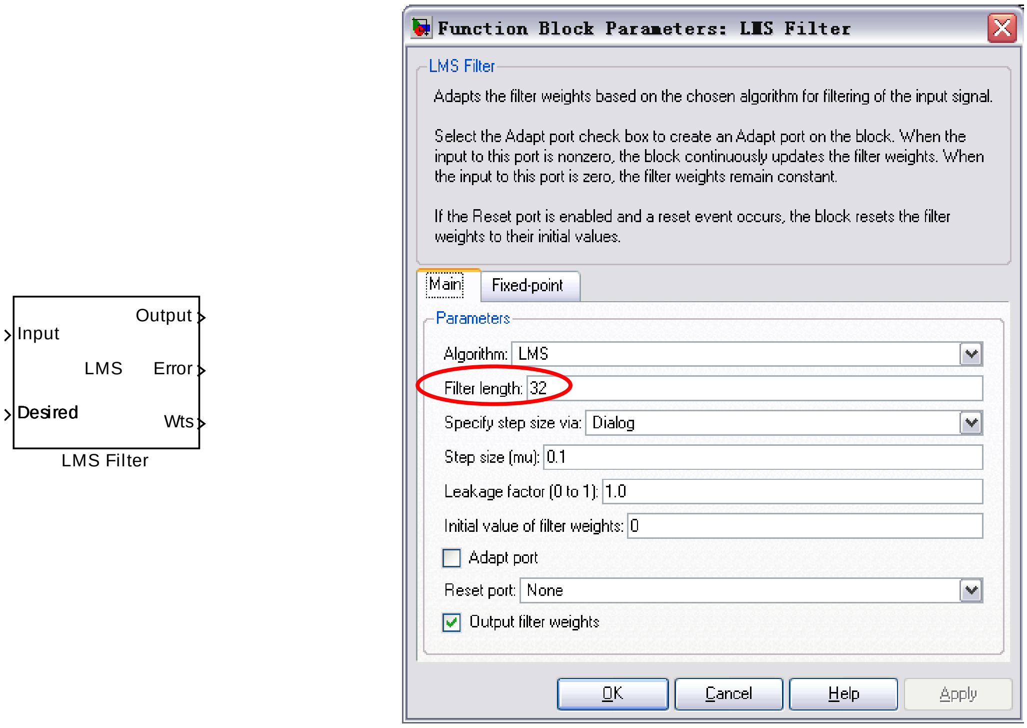


Figure 2.4

Figure 2.5 shows the plot of filter length vs. SNR. When filter length equals 20, we get the maximum SNR 19.62 dB.

Table 2-1. Filter length vs. SNR

| Filter length | SNR |
|---|---|
| 1 | 1.72031210335384 dB |
| 2 | 8.56604526368706 dB |
| 3 | 12.47327535324289 dB |
| 4 | 11.35457476444518 dB |
| 5 | 11.43119180717354 dB |
| 6 | 12.07526216307051 dB |
| 7 | 12.43498375084494 dB |
| 8 | 12.45392989433791 dB |
| 9 | 12.46250961859097 dB |
| 10 | 12.73854816161960 dB |
| 11 | 12.39754439710474 dB |
| 12 | 12.22549703517128 dB |
| 13 | 12.27491471456487 dB |
| 14 | 13.36398148618421 dB |
| 15 | 13.90951127922396 dB |
| 16 | 14.07970806510731 dB |
| 17 | 15.12792554766099 dB |
| 18 | 16.47643537874011 dB |
| 19 | 16.41396938774127 dB |
| 20 | 16.91796231065474 dB |
| 21 | 16.68136136053469 dB |
| 22 | 16.58399584154954 dB |
| 23 | 16.52479866563757 dB |
| 24 | 16.42119761533949 dB |
| 25 | 16.28812357648547 dB |
| 26 | 16.23701406221756 dB |
| 27 | 16.16651148837740 dB |
| 28 | 16.12141747604067 dB |
| 29 | 16.06216194987557 dB |
| 30 | 15.98953461785923 dB |
| 31 | 15.89441827885663 dB |
| 32 | 15.85838735465108 dB |
| 33 | 15.75131018484776 dB |
| 34 | 15.61729679537825 dB |
| 35 | 15.74075204271125 dB |
| 36 | 15.73566198337089 dB |
| 40 | 14.91762751305530 dB |

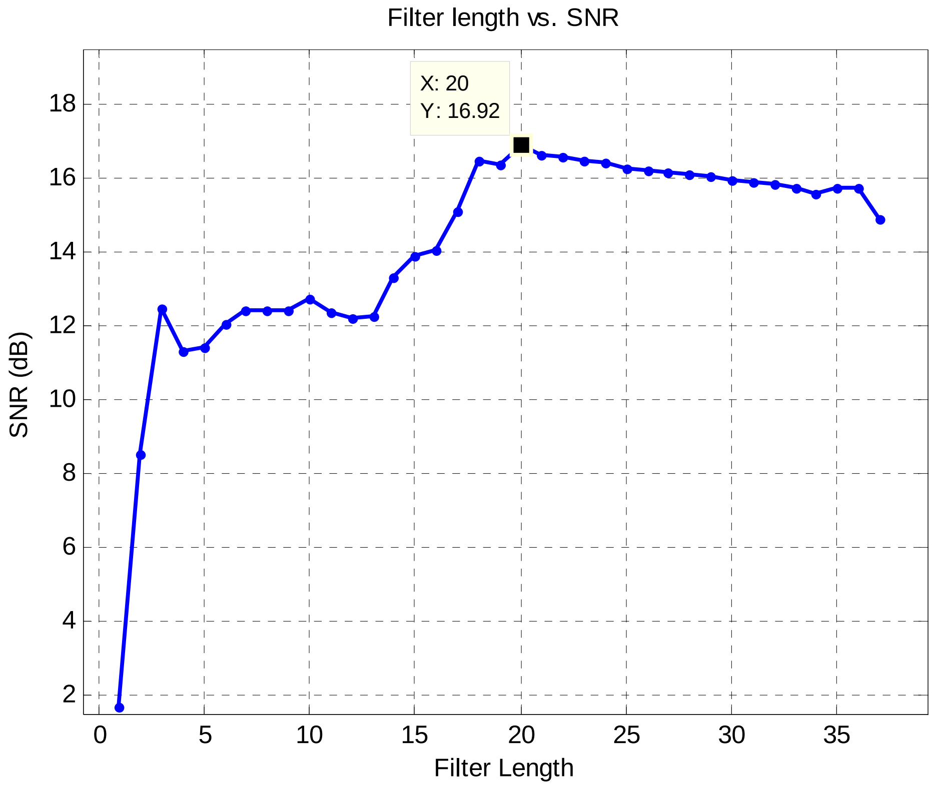


Figure 2.5

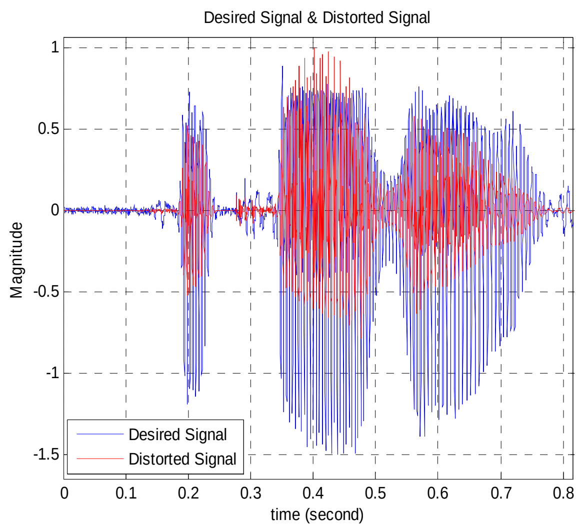


Figure 2.6

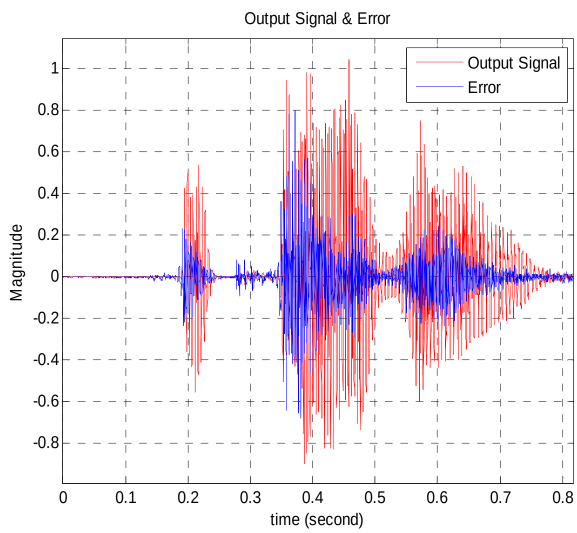


Figure 2.7

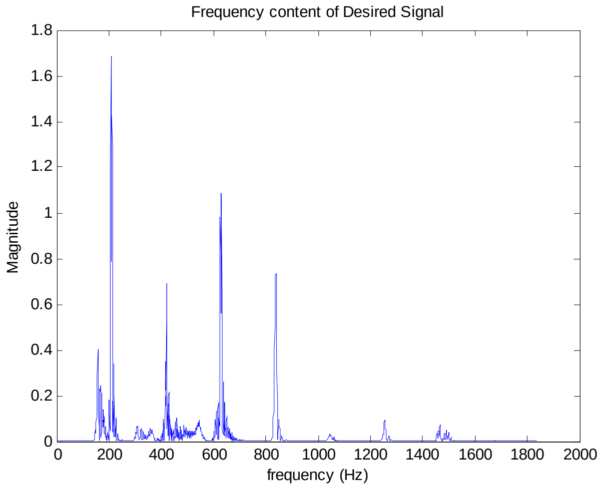


Figure 2.8

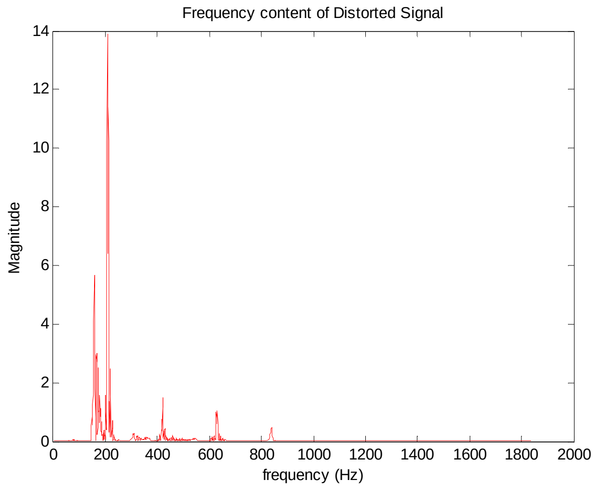


Figure 2.9

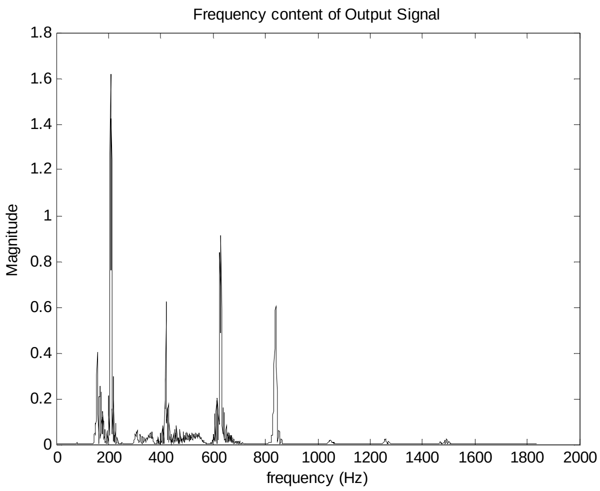


Figure 2.10

## 2.2 Simulation version #2
## (Appendix A Matlab code)

In this version, I used the following program to design a Wiener filter. The command *buffer ()* was used to create frames. Figure 2.11 shows the flowchart of this program. The key idea is to obtain the optimal filter length by calculating SNR for each frame. Once we get the maximum SNR for each frame, the current filter length will be saved. After calculating 26 filters length, we will process these 26 frames using the filter length we just got.

Table 2-2 shows the different filter length for different frames. After processing each frame, the results are 26 frames. We put these frames together as our finial output signal. Note that the finial result (output signal SNR) is **7.25762495763136 dB** which is less than the result from simulation version #1.

I think this is because of the size of each frame. Remember our current frame size is 32 milliseconds, 117 samples per frame. What if we change the size of the frames? To answer this question, I used different frame size and observe the changes of output signal SNR. Table 2-3 and figure 2.12 shows the relation between frame size and SNR.

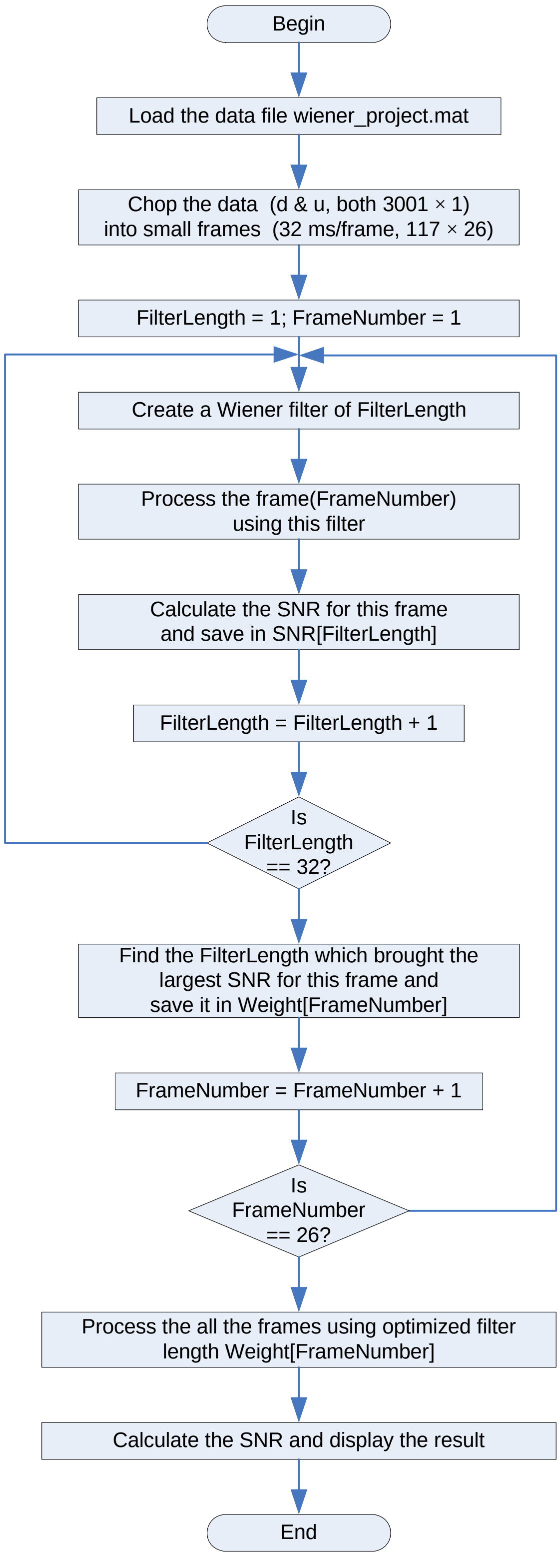


Figure 2.11

Table 2-2. Filter length vs. Maximum SNR for each frame (32 ms frame)

| Frame No. | Filter Length | Max. SNR (dB) |
|---|---|---|
| 1 | 32 | 0.00000518611192 |
| 2 | 32 | 0.00000913770197 |
| 3 | 19 | 0.00000467489172 |
| 4 | 32 | 0.00000766693364 |
| 5 | 28 | 0.00311948403408 |
| 6 | 32 | 0.04952166947020 |
| 7 | 32 | 11.47829278576712 |
| 8 | 21 | 7.04625467126175 |
| 9 | 29 | 0.00001715095883 |
| 10 | 32 | 0.16705239352022 |
| 11 | 32 | 2.38537866991049 |
| 12 | 23 | 5.43647625811268 |
| 13 | 32 | 7.80462629878529 |
| 14 | 32 | 7.82124837138370 |
| 15 | 24 | 7.63391970440910 |
| 16 | 18 | 8.15675363705034 |
| 17 | 32 | 4.55717504290340 |
| 18 | 24 | 6.30442776710502 |
| 19 | 26 | 6.64756668074481 |
| 20 | 30 | 5.39314827786473 |
| 21 | 31 | 6.09697198381866 |
| 22 | 29 | 11.93993760515972 |
| 23 | 27 | 13.94601594727313 |
| 24 | 25 | 6.99896029864786 |
| 24 | 32 | 0.12517281197559 |
| 26 | 32 | 0.68079103311180 |
| **Output signal SNR: 7.25762495763136 dB** | | |

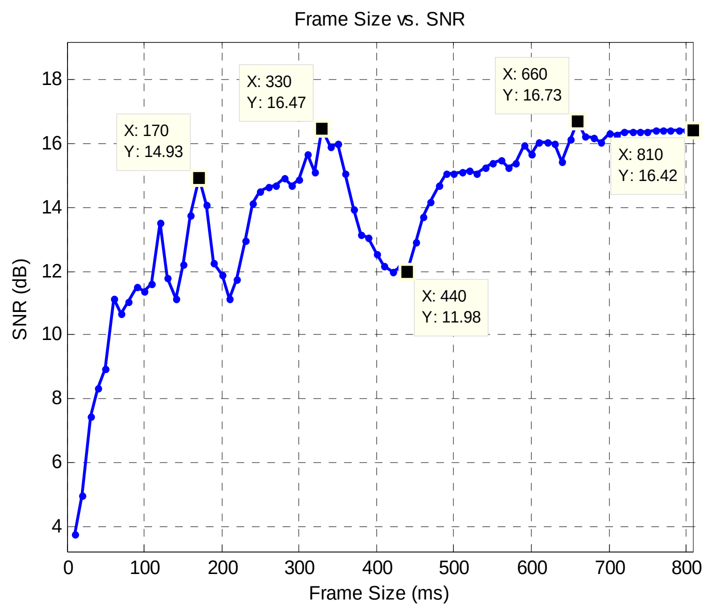


Figure 2.12

Table 2-3. Frame size vs. SNR

| Frame size | | SNR (dB) | Frame size | | SNR (dB) |
|---|---|---|---|---|---|
| ms | samples/frame | | ms | samples/frame | |
| 10 | 36 | 3.750040082 | 410 | 1506 | 12.15801916 |
| 20 | 73 | 4.957327635 | 420 | 1543 | 12.00542645 |
| 30 | 110 | 7.457736177 | 430 | 1580 | 12.13457216 |
| 40 | 147 | 8.33045108 | 440 | 1616 | 11.98140153 |
| 50 | 183 | 8.95104157 | 450 | 1653 | 12.92144607 |
| 60 | 220 | 11.1529707 | 460 | 1690 | 13.72485956 |
| 70 | 257 | 10.6826373 | 470 | 1727 | 14.20818606 |
| 80 | 294 | 11.04497061 | 480 | 1763 | 14.7143072 |
| 90 | 330 | 11.53495345 | 490 | 1800 | 15.08091943 |
| 100 | 367 | 11.38932894 | 500 | 1837 | 15.08077676 |
| 110 | 404 | 11.59163862 | 510 | 1874 | 15.10470879 |
| 120 | 440 | 13.53574629 | 520 | 1910 | 15.17052451 |
| 130 | 477 | 11.78322126 | 530 | 1947 | 15.05284045 |
| 140 | 514 | 11.12539786 | 540 | 1984 | 15.24726128 |
| 150 | 551 | 12.24111347 | 550 | 2021 | 15.38896741 |
| 160 | 588 | 13.78777547 | 560 | 2058 | 15.48267619 |
| 170 | 624 | 14.92835544 | 570 | 2094 | 15.24620808 |
| 180 | 661 | 14.08088387 | 580 | 2131 | 15.40257674 |
| 190 | 698 | 12.24744984 | 590 | 2168 | 15.94598506 |
| 200 | 734 | 11.87147551 | 600 | 2205 | 15.66516498 |
| 210 | 771 | 11.14579025 | 610 | 2241 | 16.06541747 |
| 220 | 808 | 11.7683805 | 620 | 2278 | 16.04078177 |
| 230 | 845 | 12.97366374 | 630 | 2315 | 15.99425576 |
| 240 | 881 | 14.12787458 | 640 | 2352 | 15.45113633 |
| 250 | 918 | 14.50806335 | 650 | 2388 | 16.14458568 |
| 260 | 955 | 14.63064222 | 660 | 2425 | 16.72978035 |
| 270 | 992 | 14.69326547 | 670 | 2462 | 16.22789102 |
| 280 | 1029 | 14.91356534 | 680 | 2499 | 16.1915154 |
| 290 | 1065 | 14.68922687 | 690 | 2535 | 16.06707723 |
| 300 | 1102 | 14.91118357 | 700 | 2572 | 16.36115187 |
| 310 | 1139 | 15.68606051 | 710 | 2609 | 16.28461809 |
| 320 | 1176 | 15.112939 | 720 | 2645 | 16.38720186 |
| 330 | 1212 | 16.46659882 | 730 | 2682 | 16.37511593 |
| 340 | 1249 | 15.93152099 | 740 | 2719 | 16.40011111 |
| 350 | 1286 | 15.99907606 | 750 | 2756 | 16.38876207 |
| 360 | 1322 | 15.07038081 | 760 | 2792 | 16.42436225 |
| 370 | 1359 | 13.97097176 | 770 | 2829 | 16.41682834 |
| 380 | 1396 | 13.15517332 | 780 | 2866 | 16.42204556 |
| 390 | 1433 | 13.07234799 | 790 | 2903 | 16.42132306 |
| 400 | 1469 | 12.56447433 | 800 | 2939 | 16.42291097 |

# Appendix A

```
% ------------------------------------------------------------------------------------------
% Clear workspace & command window
% ------------------------------------------------------------------------------------------
clc;
clear;
% ------------------------------------------------------------------------------------------
load('wiener_project.mat');                 % load data file
SP = 0;                                     % initialization
SN = 1;                                     % initialization
SNRF = 0;                                   % initialization
SNR = 0;                                    % initialization
for FrS = 10:10:810                         % frame size from 10 ms to 810 ms
    SP(SN) = fix(FrS / (1 / (fs / 1000)));  % Samples per frame
    dbuf = buffer(d, SP(SN));               % chop the data into frames
    ubuf = buffer(u, SP(SN));               % chop the data into frames
    y = buffer(d, SP(SN));
    e = buffer(u, SP(SN));
    mu = 0.1;                               % LMS step size.
    Weight = 0;
    for j = 1 : size(dbuf,2)
        for L = 1 : 32
            ha = adaptfilt.lms(L, mu);
            [y(:, j), e(:, j)] = filter(ha, ubuf(:, j), dbuf(:, j));
            D = fft(dbuf(:, j), 256);
            Pdd = D.* conj(D) / 256;
            Y = fft(y(:, j), 256);
            Pyy = Y.* conj(Y) / 256;
            sum1 = 0;
            sum2 = 0;
            for i = 1:numel(Pdd)
                sum1 = sum1 + Pdd(i)^2;
                sum2 = sum2 + ((Pyy(i)) - (Pdd(i)))^2;
            end
            SNRF(j, L) = 10 * log10((sum1) / (sum2));
        end
%       disp(max(SNR(j, :)));
        for i = 1 : L
            if SNRF(j, i) == max(SNRF(j, :))
                Weight(j) = i;
            end
        end
    end
    L = 1;
```

```
    for j = 1 : size(dbuf,2)        % 1:26
        ha = adaptfilt.lms(Weight(L), mu);
        [y(:, j), e(:, j)] = filter(ha, ubuf(:, j), dbuf(:, j));
        L = L + 1;
    end

    y = reshape(y, numel(dbuf), 1);
    e = reshape(e, numel(dbuf), 1);

    D = fft(d, 4096);
    Pdd = D.* conj(D) / 4096;
%     f = 3675 * (0:2048)/4096;
%     figure('Name', 'd', 'NumberTitle', 'off');
%     plot(f, Pdd(1:2049)); hold on;
%     title('Fig 2.12 Frequency content of d signal');
%     xlabel('frequency (Hz)');

%     U = fft(u, 4096);
%     Puu = U.* conj(U) / 4096;
%     f = 3675 * (0:2048)/4096;
%     figure('Name', 'u', 'NumberTitle', 'off');
%     plot(f, Puu(1:2049), 'r'); hold on;
%     title('Fig 2.12 Frequency content of u signal');
%     xlabel('frequency (Hz)');

%     sum1 = 0;
%     sum2 = 0;
%     for i = 1:numel(Pdd)
%         sum1 = sum1 + Pdd(i)^2;
%         sum2 = sum2 + ((Puu(i)) - (Pdd(i)))^2;
%     end
%     SNR = 10 * log10((sum1) / (sum2));
%     disp(SNR);

    % y = y(1:3100);
    Y = fft(y, 4096);
    Pyy = Y.* conj(Y) / 4096;
%     f = 3675 * (0:2048)/4096;
%     figure('Name', 'y', 'NumberTitle', 'off');
%     plot(f, Pyy(1:2049), 'k'); hold on;
%     title('Fig 2.12 Frequency content of y signal');
%     xlabel('frequency (Hz)');

    sum1 = 0;
    sum2 = 0;
    for i = 1:numel(Pdd)
```

```
        sum1 = sum1 + Pdd(i)^2;
        sum2 = sum2 + ((Pyy(i)) - (Pdd(i)))^2;
    end
    SNR(SN) = 10 * log10((sum1) / (sum2));
    disp(SNR(SN));
    SN = SN + 1;
end
```

# Appendix B

```
clc;
% clear;
load wiener_project.mat;
% %
% % dbuf = buffer(d, 117);
% % ubuf = buffer(u, 117);
% %
% mu = 0.1;                    % LMS step size.
% ha = adaptfilt.lms(32, mu);
% % for j = 1:size(dbuf,2)
% %     [y(:, j), e(:, j)] = filter(ha, ubuf(:, j), dbuf(:, j));
% % end
% %
% % y = reshape(y, numel(dbuf), 1);
% % e = reshape(e, numel(dbuf), 1);
% %
% [y, e] = filter(ha, u, d);

figure('Name', 'd', 'NumberTitle', 'off');
D = fft(d, 4096);
Pdd = D.* conj(D) / 4096;
f = 3675 * (0:2048)/4096;
plot(f, Pdd(1:2049)); hold on;
title('Frequency content of Desired Signal');
ylabel('Magnitude');
xlabel('frequency (Hz)');

figure('Name', 'u', 'NumberTitle', 'off');
U = fft(u, 4096);
Puu = U.* conj(U) / 4096;
f = 3675 * (0:2048)/4096;
plot(f, Puu(1:2049), 'r'); hold on;

title('Frequency content of Distorted Signal');
ylabel('Magnitude');
```

```
xlabel('frequency (Hz)');

sum1 = 0;
sum2 = 0;
for i = 1:numel(Pdd)
    sum1 = sum1 + Pdd(i)^2;
    sum2 = sum2 + ((Puu(i)) - (Pdd(i)))^2;
end
SNR = 10 * log10((sum1) / (sum2));
disp(SNR);

figure('Name', 'y', 'NumberTitle', 'off');
y = y(1:4000);
Y = fft(y, 4096);
Pyy = Y.* conj(Y) / 4096;
f = 3675 * (0:2048)/4096;
plot(f, Pyy(1:2049), 'k'); hold on;

title('Frequency content of Output Signal');
ylabel('Magnitude');
xlabel('frequency (Hz)');

sum1 = 0;
sum2 = 0;
for i = 1:numel(Puu)
    sum1 = sum1 + Pdd(i)^2;
    sum2 = sum2 + ((Pyy(i)) - (Pdd(i)))^2;
end
SNR = 10 * log10((sum1) / (sum2));
disp(SNR);
```