

**FR-7041**

**Fortran Program for Fast Fourier  
Transform**

**James R. Fisher  
Signal Processing Branch  
Acoustics Division**

**April 16, 1970**

## DOCUMENT CONTROL DATA - R &amp; D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Naval Research Laboratory Washington, D.C. 20390		2a. REPORT SECURITY CLASSIFICATION Unclassified	
		2b. GROUP	
3. REPORT TITLE FORTRAN PROGRAM FOR FAST FOURIER TRANSFORM			
4. DESCRIPTIVE NOTES (Type of report and inclusive dates) An interim report on a continuing NRL Problem			
5. AUTHOR(S) (First name, middle initial, last name) James R. Fisher			
6. REPORT DATE April 16, 1970		7a. TOTAL NO. OF PAGES 24	7b. NO. OF REFS 4
8a. CONTRACT OR GRANT NO. NRL Problem S01-39		9a. ORIGINATOR'S REPORT NUMBER(S) NRL Report 7041	
b. PROJECT NO. RF 05-552-403-4069		9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	
c.			
d.			
10. DISTRIBUTION STATEMENT This document has been approved for public release and sale; its distribution is unlimited.			
11. SUPPLEMENTARY NOTES		12. SPONSORING MILITARY ACTIVITY Department of the Navy (Office of Naval Research) Washington, D.C. 20360	
13. ABSTRACT The recent development of algorithms for the rapid computation of Fourier transforms has reduced the computation time of this powerful analysis tool by orders of magnitude, enabling previously uneconomic procedures to become commonplace.  In this report the fast Fourier transform (FFT) is derived from the basic equations and presented in matrix form as a means of illustrating the stage-by-stage reduction of the input data to Fourier coefficients by the algorithm.  Based on this development a Fortran IV program is presented, including a full description of the statements, by relating it to the theoretical requirements. Thus a complete understanding of the FFT algorithm and program can be obtained, eliminating the constraints imposed by treating the FFT as a black box beyond the manipulative powers of the user.			

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Fourier transformation Fourier series Computer programming Computer programs Fast Fourier transforms Periodic functions Signal processing Fortran Algorithms						

## CONTENTS

Abstract	ii
Problem Status	ii
Authorization	ii
<b>THEORETICAL DEVELOPMENT</b>	<b>1</b>
Introduction	1
Discrete Fourier Series	1
Decimation in Frequency	3
Coefficient Scrambling	4
Continued Reduction	5
Total Computations	8
Flow Chart	8
Coefficient Scrambling	10
<b>FORTRAN IV PROGRAM FOR FFT</b>	<b>12</b>
General	12
Subroutine FFOUR	13
Subroutine GNSCR and UNSCR	16
Subroutine COSINE	19
<b>ACKNOWLEDGMENT</b>	<b>19</b>
<b>REFERENCES</b>	<b>20</b>

## ABSTRACT

The recent development of algorithms for the rapid computation of Fourier transforms has reduced the computation time of this powerful analysis tool by orders of magnitude, enabling previously uneconomic procedures to become commonplace.

In this report the fast Fourier transform (FFT) is derived from the basic equations and presented in matrix form as a means of illustrating the stage-by-stage reduction of the input data to Fourier coefficients by the algorithm.

Based on this development a Fortran IV program is presented, including a full description of the statements, by relating it to the theoretical requirements. Thus a complete understanding of the FFT algorithm and program can be obtained, eliminating the constraints imposed by treating the FFT as a black box beyond the manipulative powers of the user.

## PROBLEM STATUS

This is an interim report on a continuing NRL Problem.

## AUTHORIZATION

NRL Problem S01-39  
Project RF 05-552-403-4069

Manuscript received December 17, 1969.

# FORTRAN PROGRAM FOR FAST FOURIER TRANSFORM

## THEORETICAL DEVELOPMENT

### Introduction

The Fourier series is a powerful tool for the analysis of periodic phenomena enabling conversion of a time related function to a more easily analyzed function of frequency. The Fourier transform can serve as the kernel of computation for commonly used processes in signal analysis such as algorithms for digital filtering, beam forming, and power spectrum determination.

The fast Fourier transform permits the reduction of redundancy in the well-known discrete Fourier transform by a factor of  $N^2/N \log_2 N$  for a data sample length  $N$  equal to a power of 2, thus permitting a substantial reduction in computation time and permitting analysis of previously uneconomic processes.

The program presented here was developed for use in the Naval Research Laboratory and utilized "decimation in frequency" (1), a variation of the familiar Cooley-Tukey algorithm (2).

Although prepackaged fast Fourier transform (FFT) packages are available from software firms, such programs are presented as black boxes, giving the programmer no understanding of the inner workings of the computations. Such a situation imposes limitations during the construction of new programs requiring subtle modifications of the FFT. It is the purpose of this report, therefore, to describe the FFT by associating its mathematical derivation with the actual computer program, complete with explanations of the Fortran IV statements.

### Discrete Fourier Series

A discrete finite series  $X_k$  resulting from sampling of a continuous function of time can be represented as a finite sum of its frequency components (3) and expressed as

$$X_k = \sum_{r=0}^{N-1} A_r \exp(2\pi jrk/N), \quad k = 0, 1, \dots, N-1, \quad (1)$$

where  $A_r$  represents the coefficients of a Fourier series and

$$A_r = \frac{1}{N} \sum_{k=0}^{N-1} X_k \exp(-2\pi jrk/N), \quad r = 0, 1, \dots, N-1, \quad (2)$$

with  $X_k$  consisting of the sample points of the discrete time series.

For simplicity of presentation of the FFT in matrix equation form, the factor  $1/N$  outside the summation sign in Eq. (2) will not be included in the following development, and the notation

$$W^n = \exp(-2\pi jn/N) \quad (3)$$

will be used in place of the complex exponential of Eq. (2) when matrix equations are presented. Thus Eq. (2) becomes

$$A_r = \sum_{k=0}^{N-1} X_k W^{rk}, \quad r = 0, 1, \dots, N-1. \quad (4)$$

The representation of Eq. (4) would be the square matrix in Fig. 1, whose elements are complex.

$$[A_r] = \begin{bmatrix} W^0 & W^0 & \dots & W^0 \\ W^0 & W^1 & \dots & W^{N-1} \\ \dots & \dots & \dots & \dots \\ W^0 & W^r & \dots & W^{r(N-1)} \\ \dots & \dots & \dots & \dots \\ W^0 & W^{N-1} & \dots & W^{(N-1)(N-1)} \end{bmatrix} \begin{bmatrix} X_0 \\ \dots \\ \dots \\ \dots \\ \dots \\ X_{N-1} \end{bmatrix}$$

Fig. 1 - Matrix form of the discrete Fourier transform

As an aid to the understanding of the arithmetic manipulations involved in the FFT, the process will be evaluated for an eight-point arbitrarily sampled time function, i.e.,  $N = 8$ . Consequently Fig. 1 is reduced to the eight-by-eight matrix of Fig. 2. Reduction of Fig. 2 to Fourier coefficients would require  $N^2 = 8^2 = 64$  complex multiplications and  $N(N-1) = 56$  complex additions.

$$[A_r] = \begin{bmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \\ A_4 \\ A_5 \\ A_6 \\ A_7 \end{bmatrix} = \begin{bmatrix} W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 & W^4 & W^5 & W^6 & W^7 \\ W^0 & W^2 & W^4 & W^6 & W^8 & W^{10} & W^{12} & W^{14} \\ W^0 & W^3 & W^6 & W^9 & W^{12} & W^{15} & W^{18} & W^{21} \\ W^0 & W^4 & W^8 & W^{12} & W^{16} & W^{20} & W^{24} & W^{28} \\ W^0 & W^5 & W^{10} & W^{15} & W^{20} & W^{25} & W^{30} & W^{35} \\ W^0 & W^6 & W^{12} & W^{18} & W^{24} & W^{30} & W^{36} & W^{42} \\ W^0 & W^7 & W^{14} & W^{21} & W^{28} & W^{35} & W^{42} & W^{49} \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \end{bmatrix}$$

Fig. 2 - Matrix form of the discrete Fourier transform for the case  $N = 8$

### Decimation in Frequency

By suitable pairing of Fourier coefficients it will be possible to reduce the number of computations required by the discrete Fourier transform. From the basic Eq. (4) we have

$$A_r = \sum_{k=0}^{N-1} X_k \exp(-2\pi j rk/N), \quad r = 0, 1, \dots, N-1. \quad (5)$$

It is to be assumed in this step and in those that follow that  $N$  is restricted to a power of 2, i.e.,  $N = 2^n$ , where  $n$  is any integer. Thus Eq. (5) may then be rewritten by halving the index and using two terms under the summation sign:

$$A_r = \sum_{k=0}^{N/2-1} (X_k \exp(-2\pi j rk/N) + X_{k+(N/2)} \exp(-2\pi j r [k+(N/2)]/N)), \quad r = 0, 1, \dots, N-1. \quad (6)$$

This can be further simplified to

$$A_r = \sum_{k=0}^{N/2-1} [X_k + X_{k+(N/2)} \exp(-\pi j r)] \exp(-2\pi j rk/N), \quad r = 0, 1, \dots, N-1. \quad (7)$$

It is the next step that yields the reduction in computation. By pairing the coefficients obtainable from Eqs. (5) and (7), we will split the  $N$  by  $N$  matrix of Figs. 1 and 2 in such a way that the sum of the computations from the new and smaller matrices is less than that of the original matrix of dimension  $N$  by  $N$ . Proceeding, we pair the coefficients  $A_r$  of Eq. (7) into odd and even parts. For even values of the subscript of  $A$  in Eq. (7) we then have

$$A_{2r} = \sum_{k=0}^{N/2-1} [X_k + X_{k+(N/2)}] \exp(-4\pi j rk/N), \quad r = 0, 1, \dots, N/2-1. \quad (8)$$

In a similar manner the odd values are obtained:

$$A_{2r+1} = \sum_{k=0}^{N/2-1} [X_k - X_{k+(N/2)}] \exp[-2\pi j (2r+1) k/N] \quad (9)$$

$$= \sum_{k=0}^{N/2-1} [X_k \mp X_{k+(N/2)}] \exp(-2\pi j k/N) \exp(-4\pi j rk/N), \quad r = 0, 1, \dots, N/2-1. \quad (10)$$

Although at first sight it might appear that a simple equation has been reduced to two equations of increased complexity, an examination of the matrix equations indicate what progress has been made to reduce the number of computations required to evaluate the  $N$  coefficients of the Fourier series. The matrix representations of Eqs. (8) and (10) are shown in Fig. 3. Assuming that  $X_0$  is added to  $X_4$ ,  $X_1$  is added to  $X_5$ , etc., in the vector  $A_{2r}$  and that  $(X_0 - X_4) W^0$ ,  $(X_1 - X_5) W^1$ , etc., is computed for vector  $A_{2r+1}$ , then the operations for reduction of  $A_{2r}$  to Fourier coefficients would require 16 complex additions and 16 complex multiplications if  $N = 8$ . The reduction of  $A_{2r+1}$  to Fourier coefficients would require 16 complex additions and 20 complex multiplications, again assuming

$$\begin{aligned}
[A_{2r}] &= \begin{bmatrix} A_0 \\ A_2 \\ A_4 \\ A_6 \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^4 & W^8 & W^{12} \\ W^0 & W^6 & W^{12} & W^{18} \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} + \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^4 & W^8 & W^{12} \\ W^0 & W^6 & W^{12} & W^{18} \end{bmatrix} \begin{bmatrix} X_4 \\ X_5 \\ X_6 \\ X_7 \end{bmatrix} \\
&= \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^4 & W^8 & W^{12} \\ W^0 & W^6 & W^{12} & W^{18} \end{bmatrix} \begin{bmatrix} X_0 + X_4 \\ X_1 + X_5 \\ X_2 + X_6 \\ X_3 + X_7 \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^4 & W^8 & W^{12} \\ W^0 & W^6 & W^{12} & W^{18} \end{bmatrix} \begin{bmatrix} X'_0 \\ X'_1 \\ X'_2 \\ X'_3 \end{bmatrix} = [B_r] \\
[A_{2r+1}] &= \begin{bmatrix} A_1 \\ A_3 \\ A_5 \\ A_7 \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^4 & W^8 & W^{12} \\ W^0 & W^6 & W^{12} & W^{18} \end{bmatrix} \begin{bmatrix} X_0 \cdot W^0 \\ X_1 \cdot W^1 \\ X_2 \cdot W^2 \\ X_3 \cdot W^3 \end{bmatrix} + \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^4 & W^8 & W^{12} \\ W^0 & W^6 & W^{12} & W^{18} \end{bmatrix} \begin{bmatrix} X_4 \cdot W^0 \\ X_5 \cdot W^1 \\ X_6 \cdot W^2 \\ X_7 \cdot W^3 \end{bmatrix} \\
&= \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^4 & W^8 & W^{12} \\ W^0 & W^6 & W^{12} & W^{18} \end{bmatrix} \begin{bmatrix} X_0 \cdot W^0 - X_4 \cdot W^0 \\ X_1 \cdot W^1 - X_5 \cdot W^1 \\ X_2 \cdot W^2 - X_6 \cdot W^2 \\ X_3 \cdot W^3 - X_7 \cdot W^3 \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^4 & W^8 & W^{12} \\ W^0 & W^6 & W^{12} & W^{18} \end{bmatrix} \begin{bmatrix} X'_4 \\ X'_5 \\ X'_6 \\ X'_7 \end{bmatrix} = [C_r]
\end{aligned}$$

Fig. 3 - The first stage of the FFT matrix reduction for the case  $N = 8$ ; for this case,  $r = 0, 1, \dots, N/2 - 1$  becomes  $r = 0, 1, 2$

$N = 8$ .  $A_{2r+1}$  is identical in structure to  $A_{2r}$  except for the complex multipliers  $W^0, W^1, W^2, W^3$ ; thus the reduction of  $A_{2r+1}$  to Fourier coefficients requires four additional multiplications compared with the reduction of  $A_{2r}$ . The total number of operations for  $A_{2r}$  and  $A_{2r+1}$  is then 32 complex additions and 36 complex multiplications. This compares with 56 complex additions and 64 complex multiplications required for solution of the  $N$  by  $N$  matrix, as previously mentioned.

### Coefficient Scrambling

An additional consequence of the FFT process should be noted at this time and must be incorporated into the final algorithm. If Eqs. (8) and (10) were processed as they now stand, the Fourier coefficients  $A_r$  would be obtained, but they would not be in the order  $r = 0, 1, 2, \dots, N - 1$  as originally suggested in Eq. (2). Instead the coefficients would be computed from the even and odd subequations (8) and (10) in the order  $r = 0, 2, 4, \dots, N - 2, 1, 3, 5, \dots, N - 1$ . This scrambling of the coefficients will continue in subsequent stages of the computation and must be remedied at the conclusion of the algorithm when the coefficients are desired in increasing order from zero through  $N - 1$ . Coefficient unscrambling will be further discussed following complete reduction of the FFT.

## Continued Reduction

Instead of computing the Fourier coefficients as suggested, the obvious next step is to reduce the  $N/2$  by  $N/2$  matrices in Fig. 3 by proceeding in a manner similar to the first stage when we progressed from Fig. 2 to Fig. 3. In preparation for this, we perform the sums indicated for  $A_{2r}$  and the differences multiplied by the constants indicated for  $A_{2r+1}$ . The new "data" values obtained will be labeled respectively  $X'_0, X'_1, X'_2, \dots$ , and for notational purposes the matrix operations indicated on the right side of  $A_{2r}$  and  $A_{2r+1}$  in Fig. 3 will be denoted by  $B_r$  and  $C_r$  respectively.

With these new data values we are now in a position to perform the next stage of computation. For the simplified case of  $N = 8$ , this will reduce  $B_r$  and  $C_r$  into two by two matrix operations. Using the notation described,

$$B_r = \sum_{k=0}^{N/2-1} X'_k \exp(-4\pi jrk/N), \quad r = 0, 1, \dots, N/2 - 1 \quad (11)$$

$$= \sum_{k=0}^{N/4-1} \{X'_k \exp(-4\pi jrk/N) + X'_{k+N/4} \exp[-4\pi jr(k+N/4)/N]\}, \quad r = 0, 1, \dots, N/2-1 \quad (12)$$

$$= \sum_{k=0}^{N/4-1} [X'_k + X'_{k+N/4} \exp(-\pi jr)] \exp(-4\pi jrk/N), \quad r = 0, 1, \dots, N/2-1, \quad (13)$$

$$B_{2r} = \sum_{k=0}^{N/4-1} [X'_k + X'_{k+N/4}] \exp(-8\pi jrk/N), \quad r = 0, 1, \dots, N/4-1, \quad (14)$$

$$B_{2r+1} = \sum_{k=0}^{N/4-1} [X'_k - X'_{k+N/4}] \exp[-4\pi j(2r+1)k/N], \quad r = 0, 1, \dots, N/4-1 \quad (15)$$

$$= \sum_{k=0}^{N/4-1} [X'_k - X'_{k+N/4}] \exp(-4\pi jk/N) \exp(-8\pi jrk/N), \quad r = 0, 1, \dots, N/4-1. \quad (16)$$

A similar development for  $C_r$  would yield

$$C_{2r} = \sum_{k=0}^{N/4-1} [X'_k + X'_{k+N/4}] \exp(-8\pi jrk/N), \quad r = 0, 1, \dots, N/4-1, \quad (17)$$

$$C_{2r+1} = \sum_{k=0}^{N/4-1} [X'_k - X'_{k+N/4}] \exp(-4\pi jk/N) \exp(-8\pi jrk/N), \quad r = 0, 1, \dots, N/4-1, \quad (18)$$

where  $X'_k$  indicates the data values of  $C_r$  in the previous stage. The matrix form of Eqs. (14), (16), (17), and (18) for the case  $N = 8$  (Fig. 4) shows the simplification resulting from the reduction of  $B_r$  and  $C_r$ .

$$\begin{aligned}
[B_{2r}] & \begin{bmatrix} A_0 \\ A_4 \end{bmatrix} = \begin{bmatrix} W^0 & W^0 \\ W^0 & W^4 \end{bmatrix} \begin{bmatrix} X'_0 \\ X'_1 \end{bmatrix} + \begin{bmatrix} W^0 & W^0 \\ W^0 & W^4 \end{bmatrix} \begin{bmatrix} X'_2 \\ X'_3 \end{bmatrix} \\
& = \begin{bmatrix} W^0 & W^0 \\ W^0 & W^4 \end{bmatrix} \begin{bmatrix} X'_0 + X'_2 \\ X'_1 + X'_3 \end{bmatrix} = \begin{bmatrix} W^0 & W^0 \\ W^0 & W^4 \end{bmatrix} \begin{bmatrix} X''_0 \\ X''_1 \end{bmatrix} = [D_r] \\
[B_{2r+1}] & = \begin{bmatrix} A_2 \\ A_6 \end{bmatrix} = \begin{bmatrix} W^0 & W^0 \\ W^0 & W^4 \end{bmatrix} \begin{bmatrix} X'_0 \cdot W^0 \\ X'_1 \cdot W^2 \end{bmatrix} - \begin{bmatrix} W^0 & W^0 \\ W^0 & W^4 \end{bmatrix} \begin{bmatrix} X'_2 \cdot W^0 \\ X'_3 \cdot W^2 \end{bmatrix} \\
& = \begin{bmatrix} W^0 & W^0 \\ W^0 & W^4 \end{bmatrix} \begin{bmatrix} X'_0 \cdot W^0 - X'_2 \cdot W^0 \\ X'_1 \cdot W^2 - X'_3 \cdot W^2 \end{bmatrix} = \begin{bmatrix} W^0 & W^0 \\ W^0 & W^4 \end{bmatrix} \begin{bmatrix} X''_2 \\ X''_3 \end{bmatrix} = [E_r] \\
[C_{2r}] & = \begin{bmatrix} A_1 \\ A_5 \end{bmatrix} = \begin{bmatrix} W^0 & W^0 \\ W^0 & W^4 \end{bmatrix} \begin{bmatrix} X'_4 \\ X'_5 \end{bmatrix} + \begin{bmatrix} W^0 & W^0 \\ W^0 & W^4 \end{bmatrix} \begin{bmatrix} X'_6 \\ X'_7 \end{bmatrix} \\
& = \begin{bmatrix} W^0 & W^0 \\ W^0 & W^4 \end{bmatrix} \begin{bmatrix} X'_4 + X'_6 \\ X'_5 + X'_7 \end{bmatrix} = \begin{bmatrix} W^0 & W^0 \\ W^0 & W^4 \end{bmatrix} \begin{bmatrix} X''_4 \\ X''_5 \end{bmatrix} = [F_r] \\
[C_{2r+1}] & = \begin{bmatrix} A_3 \\ A_7 \end{bmatrix} = \begin{bmatrix} W^0 & W^0 \\ W^0 & W^4 \end{bmatrix} \begin{bmatrix} X'_4 \cdot W^0 \\ X'_5 \cdot W^2 \end{bmatrix} - \begin{bmatrix} W^0 & W^0 \\ W^0 & W^4 \end{bmatrix} \begin{bmatrix} X'_6 \cdot W^0 \\ X'_7 \cdot W^2 \end{bmatrix} \\
& = \begin{bmatrix} W^0 & W^0 \\ W^0 & W^4 \end{bmatrix} \begin{bmatrix} X'_4 \cdot W^0 - X'_6 \cdot W^0 \\ X'_5 \cdot W^2 - X'_7 \cdot W^2 \end{bmatrix} = \begin{bmatrix} W^0 & W^0 \\ W^0 & W^4 \end{bmatrix} \begin{bmatrix} X''_6 \\ X''_7 \end{bmatrix} = [G_r]
\end{aligned}$$

Fig. 4 - The second stage of the FFT matrix reduction for the case  $N = 8$ ; for this case,  $r = 0, \dots, N/4 - 1$  becomes  $r = 0, 1$

The above process would be repeated for an arbitrary  $N = 2^n$  until the matrices are reduced to the degenerate form of the last stage, to be described later. The total number of stages required for computation of the FFT is, of course, a function of the input data length and is determined by the power of 2 required for a given  $N$ ; i.e., the number of stages required would be  $\log_2 N$ .

As in the preceding stages, only those operations necessary for simplification of the matrices are performed. Thus, as indicated in Fig. 4, the sum  $(X'_0 + X'_2)$  is replaced by  $X''_0$ ,  $(X'_1 + X'_3)$  is replaced by  $X''_1$ , etc. For  $D_r$  we then have

$$D_r = \sum_{k=0}^{N/4-1} X''_k \exp(-8\pi jrk/N), \quad r = 0, \dots, N/4 - 1 \quad (19)$$

$$= \sum_{k=0}^{N/8-1} \{X_k'' \exp(-8\pi jrk/N) + X_{k+N/8}'' \exp[-8\pi jr(k+N/8)/N]\}, \quad r = 0, \dots, N/4 - 1, \quad (20)$$

$$= \sum_{k=0}^{N/8-1} [X_k'' + X_{k+N/8}'' \exp(-\pi jr)] \exp(-8\pi jrk/N), \quad r = 0, \dots, N/4 - 1, \quad (21)$$

$$D_{2r} = \sum_{k=0}^{N/8-1} [X_k'' + X_{k+N/8}''] \exp(-16\pi jrk/N), \quad r = 0, \dots, N/8 - 1, \quad (22)$$

$$D_{2r+1} = \sum_{k=0}^{N/8-1} [X_k'' - X_{k+N/8}''] \exp[-8\pi j(2r+1)k/N], \quad r = 0, \dots, N/8 - 1 \quad (23)$$

$$= \sum_{k=0}^{N/8-1} [X_k'' - X_{k+N/8}''] \exp(-8\pi jk/N) \exp(-16\pi jrk/N), \quad r = 0, \dots, N/8 - 1. \quad (24)$$

Similar developments would simplify  $E_r$ ,  $F_r$ , and  $G_r$ . The results for this reduction to the final stage for  $N = 8$  are shown in Fig. 5.

$$\begin{aligned} D_{2r} &= [A_0] = [w^0] [X_0''] + [w^0] [X_1''] \\ D_{2r+1} &= [A_4] = [w^0] [X_0'' \cdot w^0] - [w^0] [X_1'' \cdot w^0] \\ E_{2r} &= [A_2] = [w^0] [X_2''] + [w^0] [X_3''] \\ E_{2r+1} &= [A_6] = [w^0] [X_2'' \cdot w^0] - [w^0] [X_3'' \cdot w^0] \\ F_{2r} &= [A_1] = [w^0] [X_4''] + [w^0] [X_5''] \\ F_{2r+1} &= [A_5] = [w^0] [X_4'' \cdot w^0] - [w^0] [X_5'' \cdot w^0] \\ G_{2r} &= [A_3] = [w^0] [X_6''] + [w^0] [X_7''] \\ G_{2r+1} &= [A_7] = [w^0] [X_6'' \cdot w^0] - [w^0] [X_7'' \cdot w^0] \end{aligned}$$

Fig. 5 - The last stage of the FFT matrix reduction for the case  $N = 8$ ; for this case,  $r = 0, \dots, N/8 - 1$  becomes  $r = 0$

### Total Computations

We are now in a position to compare the computational savings produced by the FFT as compared to reduction of an  $N$  by  $N$  matrix of Fig. 2 and Eq. (2). From Figs. 3, 4, and 5 we obtain from each stage (assuming  $N = 8$ ) four complex multiplications and eight complex additions. Thus 12 complex multiplications and 24 complex additions suffice for the entire FFT process. This compares with 64 complex multiplications and 56 complex additions for the standard reduction of Eq. (2). In general the fractional computation time is given by  $(N \log_2 N)/N^2$ . For example the computation in the case of 8192 data points can be performed in 0.0016 of the time required by the regular method — a saving of more than 99% in computer time.

### Flow Chart

An alternative view of the FFT can be embodied in the flow charts of Figs. 6 and 7. These show FFT's of eight points and 16 points respectively. In Fig. 7, for example, the original time sampled values are represented on the left as  $X_0$  thru  $X_{15}$  and are complex valued. The figure shows the four stages of computation with the complex operations required for each stage. When two lines connect at the same point in a stage (shown by the enlarged dots), a complex sum is indicated. The lower half of stage 1 requires complex multiplication by complex constants. This is indicated by the value  $W^n$ , equivalent to the values discussed in the preceding sections. Thus at the last data location in Fig. 7, input data value  $X_{15}$  is transferred to stage 2 after it is multiplied by  $-W^7$  and added to the product of data value  $X_7$  and  $W^7$ .

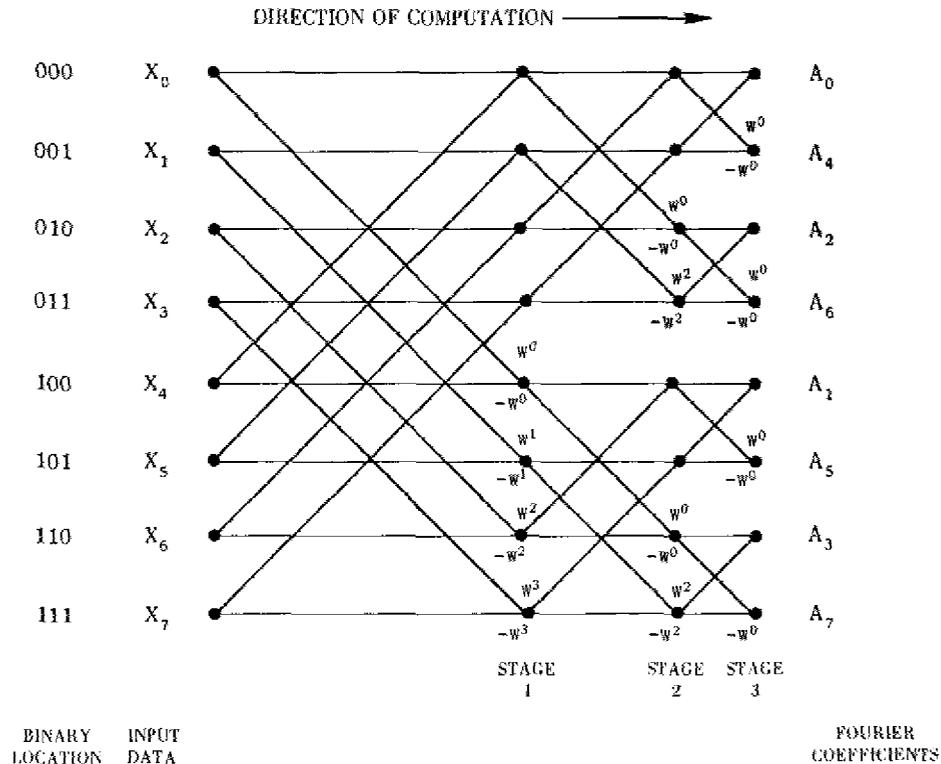


Fig. 6 - Flow chart of an eight-point FFT, showing coefficient scrambling. As an example in interpreting the chart, the dot directly above the label Stage 1 indicates the complex operation  $X_3 \cdot W^3 - X_7 \cdot W^3$ .

DIRECTION OF COMPUTATION

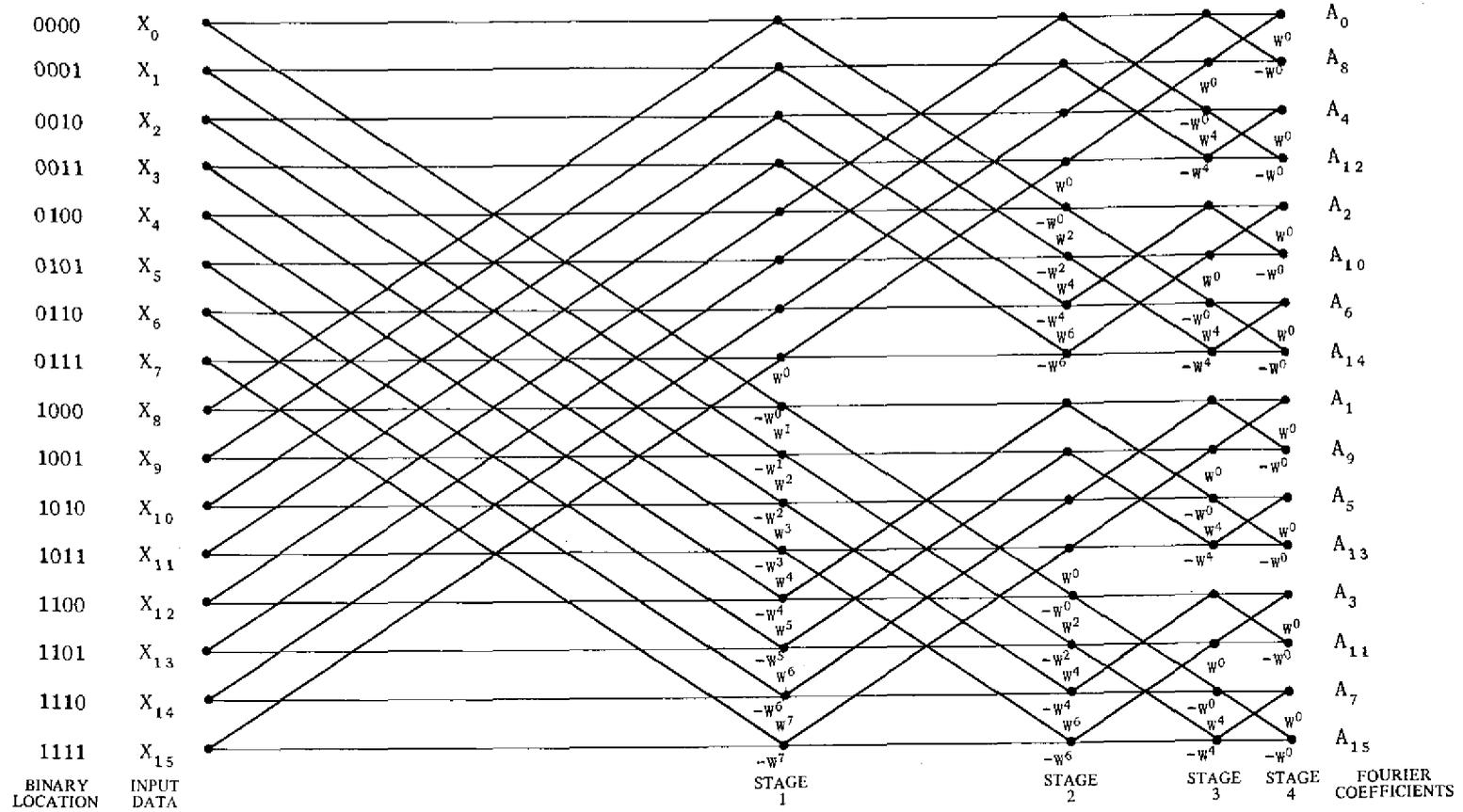


Fig. 7 - Flow chart of a 16-point FFT, showing coefficient scrambling

The flow charts of Figs. 6 and 7 indicate the process known as decimation in frequency and is the method described by Gentleman and Sande (4). The form of computation known as decimation in time is the method originally described by Cooley and Tukey (2). Comparison of both methods as well as variations to eliminate coefficient scrambling are described by Cochran, Cooley, et al. (1).

### Coefficient Scrambling

As previously indicated the coefficients have been computed in a scrambled sequence. To arrange the coefficients in the order from the lowest frequency component (the dc term) to the highest, it is necessary to select the coefficients according to their scrambled position in computer memory. As seen from Fig. 6 coefficient  $A_0$  is correct, but  $A_1$  and  $A_4$  have been interchanged,  $A_3$  and  $A_6$  have been interchanged, etc.

The scrambling, of course, has been the result of computing even-coefficient groups before odd, thus taking advantage of the symmetry relations of the FFT. In base 2 notation we see this to be the reversal of the binary number representing the subscript of the Fourier coefficient. Consequently, for an FFT of  $N = 8$  length, coefficient  $A_1 = A_{001}$  can be found in location  $100_2 = 4$ , since  $100_2$  is the binary reversal of  $001_2$ . Similarly,  $A_6 = A_{110}$  can be found in location  $011_2 = 3$ , and so on for all coefficients.

The reason for this scrambling can more easily be understood by consideration of the basic equation (Eq. 4) in its binary notation for the case  $N = 8$ . The indices  $r$  and  $k$  can be written in binary form as

$$r = r_2(2^2) + r_1(2^1) + r_0(2^0) \quad (25)$$

and

$$k = k_2(2^2) + k_1(2^1) + k_0(2^0) \quad (26)$$

where  $r_2, r_1, r_0, k_2, k_1, k_0 = 0, 1$ . Thus Eq. (4) becomes

$$A(r_2, r_1, r_0) = \sum_{k_0=0}^1 \sum_{k_1=0}^1 \sum_{k_2=0}^1 X(k_2, k_1, k_0) \cdot \left[ W^{(4r_2+2r_1+r_0)(4k_2+2k_1+k_0)} \right] \quad (27)$$

The exponent of  $W$  can be simplified by performing the multiplications suggested. Equation (27) thus becomes

$$A(r_2, r_1, r_0) = \sum_{k_0=0}^1 \sum_{k_1=0}^1 \sum_{k_2=0}^1 X(k_2, k_1, k_0) \cdot \left[ W^{(4r_2+2r_1+r_0)4k_2} W^{(4r_2+2r_1+r_0)2k_1} W^{(4r_2+2r_1+r_0)k_0} \right] \quad (28)$$

For simplicity of notation, let us consider at present the first  $W$  term of the above equation:  $W^{(4r_2+2r_1+r_0)4k_2}$ . This is easily reduced to

$$W^{16r_2k_2} W^{8r_1k_2} W^{4r_0k_2} = W^{4r_0k_2},$$

since  $W^N = \exp[-2\pi j(n)/N]$  and  $W^{16n} = W^{8n} = 1$ . Similarly  $W^{(4r_2+2r_1+r_0)2k_1}$  can be reduced to

$$W^{8r_2k_1} W^{4r_1k_1} W^{2r_0k_1} = W^{(2r_1+r_0)2k_1}$$

Equation (28) could thus be written

$$\begin{aligned} A(r_2, r_1, r_0) &= \sum_{k_0=0}^1 \sum_{k_1=0}^1 \sum_{k_2=0}^1 X(k_2, k_1, k_0) \\ &\quad \cdot \left[ W^{4r_0k_2} W^{(2r_1+r_0)2k_1} W^{(4r_2+2r_1+r_0)k_0} \right] \\ &= \sum_{k_0=0}^1 \sum_{k_1=0}^1 \sum_{k_2=0}^1 X(k_2, k_1, k_0) W^\alpha W^\beta W^\delta. \end{aligned} \quad (29)$$

If we examine the sequence of calculations in obtaining the Fourier coefficients, we find that the coefficients are obtained in a binary reversal sequence as suggested previously. This is easily seen by consideration of the summations one at a time. The inner sum including  $W^\alpha$  is a function of  $r_0$ ,  $k_1$ , and  $k_0$ . Thus we have

$$A(r_2, r_1, r_0) = \sum_{k_0=0}^1 \sum_{k_1=0}^1 f_1(r_0, k_1, k_0) W^\beta W^\delta. \quad (30)$$

We now see the new inner sum including  $W^\beta$  as a function of  $r_0$ ,  $r_1$ , and  $k_0$ . Thus

$$A(r_2, r_1, r_0) = \sum_{k_0=0}^1 f_2(r_0, r_1, k_0) W^\delta.$$

But this sum including  $W^\delta$  is a function of  $r_0$ ,  $r_1$ , and  $r_2$ . Thus

$$A(r_2, r_1, r_0) = f_3(r_0, r_1, r_2),$$

which we see to be nothing more than the binary reversal previously suggested.

The problem of unscrambling can be handled in several ways. Modifications of the FFT procedure can be derived which require no unscrambling but at a sacrifice of requiring increased computer memory for a given number of input data values. Essentially, the storage requirements of such programs would be twice the value for scrambled algorithms -- effectively halving the resolution obtainable from a given memory size.

An alternative method would require the input time samples to be stored in computer memory in scrambled positions according to the binary reversal code. Thus, on conclusion of the FFT process, the coefficients would be in correct order. Consequently, the previously developed algorithm could be used and no additional memory locations would be required. This would be the natural method to use in special purpose computers designed only to perform Fourier transforms, in which input data can be buffered directly

into the memory for processing by the arithmetic unit. The buffer device would be hardware-wired to perform the binary bit reversal and would not require additional computation time.

For general purpose computers in which input data must be handled by decisions involving the central processing unit, either input or output values must be scrambled by suitable programs. In the program described in the next section the frequency coefficients are unscrambled by the subroutine UNSCR in conjunction with subroutine GNSCR.

## FORTRAN IV PROGRAM FOR FFT

### General

The Fortran IV program consisting of subroutines FFOUR, COSINE, GNSCR, and UNSCR, is constructed in such a way that complex numbers can be computed by complex operations without recourse to considerations of the real and imaginary parts as separate quantities. In some operations however it is necessary or convenient to perform operations on the real or imaginary parts individually. This is illustrated by Fig. 8, where the general construction of the data block in computer memory is shown. The complex values are associated with array X, the real values are associated with array Y, and the two arrays are equated by an equivalence statement in the Fortran program.

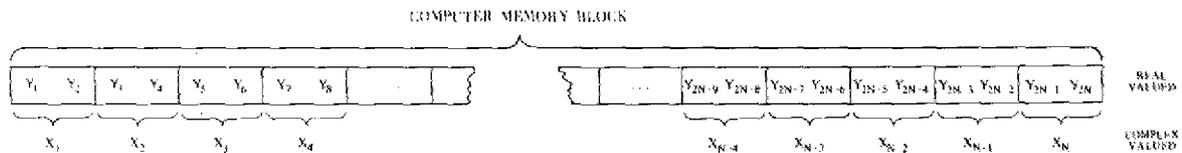


Fig. 8 - Computer memory block equating complex values with their real components

Before execution of the program begins, these arrays are filled with the data values such as might be obtained from the output of an analog-to-digital converter. If the input values are real, as is normally the case, they are placed in the odd values of the Y array (the real part of X). The even values of Y are set to zero, indicating no phase component.

During computation of each FFT stage the computed results are stored "in place"; i.e., the results of computation of each stage replace the results of the previous stage. Consequently no intermediate storage locations are required, and on completion of computation the complex Fourier coefficients remain in array X.

Subroutine FFOUR is the main subroutine, which computes the Fourier coefficients, stage by stage, according to the flow charts in Figs. 6 and 7. At the conclusion of subroutine FFOUR subroutine UNSCR is called to unscramble the coefficients and place them in correct order in array X. As shown in Fig. 9 the main program should be constructed with subroutines COSINE and GNSCR outside the FFT loop. Subroutine COSINE generates the complex constants  $W^n$  required for the FFT processing and need be computed only once when many FFT's of similar length are to be processed in sequence. Similarly GNSCR, which generates the computer addresses needed in unscrambling, should be calculated only once when the data length remains unchanged.

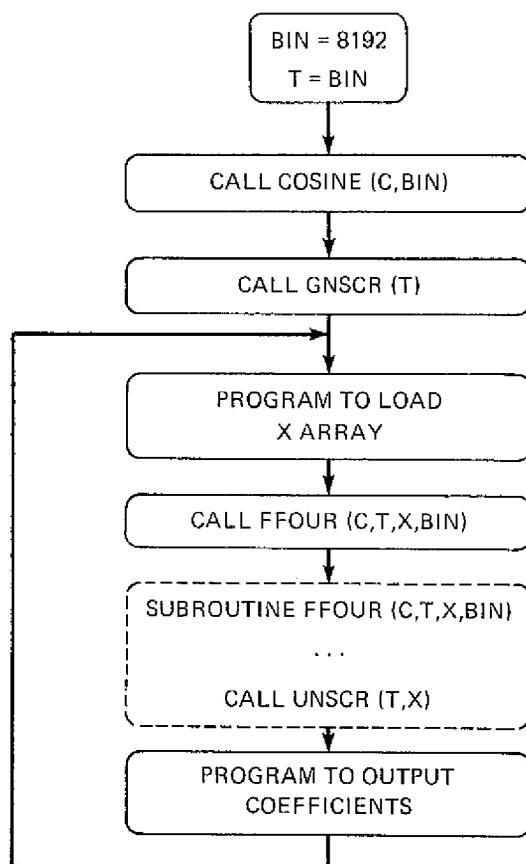


Fig. 9 - Flow chart for repetitive FFT operation

The FFT length is specified by the statement  $BIN = N = 2^n$ . This is the only statement that must be modified if FFT's of different lengths are desired in the same program. A typical deck structure for the NRL CDC 3800 computer assuming a data length of 8192 values is shown in Fig. 10.

#### Subroutine FFOUR

Subroutine FFOUR (Fig. 11) is the basic subroutine that computes the Fourier coefficients from the input data. When needed the subroutine calls on the results of additional subroutines which have the task of computing the complex exponentials associated with the Fourier transform (subroutine COSINE) and performing the coefficient unscrambling (subroutine GNSCR in conjunction with subroutine UNSCR). As shown in the parameter list, subroutine FFOUR requires values in array C for complex constants and the array X for access to original data values at the beginning of the subroutine. The data length is established by parameters T and BIN.

In addition to the dimensioned arrays C and X, subroutine FFOUR also requires the dimensioned array CSE, which is equated to the complex storage location E; location E may be accessed by a complex operator or by successive operations on the real part of E, namely, CSE(1), and on the imaginary part of E, namely, CSE(2).

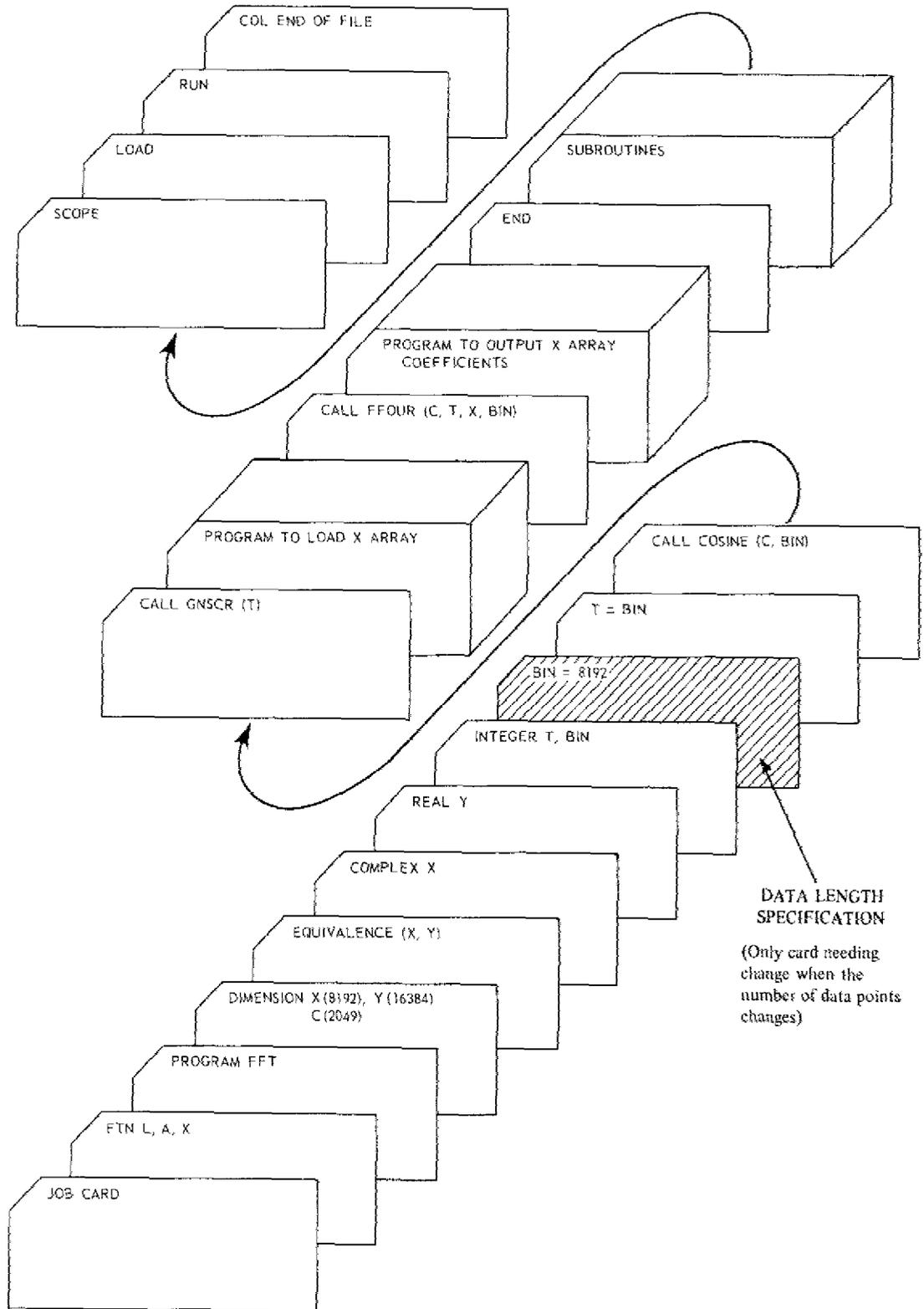


Fig. 10 - Deck structure for computing one FFT using the NRL CDC 3800 computer

NRL REPORT 7041

	SUBROUTINE FFOUR(C,T,X,BIN)	FF1
	COMPLEX E,X	FF2
	REAL CSE,C	FF3
	INTEGER T,TINC,THTA,BIN	FF4
	DIMENSION CSE(2),C(2049),X(8192)	FF5
	EQUIVALENCE(E,CSE)	FF6
	TINC=BIN/T	FF7
	LINC=T/2	FF8
10	LOC=LINC	FF9
40	THTA=0	FF10
80	CSE(1)=C(THTA+1)	FF11
	CSE(2)=-C(BIN/4+1-THTA)	FF12
60	LOC=LOC+1	FF13
	LOC1=LOC-LINC	FF14
	E=E*(X(LOC1)-X(LOC))	FF15
	X(LOC1)=X(LOC1)+X(LOC)	FF16
	X(LOC)=E	FF17
	THTA=THTA+TINC	FF18
	IF(THTA-BIN/2)20,30,30	FF19
20	IF(THTA-BIN/4)80,50,50	FF20
50	CSE(1)=-C(BIN/2+1-THTA)	FF21
	CSE(2)=-C(THTA-(BIN/4-1))	FF22
	GO TO 60	FF23
30	IF(LOC-T)90,91,91	FF24
90	LOC=LOC+LINC	FF25
	GO TO 40	FF26
91	IF(2-LINC)92,93,94	FF27
92	LINC=LINC/2	FF28
	TINC=TINC+TINC	FF29
	GO TO 10	FF30
93	DO 100 LOC=2,T,2	FF31
	LOC1=LOC-1	FF32
	E=X(LOC1)-X(LOC)	FF33
	X(LOC1)=X(LOC1)+X(LOC)	FF34
100	X(LOC)=E	FF35
94	CALL UNSCR(T,X)	FF36
	RETURN	FF37
	END	FF38

Fig. 11 - Subroutine FFOUR

eters TINC

whose argument changes by fixed amounts, its value in quadrant II can be determined by changing only the argument of  $Z$  in quadrant I. Consequently, in subroutine FFOUR access is made only to quadrant I, with corresponding sign changes for quadrant II.

The test for membership in quadrant I or II is made by statement FF20. If the vector belongs to quadrant I, the program returns to statement FF11 to repeat operations on succeeding values in the  $X$  array; should membership be in quadrant II, the real part of the vector is changed in sign in statement FF21 and then returns to FF13 for repeated operations provided the program has not arrived at the end of a stage or substage.

If we have arrived at the end of a substage (such as computing the last values in  $D_r$  and  $E_r$  in Fig. 4) we test by statement FF24 if this is the last substage in a given stage. If it is not, the storage locations of the beginning of the next substage (such as  $F_r$  and  $G_r$  in Fig. 4) are computed in statement FF25 and the complex vector is reset to its initial value in statement FF10 and the process is repeated. Should FF24 indicate that the last substage has been computed, statement FF27 is tested for determination of which stage is next to be computed. This process will be repeated until all stages except the last are computed.

The computation time of the last stage may be considerably reduced by performing it in a different manner from the preceding ones. Since the vector in this stage always equals 1 or -1, no complex multiplications are required. Such an operation involving only sums and differences is performed in the DO loop indicated by FF31 through FF35.

We have now completed the computation of  $N$  complex Fourier coefficients. However, they are in scrambled order and must be rearranged by subroutines UNSCR and GNSCR.

#### Subroutines GNSCR and UNSCR

Once the Fourier coefficients have been computed, appropriate measures must be incorporated for presenting them in sequential order. As explained previously, the coefficients will be in computer memory according to the bit reversal in binary notation associated with the order of the coefficient, upon completion of subroutines

The binary numbers are ...  
puter by ...

Continuing in the same manner we see that the next two scrambled values are  $010 = 2$  and  $110 = 6$  respectively. These values can be obtained by adding  $N/4$  to the previous values obtained (in this case 0 and 4). Combining the previous values and the present values we now have the sequence 0, 4, 2, 6. The next set of scrambled values are obtained by adding  $N/8$  to the previous values, and we obtain the complete scrambled sequence 0, 4, 2, 6, 1, 5, 3, 7.

In summary, for any power  $N$  of two we can obtain the binary reversed sequence by considering the first value to be zero (zero of course is its own reversal) and then extending the sequence by adding diminished powers of two to the previous values obtained. In the computer program the scrambled values are used as address values for obtaining the Fourier coefficients in correct order. In Fortran notation they would be considered the subscripts of a dimensioned array. One additional factor must be accounted for: the Fortran language does not permit a dimensioned subscript of zero. Thus we must increase by one the scrambled address values we wish to obtain. For  $N = 8$  we would then have the values 1, 5, 3, 7, 2, 6, 4, 8.

Subroutine GNSCR computes the scrambled values defined above for  $N$  data values. This subroutine is shown in Fig. 12. The formal parameter  $T$  specifies the data length being computed and is equal to the number of coefficients in the FFT; its value is obtained from the main program as shown in Figs. 9 and 10. As the scrambled values are computed, they are stored in array SCR. The first scrambled value is the reversal of itself, and the value in location 1 is identically equal to 1 as indicated by statement FF44. Location  $M$  in the subroutine defines how many elements are in the sequence previous to the "add" cycle. Thus for  $T = 8$ , after location 1 in array SCR is set to one by statement FF44, statement FF45 tells us that the sequence consists of one element. During each cycle of additions to previous values, the value  $M$  will be used to define the DO loop in statement FF48. Statement FF52 increases  $M$  upon completion of the DO loop in preparation of the next add cycle.

The value  $TM$  defines the algebraic quantity to be added to previous values during each add cycle. This will be  $N/2$  for the first cycle,  $N/4$  for the next cycle,  $N/8$  for the next, etc., where  $N$  is the data length. In statement FF47,  $TM$  is set equal to  $N/2$  as required at the beginning of the first add cycle: statement FF53 modifies  $TM$  for the beginning of the next cycle. The above process is repeated until the full sequence of  $N$  values has been computed. The termination of the subroutine is executed by statement FF51 when the last scrambled value has been computed. (The last value computed is the binary reversal of itself and is equal to  $N = T$ ).

Having computed the coefficient scrambling order, it is now possible to rearrange the scrambled coefficients in array  $X$  in their proper sequence of increasing ordinal number. This is performed in subroutine UNSCR, in which the scrambled value from array SCR is placed, by statement FF63, in location  $I$  for temporary storage. Ignoring statement FF64 for the moment, coefficients are interchanged by statements FF65, FF66, and FF67 according to the value obtained from array SCR. For the example  $N = 8$ , array SCR contains 1, 5, 3, 7, 2, 6, 4, 8; consequently location 5 would be interchanged with location 2, since ordinal location 2 in array SCR contains the numerical value 5. It should be noted at this time that ordinal location 5 in array SCR contains a numerical 2. When the DO loop progresses to this point, the values in array  $X$ , locations 2 and 5, would be interchanged for the second time, and the coefficients in these locations would remain unscrambled. Statement FF64 prevents such an occurrence by requiring the numerical value in array SCR to be greater than the ordinal location in that array. Hence interchanging occurs only once for a given set of coefficients, and unscrambling is correctly executed.

	SUBROUTINE GNSCR(T)	FF40
	COMMON SCR	FF41
	INTEGER SCR,T,TM	FF42
	DIMENSION SCR(8192)	FF43
	SCR(1)=1	FF44
	M=1	FF45
	N=1	FF46
	TM=T/2	FF47
11	DO 9 J=1,M	FF48
	N=N+1	FF49
9	SCR(N)=TM+SCR(J)	FF50
	IF(T-SCR(N))8,8,10	FF51
10	M=M+M	FF52
	TM=TM/2	FF53
	GO TO 11	FF54
8	RETURN	FF55
	END	FF56
	SUBROUTINE UNSCR(T,X)	FF57
	COMMON SCR	FF58
	COMPLEX X,E	FF59
	INTEGER T,SCR	FF60
	DIMENSION SCR(8192),X(8192)	FF61
	DO 9 J=1,T	FF62
	I=SCR(J)	FF63
	IF(J-1) 9,9,10	FF64
10	E=X(J)	FF65
	X(J)=X(1)	FF66
	X(1)=E	FF67
9	CONTINUE	FF68
	RETURN	FF69
	END	FF70
	SUBROUTINE COSINE(C,BIN)	FF80
	DIMENSION C(2049)	FF81
	INTEGER BIN	FF82
	ANG=(3.1415926536)/(BIN/2)	FF83
	INTB=BIN/8	FF84
	CS=COSF(ANG)	FF85
	SN=SINF(ANG)	FF86
	C(1)=1.	FF87
	C(BIN/4+1) = 0.	FF88
	DO 9 J=1,INTB	FF89
	C(J+1)=C(J)*CS-C(BIN/4+2-J)*SN	FF90
9	C(BIN/4+1-J)=C(J)*SN+C(BIN/4+2-J)*CS	FF91
	RETURN	FF92
	END	FF93

Fig. 12 - Subroutines UNSCR, COSINE, and GNSCR

### Subroutine COSINE

Subroutine COSINE is used to compute the values of the complex exponentials necessary for reduction of the FFT to complex coefficients. The complex exponentials required are of the form  $\exp(-2\pi jn/N)$ , where  $n$  varies from 0 to  $N/2 - 1$ . This is equivalent to generation of a half cycle of both a sine and a cosine wave, since by Euler's relationship  $\exp(-2\pi jn/N) = \cos(2\pi n/N) - i \sin(2\pi n/N)$ . Then by proper indexing techniques as explained for subroutine FFOUR, we can perform the complex operations by consideration of only real values of a discrete sine and cosine table extending from zero to a half cycle, in increments of  $2\pi/N$ .

If the values of a sine or cosine wave are examined, it is obvious that the absolute values in each quarter cycle are identical with the absolute values in any other quarter cycle. Furthermore, if we consider a quarter cycle of the cosine wave  $y = \cos X$ , where  $y = 1.0$  for  $X = 0$  and  $y = 0$  for  $X = \pi/2$ , we see that a sine wave can be obtained by "reading backward" on the cosine wave. Thus when  $X = \pi/2$  in  $y = \cos X$ , the value of  $y$  at that point is equivalent to the value of  $y = \sin X$  when  $X = 0$ . Consequently, only a quarter cycle of a cosine wave is computed and stored for use by subroutine FFOUR, and indexing and sign changes perform the operations for the missing quarter cycle.

In subroutine COSINE the basic increment  $2\pi/N$  is computed by statement FF83. CS and SN in statements FF85 and FF86 are the cosine and sine values of the basic increments respectively. Since we are computing a quarter cycle of a cosine wave, the first value will be 1.0 and the last value (actually the beginning value of the next quarter cycle) will be 0. These two values are determined by statements FF87 and FF88. For a quarter cycle of a cosine wave we would need to compute  $N/4$  values, and the DO loop in statement FF89 would be from 1 to  $INTB = BIN/4$ . We can however reduce the DO loop to half this value by use of statements FF90 and FF91, where the quarter cycle of cosine wave is computed from both ends. Statement FF90 computes cosine values by the trigonometric identity  $\cos(\theta + \phi) = \cos \theta \cos \phi - \sin \theta \sin \phi$ , and statement FF91 computes sine values by the identity  $\sin(\theta + \phi) = \cos \theta \sin \phi + \sin \theta \cos \phi$ , where  $\theta$  is the previous value computed in each statement and  $\phi$  is the increment value.

The advantages of this method of computing sine and cosine values are two: the computer sine and cosine routines are needed only once each, thus saving considerable computer time, and computing from both ends of the quarter cycle reduces accumulative error in the computations by reducing round-off errors inherent in adding the increment ANG in statement FF83 to itself many times.

### ACKNOWLEDGMENT

The author is indebted to Gerald C. Drew for his efforts in developing a FFT Fortran program for computing the power spectrum of a fixed data length.

## REFERENCES

1. W.T. Cochran, J.W. Cooley, et al., "What is the Fast Fourier Transform?" IEEE Trans. Audio and Electroacoustics AU-15, 45-55 (June 1967)
2. J.W. Cooley and J.W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," Math. of Computation 19, 297-301 (Apr. 1965)
3. D.A. Swick, "Discrete Finite Fourier Transforms: a Tutorial Approach," NRL Report 6557, June 29, 1967
4. W.M. Gentleman and G. Sande, "Fast Fourier Transforms - For Fun and Profit," 1966 Fall Joint Computer Conf. AFIPS Proc., Vol. 29, Washington, Spartan Books, pp. 563-578, 1966