**Department of Electrical Engineering**

|  |  |
| --- | --- |
| Faculty Member: Dr. Wajahat Hussain  Course/Section: BEE6-B | Dated: 4-4-2017  Semester: 6th Semester |

**EE-330 Digital Signal Processing**

**Lab #7 Sampling and Quantization**

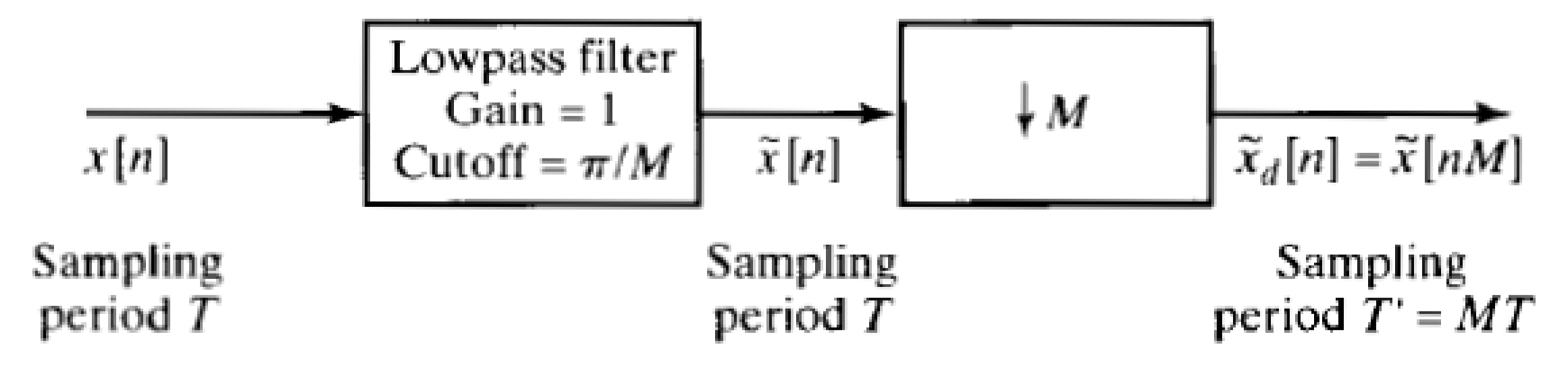
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Reg. no. | Report Marks  / 10 | Lab Quiz-  Viva Marks  / 5 | Total  / 15 |
| Saad Iqbal | 111394 |  |  |  |
| Usman Iqbal | 111393 |  |  |  |
| Abdullah Bin Asif | 111596 |  |  |  |

**Introduction:**

In this lab we will study audio signals. The process of signal acquisition and processing will be observed. Filtering and downsampling will be discussed.

We will also observe the effect on signal after downsampling.

**Change of Sampling Rate**

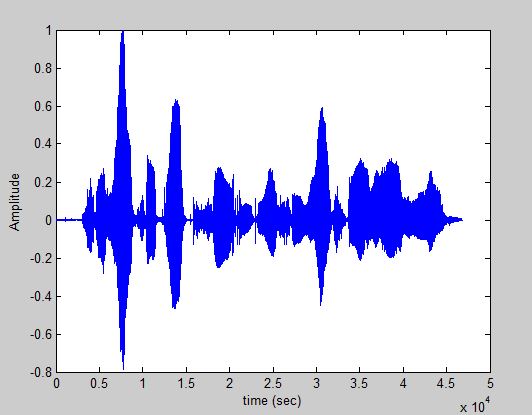


You are familiar with the above figure for downsampling. For a discrete-time signal to be sampled by the factor of *M*, you need to first pass the signal from the low pass filter with a certain cutoff frequency to avoid the frequency aliasing in the downsampled signal.

**TASK-1:**

You are given a speech signal. Consider it a discrete-time signal with the sampling frequency 𝑓𝑠 = 16 𝑘𝐻𝑧.

1. Load the signal in Matlab using the function *wavread.* Listen to the signal using *wavplay*.



*Figure 1 plot of original signal.*

1. Design a 6thP P order low-pass butterworth filter. Hint: see Matlab help for *butter* and *filter.* The butter command takes the normalized cutoff frequency (in the range 0-1) as an input argument where the maximum 1 means 𝑓𝑆/2 .

**Code**

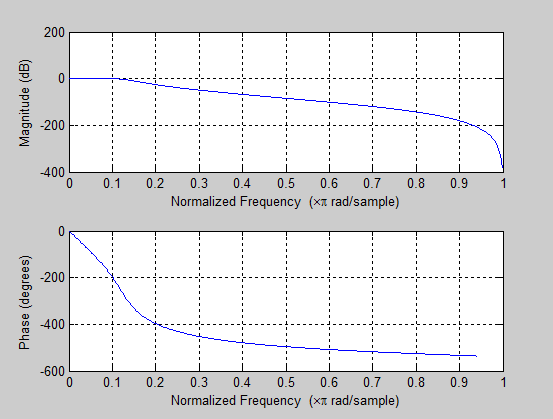
clc

[y,fs]=wavread('sample.wav');

fc=1000; %cut off frequency=1000 Hz

[b,a] = butter(6,fc/(fs/2));

freqz(b,a)



*Figure 2 Magnitude and phase response of filter*

1. Consider the maximum frequency of the speech signal 𝑓𝑁 = 𝑓𝑆/2. Apply the filter.

**Code:**

[y,fs]=wavread('sample.wav');

fc=1000; %cut off frequency=1000 Hz

[b,a] = butter(6,fc/(fs/2));

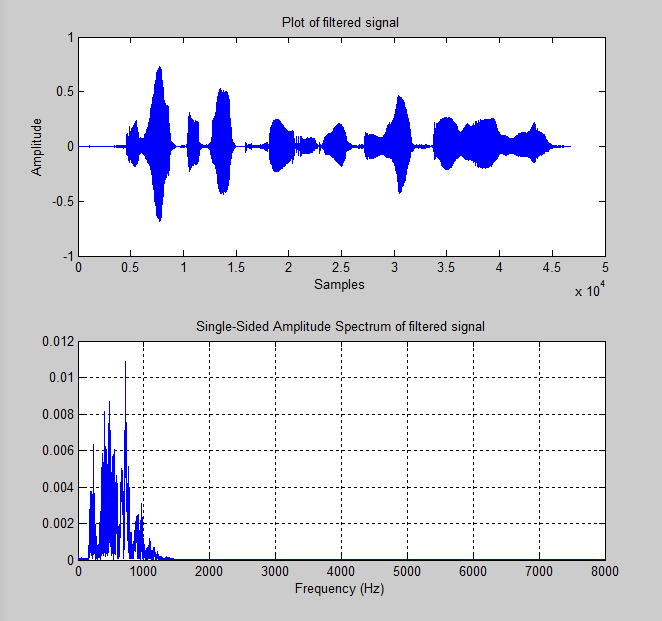
h = filter(b,a,y); % h is your filtered input signal

L=length(h);

NFFT = 2^nextpow2(L); % Next power of 2 from length of y

Y = fft(h,NFFT)/L;

f = fs/2\*linspace(0,1,NFFT/2+1); % Plot single-sided amplitude spectrum.



*Figure 3 signal and FFT plot after filtering, cut off frequency=10000Hz*

1. Now downsample the filtered signal by the factor of 2 i.e., *M*=2. Do this manually by picking up every alternative sample and storing it in a different array. (use a loop)

**Code:**

[y,fs]=wavread('sample.wav');

fc=1000; %cut off frequency=1000 Hz

[b,a] = butter(6,fc/(fs/2));

h = filter(b,a,y);

M=2; % Downsampled by 2

new\_h=h(1:M:end);

% new\_h is your downsampled filtered input signal

L=length(new\_h);

NFFT = 2^nextpow2(L);% Next power of 2 from length of y

Y = fft(new\_h,NFFT)/L;

f = fs/2\*linspace(0,1,NFFT/2+1);

% Plot single-sided amplitude spectrum.

subplot(2,1,1)

plot(new\_h)

title('Plot of downsampled filtered signal')

xlabel('Samples')

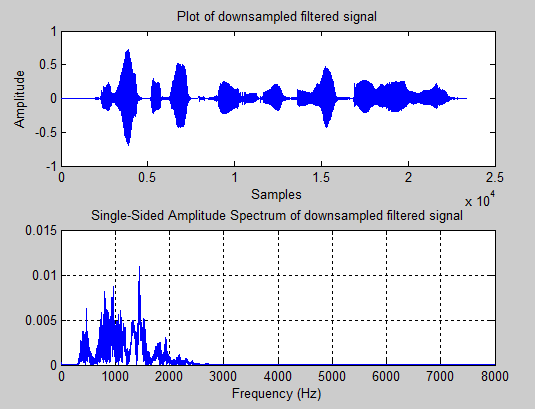
ylabel('Amplitude')

subplot(2,1,2)

plot(f,2\*abs(Y(1:NFFT/2+1))),grid on

title('Single-Sided Amplitude Spectrum of downsampled filtered signal')

xlabel('Frequency (Hz)')



*Figure 4: Down sampled signal*

**Comment:**

From the plots we can deduce that compression in time domain results in expansion in frequency domain.

1. See the Matlab help for the function *downsample*. Apply this function for downsampling the signal by the factors *M* = 3,5,10. Listen to the output signal in every case and prepare your conclusions. Also plot the spectrum of the input and output signal in a subplots for original and three cases for different *M*.

**Code:**

clear all

clc

[y,fs]=wavread('sample.wav');

fc=1000; %cut off frequency=1000 Hz

[b,a] = butter(6,fc/(fs/2));

h = filter(b,a,y);

%h is your input signal

L=length(h);

NFFT = 2^nextpow2(L); % Next power of 2 from length of h

Y = fft(h,NFFT)/L;

f = fs/2\*linspace(0,1,NFFT/2+1);

% Plot single-sided amplitude spectrum.

subplot(2,2,1)

plot(f,2\*abs(Y(1:NFFT/2+1))),grid on

title('Amplitude Spectrum of origianl filtered signal')

xlabel('Frequency (Hz)')

ylabel('Magnitude')

wavplay(h); % plays the sound

pause(1)

M=[3 5 10];

for i=1:3

h = downsample(h,M(i))

ydft = fft(h);

ydft = ydft(1:length(h)/2+1);

freq = 0:fs/length(h):fs/2

subplot(2,2,(i+1));

plot(freq,abs(ydft));

xlabel('frequency --->');

ylabel('Magnitude')

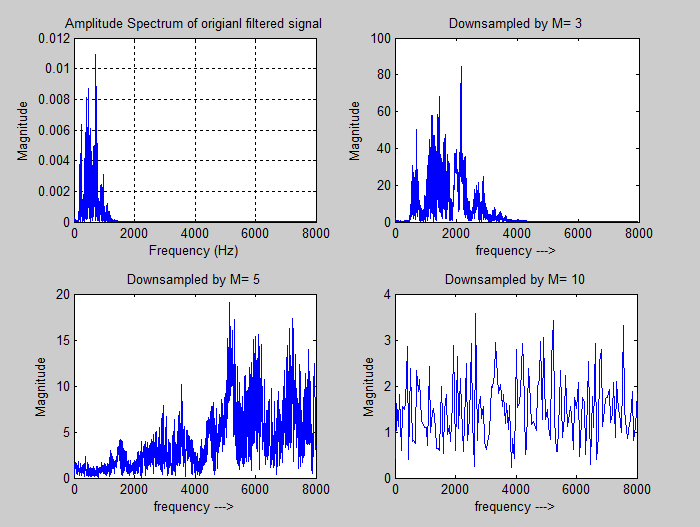
str=sprintf('Downsampled by M= %d',M(i));

title(str)

wavplay(h);

pause(1)

end



*Figure 5 FFT for different downsampled signals.*

**Comment:**

* After listening to the downsampled signals we concluded that it was hard to understand the downsampled signals at higher level of downsampling as the information gets lost due to compression.
* We also conclude that as we compressed the signal in time domain the signal gets expanded in the frequency domain.

1. For M = 10, avoid the anti-aliasing filter and directly downsample the speech to listen if there is any difference. Also plot the spectrum in subplot for input and output signal.

**Code:**

[y,fs]=wavread('sample.wav');

L=length(y);

NFFT = 2^nextpow2(L);% Next power of 2 from length of h

Y = fft(y,NFFT)/L;

f = fs/2\*linspace(0,1,NFFT/2+1);

% Plot single-sided amplitude spectrum.

subplot(2,1,1)

plot(f,2\*abs(Y(1:NFFT/2+1))),grid on

title('Amplitude Spectrum of origianl signal')

xlabel('Frequency (Hz)')

ylabel('Magnitude')

wavplay(y); % plays the sound

pause(1)

M=10; %downsampling factor

new\_h = downsample(y,M)

L=length(new\_h);

NFFT = 2^nextpow2(L);% Next power of 2 from length of h

Y = fft(new\_h,NFFT)/L;

f = fs/2\*linspace(0,1,NFFT/2+1);

% Plot single-sided amplitude spectrum.

subplot(2,1,2)

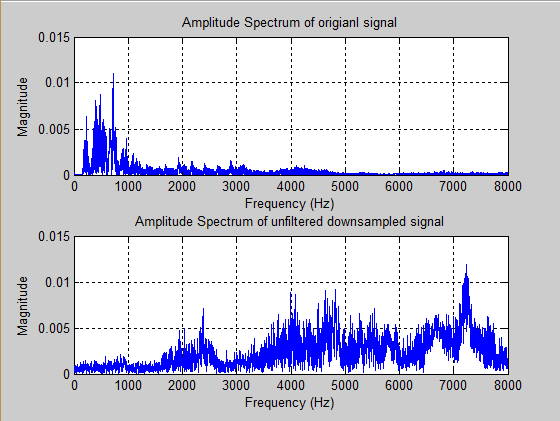
plot(f,2\*abs(Y(1:NFFT/2+1))),grid on

title('Amplitude Spectrum of unfiltered downsampled signal')

xlabel('Frequency (Hz)')

ylabel('Magnitude')

wavplay(new\_h); % plays the sound



*Figure 6 FFT of original and downsamped signal without anti-aliasing filter*

**Comment:**

U

* It was difficult to comprehend the downsampled signal due to the loss of data after compression and from the effects aliasing.

**Conclusion:**

* From this lab we got to know about the practical understanding of an audio signal.
* We analyzed the signal from its Spectrum plot.
* We learned to design different types of filter using *butter* command.
* The effects of aliasing were observed.
* Downsampling of the signal was also done.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_