DECUS
PROGRAM LIBRARY

DECUS NO.	8-143
TITLE	FFTS-R - A FAST FOURIER TRANSFORM SUBROUTINE FOR REAL VALUED FUNCTIONS
AUTHOR	James E. Rothman
COMPANY	Digital Equipment Corporation Maynard, Massachusetts
DATE	August 12, 1968
SOURCE LANGUAGE	

FFT-R - A FAST FOURIER TRANSFORM SUBROUTINE FOR REAL VALUED FUNCTIONS

DECUS Program Library Write-up

DECUS No. 8-143

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

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

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

FFTS-R (for Fast Fourier Transformation Subroutine) will transform up to 2048 real 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 7, 20 to 107, and 400 to 2401+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 real valued time series. The subroutine FFT, which begins at 0400, calculates the FFT. Register DOFFT (location 0060) points to FFT, so a JMS I DOFFT (=4460) will call FFT. The subroutine IFFT beginning at 0076 takes the inverse FFT. Since location DOIFFT (normally 0061) points to IFFT, IFFT can be executed simply by writing JMS I DOIFFT (=4461). Both FFT and IFFT assume that the real data to be handled

has already been stored in memory (see section 5). 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).

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)
2048	4.95
1024	2.20
512	.963
256	.417
128	.177
64	.074

5. DETAILS OF STORAGE

5.1 Data Storage

A JMS I DOFFT causes a real time series to be Fourier transformed. That series is stored in memory. More explicitly, the data is stored sequentially after location XRTAB (=2400). For example, the storage scheme for a N=8 point transformation would be as follows (X_i is the i^{th} time sample):

```
        *24ØØ
XRTAB,  X0
        X1
        X2
        X3
        X4
        X5
        X6
        X7
```

On exit the results of the transformation will be in core. Only half of them, however, will actually be present. This is because the program makes use of a Hermitian symmetry in the frequency domain to save both time and storage space. The symmetry is as follows:

If the time sequence $X_0, X_1, \ldots X_{N-1}$ is real valued, then the pair of Fourier co-efficients S_j and S_{N-j} obey a complex conjugate symmetry. That is $S_j = S_{N-j}^*$ where '*' denotes taking the complex conjugate. So due to this symmetry, only one half of the co-efficients need actually be computed, since either half is derivable from the other. Hence FFTS computes only S_0 through $S_{N/2}$, introducing a time and space saving factor of 2, yet sacrificing no information.

After execution the set of complex numbers $\{S_0, \ldots, S_{N/2}\}$ are to be found in memory. Unlike the original data set $\{X_0, \ldots, X_{N-1}\}$ they are not in sequential order, but rather in something called bit inverted order. Bit inversion means simply the process of re-ordering the bits in a binary number. For instance, the binary number 001 bit inverted is just 100 (=4). Thus to locate the Fourier co-efficients S_j $(j \leq N/2)$, write j as a binary number of $n-1(=\log_2(\frac{N}{2}))$ bits, bit invert j, and look in that position. For example, to locate S_2 in memory for an 8 point transformation (N=8, n=3, n-1=2) write 2 as a binary number of n-1=2 bits, $2_{10} = 10_2$. Then reverse the order of these bits giving $01 = 1_{10}$. This means that S_2 is stored in position 1. Physically, then, S_2 is to be found in location 2400 + 2 (1). The reason that bit inverted j (=1) is multiplied by 2 is that each S_j is complex, so two locations are required to store it - one for the real part, the other for the imaginary one. FFTS adopts the following format: the imaginary part of a number is stored in the register immediately following the real part. As a specific example, the storage layout for the co-efficients of an 8 point transform is written out below:

```
        *24ØØ
XRTAB,  RE(S0)      / RE ( ) MEANS REAL PART
        IM(S0)      / IM ( ) MEANS IMAGINARY PART
        RE(S2)
        IM(S2)
        RE(S1)
        IM(S1)
        RE(S3)
        IM(S3)
        RE(S4)
        IM(S4)
```

$S_{N/2}$ (here S_4) always is placed last. If $\tilde{j}$ denotes bit inverted j, then a general formula for locating the real and imaginary parts of S_j is (LOC ( ) denotes location of):

$$LOC\ (RE(S_j))=2400+2\tilde{j}$$
$$LOC\ (IM(S_j))=2401+2\tilde{j} = LOC\ (RE(S_j))+1$$

where j is written out as a binary number of n-1 bits. A subroutine INVRT (location 1036) has been provided to do bit inversion. It can be called by a JMS I INVERT (INVERT=55). See section 8.

In addition, a subroutine SORTX has been included which sorts the co-efficients and leaves them in sequential order. It can be called by a JMS I SORT (SORT=54). If SORTX were called after an 8 point transform had been completed, the data buffer would look like this:

```
            *24ØØ
XRTAB,       RE(S0)
             IM(S0)
             RE(S1)
             IM(S1)
             RE(S2)
             IM(S2)
             RE(S3)
             IM(S3)
             RE(S4)
             IM(S4)
```

The reason that the co-efficients are not automatically sorted is that time can be saved by outputing from bit inverted order, and this possibility should be allowed for.

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 exponent is found in register SCALE (=66) 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↑(n-C(SCALE)).*

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 M (location ØØ2Ø) must contain the number of points being transformed (in octal, of course) and register MU (location ØØ21) must contain the power of two which M is, that is, C(M)=2↑C(MU).C(MU) must be at least 3 and no more than 13_8, due to memory limitations.

6.2 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

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

* The inverse is defined here to be

$$X_j = \sum_{k=0}^{N-1} S_j W^{-jk}$$

(S_j are real), without the 1/N scale factor.

(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	Functions
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.
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	0053
ADDR	1134
ADDWOS	1156
ADDXTR	0760
ADD1	1173
ADD2	0041
ADD3	0777
ADJSGN	0575
AI	0052
AR	0051
ARG2	1017
ASR	7415
BI	0050
BIGSNU	0013
BILI	0037
BILR	0036
BINMLI	0035
BINMLR	0034
BR	0047
BUILD	0551
C	0040
CAM	7621
CCIA	0106
CHKPT	0524
CNOP	0107
CNOTS	0702
COSINE	0044
DATAHI	6402
DOFFT	0060
DOIFFT	0061
DVI	7407
F	0007
FFT	0400
FLIP	1044
FLIPCT	1060
GETRIG	0057
GI	0046
GR	0045
IFFT	0076
INDEX	1133
INVERT	0055
INVRT	1036
K	0033
L	0005
LOOP1	0450
LSR	7417
M	0020
MAXNU	0023
MNOVR2	0024
MQA	7501
MQL	7421
MU	0021
MULT	0056

SYMBOL TABLE

MULTIP	1000
MUY	7405
N	0003
NMI	7411
NOROT	0571
NOTNOR	1171
NOVER4	0022
NO4MIK	1132
NU	0004
P	0032
PI	0030
PR	0027
Q	0031
QI	0026
QR	0025
QUAD1	1110
QUAD2	1072
RBUILD	0704
RECHK	0706
RESETC	0705
REVERS	0713
S	0006
SCA	7441
SCALE	0066
SCL	7403
SETC	0547
SGNADJ	0075
SGNX	1314
SHFCHK	0070
SHFLAG	0067
SHFT1	1077
SHFT2	1114
SHFT3	1125
SHIFCT	0567
SHIFT1	0071
SHIFT2	0072
SHIFT3	0073
SHL	7413
SIGN	1035
SINE	0043
SINLOC	0064
SINRET	1122
SINTAB	1375
SORT	0054
SORTX	0707
SWAPED	0751
SYNTH	0062
SYNTHT	1174
TEMPR	0042
TRIGET	1061
WORD	1056
WORDP	1057
XRLOC	0065

SYMBOL TABLE

XRTAB	2400
XSGN	0074
XTRACT	1242
XTRADD	0063

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.

CORRECTION TO DECUS NO. 8-143

by

Nezih C. Geçkinli,
Department of Electrical Engineering,
Middle East Technical University,

Ankara, Turkey

October 1973

The subroutine DECUS NO. 8-143 (may be DECUS NO. 8-144 also) <u>does not consider</u> an overflow which may occur during a complex multiplication:

$$(a+ib).(\cos\theta+i\sin\theta) = (a.\cos\theta-b\sin\theta)+i(a.\sin\theta+b\cos\theta)$$
$$= c+id$$

For $\sqrt{2} < |a| + |b| < 2$, an overflow is apt to occur for some values of θ. For example, if $a = 0.9$, $b = 0.8$, $\theta = \pi/4$, then $c = 0.07$, $d = 1.20$, which causes an overflow.

An example to this phenomenon is given in the appendix.

Therefore, not to have the possibility of overflow, the numbers must be kept between $-1/\sqrt{2}$ and $1/\sqrt{2}$. However, because of the implementation difficulty, correction is made to keep the numbers between -0.5 and 0.5 (i.e., 6001_8 and 1777_8.)

CORRECTION TO DECUS NO. 8-143

ORIGINAL	CORRECTED
* 1164	* 1164
RAL	RTL

CORRECTED

5.2. Data Scaling

All inputs X_i must be scaled such that

$$6001_8 \leqslant X_i \leqslant 1777_8, \quad \forall i.$$

(If the length of the input data sequence is going to be at least doubled by extending it with zeros, input data X_i must be scaled such that $4\emptyset\emptyset1_8 \leqslant X_i \leqslant 3777_8, \; \forall i.$)

APPENDIX

An example to the error of the subroutine DECUS NO. 8-143 is given below.

INPUT DATA (OCTAL)	FOURIER ENERGY SPECTRUM OF THE INPUT DATA (DECIMAL)	
ØØØØ	1411	0**********************
Ø1ØØ	Ø256	1**** ⟵ ERROR (MUST BE 1Ø17)
Ø371	Ø354	1******
1Ø15	ØØ41	1*
1523	ØØØØ	1
2231	ØØØ1	.
2656	ØØØØ	1
3146	ØØØ6	1 ⟵ ERROR (MUST BE ØØØØ)
3246	ØØØØ	1
3146	Ø5Ø2	1******** ⟵ ERROR (MUST BE ØØØØ)
2656	ØØØØ	0
2231	ØØØØ	1
1523	ØØØØ	1
1Ø15	ØØØØ	1
Ø371	ØØØØ	1
Ø1ØØ	Ø2Ø5	.*** ⟵ ERROR (MUST BE ØØØØ)
ØØØØ	ØØØØ	1
ØØØØ		
ØØØØ	SCALING FACTOR = 2↑ØØØ4	
ØØØØ		
ØØØØ		
ØØØØ		
ØØØØ		
ØØØØ		
ØØØØ		
ØØØØ		
ØØØØ		
ØØØØ		
ØØØØ		
ØØØØ		
ØØØØ		
ØØØØ		