UNIT 8:
LINEAR PREDICTION

BASICDSP

8.1 Summary

Linear prediction is a method for signal source modelling dominant in speech signal processing and having wide application in other areas. Starting with a demonstration of the relationship between linear prediction and the general difference equation for linear systems, the unit shows how the linear prediction equations are formulated and solved. The unit then discusses the use of linear prediction for modelling the source of a signal and the signal spectrum.

When you have worked through this unit you should:

- understand how linear prediction can provide a model of a signal source
- be able to state the operational limitations of the modelling
- understand in qualitative terms how the equations are solved on a computer system
- know how and when to use linear prediction as an alternative to spectral analysis by the Fourier transform

8.2 Concepts

Nature of linear prediction

The object of linear prediction is to form a model of a linear time-invariant digital system through observation of input and output sequences. That is: to estimate a set of coefficients which will describe the behaviour of an LTI system when its design is not available to us and we cannot choose what input to present.

Specifically, linear prediction calculates a set of coefficients which provide an estimate - or *a prediction* - for a forthcoming output sample y'[n] given knowledge of previous input (x[]) and/or output (y[]) samples:

$$y'[n] = \sum_{k=0}^{p} a_k x[n-k] - \sum_{k=1}^{q} b_k y[n-k]$$

Where *a* and *b* are called *predictor coefficients*. It is immediately clear that this estimate of the next output sample given the predictor coefficients and previous input and output samples is directly related to the general system difference equation we met in Unit 4. Even though we might not know the order of some LTI system, such an estimator is an inherently sensible way to model its properties.

The commonest form of linear prediction used in signal processing is one in which the *a* coefficients are zero, so that the output estimate is made entirely on the basis of previous output samples:

$$y'[n] = -\sum_{k=1}^{q} b_k y[n-k]$$

This is called an *all-pole* model, which may be seen by analogy with the frequency response equations of LTI systems. The use of an all pole model is motivated by a number of reasons:

(i) we often do not have access to the input sequence,
(ii) many simple signal sources are faithfully modelled by an all-pole model,
(iii) the all-pole model gives a system of equations which may be efficiently solved.

The relation of the linear predictor equation with the LTI system equation may be more readily understood by introducing a *predictor error* signal e[n], defined as the difference between the real output and the prediction:

$$e[n] = y[n] - y'[n]$$

Hence we may write:

$$y[n] = e[n] - \sum_{k=1}^{q} b_k y[n-k]$$

giving us a formulation in which the error signal takes the role of the excitation signal, and the predictor coefficients define the filter.

Solution of linear prediction equations

To estimate the predictor coefficients b_k we first observe the behaviour of the system over N samples, and set the order q of the predictor required (usually N is much greater than q). We calculate the predictor coefficients by finding values which minimise the energy in the error signal over the N samples[1]. This is called a *least-squares* minimisation, and leads to a system of q equations in q unknowns which may be solved to find the best fitting predictor coefficients:

$$\sum_{k=0}^{q}\left(b_k \sum_{n=0}^{N-1} y[n-k]y[n-j]\right) = 0, \quad j = 1,2,\ldots,q$$

Where b_0 is taken to be 1.

There are a number of methods for finding the solution to this system of linear equations. The formulation above is sometimes known as the *covariance* formulation and is appropriate when estimating coefficients from a sample of a non-stationary signal. If instead we assume that the signal is zero outside the N sample region, we can find a simpler formulation:

$$\sum_{k=0}^{q}\left(b_k \sum_{n=j}^{N-1} y[n]y[n-k+j]\right) = 0, \quad j = 1,2,\ldots,q$$

This is called the *autocorrelation* formulation, since it contains components equivalent to the autocorrelation coefficient r_i defined as:

[1] This is equivalent to flattening the spectrum of the error signal.

$$r_i = \sum_{n=-\infty}^{\infty} y[n]y[n-i]$$

The autocorrelation formulation of the least squares fit of the predictor coefficients produces a system of linear equations which can be represented in matrix form and solved using standard methods such as Gaussian elimination. See `LPCSolve()`.

However the autocorrelation system of equations can be solved much more efficiently since the autocorrelation coefficients in the matrix form of the equations have a very simple and symmetric structure. This allows for a recursive solution procedure in which each predictor coefficient may be derived in turn and used to calculate the next coefficient. This is the dominant method for solving the linear prediction equations, and is usually credited to Levinson and Durbin. See `LPCAutocorrelation()`.

Spectrum Modelling

The determination of the *b* coefficients from some section of signal provides us with an all-pole filter that characterises the source on the assumption that the input signal is white (i.e. has a flat spectrum). Thus the frequency response of the derived filter gives a smooth model of the signal spectrum under the predictor assumptions. See Example program 8.2.

Root Solving

It is also possible to determine the location of the individual poles that make up the linear prediction derived filter - this is useful as the poles can often be considered as being caused by resonances in the driving system. To find the poles, it is necessary to factorise the predictor polynomial, this can be performed numerically through a root-solving procedure, see `FindRoots()`. The location of the poles can then be interpreted in terms of the frequencies and bandwidths of the system resonances.

Bibliography

T. Parsons, Voice and Speech Processing, McGraw-Hill 1987, Ch 6.
Markel & Grey, Linear Prediction of Speech, Springer-Verlag, 1976.

Algorithms

Algorithm 8.1 Linear Prediction Solution - Direct Method

```
Const EPSILON As Double = 0.0000000001

' Gauss-Jordan elimination on set of simultaneous equations
Private Shared Function Eliminate(ByRef mat(,) As Double, ByVal n As Integer)
As Double

    Dim pos As Integer          ' pivot row
    Dim max As Double           ' maximum value in column
    Dim det As Double = 1.0      ' determinant

    ' for each row
    For i As Integer = 1 To n

        ' find pivot row from max column entry
        max = 0
        pos = 1
        For j As Integer = i To n
            If (Math.Abs(mat(j, i)) > max) Then
                max = Math.Abs(mat(j, i))
                pos = j
            End If
        Next

        ' check for column of zeros
        If (max < EPSILON) Then Return (0.0)

        ' transpose current row with pivot row
        ' and normalise diagonal element to unity
        max = mat(pos, i)
        For j As Integer = 1 To n + 1
            Dim temp As Double = mat(pos, j)
            mat(pos, j) = mat(i, j)
            mat(i, j) = temp / max
        Next

        ' keep record of determinant
        If (i <> pos) Then
            det = det * -max
        Else
            det = det * max
        End If

        ' reduce matrix by dividing through by row
        For k As Integer = 1 To n
            If (k <> i) Then
                Dim val As Double = mat(k, i)
                For j As Integer = i To n + 1
                    mat(k, j) = mat(k, j) - val * mat(i, j)
                Next
            End If
        Next
    Next

    ' return determinant
    Return det

End Function
```

```
' set up and solve LPC equations
Public Shared Function Solve(ByVal x As Waveform, ByVal order As Integer) As
LTISystem

    ' compute autocorrelations
    Dim r(order) As Double
    Dim ltis As New LTISystem(0, order)
    Dim sum As Double

    For i As Integer = 0 To order
        sum = 0
        For k As Integer = 1 To x.Count - i
            sum += x(k) * x(k + i)
        Next
        r(i) = sum
    Next

    ' build set of linear equations
    Dim mat(order, order + 1) As Double
    For i As Integer = 1 To order
        For j As Integer = 1 To order
           mat(i, j) = r(Math.Abs(i - j))
        Next
        mat(i, order + 1) = -r(i)
    Next

    ' solve them
    If (Eliminate(mat, order) = 0) Then
        ltis.a(0) = 0.0
        Return ltis
    End If

    ' copy coefficients into LTI System
    ltis.a(0) = 1.0
    ltis.b(0) = 1.0
    For i As Integer = 1 To order
        ltis.b(i) = mat(i, order + 1)
    Next

    ' OK
    Return ltis

End Function
```

Algorithm 8.2 Linear Prediction Solution - Autocorrelation Method

```
' Find predictor coefficients from waveform by Levinson-Durbin recursion method
Public Shared Function Auto(ByVal x As Waveform, ByVal order As Integer, ByRef
pe As Double) As LTISystem
    Dim r(order) As Double
    Dim ltis As New LTISystem(0, order)
    Dim sum As Double

    ' compute autocorrelations
    For i As Integer = 0 To order
        sum = 0
        For k As Integer = 1 To x.Count - i
            sum += x(k) * x(k + i)
        Next
        r(i) = sum
    Next

    ' check power in signal
    If (r(0) = 0) Then
        ' no signal !!
        pe = 0
        Return ltis
    End If
```

```
    ' compute predictor coefficients
    Dim pc(order) As Double             ' temporary array
    pe = r(0)                           ' initialise error to total power
    pc(0) = 1.0                         ' first coefficient (b[0]) must = 1

    ' for each coefficient in turn
    For k As Integer = 1 To order

        ' find next coeff from pc[] and r[]
        sum = 0
        For i As Integer = 1 To k
            sum -= pc(k - i) * r(i)
        Next
        pc(k) = sum / pe

        ' perform recursion on pc[]
        For i As Integer = 1 To k \ 2
            Dim pci As Double = pc(i) + pc(k) * pc(k - i)
            Dim pcki As Double = pc(k - i) + pc(k) * pc(i)
            pc(i) = pci
            pc(k - i) = pcki
        Next

        ' calculate residual error
        pe = pe * (1.0 - pc(k) * pc(k))
        If (pe <= 0) Then
            ' no power left in signal!
            Return ltis
        End If
    Next

    ' copy coefficients into LTI System
    ltis.a(0) = 1.0
    ltis.b(0) = 1.0
    For i As Integer = 1 To order
        ltis.b(i) = pc(i)
    Next

    ' return OK
    Return ltis

End Function
```

Algorithm 8.4 Root Solving the Predictor Polynomial

```
Public Class Roots

    Private Const ROUND_ERROR As Double = 0.00000006
    Private Const IM_RANGE As Double = 0.000001

    ' Find single root using Laguerre's method
    Private Shared Function Lagmethod(ByVal p() As Complex, ByVal order As
Integer, ByVal x0 As Complex)

        ' iteratively improve initial estimate x0
        For iter As Integer = 1 To 100
            ' evaluate polynomial and its first and second derivatives at x0
            Dim x As Complex = x0
            Dim xp As Complex = 1
            Dim p0 As Complex = p(0) + p(1) * x
            Dim p1 As Complex = p(1)
            Dim p2 As Complex = 0
            For i As Integer = 2 To order
                p0 += p(i) * xp * x * x
                p1 += i * p(i) * xp * x
                p2 += i * (i - 1) * p(i) * xp
                xp = xp * x
            Next
            ' use Laguerre's iteration to improve estimate
```

```
            Dim G As Complex = p1 / p0
            Dim H As Complex = G * G - p2 / p0
            Dim a1 As Complex = order / (G + Complex.Sqrt((order - 1) * (order
* H - G * G)))
            Dim a2 As Complex = order / (G - Complex.Sqrt((order - 1) * (order
* H - G * G)))
            If (a1.Mag < a2.Mag) Then
                x0 = x - a1
                If (a1.Mag < x0.Mag * ROUND_ERROR) Then Return (x0)
            Else
                x0 = x - a2
                If (a2.Mag < x0.Mag * ROUND_ERROR) Then Return (x0)
            End If
        Next
        Return (x0)
    End Function

    ''' Solve a real polynomial of order N into N complex roots
    Public Shared Function FindRoots(ByVal ap() As Double, ByVal m As Integer)
As Complex()
        Dim p(m) As Complex
        Dim rroots(m) As Complex

        ' copy polynomial coefficients into complex array
        For j As Integer = 0 To m
            p(j) = ap(j)
        Next

        ' find each root in turn
        For j As Integer = m To 1 Step -1
            ' find a root, using origin as initial estimate
            Dim x As Complex = Lagmethod(p, j, New Complex(0, 0))
            ' eliminate very small imaginary parts
            If Math.Abs(x.Imag) <= (IM_RANGE * Math.Abs(x.Real)) Then x.Imag =
0.0
            rroots(j) = x
            ' divide polynomial through by found root
            Dim bp As Complex = p(j)
            For k As Integer = j - 1 To 0 Step -1
                Dim c As Complex = p(k)
                p(k) = bp
                bp = c + x * bp
            Next
        Next

        ' sort roots by increasing Arg
        For j As Integer = 2 To m
            Dim x As Complex = rroots(j)
            Dim i As Integer = j
            While (i > 1) AndAlso (x.Arg < rroots(i - 1).Arg)
                rroots(i) = rroots(i - 1)
                i = i - 1
            End While
            rroots(i) = x
        Next

        Return rroots
    End Function
End Class
```

Example Programs

Example 8.1 Demonstration of Linear Prediction

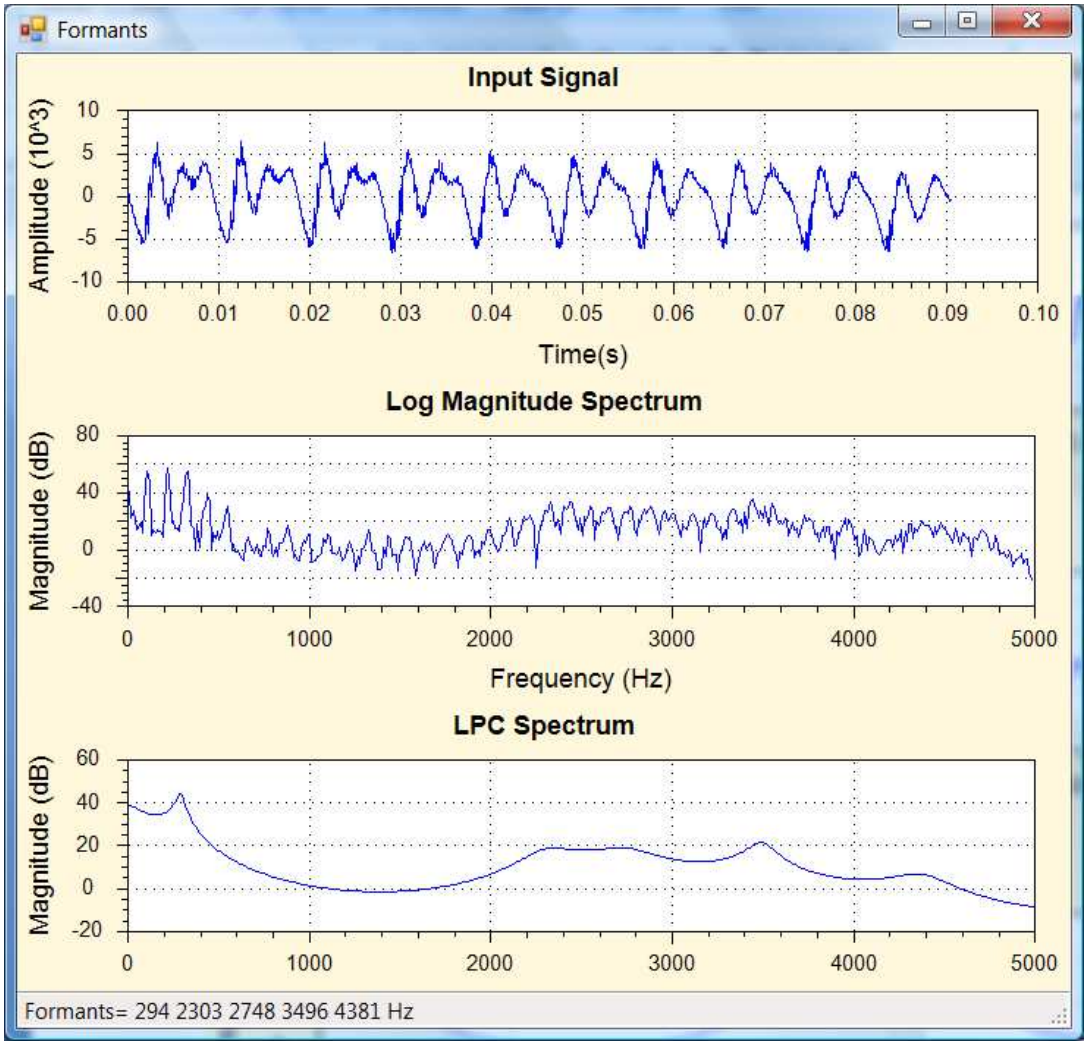
```
Imports BasicDSP
Module Module1
    Const WINDOWSIZE As Integer = 32
    Const NCOEFF As Integer = 4
    Const C1 As Double = 0.16, C2 As Double = -0.12, C3 As Double = 0.08, C4 As
Double = -0.04

    Sub Main()
        Dim iwin As New Waveform(WINDOWSIZE, 1)

        ' make a test signal from recursive filter
        iwin(1) = 0.75          ' put in pulse of power sqr(0.75)
        For i As Integer = 2 To WINDOWSIZE
            ' exploits bad index capability
            iwin(i) = -C1 * iwin(i - 1) - C2 * iwin(i - 2) - C3 * iwin(i - 3) -
C4 * iwin(i - 4)
        Next

        ' do LPC analysis
        Dim osys As LTISystem = LPC.Solve(iwin, NCOEFF)

        If (osys.a(0) = 0) Then
            Console.WriteLine("Analysis failed")
        Else
            Console.WriteLine("Predictor Coefficients:")
            For i As Integer = 1 To NCOEFF
                Console.Write("Coeff{0}={1:F2} ", i, osys.b(i))
            Next
            Console.WriteLine()
        End If

        ' do LPC analysis
        Dim pe As Double
        osys = LPC.Auto(iwin, NCOEFF, pe)

        If (osys.a(0) = 0) Then
            Console.WriteLine("Analysis failed")
        Else
            Console.WriteLine("Predictor Coefficients:")
            For i As Integer = 1 To NCOEFF
                Console.Write("Coeff{0}={1:F2} ", i, osys.b(i))
            Next
            Console.WriteLine(" residual={0:F4}", pe)
        End If

        Console.ReadLine()

    End Sub

End Module
```

Console Output:

```
Predictor Coefficients:
Coeff1=0.16 Coeff2=-0.12 Coeff3=0.08 Coeff4=-0.04
Predictor Coefficients:
Coeff1=0.16 Coeff2=-0.12 Coeff3=0.08 Coeff4=-0.04  residual=0.5625
```

Example 8.2 Linear Prediction Spectrum and Root Solving to find formants

```
Imports BasicDSP
Public Class Formants

    Private Sub Formants_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        ' load a piece of test signal
        Dim isig As New Signal("c:/sfs/demo/six.wav")
        Dim x As Waveform = isig.Cut(24120, 1810).Float

        ' plot the input waveform
        Dim gr As New Graph(Me.CreateGraphics, zgc, 3, 1)
        gr.PlotWaveform(1, x, "Input Signal")

        ' calculate a spectrum of the windowed signal
        Dim y As Spectrum = DFT.ComplexDFT(Window.Hamming(x.Complex))

        ' plot the lower 5000Hz of the spectrum
        Dim hz5000 As Integer = 5000 * y.Count / y.Rate
        gr.PlotDbSpectrum(2, y.Cut(0, hz5000), "Log Magnitude Spectrum")

        ' calculate an LP model
        Dim pe As Double
        Dim np As Integer = x.Rate / 1000
        Dim pc As LTISystem = LPC.Auto(x, np, pe)

        ' calculate frequency response
        Dim r As New Spectrum(500, 5000)
        For i As Integer = 0 To 499
            r(i) = pc.Response((5000 * i / 500) / x.Rate)
        Next

        ' plot frequency response
        gr.PlotDbSpectrum(3, r, "LPC Spectrum")

        ' get roots of filter polynomial
        Dim rroots() As Complex = Roots.FindRoots(pc.b, np)

        ' display frequencies
        Dim msg As String = "Formants="
        For i As Integer = 1 To np
            Dim f As Integer = Math.Atan2(rroots(i).Imag, rroots(i).Real) *
x.Rate / (2 * Math.PI)
            If (f > 0) And (f < 5000) Then
                msg = msg & " " & f
            End If
        Next
        txtStatus.Text = msg & " Hz"
    End Sub
End Class
```



Exercise

8.1 Generate a whispered vowel by passing noise through 3 resonators at say 500, 1500 and 2500Hz (with 100Hz bandwidths) and then use LPC to estimate the frequencies of the resonators from the signal. Display the signal, the FFT spectrum and the LPC spectrum as in Example 8.2. How might you find the exact locations of the peaks in the LPC spectrum?