UNIT 4: DIGITAL SYSTEM MODELS

BASICDSP

4.1 Introduction

This unit is concerned with the description of digital systems, it introduces the concepts of a linear time-invariant system, convolution, the general system difference equation and its implementation, and the frequency response characteristics of systems.

When you have worked through this unit you should:

- be able to define the characteristics of a linear, time-invariant system
- understand what is represented by an impulse response and a frequency response of a system
- be able to explain how and why systems may be simulated with discrete convolution
- understand how a system can be described with the general difference equation
- be able to describe one direct form method for the implementation of the general difference equation in an algorithm
- be able to state the z-transform of a sampled sequence
- appreciate that the z-transform may be used to obtain a polynomial statement of the general difference equation.
- understand how the response of a system may be stated in terms of poles and zeros
- understand that the response can be evaluated for any particular frequency to get the frequency response graph of a system
- know how to use an algorithm for implementing an LTI system and calculating its response
- be able to relate the parameters of a simple resonator to its implementation

4.2 Concepts

We are concerned with systems that transform one digital signal into another, in particular with systems that perform a *linear* processing that does not change with time (is *time-invariant*). We seek to describe the behaviour of such systems and to find a general algorithmic description suitable for digital implementation.

Generally we describe systems that modify signals in terms of their *Frequency Response* (a graph of system response against frequency) and their *Phase Response* (a graph of phase shift against frequency). A linear time-invariant (LTI) system may also be described in the time-domain by means of its *Impulse response*, that is: the graph of the output of the system after an impulse is presented on its input. This is possible because we can describe any digital signal as a sequence of impulses of a height equal to the signal amplitude occurring at the sampling instants. The output of a LTI system to any signal can therefore be predicted by a suitable linear combination of impulse responses starting at each sample instant (this is a consequence of the principle of *superposition*).

Mathematically, this process can be described as taking a signal x[] and an impulse response h[] and calculating each output sample y[n] with the formula:

$$y[n] = \sum_{k=0}^{\infty} x[n-k]\,h[k+1]$$

Assuming h[] is zero at negative times, and samples are counted from 1. The operation is analogous to laying the impulse response backwards alongside the input signal prior to *n* and then cross multiplying and adding. This process is known as *convolution*. y[n] is said to be generated by the convolution of x[] and h[].

Convolution with the impulse response is the time-domain equivalent of multiplying by the frequency response. You can see this by considering convolution of h[] with a sinusoid signal given by:

$$x[n] = \exp(i2\pi f n T_s)$$

so that the output of the system is given by:

$$y[n] = \sum_{k=0}^{\infty} \exp(i2\pi f (n-k) T_S)\, h[k+1]$$

which can be rewritten as:

$$y[n] = \exp(i2\pi f n T_S) \sum_{k=0}^{\infty} \exp(-i2\pi f k T_S)\, h[k+1] = x[n] H(f)$$

That is the output is also a sinusoid, at the same frequency as the input, but with an amplitude scaling and a phase change given by H(f).

The impulse response of a simple resonator is simply an exponentially decaying sinewave of a frequency equal to the natural frequency and of a rate of decay related to the bandwidth.

Systems that have some kind of 'memory' can have impulse responses of infinite duration (called Infinite-Impulse-Response or *IIR* systems), others have an impulse response of finite duration (called Finite-Impulse-Response or *FIR* systems). Our simple resonator theoretically continues vibrating for ever, so is an IIR system.

To describe an LTI system algorithmically, we need a formulation of its behaviour that allows both finite and infinite impulse responses. We do this by noting that the characteristics of an LTI system must be expressed in a constant linear relationship between previous output samples and previous input samples. That is, given input samples x[] and output samples y[], some linear combination of output samples is related to some linear combination of input samples:

$$y[n] + b_1\, y[n-1] + b_2\, y[n-2] + \ldots + b_q\, y[n-q] =$$
$$a_0\, x[n] + a_1\, x[n-1] + a_2\, x[n-2] + \ldots + a_p\, x[n-p].$$

This is the general difference equation of an LTI system. It defines the behaviour of an LTI system in terms of a set of coefficients a[0..p] that operate on current and previous input samples, and a set of coefficients b[1..q] that operate on previous output samples. b[0] is always taken as 1. This is perhaps easier to see with this reformulation:

$$y[n] = \sum_{i=0}^{p} a_i x[n-i] - \sum_{j=1}^{q} b_j y[n-j]$$

This can be readily implemented as a computer algorithm.

The general difference equation leads to a simple classification of LTI systems:

(i) systems with the coefficients b[1..q] are zero, in which case output samples are generated by a linear combination of previous input samples. These are called *non-recursive* or *moving average* systems.

(ii) systems in which a[0] is 1, and a[1..p] are zero, in which case the output is formed from a single input and a linear combination of past output samples. These are called *simply recursive* or *autoregressive* systems.

(iii) systems with arbitrary a[] and b[] coefficients are sometimes called *autoregressive/moving average* or *ARMA* systems. They are of course also recursive.

The *direct form* implementation of the general difference equation requires two memory arrays: one for past inputs x[] and one for past outputs y[]. It can be shown that the same result can be achieved with only a single memory of an 'internal' waveform s[]. This *direct form II* is the basis for the `LTISystem()` implementation below.

From the description of an LTI system in terms of a[] and b[] coefficients we can also determine its frequency response characteristics (note that we leave to Unit 7 the reverse problem of the determination of a[] and b[] from the frequency response). This is normally performed using a mathematical transformation of the a[] and b[] coefficients akin to the discrete Fourier transform, called the *Z transform*.

Put simply, the Z transform converts a sequence of digital samples at successive sampling instants to a polynomial of some operator z^{-1}, in which the amplitudes of the samples become the coefficients of the polynomial. Thus a finite signal x[1..n] has a Z transform:

$$X(z) = x[1]z^{-1} + x[2]z^{-2} + x[3]z^{-3} + \ldots + x[n]z^{-n}$$

that is

$$X(z) = \sum_{n} x[n]z^{-n}$$

One way of thinking of z^{-1} is that it represents a unit delay. In other words multiplying by z^{-n} delays a z-transformed signal by *n* samples. This analogy allows to view a z-transformed signal as scaled impulses delayed appropriately and added together.

The conversion of a sequence to a polynomial is useful because it allows us to write the general difference equation as two polynomials:

$$Y(z)\,(1 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_q z^{-q}) =$$
$$X(z)\,(a_0 z^{-0} + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_p z^{-p}).$$

Where X(z) is the z-transform of the input x[], and Y(z) the z-transform of the output y[]. The cross multiplication and collection of terms in z^{-n} is equivalent to convolution on the original signals. This polynomial form of the difference equation allows us to form the response of the system:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{i=0}^{p} a_i z^{-i}}{1 + \sum_{j=1}^{q} b_j z^{-j}}$$

that is, as a ratio of two polynomials in z. We can then see that this response will have peaks at the roots[1] of the denominator polynomial - these are called the *poles* of the system, and dips at the roots of the numerator polynomial - these are called the *zeros* of the system. Our LTI system can be described completely (with the addition of an overall gain factor) by the location of the poles and zeros of the system. These will in general be complex numbers and can be plotted on the complex plane on a diagram called a *z-plane diagram*.

To obtain the frequency response of our system we need to put into the system sinewaves at different frequencies but of unit amplitude and plot the output amplitude. We do this by substituting z^{-1} in our polynomial for the complex sine $e^{i\Omega}$ where Ω is the angular frequency ($= 2\pi f/F_s$) at which we require the response. We can now obtain a numerical value for the response for a range of frequencies between 0 and the sampling rate (2π). What we are actually calculating is the product of the distances between each system zero and a point on the *unit circle* specified by $e^{i\Omega}$ divided by the product of the distances between each pole and that point. Thus the nearer the poles are to the unit circle the higher the peaks in the response and the closer the zeros are to the unit circle the lower the dips. The magnitude of this ratio at a given frequency is the magnitude of the frequency response, while the argument of this ratio is the phase response.

[1]The *roots* of a polynomial in x are those values of x for which the polynomial is zero. If the polynomial is *factored* into a form (x-a)(x-b)(x-c)..(x-p), then the roots are the values a,b,c,..,p.

Algorithms

Algorithm 4.1 Convolution

```
''' Convolves two waveforms
Public Shared Function Convolve(ByRef x As Waveform, ByRef h As Waveform) As
Waveform
    Dim y As New Waveform(x.Count + h.Count - 1, x.Rate)
    Dim v As Double

    ' for each output sample
    For n As Integer = 1 To y.Count

        ' sum of product with reverse IR
        v = 0
        For k As Integer = 0 To h.Count() - 1
            v += x(n - k) * h(k + 1)
            ' exploits bad index capability of Waveform class
        Next
        y(n) = v
    Next
    Return y
End Function
```

Algorithm 4.2 Linear time-invariant system model

```
Public Class LTISystem
    ' Numerator coefficients
    Public a() As Double
    ' Denominator coefficients
    Public b() As Double
    ' Internal state memory
    Public s() As Double

    ' Create new linear system
    Public Sub New(ByVal na As Integer, ByVal nb As Integer)
        ReDim a(na)
        ReDim b(nb)
        If (na > nb) Then ReDim s(na) Else ReDim s(nb)
        Clear()
    End Sub

    ' Create a copy of a linear system
    Public Sub New(ByVal ltis As LTISystem)
        a = ltis.a
        b = ltis.b
        s = ltis.s
    End Sub

    ' Clear the state memory
    Public Sub Clear()
        System.Array.Clear(s, 0, s.Length)
    End Sub
```

```
    ' Pass a single sample value through the linear system
    Default Public ReadOnly Property Item(ByVal ival As Double) As Double
        Get
            Dim i As Integer
            ' shift state memory
            For i = s.Length - 1 To 1 Step -1
                s(i) = s(i - 1)
            Next

            ' compute s[0] from y[] coefficients
            s(0) = ival
            For i = 1 To b.Length - 1
                s(0) -= b(i) * s(i)
            Next

            ' compute output from x[] coefficients
            Dim out As Double
            For i = 0 To a.Length - 1
                out += a(i) * s(i)
            Next

            Return out
        End Get
    End Property

    ' Pass a waveform through the filter
    Public Function Filter(ByRef iwv As Waveform) As Waveform
        Dim owv As New Waveform(iwv.Count, iwv.Rate)
        Clear()
        For i As Integer = iwv.First To iwv.Last
            owv(i) = Item(iwv(i))
        Next
        Return owv
    End Function

    ' Pass a signal through the filter
    Public Function Filter(ByRef iwv As Signal) As Signal
        Dim owv As New Signal(iwv.Count, iwv.Rate)
        Clear()
        For i As Integer = iwv.First To iwv.Last
            owv(i) = Item(iwv(i))
        Next
        Return owv
    End Function

    ' Calculate frequency response of linear system at a single frequency
    Public Function Response(ByVal freq As Double) As Complex
        Dim i As Integer
        Dim omega(s.Length) As Complex
        Dim z As Complex = Complex.Exp(New Complex(0, 2 * Math.PI * freq))

        ' initialise polynomial of complex frequency
        omega(0) = New Complex(1.0)
        omega(1) = z
        For i = 2 To s.Length - 1
            omega(i) = omega(i - 1) * z
        Next

        ' compute response of numerator
        Dim rnum As New Complex(0)
        For i = 0 To a.Length - 1
            rnum = rnum + a(i) * omega(i)
        Next

        ' compute response of denominator
        Dim rden As New Complex(1.0)
        For i = 1 To b.Length - 1
            rden = rden + b(i) * omega(i)
        Next
```

```
        ' compute ratio
        If (rden.Mag = 0) Then
            Return New Complex(100000.0)        ' i.e. infinity
        Else
            Return rnum / rden
        End If

    End Function
End Class
```

Algorithm 4.3 Digital resonator

```
' Builds a linear system for a resonator
Public Shared Function Resonator(ByVal freq As Double, ByVal bwidth As Double)
As LTISystem
    Dim ltis As New LTISystem(1, 3)

    ' get parameters as angles
    Dim wf As Double = 2 * Math.PI * freq
    Dim wb As Double = 2 * Math.PI * bwidth

    ' estimate pole radius from bandwidth (rule of thumb)
    Dim r As Double = 1.0 - wb / 2

    ' set up numerator
    ltis.a(0) = 1.0

    ' set up denominator: quadratic formula
    ltis.b(1) = -2.0 * r * Math.Cos(wf)
    ltis.b(2) = r * r

    ' adjust numerator for unity gain at DC
    ltis.a(0) /= ltis.Response(0).Mag

    Return ltis
End Function
```

Bibliography

Meade & Dillon, Signals and Systems, Chapter 5.

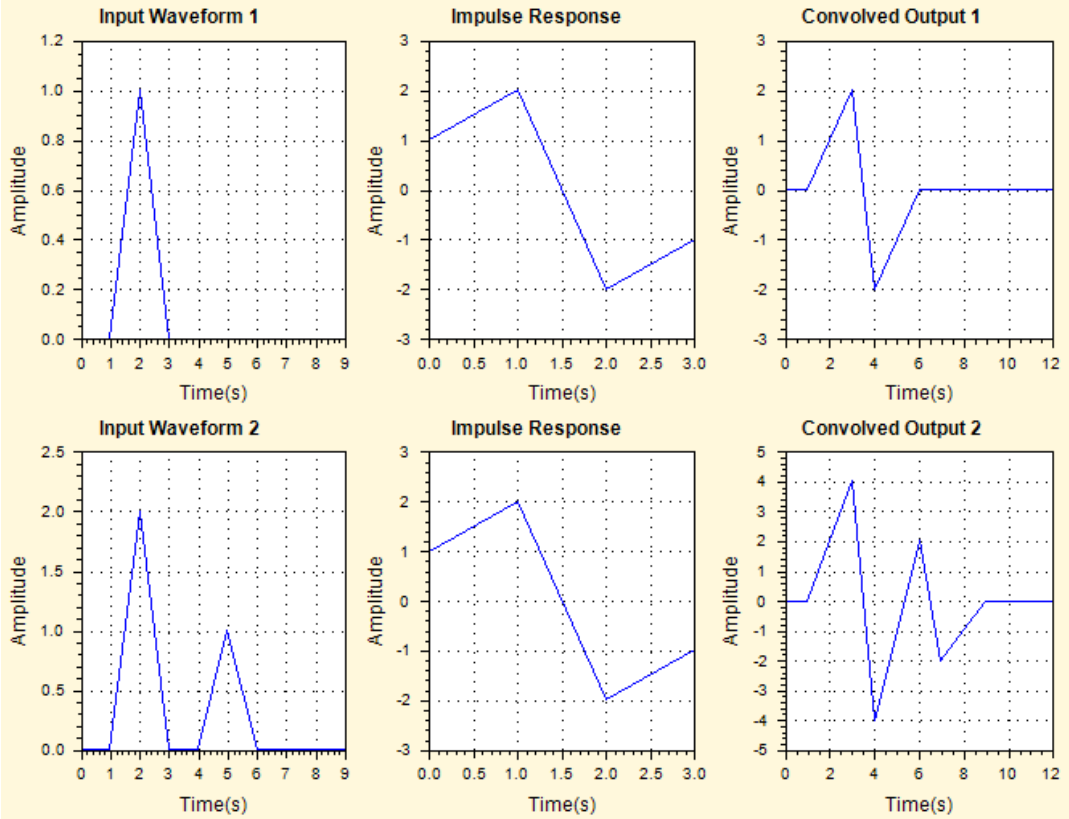
Lynn & Fuerst, Introductory Digital Signal Processing, Chapters 2 & 4.

Orfanidis, Introduction to Signal Processing, Sections 5.1, 5.4, 6.1-6.3.

Example Programs

Example 4.1 Demonstration of convolution

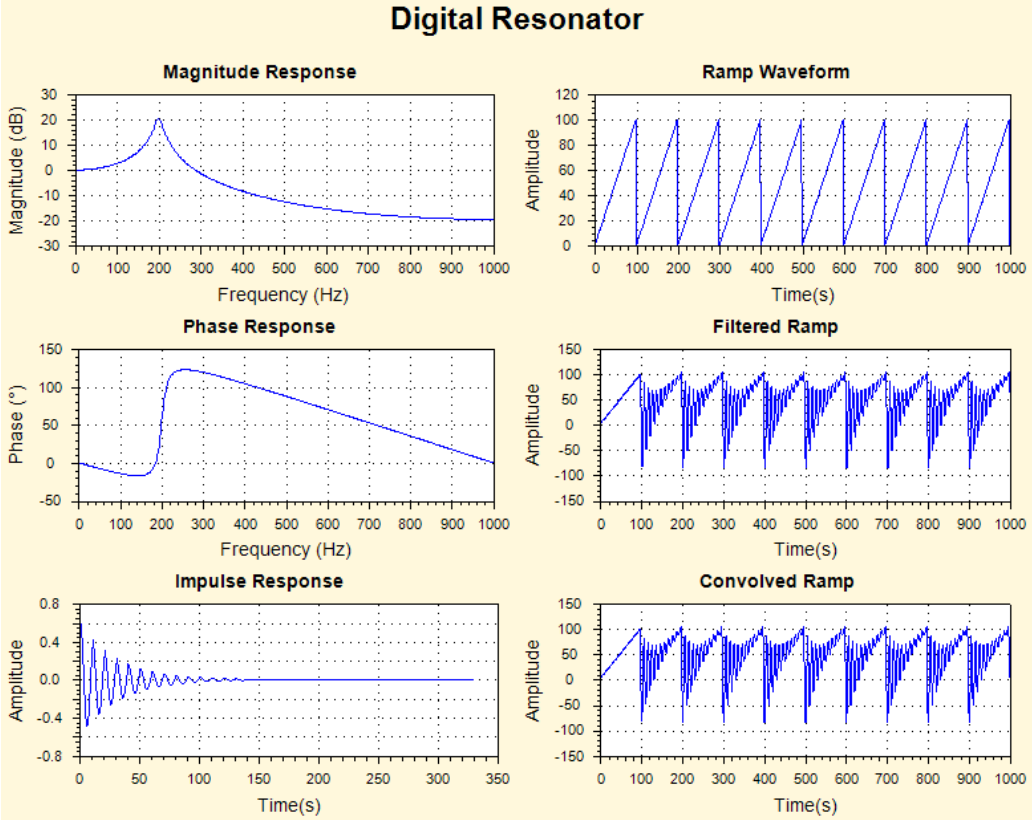
```
Imports BasicDSP
Imports ZedGraph
Public Class TestConvolve

    Private Sub TestConvolve_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, 2, 3)

        ' create input signal and display
        Dim x As New Waveform(10, 1)
        For i As Integer = 1 To 10
            x(i) = 0
        Next
        x(3) = 1
        gp.PlotWaveform(1, x, "Input Waveform 1")

        ' create impulse response
        Dim h As New Waveform(4, 1)
        h(1) = 1
        h(2) = 2
        h(3) = -2
        h(4) = -1
        gp.PlotWaveform(2, h, "Impulse Response")

        ' do convolution
        Dim y As Waveform = Filter.Convolve(x, h)
        gp.PlotWaveform(3, y, "Convolved Output 1")

        ' try a different input
        For i As Integer = 1 To 10
            x(i) = 0
        Next
        x(3) = 2
        x(6) = 1
        gp.PlotWaveform(4, x, "Input Waveform 2")

        ' same impulse response
        gp.PlotWaveform(5, h, "Impulse Response")

        ' do convolution
        y = Filter.Convolve(x, h)
        gp.PlotWaveform(6, y, "Convolved Output 2")

    End Sub
End Class
```



Example 4.2 Equivalence of filtering and convolution

```
Imports BasicDSP
Imports BasicDSP.Filter
Imports ZedGraph
Public Class TestResonator
    Const ILIMIT As Double = 0.00001    ' length limit for impulse response

    Private Sub TestResonator_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, 2, "Digital Resonator")

        ' create resonator
        Dim ltis As LTISystem = Resonator(0.1, 0.01)

        ' calculate frequency response
        Dim fresp As New Spectrum(500, 1000)
        For i As Integer = fresp.First To fresp.Last
            fresp(i) = ltis.Response(i / 1000.0)
        Next

        gp.PlotDbSpectrum(1, fresp, "Magnitude Response")
        gp.PlotPhaseSpectrum(3, fresp, "Phase Response")

        ' calculate impulse response
        Dim iresp As New Waveform(0, 1)
        Dim lval As Double              ' last output
        Dim oval As Double = ltis(1.0)  ' put in unit pulse

        While (Math.Abs(oval) > ILIMIT) Or (Math.Abs(oval - lval) > ILIMIT)
            iresp.Add(oval)         ' append sample
            lval = oval             ' remember sample
            oval = ltis(0.0)        ' get next sample
        End While
```

```
        gp.PlotWaveform(5, iresp, "Impulse Response")

        ' create a ramp waveform
        Dim ramp As New Waveform(1000, 1)
        For i As Integer = 1 To 1000
            ramp(i) = i Mod 100
        Next
        gp.PlotWaveform(2, ramp, "Ramp Waveform")

        ' filter the ramp waveform
        Dim framp As New Waveform(1000, 1)
        ltis.Clear()
        For i As Integer = 1 To 1000
            framp(i) = ltis(ramp(i))
        Next
        gp.PlotWaveform(4, framp, "Filtered Ramp")

        ' convolve ramp with impulse response
        Dim cramp As Waveform = Filter.Convolve(ramp, iresp)
        gp.PlotWaveform(6, cramp.Cut(1,1000), "Convolved Ramp")

    End Sub
End Class
```



Exercises

4.1 Explain the first three values of 'Convolved Output 2' (they are: 1, 4 & 5).

4.2 Modify example program 4.1 so that the impulse response is a square wave.

4.3 Modify example program 4.1 so that the impulse response matches the input waveform in each case.

4.4 Adapt example program 4.2 to send a stream of pulses through the resonator rather than the ramp wave, displaying the input and output signals.

4.5 Adapt example program 4.2 to generate the output from a resonator with a natural frequency of 25Hz and a bandwidth of 7.5Hz when fed with a pulse train at 10Hz lasting 1 second and sampled at 1,000Hz. Display the result. (Hint: create a Signal of 1000 samples at 1000 samples/sec, create a Resonator of centre frequency (25/1000) and bandwidth (7.5/1000), then run the resonator into the signal, putting in pulses every 100 samples)