UNIT 5: DISCRETE FOURIER TRANSFORM

BASICDSP

5.1 Introduction

This unit introduces the Discrete Fourier Transform as a means for obtaining a frequency based representation of a digital signal. The special characteristics of the Fast Fourier Transform implementation are described.

When you have worked through this unit you should:

- be able to explain what information is represented on a magnitude spectrum and on a phase spectrum of a discrete signal
- be able to state the mathematical expression for the Discrete-time discrete-frequency Fourier Transform (DFT)
- understand how the direct implementation of the DFT operates
- appreciate that the direct implementation of the DFT is very inefficient, and that the Fast discrete-time discrete-frequency Fourier Transform (FFT) provides a more efficient means of its calculation
- have a qualitative understanding of the operation of the decimation-in-time form of the FFT
- be able to predict how simple sine and cosine signals appear on the DFT in the region from 0 to F_s.

5.2 Concepts

A digital signal of finite duration x[1..N] can be specified in the time domain as a sequence of N scaled impulses occurring at regular sampling instants: each impulse taking on the amplitude of the signal at that instant. The same signal may also be described as a combination of N complex sinusoidal components X[0..N-1], each of a given frequency and phase, and each being a harmonic of the sampling rate/N. This representation, called a frequency-domain representation, may be obtained from the time-domain form through the use of the *Discrete Fourier Transform* or *DFT*. The time domain form and the frequency domain form are simply different ways of representing the same digital signal, one is chosen over the other simply in terms of utility for a given purpose.

The Discrete Fourier Transform X(f) of a signal x[1..N] is defined as:

$$X[k] = \frac{1}{N} \sum_{n=0}^{N-1} x[n+1] \; e^{-i\frac{2\pi kn}{N}}$$

where X[k] is the amplitude of the k*th* harmonic, where k varies from 0 to N-1 and where k/N represents a fractional frequency value of the sampling rate. In general X[] and x[] can hold complex values, and X[] will be complex even if x[] is purely real.

The graph of |X(f)| against frequency is known as the *magnitude spectrum*. The graph of arg X(f) against frequency is known as the *phase spectrum*.

By copying a real digital signal into a complex sequence x[1..N], the DFT has a straightforward algorithmic implementation (shown in `ComplexDFT()`), using complex multiplication with the complex exponential defined as:

$$e^{a+ib} = a\cos(b) + ia\sin(b)$$

and hence:

$$e^{-i\frac{2\pi kn}{N}} = \cos(\frac{2\pi kn}{N}) + i\sin(\frac{2\pi kn}{N}).$$

We can also define the inverse transform, from the frequency representation X[] back to the time representation x[]:

$$x[n] = \sum_{k=0}^{N-1} X[k]\ e^{i\frac{2\pi(n-1)k}{N}}$$

where n varies from 1 to N. We have chosen to place the scaling factor 1/N in the forward transform, but many authorities place it in the inverse transform. We choose to place it here because it leads to harmonic amplitudes which are normalised to the sequence length, and hence independent of the amount of signal analysed. This leads naturally to an equivalence between the energy in the signal and the energy in the spectrum:

$$Average\ Energy = \frac{1}{N}\sum_{n=1}^{N} x[n]^2 = \sum_{k=0}^{N-1} |X[k]|^2$$

The frequency spectrum is often displayed in log magnitude terms in units of *decibels (dB)*, and may be converted using:

$$A[k](dB) = 10\log_{10}(|X[k]|^2) = 20\log_{10}(|X[k]|)$$

From a real signal at a sampling rate F_s, the DFT provides N harmonic amplitudes at frequencies from 0 to $\left(\frac{N-1}{N}\right)F_S$. However the frequencies from 0 to $F_s/2$ are aliased to the region $F_s/2$ to F_s, so only the lower N/2 amplitudes are important.

The phase angles may be converted to degrees using:

$$P[k] = \frac{360}{2\pi}arg(X[k])$$

The *Fast Fourier Transform* or *FFT* is simply a particular implementation of the DFT, that gives identical results, but which takes considerably less calculation. It does this by eliminating a great deal of redundant computation in the DFT in the circumstances when the sequence length N is a power of 2 (i.e. 4, 8, 16, 32, 64, etc).

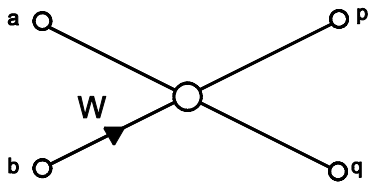
In a normal DFT, each harmonic amplitude is the result of N complex multiplies with N different complex exponentials - giving a total of N^2 multiplies for all N harmonics. When N is a power of 2, many of these multiplies concern identical numerical multiplicands and many of the complex exponentials are zero or 1. When redundant

computation is removed, the number of multiplies is $N\log_2(N)$ rather than N^2 and this represents a very large saving when N is large (e.g. for 1024 samples there is 100 times less calculation).

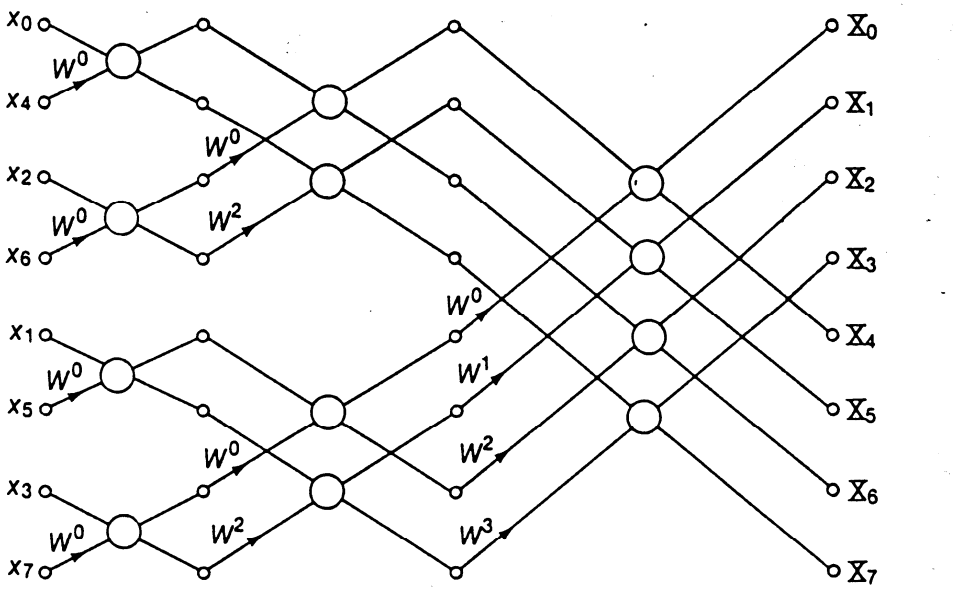
At the heart of the FFT is a simple algebraic manipulation that takes two input values, performs a multiply operation with a complex exponential and produces two output values. This basic unit is called a 'butterfly', and for two complex inputs a and b, a frequency W (=2πkn/N), and outputs p and q, the butterfly calculation is

$$p = a + be^{-iW}. \quad q = a - be^{-iW}.$$

We shall diagram this basic operation as:



This actually represents the DFT of a 2 sample waveform. Longer waveforms can be processed by combining these butterfly operations with variations on the value of W. Thus a DFT of an 8 sample waveform x[0] to x[7] can be graphed as:



Where W^j is one of the frequencies in the DFT calculation (=2πj/N). This **signal flow graph** is the basis of the `ComplexFFT()` implementation. To operate the graph, the input signal is shuffled into the spectrum array in an order known as **bit-reversed addressing**. Then each column of butterfly operations is performed, so that the signal 'moves' left-to-right through the graph, turning into the DFT spectrum.

Implementation Note

The FFT routines have not been written for maximum efficiency. Note that the calculation of the bit-reversed addresses and the values of the complex exponentials need only be performed once for a given size of transform. Since typical use of the FFT is the repeated use of the transform on constant lengths of waveform, these tables may be pre-calculated and stored.

Bibliography

Meade & Dillon, Signals and Systems, Chapter 7, pp130-144.

Lynn & Fuerst, Introductory Digital Signal Processing, Chapter 7.

Orfanidis, Introductory Signal Processing, Chapter 9.

Algorithms

Algorithm 5.1 Complex to Complex Discrete Fourier Transform

```
' Computes complex discrete Fourier transform
Public Shared Function ComplexDFT(ByVal x As ComplexWaveform) As Spectrum

    Dim s As New Spectrum(x.Count, x.Rate)

    ' for each output harmonic
    For i As Integer = s.First To s.Last
        ' get frequency
        Dim f As Double = (2 * Math.PI * i) / s.Count
        ' compute complex sum
        Dim sum As New Complex(0)
        For j As Integer = 0 To x.Count
            sum += x(j + 1) * Complex.Exp(New Complex(0.0, -f * j))
        Next
        ' scale
        s(i) = sum / x.Count
    Next
    Return s
End Function

' Computes complex inverse discrete Fourier transform
Public Shared Function ComplexIDFT(ByVal s As Spectrum) As ComplexWaveform

    Dim x As New ComplexWaveform(s.Count, s.Rate)

    ' for each output sample
    For i As Integer = 0 To x.Count
        ' get frequency
        Dim f As Double = (2 * Math.PI * i) / x.Count()
        ' compute complex sum
        Dim sum As New Complex(0)
        For j As Integer = 0 To x.Count
            sum += s(j) * Complex.Exp(New Complex(0.0, f * j))
        Next
        x(i + 1) = sum
    Next
    Return x
End Function
```

Algorithm 5.2 Complex Fast Fourier Transform

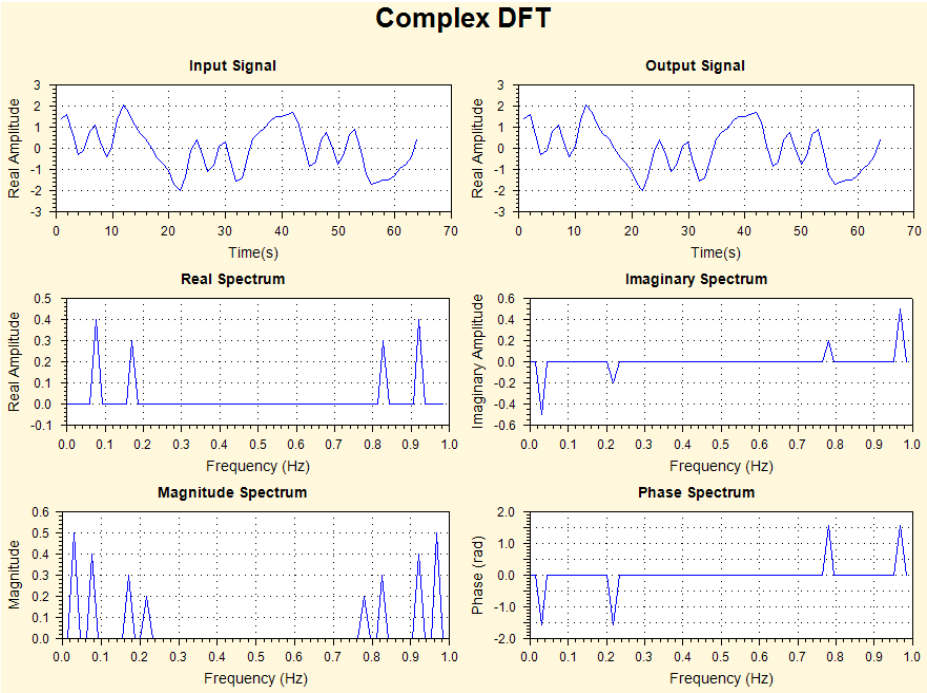
```
' integer logarithm base 2
Private Shared Function ILog2(ByVal x As Integer) As Integer
    Dim p As Integer = 0
    Dim y As Integer = 1
    While (y < x)
        p += 1
        y = 2 * y
    End While
    Return p
End Function

' integer power base 2
Private Shared Function IPow2(ByVal p As Integer) As Integer
    Dim x As Integer = 1
    While (p > 0)
        p -= 1
        x = 2 * x
    End While
    Return x
End Function

' The FFTBitReverseTable function returns a
' table of indexes for FFT in-place sample shuffling
Private Shared Sub FFTBitReverseTable(ByVal size As Integer, ByRef idx() As
Integer)

    ' find # bits involved
    Dim nbit As Integer = ILog2(size)

    ' for each table entry
    For i As Integer = 0 To size - 1
        ' store bit reversed index
        Dim a1 As Integer = i
        Dim a2 As Integer = 0
        For j As Integer = 1 To nbit
            a2 *= 2
            If (a1 And 1) Then a2 = a2 Or 1
            a1 \= 2
        Next
        idx(i) = a2

    Next
End Sub

' The FFTButterfly function performs the key FFT cross-multiply
Private Shared Sub FFTButterfly(ByRef upper As Complex, ByRef lower As Complex,
ByVal w As Complex)
    Dim temp As Complex = lower * w
    lower = upper - temp
    upper = upper + temp
End Sub

' The ComplexFFT function implements a fast complex to
' complex discrete fourier transform
Public Shared Function ComplexFFT(ByRef x As ComplexWaveform) As Spectrum

    Dim size As Integer = IPow2(ILog2(x.Count))
    Dim s As New Spectrum(size, x.Rate)

    ' get bit reverse table
    Dim amap(size) As Integer
    FFTBitReverseTable(size, amap)

    ' shuffle input data into spectrum
    For i As Integer = 0 To size - 1
        s(amap(i)) = x(i + 1)       ' uses bad index capability of x[] to pad
    Next

    ' do multiple butterfy passes over data
```

```
    ' with steps of 1,2,4,..,N
    Dim d As Integer = 1
    While (d < size)
        ' for each start position
        For j As Integer = 0 To d - 1

            Dim w As Complex = Complex.Exp(New Complex(0, -(Math.PI * j) / d))
            ' for each step
            Dim i As Integer = j
            While (i < size)
                FFTButterfly(s(i), s(i + d), w)
                i += 2 * d
            End While
        Next
        d *= 2
    End While

    ' normalise
    For i As Integer = 0 To size - 1
        s(i) /= x.Count
    Next
    Return s
End Function

' The ComplexIFFT function implements a fast complex to
' complex inverse discrete fourier transform
Public Shared Function ComplexIFFT(ByRef s As Spectrum) As ComplexWaveform

    Dim x As New ComplexWaveform(s.Count, s.Rate)

    ' get bit reverse table
    Dim amap(s.Count) As Integer
    FFTBitReverseTable(s.Count, amap)

    ' shuffle input data into waveform
    For i As Integer = 0 To s.Count - 1
        x(i + 1) = s(amap(i))
    Next

    ' do multiple butterfy passes over data
    ' with steps of 1,2,4,..,N
    Dim d As Integer = 1
    While (d < s.Count)

        ' for each start position
        For j As Integer = 0 To d - 1

            Dim w As Complex = Complex.Exp(New Complex(0, Math.PI * j / d))
            ' for each step
            Dim i As Integer = j + 1
            While (i <= s.Count)
                FFTButterfly(x(i), x(i + d), w)
                i += 2 * d
            End While
        Next
        d *= 2
    End While

    Return x
End Function
```

Example Programs

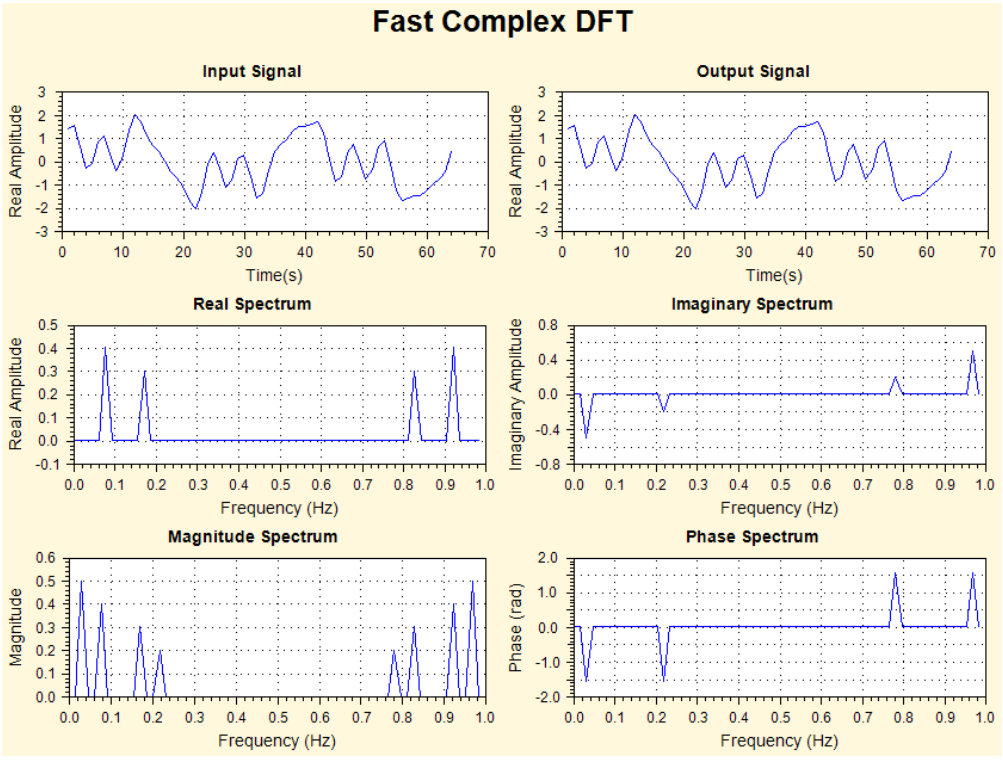
Example 5.1 Complex Discrete Fourier Transform

```
Imports BasicDSP
Imports ZedGraph
Public Class TestCDFT
    Const DFTSIZE As Integer = 64

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

        ' create a test signal
        Dim ip As New ComplexWaveform(DFTSIZE, 1)
        For i As Integer = 1 To DFTSIZE
            Dim f As Double = 2 * Math.PI * (i - 1)
            ip(i) = New Complex(1.0 * Math.Sin(2 * f / DFTSIZE) + _
                0.8 * Math.Cos(5 * f / DFTSIZE) + _
                0.6 * Math.Cos(11 * f / DFTSIZE) + _
                0.4 * Math.Sin(14 * f / DFTSIZE), 0)
        Next

        ' generate spectrum
        Dim s As Spectrum = DFT.ComplexDFT(ip)

        ' generate signal again
        Dim op As ComplexWaveform = DFT.ComplexIDFT(s)

        ' plot as graphs
        Dim gp As New Graph(Me.CreateGraphics, zgc, 3, 2, "Complex DFT")

        gp.PlotComplexWaveform(1, ip, "Input Signal")
        gp.PlotComplexWaveform(2, op, "Output Signal")
        gp.PlotSpectrumReal(3, s, "Real Spectrum")
        gp.PlotSpectrumImag(4, s, "Imaginary Spectrum")
        gp.PlotSpectrumMag(5, s, "Magnitude Spectrum")
        gp.PlotSpectrumArg(6, s, "Phase Spectrum")

    End Sub
End Class
```



Example 5.2 Complex Fast Fourier Transform

```
Imports BasicDSP
Imports ZedGraph
Public Class TestCDFT
    Const DFTSIZE As Integer = 64

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

        ' create a test signal
        Dim ip As New ComplexWaveform(DFTSIZE, 1)
        For i As Integer = 1 To DFTSIZE
            Dim f As Double = 2 * Math.PI * (i - 1)
            ip(i) = New Complex(1.0 * Math.Sin(2 * f / DFTSIZE) + _
                0.8 * Math.Cos(5 * f / DFTSIZE) + _
                0.6 * Math.Cos(11 * f / DFTSIZE) + _
                0.4 * Math.Sin(14 * f / DFTSIZE), 0)
        Next

        ' generate spectrum
        Dim s As Spectrum = DFT.ComplexFFT(ip)

        ' generate signal again
        Dim op As ComplexWaveform = DFT.ComplexIFFT(s)

        ' plot as graphs
        Dim gp As New Graph(Me.CreateGraphics, zgc, 3, 2, "Fast Complex DFT")

        gp.PlotComplexWaveform(1, ip, "Input Signal")
        gp.PlotComplexWaveform(2, op, "Output Signal")
        gp.PlotSpectrumReal(3, s, "Real Spectrum")
        gp.PlotSpectrumImag(4, s, "Imaginary Spectrum")
        gp.PlotSpectrumMag(5, s, "Magnitude Spectrum")
        gp.PlotSpectrumArg(6, s, "Phase Spectrum")

    End Sub
End Class
```



Exercises

5.1 The inverse DFT provides a simpler method to synthesize a square wave: set up the spectrum of a square wave and call `ComplexIDFT()`. Set up a spectrum as follows:

```
Spectrum(1000,0.05);      // 0..19,980Hz in 1000 steps
```

Then put in the odd harmonics of 100Hz with appropriate amplitudes (amplitude of *n*th harmonic is just 1/n). That is, put in amplitudes of 1.0 at spectrum position 5, 0.33 at position 15, 0.2 at position 25, etc. Don't forget to put in the mirror images at positions 995, 985, 975, etc. Plot your spectrum and the result of the inverse DFT.

How would you change your solution to use an FFT? Why might you want to?

5.2 Modify Examples 5.1 and 5.2 to explore the differences between an exact DFT of a 40 point test waveform and the FFT of the same waveform suffixed with 24 zeros. Plot a graph showing the magnitude spectrum and the phase spectrum of the same signal analysed in these two ways.

What differences are there between an FFT of a 64 point waveform and an FFT of the same waveform appended with 64 zeros?