भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology
Hyderabad

Department of Electrical Engineering
IIT Hyderabad

ID1370 – Digital Signal Processing

Lab Exercise - 01 11 August 2010

Digitization of Signals – Sampling and Quantization

1 Objectives

Digital signal processing is the analysis of signals using a digital machine/general purpose computer. In order to make it possible, signals must be represented in digital form, i.e., a sequence of numbers. However, most often natural signals are continuous (analog) and have to be digitized before processing.

The process of digitization, usually, involves two stages:

- *Sampling:* Discretization in time-domain, i.e., the analog signal is observed at discrete instals of time.
- *Quantization:* Discretization in the amplitude domain, i.e., the transformation of measurements into finite-precision numbers, such that they can be represented in computer memory.

The objective of this experiment is to investigate the effects of sampling and quantization on some of the natural signals. The major objectives of this experiment are as follows:

- Synthesizing sinusoidal signals, and observing effect of sampling on sinusoids.
- Effects of sampling on audio signals and speech signals.
- Effects of quantization on audio and speech signals.
- Effect of sampling on digital images.
- Effect of quantization on digital images.

2 Sinusoidal Signals

In this experiment, you have to take a closer look at the behaviour of discrete-time sinusoids from a sampling view point. Often a discrete-time signal is produced by sampling a continuous time (analog) signal, such as a constant frequency sinusoid. The relation between the frequency of the sinusoid, and the sampling frequency is the essence of Nyquist-Shannon theorem, which requires that the sampling frequency should be at least twice the highest frequency in the signal for perfect reconstruction. In general a continuous-time sinusoid is expressed as

$$x(t) = A\cos(2\pi f_0 t + \phi) \tag{1}$$

where A is the amplitude, f_0 is the frequency of the sinusoid in Hertz, t is continuous time, and ϕ is the phase. Since this signal is an analog signal, there are infinite values in any finite interval of time. So we cannot represent a continuous time sinusoid in digital computer. A continuous time sinusoid has to be necessarily digitized in order to represent it in a digital computer.

In this experiment, we synthesize discrete-time sinusoidal signals by measuring the analog signal, or in our case *e*valuating the sinusoid, at constant intervals of time. This is called uniform sampling, and the time interval T_s is called the sampling period. The sampling frequency is therefore $F_s = \frac{1}{T_s}$. The discrete-time sinusoid that is obtained by sampling the continuous time sinusoid in 1 at discrete instants of time $t = nT_s$ is given by

$$x[n] = x(nT_s) = A\cos(2\pi f_0 nT_s + \phi) = A\cos\left(2\pi \frac{f_0}{f_s} n + \phi\right) \tag{2}$$

It is important to notice that the signal, thus generated, is a sequence of numbers, and not a continuous curve. The discrete-time signal is not defined in between any two successive time indexes, say k and $k+1$. It is wrong to think that the value of the discrete-time signal in between two successive time indexes is zero.

Exercise: Discrete-time sinusoid signals

(a) Keeping a fixed sampling frequency of $f_s = 8$ kHz, generate sinusoids with frequency $f_0 = 1, 2, 3, 3.5, 4, 5, 6$ and 7 kHz. Plot the sinusoids on separate figures. Why are some of these plots similar? Do they reflect the true frequency content, i.e. the frequency of the analog sinusoids?

(b) Analog sinusoids are pure frequency signals, i.e. their continuous Fourier transform contains an impulse at their precise frequency, and nothing elsewhere. Having that in mind, and from the results in part (a), what can you suggest has happened in the frequency domain of the sampled signals at frequencies higher than half the sampling frequency? Can you give some Mathematical justification for this phenomenon.

(c) Create a new discrete-time sinusoidal signals, $y[n]$, at the same frequencies mentioned in part (a), but sampled at a ten-fold frequency, say $f_s = 80$ kHz, and extending over the same interval of time as $x[n]$, i.e. if $x[n]$ had N samples, $y[n]$ should have 10xN samples. Plot the $x[n]$ and $y[n]$ sequences corresponding to the same frequency on the same figure, in a superposed fashion, using different colors. Can you now give a time-domain explanation?

Hints: 1. Use `plot(nx/fsx,x)` for $x[n]$ and `plot(ny/fsy,y)` for $y[n]$, to plot in time units, not sample units. Note that `nx` and `ny` are the sample number sequences, while `fsx` and `fsy` are the sampling frequencies of $x[n]$ and $y[n]$ respectively. 2. Use markers when plotting, to clarify sample locations. This can be done by adding an extra parameter to the `plot`. Multiple figures can be superposed using the command `hold on` command, or from the figure GUI. Try `help plot` for more info.

3 Effect of sampling on audio and speech signals

In this experiment we study the effects of sampling frequency on human perception of audio/speech signals. Let us take music files sampled at 32 kHz. To see the effect of reducing sampling frequency, we will downsample the signals and listen to them.

Exercise: Sampling audio/speech signals

(a) Use MATLAB's `wavread()` function to read wave files. The wave files were sampled at 32 kHz. Define a downsampling rate as, say, `dsr=2`. Create a downsampled version of the audio signal as follows: `dsignal=signal(1:dsr:end)`. Use `sound` command to play `dsignal`, and specify 32000/`dsr` as new sampling frequency, since the signal has been downsampled by `dsr`. Does the music sound different? What is the audible effect of downsampling? What happens for `dsr` values of 2, 4, 8, 16, 32 or more?

(b) Repeat part (a) for the speech signal, and note down your observations. Compare your observations with audio signals.

4 Quantization

So far, we have looked at the process of sampling signals. Sampling reduces the resolution in time (or space for images) from infinity to a finite number. However, the samples still have an infinite resolution in amplitude. For the case of audio signals studied above, each sample is represented as a 16-bit number. This means that there are only 65536 possible levels: in other words, the resolution in the amplitude is finite. Approximating an infinite precision real world sample measurement by a finite precision computer number is what we call quantization. Quantization leads to a loss in the signal quality, because it introduces a quantization error. In this section, we will try to develop some intuition of this phenomenon.

4.1 Simulating Quantization

For the purpose of studying the effects of quantization, we need an easy procedure to manually control the quantization of signals. Assuming that your signal varies in the range $[a, b]$, one practical way of quantizing the signal to 2^N possible levels is the following:

$$x_q = \left\{ \frac{x-a}{b-a} \times (2^N - 1) \right\} \times \frac{b-a}{2^N - 1} + a \tag{3}$$

Here, N represents the number of bits to represent the sample value, and {} denotes the rounding operation. The signal range is divided into 2^N levels, equally spaced, as such this procedure is called uniform quantization. The equation above (1) reduces the signal to the $[0, 1]$ range, (2) scales the result up to the $[0, 2^N 1]$ range, (3) approximates values as integers (there are 2^N integers in that range), and (4) moves everything back to the $[a, b]$ range.

Exercise: Quantization of audio/speech

(a) Create a function, `quantize()`, that takes a vector/matrix, and N as input arguments, and returns the quantized version of each element of that vector/matrix. [Hint: Try help round.]

(b) The supplied audio files were quantized at 16 bits per sample. Using the above function create a 4-bit quantized version of the audio files. Play the new signal. What can you say about its quality.

(c) Now, instead of playing the quantized signal (at 4-bits), play the difference between the original audio file, and your quantized audio file. What does this sound like? Which one is better? Plot the difference signal, and try intuitively justify why the distortion caused by quantization is sometimes referred as "quantization noise"?

(d) Repeat the parts (b) and (c) for 3-bit, 2-bit, and 1-bit quantized signals. Do you have any special comments on 1-bit quantization?

Exercise: Quantization of Digital Images

Repeat the studies on effects of quantization for the gray-scale images supplied to you. Load the images to MATLAB using `imread`. The image pixel values will be in the uint8 format, do: `img=double(img)` to cast them to double, and manipulate them more flexibly. Quantize the image to 64, 32, 16, 8, 4 and 2 levels by specifying appropriate values for N. Don't forget to cast the `img` back to `uint8` before using `imshow` to display the image.