DECUS
PROGRAM LIBRARY

DECUS NO.	8-144
TITLE	FFTS-C - A FAST FOURIER TRANSFORM SUBROUTINE FOR COMPLEX DATA
AUTHOR	James Rothman
COMPANY	
DATE	August 7, 1968
SOURCE LANGUAGE	

FFTS-C - A FAST FOURIER TRANSFORM SUBROUTINE FOR COMPLEX DATA

DECUS Program Library Write-up | DECUS No. 8-144

1. ABSTRACT

The Fast Fourier Transformation enables computation of the power spectrum of a time series in a minimum of time. Specifically, it reduces the number of computations required to calculate the Discrete Fourier Transformation

$$S_j = \frac{1}{N}\sum_{k=0}^{N-1} X_k W^{kj} \quad (W = e^{-2\pi i/N},\ i = \sqrt{-1})$$

of a series of N equally time spaced samples $X_0, X_1, \ldots X_{N-1}$ where N is a power of $2 (N=2^n)$. In fact, for 1024 time samples, computation time is reduced by over 99%.

FFTS-C (for Fast Fourier Transformation Subroutine) will transform up to 1024 complex points. It is written as a subroutine, and is I/O independent. The user must tailor his own input-output procedure to his particular environment.

2. REQUIREMENTS

2.1 Hardware

A 4K PDP-8 with Extended Arithmetic Element Type 182 or a PDP-8/I with EAE Type KE-8/I option is the minimum necessary hardware.

2.2 Storage

FFTS requires locations 3 to 12, 20 to 55, 400 to 1577+N, and 3600 to 3577+N, where N is the octal number of points being transformed.

3. LOADING PROCEDURE

Make sure the BIN Loader is in core. If not, load it. Put 7777 in the SR. Press Load Address. Place FFTS on the reader and turn the reader on. Press start, and FFTS will load. Then load the user's program the same way as above and start it.

4. USAGE

4.1 Calling Sequences

FFTS enables the user to take either the Fast Fourier Transform, (FFT) or its inverse (IFFT) of a complex time series. The subroutine FFT, which begins at 0400, calculates the FFT. Register DOFFT (normal location 0043) points to FFT, so a JMS I DOFFT (=4443) will call FFT. The subroutine IFFT beginning at 0756 takes the inverse FFT. Since location DOIFFT (normally 0044) points to IFFT, IFFT can be executed simply by writing JMS I DOIFFT (=4444). Both FFT and IFFT assume that the complex data

to be handled has already been stored in memory (see sections 5 and 6.2). After the operation is complete, the results will be stored in memory in bit inverted order (see section 5.1). For FFT, the results are the complex co-efficients S_j (with the appropriate scale factors, as described in section 5.2) given by the equation in section 1 (j=0, 1,...,N-1). For IFFT the results consist of a time sequence X_j (j=0,1,....,N-1).

An example of a program that will transform a time series and then resynthesize the series from its spectrum is as follows: (see sections 5 and 6).

```
*2ØØ
/INPUT DATA AND ZERO IMAGINARY PARTS
BEGIN,
.
.
.
/SERIES STORED AWAY
     JMS I DOFFT        /TAKE FFT
     TAD SCALE          /GET SCALE FOR TRANSFORM
     DCA SCALT
     JMS I SORT         /RE-ORDER THE TRANSFORMS
     JMS I DOIFFT       /TAKE THE INVERSE
     TAD SCALE          /GET SCALE ON INVERSE
     TAD SCALT
     DCA SCALE          /RESULTS =[(ORIGINAL DATA)*2↑(SCALE)]/N
/OUTPUT RESULTS (NOW STORED IN BIT REVERSED ORDER)
.
.
.
END, JMP BEGIN          /START AGAIN
SCALT, Ø

DOFFT =   43
SORT =    37
DOIFFT=   44
SCALE =   5Ø
$
```

NOTE: <u>THE REMARKS IN THE FOLLOWING SECTIONS APPLY TO IFFT AS WELL AS FFT</u>.

4.2 Execution Times

The following is a table of execution times for the subroutine.

Number of points transformed	Time (seconds)
1024	4.47

Number of points transformed	Time (Seconds)
512	1.96
256	.845
128	.357

5. DETAILS OF STORAGE

5.1 Data Storage

A JMS I DOFFT causes a complex time series to be Fourier Transformed. That series is stored in sequential order in memory. More explicitly, the real parts of the data are stored sequentially after location XRTAB (=1600) and the imaginary parts are placed after location XITAB (=36ØØ). For example, the storage scheme for a N=4 point transform would look as follows:

```
               *16ØØ
XRTAB,          RE(X0)        /RE( ) DENOTES REAL PART.
                RE(X1)
                RE(X2)
                RE(X3)
               *36ØØ
XITAB,          IM(X0)        /IM( ) DENOTES IMAGINARY PART
                IM(X1)
                IM(X2)
                IM(X3)
```

On exit the results of the transformation will be in core. The real parts of the transforms (Fourier co-efficients) are stored in the registers following XRTAB, and the imaginary parts are stored in the locations following XITAB. But the transforms are stored in bit reversed order. This means that to find S_j, say, the order of the bits of j, written in binary, must be reversed. For example, to locate S_5 in memory after a 16 point (N=16, n=4) transformation has been completed, first write j=5 as a binary number of n=4 bits: $j=\text{Ø1Ø1}_2$ and then reverse the order of the bits, giving 1Ø1Ø_2, which is $\text{1Ø}_{\text{1Ø}}$. This means the real part S_5 is stored in the position where $X_{1Ø}$ was originally placed. In memory this is location XRTAB+9. Because the user can save time by fetching the transforms for output from bit reversed order, the subroutine does not bother to re-shuffle them in memory before exiting. However, a subroutine SORT that reshuffles the co-efficients is provided, and may be called by a JMS I SORT (SORT=37).

5.2 Data Scaling

All calculations in FFTS are done with single precision fixed point signed binary fractions. The binary point is located between bit Ø and bit 1, leaving an 11 bit signed mantissa. Bit Ø is used as a sign bit. Negative numbers are formed by taking the two's complement of the positive binary fraction. So all inputs must be scaled in magnitude to less than one. The outputs are

also formatted as above. There is also a more subtle scale factor involved. In order to utilize the maximum number of bits in the transformation it is sometimes necessary to divide by 2 in a computation. As a result of this a pseudo floating point format has been adopted in which a variable scale factor (or exponent) is imposed on all the Fourier co-efficients. This scale factor or pseudo exponents is found in register SCALE (=5Ø) after each transform has been completed. The numbers stored in memory are the Fourier co-efficients multiplied by 2 raised to the contents of SCALE. So to retrieve the co-efficients themselves, merely shift each number C(SCALE) places right. If any further computations are to be done, better accuracy will be obtained by retaining the pseudo exponent and leaving the co-efficients in "normalized form." In the case of the inverse transform, the desired results (here time samples) are the numbers stored in memory times 2↑(-C(SCALE)). In the program example of section 4.1 the scale factors after the transform and the inverse are saved, and later added. This is necessary because the inverse routine calls the transform routine, which adopts a "floating" point format. Hence the results of the inverse of the transform have to be scaled by the sums of the separate scaling factors.

6. RESTRICTIONS

6.1 Program Initialization

Because FFT is a subroutine certain registers must be primed before the first entry in order to insure proper operation. Specifically, register N (location ØØØ3) must contain the number of points being transformed (in octal, of course) and register NU (location ØØØ4) must contain the power of two which N is, that is, C(N)=2↑C(NU).C(NU) must be at least 2 and no more than 12_8, due to memory limitations.

6.2 Data Storage

FFT assumes that the complex time series has been stored in memory. So the locations after XRTAB must contain these time samples (actually, their real parts). If the time series is complex valued, the imaginary parts must be stored in the registers after XITAB. If the series is only real valued, then the N-1 registers after XITAB must be set to zero. The reason FFT does not do this itself is simply because that would not allow for the possibility of a complex transformation.

6.3 Input Restrictions

So as to prevent overflow of the single precision storage, it is absolutely necessary that <u>all data be less than 1 in magnitude</u>, subject to the <u>format described in section 5.2</u>. (The binary point is to the right of bit Ø).

7. METHODS

7.1 Algorithm

FFTS uses the algorithm discovered by Cooley and Tukey for the rapid computation of a spectrum. This algorithm, called the Fast Fourier Transformation (or FFT), permits transformation of N (which must be an integer power of 2) equally spaced time samples in a time proportional to $N\log_2 N$, whereas previous methods required times proportional to N^2. This gives a reduction of $1-\log_2 N/N$. For N=1024, this is over 99%. In essence, the algorithm makes use of the fact that

$$W^k = W^{(k \bmod N)}$$

(where $W=e^{-2\pi i/N}$) to reduce the number of multiplications necessary for a transformation. A complete description and proof of the algorithm used and its implementation can be found in an article by James Rothman which appears in DECUSCOPE, Volume 7, Number 3.

8. DETAILS OF OPERATION

The following is a list of useful subroutines and their operations: (values of the symbols may be found in the symbol table included in this document).

Name	Call By	Function
FFT	JMS I DOFFT	Takes the Fourier Transformation of the data buffer. Results in bit reversed order.
IFFT	JMS I DOIFFT	Takes the Inverse Fourier Transformation of the data buffer. Results in bit reversed order.
SORTX	JMS I SORT	Sort the data buffer so that it is in normal sequence.

Name	Call by	Function
TRIGET	JMS I GETRIG	Fetches sine and cosine values. Specifically, if the AC=K on entry, the values of sin ($2\pi K/N$) and cos ($2\pi K/N$) are fetched from an internal trig table. K must be $\leq$N/2. A register COSINE contains the cosine value and the AC contains the sine value on exit.
INVRT	JMS I INVERT	Number in AC is bit reversed and the result is in the AC on exit.
MULTIP	JMS I MULT	Rounded single precision signed multiply. Uses EAE. AC=multiplier. C(Call address + 1)=address of multiplicand. Result in AC on exit.

9. SYMBOL TABLE:

A symbol table follows:

SYMBOL TABLE

ADDER	0036
ADDR	1134
ADDWDS	1156
ADD1	1173
ADD2	0030
ADJSGN	0567
ARG2	1017
ASR	7415
BIGSNU	0012
BUILD	0543
C	0027
CAM	7621
CCIA	0767
CHKPT	0514
CNOP	0770
CNOTS	0676
COSINE	0033
DATAHI	5600
DOFFT	0043
DOIFFT	0044
DVI	7407
F	0007
FFT	0400
FLIP	1044
FLIPCT	1060
GETRIG	0042
GI	0035
GR	0034
IFFT	0756
INDEX	1133
INVERT	0040
INVRT	1036
K	0026
L	0005
LOOP1	0440
LSR	7417
MAXNU	0011
MNOVR2	0012
MQA	7501
MQL	7421
MULT	0041
MULTIP	1000
MUY	7405
N	0003
NMI	7411
NOROT	0563
NOTNOR	1171
NOVER4	0010
NO4MIK	1132
NU	0024
P	0025
PI	0023
PR	0022

SYMBOL TABLE

Q	0024
QI	0021
QR	0020
QUAD1	1110
QUAD2	1072
RBUILD	0700
RECHK	0702
RESETC	0701
REVERS	0707
S	0006
SCA	7441
SCALE	0050
SCL	7403
SETC	0541
SGNADJ	0766
SHFCHK	0052
SHFLAG	0051
SHFT1	1077
SHFT2	1114
SHFT3	1125
SHIFCT	0561
SHIFT1	0053
SHIFT2	0054
SHIFT3	0055
SHL	7413
SIGN	1035
SINE	0032
SINLOC	0045
SINRET	1122
SINTAB	1175
SQRT	0037
SQRTX	0703
SWAPED	0747
TEMPR	0031
TRIGET	1061
WORD	1056
WORDP	1057
XITAB	3600
XLOCDF	0047
XRLOC	0046
XRTAB	1600
XSUM	1174

ADDENDUM TO 8-143 and 8-144

The program was structured so to make the change of eliminating the EAE requirement with a minimum of effort.

All that need be done is replace each EAE instruction with a subroutine that performs the given operation using a pseudo multiplier-quotient. For this purpose the EAE simulator may be used. This does not allow certain microcodes, and where these occur in the FFT program, they can be separated into groups of EAE instructions, all of which together perform the designated function.

For example CLA MQL MUY (microcoed) could become the three instructions:

CLA
MQL
MQA.

CORRECTION TO DECUS NO. 8-143 AND 8-144

	ORIGINAL		CORRECTED	CHANGE
	*1ØØØ		*1ØØØ	
MULTIP,	Ø		MULTIP, Ø	
	.		.	
	.		.	
	.		.	
	RAL		RAR	*
	DCA SIGN		DCA SIGN	
	MUY		MUY	
ARG2,	HLT	ARG2,	HLT	
	SHL		SHL	
	Ø		Ø	
	DCA ARG2		DCA ARG2	
	SHL		TAD SIGN	*
	Ø		SHL	*
	MQL		Ø	*
	TAD SIGN		TAD ARG2	*
	CLL RAR		SPA	*
	TAD ARG2		CLA CLL CMA RAR	*
	MQA		NOP	*
	SZL		SZL	
	CMA IAC		CMA IAC	
	JMP I MULTIP		JMP I MULTIP	
SIGN,	Ø	SIGN,	Ø	

The error was in the way in which rounding was accomplished. This fix was tested by performing a DOFFT, SORT, DOIFFT, SORT sequence on a 512 point real valued time series with 8-144 and then summing the absolute value of the imaginary residuals. The fix above reduced the sum by 40 percent.