UNIT 3:
DIGITAL WAVEFORMS

BASICDSP

3.1 Introduction

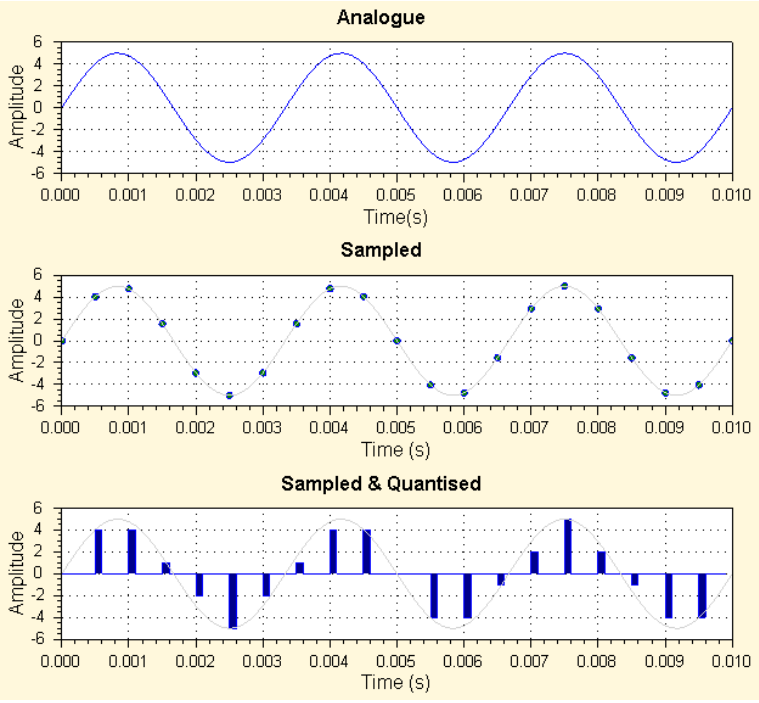
This unit is concerned with the representation and measurement of signals as digital waveforms: specifically with sampling, quantisation, the measurement of energy and the generation of sine and noise signals.

When you have worked through this unit you should:

- understand the concept of sampling
- know the limits to the setting of sampling frequency
- be able to state the sampling theorem
- be able to explain how and why aliasing occurs
- understand the concept of quantisation
- be able to explain the origin of quantisation noise
- understand how sinewaves and noise waveforms can be generated by a program
- have tried to create programs to build and replay waveforms to specification

3.2 Concepts

Sampling is the process whereby an analogue signal (continuous in time and continuous in value) is converted to a series of impulses of a size equal to the amplitude of the signal at regularly spaced instants of time. Related ideas are the *sample period* (T): the time between samples expressed in seconds, and *sampling frequency* (F_s=1/T): the number of samples taken per second of signal, usually expressed in units of samples per second or less accurately and more commonly in hertz (Hz).



Quantisation is the process of converting the sampled analogue signal (discrete-time continuous value) into impulses of discrete values of amplitude (analogous to the conversion from 'floating-point' values to 'integer' values in a program). Related ideas are *quantisation error*: the error in amplitude estimate introduced by quantisation, and *quantisation noise*: the noise introduced into a quantised signal by the approximation of the quantisation to the real analogue values. The most common forms of quantisation are *linear* and produce a binary code conveniently described in terms of numbers of bits. Each bit approximates to 6dB of additional signal/noise improvement. However, non-linear quantisation schemes are also used, primarily in telecommunication systems to make the best use of available capacity. In a logarithmic quantisation scheme, the amplitude levels are further apart at large amplitudes and closer at smaller amplitudes, thereby giving a quantisation error which is proportional to the size of the signal at that instant.

The *sampling theorem* (of Nyquist and Shannon) states that a signal which has a highest frequency component of f Hz must be sampled at a rate of at least 2f Hz if the signal is to be faithfully reconstructed from the digital samples. The consequence of not following the sampling theorem is *aliasing*, whereby the spectral components above f Hz are 'mapped' down to the frequency region 0..f Hz by the sampling process thereby distorting the representation. To prevent aliasing a *pre-sampling* or *anti-aliasing* filter is used to remove spectral components above half the sampling frequency; this is usually implemented as a low-pass filter of high order, with a corner frequency a little less than half the sampling frequency. A similar low-pass *reconstruction* filter is also used in digital-to-analogue conversion, to rebuild the analogue waveform from the digital samples in the frequency region 0..f Hz only.

To demonstrate formally that a sampled signal has many possible aliases as continuous waveforms, consider the sampled sinusoid:

$$x[n] = \sin(n\Omega)$$

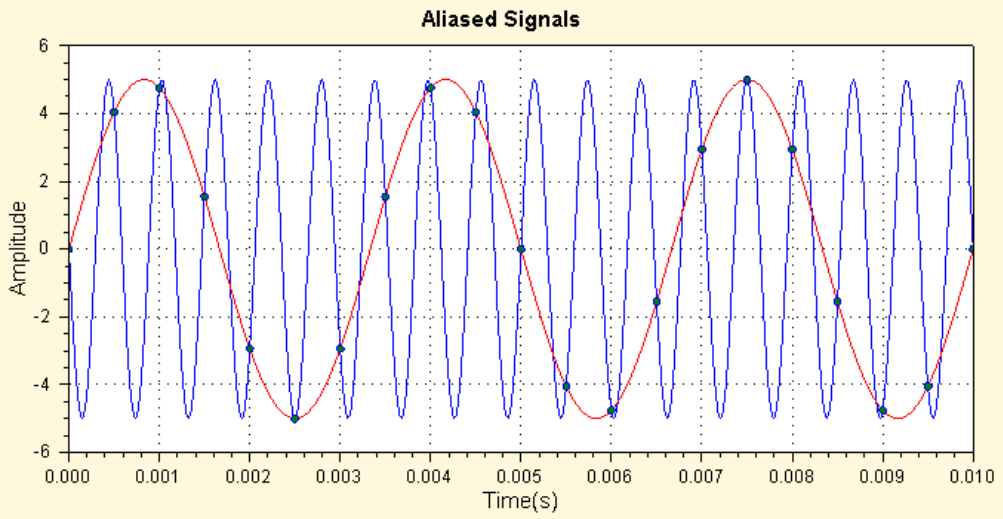
where Ω controls the frequency of the sinusoid (there are $2\pi/\Omega$ samples per period). Consider the sequence $y_1[n]$ obtained by increasing Ω by 2π:

$$\begin{aligned} y_1[n] &= \sin(n(\Omega + 2\pi)) \\ y_1[n] &= \cos(n\Omega)\sin(2\pi n) + \sin(n\Omega)\cos(2\pi n) \\ y_1[n] &= \sin(n\Omega) \end{aligned}$$

In other words, the same sequence as x[n]. Similarly the sequence $y_2[n]$ given by:

$$\begin{aligned} y_2[n] &= -\sin(2\pi n - n\Omega) \\ y_2[n] &= -[\cos(2\pi n)\sin(-n\Omega) + \sin(2\pi n)\cos(-n\Omega)] \\ y_2[n] &= -\sin(-n\Omega) \\ y_2[n] &= \sin(n\Omega) \end{aligned}$$

is also identical with x[n].



The value Ω is just the angular change per sample, expressed in radians. To convert this to a basis in time, we set Ω = ωT, where T is the sample interval, and ω is the *angular frequency* expressed in radians per second. ω is related to conventional frequency f (in hertz) by ω = 2πf. When f = F_s, then Ω = 2π. From this we can see that the aliases of a sinusoid at a frequency f are just mF_s+f and mF_s–f, (*m* is some integer), since:

$$x[n] = \sin(n2\pi fT)$$
$$y_1[n] = \sin(n2\pi(F_s + f)T)$$
$$y_2[n] = -\sin(n2\pi(F_s - f)T)$$

The *total energy* in a digital waveform is simply the sum of the squared amplitude values, the *average energy* is the total energy divided by the number of samples, and the *r.m.s. amplitude* is the square root of the average energy. For a sine wave, the r.m.s. amplitude is just the peak amplitude/√2.

Algorithms

Sinewaves can be constructed from the `Sin()` function provided in the Math library. Noise signals can be generated through the use of the Visual Basic built-in function `Rnd()`.

Algorithm 3.1 Sample.Sine - Sampling a sinusoid

```
Public Shared Function Sine(ByVal freq As Double, _
            ByVal amp As Double, ByVal phase As Double, _
            ByVal time As Double)
    ' angular frequency (radians/sec)
    Dim rfreq As Double = 2.0 * Math.PI * freq
    ' phase in radians
    Dim rphase As Double = 2.0 * Math.PI * phase / 360.0
    ' sample sine function
    Return amp * Math.Sin(rfreq * time - rphase)
End Function
```

Algorithm 3.2 Sample.Noise - Sampling a noise signal

```
Public Shared Function Noise(ByVal amp As Double)
    ' get a random co-ordinate inside the unit circle
    Dim x As Double, y As Double, r As Double
    Do
        x = (2 * Rnd()) - 1.0
        y = (2 * Rnd()) - 1.0
        r = (x * x) + (y * y)
    Loop While (r = 0) Or (r > 1)
    ' transform into a normal distribution (Box-Muller transform)
    Dim rval As Double = x * Math.Sqrt(-2.0 * Math.Log(r) / r)
    ' return scaled sample
    Return amp * rval
End Function
```

Algorithm 3.3 Sample.Quantise - Quantisation of a sample

```
Public Shared Function Quantise(ByVal amp As Double, _
                                ByVal quanta As Double) As Short
    Dim ival As Integer = CInt(amp / quanta)
    If (ival > 32767) Then
        Return 32767
    ElseIf (ival < -32768) Then
        Return -32768
    Else
        Return ival
    End If
End Function
```

Algorithm 3.4 – Waveform.Quantise – Quantisation of a Waveform

```
    Public Function Quantise(ByVal quanta As Double) As Signal
        Dim owv As New Signal(Count, Rate)
        For i As Integer = 1 To Count
            owv(i) = Sample.Quantise(Item(i), quanta)
        Next
        Return owv
    End Function

    Public Function Quantise() As Signal
        Dim owv As New Signal(Count, Rate)
        Dim max As Double = 0
        For i As Integer = 0 To Count - 1
            If (Math.Abs(Item(i)) > max) Then max = Math.Abs(Item(i))
        Next
        For i As Integer = 1 To Count
            owv(i) = Sample.Quantise(Item(i), max / 24000)
        Next
        Return owv
    End Function
```

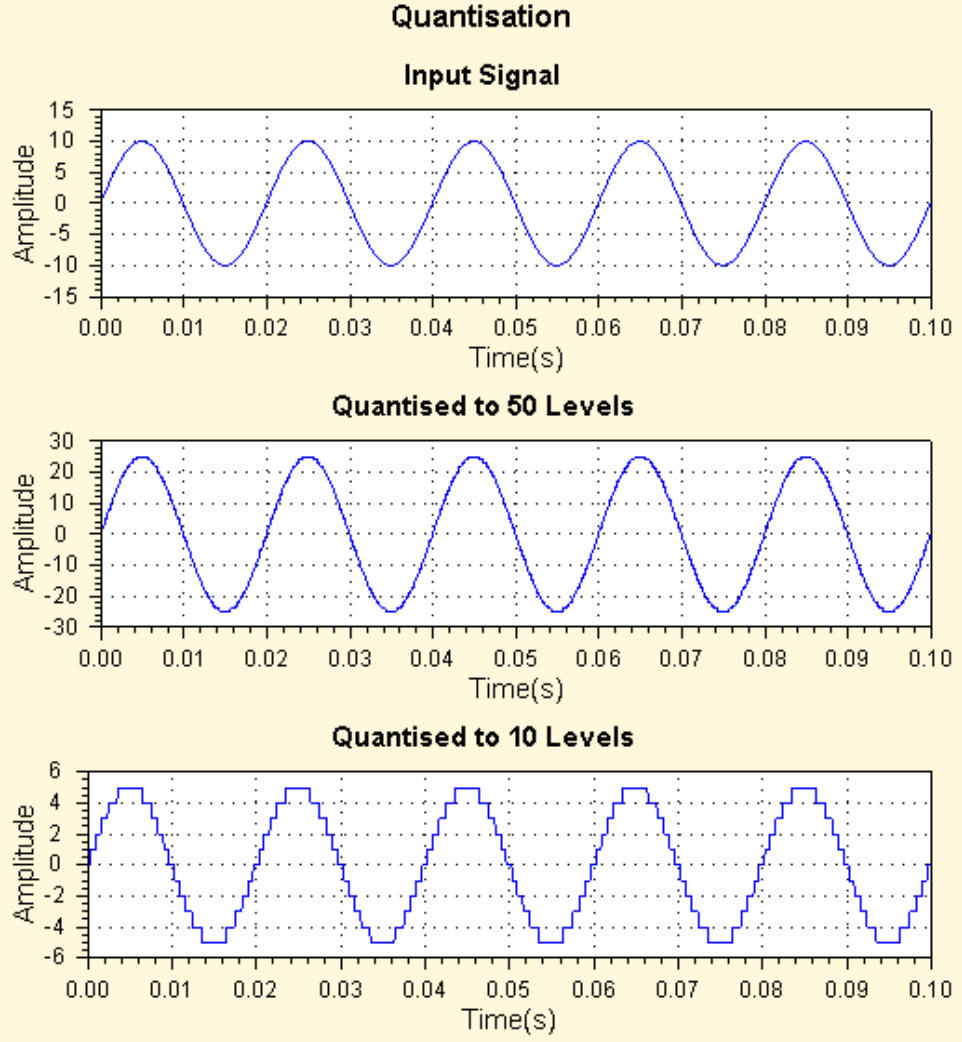
Bibliography

- Rosen & Howell, Signals and Systems for Speech and Hearing, Chapter 14.
- Meade & Dillon, Signals and Systems, Chapter 1.
- Lynn & Fuerst, Introductory Digital Signal Processing, Sections 1.1-1.4.
- Orfanidis, Introduction to Signal Processing, 1.1-1.4.

Example Programs

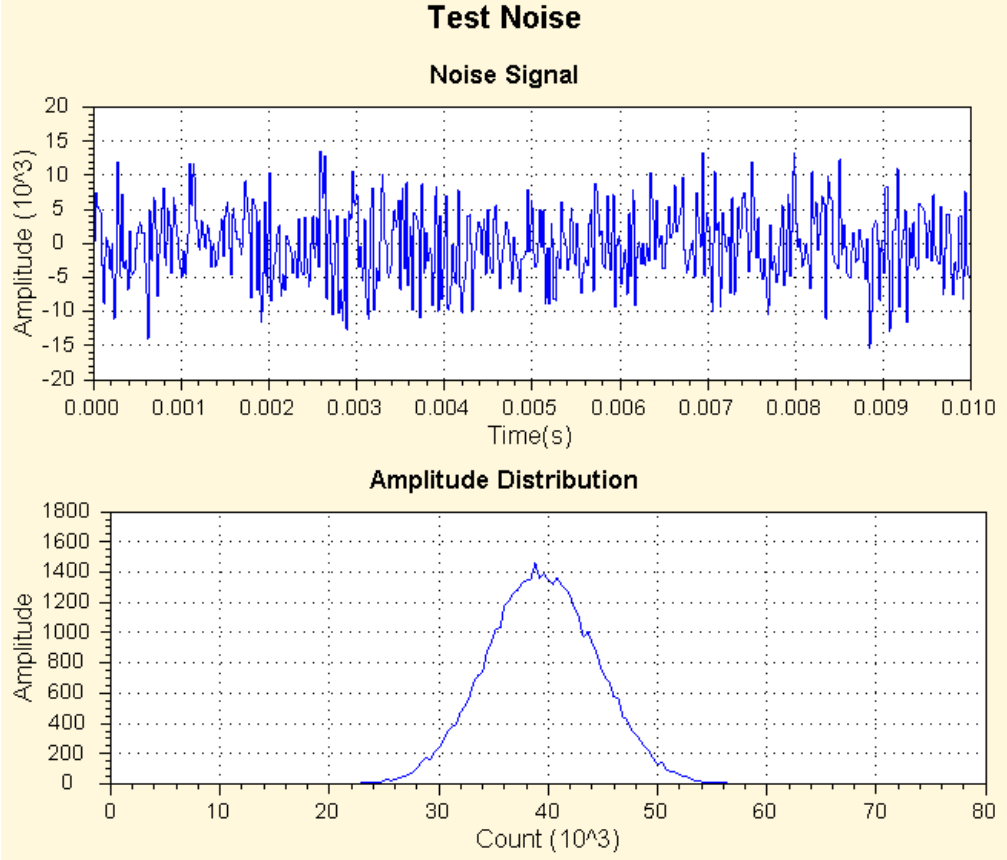
Example 3.1 TestQuantise - Demonstrate Sampling and Quantisation

```
Imports BasicDSP
Imports ZedGraph
Public Class TestQuantise
    Const NUMSAMPLE As Integer = 1000      ' number of samples
    Const SAMPRATE As Double = 10000.0      ' sampling rate
    Const SINEFREQ As Double = 50.0        ' sine at 500Hz
    Const SINEAMP As Double = 10.0       ' sine amplitude
    Dim rwv As Waveform
    Dim qwv As Signal

    Private Sub TestSine_Load(ByVal sender As System.Object, _
                  ByVal e As System.EventArgs) Handles MyBase.Load
        ' set up graphs
        Dim gp As New Graph(Me.CreateGraphics, zgc, 3, 1, _
                                                "Quantisation")
        ' calculate and display sine
        rwv = New Waveform(NUMSAMPLE, SAMPRATE)
        For i As Integer = 1 To NUMSAMPLE
            rwv(i) = Sample.Sine(SINEFREQ, SINEAMP, 0, i / SAMPRATE)
        Next
        gp.PlotWaveform(1, rwv, "Input Signal")

        ' quantise to 50 levels
        qwv = rwv.Quantise(0.4)
        gp.PlotSignal(2, qwv, "Quantised to 50 Levels")

        ' quantise to 10 levels
        qwv = rwv.Quantise(2.0)
        gp.PlotSignal(3, qwv, "Quantised to 10 Levels")

    End Sub

End Class
```


Quantisation
Input Signal
Amplitude
Time(s)
Quantised to 50 Levels
Amplitude
Time(s)
Quantised to 10 Levels
Amplitude
Time(s)

Example 3.2 TestNoise - Demonstrate noise signal generation and replay

```
Imports BasicDSP
Imports ZedGraph
Public Class TestNoise
    Const NUMSAMPLE As Integer = 44100      ' number of samples
    Const SAMPRATE As Double = 44100.0      ' sampling rate
    Const NOISEAMP As Double = 5000.0      ' amplitude
    Dim wv As Signal
    Private Sub TestNoise_Load(ByVal sender As System.Object, _
                  ByVal e As System.EventArgs) Handles MyBase.Load

        ' create graphs
        Dim gp As New Graph(Me.CreateGraphics, zgc, 2, 1, "Test Noise")

        ' build noise waveform
        Randomize()
        wv = New Signal(NUMSAMPLE, SAMPRATE)
        For i As Integer = 1 To NUMSAMPLE
            wv(i) = Sample.Noise(NOISEAMP)
        Next
        gp.PlotSignal(1, wv.Cut(1, SAMPRATE / 100), "Noise Signal")

        ' calculate and plot amplitude histogram
        Dim hist As New WaveDouble(201, 0.0025)
        Dim idx As Integer
        For i As Integer = 1 To NUMSAMPLE
            ' map -40000..40000 to 0..200
            idx = (wv(i) + 40000) \ 400
            ' keep count
            hist(idx) = hist(idx) + 1
        Next
        gp.PlotWaveDouble(2, hist, "Amplitude Distribution", _
                                                "Amplitude", "Count")

    End Sub
    Private Sub zgc_MouseClick(ByVal sender As Object, _
                  ByVal e As System.Windows.Forms.MouseEventArgs) _
                  Handles zgc.MouseClick
        If e.Button = Windows.Forms.MouseButtons.Left Then
            wv.Replay()
        End If
    End Sub
End Class
```



Exercises

3.1 Use example program 3.1 to construct a program to display 1s of a sinewave signal of 25Hz sampled at 1,000 samples per second.

3.2 Adapt your program from exercise 3.1 to display a sine signal that increases linearly in frequency from 10Hz to 50Hz over an interval of 1 second.

3.3 Adapt your program from exercise 3.1 to construct and display 1s of a 10Hz square wave made up from the sum of 16 odd harmonics. The relative amplitude of the *n*th harmonic is given by 1/*n*, i.e. $H_1 + 0.33H_3 + 0.2H_5 + \ldots$, or

$$y[m] = \sum_{n=0}^{15} \frac{\sin((2n+1)20\pi mT)}{2n+1}$$