

# COSC1229 - COSC1479

## Computational Science 1

Ron van Schyndel,  
School of Computer Science and IT, RMIT

Semester 2, 2003

## Topic 14

### Elements of 1D Signal Processing

---

S2-2003

---

Elements of 1D Signal Processing

14-1

#### References

N. S. Jayant, Peter Noll, *Digital Coding of Waveforms*, Prentice-Hall, 1984, (*Considered the bible of quantisation, and still quite relevant*)

Alan V. Oppenheim, Ronald W. Schaffer, John R. Buck, *Discrete-Time Signal Processing, second edition*, Prentice-Hall, 2001 (*Considered one of the bible of DSP*)

William Hayes, *Digital Signal Processing*, Schaum's Outline series, McGraw-Hill, 1999

James H McClellan, *et al*, *Computer-based Exercises for Signal Processing using Matlab 5*, Prentice-Hall, (c) 1998

---

S2-2003

---

Elements of 1D Signal Processing

14-2

#### Elements of 1D Signal Processing

This is necessarily a very broad overview of signal processing, but it will concentrate on the aspects that you are likely to encounter in performing computation on sampled signals.

The main concepts in Signal Processing are:

- Discrete / Continuous time and space
- Convolution / Correlation
- Fourier Transforms
- Impulsive / Periodic signals
- Statistical Processing
- Spectral Analysis

You may need to be aware of only some of the above concepts for any one computational task involve sampled data, but it is still good to know what to look up when you need it.

---

S2-2003

Mathematics generally deals with equations of the form

$$x(t) = e^{-t/16} \sin(2\pi t/16), \quad t \geq 0 \quad (1)$$

given by the graph which is a *continuous* signal, because  $x(t)$  is known for every value of  $t \geq 0$ .

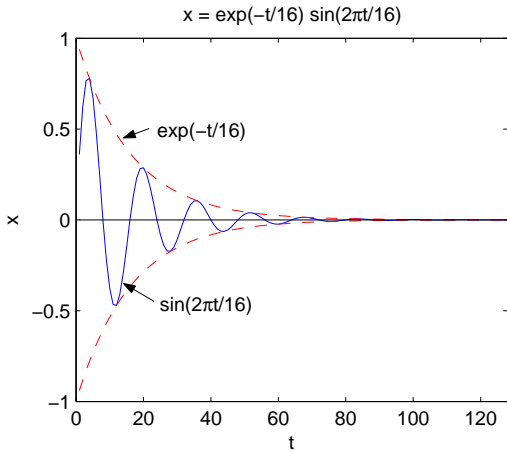


Figure 1: Continuous Signal

In the real world, continuous signals can be generated using electronics and mechanical devices, but they are not generally measured that way (one exception is a balancing device such as the *potentiometer* which balances two input voltages).

For computation, it is usual to sample the data in some way, convert the sampled values into a digital form and store the samples in a file.

In this case, the previous figure, sampled, looks like the figure below.

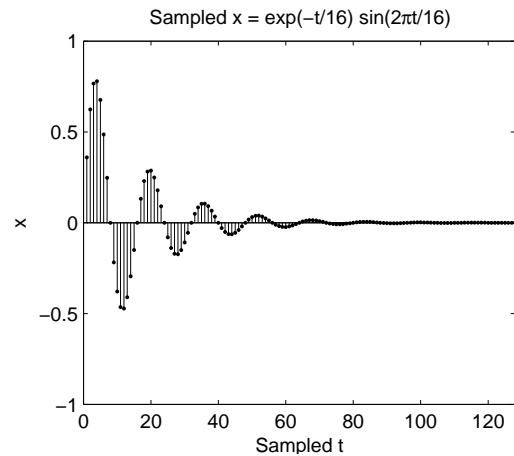


Figure 2: Sampled Signal

Overview

A typical flow-chart of the measurement process shows that the

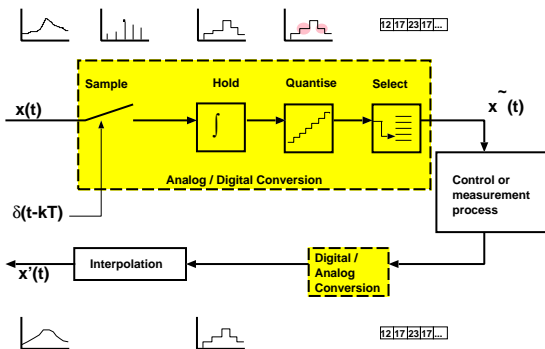


Figure 3: Sampling Overview

signal (typically voltage) is first sampled at a regular rate. Each sample value is then 'held' while it is digitised into an equivalent digital form, where it is encoded with a bit value.

Based on the program that is running, the computer may output the processed signal and have it converted back to analog mode. Notice that the output analog signal is not the same as the input even with no processing. Information has been lost by *digitisation*. This is why knowledge of quantisation effects plays a key role in understanding experimental output.

We will look at some of these steps in detail.

Conventional maths deals with sampling by using an operator called the *dirac delta*, which is defined as:

$$\delta(t) = \begin{cases} 0 & t \neq 0 \\ 1 & t = 0 \end{cases} \quad (2)$$

A signal  $x(t)$  is ideally sampled by a train of impulses

$$s_p(t) = \delta(t - kT)$$

where  $T$  is the *sampling period* and  $k$  is any integer.

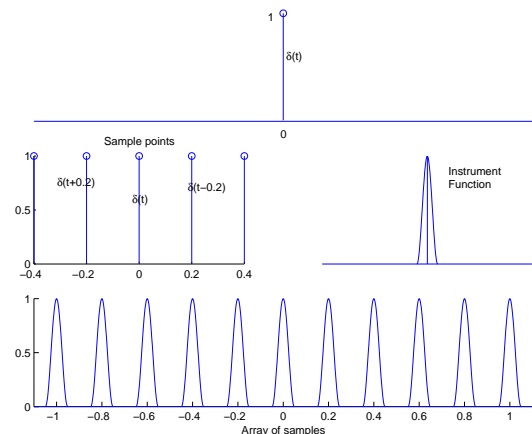


Figure 4:  $X_{Sampled}(t) = x(t) \times \text{impulses}$  where samples are obtained

A sampled signal is then processed as in

$$\tilde{x}(t) = s_p(t)x(t)$$

which in matlab can be shown using a stem plot as in the previous figure.

Equation 1 would thus be written as:

$$\begin{aligned} \tilde{x}(t) &= s_p(t)x(t) \\ &= e^{-t/16} \sin(2\pi t/16) \delta(t - kT) \end{aligned} \quad (3)$$

where  $\tilde{x}$  is the *sampled x*. Note that the sample period  $T$  is not related to the period of the sine wave

Figure 4 shows the case for ideal sampling at the top and middle-left.

A real instrument does not sample instantaneously. Instead a sample measurement will more likely look like middle-right. This is the *instrument function*, or the way that the instrument affects our measurement.

Part of experimental data collection is to find out how the instrument has modified the data.

The effect of the instrument function can be illustrated in the following figure where a 'fat' probe is measuring the surface height, and is moved horizontally by a sampling distance of  $T$ .

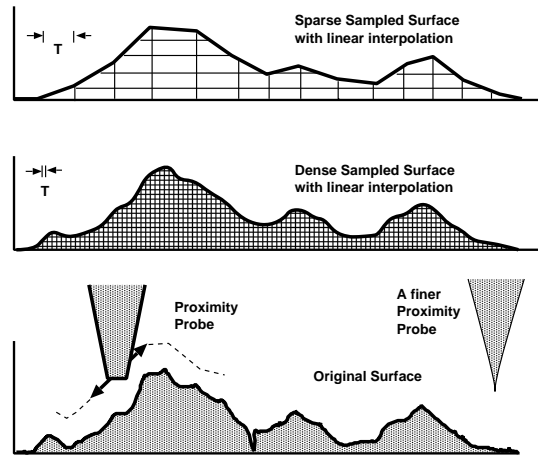


Figure 5: Differences in measured surface based on *Instrument Function*

In the top figure, it is clear that we are sampling too sparsely, filling in the gaps by interpolating, and therefore missing much of the detail. What is not so obvious is that even if we sampled densely as in the middle figure, the sampled curve still doesn't match the original. In this case, this is due to the shape of the probe (we need a finer probe!). For example, the crack in the middle would never be 'seen' by the probe.

## Over and Under Sampling

In order to reconstruct the original signal, it is critical to space the sampling points optimally according to the characteristics of the instrument. Thus in the figures below, the figure at left shows under-sampling as there are times,  $t$  when there is no measurement being made (eg  $t = 0.1$ ). The figure at right shows over-sampling, where something at time  $t$  affects more than one sample point. The figure at bottom shows optimal sampling

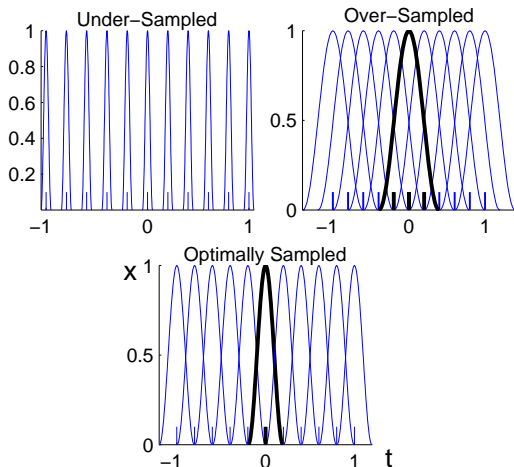


Figure 6: Optimal and Sub-optimal Sampling

## Quantisation by Digitisation

**Digitisation** is the process of converting a continuous variable into one of a finite set of discrete values.

**Quantisation** is the *interpretation* of a continuous quantity using a finite set of discrete values.

In other words, while quantisation attempts to have the digital form represent some aspect of the original continuous data, digitisation may not.

An *analog to digital converter (ADC)* is a device for converting an analog voltage into a digital signal. Most ADC's use a comparison between a generated voltage and the input voltage to determine that input voltage. Described below is the common *successive approximation* ADC algorithm.

```
int sampData = 0; // 16-bit unsigned precision
for (int bit = 32768; bit > 0; bit >>= 1)
    sampData += bit;
    if (isGreaterThanSignal(sampData)) {
        sampData -= bit; // too high, undo operation
    }
}
return sampData;
```

This code resembles a binary search and is quite fast.

isGreaterThanSignal generates a voltage according to outdata, and then compares that with the input voltage. If the generated voltage is higher, it reverses the addition. It then adds half as much in the next loop and tries again.

The accuracy of the ADC is then determined by how accurate a voltage isGreaterThanSignal can generate, and how stable the comparison device is.

An ADC typically has an integer output (usually 8, 10, 12 or 16 bits). High-end ADC's are expensive because of conversion speed, and not usually due to conversion accuracy.

In the above code, it is assumed that sampData=128 generates twice as much voltage as sampData=64. If the assumption holds, the ADC response is said to be linear.

Special-purpose speech and video ADC's are generally not linear, in that they use many bits to describe small voltages, and only few bits to describe large ones. The figure at right shows one such non-linear case (code value is the same as sampData).

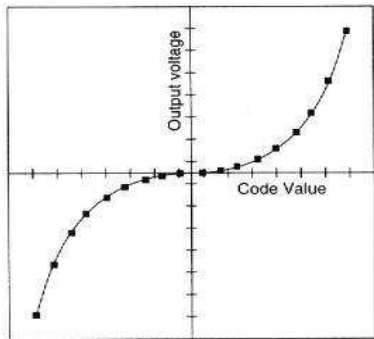


Figure 7:  $\mu$ -law compression

We have so far seen two kinds of quantisation:

1. **Sample Time/Position Quantisation:** The location or time of a sample can be limited (eg every 0.1 sec, or 0.01 cm). This is typically the X axis on a graph.
2. **Sample Value or Amplitude Quantisation:** The possible values that a sample can take is limited by the number of bits of ADC output. This is typically the Y axis on a graph.

The figure below describes the *quantiser characteristic*, a function which maps real input to quantised output. The input values of  $x_k$  are called *decision levels* or *thresholds*, and the output values of  $y_k$  are called *representation levels* or *values*.

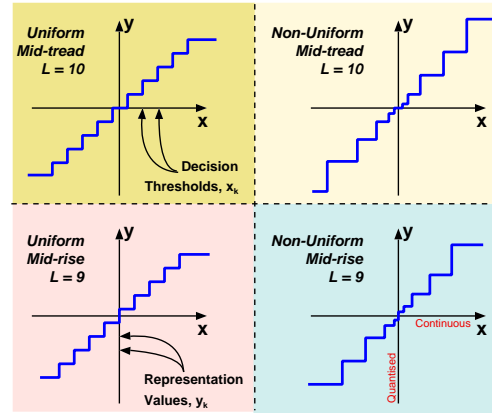


Figure 8: Types of quantisation

### Features of the Quantiser Graph

Some features are:

1. Symmetric about  $x = 0$ .
2. A staircase function, by definition, uniform about  $y = x$ .
3. *mid-rise* or *mid-tread*, depending on whether 0 is one of the representation values.
4. *uniform* or *non-uniform*, depending of whether neighbouring decision or representation levels are uniformly spaced.
5. *Even* number of levels for mid-tread. *Odd* number of levels for mid-rise (typically with 0 in the middle).

We are going to need  $R$  number of bits to represent the output value, where

$$R = \lceil \log_2(L) \rceil$$

so for example, for  $L = 10$ , we will need

$$\log_2(10) = \log(10)/\log(2) = 3.32\text{bits}$$

rounded up to 4 bits (16 possible values) to represent 10 levels.

Non-uniform quantisation is often used in sound recording, when you want soft sounds recorded accurately, but don't care about loud sounds, such as drum beats, since people don't notice errors in drum beats as much as errors in (say) flutes.

The sound in CD's and DVD's are recorded using non-uniform quantisation and over-sampling (as well as being compressed).

We can define a uniform quantisation error as being when

$$q = x - y_k, \quad \text{abs}(q) < 0.5, k = 1, 2, \dots, L - 1$$

which is shown in the graphs below. An important assumption whenever we take measurements and quantise them is that the quantisation error averages out to zero.

Note that, as in *Euler* integration, if the quantisation error does not average to zero, it will accumulate.

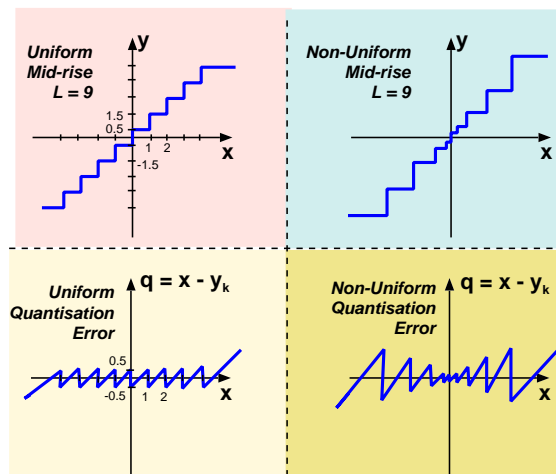


Figure 9: Quantisation Error averages to 0

If you have to quantise your data, there are some general rules you should abide by.

- The decision and reconstruction levels should be symmetric about 0.
- The reconstruction levels should include 0 if the data crosses 0.
- The decision levels should be exactly halfway between reconstruction levels.
- Traditional rounding is a uniform mid-tread operation with  $L = \text{even}$  levels.
- Uniform mid-rise = uniform mid-tread + half a decision level.

If the data doesn't cross 0, the symmetry should be about some central value, such as the mean.

For example, if you quantise  $y = \sin(x)$  to 5 levels above zero and 5 levels below 0 and 1 level for 0, then you would have the following decision ( $d_k$ ) and reconstruction ( $y_k$ ) values:

$d_k = -0.9, -0.7, -0.5, -0.3, -0.1, 0.1, 0.3, 0.5, 0.7, 0.9$   
 $y_k = -1.0, -0.8, -0.6, -0.4, -0.2, 0.0, 0.2, 0.4, 0.6, 0.8, 1.0$

and the Java quantisation code would be

```
public int snapTo(float val, float [] dk, int [] yk)
{
    int high = dk.length, low = -1, test;
    while (high - low > 1)
    {
        test = (high + low) / 2;
        if (dk[test] >= val)
            high = test;
        else
            low = test;
    }
    return yk[high];
}
```

(recognise the algorithm?)

This seems a long way to go when you could have just used a round function, but that only works for uniform integer quantisation. The above code works for all values of val - even if val is out of bounds!

Moreover, this code also works when  $d_k$  and  $y_k$  are not uniformly distributed, such as the  $\mu$ -law case previously.

### The Digital Data

Finally, we have the digitised form of the data, as samples. With the dirac delta that you encountered earlier, and the unit step function, we can manipulate digital data in a form suitable for conventional mathematical treatment. We can look at a sample by multiplying a delayed  $\delta_k(t)$  by the waveform.

We define

$$H(t) = \sum_{k=-\infty}^t \delta(t)$$

$$= \begin{cases} 0 & t < 0 \\ 1 & t \geq 0 \end{cases}$$

as the unit step function.

Also shown in the figure to right is a typical real-world impulse and periodic function.

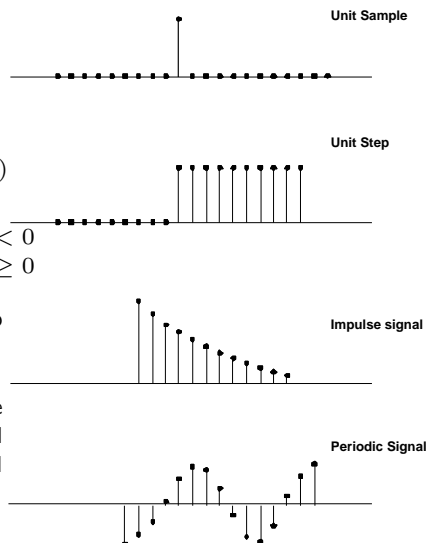


Figure 10: Types of signals

### The Dual nature of the Sampling Theorem

When a periodic signal is sampled, it may not always be clear what the real frequency of the periodicity is. This is because there are two ways to go around a circle. This is best explained with examples.

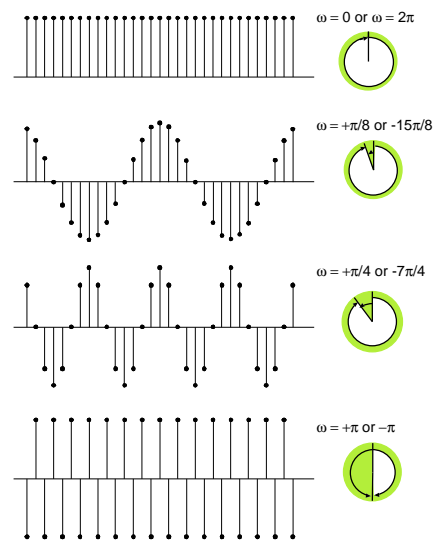


Figure 11: Uncertainty in Frequency

It was determined in the 1940's that if your sampling frequency was less than double the highest frequency present in the data, then you would not be able to reconstruct the data.

In other words, the sampling theorem indicates that a sampled continuous signal can be properly reconstructed, only if it does not contain frequency components above one-half of the sampling rate. This is why CD's are sampled at 44kHz - more than twice the upper human hearing limit (and sampling rate).

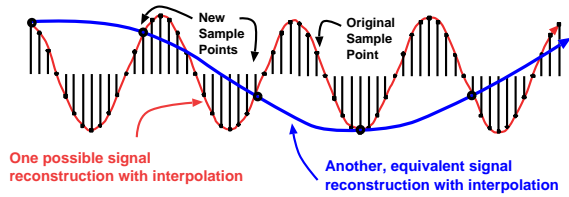


Figure 12: Too slow sampling a high frequency leads to aliasing

The figure above shows the result of sampling a high frequency signal at a rate *much* less than double the data frequency. The resultant red and blue interpolations are both possible reconstructions with nothing to distinguish them.

This is a direct result of the ambiguity in circular motion shown in the figure on the previous slide.

When you are measuring signals and you want to compare them with a standard, a common tool that is used is the *correlation function*.

You may have come across a correlation coefficient before. Well, this has little to do with it, but is often confused with it.

The digital cross-correlation of signal  $y(t)$  of length  $n$  with a standard  $s(t)$  of the same length is defined as:

$$X_{ys}(t) = \sum_{\tau=0}^{n-1} y(\tau)s(t + \tau) \quad (4)$$

To really simplify things a little, we will concentrate only on the *circular correlation*. In this version of the correlation, all array indices wrap around zero (so the value at  $y(0)$  has  $y(1)$  and  $y(n - 1)$  as neighbours where  $n$  is the length of  $y$ ).

If  $y$  were defined as a vector,  $y = 3i + 4j + 5k$ , and  $s$  is similarly defined as  $s = 1i + 2j + 3k$ , then the operation of cross-correlation at  $t = 0$  is identical to the dot-product that you have already met in VCE. i.e.

$$X_{ys}(0) = 1 \times 3 + 2 \times 4 + 3 \times 5 = 3 + 8 + 15 = 26$$

Hence the process of correlation is to line up the arrays, and multiply corresponding elements and then to sum the results.

If you now treat the vectors as arrays  $y = [3, 4, 5]$  and  $s = [1, 2, 3]$ , we can work out  $X_{ys}(1)$ , by shifting  $s$  by one with respect to  $y$ , wrapping the indexes as needed. For example

$$X_{ys}(0) = [1, 2, 3]. * [3, 4, 5] = 1 \times 3 + 2 \times 4 + 3 \times 5 = 26$$

$$X_{ys}(1) = [1, 2, 3]. * [5, 3, 4] = 1 \times 5 + 2 \times 3 + 3 \times 4 = 23$$

$$X_{ys}(2) = [1, 2, 3]. * [4, 5, 3] = 1 \times 4 + 2 \times 5 + 3 \times 3 = 23$$

where  $*$  denotes element-wise multiplication as in Matlab.

Here is some Java code that will correlate  $y$  with  $s$ .

```
public double [] CircCorrel(double [] yval,
    double [] sval) throws ArrayLengthException
{
    int ylen = yval.length;
    int slen = sval.length;
    if (ylen != slen) {
        throw new ArrayLengthException("Unequal Array Lengths");
    }
    for (int t = 0; t < ylen; t++) {
        sum = 0.0;
        for (int tau = 0; tau < slen; tau++) {
            sum += yval[ tau % ylen]
                * sval[(t+tau) % slen];
        }
        correlOutput[t] = sum;
    }
}
```

where `ArrayLengthException` is a custom exception, but should never occur. Note how the modulo `%` performs the desired index wrapping.

### Special Case: Binary Correlation

When  $y$  and  $s$  are strings of binary data, then the correlation becomes much simpler and can easily be implemented in hardware.

We line up the arrays as before, but then we XOR the corresponding elements, count the number of 0 bits and subtract the number of 1 bits in the result, giving us the correlation.

That is,

$$X_{ys}(t) = \sum (\text{number of matching bits}) - \sum (\text{number of mis-matching bits})$$

For example,

$$\begin{aligned} y &= [011010101] \\ s &= [111100111] \\ y \text{ XOR } s &= [100110010] \\ x_{ys}(t) &= \sum (0\text{'s in } (y \text{ XOR } s)) - \sum (1\text{'s in } (y \text{ XOR } s)) \quad (5) \\ &= 5 - 4 \\ &= 1 \end{aligned}$$

In this case,  $y$  and  $s$  do not correlate closely.

Things to note about correlation:

- If  $y$  and  $s$  are exactly the same, the matching part will be  $n$ , the length of  $y$  and  $s$ , so  $X = n$
- If  $y$  and  $s$  are completely different, the mis-matching part will be  $n$ , so  $X = -n$
- If  $y$  and  $s$  are not related, then on average, the matching and mismatching parts will have the same value, and  $X \rightarrow 0$ .

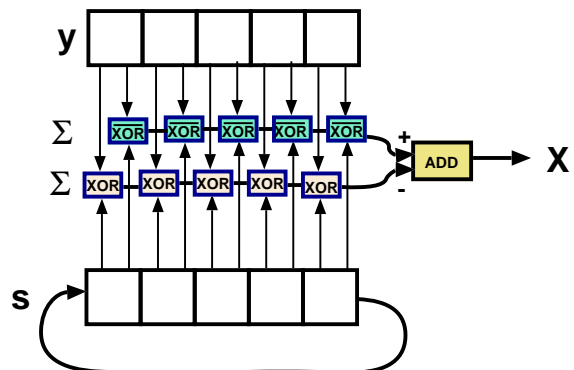


Figure 13: Hardware version of Correlation