<table>
<thead>
<tr>
<th>Technical Note</th>
<th>1967-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>N. M. Brenner</td>
<td></td>
</tr>
<tr>
<td>Three Fortran Programs</td>
<td></td>
</tr>
<tr>
<td>that Perform the Cooley-Tukey</td>
<td></td>
</tr>
<tr>
<td>Fourier Transform</td>
<td></td>
</tr>
<tr>
<td>28 July 1967</td>
<td></td>
</tr>
</tbody>
</table>

Prepared under Electronic Systems Division Contract AF 19(628)-5167 by

Lincoln Laboratory

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Lexington, Massachusetts
The work reported in this document was performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology, with the support of the U.S. Air Force under Contract AF 19(628)-5167. This report may be reproduced to satisfy needs of U.S. Government agencies.

This document has been approved for public release and sale; its distribution is unlimited.
ERRATA SHEET
for Technical Note 1967-2

Because of unclear printing in Technical Note 1967-2 (N.M. Brenner, "Three Fortran Programs that Perform the Cooley-Tukey Fourier Transform," 28 July 1967), the distinction between + and * was often lost. A list of clarifications follows on the attached sheets.

7 September 1967

Publications
M.I.T. Lincoln Laboratory
P. O. Box 73
Lexington, Massachusetts 02173
THE FOLLOWING THREE PATTERNS OCCUR FREQUENTLY.
\[ BR = WR \ast AR - WI \ast AI \]
\[ BI = AI \ast WR + AR \ast WI \]
\[ DATA(J) = DATA(I) - TEMPR \]
\[ DATA(J+1) = DATA(I+1) - TEMPI \]
\[ DATA(I) = DATA(I) + TEMPR \]
\[ DATA(I+1) = DATA(I+1) + TEMPI \]
\[ INDEX2MAX = INDEX1 + N1 - N2 \]

7
ISTEP = 2*MMAX

P. 15, L. 7

P. 21, L. 2 AND P. 17, L. 2

2
NTOT = NTOT*NN(IDIM)

P. 22, L. 5-2 AND P. 17, L. 100-2
NP2 = NP1*N

P. 22, L. 12 AND L. 51

12 OR 51
NTWO = NTWO + NTWO

P. 22, L. 70+2
I1 RNG = NP1
IF(IDIM = 4) 71, 100, 100

P. 23, L. 72+1
I1 RNG = NP0*(1 + NPREV/2)

P. 23, L. 120 AND P. 17, L. 110

110 OR 120
I1 MAX = I2 + NP1 - 2

P. 23, L. 120+3 AND P. 17, L. 110+3
J3 = J + I3 - I2

P. 23, L. 200

NWORK = 2*N

P. 23, L. 210-1
IF(ICASE = 3) 210, 220, 210

P. 23, L. 240+1
J = J + IFP1
IF(J = I3 - IFP2) 260, 250, 250

P. 24, L. 420+1 AND P. 18, L. 420+1
KMIN = IPAR*M + I1

P. 24, L. 440 AND P. 18, L. 440

440
KDIF = IPAR*MMAX

450
KSTEP = 4*KDIF

P. 24, L. 520+1 AND P. 18, L. 520+1
KMIN = 4*(KMIN - I1) + I1
KDIF = KSTEP
IF(KDIF = NP2HF) 450, 450, 530

P. 25, L. 550+1 AND P. 19, L. 550+1
WR = (WR + WI) * RTHLF
P. 25, L. 560+2 AND P. 19, L. 560+2
WI=(TEMPR+WI)*RTHLF

P. 25, L. 570+2 AND P. 19, L. 570+2
MMAX=MMAX+MMAX

P. 26, L. 650+2
J2RNG=IFPl*(1+IFACT(IF)/2)

P. 26, L. 655-2
I=1+(J3-I3)/NP1HF

P. 26, L. 665
ICONJ=1+(IFP2-2*J2+I3+J3)/NP1HF

P. 27, L. 670+1
TEMI=SUMI
SUMR=TWOWR*SUMR-OLDSR+DATA(J)
SUMI=TWOWR*SUMI-OLDSI+DATA(J+1)
OLDSR=TEMPR
OLDSI=TEMPI
J=J-IFP1
IF(J-JMIN)675,675;670
675 TEMPR=WR*SUMR-OLDSR+DATA(J)
TIEMPI=WI*SUMI
WORK(I)=TEMPR-TEMDI
WORK(ICONJ)=TEMPR+TEMPI
TEMDR=WR*SUMI-OLDSI+DATA(J+1)
TEMDI=WI*SUMR
WORK(I+1)=TEMPR+TEMPI
WORK(ICONJ+1)=TEMPR-TEMDI

P. 27, L. 690+2
I2MAX=I3+NP2-NP1

P. 27, L. 710-2
JMIN=2*NHALF-1

P. 28, L. 740
NP2=NP2+NP2

P. 28, L. 745-1
IMAX=NTOT/2+1
745 IMIN=IMAX-2*NHALF

P. 28, L. 805+1
I2MAX=I3+NP2-NP1

P. 28, L. 805+3
IMIN=I2+I1RN6
IMAX=I2+NP1-2
JMA=2*I3+NP1-IMIN

P. 28, L. 810
JMAX=JMAX+NP2
810 IF(IDIM-2)850,850;830
830 J=JMAX+NP0

P. 28, L. 840
840 J=J-2

P. 28, L. 860
J=J-NP0
THREE FORTRAN PROGRAMS THAT PERFORM
THE COOLEY-TUKEY FOURIER TRANSFORM

N. M. BRENNER

Group 31

TECHNICAL NOTE 1967-2

28 JULY 1967
This note describes and lists three programs, all written in USASI Basic Fortran, which perform the discrete Fourier transform upon a multi-dimensional array of floating point data. The data may be either real or complex, with a savings in running time for real over complex. The transform values are always complex and are returned in the array used to carry the original data. The running time is much shorter than that of any program performing a direct summation, even when sine and cosine values are precalculated and stored in a table. For example, on a CDC 3300 with floating point add time of six microseconds, a complex array of size 80 x 80 can be transformed in 19.2 seconds. Besides the main array, only a working storage array of size 160 need be supplied.

Accepted for the Air Force
Franklin C. Hudson
Chief, Lincoln Laboratory Office
This note describes and lists three programs, all written in USASI Basic Fortran, which perform the discrete Fourier transform upon a multi-dimensional array of floating point data. The data may be either real or complex, with a savings in running time for real over complex (see Timing). The transform values are always complex and are returned in the array used to carry the original data. The running time is much shorter than that of any program performing a direct summation, even when sine and cosine values are precalculated and stored in a table. For example, on a CDC 3300 with floating point add time of six microseconds, a complex array of size 80 x 80 can be transformed in 19.2 seconds. Besides the main array, only a working storage array of size 160 need be supplied.

The exact operation performed is called finite discrete Fourier transformation, also known as harmonic analysis or trigonometric interpolation. Given an array of data \( \text{DATA}(I1, I2, \ldots) \),

\[
\text{TRANSFORM}(J1, J2, \ldots) = \sum [\text{DATA}(I1, I2, \ldots) \ W1^{(I1-1)(J1-1)} \ W2^{(I2-1)(J2-1)} \ldots],
\]

where \( W1 = \exp(-2\pi i / N1) \), \( W2 = \exp(-2\pi i / N2) \), \ldots and \( I1 \) and \( J1 \) run from 1 to \( N1 \), \( I2 \) and \( J2 \) run from 1 to \( N2 \), etc. The Fortran convention of subscripts beginning at one is adhered to. This summation possesses many of the properties of the more usual infinite integral

\[
F(y) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i xy} \ dx.
\]

By interpreting the subscripts modulo \( N1, N2, \) etc. and requiring the data to represent equispaced points, we can easily prove the usual properties about linearity, orthogonality, inverse transform and relationship to convolution. See Gentleman and Sande ([3], 1966).
There is no limit on the dimensionality (number of subscripts) of the data array. A three-dimensional transform can be performed as easily as a one-dimensional transform, though in a proportionately greater time. An inverse transform can be performed, in which the sign in the exponentials is +, instead of -. If an inverse transform is performed upon an array of transformed data, the original data will reappear multiplied by \(N_1N_2\ldots\).

The length of each dimension may be any integer, and as large as storage will permit. However, the program runs faster on composite integers than on primes, and is particularly fast on numbers rich in factors of two. For example, on the CDC 3300, the following timings for a one-dimensional transform have been calculated from the timing formula:

<table>
<thead>
<tr>
<th>(N)</th>
<th>Factorization</th>
<th>Time for Complex Transform (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4094</td>
<td>(2 \times 23 \times 89)</td>
<td>80</td>
</tr>
<tr>
<td>4095</td>
<td>(3^2 \times 5 \times 7 \times 13)</td>
<td>24</td>
</tr>
<tr>
<td>4096</td>
<td>(2^{12})</td>
<td>6.2</td>
</tr>
<tr>
<td>4097</td>
<td>(17 \times 241)</td>
<td>180</td>
</tr>
<tr>
<td>4098</td>
<td>(2 \times 3 \times 683)</td>
<td>480</td>
</tr>
<tr>
<td>4099</td>
<td>prime</td>
<td>2868</td>
</tr>
<tr>
<td>4100</td>
<td>(2^2 \times 5^2 \times 41)</td>
<td>39</td>
</tr>
</tbody>
</table>

Calling Sequence

The listings of three programs are given in the appendices. FOUR1 is a subset of FOUR2, which in turn is a subset of FOURT. FOURT is the most general, accepting multidimensional arrays of any size. FOUR2 is the same speed as FOURT but accepts only complex multidimensional arrays whose dimensions are powers of two. FOUR1 is much slower than FOURT or FOUR2, and performs only one-dimensional transforms on complex arrays whose lengths are powers of two. FOUR1 is intended mainly for pedagogical purposes; it is half a page of Fortran, the others being much longer.
The calling sequences are:

CALL FOURT (DATA, NN, NDIM, ISIGN, IFORM, WORK)
CALL FOUR2 (DATA, NN, NDIM, ISIGN)
CALL FOUR1 (DATA, NN, ISIGN)

In all cases, DATA is the array used to hold the real and imaginary parts of the input data and the transform values on output. The real and imaginary parts of a datum must be placed into immediately adjacent locations in storage. This is the form of storage used by Fortran IV, and may be accomplished in Fortran II by making the first dimension of DATA of length two, referring to the real and imaginary parts. If the data placed in DATA on input are real, they must have imaginary parts of zero appended. The transform values are always complex and replace the input data. Hence, the array DATA must always be of complex format.

For FOUR1, array DATA must be one-dimensional, of length NN. For FOUR2 and FOURT, it may be multidimensional. The extent of each dimension (except for the possible first dimension referring to the real and imaginary parts) is given in the integer array NN, which is of length NDIM, the number of dimensions. That is, NN(1) = N1, NN(2) = N2, etc. *

ISIGN is an integer used to indicate the direction of the transform. It is minus one to indicate a forward transform (exponential sign is -) and plus one to indicate an inverse transform (sign is +). The scale factor 1/(N1*N2*...) frequently seen in definitions of the Fourier transform must be applied by the user.

If the data being passed to FOURT are real (i.e., have zero imaginary parts), the integer IFORM should be set to zero. This will speed execution (see Timing). For complex data, IFORM must be plus one.

WORK is an array used by FOURT when any of the dimensions of DATA is not a power of two. Since FOUR2 and FOUR1 are restricted to powers of two, WORK is not needed. If the dimensions of DATA are all powers of two in FOURT, WORK may be replaced by a zero in the calling sequence. Otherwise, it must be

* As usual, the first subscript varies the fastest in storage order.
supplied, a real floating point array of length twice the longest dimension of DATA which is not a power of two. In one dimension, for the length not a power of two, WORK occupies as many storage locations as DATA. If given, it may not be the same array as DATA.

Double precision versions of these programs may be obtained by changing the names to DFOURT, DFOUR2, and DFOUR1, declaring double precision all variables not beginning with the letters I, J, K, L, M or N, changing the references to COS and SIN to DCOS and DSIN and assigning the correct precision constants to TWOPI (2π) and RTHLF (0.5^2). DATA and WORK must then be double precision arrays.

Storage and Common

No common of any kind is used. An integer array of length thirty-two is used by FOURT. FOURT is about four hundred Fortran statements long, FOUR2 about one hundred and twenty and FOUR1 thirty-seven.

Return and Error Messages

There are no error messages, error halts or error returns in this program. If NDIM or any NN(I) is less than one, the program returns immediately.

Algorithm

A heavily modified version of the algorithm discovered independently by Danielson and Lanczos ([2], 1942), Good ([4], 1958), and Cooley and Tukey ([1], 1965) is used. The following example is an application to a one-dimensional transform of length six.

Let \( w = e^{-2\pi i/6} \). The transformation is written

\[
\begin{align*}
t_0 &= d_0 + d_1 + d_2 + d_3 + d_4 + d_5 \\
t_1 &= d_0 + wd_1 + w^2d_2 + w^3d_3 + w^4d_4 + w^5d_5 \\
t_2 &= d_0 + w^2d_1 + w^4d_2 + w^6d_3 + w^8d_4 + w^{10}d_5
\end{align*}
\]
\[ t_3 = d_0 + w^2d_1 + w^6d_2 + w^9d_3 + w^{12}d + w^{15}d_5 \]
\[ t_4 = d_0 + w^4d_1 + w^8d_2 + w^{12}d_3 + w^{16}d_4 + w^{20}d_5 \]
\[ t_5 = d_0 + w^5d_1 + w^{10}d_2 + w^{15}d_3 + w^{20}d_4 + w^{25}d_5 \]

Straightforward computation requires 25 complex multiplications and 30 complex additions. The fast Fourier transform computes as follows:

\[
\begin{align*}
u_0 &= d_0 + d_3 \\
u_1 &= d_0 + w^3d_3 \\
u_2 &= d_1 + d_4 \\
u_3 &= d_1 + w^3d_4 \\
u_4 &= d_2 + d_5 \\
u_5 &= d_2 + w^3d_5 \\
t_0 &= u_0 + u_2 + u_4 \\
t_1 &= u_1 + wu_3 + w^2u_5 \\
t_2 &= u_0 + w^2u_2 + w^4u_4 \\
t_3 &= u_1 + w^3u_3 + w^6u_5 \\
t_4 &= u_0 + w^4u_2 + w^8u_4 \\
t_5 &= u_1 + w^5u_3 + w^{10}u_5 \\
\end{align*}
\]

which requires only 13 complex multiplications and 18 complex additions. Note that \( w^3 = -1 \) and \( w^6 = 1 \).

Such a reduction in computation can be found for any length which is a composite integer. The algebraic proof may be found in the appendix. Also, the various techniques for performing multidimensional transforms, real transforms, etc. are discussed there.

**Special Cautions and Features**

The finite discrete Fourier transform places three restrictions upon the data:

1. The data must form one cycle of a periodic function. Alternately stated, the subscripts are interpreted modulo \( N \).
2. The number of input data and the number of transform values must be the same.
3. The data must be equispaced in each dimension (though, of course, the interval need not be the same for each dimension). Further, if in any dimension the input data are spaced at interval $dt$, the resulting transform values will be spaced from 0 to $2\pi(N-1)/(NdT)$ at interval $2\pi/(NdT)$ as $I$ runs from 1 to $N$. By periodicity, the upper limit is identified with $-2\pi/(NdT)$ and in fact all points above the "foldover frequency" $\pi/(NdT)$ are to be identified with the corresponding negative frequency.

Those familiar with other implementations of the fast Fourier transform may be aware that the order of the data is scrambled in the course of execution. Unscrambling is performed automatically, however, and both the input and output values are placed in ordinary sequential arrangement.

Timing

Let $N_{\text{total}}$ be the total number of points in the data array. That is, $N_{\text{total}} = N_1*N_2*...$. Decompose $N_{\text{total}}$ into its prime factors, such as $2^K_2*3^K_3*5^K_5*...$. Let $\Sigma_2$ be the sum of all the factors of two in $N_{\text{total}}$, that is, $\Sigma_2 = 2^K_2$. Let $\Sigma_f$ be the sum of all the other factors, $\Sigma_f = 3^K_3 + 5^K_5 + ...$. The time taken for a multidimensional transform is

$$T = T_0 + N_{\text{total}} [T_1 + T_2\Sigma_2 + T_3\Sigma_f].$$

For the CDC 3300,

$$T = 3000 + N_{\text{total}} [600 + 40\Sigma_2 + 175\Sigma_f] \text{ microseconds.}$$

The greater optimization apparent for factors of two is due to

1. The eight-fold symmetry of the trigonometric functions from 0 to $2\pi$.
2. The fact that Fourier transforms of length two and four require fewer complex multiplies than transforms of other lengths.

The above timing formula is accurate for complex data.

The use of real data (IFORM = 0) can reduce running time by as much as forty percent. On the CDC 3300, a $64 \times 64$ complex array was transformed in
6.1 seconds; a 64 x 64 real array took 4.2 seconds. A complex array 1500 long took 6.1 seconds, while a real 1500 array ran only 3.4 seconds.

**Accuracy**

The simplistic idea about accuracy is apparently correct: because the fast Fourier transform takes fewer steps in execution, less error creeps in. Gentleman and Sande ([3], 1966) show theoretically that the root-mean-square relative error is bounded by

\[ 1.06 \frac{N_{\text{total}}}{N} 2^{-b} \sum_j [2f_j]^{3/2} \]

where \( b \) is the number of bits in the floating-point fraction and \( f_j \) are the factors of \( N_{\text{total}} \).

Further error is introduced in this particular program by the use of recursive generation of sines and cosines for factors of \( N_{\text{total}} \) other than two. Sines and cosines needed for factors of two are computed precisely. In actual practice, out of eleven and a half digits representable on the CDC 3300, about four were lost on long one-dimensional sequences like 1500 and 4096.

**Applications**

Besides all the direct uses of discrete Fourier transforms in signal processing, lens design, crystallography, seismic studies, etc., Fourier transforms find application in techniques of correlation and convolution. The principal tool here is the convolution theorem. Denoting the convolution of two discrete functions \( f \) and \( g \) by \( f \ast g \)

\[ (f \ast g)_k = \sum_j f_j g_{k-j}, \]

where both \( j \) and \( k \) run from 1 to \( N \) and subscripts are interpreted modulo \( N \), and denoting the discrete Fourier transform of \( f \) by \( F(f) \), the convolution theorem states

\[ F(f \ast g) = F(f) F(g). \]
The difficulties here are that cyclical interpretation of subscripts may not be desirable and that $N$ may not be convenient for fastest processing via the fast Fourier transform. Appendage of zeroes to the ends of the sequences solves both problems. See Stockham ([5], 1966) and Gentleman and Sande ([3], 1966).

Examples of Use

A. FOURT

1. Forward transform of complex $50 \times 40$ array in Fortran II

   ```fortran
   DIMENSION DATA (2,50,40), WORK (100), NN (2)
   NN (1) = 50
   NN (2) = 40
   DO 1 I = 1, 50
   DO 1 J = 1, 40
   DATA (1,I,J) = real part
   1 DATA (2,I,J) = imaginary part
   CALL FOURT (DATA,NN,2,-1,1,WORK)
   ```

2. Same example as 1, but in Fortran IV

   ```fortran
   DIMENSION DATA (50,40), WORK (100), NN (2)
   COMPLEX DATA
   DATA NN/50, 40/
   DO 1 I = 1, 50
   DO 1 J = 1, 40
   1 DATA (I,J) = complex value
   CALL FOURT (DATA,NN,2,-1,1,WORK)
   ```

3. Same example as 2, but in double precision

   Add the following statement:
   `DOUBLE PRECISION DATA, WORK`
   Change the call to:
   `CALL DFOURT (DATA,NN,2,-1,1,WORK)`
4. Inverse transform of real $64 \times 32$ array in Fortran IV

```
DIMENSION DATA (64,32), NN(2)
COMPLEX DATA
DATA NN/64,32/
DO 1 I = 1, 64
   DO 1 J = 1, 32
      1 DATA(I,J) = real value
CALL FOURT (DATA,NN,2,+1,0,0)
```

B. FOUR2

Inverse transform of real $64 \times 32$ array in Fortran IV

```
DIMENSION DATA (64,32), NN(2)
COMPLEX DATA
DATA NN/64,32/
DO 1 I = 1, 64
   DO 1 J = 1, 32
      1 DATA(I,J) = real value
CALL FOUR2 (DATA,NN,2,+1)
```

C. FOUR1

Forward transform of real array of length $2048$ in Fortran II

```
DIMENSION DATA (2,2048)
DO 1 I = 1, 2048
   DATA(1,I) = real part
   1 DATA(2,I) = 0
CALL FOUR1 (DATA,2048,-1)
```

Acknowledgments

The author's interest in the fast Fourier transform was sparked by Thomas Stockham. The original program was written by Charles Rader, and the idea for digit reversal was contributed by Ralph Alter. Additional ideas were gleaned from papers by Langdon and Sande, and Bingham.
BIBLIOGRAPHY


Appendix I

Historical Sketch

In 1903 Runge published schemes for the optimal computation of twelve and twenty-four point Fourier transforms ([6]). They involved grouping and re-grouping of values in a manner similar to the modern FFT. Runge's schemes are well known and appear in many works on numerical analysis, including Runge and König ([7], 1924) and Whittaker and Robinson ([8], 1944). Nevertheless, no one thought of generalizing Runge's ideas until 1942 when Danielson and Lanczos ([2]) published an optimal algorithm for \( N \cdot 2^k \) point transforms. Their paper passed unnoticed.

Meanwhile, in 1937 Yates ([9]) had devised an algorithm for the efficient computation of the interactions of \( 2^n \) factorial experiments. This involves sums of the form

\[ t_j = \sum d_i (-1)^{i_0j_0 + i_1j_1 + \ldots} \]

where \( i_0i_1 \ldots \) and \( j_0j_1\ldots \) are the binary representations of \( i \) and \( j \). Davies et al extended the method to \( 3^n \) experiments ([10], 1954); three years later, Good, in an abstruse paper, extended it to general factorial experiments ([11], 1958). In the same paper, Good devised analogous algorithms for \( N \) point Fourier transforms, where \( N \) is decomposable into mutually prime factors. Cooley and Tukey removed this restriction and clarified Good's argument ([1], 1965). They also wrote what was probably the first computer program to perform FFT.

Cooley and Tukey's paper sparked a resurgence of interest in the Fourier transform. Despite its indispensability in many areas of signal processing, the Fourier transform had long been avoided for reasons of long computation time. The FFT revived interest to such an extent that the IEEE Audio Transactions has devoted an entire issue to it (June 1967) and three groups have proposed implementing it in hardware ([11], 1963; [12], 1967; [13], 1967).
Appendix II

The Mathematics of the Fast Fourier Transform

Mathematical descriptions of the algorithms used in the Fast Fourier Transform subroutines will be published in the near future.

Punched decks for these three subroutines are available from J. J. Fitzgerald, J-105, or from SHARE.
Appendix III

Listing of the Fortran Subroutines

The listings of the three subroutines FOUR1, FOUR2, and FOURT are given on the following pages. All three are written in USASI Basic Fortran, and, as such are compatible with the great majority of Fortran compilers.
SUBROUTINE FOUR1(DATA, NN, ISIGN)

THE COOLEY-TUKEY FAST FOURIER TRANSFORM IN USA1 BASIC FORTRAN

TRANSFORM(J) = SUM(DATA(I)*W**(I-1)*(J-1)) WHERE I AND J RUN FROM 1 TO NN AND W = EXP(ISIGN*2*I*PI/SQRT(1+NN)); DATA IS A ONE-DIMENSIONAL COMPLEX ARRAY (I.E., THE REAL AND IMAGINARY PARTS OF THE DATA ARE LOCATED IMMEDIATELY ADJACENT IN STORAGE, SUCH AS FORTRAN IV PLACES THEM) WHOSE LENGTH NN IS A POWER OF TWO; ISIGN IS +1 OR -1, GIVING THE SIGN OF THE TRANSFORM; TRANSFORM VALUES ARE RETURNED IN ARRAY DATA, REPLACING THE INPUT DATA, THE TIME IS PROPORTIONAL TO N*LOG2(N), RATHER THAN THE USUAL N^2, WRITTEN BY NORMAN BRENNER, JUNE 1987; THIS IS THE SHORTEST VERSION OF FFT KNOWN TO THE AUTHOR; AND IS INTENDED MAINLY FOR DEMONSTRATION, PROGRAMS FOUR2 AND FOURT ARE AVAILABLE THAT RUN THREE TIMES AS FAST AND OPERATE ON MULTIDIMENSIONAL ARRAYS WHOSE DIMENSIONS ARE NOT RESTRICTED TO POWERS OF TWO, (LOOKING UP SINES AND COSINES IN A TABLE WILL CUT RUNNING TIME OF FOUR1 BY A THIRD.)

SEE IEEE AUDIO TRANSACTIONS (JUNE 1987), SPECIAL ISSUE ON FFT,

DIMENSION DATA(1)

J=1
DO 5 I=1, NN
  IF(I+J)2,2,2
  TEMP=DATA(J)
  TEMP1=DATA(I+J)
  DATA(J)=DATA(I+J)
  DATA(I+J)=DATA(J)
  DATA(TEMP)=DATA(TEMP1)
  DATA(TEMP1)=DATA(TEMP)
  IF(M=1)GO TO 6
  M=M/2
  IF(M=2)GO TO 5
  IF(MAX=7)GO TO 7
  IS=Q=MAX
  DO 8 M=1, IS
    THETA=2.1415926535*FLOAT(ISIGN*(M-1))/FLOAT(IS)
    W=cos(THETA)
    W1=sin(THETA)
    DO 6 B=M,NN+1,TB
      J=TB
      TEMP1=DATA(J)*W1*DATA(J-1)
      TEMP=DATA(I+J)*W1*DATA(I)
      DATA(J)=DATA(I+J)
      DATA(I+J)=DATA(J)
      DATA(TEMP)=DATA(TEMP1)
      DATA(TEMP1)=DATA(TEMP)
      IF(M=1)GO TO 6
  8 M=M+1
  RETURN
END
SUBROUTINE FOUR2(DATA, NN, NDIM, SIGN)

THE COOLEY-TUKEY FAST FOURIER TRANSFORM IN BASIC FORTRAN

TRANSFORM(J1, J2, ..., JNDIM) = SUM[DATA(I1, I2, ..., INDIM) * EXP{SOME不仅能填充任何一种输入，还能填充一个表格，或在表格中填写一个值。}]

WHERE I1 AND J1 RUN FROM 1 TO NN(I1) AND W(A) = EXP{SOME不仅能填充任何一种输入，还能填充一个表格，或在表格中填写一个值。} / SQRT(1 / NN(I1)) ETC.

DATA IS A MULTIDIMENSIONAL FLOATING-POINT ARRAY ALL OF WHOSE

DIMENSIONS ARE POWERS OF TWO. THE LENGTH OF EACH DIMENSION IS

STORED IN THE INTEGER ARRAY NN, OF LENGTH NDIM. SIGN IS

+1 OR -1, GIVING THE SIGN OF THE TRANSFORM. THE REAL

AND IMAGINARY PARTS OF A DATUM ARE IMMEDIATELY ADJACENT IN STORAGE

(SUCH AS FORTRAN IV PLACES THEM). TRANSFORM RESULTS ARE RETURNED

IN ARRAY DATA, REPLACING THE ORIGINAL DATA. TIME IS PROPORTIONAL

TO N*LOG2(N) RATHER THAN THE USUAL N^2. NOTE THAT IF A FORWARD

TRANSFORM IS FOLLOWED BY AN INVERSE TRANSFORM, THE ORIGINAL DATA

WILL REAPPEAR MULTIPLIED BY NN(I1)*NN(I2)*... EXAMPLE=

FORWARD FOURIER TRANSFORM OF A TWO-DIMENSIONAL ARRAY IN FORTRAN II

DIMENSION DATA(2,64,32), NN(2)

NN(1) = 64
NN(2) = 32

DO 1 I = 1, 64
DO 1 J = 1, 32
DATA(I, J) = REAL PART
1 DATA(I, J) = IMAGINARY PART
CALL FOUR2(DATA, NN, 2, 1)

SAME EXAMPLE IN FORTRAN IV

DIMENSION DATA(64,32), NN(2)

COMPLEX DATA
DATA NN/64,32/
DO 1 I = 1, 64
DO 1 J = 1, 32
DATA(I, J) = REAL PART
1 DATA(I, J) = IMAGINARY PART
CALL FOUR2(DATA, NN, 2, 1)

PROGRAM BY NORMAN BRENNER FROM THE BASIC PROGRAM BY CHARLES
RADER, MAY 1967; THE IDEA FOR THE DIGIT REVERSAL WAS SUGGESTED
BY RALPH ALDER.

THIS VERSION OF THE FAST FOURIER TRANSFORM IS THE FASTEST KNOWN
TO THE AUTHOR, LOOKING UP SINES AND COSINES IN A TABLE INSTEAD OF
COMPUTING THEM WOULD DECREASE RUNNING TIME SEVEN PERCENT.

PROGRAMS FOUR1 AND FOUR2 ARE AVAILABLE FROM THE AUTHOR THAT ALSO
PERFORM THE FAST FOURIER TRANSFORM AND ARE WRITTEN IN USASI

BASIC FORTRAN. FOUR1 IS THREE TIMES AS LONG, IS NOT RESTRICTED TO

DIMENSIONS AND POWERS OF TWO.

FOUR2 IS ONE FOURTH AS LONG, ONE HALF AS FAST, AND IS RESTRICTED

to one dimension and powers of two.

SEE-- IEEE AUDIO TRANSACTIONS (JUNE 1967), SPECIAL ISSUE ON FFT,
C MAIN LOOP FOR EACH DIMENSION

NP1=2
DO 600 IDIM=1,NDIM
N=NN(IDIM)
NP2=NP1*N
IF(N.eq.1700,600,100)

C SHUFFLE DATA BY bit reverse, since N=2**K, as the SHUFFLING CAN BE DONE BY SIMPLE INTERCHANGE, NO WORKING ARRAY IS NEEDED

100 NP2HF=NP2/2
J=1
DO 160 12=1,NP2,NP1
IF(J.eq.12)110,130,130
110 I1MAX=I12*NP2/2
DO 120 I1=I12,I1MAX,2
DO 120 I3=I3-I1
TEMPJ=DATA(I3)
TEMPI=DATA(I3+1)
DATA(I3+1)=DATA(I3)
DATA(I3)=TEMPJ
DATA(I3+1)=TEMPI
120 UMAX=I1*NPl-2
DO 120 J3=UMAX,J1MAX-2
DO 120 I3=I3-J3MAX,J3
DO 120 J1=I1*J1MAX-2
DO 120 I2=I2-J1MAX,J1
DATA(I2*1)=DATA(J2*1)
DATA(J2)>DATA(I2)
DATA(J2*1)=DATA(I2*1)
DATA(I2)<DATA(J2)
120 M=NP2HF
130 IF(J.eq.160,160,130)
140 IF(J.eq.160,160,130)
150 J=J+2
160 J=J+2

C MAIN LOOP, PERFORM FOURIER TRANSFORMS OF LENGTH FOUR, WITH ONE OF LENGTH TWO IF NEEDED. THE TWIDDLE FACTOR W=EXP(I*SIGN*2*PI)/SORT(I/4*MMAX), CHECK FOR THE SPECIAL CASE W=SIGN*SORT(I/4)/SORT(2), AND REPEAT FOR W=W*(SIGN*SORT(I/4))/SORT(2).

C NP1TW=NP1*NPl
IPAR=NP1
310 IF(IPAR.eq.350,330,320)
320 IF(IPAR.eq.350,330,320)
GO TO 310
330 DO 340 I1=1,NP1,2
340 DO 340 K1=I1,NTOT,NPlTW
K2=K1-NP1
TEMPK=DATA(K2)
TEMPJ=DATA(K2+1)
DATA(K2)=DATA(K1)+TEMPK
DATA(K2+1)=DATA(K1)+TEMPJ
360 DATA(K1)=DATA(K1)+TEMPK
DATA(K1+1)=DATA(K1+1)+TEMPJ
350 MMAX=NP1
360 IF(MMAX.eq.NP2HF,370,600)
370 MMAX=MAX(0,NP1TW/MMAX/2)
DO 570 L=NP1,LMAX,NP1TW
M=L
IF(M.eq.420,420,380)
380 DATA=THOMPSON*(M/NPlTW/FLOAT(4*MMAX))
390  THETA = THETA
400  WR = COS(THETA)
410  WI = SIN(THETA)
420  DO 330 I = 1, NPar
430  KMIN = 1
440  KSTEP = KSTEP + 1
450  IF (MAX(NP1) > 390, 430, 440)
460  UI = DATA(KI) + DATA(K2)
470  U4 = DATA(K4) + DATA(K3)
480  T2R = WR * DATA(K2) + WI * DATA(K1)
490  T3R = WR * DATA(K3) + WI * DATA(K1)
500  GO TO 90
510  DATA(K4) = U4
520  DATA(K3) = U3
530  DATA(K2) = T2R
540  DATA(K1) = T1R
550  DATA(K4) = U4
560  DATA(K3) = U3
570  DATA(K2) = T2R
580  DATA(K1) = T1R
590  IF (MAX(NP2) > 390, 430, 530)
530 CONTINUE
      MMAX=LMAX
      IF (MMAX .LT. 540, 540, 970
540 IF (TMN .LT. 350, 350, 580
550 TEMP=WR
      WR=WR+1
      RTHLF
      WI=WI+1
      TEMP=RTMLF
      GO TO 410
560 TEMP=WR
      WR=WR+1
      RTHLF
      WI=WI+1
      TEMP=RTMLF
      GO TO 410
570 CONTINUE
      IPAR=0=IPAR
      MMAX=MMAX+MMAX
      GO TO 360
600 IF =IPAR
700 RETURN
END
SUBROUTINE FOURT(DATA, NN, DIM, ISIGN, IFORM, WORK)

THE COOLEY-TUKEY FAST FOURIER TRANSFORM IN USASI BASIC FORTRAN

TRANSFORM(J1, J2, ..., ) = SUM(DATA( I1, I2, ..., ) * WN * ((12*17*6*J1-1))

WHERE I1 AND J2 RUN FROM 1 TO NN(1) AND W = EXP(ISIGN * PI * 
SORT(1)/NN(1)); ETC. THERE IS NO LIMIT ON THE DIMENSIONALITY 
(NUMBER OF SUBSCRIPTS) OF THE DATA ARRAY, IF AN INVERSE 
TRANSFORM (ISIGN = 1) IS PERFORMED UPON AN ARRAY OF TRANSFORMED 
(ISIGN = 1) DATA, THE ORIGINAL DATA WILL REAPPEAR;

MULTIPLIED BY NN(1)*NN(2)*... THE ARRAY OF INPUT DATA MUST BE 
IN COMPLEX FORMAT, HOWEVER, IF ALL IMAGINARY PARTS ARE ZERO (I.E., 
THE DATA ARE DISGUISED REAL) RUNNING TIME IS CUT UP TO FORTY PER-
CENT. (FOR FASTEST TRANSFORM OF REAL DATA, NN(1) SHOULD BE EVEN.)

THE TRANSFORM VALUES ARE ALWAYS COMPLEX, AND ARE RETURNED 
in THE ORIGINAL ARRAY OF DATA, REPLACING THE INPUT DATA, THE LENGTH 
OF EACH DIMENSION OF THE DATA ARRAY MAY BE ANY INTEGER.

THE PROGRAM RUNS FASTER ON COMPOSITE INTEGERS THAN ON PRIMES, AND IS 
PARTICULARLY FAST ON NUMBERS RICH IN FACTORS OF TWO.

TIMING IS IN FACT GIVEN BY THE FOLLOWING FORMULA, LET NTOT BE THE 
TOTAL NUMBER OF POINTS (REAL OR COMPLEX) IN THE DATA ARRAY, THAT 
IS, NTOT = NN(1)*NN(2)*... DECOMPOSE NTOT INTO ITS PRIME FACTORS, 
sUCH AS 2**K2 * 3**K3 * 5**K5 * ... LET SUM2 BE THE SUM OF ALL 
THE FACTORS OF TWO IN NTOT, THAT IS, SUM2 = 2**K2, LET SUMF BE 
THE SUM OF ALL OTHER FACTORS OF NTOT, THAT IS, SUMF = 3**K3*5**K5... 
THE TIME TAKEN BY A MULTIDIMENSIONAL TRANSFORM ON THESE NTOT DATA 
IS T = T0 * NTOT* (T1* T2* SUM2* T3* SUMF), ON THE CDC 3300 (FLOATING 
POINT ADD TIME = SIX MICROSECONDS, T = 3000 * NTOT* (600* 40* SUM2* 
179 * SUMF) MICROSECONDS ON COMPLEX DATA;

IMPLEMENTATION OF THE DEFINITION BY SUMMATION WILL RUN IN A TIME 
PROPORTIONAL TO NTOT* (NN(1)*NN(2)*...), FOR HIGHLY-COMPOSITES NTOT 
THE SAVINGS OFFERED BY THIS PROGRAM CAN BE DRAMATIC, A ONE-DIMEN-
SIONAL ARRAY 4000 IN LENGTH WILL BE TRANSFORMED IN 4000*400* 
40* (2*2*2*2)*175 = 140, SECONDS CUMULATIVE ABOUT 4000* 
4000*175 = 2600 SECONDS FOR THE STRAIGHTFORWARD TECHNIQUE;

THE FAST FOURIER TRANSFORM PLACES THREE RESTRICTIONS UPON THE 
DATA;
1. THE NUMBER OF INPUT DATA AND THE NUMBER OF TRANSFORM VALUES 
MUST BE THE SAME;
2. BOTH THE INPUT DATA AND THE TRANSFORM VALUES MUST REPRESENT 
eQUISPACED POINTS IN THEIR RESPECTIVE DOMAINS OF TIME AND 
FREQUENCY, CALLING THESE SPACINGS DELTAT AND DELTAF, IT MUST BE 
TRUE THAT DELTAF = 2*PI/(NN(1)*DELTAT), OF COURSE, DELTAT NEED NOT 
BE THE SAME FOR EVERY DIMENSION;
3. CONCEPTUALLY AT LEAST, THE INPUT DATA AND THE TRANSFORM OUTPUT 
REPRESENT SINGLE CYCLES OF PERIODIC FUNCTIONS;

THE CALLING SEQUENCE IS=

CALL FOURT(DATA, NN, DIM, ISIGN, IFORM, WORK)

DATA IS THE ARRAY USED TO HOLD THE REAL AND IMAGINARY PARTS 
OF THE DATA ON INPUT AND THE TRANSFORM VALUES ON OUTPUT, IT 
IS A MULTIDIMENSIONAL FLOATING POINT ARRAY, WITH THE REAL AND 
IMAGINARY PARTS OF A DATUM STORED IMMEDIATELY ADJACENT IN STORAGE 
(SUCH AS FORTRAN IV PLACES THEM), NORMAL FORTRAN ORDERING IS
EXPECTED, THE FIRST SUBSCRIPT CHANGING FASTEST, THE DIMENSIONS ARE GIVEN IN THE INTEGER ARRAY NN; OF LENGTH NDIM, I SIGN IS +1 TO INDICATE A FORWARD TRANSFORM (EXPONENTIAL SIGN IS +1) AND -1 FOR AN INVERSE TRANSFORM (SIGN IS +1, IF Form IS *1 IF THE DATA ARE COMPLEX, 0 IF THE DATA ARE REAL, IF IT IS 0, THE IMAGINARY PARTS OF THE DATA MUST BE SET TO ZERO, AS EXPLAINED ABOVE, THE TRANSFORM VALUES ARE ALWAYS COMPLEX AND ARE STORED IN ARRAY DATA, WORK IS AN ARRAY USED FOR WORKING STORAGE, IT IS FLOATING POINT REAL, ONE DIMENSIONAL OF LENGTH EQUAL TO TWICE THE LARGEST ARRAY DIMENSION NN(I) THAT IS NOT A POWER OF TWO, IF ALL NN(I) ARE POWERS OF TWO, IT IS NOT NEEDED AND MAY BE REPLACED BY ZERO IN THE CALLING SEQUENCE, THIS, FOR A ONE-DIMENSIONAL ARRAY, NN(1) ODD, WORK OCCUPIES AS MANY STORAGE LOCATIONS AS DATA, IF SUPPLIED, WORK MUST NOT BE THE SAME ARRAY AS DATA, ALL SUBSCRIPTS OF ALL ARRAYS BEGIN AT ONE,

EXAMPLE 1, THREE-DIMENSIONAL FORWARD FOURIER TRANSFORM OF A COMPLEX ARRAY DIMENSIONED 32 BY 25 BY 13 IN FORTRAN IV,

DIMENSION DATA(32,25,13), WORK(59) = NN(3)

REAL DATA
DATA NN/32,25,13/
DO 1 I=1,32
DO 1 J=1,25
DO 1 K=1,13
1 DATA(I,J,K) = REAL VALUE
CALL FOURT(DATA,NN=3, W0RK)

EXAMPLE 2, ONE-DIMENSIONAL FORWARD TRANSFORM OF A REAL ARRAY OF LENGTH 64 IN FORTRAN II,

DIMENSION DATA(2,64)
DO 2 IP=1,64
DATA(2,IP) = REAL PART
2 DATA(2,IP) = REAL PART
CALL FOURT(DATA,64,IP=1,IP=0)

THERE ARE NO ERROR MESSAGES OR ERROR HALTS IN THIS PROGRAM, THE PROGRAM RETURNS IMMEDIATELY IF NDIM OR ANY NN(I) IS LESS THAN ONE,

PROGRAM BY NORMAN BRENNER FROM THE BASIC PROGRAM BY CHARLES RADER, JUNE 1967, THE IDEA FOR THE DIGIT REVERSAL WAS SUGGESTED BY RALPH ALTER,

THIS IS THE FASTEST AND MOST VERSATILE VERSION OF THE FFT KNOWN TO THE AUTHOR, A PROGRAM CALLED FOUR2 IS AVAILABLE THAT ALSO PERFORMS THE FAST FOURIER TRANSFORM AND IS WRITTEN IN USASI BASIC FORTRAN, IT IS ABOUT ONE THIRD AS LONG AND restricts THE DIMENSIONS OF THE INPUT ARRAY WHICH MUST BE COMPLEX) TO BE POWERS OF TWO, ANOTHER PROGRAM, CALLED FOUR1, IS ONE TENTH AS LONG AND RUNS TWO THIRDS AS FAST ON A ONE-DIMENSIONAL COMPLEX ARRAY WHERE LENGTH IS A POWER OF TWO,

REFERENCE--
IEEE AUDIO TRANSACTIONS (JUNE 1967), SPECIAL ISSUE ON THE FFT,

DIMENSION DATA(1), NN(1), FACT(32) = W0RK(1)
TWOPI=6.283185307
RTMLF=7.70710 67812
IP = NDIM-1
S01=1
1 N07=2
DO 2 IP=1,NDIM
IF( NN(IP)=1) NDIM
2 NTOT=NT0T*NN(IP)

21
MAIN LOOP FOR EACH DIMENSION

DO 900 IDIM=1,NDIM

N=N(IDIM)
NP1=N+1
IP(N-1)=920,900,5

IS N A POWER OF TWO AND IF NOT, WHAT ARE ITS FACTORS

MN

NTWO=NP$5
IF=1
IDIV=2
IQUOT=M/IDIV
INREM=IQUOT
IF(INREM)/DIV=11,11
IF(INREM)/DIV=2,2,20
NTWO=NTWO×NTWO
IFACT(IF)=IDIV
IF=IF-1
M=IQUOT
GO TO 10

IDIV=3
INREM=IF
IQUOT=M/IDIV
INREM=IQUOT
IF(INREM)/DIV=0,34,31
IF(INREM)/DIV=40,40,40
IFACT(IF)=IDIV
IF=IF-1
M=IQUOT
GO TO 30

IDIV=IDIV/2
GO TO 50

ITWO=ITWO×ITWO
GO TO 70

SEPARATE FOUR CASES=

1, COMPLEX TRANSFORM OR REAL TRANSFORM FOR THE 4TH, 8TH, 16TH, ...

DIMENSIONS,

2, REAL TRANSFORM FOR THE 2ND OR 3RD DIMENSION, METHOD=

TRANSFORM HALF THE DATA, SUPPLYING THE OTHER HALF BY CONJUGATE SYMMETRY,

3, REAL TRANSFORM FOR THE 1ST DIMENSION, N ODD, METHOD=

SET THE IMAGINARY PARTS TO ZERO,

4, REAL TRANSFORM FOR THE 1ST DIMENSION, N EVEN, METHOD=

TRANSFORM A COMPLEX ARRAY OF LENGTH N/2 WHOSE REAL PARTS ARE THE EVEN NUMERATED REAL VALUES, AND WHOSE IMAGINARY PARTS ARE THE ODD NUMBERED REAL VALUES; SEPARATE AND SUPPLY THE SECOND HALF BY CONJUGATE SYMMETRY,

ITWO=ITWO×ITWO
N=IDIM+1,100,100
SHUFFLE DATA BY BIT REVERSAL: BIDGE N=3K, AS THE SHUFFLING CAN BE DONE BY SIMPLE INTERCHANGE, NO WORKING ARRAY IS NEEDED

100 IF(NTHN1[NP1]=NP1[NP1])
110 NP1[NP1]=NTHN1[NP1]
200 IF(J=J+1)
300 IF(TEMP=DATA[J])
400 IF(INV=1)
500 IF(INV=1)
600 IF(INV=1)

SHUFFLE DATA BY DIGIT REVERSAL FOR GENERAL N

200 NWORK=N
210 WORK(I)=DATA[J]
220 WORK(I)=DATA[J]
230 IF(IFM=1)
240 IF(IFM=1)
```
250 JM=IFP2
    IFP2=IFP1
    IF#IF=1
    IF(IFP2#NP1)260,360,240
260 CONTINUE
    I2MAX=3*NP2#NP1
    IF
    DO 270 I=1,12MAX,12MAX#NP1
        DATA(I)=WORK(I)
        DATA(I+12)=WORK(I+1)
    270 I=1
    MAIN LOOP FOR FACTORS OF TWO: PERFORM FOURIER TRANSFORMS OF
    LENGTH FOUR. WITH ONE OF LENGTH TWO IF NEEDED; THE 'MIDDLE' FACTOR
    W#NP1(ISIGN#2#NP1=SORT(=1)*M/(4#MAX)), CHECK FOR I=SIGN=SORT(=1)
    AND REPEAT FOR #W#I=SIGN#SORT(=1)SORT(=2).
    C
    C
    300 IF(NW#NP1)360,600,309
    309 NW=#NP2#NP1
    IPAR=NW#NP1
    310 IF(IPAR#2)350,330,320
    320 IPAR=IPAR/4
    GO TO 310
    330 DO 340 I=1,11#NW,2
        DO 340 K=1,11#NTOT#NP1#NW
            KP=K#NP1
            TEMPR=DATA(K)
            DATA(K)=DATA(KP+1)
            DATA(KP+1)=TEMPR
        340 K=1
    350 LH#MAX#NP1
    360 IF(M#MAX#NP1(2)370,600,600
    370 LM#MAX#NP1#NW
    WM
    380 TM#NP1#NP1(2)390,420,360
    390 I=SIGN=THETA
    400 WR=COS(THETA)
    410 WI=SIN(THETA)
    420 DO 530 I=1,11#NW,2
        DO 530 K=1#KM#NP1(2)430,430,440
            KDIF=IPAR#KM
            IF(K#NP1(2)460,460,530
            DO 520 K1#KMIN#NTOT#KSTEP
                DO 520 K1#K#KDIF
                    DO 520 K1#K#KDIF
                        K2=K1+KDIF
                        K3=K2+KDIF
                        IF(K#NP1(2)470,470,480
                510 U1=DATA(K1)*DATA(K2)
                          U2=DATA(K1)*DATA(K2+1)
                          U3=DATA(K3)*DATA(K4)
```

U21 = DATA(K3+1) + DATA(K4+1)
U3R = DATA(K1) + DATA(K2)
U3I = DATA(K1+1) + DATA(K2+1)
IF(ISIGN) 471, 472, 472
471 U4R = DATA(K3+1) + DATA(K4+1)
U4I = DATA(K4) + DATA(K3)
GO TO 500
472 U4R = DATA(K4+1) + DATA(K3+1)
U4I = DATA(K3) + DATA(K4)
GO TO 500
T2R = W2R + DATA(K2) + W2R*DATA(K2)
T2I = W2R + DATA(K2+1) + W2R*DATA(K2)
T3R = W2R + DATA(K3) + W2R*DATA(K3)
T3I = W2R + DATA(K3+1) + W2R*DATA(K3)
T4R = W2R + DATA(K4) + W2R*DATA(K4)
T4I = W2R + DATA(K4+1) + W2R*DATA(K4)
U1R = DATA(K1) + T2R
U1I = DATA(K1+1) + T2I
U2R = T3R + T4R
U2I = T3I + T4I
U3R = DATA(K1) + T2R
U3I = DATA(K1+1) + T2I
IF(ISIGN) 490, 500, 500
490 U4R = T3I + T4I
U4I = T4R + T3R
GO TO 520
500 U4R = T4I + T3I
U4I = T3R + T4R
GO TO 520
510 DATA(K1) = U1R + U2R
DATA(K1+1) = U1I + U2I
DATA(K2) = U3R + U4R
DATA(K2+1) = U3I + U4I
DATA(K3) = U1R + U2R
DATA(K3+1) = U1I + U2I
DATA(K4) = U3R + U4R
DATA(K4+1) = U3I + U4I
KDF = KSTEP
KMIN = 4*(KMIN-1)+1
GO TO 500
530 CONTINUE
M8H = LMAX
IF(M = MMAX) 540, 540, 570
540 IF(ISIGN) 550, 560, 560
550 TEMPR = WR
WR = (WR+1) + RTHLF
WT = (WT+TEMPR) + RTHLF
GO TO 410
560 TEMPR = WR
WR = (WR+1) + RTHLF
WT = (TEMPR+1) + RTHLF
GO TO 410
570 CONTINUE
IPAR = 3*IPAR
MAX = MMAX = MMAX
GO TO 360
C MAIN LOOP FOR FACTORS NOT EQUAL TO TWO, APPLY THE TWIDDLE FACTOR
WREXP(I S I G N) = P + (SORT(1) = (U2Z+1) + (22=1) + (FP1+1) + (IPP2), THEN
PERFORM A FOURIER TRANSFORM OF LENGTH IFACT(IF), MAKING USE OF
CONJUGATE SYMMETRIES
600 IF(NTW = NIP2) 605, 700, 700
IF=INONZ
NP1=NP1/2
610 IFP2=IFACT(IF)*IFP4
JMIN=NP1/2
615 IF(JMIN=IFP1)619,615,640
619 DO 635 J=1,JMIN,JFP4,1
THETA=TWOPI/FLOAT(J1)/2/FLOAT(IFP2)
JF(I SIGN)629*620,680
620 THETA=THETA
625 WSTPRE=COS(THETA)
626 WSTPI=SIN(THETA)
W=NSTPI
JMIN=J1*IFP1
JMAX=J1*IFP2=IFP1
1HAX=J2=I1RNO=2
630 DO 635 J=1,JMIN,JMAX,1
THETA=TWOPI/FLOAT(1F)*1F)
640 THETA=THETA
650 WSTPRE=COS(THETA)
651 WSTPI=SIN(THETA)
J2RNO=IFP1*(1+FACT(IF)/2)
655 SUMR=0;
SUMR=0;
660 SUMR=SUMR+DATA(J)
SUH=SUH+DATA(J)
WORK(I)=SUMR
WORK(I)=SUM
DO TO 680
665 JONJL=1*IFP2=2=J2=13=J3J/NP5HF
JMAX=J2=I1RNO=2
SUH=DATA(J)
SUH=DATA(J)
OLDSR=0
OLDSR=0
UM=1*IFP5
670 TEMPR=SUMR
COMPLETE A REAL TRANSFORM IN THE $1^{st}$ DIMENSION, N EVEN, BY CONJUGATE SYMMETRIES,

GO TO 700

C

CASE NHALF=N

THETA=THETA/FLOAT(N), IF(ISIGN)703,702,702

THETA=THETA

HSTPR=HSTPR, WSTPR=WSTPR, IMIN=3

JMIN=2*NHALF+1

GO TO 729

DO 700 JMIN,JMAX=INT(N),N+1

SUMR=(DATA(J)+DATA(J+1))/2,

SUMI=(DATA(J)+DATA(J+1))/2,

DPR=(DATA(J)+DATA(J+1))/2,

DPI=(DATA(J)+DATA(J+1))/2,

TEMPR=SUMR*HSTPR

TEMPR=SUMR*WSTPR

DATA(J+1)=DPR*TEMPR

DATA(J+1)=DPI*TEMPR

DATA(J+1)=DPR*TEMPR

DATA(J+1)=DPI*TEMPR
COMPLETE A REAL TRANSFORM FOR THE 2ND OR 3RD DIMENSION BY
CONJUGATE SYMMETRIES.

IF(1*IMIN)1305,900,900
DO 830 13=1,NT0T,NP2
2MAX=13*NP2
830 GO TO 880

IMIN=IMIN+2

IF(J=1)500,500,500
DATA(I)=DATA(J)
DATA(J)=DATA(I)
GO TO 790

DATA(I)=DATA(J)
DATA(J)=DATA(I)
GO TO 780

DATA(I)=DATA(1)+DATA(2)
GO TO 800
END OF LOOP ON EACH DIMENSION

900 NR1=NP1
   NR1=NP2
910 NPREV=N
920 RETURN
END
Three programs are described and listed, all written in USASI Basic Fortran, which perform the discrete Fourier transform upon a multidimensional array of floating point data. The data may be either real or complex, with a savings in running time for real over complex. The transform values are always complex and are returned in the array used to carry the original data. The running time is much shorter than that of any program performing a direct summation, even when sine and cosine values are precalculated and stored in a table. For example, on a CDC 3300 with floating point add time of six microseconds, a complex array of size 80 x 80 can be transformed in 19.2 seconds. Besides the main array, only a working storage array of size 160 need be supplied.