Experiment 2

VOICE AND AUDIO DIGITIZATION AND SAMPLING RATE CONVERSION

I Introduction

In a microphone, physical sound pressure waves are converted to corresponding electrical signals by acoustic transducer, such as microphone or phonograph cartridge. The electrical output of the transducer is called an analog signal because the electrical signal is analogous to (representative of) the pressure patterns of the sound wave that created it. Sound signals are most often represented in the form of two-dimensional wave patterns whose Y-axis is the intensity or amplitude of the signal and X-axis represents the passage of time. Figure 2.1 shows an analog waveform of a series of sound waves from a chime, changed into an analog voltage by a microphone and amplified to a maximum level (amplitude) of +/- 0.5 volt, also called an amplitude of 1 Volt peak-to-peak, abbreviated V(pp) or Vpp.

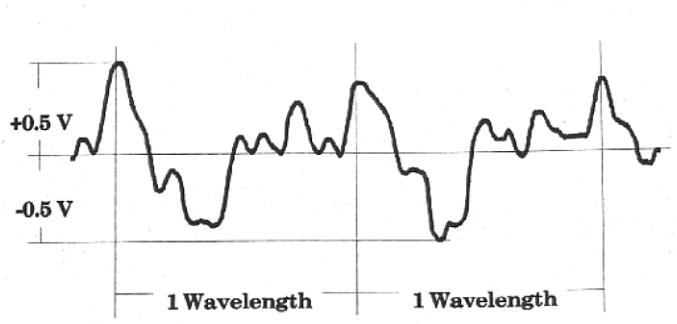


Fig 2.1 A typical sound waveform

The frequency of a wave is determined by the time that elapses between repetitions, called the *wavelength*. Most sound waves do not repeat precisely, but you can observe a pattern in the waveforms created by most musical instruments. The wavelength of an electrical sound is expressed in milliseconds or microseconds, abbreviated λ. The frequency, f, measured in Hz (Cycles per second), is determined as the reciprocal of λ, i.e. $f = 1/\lambda$.

Human speech or music from musical instruments can be decomposed into a basic wave and many additional waves. The additional waves superimposed on the basic wave are called overtones. Overtones are higher-frequency waves, with frequencies that are multiples of the basic wave (called harmonics), giving sounds their characteristic waveforms of the human voice and acoustic musical instruments. When converting a sound signal into digital signals, the required sampling rate depends on the frequencies of the overtones existing in the signals.

In this experiment you will have the opportunity to play with the sampled signals obtained with different sampling rates, which offer different qualities of voice and audio reproduction at different storage requirements.

II Theory

II.1 Sampling of a Continuous Signal & the Nyquist Sampling Theory

Analog sound signals are continuous, usually diminishing in amplitude, until the source of the sound is extinguished. Computers, on the other hand, store their data in digital form: a stream of bits representing 1s and 0s. Digital data are fundamentally discontinuous in nature, because the 1 or 0 value of digital data is valid only at a particular instant of time. Thus, a continuous analog sound signal must be converted into a discontinuous digital form so that the computer can store or process the sound. The digital data must be translated back to an analog form so that you can hear it through a sound system. The two-way conversion between analog and digital signals is the basic function of all audio adapter cards or sound cards.

II.1.1 Periodic Sampling (Analog to Digital Conversion)

The typical method of obtaining a discrete-time representation of a continuous-time (analog) signal is through periodic sampling, wherein a sequence of samples x[n] is obtained from a continuous-time signal $x_c(t)$ according to the relation

$$x[n] = x_c(nT), \qquad -\infty < n < \infty . \tag{2.1}$$

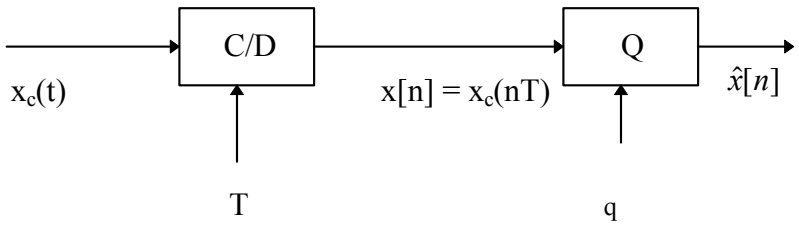


Fig. 2.2 An ideal analog-to-digital (A/D) converter.

In Eq. (2.1), T is the sampling period, and its reciprocal, $f_s = 1/T$, is the sampling frequency, in samples per second (usually denoted by Hz). We refer to a system that implements the operation of Eq. (2.1) as an ideal continuous-to-discrete (C/D) converter. In order to store the sampled values by a computer which has a finite precision, the continuous value has to be quantized into one of a few pre-scaled values. This operation of sampling and quantization is what is being done in an analog-to-digital (A/D) converter, and is depicted in Fig. 2.2. A sound card contains an analog to digital converter and a digital to analog converter, which performs the opposite function. Most sound cards store either 8 bits per sample or 16 bits per sample for higher quality sound.

II.1.2 The Nyquist Sampling Theorem

The sampling theory tells us how fast we must sample a signal to obtain a good representation of the original signal. Intuitively, we know that if a signal changes very fast, then we must also samples at a close interval, so that we do not miss any intermediate variations. A good example is monitoring the stock market vs. monitoring the weather. Because the stock market changes very rapidly, it's customary to respond it every few minutes. On the other hand, with weather, once every hour will be sufficient.

Now, let us look at the theory how exactly we should determine T.

Nyquist Sampling Theorem: Let $x_c(t)$ be a bandlimited signal and $X_c(j\Omega)$ be its Fourier Transform, which satisfies

$$X_c(j\Omega)=0 \qquad for\ |\Omega| > \Omega_N \quad . \tag{2.2}$$

Then $x_c(t)$ is uniquely determined by its samples $x[n]= x_c(nT)$, $n=0, \pm 1, \pm 2, \ldots\ldots$, if the sampling interval T or sampling frequency Ω_s satisfies

$$\Omega_s = \frac{2\pi}{T} > 2\Omega_N \quad . \tag{2.3}$$

The above result is first discovered by Nyquist, known as the Nyquist sampling theorem. The frequency $2\Omega_N$ that must be exceeded by the sampling frequency is called the Nyquist sampling rate.

To prove the above theorem, let us express $X\ (e^{j\omega})$, the discrete-time Fourier transform of the sequence x[n], in term of $X_c(j\Omega)$, the continuous Fourier transform of $x_c(t)$. To this end, let us consider the following impulse train signal:

$$\begin{aligned} x_s(t) &= x_c\ (t)\cdot \sum_{n=-\infty}^{\infty} \delta\,(t-nT) \\ &= \sum_{n=-\infty}^{\infty} x_c\,(nT)\,\delta\,(t-nT) \quad . \end{aligned} \tag{2.4}$$

It can be shown that the Fourier transform of $x_s(t)$ is

$$X_s\,(j\Omega) = \frac{1}{T}\sum_{k=-\infty}^{\infty} X_c\,(j\Omega - k\,\Omega_s) \quad , \tag{2.5}$$

then, by definition,

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]\,e^{-jn\omega} \tag{2.6}$$

$$X_s(j\Omega) = \int_{-\infty}^{\infty} x_s(t) e^{-j\Omega t}\, dt = \sum_{n=-\infty}^{\infty} x_c(nT) e^{-jn\Omega T} \quad , \tag{2.7}$$

it follows that

$$X(e^{j\omega}) = X_s(j\Omega)\big|_{\Omega=\frac{\omega}{T}} \quad . \tag{2.8}$$

Consequently, from Eqs. (2.5) and (2.8)

$$X(e^{j\omega}) = \frac{1}{T}\sum_{k=-\infty}^{\infty} X_c(j\frac{\omega}{T} - j\frac{2\pi k}{T}) \quad . \tag{2.9}$$

From Eq. (2.9) we see that $X(e^{j\omega})$ is the summation of scaled version of $X_c(j\Omega)$ and its shifted versions. The frequency scaling is specified by $\Omega=\frac{\omega}{T}$, while the shifts are equal to multiples of the sampling frequency $\Omega_s = \frac{2\pi}{T}$. The relation in Eq. (2.9) is illustrated in Figs. 2.5(a) ~ 2.5(c).

II.1.3 Reconstruction of A Bandlimited Signal from its Samples (Digital to Analog Conversion)

From Fig. 2.5, if the original continuous signal is bandlimited so that $X_c(j\Omega) = 0$ for $\Omega > \Omega_N$, where $\Omega_N \le \frac{\Omega_s}{2}$, then it is possible to recover $X_c(j\Omega)$ from $X(j\omega)$. More specifically, we can first form the impulse train signal $x_s(t)$,

$$x_s(t) = \sum_{n=-\infty}^{\infty} x[n]\delta(t - nT) \quad . \tag{2.10}$$

We can then apply an ideal low-pass filter with the cut-off frequency $\Omega_c = \frac{\Omega_s}{2} = \frac{\pi}{T}$, i.e.

$$H_r(j\Omega) = \begin{cases} T & |\Omega| < \Omega_c \\ 0 & otherwise \end{cases} \tag{2.11}$$

to $x_s(t)$. If $X_c(j\Omega)$ is bandlimited as defined above, then the filtered signal has exactly the same Fourier transform as $X_c(j\Omega)$, i.e.

$$X_r(j\Omega) = X_s(j\Omega) \cdot H_r(j\Omega) = X_c(j\Omega) \quad .$$

In the time domain, the ideal reconstruction filter $h_r(t)$ is given as

$$h_r(t) = \frac{\sin \pi t / T}{\pi t / T} \quad . \tag{2.12}$$

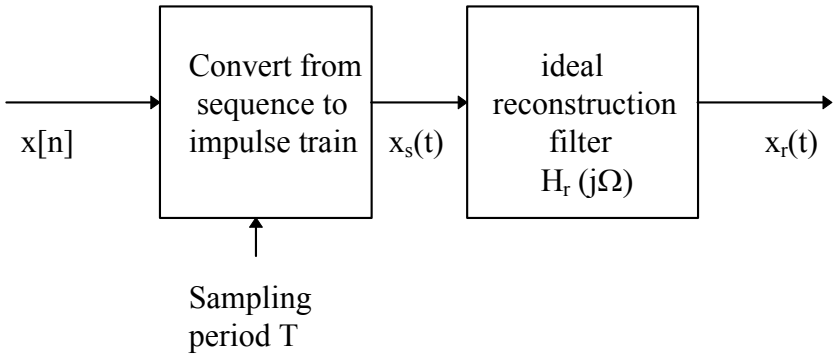
The reconstructed signal is

$$x_r(t) = x_s(t) * h_r(t) = \sum_{n=-\infty}^{\infty} x[n] h_r(t - nT) = \sum_{n=-\infty}^{\infty} x[n] \frac{\sin[\pi(t - nT)/T]}{\pi(t - nT)/T} \quad . \qquad (2.13)$$

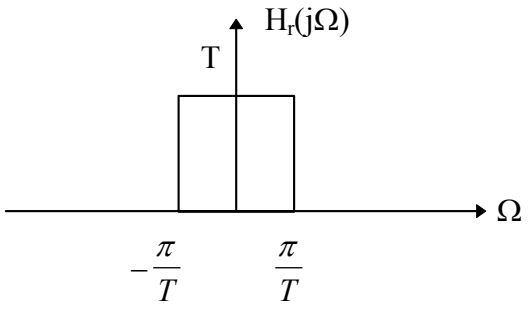
Fig. 2.3 shows the block diagram of this signal reconstruction process.

From the above result, samples of a continuous-time bandlimited signal taken frequently enough (i.e. $\Omega_s > 2\Omega_N$) are sufficient to represent the signal exactly in the sense that the signal can be recovered from the samples and from knowledge of the sampling period.

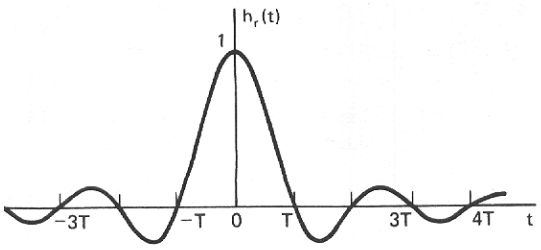
In reality, the ideal low-pass filter is not realizable, and must be approximated. Furthermore, a real signal may have a bandwidth too high that cannot be accommodated by a sampling system. In practice, for a given sampling frequency Ω_s, one should first apply a prefilter which is low pass with a cut-off at $\Omega_c \leq \Omega_s/2$ so that $\Omega_N \leq \Omega_s/2$.



(a)



(b)



(c)

Fig. 2.3 (a) An ideal bandlimited signal reconstruction system. (b) Frequency response of an ideal reconstruction filter. (c) Impulse response of an ideal reconstruction filter.

II.2 Downsampling of Digital Signals

The sampling rate of a sequence can be reduced by "sampling" it, i.e., by defining a new sequence

$$x_d[n] = x[nM] = x_c(nMT) \quad . \tag{2.14}$$

From Eq. (2.14) it is clear that $x_d[n]$ could be obtained directly from $x_c(t)$ by sampling with period T' = MT. Furthermore, if $X_c(j\Omega) = 0$ for $|\Omega| > \Omega_N$, then $x_d[n]$ is an exact representation of $x_c(t)$ if $\Omega_s' = \frac{2\pi}{T'} = \frac{1}{M}\Omega_s > 2\Omega_N$. That is, the sampling rate can be reduced by a factor of M, if the original sampling rate was at least M times the Nyquist rate. In general, to avoid aliasing the bandwidth of the sequence should be first reduced by a factor of M by discrete-time filtering. The block diagram of a down-sampler is shown in Fig 2.4 .

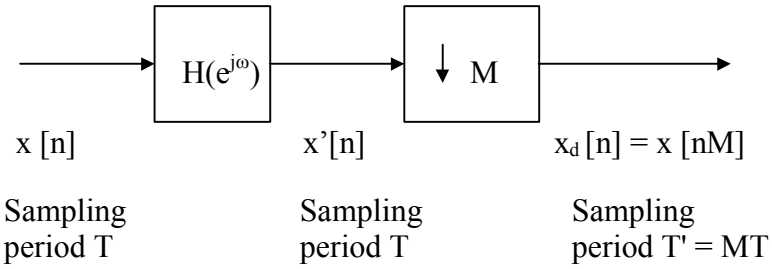


Fig 2.4 Down-sampling by a factor of M, where $H(e^{j\omega})$ is a prefilter. In the ideal case, it should be a low-pass filter with cut-off frequency at $\Omega_c = \pi/M$.

To determine the relationship between the Fourior transform of x[n] and $x_d[n]$, first recall the discrete-time Fourier transform of $x[n] = x_c(nT)$ is

$$X(e^{j\omega}) = \frac{1}{T}\sum_{k=-\infty}^{\infty} X_c\left(j\frac{\omega}{T} - j\frac{2\pi k}{T}\right) \quad . \tag{2.15}$$

Similarly, the discrete-time Fourier transform of $x_d[n] = x[nM] = x_c(nT')$ with T' = MT is

$$X_d(e^{j\omega}) = \frac{1}{T'}\sum_{r=-\infty}^{\infty} X_c\left(j\frac{\omega}{T'} - j\frac{2\pi r}{T'}\right) = \frac{1}{MT}\sum_{r=-\infty}^{\infty} X_c\left(j\frac{\omega}{MT} - j\frac{2\pi r}{MT}\right) \quad . \tag{2.16}$$

Note that the summation index r in Eq.(2.16) can be expressed as

$$r = i + kM \ , \tag{2.17}$$

where k and i are integers such that $-\infty < k < \infty$ and $0 < i < M\text{-}1$. Clearly r is still an integer ranging from $-\infty$ to ∞, but now Eq. (2.17) can be expressed as

$$X_d(e^{j\omega}) = \frac{1}{M}\sum_{i=0}^{M-1}\left[\ \frac{1}{T}\sum_{k=-\infty}^{\infty} X_c\left(j\frac{\omega}{MT} - j\frac{2\pi k}{T} - j\frac{2\pi i}{MT}\right)\ \right] . \qquad (2.18)$$

The term inside the square brackets in Eq. (2.18) is recognized from Eq. (2.15) as

$$X(e^{j(\omega-2\pi i)/M}) = \frac{1}{T}\sum_{k=-\infty}^{\infty} X_c\left(j\frac{\omega-2\pi i}{MT} - j\frac{2\pi k}{T}\right) . \qquad (2.19)$$

Thus we can express Eq. (2.18) as

$$X_d(e^{j\omega}) = \frac{1}{M}\sum_{i=0}^{M-1} X(e^{j(\omega-2\pi i)/M}) . \qquad (2.20)$$

This is illustrated in Fig. 2.5 for M = 2 and in Fig 2.6 for M = 3. It can be seen that when M = 2, downsampling does not cause overlapping of the original spectrum. On the other hand, when M = 3, overlapping between repeated spectra (referred to as *aliasing*) occurs. In order to avoid aliasing, prefiltering is required before downsampling.

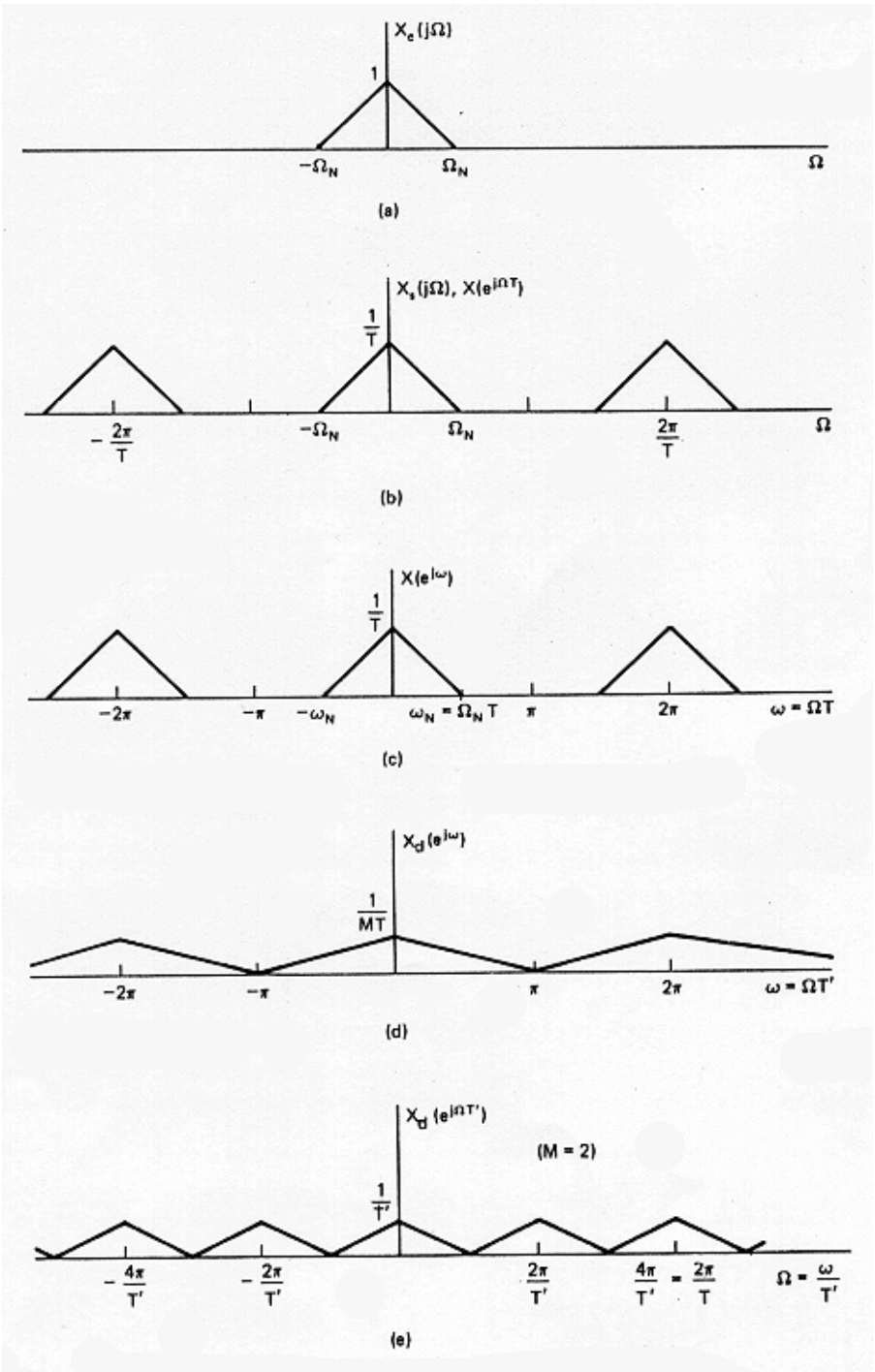


Fig. 2.5 Frequency-domain illustration of downsampling (M = 2)

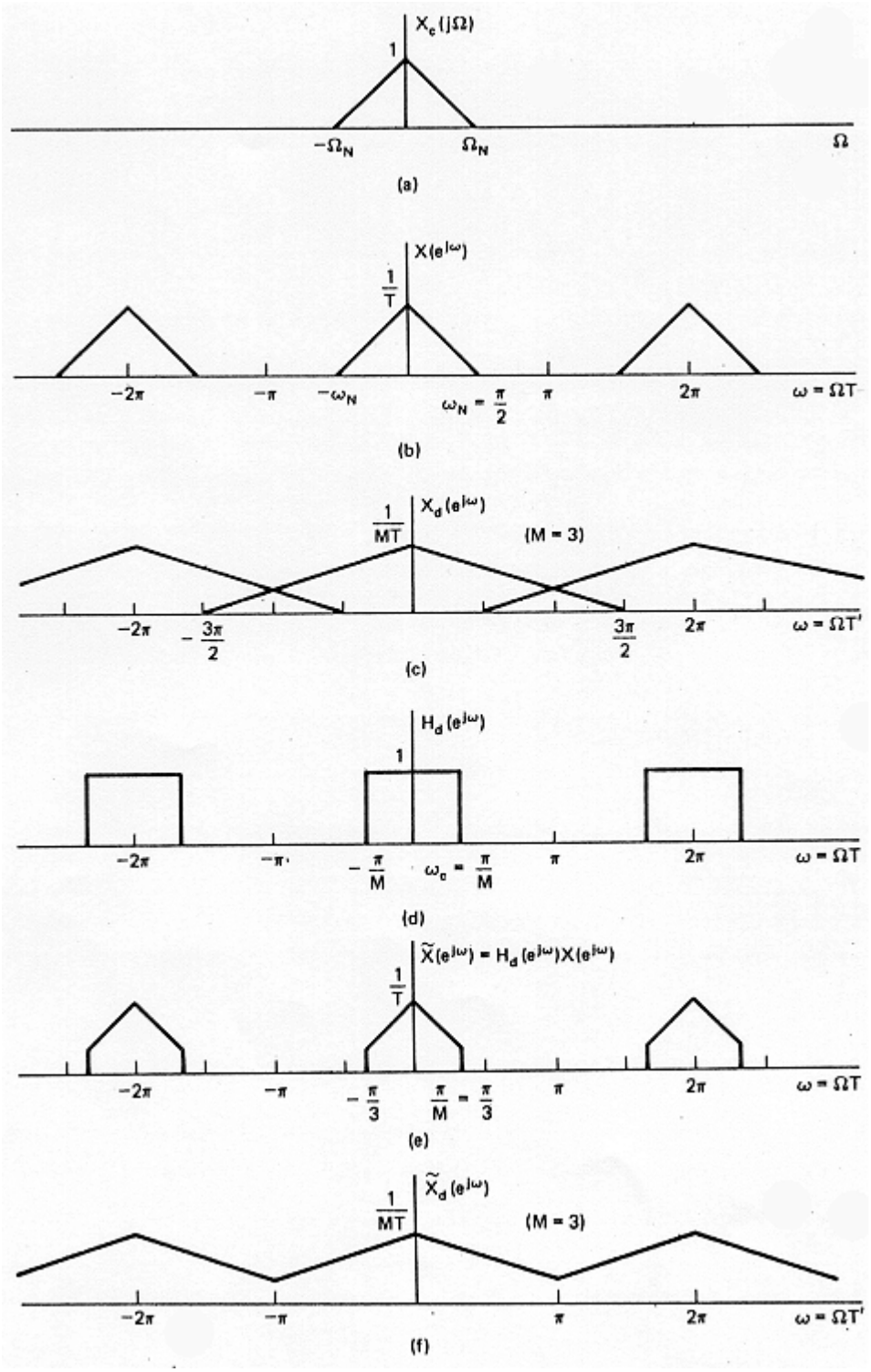


Fig. 2.6 Downsampling by a factor of M=3. (a)-(c) without prefiltering so that the downsampled signal suffer from aliasing. (d)-(f) Downsampling with pre-filtering to avoid aliasing.

II.3 Upsampling of Digital Signals

We have seen the reduction of the sampling rate of a discrete-time signal by an integer factor involves sampling the sequence in a manner analogous to sampling a continuous-time signal. Not surprisingly, increasing the sampling rate involves operations analogous to D/C conversion. To see this, consider a signal x[n] whose sampling rate we wish to increase by a factor of L. If we consider the underlying continuous-time signal $x_c(t)$, the objective is to obtain samples

$$x_i[n] = x_c(nT'), \tag{2.21}$$

where T' = T/L, from the sequence of samples

$$x[n] = x_c(nT). \tag{2.22}$$

We refer to the operation of increasing the sampling rate as upsampling.

It is clear from Eqs. (2.21) and (2.22) that

$$x_i[n] = x[n/L] = x_c(nT/L), \quad n=0, \pm L, \pm 2L, \ldots\ldots \tag{2.23}$$

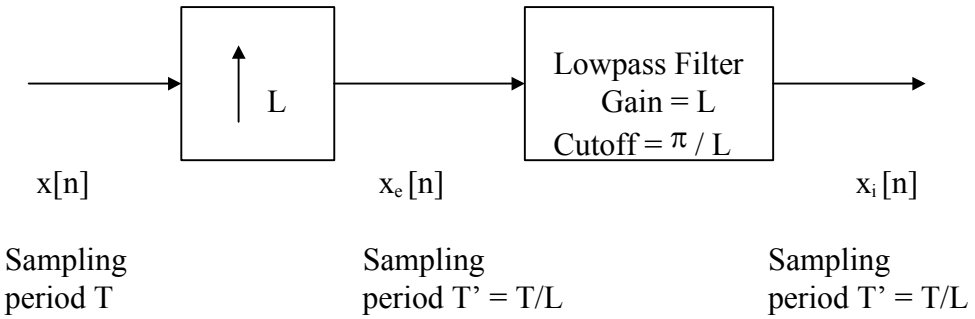


Fig. 2.7 Up-Sampling (Interpolation) Process

Figure 2.7 shows a system for obtaining $x_i[n]$ from x[n] using only discrete-time processing. The system on the left is called a *sampling rate expander* or simply an *expander*. Its output is

$$x_e[n] = \begin{cases} x[n/L], & n = 0, \pm L, \pm 2L, \ldots\ldots \\ 0, & \textit{otherwise} \end{cases} \tag{2.24}$$

or, equivalently,

$$x_e[n] = \sum_{k=-\infty}^{\infty} x[k]\delta[n - kL] \quad . \tag{2.25}$$

The Fourier transform of $x_e[n]$ can be expressed as

$$X_e(e^{j\omega}) = \sum_{n=-\infty}^{\infty}\Big(\sum_{k=-\infty}^{\infty} x[k]\delta\,[n - k\,L]\Big) e^{-j\omega n} \tag{2.26}$$

The above relation is illustrated in Fig 2.8(b) and 2.8(c). To recover $x_i[n]$ from $x_e[n]$, one needs to apply an ideal low pass filter with cut-off frequency at $\Omega_c = \pi/L$ and with gain = L, as shown in Fig 2.8(d) and 2.8(e) .

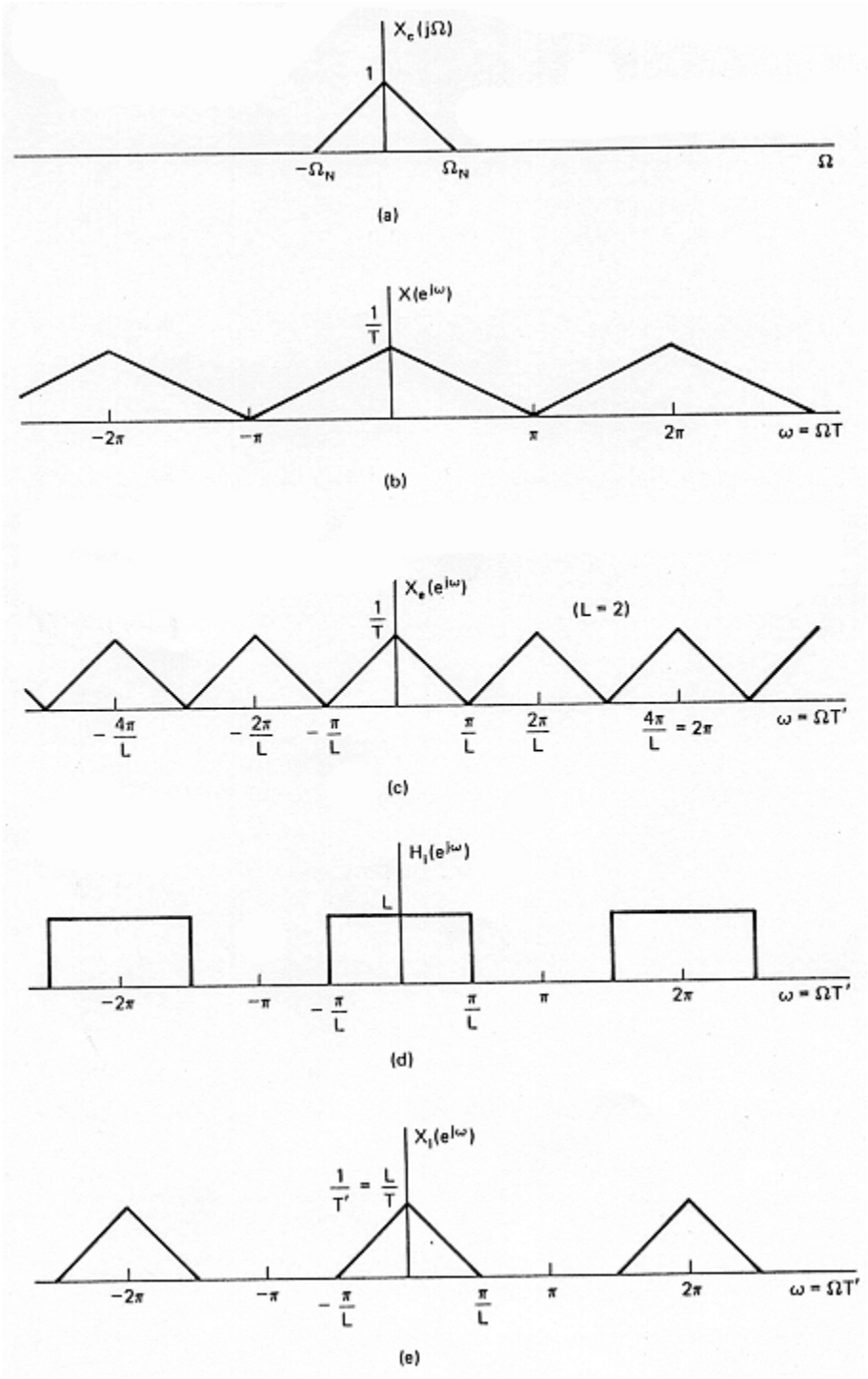


Fig. 2.8 Frequency-domain illustration of interpolation

III Experiments

III.1 Sampling Audio at Different Frequencies

In this experiment, you will record audio files from your own speech, the CD-ROM, and the MIDI player.

1) Record your own speech.

   a) Make sure the connection of the microphone is correct, i.e the mircophone is connected to the "MIC_in" in the sound card, in the rear side of the computer.

   b) Open three Sound Recorder windows. First click on File_Properties_Convert. Then choose 8,000Hz, 8 bit, Mono 8KB/s. Record 5 seconds of speech and save it as rec8.wav under your own directory.

   c) Record 11,025Hz, 8bit, Mono 11KB/s and 22,050Hz, 8bit, Mono 22KB/s for the second and third Sound Recorder. Save files as rec11.wav and rec22.wav.

   d) Play the sounds one by one and compare the sound qualities.

2) Record the audio from CD-ROM.

Play the SB 16 CD-ROM by using the CD Player. Record 5 seconds of CD Audio at 11k, 22k, and 44k at 8 bits/sample. Save the files as cd11.wav, cd22.wav, and cd44.wav under your own directory.

3) Record the MIDI audio for the computer.

Play the MIDI file by using the Media Player. Record 5 seconds of MIDI music at 11k, 22k, and 44k. Then save files as midi11.wav, midi22.wav, and midi44.wav to your own directory.

4) Comment on all the sound qualities from different sampling rates and sources.

5) Repeat 1 to 4 using 16 bits per sample instead of 8 bits per sample.

III.2 Sound Processing by MATLAB

In this experiment, you are required to write Matlab Scripts to accomplish the solution.

1) Take a sound signal sampled at 22 kHz, 8 bits, reduce the sampling rate by half, without prefiltering, and then using a linear interpolation filter to interpolate it back to the original captured sampling rate.

2) Display the spectra of downsampled and interpolated signals, and compare them to the original signal. Calculate the mean squares error (MSE) between the reconstructed and original signal at 22 kHz. The MSE between two length N signals x(n) and y(n) is defined as

$$MSE = \sum_{n=1}^{N} [\, x(n) - y(n) \,]^2 \,/\mathrm{N} \ .$$

3) Repeat 1) & 2) for the following:

   a) Using an averaging filter (you can choose the length of the averaging filter) for prefiltering, and

   b) Use fir1() function from Matlab to design a better filter. Make use of the interp() function for the interpolation. Try different length for the interpolation filter. Display the spectra for the pre- and post- filters, in addition to those of the downsampled interpolated signal.

4) (Optional): Repeat 1 -- 3 for a signal sampled at 11 kHz.

5) Print out all the results.

For 1) & 2) three sample Matlab scripts are provided in Appendix A (sp.m, sp1.m, and spfilter.m) . You should be able to accomplish the others by modifying these scripts slightly. Hint: filtering of a sequence X( ) by a filter can be accomplished by conv( ) . You can use the online help by simply type the help topics. For example : help conv.

In short , you will have to modify those programs in Appendix A to perform

   a) No-Prefiltering, Downsample by 2, Interploation back to the original size.

   b) Averaging filter , Downsample by 2, Interploation back to the original size.

   c) fir1() , Downsample by 2, Interploation back to the original size.

Hints: You should copy all the M-files from h:\el514\exp2 to your own directory before you try to use them. You can following the following steps:

   a) Launch Matlab by double_clicking the Matlab icon.

   b) Copy files from server to your working directory

   c) Change directory to your working directory

   d) RUN : sp1('chimes.wav');

IV Report

Include all .m file, your filters and output figures in your report.

Answer the following questions in your report:

1) If the input signals is $\sin(2\pi f t)$, where $f = 6$ kHz , and the sampling frequency is 8 kHz. What will be the sampled signals? What will be the reconstructed signal using an ideal low pass filter with cut-off frequency at 4 kHz. (Derive the problem and write down all the steps in your report.)

2) Comment on the quality of those sampled files from different sampling rates and bits per sample. Comment on the differences between speech and audio in terms of required sampling rates.

3) Compare the interpolated sounds after being downsampled by half, the downsampled signals, and the original signals. Explain why the reconstructed signals are better in some cases.

V References

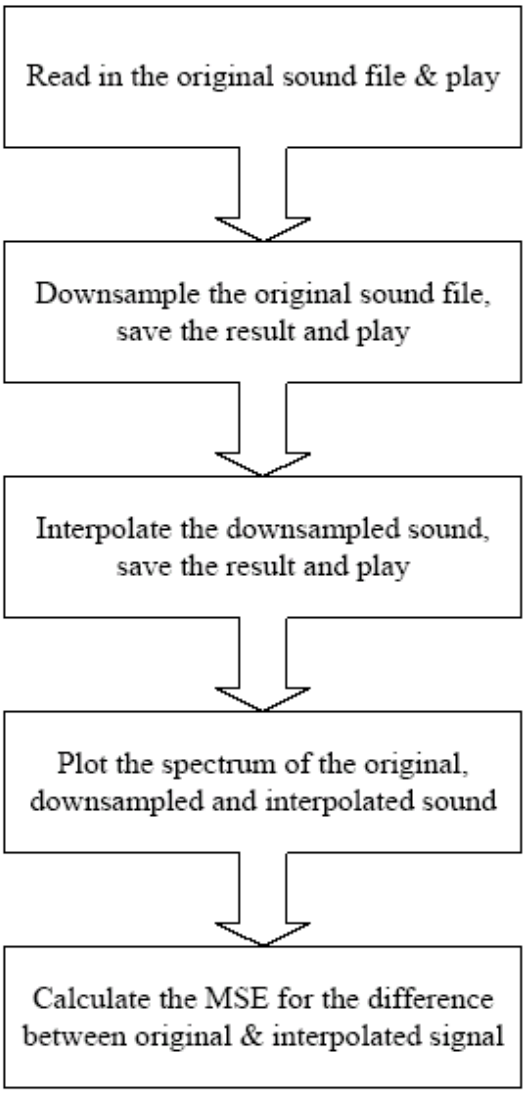
1) The MathWorks Inc., Matlab User's Guide, 1993. MATLAB USER'S GUIDE, 1993.

2) The MathWorks Inc., MATLAB REFERENCE GUIDE , 1992.

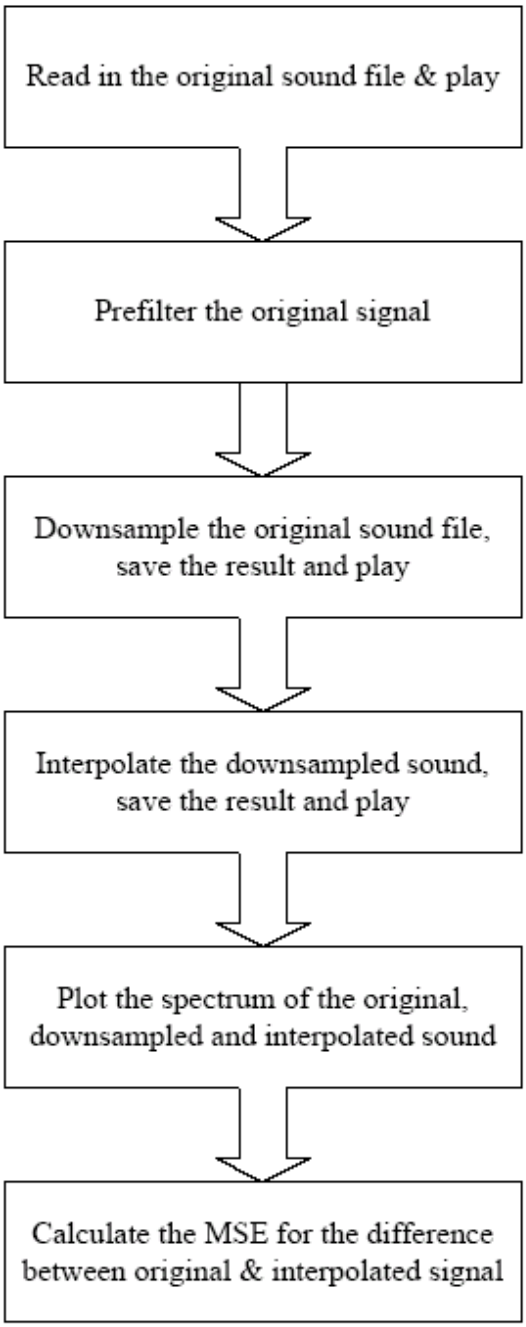
3) Wilsky and Oppenheim, Signals & System, Chapter 8.

Appendix A

1. Flow charts for sample programs.
2. Matlab scripts for down- and up- sampling a sound file.



1. block diagram for sp1.m



2. block diagram for spfilter.m

****

sp1.m (No pre-filtering , Interp() for post-filtering, Normalized output)

```
% Downsample sound file and save the result
% Then Interpolate it back to the original sampling rate
% Display the spectrums
% Usage : sp1('Infile')

function[]=sp(InFilename);

fprintf('\n The original sound \n')
[y,Fs]=wavread(InFilename);

% Play it
sound(y,Fs);

if rem(length(y),2) ~=0
   y = y(1:length(y)-1);
end

% Downsample
fprintf('\n The downsampled sound \n')
x=y(1:2:length(y));

sound(x,Fs/2);

% Save the result as down.wav
wavwrite(x,Fs/2,'down.wav');

%Interpolation
fprintf('\n The interpolated sound \n')
z=interp(x,2);

sound(z,Fs);

% Save as up.wav
wavwrite(z,Fs,'up.wav');

%Spectrums
py=spectrum(y);
px=spectrum(x);
pz=spectrum(z);
pxx=px(:,1);


xss=1.0/length(py(:,1)):1.0/length(py(:,1)):1;
```

```
l=length(pxx(1:2:length(pxx)));

xss1=1.0/l:1.0/l:1;

subplot(3,1,1),semilogy(xss,py(:,1));
xlabel('Original Signal');
h=axis;

subplot(3,1,2),semilogy(xss1,pxx(1:2:length(pxx)));
xlabel('Downsampled Signal');
axis(h);

subplot(3,1,3),semilogy(xss,pz(:,1));
xlabel('Interpolated Signal');
axis(h);

% Calculate the MSE
D=y-z;
MSE=mean (D.^2);
fprintf('\n Mean Square Error = %g\n\n',MSE )
```

****

spfilter.m (Averaging for pre-filtering , Linear for post-filtering)

```
% Downsample sound file and save the result
% Then Interpolate it back to the original sampling rate
% Display the spectrums
% Incorporate pre-filter & post-filter
% Usage : spfilter('Infile')

function[]=spfilter(InFilename);

fprintf('\n The original sound \n')
[y,Fs]=wavread(InFilename);

% Play it
if rem(length(y),2) ~=0
   y = y(1:length(y)-1);
end

sound(y,Fs);

%Pre-filter the original signal
fprintf('\n Pre-filtering ...    &   ')
f=[1 1 1 1 1];
```

```
f=f/5;
yy=conv(y,f);

        %Prune to same length
        n=length(f)/2;
        yy=yy(n:length(yy)-n);

% Downsample
fprintf(' The downsampled sound \n')
x=yy(1:2:length(yy));

sound(x,Fs/2);

% Save the result as down.wav
wavwrite(x,Fs/2,'down.wav');



%Interpolation (Post-filtering)
%A simple average filter

fprintf('\n The interpolated sound \n')
z=zeros(1,2*length(x));
for i=1:length(x);
        z(2*i)=x(i);
        if i<length(x)
                z(2*i+1)=(x(i)+x(i+1))/2;
        end
end


sound(z,Fs);

% Save as up.wav
wavwrite(z,Fs,'up.wav');

%Spectrums
py=spectrum(y);
px=spectrum(x);
pz=spectrum(z);
pxx=px(:,1);

xss=1.0/length(py(:,1)):1.0/length(py(:,1)):1;
l=length(pxx(1:2:length(pxx)));

xss1=1.0/l:1.0/l:1;
```

```
subplot(3,1,1),semilogy(xss,py(:,1));
xlabel('Original Signal');
h=axis;

subplot(3,1,2),semilogy(xss1,pxx(1:2:length(pxx)));
xlabel('Downsampled Signal');
axis(h);

subplot(3,1,3),semilogy(xss,pz(:,1));
xlabel('Interpolated Signal');
axis(h);


% Calculate the MSE

D=y-z';
MSE=mean(D.^2);
fprintf('\n Mean Square Error = %g\n\n',MSE )
```