An Algorithm for the Machine Calculation of Complex Fourier Series

By James W. Cooley and John W. Tukey

An efficient method for the calculation of the interactions of a 2^m factorial experiment was introduced by Yates and is widely known by his name. The generalization to 3^m was given by Box et al. [1]. Good [2] generalized these methods and gave elegant algorithms for which one class of applications is the calculation of Fourier series. In their full generality, Good's methods are applicable to certain problems in which one must multiply an N-vector by an $N \times N$ matrix which can be factored into m sparse matrices, where m is proportional to $\log N$. This results in a procedure requiring a number of operations proportional to $N \log N$ rather than N^2. These methods are applied here to the calculation of complex Fourier series. They are useful in situations where the number of data points is, or can be chosen to be, a highly composite number. The algorithm is here derived and presented in a rather different form. Attention is given to the choice of N. It is also shown how special advantage can be obtained in the use of a binary computer with $N = 2^m$ and how the entire calculation can be performed within the array of N data storage locations used for the given Fourier coefficients.

Consider the problem of calculating the complex Fourier series

$$X(j) = \sum_{k=0}^{N-1} A(k) \cdot W^{jk}, \qquad j = 0, 1, \cdots, N-1, \tag{1}$$

where the given Fourier coefficients $A(k)$ are complex and W is the principal Nth root of unity,

$$W = e^{2\pi i/N}. \tag{2}$$

A straightforward calculation using (1) would require N^2 operations where "operation" means, as it will throughout this note, a complex multiplication followed by a complex addition.

The algorithm described here iterates on the array of given complex Fourier amplitudes and yields the result in less than $2N \log_2 N$ operations without requiring more data storage than is required for the given array A. To derive the algorithm, suppose N is a composite, i.e., $N = r_1 \cdot r_2$. Then let the indices in (1) be expressed

$$\begin{aligned} j &= j_1 r_1 + j_0, \qquad j_0 = 0, 1, \cdots, r_1 - 1, \qquad j_1 = 0, 1, \cdots, r_2 - 1, \\ k &= k_1 r_2 + k_0, \qquad k_0 = 0, 1, \cdots, r_2 - 1, \qquad k_1 = 0, 1, \cdots, r_1 - 1. \end{aligned} \tag{3}$$

Then, one can write

$$X(j_1, j_0) = \sum_{k_0} \sum_{k_1} A(k_1, k_0) \cdot W^{jk_1 r_2} W^{jk_0}. \tag{4}$$

Received August 17, 1964. Research in part at Princeton University under the sponsorship of the Army Research Office (Durham). The authors wish to thank Richard Garwin for his essential role in communication and encouragement.

Since

$$W^{jk_1r_2} = W^{j_0k_1r_2}, \tag{5}$$

the inner sum, over k_1, depends only on j_0 and k_0 and can be defined as a new array,

$$A_1(j_0, k_0) = \sum_{k_1} A(k_1, k_0) \cdot W^{j_0k_1r_2}. \tag{6}$$

The result can then be written

$$X(j_1, j_0) = \sum_{k_0} A_1(j_0, k_0) \cdot W^{(j_1r_1+j_0)k_0}. \tag{7}$$

There are N elements in the array A_1, each requiring r_1 operations, giving a total of Nr_1 operations to obtain A_1. Similarly, it takes Nr_2 operations to calculate X from A_1. Therefore, this two-step algorithm, given by (6) and (7), requires a total of

$$T = N(r_1 + r_2) \tag{8}$$

operations.

It is easy to see how successive applications of the above procedure, starting with its application to (6), give an m-step algorithm requiring

$$T = N(r_1 + r_2 + \cdots + r_m) \tag{9}$$

operations, where

$$N = r_1 \cdot r_2 \cdots r_m . \tag{10}$$

If $r_j = s_jt_j$ with $s_j, t_j > 1$, then $s_j + t_j < r_j$ unless $s_j = t_j = 2$, when $s_j + t_j = r_j$. In general, then, using as many factors as possible provides a minimum to (9), but factors of 2 can be combined in pairs without loss. If we are able to choose N to be highly composite, we may make very real gains. If all r_j are equal to r, then, from (10) we have

$$m = \log_r N \tag{11}$$

and the total number of operations is

$$T(r) = rN \log_r N. \tag{12}$$

If $N = r^m s^n t^p \cdots$, then we find that

$$\begin{aligned} \frac{T}{N} &= m \cdot r + n \cdot s + p \cdot t + \cdots , \\ \log_2 N &= m \cdot \log_2 r + n \cdot \log_2 s + p \cdot \log_2 t + \cdots , \end{aligned} \tag{13}$$

so that

$$\frac{T}{N \log_2 N}$$

is a weighted mean of the quantities

$$\frac{r}{\log_2 r}, \quad \frac{s}{\log_2 s}, \quad \frac{t}{\log_2 t}, \cdots ,$$

whose values run as follows

r	$\frac{r}{\log_2 r}$
2	2.00
3	1.88
4	2.00
5	2.15
6	2.31
7	2.49
8	2.67
9	2.82
10	3.01.

The use of $r_j = 3$ is formally most efficient, but the gain is only about 6% over the use of 2 or 4, which have other advantages. If necessary, the use of r_j up to 10 can increase the number of computations by no more than 50%. Accordingly, we can find "highly composite" values of N within a few percent of any given large number.

Whenever possible, the use of $N = r^m$ with $r = 2$ or 4 offers important advantages for computers with binary arithmetic, both in addressing and in multiplication economy.

The algorithm with $r = 2$ is derived by expressing the indices in the form

$$\begin{aligned} j &= j_{m-1}\cdot 2^{m-1} + \cdots + j_1\cdot 2 + j_0, \\ k &= k_{m-1}\cdot 2^{m-1} + \cdots + k_1\cdot 2 + k_0, \end{aligned} \tag{14}$$

where j_v and k_v are equal to 0 or 1 and are the contents of the respective bit positions in the binary representation of j and k. All arrays will now be written as functions of the bits of their indices. With this convention (1) is written

$$X(j_{m-1}, \cdots, j_0) = \sum_{k_0}\sum_{k_1}\cdots\sum_{k_{m-1}} A(k_{m-1}, \cdots, k_0)\cdot W^{jk_{m-1}\cdot 2^{m-1}+\cdots+jk_0}, \tag{15}$$

where the sums are over $k_v = 0, 1$. Since

$$W^{jk_{m-1}\cdot 2^{m-1}} = W^{j_0 k_{m-1}\cdot 2^{m-1}}, \tag{16}$$

the innermost sum of (15), over k_{m-1}, depends only on $j_0, k_{m-2}, \cdots, k_0$ and can be written

$$A_1(j_0, k_{m-2}, \cdots, k_0) = \sum_{k_{m-1}} A(k_{m-1}, \cdots, k_0)\cdot W^{j_0 k_{m-1}\cdot 2^{m-1}}. \tag{17}$$

Proceeding to the next innermost sum, over k_{m-2}, and so on, and using

$$W^{j\cdot k_{m-l}\cdot 2^{m-l}} = W^{(j_{l-1}\cdot 2^{l-1}+\cdots+j_0)k_{m-l}\cdot 2^{m-l}}, \tag{18}$$

one obtains successive arrays,

$$\begin{aligned} &A_l(j_0, \cdots, j_{l-1}, k_{m-l-1}, \cdots, k_0) \\ &\quad = \sum_{k_{m-l}} A_{l-1}(j_0, \cdots, j_{l-2}, k_{m-l}, \cdots, k_0)\cdot W^{(j_{l-1}\cdot 2^{l-1}+\cdots+j_0)\cdot k_{m-l}\cdot 2^{m-l}} \end{aligned} \tag{19}$$

for $l = 1, 2, \cdots, m$.

Writing out the sum this appears as

$$
\begin{aligned}
&A_l(j_0, \cdots, j_{l-1}, k_{m-l-1}, \cdots, k_0) \\
&\qquad = A_{l-1}(j_0, \cdots, j_{l-2}, 0, k_{m-l-1}, \cdots, k_0) \\
&\qquad\quad + (-1)^{j_{l-1}} i^{j_{l-2}} A_{l-1}(j_0, \cdots, j_{l-2}, 1, k_{m-l-1}, \cdots, k_0) \\
&\qquad\quad \cdot W^{(j_{l-3}\cdot 2^{l-3} + \cdots + j_0)\cdot 2^{m-l}}, \qquad j_{l-1} = 0, 1.
\end{aligned} \tag{20}
$$

According to the indexing convention, this is stored in a location whose index is

$$j_0 \cdot 2^{m-1} + \cdots + j_{l-1} \cdot 2^{m-l} + k_{m-l-1} \cdot 2^{m-l-1} + \cdots + k_0 . \tag{21}$$

It can be seen in (20) that only the two storage locations with indices having 0 and 1 in the 2^{m-l} bit position are involved in the computation. Parallel computation is permitted since the operation described by (20) can be carried out with all values of $j_0, \cdots, j_{l-2}$, and $k_0, \cdots, k_{m-l-1}$ simultaneously. In some applications* it is convenient to use (20) to express A_l in terms of A_{l-2}, giving what is equivalent to an algorithm with $r = 4$.

The last array calculated gives the desired Fourier sums,

$$X(j_{m-1}, \cdots, j_0) = A_m(j_0, \cdots, j_{m-1}) \tag{22}$$

in such an order that the index of an X must have its binary bits put in reverse order to yield its index in the array A_m.

In some applications, where Fourier sums are to be evaluated twice, the above procedure could be programmed so that no bit-inversion is necessary. For example, consider the solution of the difference equation,

$$aX(j+1) + bX(j) + cX(j-1) = F(j). \tag{23}$$

The present method could be first applied to calculate the Fourier amplitudes of $F(j)$ from the formula

$$B(k) = \frac{1}{N} \sum_j F(j) W^{-jk}. \tag{24}$$

The Fourier amplitudes of the solution are, then,

$$A(k) = \frac{B(k)}{aW^k + b + cW^{-k}}. \tag{25}$$

The $B(k)$ and $A(k)$ arrays are in bit-inverted order, but with an obvious modification of (20), $A(k)$ can be used to yield the solution with correct indexing.

A computer program for the IBM 7094 has been written which calculates three-dimensional Fourier sums by the above method. The computing time taken for computing three-dimensional $2^a \times 2^b \times 2^c$ arrays of data points was as follows:

---

* A multiple-processing circuit using this algorithm was designed by R. E. Miller and S. Winograd of the IBM Watson Research Center. In this case $r = 4$ was found to be most practical.

a	b	c	No. Pts.	Time (minutes)
4	4	3	2^{11}	.02
11	0	0	2^{11}	.02
4	4	4	2^{12}	.04
12	0	0	2^{12}	.07
5	4	4	2^{13}	.10
5	5	3	2^{13}	.12
13	0	0	2^{13}	.13

IBM Watson Research Center
Yorktown Heights, New York

Bell Telephone Laboratories,
Murray Hill, New Jersey

Princeton University
Princeton, New Jersey

1. G. E. P. Box, L. R. Connor, W. R. Cousins, O. L. Davies (Ed.), F. R. Hirnsworth & G. P. Silitto, *The Design and Analysis of Industrial Experiments*, Oliver & Boyd, Edinburgh, 1954.

→ I. J. Good, "The interaction algorithm and practical Fourier series," *J. Roy. Statist. Soc. Ser. B.*, v. 20, 1958, p. 361–372; Addendum, v. 22, 1960, p. 372–375. MR **21** #1674; MR **23** #A4231.