The fast Fourier transform

A detailed explanation is offered of an algorithm that reduces computer time and allows its user to employ powerful frequency techniques once considered inefficient

E. O. Brigham, R. E. Morrow *LTV Electrosystems, Inc.*

The fast Fourier transform (FFT), a computer algorithm that computes the discrete Fourier transform much faster than other algorithms, is explained. Examples and detailed procedures are provided to assist the reader in learning how to use the algorithm. The savings in computer time can be huge; for example, an $N = 2^{10}$-point transform can be computed with the FFT 100 times faster than with the use of a direct approach.

A computer algorithm, called the fast Fourier transform, has opened new avenues of scientific investigation. Problem-solving techniques once considered impractical are now efficiently implemented by the use of the FFT algorithm. Because of its development, many areas in computing have been completely revolutionized. The FFT procedure for synthesizing and analyzing Fourier series was disclosed in a paper by Cooley and Tukey[1]—a paper that is very subtle and somewhat difficult to understand.

It is the intent here to present a more straightforward development of the FFT algorithm. Before we proceed, however, some basic concepts of the Fourier transform will be examined.

Some basics

For continuous periodic functions of time $f(t)$, a familiar tool for analysis is the Fourier-series representation of the function

$$f(t) = \sum_{n=-\infty}^{\infty} F(n) e^{jn2\pi f_1 t} \tag{1}$$

The period of the function is $T_1 = 1/f_1$ and $F(n)$ is the complex Fourier coefficient, given by

$$F(n) = \frac{1}{T_1} \int_{-T_1/2}^{T_1/2} f(t) e^{-jn2\pi f_1 t}\, dt \tag{2}$$

A similar representation for continuous aperiodic functions of time $x(t)$ is given by the integral

$$x(t) = \int_{-\infty}^{\infty} S(f) e^{j2\pi f t}\, df \tag{3}$$

This expression is frequently written in terms of $\omega = 2\pi f$, as

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} S(\omega) e^{j\omega t}\, d\omega$$

The complex continuous frequency spectrum of the aperiodic function $S(f)$ is expressed by

$$S(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi f t}\, dt \tag{4}$$

The reciprocal relations of (3) and (4) are known as a Fourier-transform pair and $S(f)$ is commonly referred to as the Fourier transform of $x(t)$. If $x(t)$ is a periodic function with period T_1, then the Fourier transform $S(f)$ defined in (4) becomes a sequence of weighted impulse functions at integral multiples of the fundamental frequency f_1. The weights correspond to the coefficients of a Fourier series determined from (2).

For functions of time that are zero for negative time, the Fourier transform is closely related to the Laplace transform, which is defined as

$$S(p) = \int_0^\infty x(t)e^{-pt}\,dt \tag{5}$$

Variable p is a complex quantity: $p = \alpha + j\omega$. Thus if the real part of p is zero, the Fourier transform given by (4) and the Laplace transform of (5) are identical.

The Fourier and Laplace transforms are useful in a number of applications. In particular, the solution of differential equations is simplified because either of the transforms converts a differential equation with time as the independent variable into an algebraic equation with p or f as the independent variable. The transforms provide an easy means for relating the input and output variables of linear time-invariant systems encountered in electrical, mechanical, and optical systems.

The Laplace transform is most useful for analytical studies of systems because the transformation yields an analytical function defined in two-dimensional space; the location of the roots of the function in two-dimensional space determines system behavior. The inverse Laplace transform involves a contour integration, in two-dimensional space, that is not readily adapted to numerical methods. On the other hand, the Fourier transform yields a complex function defined in a one-dimensional frequency space; consequently, the inverse transformation is easily adapted to numerical methods. The Fourier transform is superior to the Laplace transform for the analysis of physical data because numerical methods and a digital computer are generally used for this purpose.

One Fourier-transform property having application in the analysis of linear time-invariant systems is that the transform of the convolution of two functions is equal to the product of the transforms of the individual functions. To be specific, consider two time functions $x_1(t)$ and $x_2(t)$ having Fourier transforms $S_1(f)$ and $S_2(f)$ given by (4); the convolution $x_3(t)$ of $x_1(t)$ and $x_2(t)$ is

$$x_3(t) = \int_{-\infty}^{\infty} x_1(\tau)x_2(t-\tau)\,d\tau = x_1(t) * x_2(t)$$

The Fourier transform of $x_3(t)$ is given by

$$S_3(f) = S_1(f)S_2(f)$$

One should also note that multiplication in the time domain corresponds to convolution in the frequency domain.

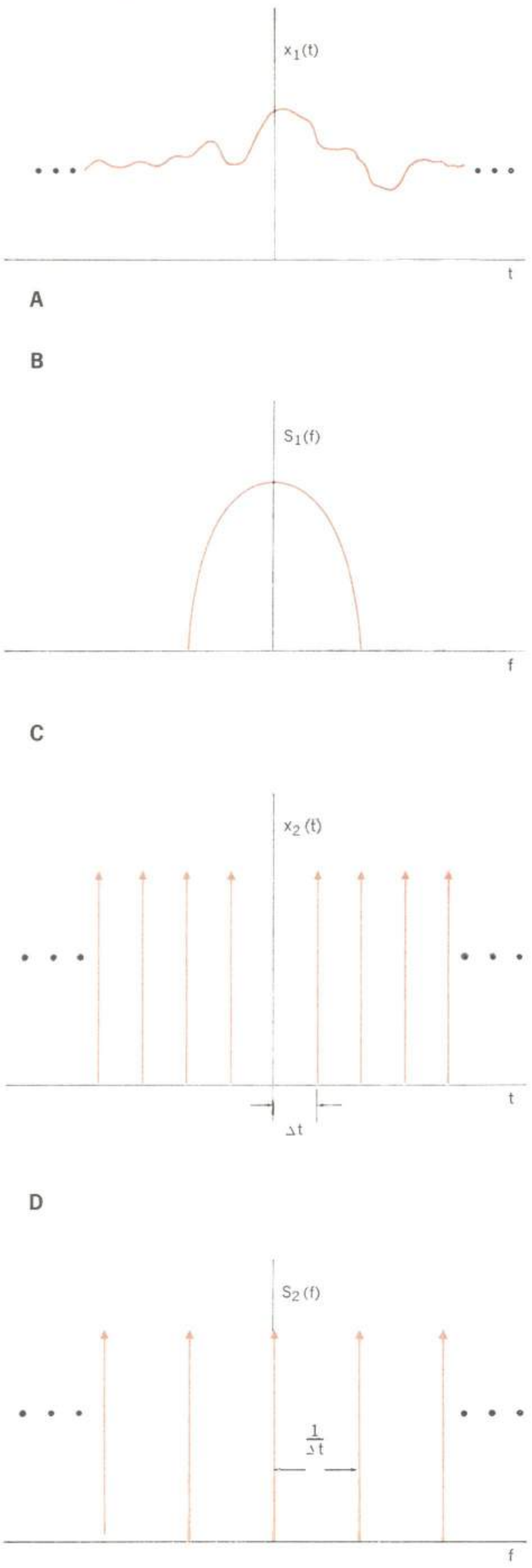
Another application of the Fourier transform is the calculation of power spectra. A power spectrum is defined as a Fourier transformation of the autocorrelation function. For random functions of time the autocorrelation function is expressed as

$$\varphi(t) = \lim_{T\to\infty} \frac{1}{2T}\int_{-T}^{T} x(\tau)x(\tau+t)\,d\tau$$

The power spectrum is the Fourier transform of $\varphi(t)$.

Because the Fourier transform is indeed a powerful tool of analysis, it is not surprising that a search was begun to establish techniques to compute the Fourier transform numerically. The discrete Fourier transform therefore evolved, but it was considered impractical until the development of the fast Fourier transform.

FIGURE 1. Fourier transform pairs.



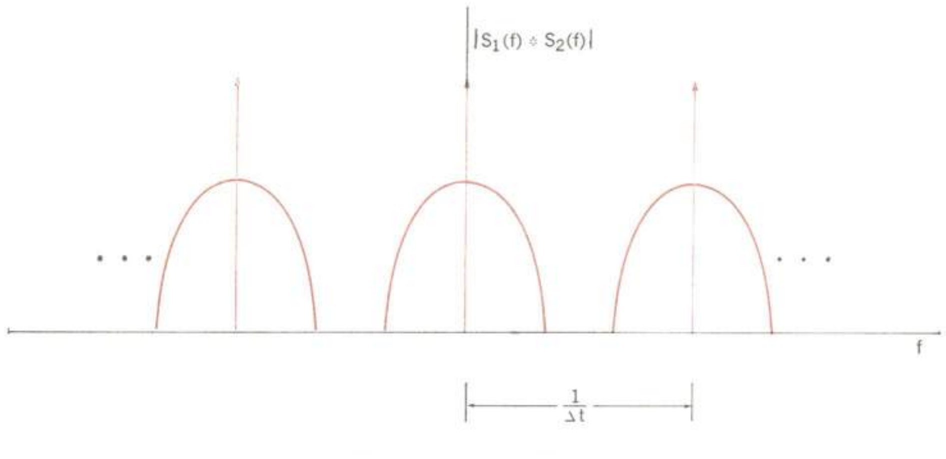


FIGURE 2. Fourier transform of a sampled time function.

Discrete Fourier transform

The relationship between the Fourier transform of continuous and sampled time functions can be established by considering $x_1(t)$ and its Fourier transform $S_1(f)$, illustrated in Figs. 1(A) and (B). One can regard the sampling of $x_1(t)$ as the multiplication of $x_1(t)$ by function $x_2(t)$, an infinite sequence of impulse functions of Fig. 1(C); the Fourier transform of $x_2(t)$ is given in Fig. 1(D). Recall that the multiplication–convolution process forms a Fourier transform pair; therefore, if $x_1(t)$ and $x_2(t)$ are multiplied, $S_1(f)$ and $S_2(f)$ are convolved, yielding the function shown in Fig. 2. The Fourier transform of a sampled time function, from Fig. 2, is then a periodic function of period $1/\Delta t$, where each period contains complete information of the frequency spectrum of $x_1(t)$.

When it is desired to compute the discrete Fourier transform with digital machines, only a finite number of discrete samples of both the time function and the spectrum can be considered. The preceding illustrations apply now heuristically, except that only a sampled version of the periodic spectrum of Fig. 2 can be realized.

In the sampled-data case, the Fourier transform pair given in Eqs. (3) and (4) for N samples becomes

$$S(f_n) = \Delta t \sum_{k=0}^{N-1} x(t_k) e^{-j2\pi f_n t_k} \qquad n = 0, \pm 1, \ldots, \pm N/2$$

$$x(t_k) = \Delta f \sum_{n=-N/2}^{N/2} S(f_n) e^{j2\pi f_n t_k} \qquad k = 0, 1, \ldots, N-1 \quad (6)$$

If we let $t_k = k\Delta t$, $f_n = n\Delta f$, and if we note that $\Delta t = T/N$ and $\Delta f = 1/T$, Eq. (6) becomes

$$S(n) = \Delta t \sum_{k=0}^{N-1} x(k) e^{-j2\pi(nk)/N} \qquad n = 0, 1, \ldots, N-1$$

$$x(k) = \Delta f \sum_{n=0}^{N-1} S(n) e^{j2\pi(nk)/N} \qquad k = 0, 1, \ldots, N-1 \quad (7)$$

where argument n takes on the values $0, 1, \ldots, N-1$, rather than $0, \pm 1, \ldots, \pm N/2$. This substitution in no way alters the expression; it is done to simplify the computational procedure. Term $n = N/2$ corresponds to the *aliasing* or *Nyquist folding* frequency.

Computer evaluation of the first equation of pair (7) is the subject of this article. For computational purposes, the equation can be more easily represented in matrix form as

$$[S(n)] = [W^{nk}] [X_0(k)] \qquad (8)$$

where $[S(n)]$ and $[X_0(k)]$ are $N \times 1$ column matrices and $[W^{nk}]$ is an $N \times N$ matrix with

$$W = e^{-j2\pi/N}$$

Scaling term Δt will be eliminated for clarity of presentation and subscript zero added to vector $[X(k)]$ for continuity of notation in the remainder of the article.

Consider a simple example of performing the computation indicated by Eq. (8). If we choose the number of sample points $N = 4$, Eq. (8) becomes

$$\begin{bmatrix} S(0) \\ S(1) \\ S(2) \\ S(3) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^9 \end{bmatrix} \begin{bmatrix} x_0(0) \\ x_0(1) \\ x_0(2) \\ x_0(3) \end{bmatrix} \qquad (9)$$

In general, execution of (9) requires N^2 complex (W is complex) multiplications and additions.

The fast Fourier transform owes its success to the reduction of the number of complex multiplications and additions. The following discussion presents, on an intuitive level, how the reduction is accomplished. At first reading, do not question the reasoning behind the algorithm, but understand its results.

Intuitive development of the FFT

In using the FFT, it is convenient to choose the number of sample points of $x_0(t)$ according to the relation $N = 2^\gamma$, where γ is an integer. In the preceding example that yielded (9), $N = 4 = 2^\gamma = 2^2$; because $\gamma = 2$, then the FFT could be applied. The first step is to write the matrix $[W^{nk}]$ in (9) as

$$\begin{bmatrix} S(0) \\ S(1) \\ S(2) \\ S(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W^1 & W^2 & W^3 \\ 1 & W^2 & W^0 & W^2 \\ 1 & W^3 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} x_0(0) \\ x_0(1) \\ x_0(2) \\ x_0(3) \end{bmatrix} \tag{10a}$$

Equation (10a) was derived from (9) by using the relation $W^{nk} = W^{nk \bmod N}$. For example, if $N = 4$, $n = 2$, and $k = 3$, then $nk = 6$ and $nk \bmod N = 2$. Recall that $nk \bmod N$ is the remainder upon division of nk by N; hence,

$$W^{nk} = \exp\left[\frac{-j2\pi}{4}(6)\right] = \exp[-j3\pi]$$
$$= W^{nk \bmod N} = \exp\left[\frac{-j2\pi}{4}(2)\right] = \exp[-j\pi]$$

In the second step, (10a) is rewritten as

$$\begin{bmatrix} S(0) \\ S(2) \\ S(1) \\ S(3) \end{bmatrix} = \begin{bmatrix} 1 & W^0 & 0 & 0 \\ 1 & W^2 & 0 & 0 \\ 0 & 0 & 1 & W^1 \\ 0 & 0 & 1 & W^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^2 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \begin{bmatrix} x_0(0) \\ x_0(1) \\ x_0(2) \\ x_0(3) \end{bmatrix} \tag{10b}$$

The method of factorization will be explained later. The value of W^0 is equivalent to unity; both the quantity W^0 and 1 are used in (10b) to develop a generalized final result. Equation (10b) may be verified by multiplying the two square matrices and realizing that the resultant square matrix is equal to $[W^{nk}]$ in (10a), with the exception that rows 1 and 2 have been interchanged. This interchange has been accounted for in (10b) by rewriting the column vector $[S(n)]$; let the row-interchanged vector be denoted by

$$[\overline{S(n)}] = \begin{bmatrix} S(0) \\ S(2) \\ S(1) \\ S(3) \end{bmatrix}$$

Having accepted the fact that (10b) will yield correct computed results, although they are scrambled, one should examine now the number of complex multiplications and additions represented by the equation. First, let

$$\begin{bmatrix} x_1(0) \\ x_1(1) \\ x_1(2) \\ x_1(3) \end{bmatrix} = \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^0 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \begin{bmatrix} x_0(0) \\ x_0(1) \\ x_0(2) \\ x_0(3) \end{bmatrix} \tag{11}$$

that is, column vector $[X_1(k)]$ is equal to the multiplication of the two matrices on the right in (10b). Element $x_1(0)$ is determined by one complex multiplication and one addition

$$x_1(0) = x_0(0) + W^0 x_0(2) \tag{12}$$

It is realized that $W^0 = 1$ and a multiplication is not necessary; in order to develop a generalized result, however, this will be considered as one multiplication. Element $x_1(1)$ is also determined by one complex multiplication and addition. One complex addition is all that is required for finding $x_1(2)$. This follows from the fact that $W^0 = -W^2$; hence,

$$x_1(2) = x_0(0) + W^2 x_0(2)$$
$$= x_0(0) - W^0 x_0(2)$$

where the complex multiplication $W^0 x_0(2)$ has already been performed in the calculation of $x_1(0)$. By the same reasoning, $x_1(3)$ is found by only one complex addition and no multiplications. Vector $[X_1(k)]$ is therefore determined by the use of four additions and two multiplications.

Let us continue by computing

$$\begin{bmatrix} S(0) \\ S(2) \\ S(1) \\ S(3) \end{bmatrix} = \begin{bmatrix} 1 & W^0 & 0 & 0 \\ 1 & W^2 & 0 & 0 \\ 0 & 0 & 1 & W^1 \\ 0 & 0 & 1 & W^3 \end{bmatrix} \begin{bmatrix} x_1(0) \\ x_1(1) \\ x_1(2) \\ x_1(3) \end{bmatrix} \tag{13}$$

As before, $S(0)$ is determined by one multiplication and addition

$$S(0) = x_1(0) + W^0 x_1(1)$$

Because $W^0 = -W^2$, element $S(2)$ is found from one addition. By similar reasoning, $S(1)$ is determined by one complex multiplication and addition and $S(3)$ by only one addition. Vector $[\overline{S(n)}]$ has been computed by a total of $N\gamma/2 = 4$ complex multiplications and $N\gamma = 8$ complex additions; the computation of (9) required $N^2 = 16$ complex multiplications and additions. Obviously, for large values of N the savings become huge and can actually approach 99 percent.

The Cooley–Tukey FFT algorithm can then be considered as a method of factoring an $N \times N$ matrix into γ $N \times N$ $(N = 2^\gamma)$ matrices such that each of the new factored matrices has the special property of minimizing the number of complex multiplications and additions. Referring to the previous example, it is seen that the reduction of multiplications and additions is accomplished by the zero terms that were introduced by matrix factoring.

The scheme of matrix factoring does introduce one discrepancy, however; recall that the computation of (10b) yielded $[\overline{S(n)}]$ instead of $[S(n)]$

$$[\overline{S(n)}] = \begin{bmatrix} S(0) \\ S(2) \\ S(1) \\ S(3) \end{bmatrix} \quad \text{instead of} \quad [S(n)] = \begin{bmatrix} S(0) \\ S(1) \\ S(2) \\ S(3) \end{bmatrix}$$

This rearrangement is inherent in the optimum matrix factoring process and is a relatively minor problem because we can easily generalize a scheme to rearrange vector $[\overline{S(n)}]$ for obtaining $[S(n)]$. The rearrangement procedure can be explained as follows: Rewrite vector $[S(n)]$ computed in (9) by replacing argument n with its binary equivalent

$$\begin{bmatrix} S(0) \\ S(1) \\ S(2) \\ S(3) \end{bmatrix} \quad \text{becomes} \quad \begin{bmatrix} S(00) \\ S(01) \\ S(10) \\ S(11) \end{bmatrix} \tag{14}$$

Observe the result if the binary arguments of (14) are flipped or bit-reversed; that is, 01 becomes 10, 10 becomes 01, etc. Then

$$[S(n)] = \begin{bmatrix} S(0) \\ S(1) \\ S(2) \\ S(3) \end{bmatrix} \quad \begin{matrix} \text{flips} \\ \text{to} \end{matrix} \quad \begin{bmatrix} S(00) \\ S(10) \\ S(01) \\ S(11) \end{bmatrix} = [\overline{(Sn)}]$$

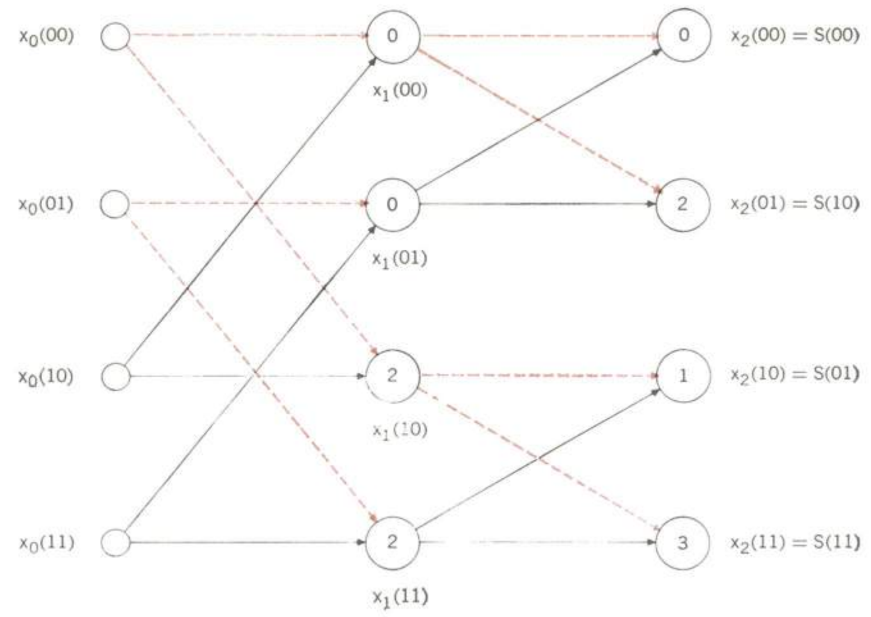


FIGURE 3. Tree-graph representation of factored matrices.

The scrambled vector $[\overline{S(n)}]$ is merely $[S(n)]$ with the binary argument flipped. To illustrate this point further, suppose $N = 8$; the determination of vector $[\overline{S(n)}]$ by matrix factorization yields

$$[\overline{S(n)}] = \begin{bmatrix} S(000) \\ S(100) \\ S(010) \\ S(110) \\ S(001) \\ S(101) \\ S(011) \\ S(111) \end{bmatrix}$$

$$= \begin{bmatrix} S(0) \\ S(4) \\ S(2) \\ S(6) \\ S(1) \\ S(5) \\ S(3) \\ S(7) \end{bmatrix} \text{ instead of } \begin{bmatrix} S(000) \\ S(001) \\ S(010) \\ S(011) \\ S(100) \\ S(101) \\ S(110) \\ S(111) \end{bmatrix} = [S(n)]$$

Therefore, by bit-reversing the argument of vector $[\overline{S(n)}]$, one knows exactly where each component of the scrambled vector $[\overline{S(n)}]$ actually belongs.

An obvious question at this stage in the development of the FFT algorithm is "How does one obtain the general form of the factored matrices for an $N \times N$ matrix?" To answer this question, let us continue the example for $N = 4$ and convert (10b) to the tree graph of Fig. 3. The sampled data $[X_0(k)]$ are represented by a vertical column of nodes on the left of the graph; argument k has been replaced by its binary equivalent. The center vertical column of nodes corresponds to vector $[X_1(k)]$ computed in (11) and the right-hand column of nodes represents vector $[X_2(k)] = [\overline{S(n)}]$; see Eq. (13). Note that each node is entered by a dashed and a solid line; within each node is an integer.

The solid line brings a quantity from one of the nodes in a previous column, multiplies the quantity by W^p, where p is the integer in the circle, and the product is added to the quantity brought by the dashed line. If we were at node $x_1(00)$, then

$$x_1(00) = x_0(00) + W^0 x_0(10) \qquad (15)$$

Equation (15) is just (12) with the arguments expressed as binary numbers. A similar procedure is used for expressing each of the remaining nodes.

To summarize, Fig. 3 is a graph of the computational procedure that was used in the factored-matrices representation of (10). Further, γ vertical arrays or columns are computed; that is, each vertical column computed corresponds to one of the factored matrices.

Generalized graph construction

A general method of establishing a graph of the form of Fig. 3 for any $N = 2^\gamma$ will be formulated by the following set of rules:

1. Assume N sample values of the time function $x_0(t)$ constitute the first vertical array of nodes. Let the array of nodes or vector be denoted by $[X_0(k)]$, where $k = 0, 1, \ldots, N - 1$ represent the sample times or address (location) of function $x_0(t)$; argument k is expressed as a binary number. This binary number determines the location of node $x_0(k)$; see Fig. 3. The binary arguments or locations are expressed by γ bits; for $N = 4$, each address is expressed by $\gamma = 2$ bits.

The other arrays $[X_l]$ are drawn successively to the right, and the nodes in each array are addressed as binary numbers according to the previous discussion. Nodes or locations on the same horizontal level have the same binary address. At each node, a circle is drawn and a number is written in this circle, as explained by rule 2.

2. The number in the circle of the kth node in the lth array (note that the left-most array is the 0th array) is found by (a) writing the binary number k, (b) scaling

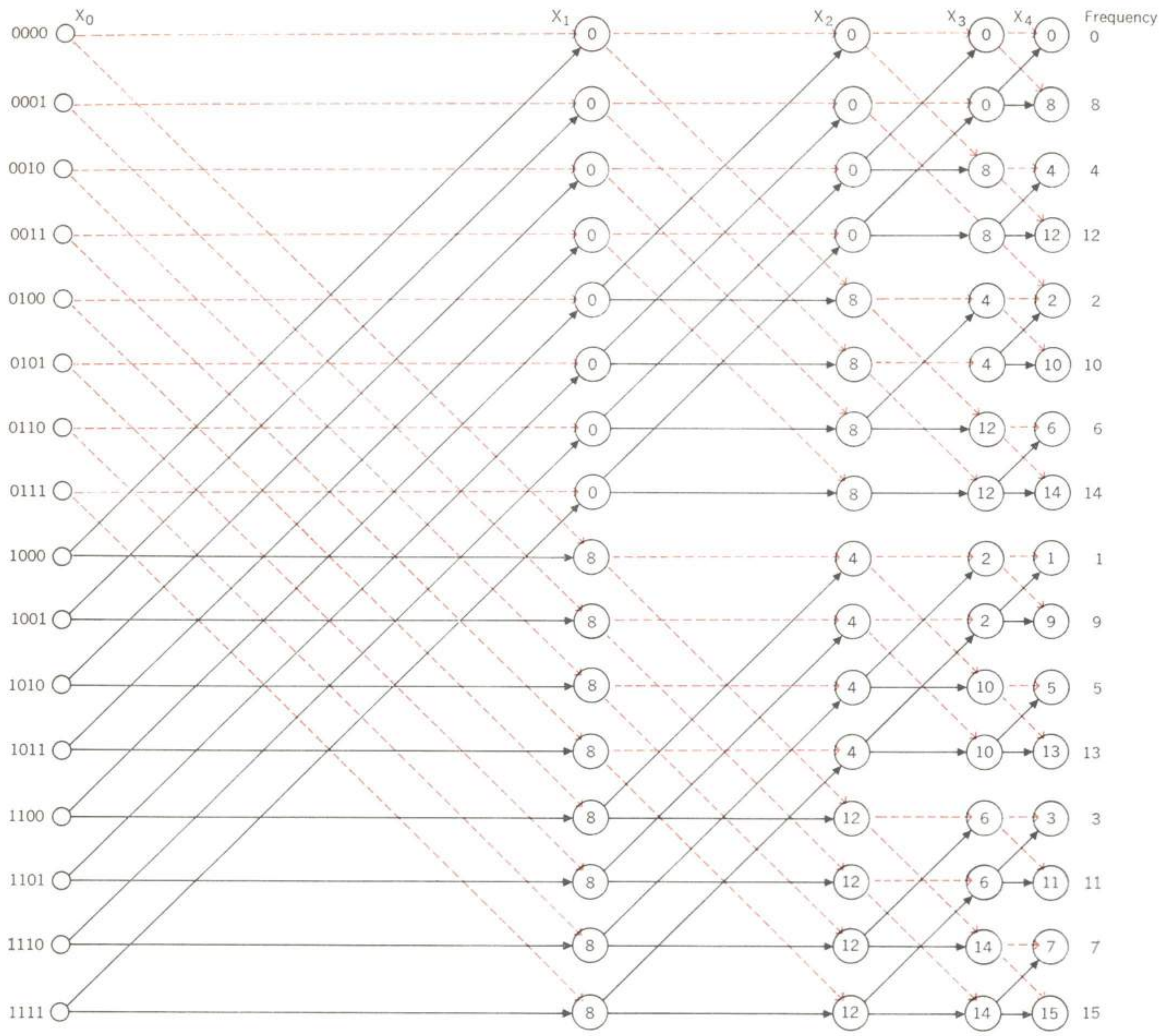


FIGURE 4. Tree graph for N = 16

or sliding this number ($\gamma - l$) places to the right and filling in the newly opened bit positions on the left by zeros, and (c) reversing the order of the bits.

For example, referring to Fig. 3, let $\gamma = 2$, $k = 2$, and $l = 1$; then k in binary is 10. Scale $\gamma - l = 2 - 1 = 1$ to the right, that is 01, and then reverse the order of bits yielding 10 or integer 2.

3. For generality, let the binary representation of k be $k_{\gamma-1} \ldots k_1k_0$. For example, if $\gamma = 2$ and $k = 2$, then $k_0 = 0$ and $k_1 = 1$; therefore, $k_1k_0 = 10$. This additional nomenclature is needed for the following general statement: In the lth array, node k (in binary form $k_{\gamma-1} \ldots k_1k_0$) has a solid line drawn to it from a node in the $(l - 1)$th array. The address of the node in the $(l - 1)$th array is the same as node k, except that bit $k_{\gamma-l}$ must be a one. The dashed line comes from a node in the $(l - 1)$th array whose address is the same, but bit $k_{\gamma-l}$ must be a zero.

For example, let $\gamma = 2$, $l = 1$, and $k = 1$; k in binary is $k_1k_0 = 01$. The bit in question is $k_{\gamma-1} = k_1$. The solid line will come from address (11) in the 0th array; similarly, the dashed line will come from address (01) in the 0th array.

The preceding rules are sufficient to sketch the tree graph for computing Fourier transform values for any value of $N = 2^\gamma$. To clarify these rules and to develop another tree graph for further discussion, let us sketch the graph for $N = 16$ ($\gamma = 4$). By rule 1, the 16 data points form the left vertical array denoted by $[X_0]$ in Fig. 4; the location or address of each point is described by a binary number having $\gamma = 4$ bits. Because $\gamma = 4$, four additional arrays $[X_l]$ are drawn successively to the right: that is, $[X_1]$, $[X_2]$, $[X_3]$, and $[X_4]$.

If we apply rule 2, integers in the circles at each node are determined. For example, for array $l = 3$, mode 8 (the left-most array is array $l = 0$), the integer in the node is found by writing 8 in binary as 1000; scaling it $(\gamma - l) = 1$ places to the right yields 0100; and reversing the order of the bits yields 0010. The integer 2 therefore appears in the circle of the node in question.

Finally, by rule 3, the origin of the solid and dashed lines to each node is determined. In Fig. 4, consider array 1, node 8. By rule 3, a solid line emanates from the node in array $l = 0$ whose location is the same as 1000, except that bit $k_{\gamma-1} = k_3$ must be one (location 1000). The dashed line comes from a node in array $l = 0$ whose location is the same as 1000 except that bit k_3 is zero, or location 0000. Similar reasoning can be employed to determine the origin of all the dashed and solid lines shown in Fig. 4.

Note that node location 1000 in array $l = 1$ is only affected by node locations 0000 and 1000 in array $l = 0$. Node locations 0000 and 1000 in the 0th array only affect one other node in the first array, node location 0000. In general, there exist two nodes in array 1 that are affected by the same pair of nodes in array 0; no other nodes in 1 are affected by either of the two nodes in array 0. This statement implies that there is a savings of half the number of multiplications indicated by the tree graph.

Examining further the node pair 0000 and 1000 in array 1, we see that the solid line entering the pair stems from the same node in array 0 and that the dashed line follows a similar pattern. A solid line going to a node implies the quantity brought by the solid line is to be multiplied by W raised to the integer power in the node; the integers in nodes 0000 and 1000 differ by $N/2$. Because $W^{N/2} = (e^{-j2\pi/N})^{N/2} = -1$, then $W^{\beta} = -W^{\beta+N/2}$, where β is integer valued. For the two nodes in question, therefore, one multiplication is saved since the solid line entering the two nodes stems from the same node in array 0.

Each node in arrays $l = 1$, 2, 3, and 4 has associated with it a node to which all the foregoing arguments apply; in each array half the number of multiplications implied by the nodes can be saved. This argument is just a tree-graph explanation of the multiplication savings presented in the factored matrices discussed earlier.

The Fourier transform values desired are found in array $l = 4$ in Fig. 4. As indicated before, the location or argument of these transform values is scrambled. For example, the transform component located in 0001 is actually the frequency component $S(1000)$. This is determined by bit-reversing the location, where 0001 flips to 1000. The address of array $[X_4]$ must be bit-reversed before the final transform values are obtained.

Verification of tree graphs

To verify a tree graph, such as Fig. 4, it is necessary to show that the γth array corresponds to those values obtained from

$$S(n) = \sum_{k=0}^{N-1} x_0(k) e^{-j2\pi nk/N} \quad n = 0, 1, \ldots N-1 \qquad (16)$$

From (16), each value of transform $S(n)$ is a weighted sum of all the discrete signal values, $x_0(k)$. Also, from Fig. 4, each node in the γth array is a weighted sum of all points in the $[X_0]$th array. To verify the FFT it is necessary to show that the weights in the tree graph and (16) are in correspondence.

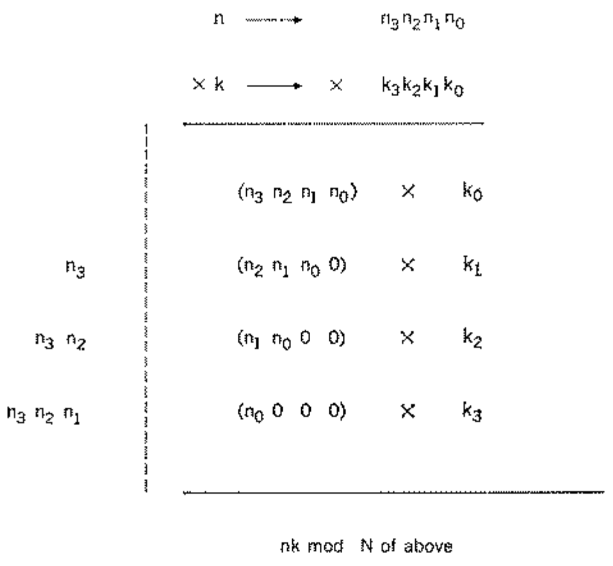


FIGURE 5. Binary computation of nk mod N.

To show this correspondence, first examine the weight contributed by each signal value $x_0(k)$ in (16); each sample contributes

$$e^{-j2\pi nk/N} = W^{nk} = W^{nk \bmod N}$$

It is of interest to indicate the binary computation of nk mod N. Let the binary representation of n and k be $n \rightarrow n_3n_2n_1n_0$ and $k \rightarrow k_3k_2k_1k_0$, where the notation is exactly as previously discussed. Note that only four bits were assumed in the binary representation; that is, γ was assumed to be 4. In Fig. 5 the binary computation of nk mod N is shown; the peculiar form of writing the multiplication is necessary for illustrating further points to be developed. The terms to be left of the dashed line do not contribute to the sum mod N. For example, consider $n = 13$, $k = 5$, and $\gamma = 4$:

$$\begin{array}{rl}
n = 13 \rightarrow & 1101 \\
\times\, k = 5 \rightarrow & \times\ 0101 \\
\hline
 & (1101) \times 1 \\
1 & (1010) \times 0 \\
1\ 1 & (0100) \times 1 \\
1\ 1\ 0 & (1000) \times 0 \\
\hline
1\ 0\ 0 & 0001 \qquad 65 \bmod 16 = 1
\end{array}$$

If we use the binary representation of the computation of nk mod N, W^{nk} can be factored as

$$\begin{aligned} W^{nk} &= W^{nk \bmod N} \\ &= W^{k_3(n_0000)}\, W^{k_2(n_1n_000)}\, W^{k_1(n_2n_1n_00)}\, W^{k_0(n_3n_2n_1n_0)} \quad (17) \end{aligned}$$

It will be shown that a tree graph corresponds to the factoring of W^{nk} in (17).

Examination of Fig. 4 shows that node $x_0(k)$, the kth time sample, is connected to any of the spectrum nodes $x_4(k)$ by γ lines. For example, $x_0(0000)$ and $x_4(0000)$ are connected only by four horizontal dashed lines; these γ lines correspond to the factoring of W^{nk} in (17). The solid or dashed characteristics of the lines depend on the value of k_i. If $k_i = 1$, the line is solid; for $k_i = 0$, it is

Historical evolution of the fast Fourier transform

The beginnings of the modern-day fast Fourier transform dates back to 1903 when Runge[2] described a computational technique for 12- and 24-point Fourier transforms. Runge's scheme was not generalized until 1942 when Danielson and Lanczos[3] published a method for the optimal computation of $N = 2^k$-point Fourier transforms; this efficient method, however, passed unnoticed to interested researchers.

Another line of development was introduced in 1937 by Yates,[4] whose algorithm efficiently computed the interaction of 2^n factorial experiments; the form of the algorithm differed from that of Danielson and Lanczos.[3] Davies and others[5] extended Yates' method of 3^n experiments. Good[6] later extended this approach to general factorial experiments and at the same time outlined a procedure for the computation of N-point Fourier transforms where $N = r_1r_2 \ldots r_m$, that is, composite with mutually prime factors. This restriction of N was removed by Cooley and Tukey[1] and the first computer program to compute the FFT was probably written at this time. Still later a different form of the algorithm was introduced by Gentleman and Sande.[7] The Cooley–Tukey paper sparked a renewed interest in the Fourier transform as an analysis tool.

Stockham[8] disclosed the efficiency of the fast Fourier transform as applied to convolution and correlation processing. Because of the high interest in the fast Fourier transform, the entire June 1967 issue of IEEE TRANSACTIONS ON AUDIO AND ELECTROACOUSTICS was devoted to the FFT.

dashed. This is consistent with the previously defined rule that a solid line entering a node implies a multiplication by W raised to the integer in that node.

Assume we are to compute the Fourier transform value with the argument $n_3n_2n_1n_0$; this value will end up in position $n_0n_1n_2n_3$. Let us start in the 0th array at position $k_3k_2k_1k_0$ and proceed by the solid and dashed lines to location $n_0n_1n_2n_3$ (see Fig. 4). One takes the following positions in the various arrays (an example is indicated in parentheses; start at location 0111 and head for 0011).

Array	Position	Power of W	
0	$k_3k_2k_1k_0$ (0111)		
1	$n_0k_2k_1k_0$ (0111)	n_0000	(0000)
2	$n_0n_1k_1k_0$ (0011)	n_1n_000	(0000)
3	$n_0n_1n_2k_0$ (0011)	$n_2n_1n_00$	(1000)

The power of W denoted by the contents of the circle at each node is given in the third column and is determined by rule 2 (address scaled $\gamma - l$ places to the right with the bits reversed). These weights then correspond to the terms in (17) and the correspondence between Fig. 4 and (17) is complete. The tree-graph representation and the rules presented here are essentially derived from an unpublished memorandum by C. M. Rader of the M.I.T. Lincoln Laboratory.

Conclusions

The value of the fast Fourier transform is in the reduction of computer time in evaluating the discrete Fourier transform. An N-point transformation by the direct method requires a time proportional to N^2 whereas the FFT requires a time proportional to $N \log_2 N$. The approximate ratio of FFT to direct computing time is given by

$$\frac{N \log_2 N}{N^2} = \frac{\log_2 N}{N} = \frac{\gamma}{N}$$

where $N = 2^\gamma$. For example, if $N = 2^{10}$, the FFT requires less than 1/100 of the normal computing time.

Convolution and correlation, both extremely useful mathematical techniques in time-series analysis, are usually computed digitally by forming the lagged product

$$\frac{1}{N} \sum_{\tau=0}^{N-1} x_1(t - \tau)x_2(\tau)$$

This calculation consumes considerable computer time with conventional techniques. Using the FFT, one can reduce computing time as follows: First, using the FFT, $x_1(t)$ and $x_2(t)$ are Fourier-transformed, yielding $S_1(f)$ and $S_2(f)$. Terms $S_1(f)$ and $S_2(f)$ are then multiplied and the resultant is inverse-Fourier-transformed by use of the FFT. Stockham[8] showed that for $N \approx 28$, the FFT method is faster than the conventional lagged-products approach. For $N = 4096$, Stockham estimates that the FFT technique is 80 times faster.

General areas in which the FFT is finding successful application include digital signal enhancement, image enhancement in character recognition, spatial filtering, real-time digital speech analysis, power spectra estimation, and system simulation. In fact, the computer algorithm has opened new avenues of scientific investigation never before considered practical because of exorbitant computing times. In essence, the scientific analyst now has opened to him all those frequency-domain analysis techniques once considered inefficient.

REFERENCES

1. Cooley, J. W., and Tukey, J. W., "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, pp. 297–301, Apr. 1965.
2. Runge, C., *Z. Math. Phys.*, vol. 48, p. 443, 1903.
3. Danielson, G. C., and Lanczos, C., "Some improvements in practical Fourier analysis and their application to X-ray scattering from liquids," *J. Franklin Inst.*, vol. 233, pp. 365–380 and 435–452, Apr. 1942.
4. Yates, F., "The design and analysis of factorial experiments," Commonwealth Agriculture Bureaux, Farnam Royal, Bucks., England, 1937.
5. Box, G. E. P., Connor, L. R., Cousins, W. R., Davies, O. L. (ed.), Himsworth, F. R., and Sillitto, G. P., *The Design and Analysis of Industrial Experiments.* London: Oliver & Boyd, 1954, pp. 363–366.
6. Good, I. J., "The interaction algorithm and practical Fourier analysis" *J. Roy. Statist. Soc.*, ser. B, vol. 20, pp. 361–372, 1958.
7. Gentleman, W. M., and Sande, G., "Fast Fourier transforms for fun and profit," *1966 Fall Joint Computer Conference, AFIPS Proc.*, vol. 29, pp. 563–578.
8. Stockham, T. G., "High-speed convolution and correlation," *1966 Joint Computer Conference, AFIPS Proc.*, vol. 28, pp. 229–233.