Math 5600 6/10/14

- The Fast Fourier Transform FFT
- There are many different versions of the FFT
- The concept goes back to Gauss
- But the first modern implementation was described in a 1965 paper by Cooley and Tukey
- Clearest explanation, in my opinion, is in Rulston and Rabinowitz, A First Course in Numerical Analysis, 1965, 1978, Dover 048641454x still in print
- Here is a fresh look at the Fourier Series:

$$F(t) = \frac{a_0}{2} + \sum_{j=1}^{\infty} (a_j \cos jt + b_j \sin jt)$$

where

$$a_j = \frac{1}{2}(c_j + \bar{c}_j) \qquad b_j = \frac{1}{2i}(c_j - \bar{c}_j)$$

and $$c_j = \int_{-\pi}^{\pi} e^{ijt} f(t)\,dt = a_j + ib_j$$

$$i^2 = -1$$

- This is based on Euler's Formulas

$$e^{i\theta} = \cos\theta + i\sin\theta$$

$$e^{-i\theta} = \cos\theta - i\sin\theta$$

$$\cos\theta = \frac{1}{2}\left(e^{i\theta} + e^{-i\theta}\right)$$

$$\sin\theta = \frac{1}{2i}\left(e^{i\theta} - e^{-i\theta}\right)$$

- To be consistent with Ralston/Rabinowitz let's change f to g, and change the periodicity to 1

$$G_j = \int_0^1 g(t)\, e^{2\pi i j t}\, dt \qquad i = \sqrt{-1} \qquad (*)$$

- (Substitution introduces a factor 2π, we'll ignore that.

- Suppose further that we can only evaluate g at N evenly spaced points

$$g_k = g(x_k) \qquad x_k = \frac{k}{N} \qquad k = 0, \ldots, N-1$$

- Replacing the integral in (4) with a sum leads to

$$G_j = \sum_{k=0}^{N-1} g_k e^{\frac{2\pi i j k}{N}} \qquad j = 0, \ldots, N-1$$

- we want to compute N quantities G_j.
- Each sum has N terms
- N^2 multiplications.
- The FFT reduces this to $N \log_2(N)$
- $$G_j = \sum_{k=0}^{N-1} g_k W^{jk} \qquad W = e^{\frac{2\pi i}{N}}$$
- W^j is N periodic. The powers of W can be computed once and stored
- In practice N is a power of 2, but the FFT is easier to explain with different factors

$$N = r_1 r_2 \cdots r_t$$

- We define t-tuples $(j_1, \dots, j_t)$ and $(k_1, \dots, k_t)$ such that

$$j = j_1 + r_1 j_2 + r_1 r_2 j_3 + \dots + r_1 r_2 \dots r_{t-1} j_t$$

$$j_s = 0, 1, \dots, r_s - 1$$

$$k = k_t + r_t k_{t-1} + r_t r_{t-1} k_{t-2} + \dots + r_t \dots r_2 k_1$$

$$k_s = 0, 1, \dots, r_s - 1$$

$$s = 1, \dots, t$$

- The j's and k's are the "digits" of j and k. If all the r_i were equal they'd be ordinary digits with base r. The above is a "mixed radix" representation.

- Example $N = \overset{r_1}{2} \cdot \overset{r_2}{3} \cdot \overset{r_3}{5} = 30$

$$j = j_1 + 2j_2 + 6j_3$$

$$k = k_3 + 5k_2 + 15k_1$$

$j_1, k_1 = 0, 1 \qquad j_2, k_2 = 0, 1, 2, \qquad j_3, k_3 = 0, 1, 2, 3, 4$

- Ex. $j = 23 = 1 + 2 \cdot 2 + 6 \cdot 3 \qquad j_3 = 3 \quad j_2 = 2 \quad j_1 = 1$

$$k = 23 = 15 \cdot 1 + 5 \cdot 1 + 3$$

$$k_1 = 1 \quad k_2 = 1 \quad k_3 = 3$$

- Now suppose that $t=3$ to keep the Algebra simple. Recall $w^N = 1$

$$G_j = \sum_{k=0}^{N-1} g_k w^{jk}$$

$$= \sum_{k=0}^{N-1} g_k w^{j(k_3 + k_2 r_3 + k_1 r_3 r_2)}$$

$$= \sum_{k=0}^{r_3-1} \sum_{k_2=0}^{r_2-1} \sum_{k_1=0}^{r_1-1} g_k w^{k_1 j r_3 r_2} w^{k_2 j r_3} w^{j k_3}$$

$$w^{k_1 j r_3 r_2} = w^{k_1 (j_1 + r_1 j_2 + r_1 r_2 j_3) r_3 r_2}$$

$$= w^{k_1 j_1 r_3 r_2} \underbrace{w^{k_1 r_1 j_2 r_3 r_2}}_{=1} \underbrace{w^{k_1 r_1 r_2 j_3 r_3 r_2}}_{=1}$$

$$= w^{k_1 j_1 r_3 r_2}$$

$$w^{k_2 j r_3} = w^{k_2 (j_1 + r_1 j_2 + r_1 r_2 j_3) r_3}$$

$$= w^{k_2 (j_1 + r_1 j_2) r_3}$$

$$G_j = \sum_{k=0}^{r_3-1} \left(\sum_{k_2=0}^{r_2-1} \left(\sum_{k_1=0}^{r_1-1} g_k w^{k_1 j_1 r_3 r_2} \right) w^{k_2 (j_1 + j_2 r_1) r_3} \right) w^{j k_3}$$

↑ depends only on k_1, only k_2, only k_3

- This sum can be evaluated as follows:
  - compute the innermost sum for all (j_1, k_2, k_3)
  - compute the next sum for all (j_1, j_2, k_3)
  - compute the outermost sum for all (j_1, j_2, j_3)

$$f_0(k_1,k_2,k_3) \leftarrow g_k$$

$$f_1(j_1,k_2,k_3) \leftarrow \sum_{k_1=0}^{r_1-1} f_0(k_1,k_2,k_3)\, w^{k_1 j_1 r_2 r_3}$$

$$f_2(j_1,j_2,k_3) \leftarrow \sum_{k_2=0}^{r_2-1} f_1(j_1,k_2,k_3)\, w^{k_2(j_1+j_2 r_1) r_3}$$

$$G_j = f_3(j_1,j_2,j_3) \leftarrow \sum_{k_3=0}^{r_3-1} f_2(j_1,j_2,k_3)\, w^{j k_3}$$

- Total effort $= N(r_1+r_2+r_3)$ operations.
- In our example $N = 30 = 2 \cdot 3 \cdot 5$

  300 instead of 900
- Suppose $N = r^t$

Example: $N = 2^{14} \approx 16{,}000$ (16 kHz)

effort is $2^{14} \cdot 28$ instead of 2^{28}

save by about 585

- what are the best values of r and t?

- suppose $N = r^t$ is fixed $\quad t = \frac{\log N}{\log r}$

- we want to minimize

$$f(r) = N \cdot t \cdot r = N \frac{\log N \cdot r}{\log r}$$

$$f(2) = \frac{2N \ln(N)}{\ln(2)} \qquad f(3) = \frac{3N \ln(N)}{\ln(3)}$$

$\frac{f(2)}{f(3)} =$ 1.06

$$f'(r) = N \log N \left(\frac{\log r - 1}{\log^2 r} \right) = 0$$

$$r = e$$

- So technically 3 would be more efficient than 2, but 2 has other advantages.

- Major problem with Fourier Series: problem at one point effects all coefficients.

- To overcome that drawback: use wavelets.

ient one; i.e., in effect to normalize the independent variable. Such a convenient interval [Forsythe (1957)] when $w(x) = 1$ is $[-2, 2]$.

It is important that the reader clearly distinguish the two types of errors considered here. On the one hand, the ill condition of G causes the difference between the calculated coefficients given by (6.5-5) and (6.5-10). On the other hand, the difference between the coefficients in (6.5-5) and the true coefficients (6.5-6) is due to the inherent empirical errors in the data.

6.6 THE FOURIER APPROXIMATION

Particularly when the data are from a real-time application, there may be physical knowledge of the function $f(x)$ which indicates that it is periodic. In this case it is advantageous to use the trigonometric (or Fourier) functions instead of polynomials as the least-squares approximating functions. In Sec. 6.6-2 we consider least-squares approximations based on the Fourier functions, but because the summations which arise in such approximations play a role in many applications besides least-squares approximations, we consider first, in Sec. 6.6-1, the evaluation of such sums by the algorithm known as the fast Fourier transform.

6.6-1 The Fast Fourier Transform

Let g_k, $k = 0, 1, \ldots, N-1$ be a set of complex numbers and let

$$G_j = \sum_{k=0}^{N-1} g_k e^{2\pi i jk/N} \qquad j = 0, 1, \ldots, N-1$$
$$= \sum_{k=0}^{N-1} g_k w^{jk} \qquad \text{where } w = e^{2\pi i/N} \tag{6.6-1}$$

Equation (6.6-1) is often called the *discrete Fourier transform* (DFT) of the sequence $\{g_k\}$ by analogy with the (continuous) Fourier transform

$$G(x) = \int_{-\infty}^{\infty} g(t) e^{2\pi i x t}\, dt \tag{6.6-2}$$

Indeed, there is a direct relationship between the discrete and continuous transforms, whose derivation we leave to a problem {21}. In the same way that the continuous Fourier transform can be inverted, so can the discrete transform, to yield

$$g_k = \frac{1}{N} \sum_{j=0}^{N-1} G_j w^{-jk} \qquad k = 0, 1, \ldots, N-1 \tag{6.6-3}$$

Using the orthogonality relationship {22}

$$\sum_{k=0}^{N-1} w^{jk} w^{-rk} = \begin{cases} N & \text{if } j \equiv r \pmod{N} \\ 0 & \text{otherwise} \end{cases} \tag{6.6-4}$$

it is not hard to show that if G_j given by (6.6-1) is substituted into (6.6-3), the right-hand side gives back g_k {22}. The G_j and g_k thus form a *transform pair*, and it is convenient to use the notation

$$G_j \leftrightarrow g_k$$

to denote this. Moreover, since both (6.6-1) and (6.6-3) are periodic with period N, we may consider G_j and g_k to be defined for all j and k with $G_{j+rN} = G_j$ and $g_{k+rN} = g_k$.

Among the properties of the DFT we state the following, leaving the proofs to a problem {22}.

Property 1 *Linearity* If $G_j \leftrightarrow g_k$ and $H_j \leftrightarrow h_k$ and α and β are any complex constants, then $\alpha G_j + \beta H_j \leftrightarrow \alpha g_k + \beta h_k$.

Property 2 *Shifting* If $G_j \leftrightarrow g_k$, then

$$w^{jr} G_j \leftrightarrow g_{k-r} \qquad \text{and} \qquad G_{j-s} \leftrightarrow w^{-ks} g_k$$

Property 3 *Convolution* If $G_j \leftrightarrow g_k$ and $H_j \leftrightarrow h_k$, then

$$\frac{1}{N} \sum_{r=0}^{N-1} G_r H_{j-r} \leftrightarrow g_k h_k \qquad \text{and} \qquad G_j H_j \leftrightarrow \sum_{r=0}^{N-1} g_k h_{r-k}$$

In addition to these properties, there are many other results about the DFT, some of which are considered in the problems {21 to 24}. Our main interest here, however, is the calculation of the DFT, which we now consider.

We begin by noting that, given the g_k's, calculation of the G_j using (6.6-1) as given would require N complex multiplications and additions for each j or N^2 for all the G_j's. Now suppose N can be factored into

$$N = r_1 r_2 \cdots r_t \tag{6.6-5}$$

Corresponding to the indices j and k we define t-tuples $(j_1, \ldots, j_t)$ and $(k_1, \ldots, k_t)$ such that

$$\begin{gathered} j = j_1 + r_1 j_2 + r_1 r_2 j_3 + \cdots + r_1 r_2 \cdots r_{t-1} j_t \\ j_s = 0, 1, \ldots, r_s - 1 \\ s = 1, \ldots, t \\ k = k_t + r_t k_{t-1} + r_t r_{t-1} k_{t-2} + \cdots + r_t \cdots r_2 k_1 \\ k_s = 0, 1, \ldots, r_s - 1 \\ s = 1, \ldots, t \end{gathered} \tag{6.6-6}$$

The integers $j_1, \ldots, j_t$ and $k_1, \ldots, k_t$ are called the *digits* of j and k, respectively. For example, if $N = 30$, then

$$N = 2 \cdot 3 \cdot 5 \qquad j = j_1 + 2j_2 + 6j_3 \qquad k = k_3 + 5k_2 + 15k_1 \quad (6.6\text{-}7)$$

If $j = 23$, then $j_1 = 1, j_2 = 2, j_3 = 3$, and if $k = 8$, then $k_1 = 0, k_2 = 1, k_3 = 3$.

Now we shall develop the *fast Fourier transform* (FFT) algorithm for the case $t = 3$ in order to keep the algebra relatively simple. The generalization for an arbitrary value of t will be fairly obvious {25}. Using (6.6-6), we write

$$w^{jk} = w^{j(k_3 + r_3 k_2 + r_3 r_2 k_1)} \quad (6.6\text{-}8)$$

and substitute into (6.6-1) to obtain

$$G_j = \sum_{k_3=0}^{r_3-1} \sum_{k_2=0}^{r_2-1} \sum_{k_1=0}^{r_1-1} g_k w^{j(r_3 r_2)k_1} w^{j r_3 k_2} w^{j k_3} \quad (6.6\text{-}9)$$

Now using (6.6-6), we have

$$w^{j r_3 r_2} = w^{(j_1 + r_1 j_2 + r_1 r_2 j_3)(r_3 r_2)} = w^{j_1 r_3 r_2} \quad (6.6\text{-}10)$$

since all other exponents have terms in $r_1 r_2 r_3 = N$ and $w^{\alpha N} = 1$, where α is any integer. Similarly

$$w^{j r_3} = w^{(j_1 + r_1 j_2) r_3} \quad (6.6\text{-}11)$$

Substituting (6.6-10) and (6.6-11) into (6.6-9), we obtain

$$G_{j_1, j_2, j_3} = \sum_{k_3=0}^{r_3-1} \left\{ \sum_{k_2=0}^{r_2-1} \left[\sum_{k_1=0}^{r_1-1} g_k w^{j_1 r_3 r_2 k_1} \right] w^{(j_1 + r_1 j_2) r_3 k_2} \right\} w^{(j_1 + r_1 j_2 + r_1 r_2 j_3) k_3} \quad (6.6\text{-}12)$$

Noting that the term in square brackets depends only on k_1 and the term in braces only on k_2, we describe the FFT algorithm as follows:

Input

g_k — N values stored in increasing order of the index k, that is, from $(k_1, k_2, k_3) = (0, 0, 0)$ to $(r_1 - 1, r_2 - 1, r_3 - 1)$

Algorithm

$$f_0(k_1, k_2, k_3) \leftarrow g_k$$

$$f_1(j_1, k_2, k_3) \leftarrow \sum_{k_1=0}^{r_1-1} f_0(k_1, k_2, k_3) w^{j_1 r_3 r_2 k_1} \quad (6.6\text{-}13)$$

$$f_2(j_1, j_2, k_3) \leftarrow \sum_{k_2=0}^{r_2-1} f_1(j_1, k_2, k_3) w^{(j_1 + r_1 j_2) r_3 k_2}$$

$$f_3(j_1, j_2, j_3) \leftarrow \sum_{k_3=0}^{r_3-1} f_2(j_1, j_2, k_3) w^{(j_1 + r_1 j_2 + r_1 r_2 j_3) k_3}$$

Output

$$G_j = f_3(j_1, j_2, j_3)$$

From (6.6-12) it follows that this algorithm does indeed compute the G_j. Moreover, the number of complex multiplications and additions required in (6.6-13) for each value of the argument triple on the left-hand side is

$$r_1 + r_2 + r_3$$

Since there are N possible argument triples, the total number of operations is

$$N(r_1 + r_2 + r_3) \tag{6.6-14}$$

which is never greater than N^2 and when r_1, r_2 and r_3 are substantially greater than 1, is much less than N^2. Moreover, if $t > 3$, the equation corresponding to (6.6-14) is {25}

$$N(r_1 + r_2 + \cdots + r_t) \tag{6.6-15}$$

where the inequality with respect to N^2 is likely to be even more pronounced. Thus, the FFT algorithm is, at least at first glance, more efficient than direct evaluation of the DFT, and for large N the reduction in computation may be quite dramatic. There is, of course, a substantial amount of bookkeeping implied by (6.6-13), which would seem to lessen the overall computational advantage of the FFT algorithm; we shall return to this point below.

One aspect of (6.6-13) which tends to be confusing on initial exposure to the FFT is that the order of G_j's computed is different from the natural order. Consider the following case:

$$N = 12 = 2 \cdot 2 \cdot 3 \qquad j = j_1 + 2j_2 + 4j_3 \qquad k = k_3 + 3k_2 + 6k_1 \tag{6.6-16}$$

In organizing the computation of (6.6-13) we would expect to choose the natural order of k from 0 to 11, as shown in Table 6.1. Equation (6.6-13) makes it clear that if we begin with $g_0, \ldots, g_{11}$ in 12 successive memory locations, then at successive steps of the algorithm we may

Overwrite $f_1(j_1, k_2, k_3)$ on $g_k = f_0(k_1, k_2, k_3)$
Overwrite $f_2(j_1, j_2, k_3)$ on $f_1(j_1, k_2, k_3)$
Overwrite $f_3(j_1, j_2, j_3) = G_j$ on $f_2(j_1, j_2, k_3)$

But Table 6.1 indicates that the order of the G_j in these 12 locations is not the natural order and that to obtain the natural order therefore requires some unscrambling. But we do note here that if we reverse the order of the digits j_1, j_2, and j_3 and use the natural order of the reversed digits, we obtain the natural order of j as shown in Table 6.1. It can be proved that this *digit reversal* always results in the natural order of j.

Table 6.1 Correspondence between digits of k and j for case $N = 12$

k	j_1 k_1	j_2 k_2	j_3 k_3	j	j_3	j_2	j_1	j
0	0	0	0	0	0	0	0	0
1	0	0	1	4	0	0	1	1
2	0	0	2	8	0	1	0	2
3	0	1	0	2	0	1	1	3
4	0	1	1	6	1	0	0	4
5	0	1	2	10	1	0	1	5
6	1	0	0	1	1	1	0	6
7	1	0	1	5	1	1	1	7
8	1	0	2	9	2	0	0	8
9	1	1	0	3	2	0	1	9
10	1	1	1	7	2	1	0	10
11	1	1	2	11	2	1	1	11

Implementation of the FFT algorithm is most convenient and efficient when N is chosen so that

$$N = r^t$$

Then from (6.6-15) it follows that the number of operations required is

$$N(rt) = Nr \log_r N = \frac{r}{\log_2 r} N \log_2 N \tag{6.6-17}$$

For fixed N the coefficient of $N \log_2 N$ in (6.6-17) is minimized when $r = 3$, which implies that N should be chosen as a power of 3. But for various reasons, most notably that the calculation and use of the powers of w is simplified, it is most common to choose $r = 2$. Because of the importance of this case, we now consider it in some detail.

When $N = 2^t$, the digits of j and k are all either 0 or 1. Indeed, $j_t, \ldots, j_1$ and $k_1, \ldots, k_t$ are, respectively, just the bits of the binary representations of j and k. The algorithm (6.6-13), now expressed for a general t, becomes

$$\begin{aligned}
&f_0(k_1, k_2, \ldots, k_t) = g_k \\
&\cdots\cdots\cdots\cdots\cdots\cdots \\
&f_l(j_1, j_2, \ldots, j_l, k_{l+1}, \ldots, k_t) \\
&\qquad = \sum_{k_l=0}^{1} f_{l-1}(j_1, \ldots, j_{l-1}, k_l, \ldots, k_t) w^{(\sum_{s=1}^{l} j_s 2^{s-1}) 2^{t-l} k_l} \\
&\qquad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
&G_j = f_t(j_1, j_2, \ldots, j_t) = \sum_{k_t=0}^{1} f_{t-1}(j_1, \ldots, j_{t-1}, k_t) w^{jk_t}
\end{aligned} \tag{6.6-18}$$

The statement of this algorithm, however, can be much simplified, as follows. Define

$$J_1 = 0$$
$$J_l = j_1 + 2j_2 + \cdots + 2^{l-2}j_{l-1} \qquad l = 2, \ldots, t \tag{6.6-19}$$

and

$$K_l = k_{l+1} + 2k_{l+2} + \cdots + 2^{t-l-1}k_t \qquad l = 0, \ldots, t-1$$
$$K_t = 0 \tag{6.6-20}$$

Then we can write the index j in (6.6-18) as

$$j = J_l + 2^{l-1}j_l + 2^l K_l \qquad l = 1, \ldots, t \tag{6.6-21}$$

It is not hard to show {26} that with J_l and K_l defined as in (6.6-19) and (6.6-20), the index j in (6.6-21) goes through its full range of values from 0 to $2^t - 1$ for each l. The middle equation of (6.6-18) can then be written

$$f_l(J_l + 2^{l-1}j_l + 2^l K_l) = \sum_{k_l=0}^{1} f_{l-1}(J_l + 2^{l-1}k_l + 2^l K_l)w^{J_{l+1}2^{l-1}k_l} \tag{6.6-22}$$

where the ranges of the indices are

$$J_l = 0, 1, \ldots, 2^{l-1} - 1 \qquad K_l = 0, 1, \ldots, 2^{t-l} - 1 \tag{6.6-23}$$

When $l = t$ in (6.6-22), the index of f_t is, using (6.6-19) and (6.6-20),

$$J_t + 2^{t-1}j_t + 2^t K_t = J_t + 2^{t-1}j_t$$
$$= j_1 + 2j_2 + \cdots + 2^{t-2}j_{t-1} + 2^{t-1}j_t$$

which, using (6.6-6), is precisely j. Thus, $f_t(0), f_t(1), \ldots, f_t(2^{t-1})$ correspond to $j = 0, 1, \ldots, 2^{t-1}$, and therefore the G_j's are calculated in their natural order. But this happens because

$$K_0 = k_1 + 2k_2 + \cdots + 2^{t-1}k_t$$

has its bits reversed from those of k, as can be seen from (6.6-6). Thus, in order for the G_j's to be computed in their natural order, the g_k's must be ordered in bit-reversal sequence; i.e., the initial order of the g_k must correspond to the order resulting from taking the bits of the binary expansion of K, namely $k_t, k_{t-1}, \ldots, k_1$, and computing K_0 as above. Example 6.1 will illustrate this explicitly.

Now we split (6.6-22) into two equations, one for $j_l = 0$ and one for $j_l = 1$, and write out both terms in the right-hand side sum to obtain

$$f_l(J_l + 2^l K_l) = f_{l-1}(J_l + 2^l K_l) + f_{l-1}(J_l + 2^l K_l + 2^{l-1})w^{J_l 2^{l-1}}$$
$$f_l(J_l + 2^l K_l + 2^{l-1}) = f_{l-1}(J_l + 2^l K_l) + f_{l-1}(J_l + 2^l K_l + 2^{l-1})w^{(J_l + 2^{l-1})2^{l-1}} \tag{6.6-24}$$

Finally, we note that $w^{(2^{l-1})(2^{l-1})} = w^{2^{l-1}} = -1$, and we define

$$p = J_l + 2^l K_l \qquad q = p + 2^{l-1} \tag{6.6-25}$$

so that p and q both take on half the values from 0 to $2^l - 1$. Then we obtain the complete algorithm for this case as

$$\begin{aligned} &f_0(k) = g_k \\ &\cdots\cdots\cdots \\ &f_l(p) = f_{l-1}(p) + f_{l-1}(q)w^{J_l 2^{l-1}} \\ &f_l(q) = f_{l-1}(p) - f_{l-1}(q)w^{J_l 2^{l-1}} \\ &\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ &G_p = f_l(p) \qquad G_q = f_l(q) \end{aligned} \tag{6.6-26}$$

Thus, at each stage of the algorithm we proceed as follows:

1. Let J_l and K_l run through all possible pairs of values in (6.6-23).
2. For each pair calculate p and q as in (6.6-25).
3. Then calculate $f_l(p)$ and $f_l(q)$ from (6.6-26).

Normally in the computer implementation of the algorithm the quantities $w^{J_l 2^{l-1}}$ which appear in (6.6-26) are precalculated and stored in a table (or a set of coefficients from which they are easily calculated is stored in a table {26}). Note that after the complete calculation implied by (6.6-26), we must perform a digit reversal, as described above, to get the G_j in their natural order. In the binary case we are considering, this *bit* reversal consists only of taking the bits of the binary expansion of j and computing j' as the binary expansion of the reversal of these bits. The correct position of G_j as calculated is then $G_{j'}$. Alternatively, as in Example 6.1 below, we may reorder the g_k so that the G_j are calculated in their correct order.

These remarks and the algorithm itself imply that the bookkeeping in (6.6-26) is quite simple and straightforward. The savings achievable, therefore, by reducing the N^2 operations for the DFT to the $N \log_2 N$ are considerable and for large N can be dramatic. The FFT algorithm in the form (6.6-26) or equivalent forms {27} has become extremely popular and useful.

To conclude this section we note a couple of other features of the FFT:

1. If we are given the G_j instead of the g_k, the algorithms (6.6-13) or (6.6-26) are easily applied, the only difference being the change of sign of the exponent of w from plus to minus.
2. An important case is the one where the g_k are all real and we are interested in the cosine transform

$$G_j = \sum_{k=0}^{N-1} g_k \cos \frac{2\pi jk}{N} \tag{6.6-27}$$

The statement of the FFT algorithm for this case is easily derived from (6.6-13) or (6.6-26) {27}. Similarly we may also consider the sine transform

$$G_j = \sum_{k=1}^{N-1} g_k \sin \frac{2\pi jk}{N} \tag{6.6-28}$$

The example which follows has too small a value of N for the FFT algorithm to show any substantial efficiency over direct calculation of the DFT, but it does illustrate how the algorithm works.

Example 6.1 Given the data

k	0	1	2	3	4	5	6	7
g_k	1	$1+i$	0	$1-i$	0	$1+i$	0	$1-i$

compute G_j, $j = 0, 1, \ldots, 7$.

Since $N = 8 = 2^3$, we use (6.6-26). First we perform a bit reversal and reorder the g_k to get f_0. We do this by taking the bits of k and reversing them as in the table below:

k	k_3	k_2	k_1	$4k_1 + 2k_2 + k_3 = k'$	k	k_3	k_2	k_1	$4k_1 + 2k_2 + k_3 = k'$
0	0	0	0	0	4	1	0	0	1
1	0	0	1	4	5	1	0	1	5
2	0	1	0	2	6	1	1	0	3
3	0	1	1	6	7	1	1	1	7

Thus the order of f_0 is with k' in its natural order.

We begin by calculating the necessary powers of w. We have

$$w = e^{2\pi i/8} = \cos\frac{\pi}{4} + i \sin\frac{\pi}{4} = \frac{\sqrt{2}}{2}(1+i)$$

$$w^2 = i \qquad w^3 = \frac{\sqrt{2}}{2}(-1+i)$$

Using these and (6.6-25) and (6.6-26), we then calculate

	0	1	2	3	4	5	6	7
f_0	1	0	0	0	$1+i$	$1+i$	$1-i$	$1-i$
f_1	1	1	0	0	$2+2i$	0	$2-2i$	0
f_2	1	1	1	1	1	4	$4i$	0
$f_3 = G_j$	5	1	-3	1	-3	1	5	1

The reader who wishes to check the values of G_j using (6.6-1) will find that even in this simple case the FFT algorithm has much to recommend it over brute force.