UNIT 7: DIGITAL FILTER DESIGN

BASICDSP

7.1 Introduction

This unit is concerned primarily with the design of digital systems having frequency response characteristics appropriate to low-pass, high-pass and band-pass filters. It is concerned with both non-recursive and recursive designs: the former based on the Fourier transform method, the second through the so-called 'bilinear' transformation of analogue designs.

When you have worked through this unit you should:

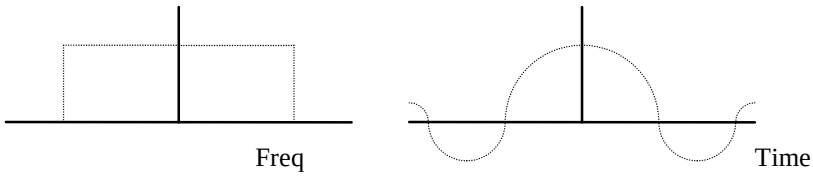
- be able to explain how simple non-recursive filters may be designed by the DFT method
- understand how low-pass non-recursive filters may be transformed into high-pass and band-pass designs
- understand how the Kaiser window provides a means for building a simple non-recursive filter from specifications of the stop-band ripple and the width of the transition region.
- be able to explain how simple recursive filters may be designed through the use of the Butterworth formulae
- understand how the use of cascaded second-order sections provides a means for implementing Butterworth filters from the pole-zero diagram
- understand how recursive low-pass filters may be transformed into high-pass and band-pass designs.
- be able to discuss the relative merits of recursive and non-recursive designs for different applications.

7.2 Concepts

Non-recursive Filter Design

We seek to find a set of a[] (numerator) coefficients for a linear time-invariant system that would implement a low-pass filter: that is the impulse response of a low-pass non-recursive filter. We then consider how to *transform* these coefficients into high-pass or band-pass forms.

We first note that the Fourier transform of a rectangular window has a characteristic shape known as a *sin(x)/x* or *sinc(x)* function. Thus for an ideal low-pass filter response, which has a rectangular shape, the related impulse response must follow this sin(x)/x shape. Since the response of a digital system repeats periodically with frequency (once per sampling frequency), we can choose to consider a rectangular filter within a response that extends from -sample-rate/2 to +sample-rate/2, and this will give us an impulse response that is also symmetric about t=0:



For a cut-off angular frequency of w_c the formula for this impulse response is just:

$$h[n] = \frac{1}{n\pi}\sin(nw_c) = \frac{w_c}{\pi}\mathrm{sinc}(nw_c)$$

where $w_c = 2\pi f_c$, where f_c = fraction of sampling rate, and where n extends over all positive and negative times.

To generate an impulse response of a certain maximum size, it is best to window the resulting coefficients, rather than simply truncate them. This can be easily performed with a Hamming window. We may then shift the filter coefficients in time to get a causal filter. See `NRLowPass()`.

To obtain the impulse response of a high-pass or band-pass filter, we note that all we need do is shift the frequency response curve along the frequency axis - so the rectangular response is now either symmetric about the half-sample-rate frequency - in which case we get a high-pass filter, or is now over a block of frequencies - in which case we get a band-pass filter. The process of shifting the response in the frequency domain is a process of convolution with another response which consists solely of an impulse at the new centre frequency of the block, that is the spectrum of a signal having a single frequency component at the new centre frequency, that is the spectrum of a sinusoidal signal. This convolution in the frequency domain manifests itself in the time domain by a multiplication of the impulse response by a cosine wave of the appropriate frequency. So to shift the impulse response of a low-pass filter up to a new centre angular frequency w, we generate:

$$h'[n] = h[n]\cos(nw)$$

For the special case that the new rate is half the sample rate (i.e. when $w=\pi$) the modulation consists of multiplying h[] by +1, -1, +1, -1, ... See the implementations of `NRHighPass()` and `NRBandPass()` to exemplify this.

While the Hamming window is a simple and useful method for truncating the impulse response, it is rather inflexible. The Hamming window gives us a flat pass band and fairly good attenuation in the stop band, but at the expense of a fairly broad transition band. Sometimes we wouldn't mind more ripple in the pass band if that allowed us to have a narrower transition region. We can achieve this by designing a windowing function with the appropriate properties and use that to truncate the sinc function impulse response. A simple way to achieve this is with the parametric **Kaiser** window.

The Kaiser window is defined as:

$$w[n] = \frac{I_0\left(\alpha\sqrt{1-\left(\frac{n}{M}\right)^2}\right)}{I_0(\alpha)}, \quad -M \le n \le M$$

Where I_0 is the zeroth order modified Bessel function of the first kind, which has a complex mathematical description, but a straightforward algorithmic description shown in Algorithm 7.2. The parameter α controls the tradeoff between transition width and ripple. We can give a heuristic for the selection of α and *M* from a specification of (i) the allowed stop band ripple *A* expressed as decibels below the passband, and (ii) the transition width Δ expressed as a fraction of the sample rate:

if A >= 50,	then $\alpha = 0.1102(A - 8.7)$
if 21 < A < 50,	then $\alpha = 0.5842(A - 21)^{0.4} + 0.07886(A - 21)$
if A <= 21,	then $\alpha = 0$

Where the ripple level is less than 21dB from the pass band, then we end up with a rectangular window. *M*, the half-size of the window is then given by:

$$M \ge \frac{A - 7.95}{28.72\Delta}$$

Algorithm 7.2, Kaiser designs a window according to these formulae, which could be used to substitute for the Hamming window used in `NRLowPass`, etc.

Recursive Filter Design

The design of recursive filters is considerably more complex than the design of non-recursive filters. The design task is to place poles and zeros on the complex plane, such that the frequency response has the desired characteristics. This can either be performed through experience or with an interactive design tool. We choose to look at the design formulae developed for analogue filters: specifically for Butterworth designs. These can then be transformed into the digital domain through the use of an algebraic transformation that maps the analogue frequencies 0..∞ into the periodically repeating angular frequencies 0..2π. The mechanism used is called the *bilinear transformation*.

A digital Butterworth low-pass filter of order *n* has n poles arranged on a circular arc on the complex plane and n zeros located at z = (-1,0). If *n* is even, the poles form *n*/2 conjugate pairs. If *n* is odd, then a pole on the real axis is added. If we keep to even *n* for convenience the location of the poles are given by the formulae:

$$polepair[m] = \frac{RE[m]}{d} \pm i\frac{IM[m]}{d}, \; m = \frac{n}{2},..,n-1$$

where the real and imaginary parts are given by:

$$RE[m] = 1 - \tan^2\left(\frac{w_c}{2}\right)$$

$$IM[m] = 2\tan\left(\frac{w_c}{2}\right)\sin\left(\frac{(2m+1)\pi}{2n}\right)$$

and where

$$d = 1 - 2\tan(\frac{w_c}{2})\cos(\frac{(2m+1)\pi}{2n}) + \tan^2(\frac{w_c}{2})$$

For simplicity of implementation, the routine `ButterworthPoles()` calculates the positions of n/2 poles on the upper half of the complex plane, and then each pole is combined with its complex conjugate to implement a separate linear system (a *second-order section*) in the routine `ButterworthLowPass()`. The resulting chain of linear systems is returned for filtering operation.

The transformation of the low-pass design into high-pass is straightforward: it simply involves reflecting the poles and zeros around the imaginary axis. See `ButterworthHighPass()`.

The transformation of the low-pass design into band-pass is more complex and involves a doubling of the filter order and a rotation of the poles around the unit circle. For a band-pass filter extending from angular frequencies w_l to w_h, we first calculate the poles and zeros for a low pass filter with a cut-off frequency $w_c = (w_h - w_l)/2$. Then each pole or zero at z is moved to a new location by the formula:

$$z' = 0.5A(1+z) \pm \sqrt{0.25A^2(1+z)^2 - z}$$

where A is given by:

$$A = \frac{\cos(\frac{w_h + w_l}{2})}{\cos(\frac{w_h - w_l}{2})}$$

See `ButterworthBandPass()`.

A Butterworth design gives us a filter with a maximally-flat passband and good stop-band attenuation, so that it makes a good general purpose filter. As before, however, other designs can give us a narrower transition region at the cost of greater ripples in the pass- and/or stop-bands. Many DSP packages give design programs for Chebyshev and Elliptic filter designs, but their essential operation is the same as the Butterworth design.

Algorithms

Algorithm 7.1 Non-recursive filter design

```
' sinc(x) = sin(x) / x
Public Function sinc(ByVal x As Double) As Double
    If (x = 0) Then
        Return 1
    Else
        Return Math.Sin(x) / x
    End If
End Function

' Design non-recursive low-pass filter.
Public Shared Function NRLowPass(ByVal freq As Double, ByVal ncoeff As Integer)
As LTISystem

    ' convert frequency
    Dim omega As Double = 2 * Math.PI * freq

    ' build half-sized window from sinc function
    Dim nhalf As Integer = ncoeff \ 2
    Dim hwv As New Waveform(nhalf, 1.0)
    For i As Integer = 1 To nhalf
        hwv(i) = omega * sinc(i * omega) / Math.PI
    Next

    ' window with (half-)hamming window
    For i As Integer = 1 To nhalf
        hwv(i) *= 0.54 + 0.46 * Math.Cos(i * Math.PI / nhalf)
    Next

    ' create new LTI System
    Dim lpfilt As New LTISystem(2 * nhalf, 0)

    ' copy impulse response into system (indexed -nhalf .. 0 .. nhalf)
    lpfilt.a(nhalf) = omega / Math.PI
    For i As Integer = 1 To nhalf
        lpfilt.a(nhalf - i) = hwv(i)
        lpfilt.a(nhalf + i) = hwv(i)
    Next

    Return lpfilt

End Function

' Transform non-recursive low-pass filter to high-pass
Public Shared Function NRHighPassTransform(ByVal lpfilt As LTISystem) As
LTISystem
    Dim hpfilt As New LTISystem(lpfilt)

    ' now modulate with cos(n*PI) = +1,-1,+1,-1,...
    Dim nhalf As Integer = lpfilt.a.Length \ 2
    hpfilt.a(nhalf) = lpfilt.a(nhalf)
    For i As Integer = 1 To nhalf Step 2
        hpfilt.a(nhalf - i) = -lpfilt.a(nhalf - i)
        hpfilt.a(nhalf + i) = -lpfilt.a(nhalf + i)
    Next
    Return hpfilt
End Function

' Transform non-recursive low-pass filter to band-pass
Public Shared Function NRBandPassTransform(ByVal lpfilt As LTISystem, ByVal
freq As Double) As LTISystem
    Dim bpfilt As New LTISystem(lpfilt)

    ' now modulate with cos(n*centrefreq)
    Dim nhalf As Integer = lpfilt.a.Length \ 2
    Dim cf As Double = 2 * Math.PI * freq
    bpfilt.a(nhalf) = 2 * lpfilt.a(nhalf)
```

```
    For i As Integer = 1 To nhalf
        bpfilt.a(nhalf - i) = 2 * lpfilt.a(nhalf - i) * Math.Cos(i * cf)
        bpfilt.a(nhalf + i) = 2 * bpfilt.a(nhalf + i) * Math.Cos(i * cf)
    Next

    Return bpfilt
End Function

' Design non-recursive high-pass filter
Public Shared Function NRHighPass(ByVal freq As Double, ByVal ncoeff As
Integer) As LTISystem
    Dim hpfilt As LTISystem = NRHighPassTransform(NRLowPass(0.5 - freq,
ncoeff))
    Return hpfilt
End Function

' Design non-recursive band-pass filter
Public Shared Function NRBandPass(ByVal lofreq As Double, ByVal hifreq As
Double, ByVal ncoeff As Integer) As LTISystem
    Dim bpfilt As LTISystem = NRBandPassTransform(NRLowPass((hifreq - lofreq) /
2, ncoeff), (lofreq + hifreq) / 2)
    Return bpfilt
End Function
```

Algorithm 7.2 - Kaiser Window

```
' zeroth order modified Bessel function of the first kind
Private Const ITERLIMIT As Integer = 15
Private Const CONVERGE As Double = 0.00000001

Private Shared Function besselI0(ByVal p As Double) As Double
    ' initialise iterative loop
    p = p / 2
    Dim n As Double = 1
    Dim t As Double = 1
    Dim d As Double = 1

    ' iteration
    Dim k As Integer = 1
    Dim v As Double
    Do
        n = n * p
        d = d * k
        v = n / d
        t = t + v * v
        k += 1
    Loop While ((k < ITERLIMIT) And (v > CONVERGE))

    Return t
End Function

' Calculates Kaiser window to specification
Public Shared Function Kaiser(ByVal ripple As Double, ByVal twidth As Double,
ByVal kmaxlen As Integer) As Waveform
    Dim alpha As Double     ' window coefficient
    Dim hlen As Integer     ' half window length
    Dim v As Double, d As Double

    ' set Kaiser window coefficient (design rule)
    If (ripple <= 21) Then
        alpha = 0
    ElseIf (ripple < 50) Then
        alpha = 0.5842 * Math.Exp(0.4 * Math.Log(ripple - 21)) + 0.07886 *
(ripple - 21)
    Else
        alpha = 0.1102 * (ripple - 8.7)
    End If

    ' set Kaiser window size (design rule)
    If (ripple < 7.95) Then
        hlen = 0
    Else
        hlen = 1 + ((ripple - 7.95) / (28.72 * twidth))
    End If
    If ((hlen + 1) > kmaxlen) Then hlen = kmaxlen - 1

    ' build output window
    Dim kwin As New Waveform(hlen + 1, 1.0)

    ' calculate window 1..hlen+1
    d = besselI0(alpha)
    kwin(1) = 1.0
    For n As Integer = 1 To hlen
        v = n / hlen
        kwin(n + 1) = besselI0(alpha * Math.Sqrt(1 - v * v)) / d
    Next

    ' return windows
    Return kwin

End Function
```

Algorithm 7.3 Butterworth Recursive Filter Design - Chain of Linear Systems

```
' Supports chain of linear systems
Public Class LTISystemChain
    Private _nsection As Integer = 0
    Private _section() As LTISystem

    ''' Creates chain of linear systems
    Public Sub New(ByVal n As Integer)
        _nsection = n
        ReDim _section(_nsection - 1)
    End Sub

    ''' Creates copy of chain of linear systems.
    Public Sub New(ByVal lch As LTISystemChain)
        _nsection = lch.NSection
        ReDim _section(_nsection - 1)
        For i As Integer = 0 To lch.NSection - 1
            _section(i) = lch.Section(i)
        Next
    End Sub

    ''' Gets number of linear systems in chain.
    Public ReadOnly Property NSection() As Integer
        Get
            Return _nsection
        End Get
    End Property

    ''' Gets/sets the linear systems in the chain
    Public Property Section(ByVal idx As Integer) As LTISystem
        Get
            Return _section(idx)
        End Get
        Set(ByVal value As LTISystem)
            _section(idx) = value
        End Set
    End Property

    ''' Clears state memory in all linear system chain
    Public Sub Clear()
        For i As Integer = 0 To _nsection - 1
            _section(i).Clear()
        Next
    End Sub

    ''' Processes a single sample value through the chain
    Default Public ReadOnly Property Item(ByVal ival As Double) As Double
        Get
            For i As Integer = 0 To _nsection - 1
                ival = _section(i)(ival)
            Next
            Return ival
        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

    ''' Gets response of linear system chain at given frequency
    Public ReadOnly Property Response(ByVal freq As Double) As Complex
        Get
            If (_nsection = 0) Then Return New Complex(0, 0)
            Dim resp As Complex = _section(0).Response(freq)
            For i As Integer = 1 To _nsection - 1
                resp = resp * _section(i).Response(freq)
            Next
            Return resp
        End Get
    End Property
End Class
```

Butterworth filter design:

```
' Calculates pole positions for Butterworth design low-pass filter
Public Shared Function ButterworthPoles(ByVal freq As Double, ByVal nsection As
Integer) As Complex()

    ' get array of complex values
    Dim poles(nsection - 1) As Complex

    ' calculate angles
    Dim w As Double = Math.PI * freq
    Dim tanw As Double = Math.Sin(w) / Math.Cos(w)

    ' calculate +im pole position for each section
    For m As Integer = nsection To 2 * nsection - 1
        ' Butterworth formula adapted to z-plane
        Dim ang As Double = (2 * m + 1) * Math.PI / (4 * nsection)
        Dim d As Double = 1 - 2 * tanw * Math.Cos(ang) + tanw * tanw
        poles(m - nsection) = New Complex((1 - tanw * tanw) / d, 2 * tanw *
Math.Sin(ang) / d)
    Next
    Return poles
End Function

' Design Butterworth low-pass recursive filter
Public Shared Function ButterworthLowPass(ByVal freq As Double, ByVal nsection
As Integer) As LTISystemChain
    ' create empty system chain
    Dim lpfilt As New LTISystemChain(nsection)

    ' get pole positions
    Dim pol() As Complex = ButterworthPoles(freq, nsection)

    ' convert each conjugate pole pair to difference equation
    For i As Integer = 0 To nsection - 1
        lpfilt.Section(i) = New LTISystem(2, 2)
        ' put in conjugate pole pair
        lpfilt.Section(i).b(1) = -2.0 * pol(i).Real
        lpfilt.Section(i).b(2) = pol(i).Real * pol(i).Real + pol(i).Imag *
pol(i).Imag
        ' put 2 zeros at (-1,0)
        Dim tot As Double = 4.0 / (1 + lpfilt.Section(i).b(1) +
lpfilt.Section(i).b(2))
        lpfilt.Section(i).a(0) = 1.0 / tot
        lpfilt.Section(i).a(1) = 2.0 / tot
```

```
        lpfilt.Section(i).a(2) = 1.0 / tot
    Next

    Return lpfilt

End Function

' Design Butterworth high-pass recursive filter
Public Shared Function ButterworthHighPass(ByVal freq As Double, ByVal nsection
As Integer) As LTISystemChain
    ' create empty system chain
    Dim hpfilt As New LTISystemChain(nsection)

    ' get pole positions for LP prototype
    Dim pol() As Complex = ButterworthPoles(0.5 - freq, nsection)

    ' flip all the poles over to get high pass
    For i As Integer = 0 To nsection - 1
        pol(i) = New Complex(-pol(i).Real(), pol(i).Imag())
    Next

    ' convert each conjugate pole pair to difference equation
    For i As Integer = 0 To nsection - 1
        hpfilt.Section(i) = New LTISystem(2, 2)
        ' put in conjugate pole pair
        hpfilt.Section(i).b(1) = -2.0 * pol(i).Real
        hpfilt.Section(i).b(2) = pol(i).Real * pol(i).Real + pol(i).Imag *
pol(i).Imag
        ' put 2 zeros at (1,0)
        hpfilt.Section(i).a(0) = 1.0
        hpfilt.Section(i).a(1) = -2.0
        hpfilt.Section(i).a(2) = 1.0
        ' normalise to unity gain at Fs/2
        Dim gain As Double = hpfilt.Section(i).Response(0.5).Mag
        hpfilt.Section(i).a(0) = hpfilt.Section(i).a(0) / gain
        hpfilt.Section(i).a(1) = hpfilt.Section(i).a(1) / gain
        hpfilt.Section(i).a(2) = hpfilt.Section(i).a(2) / gain
    Next

    Return hpfilt

End Function

' Design Butterworth band-pass recursive filter
Public Shared Function ButterworthBandPass(ByVal lofreq As Double, ByVal hifreq
As Double, ByVal nsection As Integer) As LTISystemChain
    ' create empty system chain
    If (nsection Mod 2) = 1 Then nsection += 1
    Dim bpfilt As New LTISystemChain(nsection)

    ' get pole positions for LP prototype
    Dim pol() As Complex = ButterworthPoles(hifreq - lofreq, nsection / 2)

    ' translate the poles to band-pass position
    Dim bpol(nsection) As Complex
    Dim wlo As Double = 2 * Math.PI * lofreq
    Dim whi As Double = 2 * Math.PI * hifreq
    Dim ang As Double = Math.Cos((whi + wlo) / 2) / Math.Cos((whi - wlo) / 2)
    For i As Integer = 0 To nsection / 2 - 1
        Dim p1 As New Complex(pol(i).Real() + 1, pol(i).Imag())
        Dim tmp As Complex = Complex.Sqrt(p1 * p1 * ang * ang * 0.25 - pol(i))
        bpol(2 * i) = (p1 * ang * 0.5) + tmp
        bpol(2 * i + 1) = (p1 * ang * 0.5) - tmp
    Next

    ' convert each conjugate pole pair to difference equation
    For i As Integer = 0 To nsection - 1
        bpfilt.Section(i) = New LTISystem(2, 2)
        ' put in conjugate pole pair
        bpfilt.Section(i).b(1) = -2.0 * bpol(i).Real
        bpfilt.Section(i).b(2) = bpol(i).Real * bpol(i).Real + bpol(i).Imag *
bpol(i).Imag
```

```
        ' put zeros at (-1,0) and (1,0)
        bpfilt.Section(i).a(0) = 1.0
        bpfilt.Section(i).a(1) = 0.0
        bpfilt.Section(i).a(2) = -1.0
        ' normalise to unity gain at centre of filter
        Dim gain As Double = bpfilt.Section(i).Response((hifreq + lofreq) /
2).Mag
        bpfilt.Section(i).a(0) = bpfilt.Section(i).a(0) / gain
        bpfilt.Section(i).a(1) = bpfilt.Section(i).a(1) / gain
        bpfilt.Section(i).a(2) = bpfilt.Section(i).a(2) / gain
    Next

    Return bpfilt

End Function
```

Bibliography

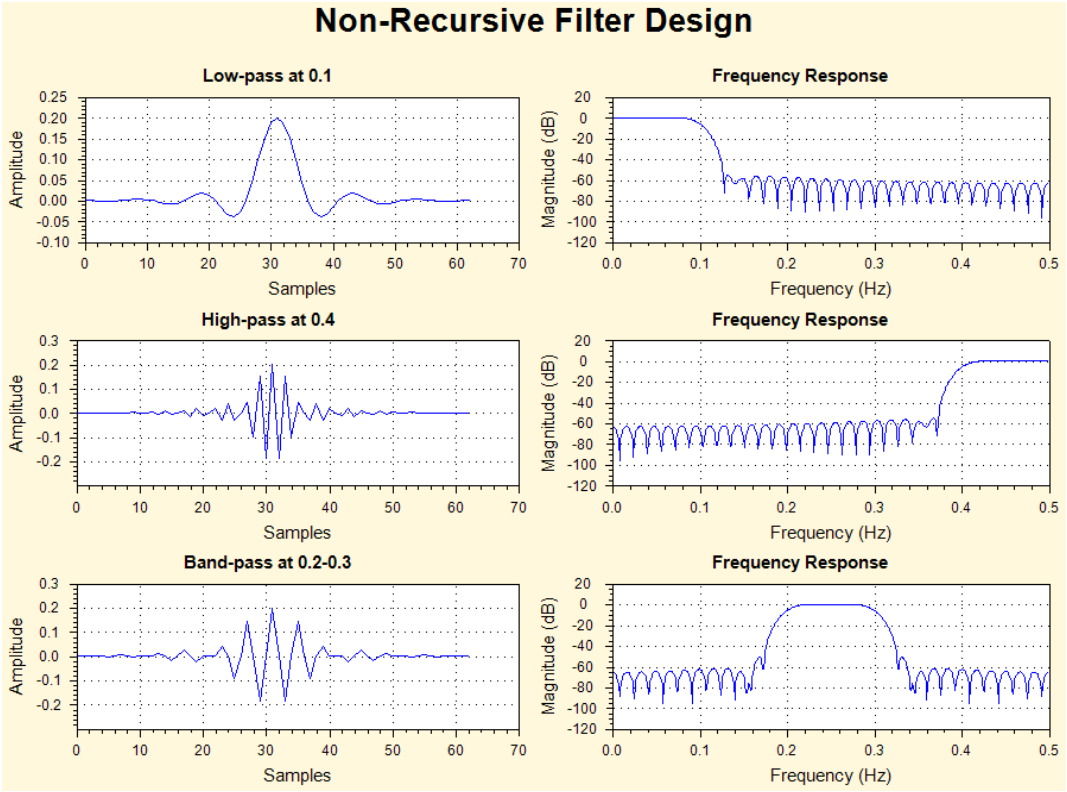
Lynn & Fuerst, Introductory Digital Signal Processing, Chapter 5.

Orfanidis, Introduction to Signal Processing, Chapters 10 & 11.

Example Programs

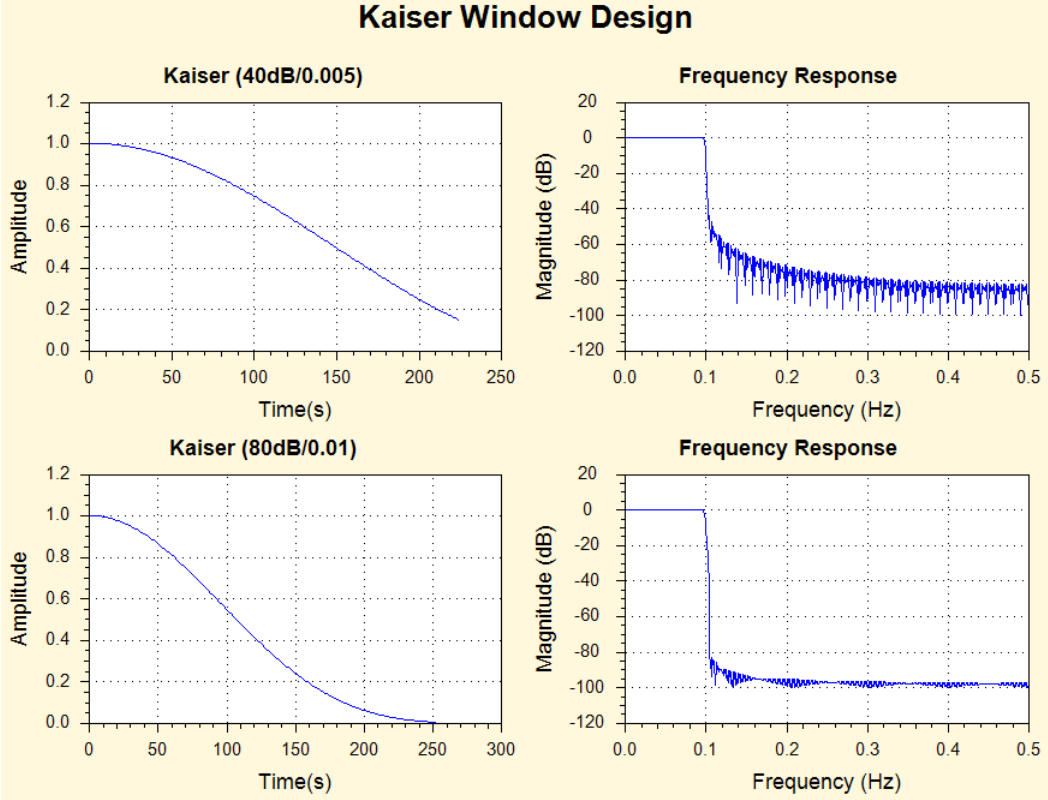
Example 7.1 Non-recursive filter design

```
Imports BasicDSP
Imports ZedGraph
Public Class TestNonRecFilter

    Private Sub TestNonRecFilter_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        ' create graphs
        Dim gp As New Graph(Me.CreateGraphics, zgc, 3, 2, "Non-Recursive Filter
Design")

        ' calculate low-pass filter and plot
        Dim lp As LTISystem = Filter.NRLowPass(0.1, 63)
        gp.PlotCoeff(1, lp.a, "Low-pass at 0.1", "Amplitude", "Samples")

        ' calculate frequency response and plot
        Dim lpf As New Spectrum(500, 0.5)
        For i As Integer = 0 To 499
            lpf(i) = lp.Response(i / 1000.0)
        Next
        gp.PlotDbSpectrum(2, lpf, "Frequency Response")

        ' calculate high-pass filter and plot
        Dim hp As LTISystem = Filter.NRHighPass(0.4, 63)
        gp.PlotCoeff(3, hp.a, "High-pass at 0.4", "Amplitude", "Samples")

        ' calculate frequency response and plot
        Dim hpf As New Spectrum(500, 0.5)
        For i As Integer = 0 To 499
            hpf(i) = hp.Response(i / 1000.0)
        Next
        gp.PlotDbSpectrum(4, hpf, "Frequency Response")

        ' calculate band-pass filter and plot
        Dim bp As LTISystem = Filter.NRBandPass(0.2, 0.3, 63)
        gp.PlotCoeff(5, bp.a, "Band-pass at 0.2-0.3", "Amplitude", "Samples")

        ' calculate frequency response and plot
        Dim bpf As New Spectrum(500, 0.5)
        For i As Integer = 0 To 499
            bpf(i) = bp.Response(i / 1000.0)
        Next
        gp.PlotDbSpectrum(6, bpf, "Frequency Response")

    End Sub
End Class
```


Non-Recursive Filter Design
Low-pass at 0.1
Amplitude
Samples
Frequency Response
Magnitude (dB)
Frequency (Hz)
High-pass at 0.4
Amplitude
Samples
Frequency Response
Magnitude (dB)
Frequency (Hz)
Band-pass at 0.2-0.3
Amplitude
Samples
Frequency Response
Magnitude (dB)
Frequency (Hz)

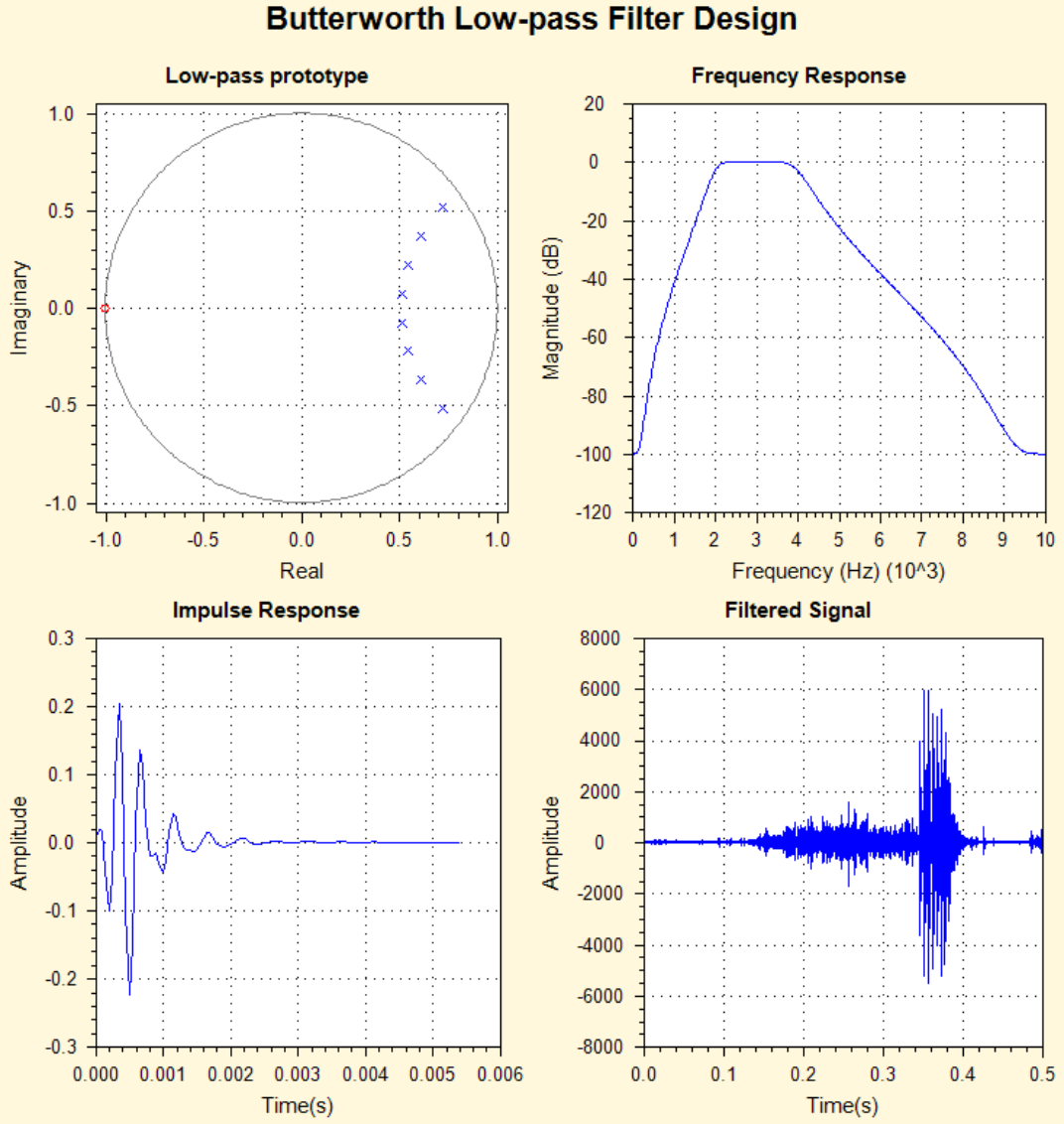
Example 7.2 - Demonstrate Kaiser Window

```
Imports BasicDSP
Imports ZedGraph
Public Class TestKaiser
    Const MAXKAISERWIN As Integer = 256
    ' Create non-recursive low-pass filter using Kaiser window
    ' freq      corner freq  (fraction of sampling rate)
    ' ripple    allowed ripple (dB)
    ' twidth    transition width  (fraction of sampling rate)
    Public Function KaiserLowPass(ByVal freq As Double, ByVal ripple As Double,
ByVal twidth As Double) As LTISystem

        ' get (half-)Kaiser window
        Dim kwin As Waveform = Window.Kaiser(ripple, twidth, MAXKAISERWIN)
        Dim nhalf As Integer = kwin.Count() - 1

        ' generate one half of coefficients from windowed sinc() function
        Dim omega As Double = 2 * Math.PI * freq
        For i As Integer = 0 To nhalf
            kwin(i + 1) *= omega * sinc(i * omega) / Math.PI
        Next

        ' copy into LTI System
        Dim lpfilt As New LTISystem(2 * nhalf, 0)
        lpfilt.a(nhalf) = kwin(1)
        For i As Integer = 1 To nhalf
            lpfilt.a(nhalf - i) = kwin(i + 1)
            lpfilt.a(nhalf + i) = kwin(i + 1)
        Next

        Return lpfilt
    End Function

    Private Sub TestKaiser_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        ' initialise graphics
        Dim gp As New Graph(Me.CreateGraphics, zgc, 2, 2, "Kaiser Window
Design")

        ' plot Kaiser window 1
        Dim kwv1 As Waveform = Window.Kaiser(40.0, 0.005, MAXKAISERWIN)
        gp.PlotWaveform(1, kwv1, "Kaiser (40dB/0.005)")

        ' calculate and plot frequency response
        Dim lp1 As LTISystem = KaiserLowPass(0.1, 40.0, 0.005)
        Dim lpf1 As New Spectrum(500, 0.5)
        For i As Integer = 0 To 499
            lpf1(i) = lp1.Response(i / 1000.0)
        Next
        gp.PlotDbSpectrum(2, lpf1, "Frequency Response")

        ' plot Kaiser window 2
        Dim kwv2 As Waveform = Window.Kaiser(80.0, 0.01, MAXKAISERWIN)
        gp.PlotWaveform(3, kwv2, "Kaiser (80dB/0.01)")

        ' calculate and plot frequency response
        Dim lp2 As LTISystem = KaiserLowPass(0.1, 80.0, 0.01)
        Dim lpf2 As New Spectrum(500, 0.5)
        For i As Integer = 0 To 499
            lpf2(i) = lp2.Response(i / 1000.0)
        Next
        gp.PlotDbSpectrum(4, lpf2, "Frequency Response")

    End Sub
End Class
```


Kaiser Window Design
Kaiser (40dB/0.005)
Amplitude
Time(s)
Frequency Response
Magnitude (dB)
Frequency (Hz)
Kaiser (80dB/0.01)
Amplitude
Time(s)
Frequency Response
Magnitude (dB)
Frequency (Hz)

E**xample 7.3** Recursive filter design

```
Imports BasicDSP
Imports ZedGraph
Public Class TestButter
    Const ILIMIT As Double = 0.00001    ' length limit for impulse response
    Const FILENAME As String = "c:/sfs/demo/six.wav"
    Const SAMPLE As Double = 0.5        ' how much waveform to display
    Const CUTFREQ As Double = 2000      ' cut-off frequency in Hertz
    Const NSECTION As Integer = 4       ' number of filter sections

    Private Sub TestButter_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        ' initialise graphics
        Dim gp As New Graph(Me.CreateGraphics, zgc, 2, 2, "Butterworth Low-pass
Filter Design")

        ' get waveform to set sample rate for graphs
        Dim isig As New Signal(FILENAME)

        ' get poles and plot them
        Dim pol() As Complex = Filter.ButterworthPoles(CUTFREQ / isig.Rate,
NSECTION)
        Dim poles(2 * pol.Length - 1) As Complex
        Dim zeros(0) As Complex
        For i As Integer = 0 To pol.Length - 1
            poles(i) = New Complex(pol(i))
            poles(pol.Length + i) = New Complex(pol(i).Real, -pol(i).Imag)
        Next
        zeros(0) = New Complex(-1, 0)
        gp.PlotZPlane(1, poles, zeros, "Low-pass prototype")

        ' build chain of second order sections
        Dim lpfilt As LTISystemChain = Filter.ButterworthBandPass(CUTFREQ /
isig.Rate, 4000 / isig.Rate, NSECTION)

        ' calculate frequency response
        Dim lpfr As New Spectrum(500, isig.Rate / 2)
        For i As Integer = 0 To 499
            lpfr(i) = lpfilt.Response(i / 1000.0)
        Next
        gp.PlotDbSpectrum(2, lpfr, "Frequency Response")

        ' measure and plot impulse response
        Dim lpir As New Waveform(0, isig.Rate)
        Dim lval As Double = 0          ' last output
        Dim oval As Double = lpfilt(1.0)    ' put in unit pulse
        While ((Math.Abs(oval) > ILIMIT) Or (Math.Abs(oval - lval) > ILIMIT) Or
(lpir.Count < 100))
            lpir.Add(oval)          ' append sample
            lval = oval             ' remember sample
            oval = lpfilt(0.0)      ' get next sample
        End While
        gp.PlotWaveform(3, lpir, "Impulse Response")

        ' plot some filtered speech
        isig = isig.Cut(1, SAMPLE * isig.Rate)
        Dim fwv As New Waveform(isig.Count, isig.Rate)
        lpfilt.Clear()
        For i As Integer = 1 To isig.Count
            fwv(i) = lpfilt(isig(i))
        Next
        gp.PlotWaveform(4, fwv, "Filtered Signal")

    End Sub
End Class
```



Exercises

7.1. Implement a band-pass filter that emulates the telephone system, using a non-recursive filter with 32 coefficients with a response extending from 300Hz to 3500Hz. Use it to filter and display a speech signal along with some representative spectra and a frequency response curve.

7.2. Adapt the program from exercise 7.1 to use a recursive filter with 2 second-order sections. Find methods for timing the execution speed of programs and compare the efficiency of this program to the original (you may need to comment out the display part for timing).