6 The DFT and FFT

6.1 INTRODUCTION

This chapter will look at the discrete Fourier transform (DFT), and at algorithms for computing it efficiently, particularly the fast Fourier transform (FFT) algorithm. The DFT is a computable transform for sequences of finite length, which can be used to represent the spectral characteristics of a signal. The DFT plays a very important role in digital signal processing (DSP). Thus algorithms for its efficient computation are also important and are treated in this chapter. This study leads naturally to a number of issues related to the computational complexity of signal processing algorithms.

Let $x[n]$ be an N-point complex-valued sequence that is nonzero for values of n between 0 and $N-1$. The DFT of that sequence, which will be denoted $X[k]$, is the N-point sequence that is defined by the summation

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk}, \qquad \text{for } k = 0, 1, \ldots, N-1, \tag{6.1}$$

where

$$W_N = e^{-j(2\pi/N)}.$$

The inverse DFT can be evaluated using a similar formula:

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] W_N^{-nk}, \qquad \text{for } n = 0, 1, \ldots, N-1. \tag{6.2}$$

There are at least three interpretations for the DFT, each of which can be helpful in understanding its uses and properties. One point of view regards the DFT as a com-

putable, invertible, linear transformation between two vector spaces—the signal space in which signals are represented as superpositions of delayed unit impulse sequences, and the transform space in which signals are represented as superpositions of complex sinusoids. This interpretation motivates the use of the DFT for waveform coding. A second interpretation regards the DFT as samples of the discrete-time Fourier transform or of the z-transform of the sequence. Comparisons of equation (6.1) with the summations that define the discrete-time Fourier transform (DTFT) and z-transform show that

$$X[k] = X(e^{j\omega})|_{\omega=(2\pi k/N)} = X(z)|_{z=e^{j(2\pi k/N)}}. \tag{6.3}$$

The DFT may be viewed as the z-transform evaluated at the N uniformly spaced points on the unit circle shown in Fig. 6.1. This interpretation motivates the use of the DFT for spectrum analysis. The third interpretation of the DFT regards it as an exact Fourier series representation for the periodic extension of $x[n]$ with period N. The periodic extension of $x[n]$ is formed by repeating $x[n]$ every N samples in both the positive and negative directions. This third interpretation is the most useful for understanding many of the properties of $X[k]$. These are briefly summarized in the next section.

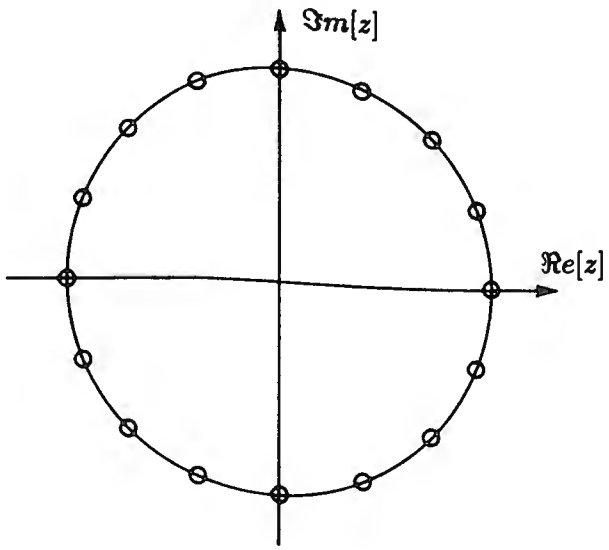

Figure 6.1. The z-plane locations of the samples of a 16-point DFT.

EXERCISE 6.1.1. **Some Simple DFT Properties**

The programs `x dft` and `x idft` are unsophisticated programs for evaluating the forward and inverse discrete Fourier transforms of an N-point sequence. They evaluate the DFT and IDFT using the formulas given in equations (6.1) and (6.2) for any positive integer value of N. In this exercise, you examine some simple properties

of the DFT, first experimentally by computing specific DFTs and then analytically. In working the various parts of this exercise, you will be asked to consider arbitrary sequences that satisfy certain conditions. You may wish to repeat the experiments for several different sequences that satisfy the conditions to obtain a better illustration of the particular property.

(a) Consider a 16-point *circularly antisymmetric* sequence, $x_a[n]$. This is a sequence that satisfies the relation

$$x_a[n] = -x_a[15 - n], \qquad n = 0, 1, \ldots, 15.$$

Use the function **x fdesign** to design a filter of this type. Specifically, specify a *highpass* filter with a cutoff frequency of 0.3 radian. Using **x view**, display the signal and provide a sketch.

Next consider a 16-point sequence, $x_s[n]$, that satisfies the relation

$$x_s[n] = x_s[15 - n], \qquad n = 0, 1, \ldots, 15.$$

Such a sequence is said to be *circularly symmetric*. Use **x mag** to form the sequence $x_s[n] = |x_a[n]|$. Again use **x view** to display the signal. Provide a sketch.

The DFTs of circularly symmetric and antisymmetric signals exhibit a characteristic symmetry. Use **x dft** to compute $X_a[k]$ and $X_s[k]$. Display the real and imaginary parts of these transforms using **x view2**. Clearly describe the characteristics of the DFTs for the circularly symmetric and antisymmetric cases. Feel free to try other test examples before drawing your conclusions.

(b) Compute the *sum* of $x_a[n]$ and $x_s[n]$ that you generated in part (a) using **x add**. Display and sketch the real and imaginary parts of its DFT using **x dft** and **x view**. How are these parts related to those of the two earlier DFTs?

(c) Consider a 16-point "periodic" sequence, $x_p[n]$, that satisfies the relation

$$x_p[n] = x_p[n + 4] = x_p[n + 8] = x_p[n + 12], \qquad n = 0, 1, 2, 3.$$

Use **x siggen** to design a 4-point ramp function with a slope of one and starting point zero. Next use **x lshift** and **x add** in succession to produce $x_p[n]$. Use **x view** to display $x_p[n]$ and verify that it is correct. Compute the DFT and display its real and imaginary parts. Describe the structure imposed on the DFT by the periodic nature of $x_p[n]$. Provide sketches of the DFTs.

(d) Consider a 16-point sequence, $x[n]$, that is zero for its odd-indexed samples, i.e., $x[n] = 0$ for $n = 1, 3, \ldots, 15$. Design an 8-point random signal using **x rgen**. Then use **x upsample** with $M = 2$. This results in a 15-point sequence. Use **x zeropad** with ending value 15 to append the final zero. This will produce $x[n]$. Display $x[n]$ with **x view** to verify that it is correct. Compute and sketch the DFT of $x[n]$ and describe the structure you observe.

(e) Generate the 16-point *sinusoid*

$$x[n] = \cos\left(\frac{5\pi}{8}n\right), \qquad n = 0, 1, \ldots, 15$$

using `x siggen`. Compute the DFT and display its real and imaginary parts. Describe the special nature of $X[k]$. Include a sketch of $X[k]$ with its amplitude scale carefully labeled.

(f) Each of these examples illustrates a property of the DFT. In each case, state the property and show analytically, using the definition of the DFT, that it is true.

— ***EXERCISE 6.1.2.*** **Direct DFT Evaluation**

The DFT definition in equation (6.1) suggests that the total number of multiplications and additions should be proportional to N^2, since each sample of $X[k]$ requires about N complex multiplications and additions and N samples of $X(k)$ need to be computed. This quadratic relationship between the DFT length and the amount of computation makes the direct evaluation of large DFTs computationally unattractive. To illustrate this quadratic dependency use the `x dft` function to evaluate the discrete Fourier transforms of several arbitrary sequences of lengths 32, 64, 128, and 256. The function `x rgen` may be used to create the sequences. If your machine takes longer than five or ten minutes to evaluate the longer DFTs, you may wish to terminate the program prematurely. If, on the other hand, your machine is very fast and evaluates all of the DTFs in only a few seconds, increase the transform length appropriately to obtain useful measurements.

Record the computation times required for each case. Observe that as the DFT size N is increased linearly, the computation time increases at a disproportionate rate. The data you record may not show an exact quadratic relationship because `x dft` must also perform I/O operations that do not have a quadratic time dependence with respect to sequence size.

— ***EXERCISE 6.1.3.*** **Zero Padding**

The DFT of an N-point complex sequence is an N-point complex sequence in general. There are occasions, however, when the length of the DFT should be longer than the length of the sequence. This will be true, for example, when the DFTs of two finite length sequences of different lengths need to be added together. The length of the DFT of the shorter sequence needs to be increased before adding. Another example would be in the display of a high-resolution spectrum of a short sequence. There a large number of spectral or DFT samples is required. In this exercise, we consider how to expand the length of the DFT without distorting the general properties of the transform.

(a) Design a 16-point lowpass filter, $h[n]$, with a cutoff frequency of $\pi/2$ radians using the function `x fdesign`. Evaluate and sketch its magnitude response using `x dtft`. Next evaluate and sketch the magnitude of the 16-point DFT of $h[n]$ using

x dft and x view. Since the DFT is a sampled version of the DTFT, the appearance of $|H(k)|$ is known. However, there is an issue associated with the frequency interval being displayed in the output. The function x dtft displays the frequency response in the interval $-\pi \leq \omega \leq \pi$. Examine the definition of the DFT as well as your plot and determine the frequency interval in which the DFT samples occur. $\Delta\omega = 2\pi/N \cdot \Delta t$

(b) Now embed this sequence $h[n]$ into an $M = 64$-point sequence, $y[n]$, defined as

$$y[n] = \begin{cases} h[n], & n = 0, 1, \ldots, 15 \\ 0, & \text{otherwise.} \end{cases}$$

This operation is called *zero padding*. It can be implemented using the function x zeropad where the ending point should be specified to be 63. Evaluate the 64-point DFT of $y[n]$. Since the direct implementation of the DFT is time-consuming for long sequences, you may wish to use the equivalent, but more efficient realization x fft in place of x dft. Use x view to display and sketch the magnitude of $Y[k]$. How is $Y[k]$ related to $H[k]$?

(c) Repeat part (b) for $M = 256$. In this case the ending point parameter in x zeropad will be 255.

(d) Relate the values of the M-point DFT, $Y[k]$, to the discrete-time Fourier transform and z-transform of $h[n]$ following equation (6.3). Using the idea of zero padding, write a macro to display the DTFT magnitude of $h[n]$ that essentially mimics the function x dtft. You may use the x look function to display your output as a continuous plot. Test the operation of your macro on $h[n]$ and sketch the result.

— *EXERCISE 6.1.4.* **Spatial Aliasing**

It is occasionally desirable to have the number of frequency-domain samples be less than the original number of time samples. For example, we may have a very long sequence and wish to obtain only a small number of evenly spaced samples in the frequency domain. This case is the complement of the case addressed in the previous exercise. Here we are interested in evaluating M samples of the Fourier transform of an N-point sequence when $N > M$.

(a) Using x siggen, generate the exponential

$$x[n] = \alpha^n, \qquad n = 0, 1, \ldots, 127$$

where $\alpha = \frac{39}{40}$. Display and sketch the DFT magnitude of this sequence using x fft and x view.

(b) Now generate a new 64-point sequence $y[n]$ by *spatial aliasing*. This is defined by

$$y[n] = x[n] + x[n + 64], \qquad n = 0, 1, \ldots, 63$$

and can be created using `x lshift` and `x add`. The result of shifting and adding is a sequence beginning at -64 instead of zero. Use `x extract` to extract the samples from 0 to 63. Evaluate the DFT magnitude of $y[n]$ and compare it with the result in (a).

(c) Again consider the *spatial aliasing* concept on the sequence $x[n]$ defined in part (a). Outline the steps for computing the 32-point DFT of this sequence. Compute the 32-point DFT in this manner and sketch the DFT magnitude.

6.2 PROPERTIES OF THE DFT

Many properties of the DFT can be exploited to reduce computation in the design of systems. Some of these are similar to properties of the discrete-time Fourier transform and of the z-transform. Table 6.1 contains an abbreviated list of these properties. In this table the notation $x[[n]]_N$ refers to the N-point periodic extension of $x[n]$ evaluated over the interval $0, 1, \ldots, N-1$. In other words, $x[[n]]_N = x[n]$ in the interval $0, 1, \ldots, N-1$. For n outside of this interval, $x[[n]]_N = x[n + kN]$ where k is the integer (negative or positive) such that $n + kN$ lies within the interval. $0, 1, \ldots, N-1$.

Table 6.1. Some Properties of the DFT[a]

	N-Point Sequence	N-Point DFT
1.	$x[n]$	$X[k]$
2.	$y[n]$	$Y[k]$
3.	$ax[n] + by[n]$	$aX[k] + bY[k]$
4.	$x[[n + n_0]]_N$	$W_N^{-kn_0} X[k]$
5.	$W_N^{nk_0} x[n]$	$X[[k - k_0]]_N$
6.	$\sum_{m=0}^{N-1} x[m] y[[n - m]]_N$	$X[k]Y[k]$
7.	$x[n]y[n]$	$\frac{1}{N} \sum_{\ell=0}^{N-1} X[\ell] Y[[k - \ell]]_N$
8.	$x^*[n]$	$X^*[[-k]]_N$

[a]The N-point signals and N-point DFTs are always assumed to be sequences beginning at 0 and ending at $N-1$.

Since the DFT is an N-point to N-point transformation, only the points in the region $0 \le n \le N-1$ or $0 \le k \le N-1$ are important. Thus a sequence of this form can always be multiplied by $r_N[n]$ where

$$r_N[n] = \begin{cases} 1, & n = 0, 1, \ldots, N-1 \\ 0, & \text{otherwise} \end{cases}$$

without changing the value of the N-point sequence. Rectangular windows are often used to delineate the index range.

The exercises in this section were chosen to help understand a number of these properties.

— *EXERCISE 6.2.1.* **Introduction to Circular Shifting**

The DFT and DTFT have many similar properties. The major difference, however, is that operations for the DFT are assumed to be circular. The DFT is an N-point to N-point transformation. The input sequence is always considered over the range $0 \leq n \leq N-1$, and the transform is always defined for $0 \leq k \leq N-1$. Consequently, operations such as shifting (which inherently translates the sequence to a different interval) must be interpreted differently here.

In this exercise, we investigate the operation of performing circular shifts in the time and DFT domains.

(a) Generate the 16-point sequence

$$x[n] = \begin{cases} 1, & n = 0 \\ 0, & n = 1, 2, \ldots, 15 \end{cases}$$

using the *square wave* option in x **siggen**. Display and sketch $x[n]$ using x **view**. Use the circular shifting function x **cshift** to shift $x[n]$ by 5, 10, 15, 17, and 20. Display and sketch these shifted signals. These circular shifts can be expressed mathematically as $x[[n - n_0]]_N$, where in this case $N = 16$. When n_0 is positive, we have a circular right shift. For n_0 negative, a circular left shift results. Try sketching $x[[n + 6]]_N$ without the aid of the computer. Now check your answer.

(b) Circular shifts can be applied to any N-point sequence regardless of whether it is a time-domain sequence like $x[n]$ or a transformed sequence like $X[k]$. Let $v[n] = x[[n - 18]]_N$. Compute and display the real and imaginary parts of $V[k]$ and $V[[k + 2]]_N$ using x **dft**, x **cshift**, and x **view2**. Sketch the results.

— *EXERCISE 6.2.2.* **Circular Convolution**

The *circular convolution property* (property 6 in Table 6.1) states that the inverse DFT of the product of two DFTs is the circular convolution of the two sequences. The circular convolution depends on the value of the parameter, N.

(a) Generate the 10-point sequence $x[n]$ where

$$x[n] = \begin{cases} 1, & n = 0, 1, \ldots, 7 \\ 0, & n = 8, 9. \end{cases}$$

Form the 10-point circular convolution of $x[n]$ with itself using the function **x cconvolve**. Display and sketch the result using **x view**.

(b) Now compute this circular convolution by computing the 10-point DFT of $x[n]$ using **x dft**, squaring the result using **x multiply**, and computing an inverse DFT using **x idft**. Again display and sketch the output and compare it with your results in part (a). Observe that your output is complex. Explain why the imaginary part is not exactly zero as you would expect.

— ***EXERCISE 6.2.3.*** **Circular and Linear Convolutions**

The N-point sequence length constraint implied by the DFT prohibits operations that expand the sequence length beyond N-points. Linear convolution has the property that the convolution of two N-point sequences is $2N-1$ points long. Circular convolution, by contrast, produces a sequence of length N and is defined as

$$x[n] \circledast_N h[n] = \left(\sum_{m=0}^{N-1} x[[n-m]]_N \, h[m] \right) r_N[n]$$

where $r_N[n]$ is the rectangular window

$$r_N[n] = \begin{cases} 1, & n = 0, 1, 2, \ldots, N-1 \\ 0, & \text{otherwise.} \end{cases}$$

(a) Use **x siggen** to generate a 16-point ramp function beginning at $n = 0$. Now use **x cconvolve** to form the 16-point circular convolution

$$y_1[n] = x[n] \circledast_{16} x[n].$$

Display and sketch the output using **x view**.

(b) Use **x convolve** to perform the linear convolution

$$y_2[n] = x[n] * x[n].$$

Use **x view2** to display and sketch $y_1[n]$ and $y_2[n]$ and observe that they appear to be quite different.

(c) Now take $y_2[n]$ and spatially alias it with itself, by forming the sequence

$$y_3[n] = \begin{cases} y_2[n] + y_2[n+16], & n = 0, 1, 2, \ldots, 15 \\ 0, & \text{otherwise.} \end{cases}$$

Compare the result with $y_1[n]$. *Note:* you may use **x lshift** with a shift of -16 to perform the shifting and the **x extract** function to extract the samples between 0 to 15. Observe that circular convolution can be interpreted in terms of linear convolution.

—*EXERCISE 6.2.4.* **Linear Convolutions via Circular Convolutions**

(a) Create a 64-point square wave, $x[n]$, with pulse length 5 and period 8 starting at $n = 0$ using `x siggen`. Then create the signal $h[n] = u[n] - u[n-16]$ using the block option in `x siggen`. Find the *linear* convolution of these two signals using `x convolve` and use `x view` to display it. Give a rough sketch of the result.

(b) Extend $h[n]$ to 64 points using `x zeropad` and form the 64-point circular convolution of $x[n]$ with $h[n]$ using `x cconvolve`. Provide a rough sketch of the result.

(c) Now use `x zeropad` to extend each of the sequences to 128 points. Evaluate the convolution as before, using N-point circular convolution with $N = 128$. How does your result compare with your answers in (a) and (b) above? What is the smallest value of N that will make the circular convolution equal to the linear convolution for all values of n between 0 and $N - 1$? Check your answer using the computer.

—*EXERCISE 6.2.5.* **Circular Shift**

Property 4 in Table 6.1 is known as the *circular-shift property*.

(a) The sequence $y[n] = x[[n + n_0]]_N r_N[n]$ is a (left) circular shift of $x[n]$ of n_0 samples where $r_N[n]$ was defined in Exercise 6.2.3. Use `x siggen` to generate a 15-point triangular sequence, $x[n]$. Then generate another 15-point sequence, $y[n] = x[[n-3]]_N r_N[n]$, using `x cshift`.

(b) Evaluate the two 15-point DFTs, $X[k]$ and $Y[k]$, using `x dft`. Display and sketch the magnitudes of the two DFTs using `x view2`.

(c) The circular shift can be achieved by circularly convolving $x[n]$ with a particular sequence $h[n]$. Determine the sequence $h[n]$ and sketch the magnitude and phase of its DFT.

(d) Compute and record the inverse 15-point DFT of $Y[k]/X[k]$ using `x divide`, `x idft`, and `x view`, and compare it with your result in (c). Summarize your observations.

—*EXERCISE 6.2.6.* **Circular Modulation**

When a sequence is multiplied by a complex exponential, the DFT of the sequence is circularly shifted. This is called the *circular-modulation property*, and it is the dual of the circular-shift property.

(a) Use `x siggen` to generate the 64-point sequence

$$x[n] = \begin{cases} 1, & 0 \le n \le 15 \\ 0, & 16 \le n \le 64. \end{cases}$$

Plot and sketch the DFT magnitude of this 64-point sequence using the function `x fft` and `x view`.

(b) Evaluate and plot the DFTs of the three related sequences

(i) $(-1)^n x[n]$;

(ii) $(j)^n x[n]$;

(iii) $(-j)^n x[n]$.

(This modulation can be performed using `x cexp` by specifying $\omega_0 = \pi, (\pi/2)$ and $-(\pi/2)$, respectively.) How are these related to $X[k]$?

6.3 UNDERSTANDING THE FFT

Because the DFT is important in signal processing, algorithms for its evaluation have been actively studied for more than thirty years, and a number of highly efficient algorithms for calculating the DFT have been discovered. Collectively these are known as fast Fourier transform (FFT) algorithms. They exploit the periodicity and the structure of the kernels W_N^{nk} for various values of N. FFT algorithms are of two types—Cooley–Tukey algorithms, which have been known to the signal processing community since 1965, and prime factor algorithms, which were discovered by Good in 1959, and more fully developed in the 1970s. The former can be more easily programmed, particularly for cases where the length, N, is variable, but the latter algorithms are more efficient.

Cooley–Tukey algorithms require that the DFT length, N, be composite (i.e., factorable). Therefore, let $N = N_1 N_2$ for integer values of N_1 and N_2 and define

$$\begin{aligned} n &= N_1 n_2 + n_1, \quad n_1 = 0, 1, \ldots, N_1 - 1; \quad n_2 = 0, 1, \ldots, N_2 - 1 \\ k &= k_2 + N_2 k_1, \quad k_1 = 0, 1, \ldots, N_1 - 1; \quad k_2 = 0, 1, \ldots, N_2 - 1. \end{aligned}$$

Then the DFT sum can be written as:

$$\begin{aligned} X[k_2 + N_2 k_1] &= \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x[N_1 n_2 + n_1] W_N^{(N_1 n_2 + n_1)(k_2 + N_2 k_1)} \\ &= \sum_{n_1=0}^{N_1-1} W_N^{N_2 n_1 k_1} \left\{ W_N^{n_1 k_2} \sum_{n_2=0}^{N_2-1} x[N_1 n_2 + n_1] W_N^{N_1 n_2 k_2} \right\} \\ &= \sum_{n_1=0}^{N_1-1} W_{N_1}^{n_1 k_1} \left\{ W_N^{n_1 k_2} \sum_{n_2=0}^{N_2-1} x[N_1 n_2 + n_1] W_{N_2}^{n_2 k_2} \right\}, \end{aligned}$$

for $k_1 = 0, 1, \ldots, N_1 - 1$, $k_2 = 0, 1, \ldots, N_2 - 1$. (Observe in the process of expanding the kernel $W_N^{(N_1 n_2 + n_1)(k_2 + N_2 k_1)}$ into the separate terms $W_N^{N_2 n_1 k_1}$, $W_N^{n_1 k_2}$, $W_N^{N_1 n_2 k_2}$, and $W_N^{N_1 N_2 n_2 k_1}$ that $W_N^{N_1 N_2 n_2 k_1} = e^{j 2\pi n_2 k_1} = 1$. In addition, note that

$W_N^{N_2 n_1 k_1} = W_{N_1}^{n_1 k_1}$ and $W_N^{N_1 n_2 k_2} = W_{N_2}^{n_2 k_2}$.) The inner summation represents an N_2-point DFT for each value of n_1, and the summation outside the braces represents an additional N_1-point DFT for each value of k_2. This decomposition has thus succeeded in breaking down the N-point DFT into several N_1-point and N_2-point DFTs. If C_N represents the number of complex multiplications required by the algorithm to evaluate the N-point DFT, then

$$C_N = N_2 C_{N_1} + N_1 C_{N_2} + N.$$

The last term in this expression is due to the *twiddle factor* multiplications by $W_N^{n_1 k_2}$. When either N_1 or N_2 is composite, the same decomposition can be used to evaluate the smaller DFTs and this procedure can be continued until the lengths of all of the individual DFTs are prime.

In general, efficient FFT algorithms rely on N being a highly composite integer. N-point FFTs of this type, which exploit composite integers of the form $N = R^\nu$, are called *radix-R* algorithms, the most popular being radix-2 and radix-4 algorithms. For such composite integers the number of complex multiplies required to evaluate an N-point DFT is approximately $(N/2)\log_2 N$. Since one complex multiply requires four real multiplies and two real adds, this is equivalent to $2N \log_2 N$ real multiplies. The total number of real adds required is approximately $3N \log_2 N$. If the FFT is based on factorizations by factors other than powers of R, it is called a *mixed-radix* algorithm.

This method of decomposing the input sequence into smaller subsequences can be viewed as *decimation* of the time signal. The smaller DFTs are then performed on these decimated sequences. This property is the motivation for the name *decimation-in-time* or *DIT* FFT algorithm. The flow graph of an 8-point radix-2 decimation-in-time Cooley–Tukey algorithm is shown in Fig. 6.2. The transpose of the DIT flow graph results in an alternate or dual form of this FFT algorithm called *decimation-in-frequency* or *DIF*.

The prime factor algorithms are similar to the extent that a large DFT evaluation is broken down into a number of small ones; the algorithms differ, however, in their details. The prime factor algorithms also require that N be composite; more specifically, they require that $N = N_1 N_2$, where N_1 and N_2 are relatively prime. They exploit the substitutions

$$n = [[\mu_1 n_1 + \mu_2 n_2]]_N$$
$$k = [[\mu_1 k_1 + \mu_2 k_2]]_N.$$

Results from number theory (specifically the Chinese remainder theorem) have established that the numbers μ_1 and μ_2 exist such that values of n_1 in the range between 0 and $N_1 - 1$, and n_2 in the range between 0 and $N_2 - 1$ will map to unique values of n between 0 and $N_1 N_2 - 1$. With this substitution and some simplification of the resulting expression (aided by some additional results from number theory) the DFT summation becomes

$$X[[\mu_1 k_1 + \mu_2 k_2]]_N = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x[[\mu_1 n_1 + \mu_2 n_2]]_N W_{N_1}^{R_1 n_1 k_1} W_{N_2}^{R_2 n_2 k_2}.$$

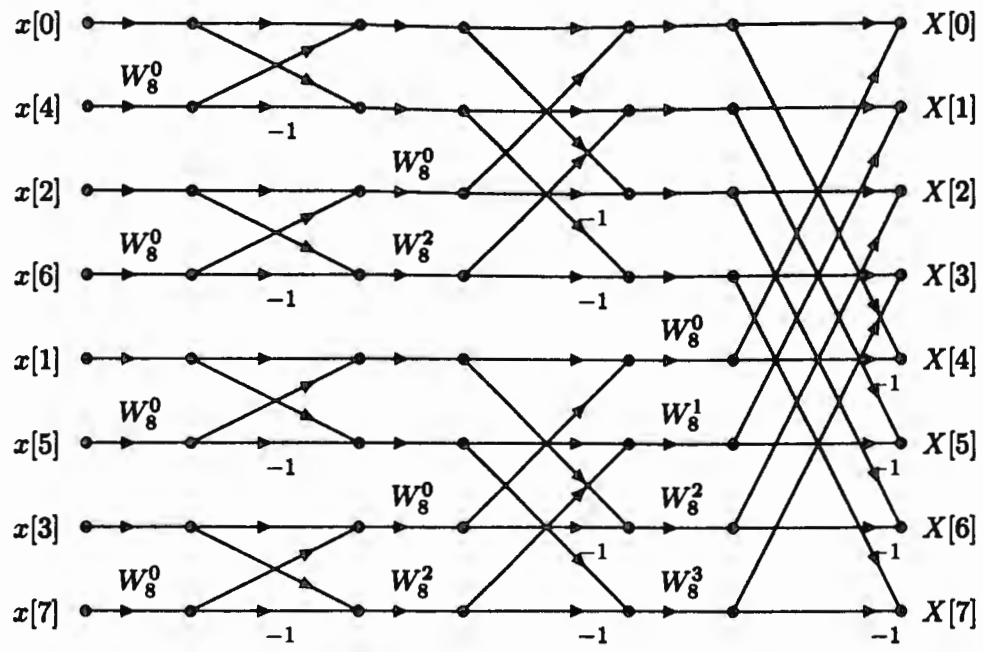

Figure 6.2. A radix-2 decimation-in-time Cooley–Tukey FFT for $N = 8$.

Defining

$$m_1 = [[R_1 n_1]]_{N_1}$$
$$m_2 = [[R_2 n_2]]_{N_2}$$

it is seen that the DFT can be evaluated by a rearrangement of the input sequence followed by N_1 DFTs of length N_2 points taken with respect to the variable m_2. These, in turn, are followed by N_2 N_1-point DFTs taken with respect to m_1. The total computation is

$$C_N = N_2 C_{N_1} + N_1 C_{N_2}.$$

This is less than required by the Cooley–Tukey algorithms because there are no twiddle factor multiplications. The requirement that N_1 and N_2 be relatively prime, however, complicates the programming of a general subroutine that will work for many values of N.

The exercises in this section are intended to help you understand the FFT algorithm and some of its relatives. In many cases this involves writing macros and running them. In this computing environment running macros is inefficient and the computational savings that are made possible by many of the algorithmic tricks may not be reflected in reductions in clock times. To get real savings in time requires explicitly writing new programs and compiling and running them.

—*EXERCISE 6.3.1.* **Radix-2 FFT**

In a radix-2 decimation-in-time FFT, $N = 2^\nu$ and, in the terminology of this section, $N_1 = 2$ and $N_2 = N/2$. This leads to

$$X[k] = G[k] + W_N^k H[k], \qquad k = 0, 1, \ldots, N-1$$

where $G[k]$ is formed by computing the $N/2$-point DFT of the even samples of $x[n]$ and $H[k]$ is formed by computing the $N/2$-point DFT of the odd samples. The two smaller DFTs are extended to N samples by periodic extension before this formula is applied. The intent of this exercise is to verify this formula.

(a) Use the *square wave* option in `x siggen` to produce a 64-point signal consisting of 9 ones followed by 55 zeros and starting at $n = 0$. This signal will be $x[n]$. Compute its 64-point DFT, $X[k]$, using `x fft`. Use `x view` to display and sketch its real and imaginary parts.

(b) Write a macro to calculate $X[k]$ using the formula above. One scheme for doing this would involve the following steps:

(i) Use `x dnsample` and `x lshift` to generate the 32-point sequences $g[n]$ and $h[n]$.

(ii) Use `x fft` to calculate the 32-point DFTs, $G[k]$ and $H[k]$.

(iii) Use `x lshift` and `x add` to periodically extend $G[k]$ and $H[k]$ to 64-point sequences. In other words, let

$$G[k] \Rightarrow G[k] + G[k-32]$$
$$H[k] \Rightarrow H[k] + H[k-32].$$

(iv) Then use `x cexp` to perform the $W_N^k H(k)$ operation prior to adding the sequences.

(c) Use your macro to evaluate the DFT of $x[n]$ from part **(a)**. Plot your result with the result from part (a) using `x view2` and compare them. Are they identical?

—*EXERCISE 6.3.2.* **Inverse FFT**

The evaluation of an inverse DFT is obviously very similar to the evaluation of the DFT itself. This exercise will consider means for using FFT programs to calculate inverse DFTs. There are at least three ways by which this can be done.

Method I: Modify the program itself. In the inverse DFT, W_N is replaced by its complex conjugate and the result is divided by N.

Method II: Use the fact that $x^*[n] = \text{DFT}\{X^*[k]\}/N$ to evaluate the inverse DFT using a forward DFT program. This means that you should conjugate $X[k]$, pass the result through a (forward) DFT subroutine, conjugate the result, and divide it by N.

Method III: Use the fact that $x[[N-n]]_N = \text{DFT}\{X[k]\}/N$. This means that you should pass $X[k]$ through a DFT subroutine, (circularly) reverse the order of the result, and divide it by N.

(a) Write a macro that will evaluate an inverse DFT of an 8-point sequence using the procedure outlined in Method II above. The macro should contain the functions **x conjugate**, **x fft**, and **x gain**.

(b) Write a macro for the inverse DFT of an 8-point sequence based on Method III. Carefully consider the operations implied by $x[N-n]$. It will involve the use of **x reverse**, **x lshift**, and **x cshift**.

(c) Use **x siggen** to create an 8-point ramp sequence with starting point zero. Compute the inverse DFT of the ramp using the macros in parts (a) and (b). Display their real and imaginary parts using **x view2**. Compute the inverse directly using **x ifft** and verify that your results are correct. Sketch the inverse DFT of the ramp.

— ***EXERCISE 6.3.3.*** **Computational Complexity of the FFT**

In this exercise, you will measure the execution times of the FFT algorithm for different length sequences to assess the arithmetic complexity of the algorithm as a function of length. Plot the FFT execution times of arbitrary random sequences with lengths that are a power of 2 on a time scale from zero to about five minutes. Start by creating arbitrary sequences of lengths 16, 64, 256, 1024, and 2048 using **x rgen**. Evaluate the FFT using **x fft** and record the computation times. If the execution times are very short due to the high speed of your computer, use longer length sequences. If, on the other hand, execution times are very long, choose smaller lengths that result in execution times of no more than a few minutes.

Note that for the shorter sequences the I/O overhead of the program will dominate the computation time of the FFT itself. How well do your measurements fit the complexity formula (which is proportional to $N \log_2 N$) based on counting multiplications and additions? Compare the run times with those of the straightforward function **x dft** on those sequences for which **x dft** can run in less than 5 minutes.

— ***EXERCISE 6.3.4.*** **Decimation-in-Frequency (DIF) FFT**

The radix-2 decimation-in-time (DIT) DFT is based on the substitutions

$$n = N_1 n_2 + n_1$$
$$k = k_2 + N_2 k_1$$

with $N_1 = 2$ and $N_2 = N/2$ at the first stage. Decimation-in-frequency or DIF algorithms result by reversing the roles of N_1 and N_2. The flowchart of a DIF algorithm is the transpose of that of a DIT. This leads to the following basic decomposition for the decimation-in-frequency FFT:

$$W_N^n = e^{-j\frac{2\pi n}{N}}$$

$$g[n] = (x[n] + x[n + N/2])$$
$$h[n] = (x[n] - x[n + N/2]) \cdot W_N^n$$

and

$$X[2k] = G[k] \tag{6.4}$$
$$X[2k + 1] = H[k]. \tag{6.5}$$

The DFT of $g[n]$ yields the even samples of $X[k]$, and the DFT of $h[n]$ yields its odd samples. The same decomposition can be used to compute the $N/2$-point DFTs of $g[n]$ and $h[n]$ to achieve further efficiency.

(a) Draw the flowchart of a complete 8-point DIF FFT.

(b) Using Exercise 6.3.1 as a guide, write a macro to calculate $X(k)$ using equations (6.4) and (6.5).

(c) (*Optional*) Using the flowchart in (a) as a guide, write a subroutine in a language that is supported on your computer that will evaluate an 8-point decimation-in-frequency FFT. Your program should accept an input file and produce an output file with the standard headers that were described in Section 1.2.

— ***EXERCISE 6.3.5.*** **Two-for-One FFT**

All of the FFT algorithms that have been presented assume that the input sequence $x[n]$ is *complex*. Often, however, signals of interest are real. When this is the case, there is a further computational gain that can be realized by evaluating the DFTs of *two* real N-point sequences using *one* N-point complex DFT evaluation. Let the two real sequences be denoted by $x[n]$ and $y[n]$ and form the complex sequence

$$z[n] = x[n] + jy[n].$$

(a) Verify analytically that $X[k]$ and $Y[k]$ can be determined from $Z[k]$ using

$$X[k] = (Z[k] + Z^*[[N - k]]_N)/2 \quad \text{even}$$
$$Y[k] = (Z[k] - Z^*[[N - k]]_N)/2j. \quad \text{odd} \quad \text{sequence}$$

(b) Write a macro (called **fft241.bat**) that will evaluate the DFTs of two sequences $x[n]$ and $y[n]$ of length 16 using only one invocation of the routine **x fft**. Computing $X[k]$ and $Y[k]$ involves the use of the **x reverse**, **x lshift**, **x cshift**, **x conjugate**, **x add**, **x subtract**, and **x gain** functions. Verify the operation of your macro. Practically speaking, it is only necessary to produce $X[k]$ and $Y[k]$ for $k = 0, 1, \ldots, N/2$ in order to completely specify both signals. Why is this true? To simulate this feature, modify your macro (by using **x truncate** so that only $N/2 + 1$ output points of $X(k)$ and $Y(k)$ appear in the output.

(c) Develop the corresponding procedure for calculating the two inverse FFTs using one N-point inverse FFT evaluation. Your procedure should generate the two real N-point sequences $x[n]$ and $y[n]$ from $X[k]$ and $Y[k]$, which are specified for $k = 0, 1, \ldots, N/2$. Write a macro to implement this procedure and verify its operation. One approach is to first extend $X[k]$ and $Y[k]$ to their full lengths, then use these to recreate the sequence $Z[k]$, calculate the inverse transform of $Z[k]$, and finally take the real and imaginary parts of the result. The actual functions that you should use are not specified except that there should be only one invocation of `x ifft`.

— ***EXERCISE 6.3.6.*** **FFT for Real Input Sequences**

The two-for-one approach of the previous exercise can be extended so that one $N/2$-point complex FFT program is used to compute the DFT of an N-point real sequence. This can be seen by looking at the flowchart shown in Fig. 6.3. It illustrates the first stage of decimation for an N-point radix-2 decimation-in-time FFT for the special case where $N = 8$. The two $N/2$-point DFTs indicated in that figure have real inputs and can be evaluated using the two-for-one FFT macro **fft241.bat** that was developed in the previous exercise.

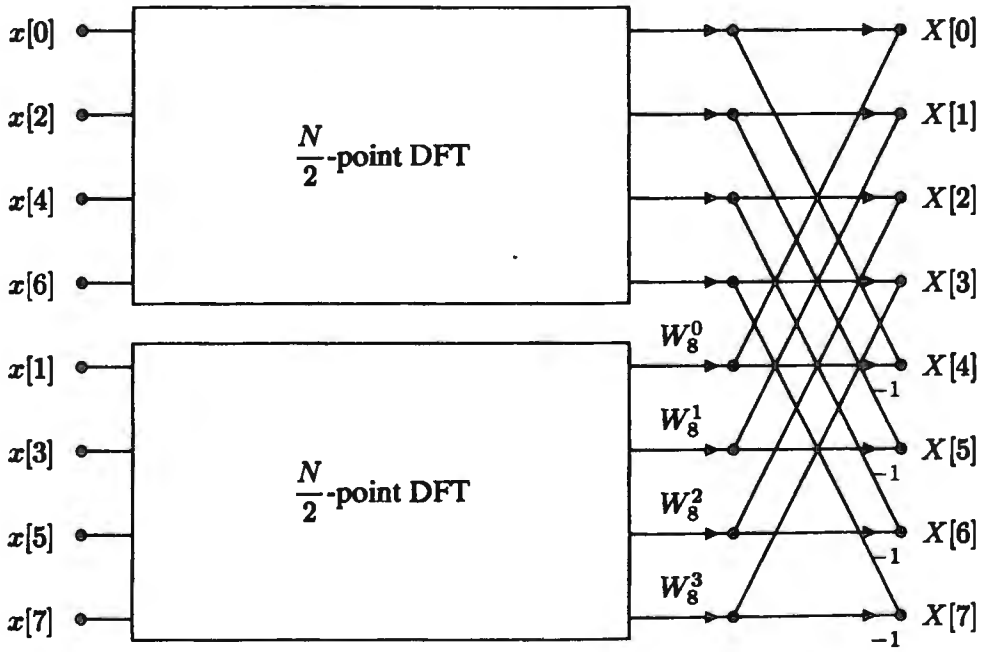

Figure 6.3. The first decimation stage of a radix-2 DIT FFT algorithm.

(a) Use the macro **fft241.bat** as the basis for generating a new macro **fftreal.bat** that will evaluate the FFT of a 32-point real sequence. Estimate the number of

multiplies and adds required using **fftreal.bat** and compare this with using a conventional 32-point radix-2 DIT FFT. What is the improvement in efficiency, if any?

(b) Write a companion macro **ifftreal.bat** that will evaluate the corresponding inverse FFT. Use your result from Exercise 6.3.5(c) as the basis for the macro.

— ***EXERCISE 6.3.7.*** **Goertzel's Algorithm**

Goertzel's algorithm predates the modern rediscoveries of the FFT. Like the DFT definition it is quadratic in complexity, but it can be useful when only limited samples of the DTFT need to be evaluated or when a DFT needs to be calculated for a value of N that is prime. It is based on the observation that the DFT sample at location k

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk} \qquad W_N = e^{-j\frac{2\pi}{N}}$$

can be expressed as one sample of the output of a linear filter

$$X[k] = y_k[N] = \sum_{m=0}^{N-1} x[m] h_k[N-m],$$

when the impulse response of the filter is chosen to be

$$h_k[m] = W_N^{-km} u[m]. \tag{6.6}$$

Because the Nth sample of the output corresponds to only one sample of the DFT, a different network has to be implemented for each value of k.

The filter in equation (6.6) has the transfer function

$$H_k(z) = \frac{1}{1 - W_N^k z^{-1}} = \frac{1}{1 - W_N^{-k} z^{-1}} \tag{6.7}$$

$$= \frac{1 - W_N^{+k} z^{-1}}{1 - 2\cos((2\pi k)/N) z^{-1} + z^{-2}}. \tag{6.8}$$

When the transfer function is written in the form in equation (6.7) the filter can be implemented using a first-order difference equation with a complex coefficient. It appears that one complex multiplication needs to be calculated for each value of m and k, but when the network is written in the form in equation (6.8) it can be implemented using a second-order system that has only one real feedback coefficient. The feedforward term only needs to be implemented when the Nth sample is computed. Therefore, one sample of the DFT requires $N+4$ real multiplies using the Goertzel algorithm, as opposed to $4N$ real multiplies for the direct implementation. Remember that one complex multiply is equivalent to four real multiplies and two real adds.

Create the sequence $x[n]$ given by

$$x[n] = \begin{cases} 1, & n = 0, 1, 2 \\ 0, & n = 3, 4, 5, 6 \end{cases}$$

using the *block* option in **x siggen**.

(a) Consider computing the first three samples of the 7-point DFT of $x[n]$ using Goertzel's algorithm. You will need to generate the IIR filters $H_k(z)$ for $k = 0, 1, 2$ as defined in equation (6.8). These filters can be designed using the *create file* option in **x siggen**. Use the function **x filter** to implement the filtering required by Goertzel's algorithm. Evaluate and record the values of $X(0)$, $X(1)$, and $X(2)$ using this approach. Now evaluate the DFT using **x dft**. Record and compare the values for $k = 0, 1, 2$.

(b) Compare the number of multiplies required by the Goertzel algorithm to that required by the FFT for the case where only one sample of the DFT of a 64-point sequence is desired. Describe when it is beneficial to use the Goertzel algorithm instead of the FFT.

(c) It was stated previously that multiplies and adds required by the numerator of $H_k(z)$ only needed to be performed for the Nth sample in the filtered sequence. This observation leads to additional savings in computation if all samples of the DFT are desired. The denominator polynomials for $k = 0$ and $k = N - 1$, $k = 1$ and $k = N - 2$, $k = 2$ and $k = N - 3$, and so on are identical. Hence, only about half of the recursive filters implied by the denominators need be computed. Consider this point and determine the total number of real multiplies and real adds needed to compute an N-point DFT by exploiting this property of the Goertzel algorithm.

6.4 FAST CONVOLUTION

The term *fast convolution* refers to a number of techniques for evaluating linear convolutions using discrete transforms that are implemented using fast transform algorithms. The DFT and the various FFT algorithms are often used for this purpose, but other transforms will also work. The adjective "fast" is used to contrast the efficiency between the convolution sum and the transform approaches. Evaluation of the convolution sum requires N^2 real multiplies as opposed to roughly "$4N \log_2 N + 4N$" real multiplies for the FFT transform method. For very short transforms, the convolution sum is preferable, but for large values of N, the transform techniques offer a significant reduction in the total number of operations.

— *EXERCISE 6.4.1.* **Evaluating Linear Convolutions Using the DFT**

You have already seen that the product of two DFTs corresponds to circular convolution of their time sequences. When the length of the DFTs used is at least as long as the linear convolution of the two sequences, the circular convolution and the linear convolution are the same. i.e. $N + M - 1$.

!!

(a) Write a macro that will evaluate a fast convolution using **x zeropad**,[1] **x fft**, **x multiply**, and **x ifft**. It should accept as inputs two real sequences both of length $N = 2^{\nu}$ where ν is a positive integer. Evaluate the number of multiplies required in order to perform direct convolution and fast convolution as a function of N. Assume that each invocation of an N-point FFT requires $2N \log_2 N$ real multiplies. At what value of N is it more cost effective to use FFT-based convolution? Test your macro by convolving two block functions of length 128 and starting point zero designed using **x siggen**. To what length must the block sequences be zero padded?

(b) Assume that one sequence is the impulse response $h[n]$ of an LTI system where $h[n]$ is of finite duration with length less than 100. The input to the system is a sampled speech signal that is very long in duration. What difficulties might be encountered in implementing fast convolution for this particular case?

— *EXERCISE 6.4.2.* **Overlap-and-Add Method**

Fast convolution can be difficult to apply when one of the sequences to be convolved is much longer than the other.

(a) Generate a triangular wave sequence $x[n]$ with 15 periods using **x siggen** with period 65, unity amplitude, and starting point zero. Use **x truncate** with ending value 969 to produce a sequence with a total length of 970 samples. Next design a 63-point highpass filter, $h[n]$, using **x fdesign**. Feel free to select an arbitrary value for the cutoff frequency and to use any one of the window options. Filter $x[n]$ by implementing the convolution directly, i.e., by using **x convolve**. How many multiplies and adds are required for this direct form implementation? Use **x view2** to display $x[n]$ and the filtered signal. Provide a sketch of the filtered signal.

(b) Now evaluate the convolution using the macro from Exercise 6.4.1 with $N = 1024$. You should modify the macro so that the DFT of the filter $h[n]$ is precomputed and used as an internal file in the macro. How many real multiplies and real adds are required assuming that $H[k]$ is precomputed? Display the results of the convolution using **x view**. Do they match those of the direct convolution?

(c) Consider breaking the input sequence into five nonoverlapping 194-point segments or blocks, $x_k[n]$, $k = 1, 2, 3, 4, 5$. Note that a block-wise partitioning of the input yields an equivalent expression for the convolution:

$$x[n] = \sum_{k=1}^{5} x_k[n]$$

$$y[n] = x[n] * h[n]$$

$$= \sum_{k=1}^{5} (x_k[n] * h[n]) = \sum_{k=1}^{5} y_k[n].$$

[1] Remember, to obtain a zero padded sequence of length N starting at zero, you should pad with zeros out to index $N - 1$.

Each of these smaller convolutions can be evaluated using fast convolution. Remember that the fast convolution will involve zero padding and should utilize FFTs. The resulting blocks will be longer than the original segments $x_k[n]$. Consequently, they partially overlap. For this reason the method is called *overlap and add.* Write a macro that will evaluate the five short convolutions of the segments and combine them to produce the output sequence $y[n]$. It is important to remember that each of the $x_k[n]$'s begins at a different location. Similarly, each of the $y_k[n]$'s also has a different starting point. The function **x extract** should be used to extract the blocks $x_k[n]$ and reposition them with a starting point of zero. Each block should then be efficiently convolved with $h[n]$ using the procedure developed in part (b). Some modification of the macro is needed to accommodate the different block size. After convolution is performed on each block, **x lshift** should be used to shift each block to its original starting point prior to adding up the blocks. Evaluate $y[n]$ using your macro and display it together with the filtered results from part (b).

(d) Repeat part (c), but modify the partitioning of $x[n]$ so that only 128-point FFTs are used in the fast convolution procedure.

— *EXERCISE 6.4.3.* **Extracting Good Values from Circular Convolutions**

Generate a random 64-point sequence $x[n]$ using **x rgen** and a 16-point lowpass FIR filter, $h[n]$ with a cutoff frequency of $\pi/3$. Use the program **x fdesign** to design the filter with the *Hamming window* option.

(a) Use **x convolve** and **x cconvolve** to evaluate the following two convolutions:

- The linear convolution of $x[n]$ with $h[n]$.
- The circular convolution of $x[n]$ with $h[n]$. Use $N = 64$.

Display the two convolutions on the screen using **x view2** and sketch the results.

(b) The circular and linear convolution results are not equal, but you should be able to observe that some of the samples of the circular convolution are equal to some of the samples of the linear convolution. For which values of the index n are these two convolutions the same? (*Hint:* An easy way to determine this information is to look at the difference of the two convolutions using **x subtract**.)

(c) Generalize your observation in part (b). Assume that $h[n]$ has length N_1, $x[n]$ has length N_2, and N_2-point circular convolution is being performed where $N_1 < N_2$. In terms of N_1 and N_2, which points in $x[n] \circledast_{N_2} h[n]$ are identical to those in $x[n] * h[n]$?

— *EXERCISE 6.4.4.* **Overlap-Save Method**

The *overlap-save method* for block convolution is the dual of the overlap-add method. It also addresses the problem of applying fast convolution when one segment is much longer than the other. It exploits the property observed in Exercise 6.4.3 that

when two sequences of dissimilar lengths are circularly convolved, many of the samples of the circular convolution are equal to samples of the linear one. To develop the method, consider the convolution

$$y[n] = x[n] * h[n]$$

where the input $x[n]$ is long and the impulse response $h[n]$ is a relatively short N_1-point sequence. The approach consists of two steps:

- The input is first divided into length-N overlapping blocks. For example, the first block $x_1[n]$ might begin at $n = -25$ and end at $n = 75$, the second block $x_2[n]$ might begin at 50 and end at 150, the third $x_3[n]$ at 125 and 225, and so on.
- Within each block the N-point circular convolution

  $$x_k[n] \circledast_N h[n]$$

  is performed where $N_1 < N$. This results in an initial set of corrupted points (due to the circular convolution) at the beginning of each block. These corrupted points are discarded, leaving only the noncorrupted points. These input blocks should be chosen to overlap by the number of samples that are discarded so that all of the samples of the linear convolution are computed. If the block lengths N and the block overlaps are chosen correctly, the resulting uncorrupted points will form contiguous blocks that can be abutted to form the correct output sequence.

Write a macro for implementing the overlap-save convolution method. It should use the **x fft**, **x multiply**, **x ifft**, **x extract**, **x lshift**, and **x add** functions. Use your macro to convolve the sequences $x[n]$ and $h[n]$ from Exercise 6.4.2 using 256-point FFTs. How many FFTs need to be performed?

— *EXERCISE 6.4.5.* **Use of the DFT for Deconvolution**

Deconvolution is the process of recovering one signal that has been convolved with another (often a distortion). Consider the relation

$$y[n] = x[n] * h[n].$$

The deconvolution problem may be stated as follows: given $y[n]$ and $x[n]$, find $h[n]$.

When DFTs are used for deconvolution, the quality of the results may be limited by circular convolution effects if special care is not taken. This exercise addresses this issue. Generate the two 32-point sequences

$$x[n] = (0.98)^n u[n]$$
$$y[n] = (0.98)^n u[n] + 0.5(0.98)^{n-20} u[n-20]$$

for $n = 0, 1, 2, \ldots, 31$ using **x siggen**, **x gain**, **x lshift**, and **x add**.

(a) Compute the 32-point DFTs of $x[n]$ and $y[n]$ using **x fft** and sketch their DFT magnitudes. Perform the deconvolution by evaluating the inverse DFT of

$Y[k]/X[k]$ using x divide and x ifft. Use x view to display the real and imaginary parts of the inverse DFT. Sketch these results.

(b) Repeat part (a) using 64-point DFTs. Here it will be necessary to use x zeropad. Describe the effect of the DFT length on the accuracy of your result.

6.5 APPLICATIONS AND RELATIVES OF THE DFT

— *EXERCISE 6.5.1.* **The Zero-Phase DFT**

The DFT is a transform performed on causal sequences. Since the summation that defines it begins at $n = 0$ and ends at $n = N - 1$, its direct use on zero-phase sequences is precluded. Nonetheless, the DFT can be modified to calculate N samples of the DTFT of a zero-phase (noncausal) sequence by a simple two-step process. To illustrate this point consider an odd-length symmetric sequence $x[n]$ where

$$x[n] = \begin{cases} x[-n], & n = 0, 1, \ldots, (N-1)/2 \\ 0, & \text{otherwise.} \end{cases}$$

1. Shift $x[n]$ so that its center sample is at $n = 0$. This is illustrated in Fig. 6.4*a* for the case $N = 7$. Next circularly extend $x[n]$ to form $\tilde{x}[n]$, as illustrated in Fig. 6.4*b*. The circularly shifted sequence $\tilde{x}[n]$ should again lie in the range $0 \le n \le N - 1$.
2. Take the N-point DFT of $\tilde{x}[n]$.

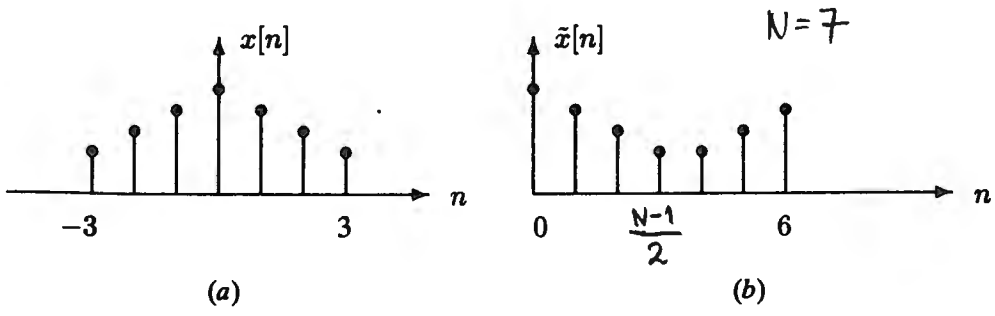

Figure 6.4. An illustration of circular extension. (a) Zero-phase sequence. (b) Causal sequence obtained by circular rotation ($N = 7$).

(a) Write a macro that implements the zero-phase DFT for a 15-point symmetric sequence. The functions x lshift, x add and x extract can be used to form $\tilde{x}[n]$. Display $\tilde{x}[n]$ and verify that it is correct. Next use x dft to compute the DFT. To test your program, generate 15-point lowpass and highpass filters using x fdesign. The choice of cutoff frequency and window type is left to your discretion. The output of x fdesign will be a linear-phase filter. This can be converted to a zero-phase design by shifting with x lshift. Test your

macro by applying it to these sequences. Display the sequences (using x view) and sketch their magnitudes and phases and the real and imaginary parts. Note that plots for zero-phase signals often have phases that are nonzero. When the DTFT of a zero-phase sequence is purely real, the phase can have a value of π or $-\pi$ whenever the Fourier transform is negative. When the DTFT is purely imaginary, the phase can assume values of $\pi/2$ or $-\pi/2$.

(b) Frequency-response plots for a finite length sequence can be obtained by zero padding the sequence to some large length N and taking the FFT. These N points can then be displayed as a continuous signal to produce a representation of the DTFT. This, in essence, is how x dtft works. Consider a similar routine that will take the zero-phase DTFT of a symmetric or antisymmetric sequence following the procedure described in part (a). Write a macro to display the zero-phase DTFT given a 15-point symmetric or antisymmetric sequence. In this case the length of the DFT will be longer than the length of the 15-point sequence. This can be accomplished by zero padding. Use zeropad to pad the sequence out to a total length of 128 and take the 128-point DFT. Should the zeros be added before or after the circular rotation? Use x dtft to display the real and imaginary parts of your input. Next use your macro and x look to do the same. The x look function will display the output as a continuous plot. Provide sketches of all plots.

EXERCISE 6.5.2. **Phase Unwrapping**

The phase response, $\Phi(\omega)$, of a system is a multivalued function; replacing $\Phi(\omega)$ by $\Phi(\omega) + 2\pi m$ (where m is an arbitrary integer) has no effect on the Fourier transform. Certain applications require that this ambiguity in the phase function be resolved to produce a continuous (i.e., smooth) phase function. This process is called *phase unwrapping*. Phase unwrapping adds or subtracts multiples of 2π to $\Phi(\omega)$ at each value of ω to make it maximally smooth.

(a) To examine the concept of phase unwrapping, create a 4-point ramp sequence using x siggen. Extend it to 32 points by zero padding out to index 31 using x zeropad and evaluate its DFT using x fft. Explicitly compute files containing the magnitude and phase responses using the x mag and x phase functions. Display the phase using x view and sketch it. Carefully examine the phase response and identify all of the 2π discontinuities. Record these locations.

(b) Perform phase unwrapping manually by adding or subtracting appropriately shifted block functions with amplitude chosen to be multiples of 2π. This can be done using the *block* option in x siggen to generate the blocks, and the x gain and x lshift functions to scale and move the blocks appropriately. Sketch and display the unwrapped phase.

(c) Recombine the unwrapped phase from part (b) and the magnitude sequence obtained in part (a). This can be done by using x gain with the *gain* parameter 0 1 to create a purely imaginary phase sequence. This can then be added

to the purely real magnitude sequence using **x add** resulting in a complex sequence in polar form. The function **x cartesian** can be used next to convert the sequence into its real and imaginary parts. Use **x polar** to recalculate the magnitude and phase and display the phase using **x view**. Explain what happened to the phase of the signal.

(d) Use the *square wave* option in **x siggen** to generate one period of a square wave with a pulse length of 3 and a starting point at zero. Compute its DTFT using **x dtft** and display the magnitude and phase. You should observe discontinuities of π in the phase. Unlike the previous example, these discontinuities are not due to the fact that the phase is a multivalued function. Explain why they are present.

— ***EXERCISE 6.5.3.*** **The Discrete Cosine Transform (DCT)**

The discrete cosine transform (DCT) is a close relative of the DFT that has found applications in speech and image coding. One form of the DCT is a real transform that is defined by

$$X_c[k] = e[k] \sum_{n=0}^{N-1} x[n] \cos\left(\frac{\pi(2n+1)k}{2N}\right)$$

where

$$e[k] = \begin{cases} \frac{1}{\sqrt{2}}, & k = 0 \\ 1, & k \neq 0. \end{cases}$$

The corresponding inverse DCT is similar.

$$x[n] = \frac{2}{N} \sum_{k=0}^{N-1} e[k] X_c[k] \cos\left(\frac{\pi(2n+1)k}{2N}\right).$$

(a) Express the DCT of $x[n]$ in terms of a DFT of a symmetric sequence that is related to $x[n]$.

(b) Using this relationship, produce a macro to implement the DCT using the functions **x reverse**, **x lshift**, **x cshift**, **x fft**, and **x add**.

(c) Experiment with your program to determine if some of the well-known DFT properties also hold for the DCT. For example, what is the DCT of $ax[n] + bv[n]$? If $x[n]$ is shifted, what happens to its DCT? If two sequences are convolved, do their DCTs multiply?

— ***EXERCISE 6.5.4.*** **The Discrete Hartley Transform (DHT)**

The discrete Hartley transform (DHT) is another real transform and is defined as the difference of the real and imaginary parts of the discrete Fourier transform. Thus

$$X_h[k] = \sum_{n=0}^{N-1} x[n] \left(\cos(\frac{2\pi nk}{N}) + \sin(\frac{2\pi nk}{N})\right).$$

The DHT is its own inverse (apart from a scaling factor). The convolution

$$y[n] = x[n] * h[n]$$

in the time domain becomes the following operation in the transform domain:

$$Y_h[k] = X_h[k]H_{he}[k] + X_h[[-k]]_N H_{ho}[k]$$

where $H_{he}[k]$ and $H_{ho}[k]$ are the even and odd parts of the Hartley transform of $h[n]$ and where $[[\cdot]]_N$ means to evaluate the argument modulo N.

(a) Create a macro **hartley.bat** to compute the DHT and inverse DHT. You should assume that the input sequence is real and of length 32. One of the simplest approaches uses the **x fft**, **x realpart**, **x imagpart**, and **x subtract** functions. Test your macro by taking the forward and inverse DHT of a random sequence.

(b) Describe a procedure for implementing a linear convolution using your DHT program.

(c) Write a macro that implements linear convolution using the DHT. This will use the function **hartley.bat** that you developed in part (a) along with the **x reverse**, **x add**, **x subtract**, **x gain**, **x lshift**, and **x cshift** functions.

Comment. Detailed comparisons of the relative complexity of the DFT and the DHT have shown that when optimally implemented they require exactly the same number of multiplies and nearly the same number of adds. (The DHT actually requires a few more additions.)

— ***EXERCISE 6.5.5.*** **Transform Coding**

Coding is the process of limiting the number of bits required to represent a sequence. Chapter 4 presented some exercises in which quantization of sequence values was used to code a waveform. This exercise considers, as an alternative, the quantization of transform values. Let $x[n]$ denote the signal that is stored in file **sig2** and let $\hat{x}[n]$ denote its encoded representation. In general transform coding is performed in three steps:

(i) The transform of $x[n]$ is computed. There are several transforms that are popular in practice such as the DFT and DCT.

(ii) The transform values are quantized. If the transform is complex, this means that both the real and imaginary parts must be quantized.

(iii) The inverse transform of the quantized transform values is computed.

(a) As a first step, consider quantizing $x[n]$ in the conventional way to form $\hat{v}[n]$. Use **x quantize** to quantize $x[n]$ to 17 levels with minimum and maximum amplitudes of -4.82 and 4.82, respectively. Display $\hat{v}[n]$ and evaluate its SNR using **x snr**.

(b) Now consider quantizing the coefficients of a transform as opposed to the sequence values themselves. Compute the DFT of $x[n]$ using `x fft` and put the real and imaginary parts in separate files using `x realpart` and `x imagpart`. Separately quantize the real and imaginary parts using `x quantize` to 17 levels. Use amplitude ranges of -80 to 80 for the real part and -61 to 61 for the imaginary part. Next recombine the quantized signals into one complex signal. This may be done by multiplying the imaginary part of the signal by "j" and adding the sequences. Use the `x gain` function with *gain* of 0 1 and the `x add` function to perform these operations on the computer. Next, compute $\hat{x}[n]$ using `x ifft` to take the inverse DFT. Display and sketch $\hat{v}[n]$ and $\hat{x}[n]$ using `x view2`. Compute the SNR and compare it to the SNR found in part (a).

A true transform coder typically employs *adaptive bit allocation*, a procedure in which the number of levels used to represent each sample of the transform is determined adaptively based on a prescribed formula. The total number of bits used to code the sequence block is fixed initially. The formula distributes the bits to the samples based on some criterion such as energy values. In this case, assume the very simple (nonadaptive) formula in which all $X[k]$ for $k > 31$ are discarded. Thus to measure the total number of bits used to code this sequence block, add the number of bits used to quantize $X[k]$ for k in the range $0 \leq k \leq 32$. Note that due to Hermitian symmetry, only half of the 128-point DFT is needed to uniquely represent $x[n]$. However, there are two components involved, the real and imaginary parts. Thus 32 samples of the real part and 32 samples of the imaginary part are considered when determining the total number of bits. Find the number of bits required to represent $x[n]$ using the proposed transform coding method. How many bits were required to represent $x[n]$ using the approach outlined in part (a)?

(c) Apply the transform coding method outlined in part (b) using the discrete cosine transform defined in Exercise 6.5.3 and compute the SNR. Note that the DCT produces a real transform with different properties. This calls for modification to the bit allocation formula and to the quantizer amplitude range.

(d) Apply the transform coding method using the discrete Hartley transform defined in Exercise 6.5.4 and compute the SNR. Again, the DHT produces a real transform, but has structure different from that of the DCT and DFT. Make the appropriate modifications in bit allocation and quantizer range.

—*EXERCISE 6.5.6.* **The Chirp z-Transform (CZT)**

The chirp z-transform is an algorithm for evaluating equispaced samples of the z-transform of a finite length sequence with FFT-like efficiency. It accomplishes this by mapping a DFT evaluation into a convolution, which is then implemented using fast convolution techniques (i.e., by using the FFT).

The CZT evaluates the sequence

$$X_k = X(z)|_{z=AW^k}, \qquad k = 0, 1, \ldots, M-1$$

$$A = |A_0|e^{j\phi_0}$$
$$W = |W_0|e^{j\theta_0}.$$

If $|W_0| \neq 1$ the samples lie on a spiral contour with an angular separation of θ_0. The complex number A controls the location of the first sample. By making use of the identity $nk = \frac{1}{2}n^2 + \frac{1}{2}k^2 - \frac{1}{2}(n-k)^2$ this sequence can be written as

$$X_k = W^{-k^2/2} \sum_{n=0}^{N-1} x[n]A^{-n}W^{-n^2/2}W^{(n-k)^2/2}.$$

This can be evaluated in three steps:

(i) Form the sequence $g[n] = x[n]A^{-n}W^{-n^2/2}$.

(ii) Convolve $g[n]$ with $W^{n^2/2}$ using a high-speed technique.

(iii) Multiply the result by $W^{-k^2/2}$.

(a) Sketch the locations of the sample locations in the z-plane that would be computed by the CZT algorithm. Indicate clearly the effect of the parameters $|A_0|$, ϕ_0, $|W_0|$, θ_0, and M.

(b) Earlier in this chapter you explored techniques for evaluating linear convolutions of sequences of finite length using DFTs. The situation here is a little different. Step (ii) of the CZT procedure requires that $g[n]$, a sequence of length N, be convolved with $W^{n^2/2}$, a sequence of infinite length. However, the convolution only needs to be evaluated for M different values of its argument. This means that only a limited number of samples of $W^{n^2/2}$ are needed. Show that the convolution in step (ii) can be performed using only the samples of $W^{n^2/2}$ between limits N_L and N_U and determine the values of these parameters.

(c) Write a macro that will evaluate a 31-point CZT and verify its operation by using it to evaluate a DFT. Note that the term W^{-n^2} is equivalent to $e^{\alpha n^2 + j\theta_0 n^2}$, which can be created using the DSP functions. This can be done by generating a ramp function for n and using `x gain` and `x nlinear` to create $\alpha n^2 + j\theta_0 n^2$. The function `x nlinear` may now be used to perform the exponentiation needed to produce W^{n^2}.

(d) Compare the number of real multiplies and real adds of this program with that of a direct DFT evaluation and Goertzel's algorithm for evaluating the 31-point DFT.

(e) Use `x fdesign` to design a length-32 lowpass filter with cutoff frequency $.45\pi$. Modify the CZT macro to evaluate 100 samples of the discrete-time Fourier transform of this lowpass filter between the frequencies 0.3π and 0.6π.

6.6 REFERENCES

[1] J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Mathematics of Computation*, Vol. 19, No. 90, pp. 297–301, 1965.

[2] I. J. Good, "The Interaction Algorithm and Practical Fourier Analysis," *J. Royal Statistical Society B*, Vol. 20, pp. 361–372, 1960.

[3] M. T. Heideman, D. H. Johnson, and C. S. Burrus, "Gauss and the History of the Fast Fourier Transform," *IEEE ASSP Magazine*, pp. 14–21, Oct. 1984.

CZT Formal Definition:

$$z_k = A \cdot W^{-k}, \quad k = 0, 1, \ldots, M-1$$

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot A^{-n} \cdot W^{nk} = \sum_{n=0}^{N-1} x[n] \cdot A^{-n} \cdot W^{n^2/2} \cdot W^{k^2/2} \cdot W^{-(n-k)^2/2}$$

This can be evaluated by the following three-step process:

1) Form a new sequence $y[n] = x[n] \cdot A^{-n} \cdot W^{n^2/2}$, $n = 0, 1, \ldots, N-1$.

2) Convolve $y[n]$ with the sequence $v[n]$ defined as:

$$v[n] = W^{-n^2/2}$$

to give the sequence $g[k]$:

$$g[k] = \sum_{n=0}^{N-1} y[n] \cdot v[k-n], \quad k = 0, 1, \ldots, M-1.$$

3) multiply $g[k]$ by $W^{k^2/2}$ to obtain $X[k]$:

$$X[k] = g[k] \cdot W^{k^2/2}, \quad k = 0, 1, \ldots, M-1.$$

$$X[k] = W^{\frac{k^2}{2}} \cdot \left\{ \left(x[n] \cdot A^{-n} \cdot W^{\frac{n^2}{2}} \right) * W^{-\frac{n^2}{2}} \right\}, \quad \begin{matrix} k = 0, 1, \ldots, M-1 \\ n = 0, 1, \ldots, N-1 \end{matrix}$$