Lab 5 – Fast Fourier Transform

5.1 Introduction

The last laboratory covers the Discrete Fourier Transform (DFT). This laboratory will continue the discussion of the DFT and will introduce the FFT.

5.2 Continuation of DFT Analysis

This section continues the analysis of the DFT started in the previous week's laboratory.

$$\text{(DFT)} \quad X_N[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi kn/N}$$
$$\text{(inverse DFT)} \quad x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X_N[k]\, e^{j2\pi kn/N} \tag{5.1}$$

5.2.1 Shifting the Frequency Range

In this section, we will illustrate a representation for the DFT of (5.1) that is a bit more intuitive. First create a Hamming window x of length $N = 20$, using the Matlab command x = hamming(20) . Then use your Matlab function DFTsum to compute the 20 point DFT of x. Plot the magnitude of the DFT, $|X_{20}[k]|$, versus the index k. Remember that the DFT index k starts at 0 not 1!

> INLAB REPORT: Hand in the plot of the $|X_{20}[k]|$. Circle the regions of the plot corresponding to low frequency components.

Our plot of the DFT has two disadvantages. First, the DFT values are plotted against k rather then the frequency ω. Second, the arrangement of frequency samples in the DFT goes from 0 to 2π rather than from $-\pi$ to π, as is conventional with the DTFT. In order to plot the DFT values similar to a conventional DTFT plot, we must compute the vector of frequencies in radians per sample, and then "rotate" the plot to produce the more familiar range, $-\pi$ to π.

Let's first consider the vector w of frequencies in radians per sample. Each element of w should be the frequency of the corresponding DFT sample $X(k)$, which can be computed by

$$\omega = 2\pi k/N \quad k \in [0, \ldots, N-1] \tag{5.2}$$

However, the frequencies should also lie in the range from $-\pi$ to π. Therefore, if $\omega \geq \pi$, then it should be set to $\omega - 2\pi$. An easy way of making this change in Matlab is (w>=pi) = w(w>=pi)-2*pi .

The resulting vectors X and w are correct, but out of order. To reorder them, we must swap the first and second halves of the vectors. Fortunately, Matlab provides a function specifically for this purpose, called **fftshift**.

Write a Matlab function to compute samples of the DTFT and their corresponding frequencies in the range $-\pi$ to π. Use the syntax

[X,w] = DTFTsamples(x)

where x is an N point vector, X is the length N vector of DTFT samples, and w is the length N vector of corresponding radial frequencies. Your function DTFT samples should call your function DFTsum and use the Matlab function fftshift.

Use your function DTFT samples to compute DTFT samples of the Hamming window of length $N = 20$. Plot the magnitude of these DTFT samples versus frequency in rad/sample.

INLAB REPORT: Hand in the code for your function DTFTsamples. Also hand in the plot of the magnitude of the DTFT samples.

5.2.2 Zero Padding

The spacing between samples of the DTFT is determined by the number of points in the DFT. This can lead to surprising results when the number of samples is too small. In order to illustrate this effect, consider the finite-duration signal

$$x[n] = \begin{cases} \sin(0.1\pi n) & 0 \le n \le 49 \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

In the following, you will compute the DTFT samples of $x(n)$ using both $N = 50$ and $N = 200$ point FT's. Notice that when $N = 200$, most of the samples of $x[n]$ will be zeros because $x[n] = 0$ for $n \ge 50$.This technique is known as “zero padding”, and may be used to produce a finer sampling of the DTFT.

For $N = 50$ and $N = 200$, do the following:

1. Compute the vector x containing the values $x[0], \dots, x[N-1]$.
2. Compute the samples of $X[k]$ using your function DTFTsamples.
3. Plot the magnitude of the DTFT samples versus frequency in rad/sample.

INLAB REPORT: Submit your two plots of the DTFT samples for $N = 50$ and $N = 200$. Which plot looks more like the true DTFT? Explain why the plots look so different.

5.3 The Fast Fourier Transform Algorithm

We have seen in the preceding sections that the DFT is a very computationally intensive operation. In 1965, Cooley and Tukey ([1]) published an algorithm that could be used to compute the DFT much more efficiently. Various forms of their algorithm, which came to be known as the fast Fourier transform (FFT), had actually been developed much earlier by other mathematicians (even dating back to Gauss). It was their paper, however, which stimulated a revolution in the field of signal processing.

It is important to keep in mind at the outset that the FFT is **not** a new transform. It is simply a very efficient way to compute an existing transform, namely the DFT. As we saw, a straightforward implementation of the DFT can be computationally expensive because the number of multiplies grows as the square of the input length (i.e. N^2 for an N point DFT). The FFT reduces this computation using two simple but important concepts. The first concept, known as divide-and-conquer, splits the problem into two smaller problems. The second concept, known as

recursion, applies this divide-and-conquer method repeatedly until the problem is solved.

Consider the defining equation for the DFT and assume that N is even, so that $N/2$ is an integer:

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N} \tag{5.4}$$

Here we have dropped the subscript of N in the notation for $X[k]$. We will also use the notation

$$X[k] = \mathrm{DFT}_N\{x[n]\} \tag{5.5}$$

to denote the N point DFT of the signal $x[n]$.

Suppose we break the sum in (5.4) into two sums, one containing all the terms for which n is even, and one containing all the terms for which n is odd:

$$X[k] = \sum_{\substack{n=0 \\ n \text{ even}}}^{N-1} x[n]e^{-j2\pi kn/N} + \sum_{\substack{n=0 \\ n \text{ odd}}}^{N-1} x[n]e^{-j2\pi kn/N} \tag{5.6}$$

We can eliminate the conditions "n even" and "n odd" in (5.6) by making a change of variable in each sum. In the first sum, we replace n by $2m$. Then as we sum m from 0 to $N/2-1, n = 2m$ will take on all even integer values between 0 and $N-2$. Similarly, in the second sum, we replace n by $2m+1$. Then as we sum m from 0 to $N/2-1, n = 2m+1$ will take on all odd integer values between 0 and $N-1$. Thus, we can write

$$X[k] = \sum_{m=0}^{N/2-1} x[2m]\, e^{-j2\pi k2m/N} + \sum_{m=0}^{N/2-1} x[2m+1]e^{-j2\pi k(2m+1)/N} \tag{5.7}$$

Next we rearrange the exponent of the complex exponential in the first sum, and split and rearrange the exponent in the second sum to yield

$$X[k] = \sum_{m=0}^{N/2-1} x[2m]\, e^{-j2\pi km/(N/2)} + e^{-j2\pi k/N} \sum_{m=0}^{N/2-1} x[2m+1]e^{-j2\pi km/(N/2)} \tag{5.8}$$

Now compare the first sum in (5.8) with the definition for the DFT given by (5.4). They have exactly the same form if we replace N everywhere in (5.4) by $N/2$. Thus the first sum in (5.8) is an $N/2$ point DFT of the even-numbered data points in the original sequence. Similarly, the second sum in (5.8) is an $N/2$ point DFT of the odd-numbered data points in the original sequence. To obtain the N point DFT of the complete sequence, we multiply the DFT of the odd-numbered data points by the complex exponential factor $e^{-j2\pi k/N}$, and then simply sum the two $N/2$ point DFTs.

To summarize, we will rewrite (5.8) according to this interpretation. First, we define two new $N/2$ point data sequences $x_0[n]$ and $x_1[n]$, which contain the even and odd-numbered data points from the original N point sequence, respectively:

$$x_0[n] = x[2n]$$
$$x_1[n] = x[2n+1]$$

(5.9)

where $n = 0, \ldots, N/2 - 1$. This separation of even and odd points is called **decimation in time**. The N point DFT of $x[n]$ is then given by

$$X[k] = X_0[k] + e^{-j2\pi k/N} X_1[k] \text{ for } k = 0, \ldots, N-1$$

(5.10)

where $X_0[k]$ and $X_1[k]$ are the $N/2$ point DFT's of the even and odd points.

$$X_0[k] = \text{DFT}_{N/2}\{x_0[n]\}$$

$$X_1(k) = \text{DFT}_{N/2}\{x_1[n]\}$$

(5.11)

While (5.10) requires less computation than the original *N* point DFT, it can still be further simplified. First, note that each $N/2$ point DFT is periodic with period $N/2$. This means that we need to only compute $X_0[k]$ and $X_1[k]$ for $N/2$ values of k rather than the N values shown in (5.10). Furthermore, the complex exponential factor $e^{-j2\pi k/N}$ has the property that

$$-e^{-j2\pi\frac{k}{N}} = e^{-j2\pi\frac{k+N/2}{N}}$$

(5.12)

These two facts may be combined to yield a simpler expression for the N point DFT:

$$\left.\begin{aligned} X[k] &= X_0[k] + W_N^k X_1[k] \\ X[k+N/2] &= X_0[k] - W_N^k X_1[k] \end{aligned}\right\} \text{for } k = 0, \ldots, N/2-1$$

(5.13)

where the complex constants defined by $W_N^k = e^{-j2\pi k/N}$ are commonly known as the **twiddle factors**.

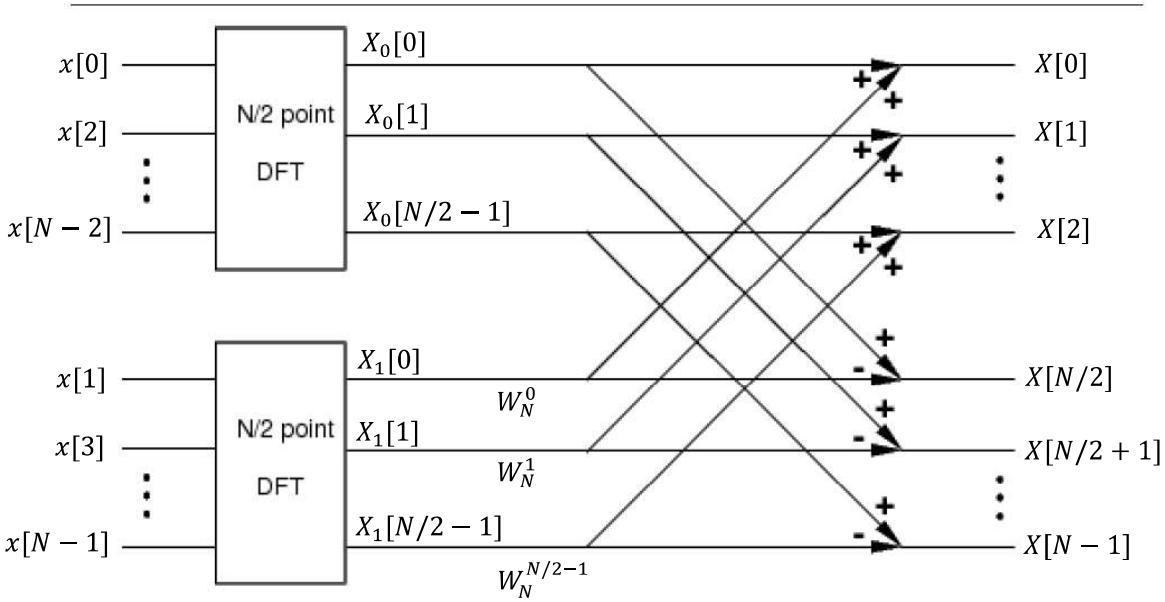

Figure 5.1: Divide and conquer DFT of equation (5.13). The *N*-point DFT is computed using the two $N/2$-point DFT's $X_0^{(N/2)}[k]$ and $X_1^{(N/2)}[k]$.

Figure 5.1 shows a graphical interpretation of (5.13) which we will refer to as the "divide-and-conquer DFT". We start on the left side with the data separated into even and odd subsets. We perform an $N/2$ point DFT on each subset, and then multiply the output of the odd DFT by the required twiddle factors. The first half of the output is computed by adding the two branches, while the second half is formed by subtraction. This type of flow diagram is conventionally used to describe a fast Fourier transform algorithm.

5.3.1 Implementation of Divide-and-Conquer DFT

In this section, you will implement the DFT transformation using (5.13) and the illustration in Figure 5.1. Write a Matlab function with the syntax

X = dcDFT(x)

where x is a vector of even length N, and X is its DFT. Your function dcDFT should do the following:

1. Separate the samples of x into even and odd points.

HINT: The Matlab command x0 = x(1:2:N) can be used to obtain the "even" points.

2. Use your function DFTsum to compute the two $N/2$ point DFT's.
3. Multiply by the twiddle factors $W_N^k = e^{-j2\pi k/N}$.
4. Combine the two DFT's to form X.

Test your function dcDFT by using it to compute the DFT's of the following signals.

1. $x[n] = \delta[n]$ for $N = 10$.
2. $x[n] = 1$ for $N = 10$.
3. $x[n] = e^{j2\pi n/N}$ for $N = 10$.

INLAB REPORT

1. Submit the code for your function dcDFT.
2. Determine the number of multiplies that are required in this approach to computing an N point DFT. (Consider a multiply to be one multiplication of real or complex numbers.)

HINT: Refer to the diagram of Figure 5.1, and remember to consider the $N/2$ point DFTs.