Numerical Analysis

J. F. TRAUB, Editor

A Fast Fourier Transform Algorithm for Real-Valued Series

GLENN D. BERGLAND
Bell Telephone Laboratories, Whippany, New Jersey

A new procedure is presented for calculating the complex, discrete Fourier transform of real-valued time series. This procedure is described for an example where the number of points in the series is an integral power of two. This algorithm preserves the order and symmetry of the Cooley-Tukey fast Fourier transform algorithm while effecting the two-to-one reduction in computation and storage which can be achieved when the series is real. Also discussed are hardware and software implementations of the algorithm which perform only $(N/4)\log_2(N/2)$ complex multiply and add operations, and which require only N real storage locations in analyzing each N-point record.

KEY WORDS AND PHRASES: fast Fourier transform, time series analysis, digital filtering, spectral analysis, real-time spectrum analyzers, Fourier analysis, discrete Fourier transform, digital spectrum analysis, Fourier analysis algorithm, Fourier synthesis algorithm
CR CATEGORIES: 3.80, 3.81, 4.9, 5.49, 6.22

Introduction

Recent papers regarding the fast Fourier transform (FFT) have largely dealt with applications and refinements of the algorithms originally reported by Cooley and Tukey [1]. The technique has been applied to vocoding [2, 3], digital filtering [4], analysis of underwater sound recordings [5], analysis of electroencephalographic data [6], and many other areas. Because of this wide range of application, several groups have proposed hardware implementations of the algorithm [7–9]. Refinements of the algorithm include: rearranging the calculations or exploiting the symmetries of the complex exponential weights to obtain efficient software implementations [10–14]; and improving the computational efficiency when the input series consists solely of real numbers [14, 15].

There have been two basic approaches to the evaluation of real-valued time series. The first approach makes use of the conventional complex FFT algorithm and depends upon forming an artificial $N/2$-term complex record from each N-term real record [15]. A more direct approach has recently been proposed by Edson [16] in which he suggests specializing the complex FFT algorithm by eliminating those computations which lead to redundant results. The algorithm discussed in this paper extends this latter approach not only to decrease the required computation but also to preserve the order and symmetry which made the original complex Cooley-Tukey algorithm so convenient to implement in both hardware and software.

Redundancy in the Cooley-Tukey Algorithm

An interpretation of the A_i values of the Cooley-Tukey algorithm for $i = 0, 1, 2, \cdots, m$, where $N = 2^m$, has been noted by Cooley [17] and described in detail by Shively [7] and others [18]. This interpretation describes the A_i values at each stage as being sets of unnormalized Fourier coefficients formed from interleaved sets of samples. Although this concept can be conveniently generalized to apply when $N = r_1 r_2 \cdots r_m$, an example will be carried through for $N = 2^m$. In this case the original samples (i.e. the A_0 values) can be thought of as N distinct one-term Fourier series representations of the dc (direct current) value of the time function. In Figure 1

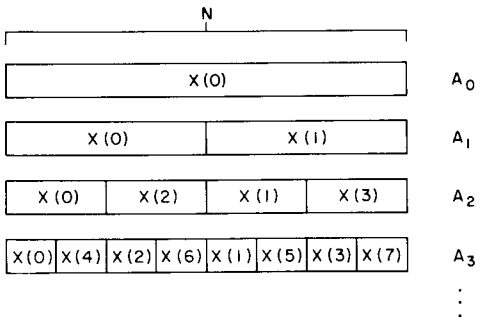

FIG. 1. Intermediate results of the Cooley-Tukey algorithm interpreted as Fourier coefficients of interleaved sets of samples

the N-term sequence of A_0s has been denoted by the top rectangle. As these may be interpreted as N separate estimates of dc, they have been labeled $X(0)$.

The A_1 values are estimates of the dc term and the first harmonic (i.e. the $X(0)$ and $X(1)$ Fourier coefficients) as evaluated from $N/2$ separate 2-term Fourier series. As shown in Figure 1, the first half of the A_1s represents $X(0)$ coefficients and the second half represents $X(1)$ coefficients.

In like manner the A_2s represent $X(0)$, $X(1)$, $X(2)$, and $X(3)$ Fourier coefficients as evaluated from $N/4$ separate 4-term Fourier series, and the A_3s represent the Fourier coefficients evaluated from $N/8$ separate 8-term Fourier series.

In each case a stage in the recursive algorithm represents combining unnormalized Fourier coefficients formed from two interleaved sets of samples to form twice as many Fourier coefficients (because the effective sampling rate is doubled) from the combined set.

When the original time series consists of only real numbers, this interpretation can be profitably carried one step further. In Figure 2 a set of graphs shows the A_0s, A_1s, etc., as Fourier coefficients displayed as a function of

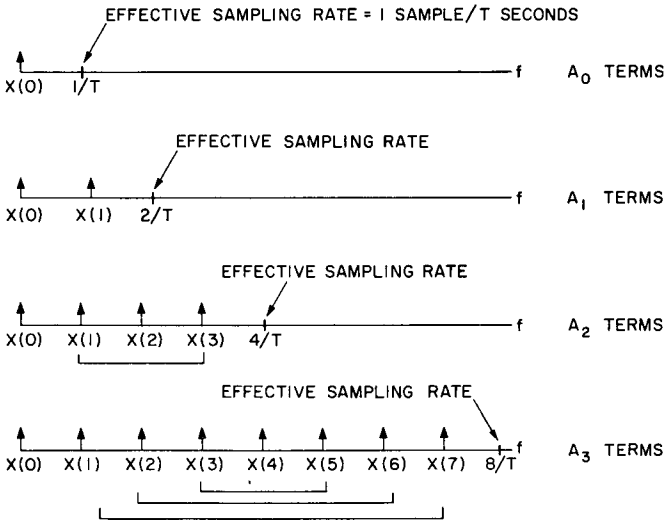

FIG. 2. Intermediate results of Cooley-Tukey algorithm plotted as a function of the frequencies they represent

frequency. Note that the set of A_0 and A_1 values are both composed of only real numbers; however, the set of A_2-values consists of $N/2$ real numbers (the $X(0)$ and $X(2)$ terms), $N/4$ complex $X(1)$ terms, and $N/4$ complex $X(3)$ terms. From Figure 2 it is apparent that the third harmonic terms are above one-half the effective sampling frequency, and as they were formed from real-valued time samples, the set of $X(3)$ terms and the set of $X(1)$ terms are complex conjugates. Thus only the $N/4$ values of the first harmonic are truly independent. This means that to store the independent A_2 values we only need storage for $N/2$ real numbers and $N/4$ complex numbers, i.e. N storage locations.

Analogously, the A_3 values representing the 5th, 6th, and 7th harmonics are complex conjugates of the 3rd, 2nd, and 1st harmonics, respectively. Since the $X(0)$ and $X(4)$ terms are real and only the $X(1)$, $X(2)$, and $X(3)$ complex terms must be saved, we still require only N storage locations.

This process continues until in the last stage the $N/2 - 1$ independent complex Fourier coefficients and the 2 independent real Fourier coefficients are formed by combining all N of the original real samples. Thus it is apparent that for a real-valued time series of N samples we have only N independent numbers at the beginning and N independent numbers at the end (i.e. $N/2 - 1$ complex numbers and 2 real numbers), and no additional storage is required in the intervening steps. Since the discarded intermediate results at any stage can be formed by simply conjugating the appropriate saved value, this process proves to be quite convenient to implement.

Developing the Algorithm

An algorithm is developed for computing only the $N/2 + 1$ independent Fourier coefficients while preserving the order and symmetry, which makes the complex Cooley-Tukey algorithm so attractive. Since the input series consists of only N real numbers and the resulting Fourier coefficients consist of only $N/2 - 1$ complex numbers and 2 real numbers, it is convenient to perform all of the iterations using an array of only N real storage locations. This compares with the N complex storage locations required in implementing the complex Cooley-Tukey algorithm.

It is also desirable to keep the indexing of intermediate results regular and the number of different mathematical operations required to a minimum. By deviating from the Cooley-Tukey order of computation, one can achieve both a regular indexing pattern and a standard set of mathematical operations.

The Cooley-Tukey algorithm for complex time series is shown diagrammatically in Figure 3 for the example of $N = 16$. The basic set of mathematical operations is represented by the block entitled "Complex Calculation." This block denotes that: (1) the second complex input is multiplied by the appropriate power of W (where $W = \exp(2\pi i/N)$); (2) the resulting product is added to the first complex input to form the first output term; (3) this product is subtracted from the first input term to form the second output term. Note that in Figure 3 the "G" denotes temporary storage of the first input and does not denote an arithmetic operation.

Although this set of "complex calculations" is actually

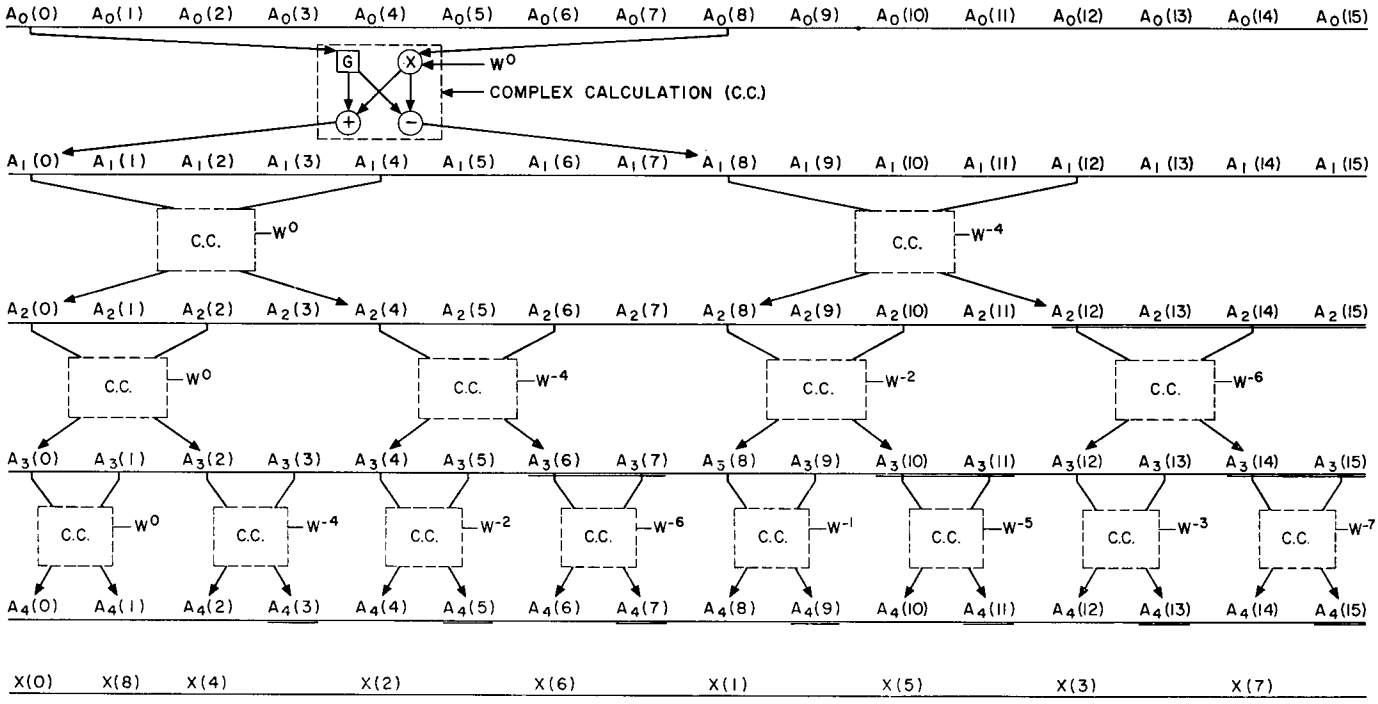

FIG. 3. The Cooley-Tukey complex fast Fourier transform algorithm shown diagrammatically for the example of $N = 16$

performed $N/2$ times during each iteration, Figure 3 shows only the accessing and storage patterns for the first set in each group. Each pattern shown is applied sequentially to all of the operands in its group. In the first iteration, for example, the set of "complex calculations" is shown applied to the $A_0(0)$ and $A_0(8)$ terms. It is next applied to the $A_0(1)$ and $A_0(9)$ terms, the $A_0(2)$ and $A_0(10)$ terms, and so on. By showing only the first operation of each sequence, the diagram can be kept quite simple while still denoting all of the necessary information.

From the representation of the Cooley-Tukey algorithm shown in Figure 3, it is convenient to describe the real-valued input algorithm in terms of its deviation from the Cooley-Tukey procedure. The real-valued input algorithm computes intermediate results in a different order from the Cooley-Tukey algorithm, but the Cooley-Tukey labeling of the intermediate results can be carried through to verify that all of the required operations are being performed. As discussed in the previous section, the essential differences will be that the redundant intermediate results of the Cooley-Tukey algorithm will not be computed, and real storage locations will be assumed instead of N complex storage locations. (Note that the redundant Cooley-Tukey intermediate results have a double line under them in Figure 3.)

The computation of the independent Cooley-Tukey intermediate results via the real-valued input algorithm is shown diagrammatically in Figure 4. Note that the set of "complex calculations" still consists of operating on two complex inputs and forming two complex outputs, but the real and imaginary parts of these terms are stored in different locations. Note also that the terms which are to be subtracted are conjugated. (Or as an alternative, the result of the subtraction could be conjugated.)

In Figure 4 the nonredundant results of each iteration of the Cooley-Tukey algorithm are shown on separate lines with the imaginary part of each term being labeled with the prefix i. Note that the imaginary parts of the saved terms are stored in the locations vacated by the discarded terms. By deviating from the computational order of the Cooley-Tukey algorithm, an orderly indexing pattern is obtained, and the real and imaginary parts of each Fourier coefficient are formed in adjacent locations.

A common set of arithmetic operations can be carried through the entire algorithm as denoted by the CC (complex calculation) box in Figure 4. This calculation differs from that performed in the Cooley-Tukey algorithm only in that one of the results must be conjugated before being stored back in memory.

The objective of maintaining an orderly and easily implemented indexing pattern is met by making one small change to the diagram of Figure 4. Note that the operation of multiplying by $W(N/4)$ (i.e. $-i$) is performed in

Figure 4 by a negation and a relabeling. The relabeling occurs in the "No Operations Necessary" space. If the operations below these expressions are all performed one iteration earlier, the accessing of the new "regrouped" iteration will have the regularity being sought. This procedure has the added effect of eliminating the mth iteration of the algorithm except for one addition and one subtraction.

Based on the observations made above, one can define a new fast Fourier transform algorithm for real-valued inputs. If the original series is designated $B_0(k)$ for $k = 0, 1, \cdots, N-1$, the operations shown in Figure 5 will compute the Fourier coefficients provided that one of the following conditions is met:

1. The Fourier coefficients corresponding to DC and one-half the sampling frequency must be zero; or

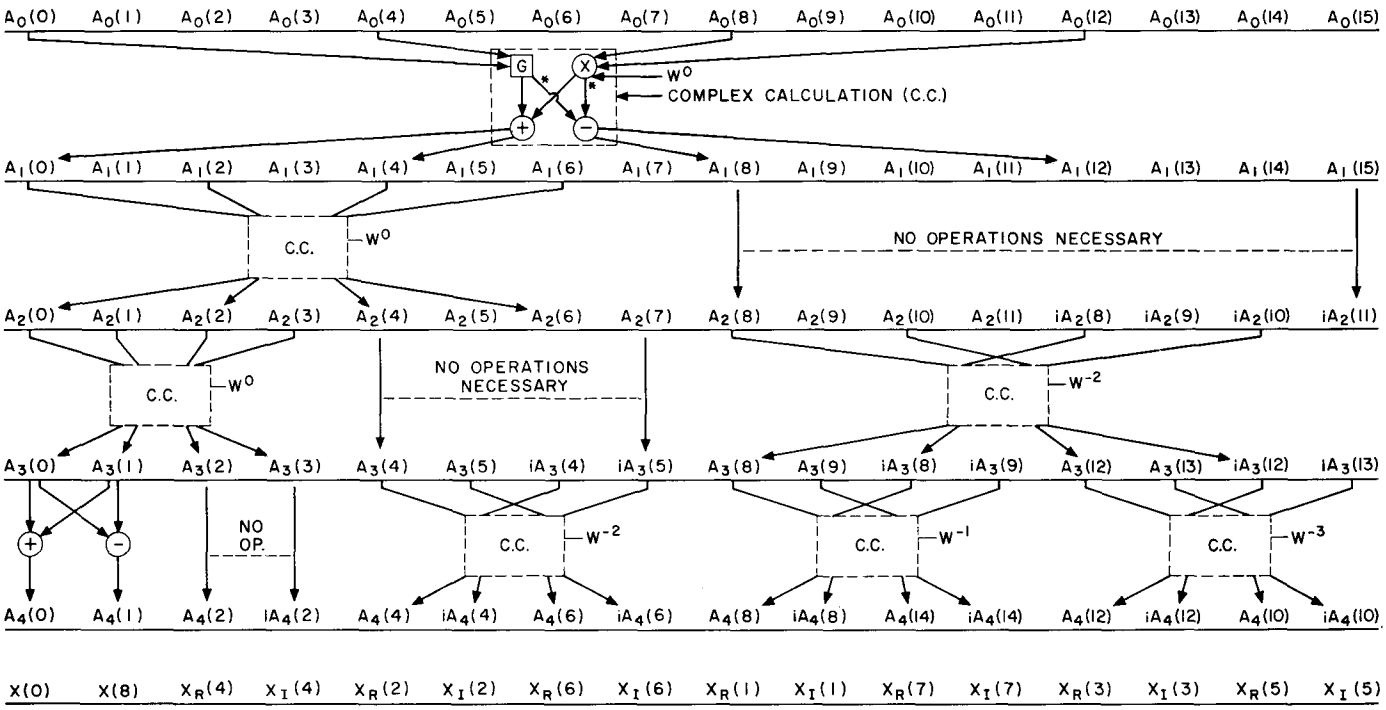

FIG. 4. The nonredundant intermediate results of the Cooley-Tukey algorithm as they are formed by the real-valued input algorithm

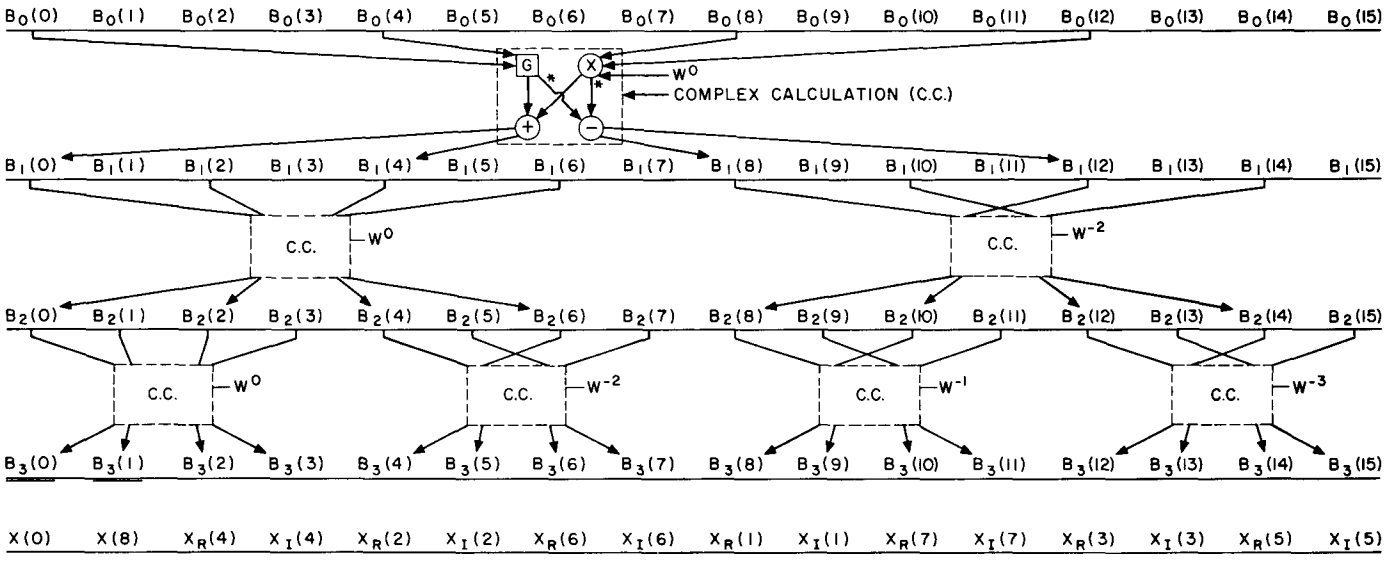

FIG. 5. The real-valued input fast Fourier transform algorithm for $N = 16$

2. $B_3(0)$ and $B_3(1)$ must be replaced by their sum and difference, respectively.

The properties of this algorithm can be summarized as follows:

1. The redundant Fourier coefficients in each iteration above one half the effective sampling frequency are neither computed nor stored.
2. The intermediate results are accessed and stored in a regular and easily implemented pattern.
3. The real and imaginary parts of the final Fourier coefficients are formed in adjacent storage locations.
4. Only N real storage locations are required throughout the computation. These store the original data points, the intermediate results, and the final results.
5. The same set of complex arithmetic operations is performed during the entire algorithm. Only the accessing order has to be changed when the operands are actually real.
6. The powers of W are called in the same order during each iteration.
7. Only 3 ($m-1$ when $N = 2^m$) complete iterations are required.

Fourier Analysis Recursive Equations

The real-valued input algorithm is best described for the case of $N = 2^m$ by the recursive equations which are developed below.

Consider the problem of evaluating a complex Fourier series of the form

$$X(j) = \sum_{k=0}^{N-1} B_0(k) W^{-jk} \tag{1}$$

where $W = e^{2\pi i/N}$, $B_0(k)$ is real, $j = 0, 1, \cdots, N/2$, and $N = 2^m$.

If k is expressed in the form

$$k = k_{m-1} \cdot 2^{m-1} + \cdots + k_1 \cdot 2^1 + k_0, \tag{2}$$

the original N storage locations into which the $B_0(k)$ values are stored can be labeled as $B(k_{m-1}, \cdots, k_1, k_0)$ where $k_{m-1} = \cdots = k_1 = k_0 = 0, 1$. Since two locations are required to specify each complex number, it is convenient to define the following notation:

$$\begin{aligned} &\hat{B}_L\left(k_{m-1}, \cdots, k_{m-L+1}, \begin{pmatrix} a_1, a_0 \\ b_1, b_0 \end{pmatrix}, k_{m-L-2}, \cdots, k_0\right) \\ &\quad = B_L(k_{m-1}, \cdots, k_{m-L+1}, a_1, a_0, k_{m-L-2}, \cdots, k_0) \\ &\quad + iB_L(k_{m-1}, \cdots, k_{m-L+1}, b_1, b_0, k_{m-L-2}, \cdots, k_0). \end{aligned} \tag{3}$$

The L subscript denotes that these values are the results of the Lth iteration of the algorithm as outlined in Figure 5.

As shown in Figure 5 the operands of the first group in each iteration are accessed in an order different from the operands of the other groups. For $N = 2^m$, the recursive equations for the first group can be written in the form

$$\begin{aligned} &\hat{B}_L\left(k_{m-1}, \cdots, k_{m-L+1}, \begin{pmatrix} 0,0 \\ 0,1 \end{pmatrix}, k_{m-L-2}, \cdots, k_0\right) \\ &\quad = \hat{B}_{L-1}\left(k_{m-1}, \cdots, k_{m-L+1}, \begin{pmatrix} 0,0 \\ 0,1 \end{pmatrix}, k_{m-L-2}, \cdots, k_0\right) \\ &\quad + \hat{B}_{L-1}\left(k_{m-1}, \cdots, k_{m-L+1}, \begin{pmatrix} 1,0 \\ 1,1 \end{pmatrix}, k_{m-L-2}, \cdots, k_0\right) \\ &\quad \cdot W(k_{m-1}, \cdots, k_{m-L+1}) \end{aligned} \tag{4}$$

and

$$\begin{aligned} &\hat{B}_L\left(k_{m-1}, \cdots, k_{m-L+1}, \begin{pmatrix} 1,0 \\ 1,1 \end{pmatrix}, k_{m-L-2}, \cdots, k_0\right) \\ &\quad = \hat{B}^*_{L-1}\left(k_{m-1}, \cdots, k_{m-L+1}, \begin{pmatrix} 0,0 \\ 0,1 \end{pmatrix}, k_{m-L-2}, \cdots, k_0\right) \\ &\quad - \hat{B}^*_{L-1}\left(k_{m-1}, \cdots, k_{m-L+1}, \begin{pmatrix} 1,0 \\ 1,1 \end{pmatrix}, k_{m-L-2}, \cdots, k_0\right) \\ &\quad \cdot W^*(k_{m-1}, \cdots, k_{m-L+1}) \end{aligned} \tag{5}$$

where $k_{m-1} = \cdots = k_{m-L+1} = 0$, the asterisk denotes a complex conjugate, and $L = 1, 2, \cdots, m-1$. Since $W(0)$ is actually equal to $1 + i0$, the multiplication could be dropped from these equations. The product is included above, however, to show the tie between the operations of the first group and succeeding groups.

For all groups where $W(0)$ is not the multiplier, the following equations hold.

$$\begin{aligned} &\hat{B}_L\left(k_{m-1}, \cdots, k_{m-L+1}, \begin{pmatrix} 0,0 \\ 0,1 \end{pmatrix}, k_{m-L-2}, \cdots, k_0\right) \\ &\quad = \hat{B}_{L-1}\left(k_{m-1}, \cdots, k_{m-L+1}, \begin{pmatrix} 0,0 \\ 1,0 \end{pmatrix}, k_{m-L-2}, \cdots, k_0\right) \\ &\quad + \hat{B}_{L-1}\left(k_{m-1}, \cdots, k_{m-L+1}, \begin{pmatrix} 0,1 \\ 1,1 \end{pmatrix}, k_{m-L-2}, \cdots, k_0\right) \\ &\quad \cdot W(k_{m-1}, \cdots, k_{m-L+1}) \end{aligned} \tag{6}$$

and

$$\begin{aligned} &\hat{B}_L\left(k_{m-1}, \cdots, k_{m-L+1}, \begin{pmatrix} 1,0 \\ 1,1 \end{pmatrix}, k_{m-L-2}, \cdots, k_0\right) \\ &\quad = \hat{B}^*_{L-1}\left(k_{m-1}, \cdots, k_{m-L+1}, \begin{pmatrix} 0,0 \\ 1,0 \end{pmatrix}, k_{m-L-2}, \cdots, k_0\right) \\ &\quad - \hat{B}^*_{L-1}\left(k_{m-1}, \cdots, k_{m-L+1}, \begin{pmatrix} 0,1 \\ 1,1 \end{pmatrix}, k_{m-L-2}, \cdots, k_0\right) \\ &\quad \cdot W^*(k_{m-1}, \cdots, k_{m-L+1}). \end{aligned} \tag{7}$$

These equations hold for $L = 1, 2, \cdots, m-1$. The Fourier transformation is completed by replacing $B_{m-1}(0)$ and $B_{m-1}(1)$ with their sum and difference, respectively. When the highest frequency computed is known to be zero, an equivalent operation would be to double $B_{m-1}(0)$ and set $B_{m-1}(1) = 0$. However, for a one-sided transform even the doubling could be eliminated, resulting in a common normalizing factor of $N/2$ for all of the Fourier coefficients.

With the exception of $B_{m-1}(0)$ and $B_{m-1}(1)$, the $B_{m-1}(k_{m-1}, \cdots, k_2, k_1, \binom{0}{1})$ terms represent the complex Fourier coefficients of the original B_0 series. The $B_{m-1}(0)$ term is a real number representing the dc term and the $B_{m-1}(1)$ term represents the $N/2$-th harmonic. As in the case of the Cooley-Tukey results, these coefficients are not in order of ascending frequency, so reordering is required. Methods of performing this reordering are discussed later.

Note that the argument of each W term refers to a member of a sequence of stored or computed powers of W and it does not represent an exponent. For example, the expressions $W(0)$, $W(1)$, $W(2)$, and $W(3)$ refer to W^0, $W^{-N/8}$, $W^{-N/16}$, and $W^{-3N/16}$ respectively. As this progression is continued, the equations call upon $N/4$ complex exponential weights. With one additional observation, it is possible to reduce this table to $N/8$ complex exponential weights plus the term $1 + i0$. Note that the last two values of W called in Figure 5 were W^{-1} and W^{-3}. Since, for $N = 16$ we have $W^{-3} = -i(W^{-1*})$, one could simply negate and interchange the real and imaginary parts of the W^{-1} to obtain W^{-3}. This property generalizes conveniently such that powers of W of greater magnitude than $N/8$ can always be obtained by simply negating and interchanging the real and imaginary parts of the preceding W. This operation can be performed quite conveniently in either software or hardware realizations of the algorithm. Thus it is convenient to perform the Fourier transform using only N/8 powers of W.

When an algorithm using positive powers of W is used, the use of $N/8$ complex weights is made even simpler. In this case, starting with the fourth group in any iteration, the powers of W required for the even numbered groups can be formed by simply interchanging the real and imaginary parts of the W used in the preceding group. The only disadvantage of the algorithm using the positive powers of W is that the same "Complex Calculation" cannot be used for all of the arithmetic operations without a special modification for the first group. Note that when the negative powers of W were used in Figure 5, the same arithmetic unit was used without modification in all of the $m - 1$ iterations.

SEQUENCE OF				
2 TERMS	4 TERMS	8 TERMS	16 TERMS	32 TERMS
0	0	0	0	0
				16
			8	8
				24
		4	4	4
				28
			12	12
				20
	2	2	2	2
				30
			14	14
				18
		6	6	6
				26
			10	10
				22
1	1	1	1	1
				31
			15	15
				17
		7	7	7
				25
			9	9
				23
	3	3	3	3
				29
			13	13
				19
		5	5	5
				27
			11	11
				21

FIG. 6. A method of generating the sequence of W exponents

Complex Exponential Weight Table

The Fourier analysis recursive equations presuppose a table of complex exponential weights which can be accessed sequentially in performing each iteration of the algorithm, or they presuppose a method of sequentially computing these weights. One way of generating the number sequence required for forming a W table is shown schematically in Figure 6. For a given value of N, the sequence can be progressively doubled until the $N/4$ or $N/8$ exponents required have been generated.

The algorithm for doubling the length of each number sequence is: (1) multiply the second entry of the sequence by two and make this product the second entry of the new sequence; (2) subtract each nonzero entry of the sequence from twice the product formed in step 1 (these differences form the rest of the even entries of the new sequence); (3) take the odd entries of the new sequence as the numbers of the original sequence.

Once the required sequence of W exponents is found, the corresponding W terms can be found and stored in this "scrambled" order. This table, computed for $N = 2^m$, is called sequentially during each iteration and can be used without change when $N = 2^q$ where $q \leqq m$.

An alternative approach consists of simply generating the sequence of W terms directly, without the intervening step of generating the sequence of exponents. Since only additions and subtractions are required in generating the exponents, the W terms can be generated directly as a sequence of products. In fact the entire $N/4$ term W table can be generated by using the $\log_2 (N)$ term sequence W^0, W^1, W^2, W^4, $\cdots$, $W^{N/4}$ and the expression $W^p W^q = W^{p+q}$.

Reordering

In a hardware implementation of the algorithm, the output of a binary counter can be suitably modified to read

the resulting Fourier coefficients out of the N real locations in order of ascending frequency. Although in-place reordering can be performed, it has not proved to be very efficient for the form of the algorithm discussed here.

In a hardware indexing unit, the address sequence for reordering requires performing bitwise tests on the "flipped" binary equivalent of each harmonic number, from the least significant bit to, but not including, the leftmost "one." As this is being done, each bit tested is replaced by a "one" if the state sought matches the state found, and by a "zero" otherwise. The state sought continues to alternate between "one" and "zero." It starts as a "one" and changes after each match is found.

Although the same number sequence can be generated with software, thus far it has been more efficient to store part of the number sequence of Figure 6 in a table and read the scrambled Fourier coefficients into a second array reordered.

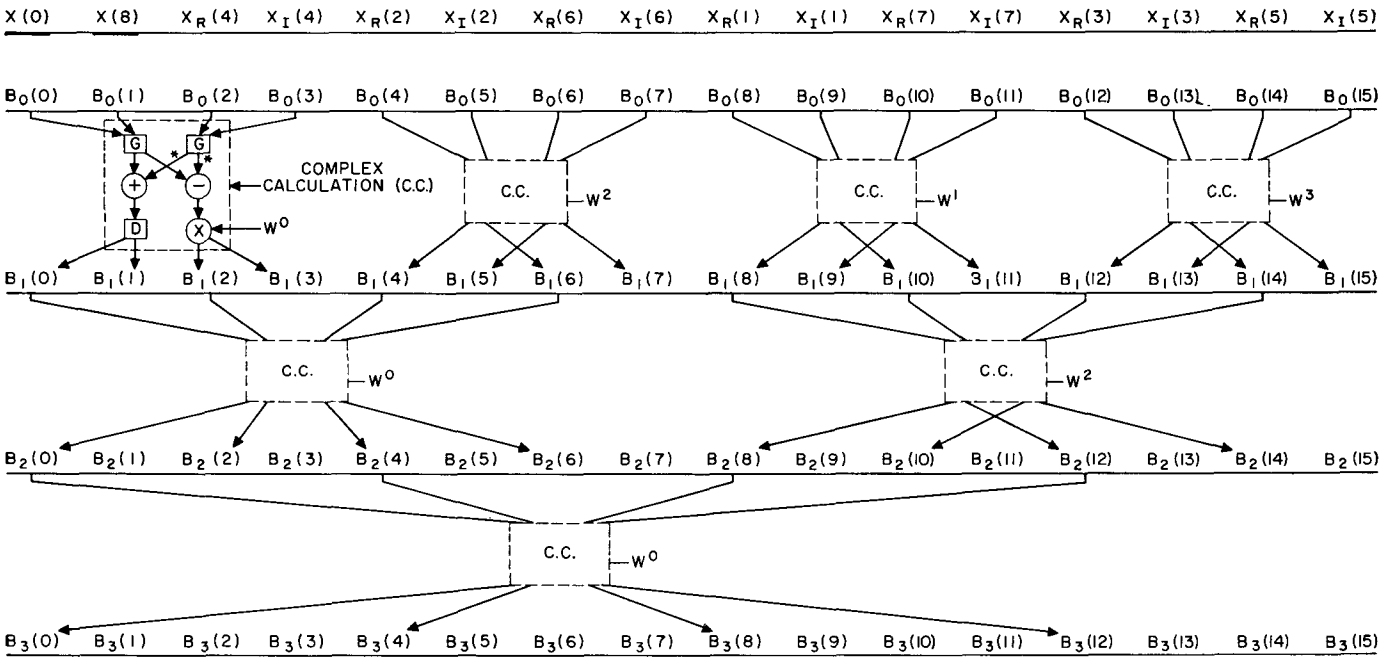

FIG. 7. The real-valued input fast Fourier synthesis algorithm for $N = 16$

Extensions

Several variations of the real-valued input algorithm have been considered and are described briefly below.

By reordering the original real-valued series before performing the Fourier analysis, an algorithm should result in which the powers of W are required in ascending order and therefore can be conveniently computed recursively. One of the recursive forms of DeMoivre's formula described by Singleton [11] may be sufficiently accurate for a software implementation.

One can "unwrap" the operations of the real-valued Fourier analysis algorithm to obtain a Fourier synthesis algorithm which has the same properties. It is convenient to start with the Fourier coefficients in the order in which the Fourier analysis algorithm would have computed them. Thus by starting with N/2 − 1 complex and 2 real numbers in a scrambled form, the N real term series can be computed in the correct order. The form of the Fourier synthesis algorithm, for the example of $N = 16$, is shown in Figure 7. Note that the "Complex Calculation" consists of the operations shown in Figure 5 done in reversed order. This algorithm is related to the Fourier analysis algorithm in much the same way that the Sande-Tukey algorithm [19] is related to the Cooley-Tukey algorithm.

A variation of the real-valued input algorithm has been developed which alters the regular structure of the indexing, but makes reordering more convenient and allows more convenient extensions to radix-4, radix-8, and arbitrary-radix algorithms.

In forming an arbitrary-radix, real-valued algorithm, the recursive equations when N is the product of an arbitrary set of integers [20] are also interpreted as forming sets of unnormalized Fourier coefficients. If N is represented as the product $r_1 r_2 \cdots r_n$, the first iteration consists of computing $r_2 r_3 \cdots r_n$ sets of r_1 term Fourier series.

The second iteration consists of computing $r_3r_4 \cdots r_n$ sets of r_1r_2 term Fourier series. Thus the Lth iteration consists of computing $r_{L+1}r_{L+2} \cdots r_n$ sets of $r_1r_2 \cdots r_L$ term Fourier series.

When the original time series is real, the symmetry about the folding frequency will still always exist and complex numbers will always appear in conjugate pairs. The only departure from the diagram will be for cases where the product $r_1r_2 \cdots r_L$ is an odd number. In this case the folding frequency is not one of the coefficients which is computed, so each set will consist of only one real number (the dc term) together with $(r_1r_2 \cdots r_L)/2$ complex conjugate pairs.

As in the radix-2 algorithm discussed in this paper, it is again convenient to compute and store only those coefficients at or below the folding frequency. The storing procedure, however, is changed so that the imaginary part of each saved intermediate result is stored in the relative location which would ordinarily have held its redundant complex conjugate. This results in an orderly procedure which extends conveniently to radix-4, radix-8, and arbitrary-radix algorithms. It also makes inplace reordering more manageable.

Results

The Fourier analysis algorithm discussed in this paper requires essentially one half the arithmetic operations of the original complex Cooley-Tukey radix-2 algorithm. For $N = 2^m$ the number of operations required by the Cooley-Tukey radix-2 algorithm and the Fourier-transform-real-valued-input (FTRVI) algorithm are given in Table I.

TABLE I. Arithmetic Operations Required for $N = 2^m$

Algorithm	*Real multiplications*	*Real additions*
FTRVI	$(m - 3.5)N + 6$	$(1.5m - 2.5)N + 4$
Radix 2	$(2m - 7)N + 12$	$(3m - 3)N + 4$

The expressions in Table I assume that rereferencing factors (or twiddle factors) of $\exp(i0)$, $\exp(\pm i\pi/2)$ and $\pm\exp(\pm i\pi/4)$ are treated as special cases. Also, in the FTRVI algorithm, the normalizing factor for everything but the dc term and the folding frequency is $N/2$ instead of N. If these Fourier coefficients were doubled, this would change the number of real additions in FTRVI to $(1.5m-1.5)N + 2$ which is exactly half of the corresponding number for the complex radix-2 algorithm. Since this doubling is usually not necessary, the number of additions is shown as slightly less than half.

The savings effected by radix-4 and radix-8 complex input algorithms [10, 13] can also be made in the real-valued input algorithm. Thus a further computational reduction of approximately 30 percent is anticipated.

In terms of hardware implementations, the execution times and memory requirements of the fast Fourier transform processor described by Shively [7] can be cut in half.

Received April, 1968; revised April, 1968

REFERENCES

1. Cooley, J. W., and Tukey, J. W. An algorithm for the machine calculation of complex Fourier series. *Math. Comput. 19* (Apr. 1965), 297–301.
2. McKinney, T. H. A digital spectrum channel analyzer. Conference on Speech Communication and Processing Preprints, Nov. 1967, pp. 442–444.
3. Freudberg, R., DeLellis, J., Howard, C., and Schaffer, H. An all digital pitch excited vocoder technique using the FFT algorithm. Conference on Speech Communication and Processing Preprints, Nov. 1967, pp. 297–310.
4. Helms, H. D. Fast Fourier transform method of computing difference equations and simulating filters. *IEEE Trans. AU-15* (June 1967), 85–90.
5. Singleton, R. C., and Poulter, T. C. Spectral analysis of the call of the male killer whale. *IEEE Trans. AU* (June 1967), 104–113.
6. Dumermuth, G., and Fluhler, H. Some modern aspects in numerical spectrum analysis of multichannel electroencephalographic data. *Med. and Biol. Eng. 5* (1967), 319–331.
7. Shively, R. R. A digital processor to perform a fast Fourier transform. Proc. First Ann. IEEE Comput. Conf., Sept. 1967, pp. 21–24.
8. Bergland, G. D., and Hale, H. W. Digital real-time spectral analysis. *IEEE Trans. EC-16*, (Apr. 1967), 180–185.
9. Van Blerkom, R. IBM Federal Systems Center. (Private communication)
10. Gentleman, W. M., and Sande, G. Fast Fourier transforms —for fun and profit. Proc. AFIPS 1966 Fall Joint Comput. Conf., Vol. 29. Spartan Books, New York, pp. 563–578.
11. Singleton, R. C. On computing the fast Fourier transform. *Comm. ACM 10* (Oct. 1967), 647–654.
12. Cooley, J. W. Complex finite Fourier transform subroutine. SHARE Doc. 3465, Sept. 8, 1966.
13. Bergland, G. D. A fast Fourier transform algorithm using base 8 iterations. *Math. Comput., 22* (Apr. 1968), 275–279.
14. Brenner, N. M. Three Fortran programs that perform the Cooley-Tukey Fourier transform. Tech. Note 1967-2, Lincoln Lab., MIT, July 1967.
15. Cooley, J. W., Lewis, P. A. W., and Welch, P. D. The fast Fourier transform algorithm and its applications. IBM Res. Paper RC-1743, Feb. 1967.
16. Edson, J. O. Bell Telephone Laboratories. (Private communication)
17. Cooley, J. W. Applications of the fast Fourier transform method. Proc. IBM Scientific Computing Symposium, Thomas J. Watson Research Center, Yorktown Heights, N.Y., June 1966.
18. G-AE Subcommittee on Measurement Concepts. What is the fast Fourier transform? *IEEE Trans. AU-15* (June 1967), 45–55.
19. Bingham, C., Godfrey, M. D., and Tukey, J. W. Modern techniques of power spectrum estimation. *IEEE Trans. AU-15* (June 1967), 56–66.
20. Bergland, G. D. The fast Fourier transform recursive equations for arbitrary length records. *Math. Comput. 21* (Apr. 1967), 236–238.