STANFORD RESEARCH INSTITUTE
MENLO PARK, CALIFORNIA

Mathematical Sciences Department

Research Memorandum

November 1966

ALGOL PROCEDURES FOR THE FAST FOURIER TRANSFORM

by

Richard C. Singleton

Prepared for:

INSTITUTE RESEARCH AND DEVELOPMENT
SRI Project 181531-132

Approved:

W. G. Madow, Head, Mathematical Statistics and Operations Research Group

R. C. Alverson, Manager, Mathematical Sciences Department

ABSTRACT

The ~~body of this~~ report ~~was written as a contribution to the~~ Algorithms ~~section of the Communications of the ACM and~~ consists of six ALGOL procedures with comments. Procedure FASTTRANSFORM computes the complex finite Fourier transform or its inverse, using a modified version of the fast Fourier transform algorithm proposed by Cooley and Tukey. Procedure REALTRANSFORM similarly computes the real Fourier transform and inverse. The remaining four procedures are building blocks used in the first two procedures: they may be combined in other ways, for example, to form procedures for computing convolutions and power spectral density function estimates. The fast Fourier transform is a significant advance over previous methods, in that the number of arithmetic operations is proportional to $n \log_2 n$ instead of n^2. Detailed methods of computing this transform are shown here in the language of ALGOL. A new approach to organizing the computations is used, one that makes practical the solution of large problems in which data overlay within high speed storage will occur.

ALGOL PROCEDURES FOR THE FAST FOURIER TRANSFORM

Richard C. Singleton *
Stanford Research Institute,
Menlo Park, California.

The following procedures are based on the Cooley-Tukey algorithm [1,2,3] for computing the finite Fourier transform of a complex data vector; the dimension of the data vector is assumed here to be a power of two. Procedure FASTTRANSFORM computes either the complex Fourier transform or its inverse. Procedure REALTRANSFORM computes either the Fourier coefficients of a sequence of real data points or evaluates a Fourier series with given cosine and sine coefficients. The number of arithmetic operations for either procedure is proportional to $n \log_2 n$, where n is the number of data points.

Procedures FASTFOURIER, REVERSEFOURIER, REORDER, and REALTRAN are building blocks, and are used in the two complete procedures mentioned above. The fast transform can be computed in a number of different ways, and these building block procedures were written so as to make practical the computing of large transforms on a system with multiprogramming and/or virtual memory. Data is accessed in sub-sequences of consecutive array elements, and as much computing as possible is done in one section of the data before moving on to another. Procedure FASTFOURIER computes the Fourier transform, or inverse, of data in reverse binary order and leaves the result in normal binary order. Procedure REORDER permutes a complex vector from binary to reverse binary order or from reverse binary to binary order; this procedure also permutes real data in preparation for efficient use of the complex Fourier transform. The procedure REALTRAN is used to unscramble and combine the complex transforms of the even and odd numbered elements of a sequence of real data points; this procedure is not restricted to powers of two and requires only that the number of data points be even.

---

* This work was supported by Stanford Research Institute, out of Research and Development funds

```
procedure FASTTRANSFORM(A,B,m,inverse);
   value m,inverse; integer m; Boolean inverse;
   array A,B;
comment Computes the finite Fourier transform of 2^m complex
   data points, using the Cooley-Tukey algorithm [1]. The
   parameter m determines the dimension n=2^m of the transform .
   m>1 is assumed. The arrays A[0:n-1] and B[0:n-1] initially
   contain the real and imaginary components of the data
   vector, and, upon completion, contain the transformed values.
   If inverse is false, the Fourier transform
```

$$(x_j+iy_j) = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} (a_k+ib_k) \exp(i2\pi jk/n) \qquad \text{for } j=0,1,\ldots,n-1$$

```
   is computed, where the terms (a_k+ib_k) represent the initial data
   array values and (x_j+iy_j) represent the transformed
   values. If inverse is true, the inverse (complex
   conjugate) Fourier transform
```

$$(x_j+iy_j) = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} (a_k+ib_k) \exp(-i2\pi jk/n) \qquad \text{for } j=0,1,\ldots,n-1$$

```
   is computed, where (a_k+ib_k) and (x_j+iy_j) again represent
   the initial and transformed values. The transform
   followed by the inverse transform or the inverse
   transform followed by the transform gives an identity
   transformation. The procedures FASTFOURIER and REORDER
   are used by this procedure and must also be declared;
begin if inverse then
   begin FASTFOURIER(A,B,m,1/sqrt(2↑m),true);
      REORDER(A,B,m,false);
```

```
    end else
    begin FASTFOURIER(A,B,m,1/sqrt(2↑m),false);
       REORDER(A,B,m,false);
    end
end FASTTRANSFORM;
```

<u>procedure</u> REALTRANSFORM(A,B,m,inverse);

<u>value</u> m,inverse; <u>integer</u> m; <u>Boolean</u> inverse;

<u>array</u> A,B;

<u>comment</u> Computes the finite Fourier transform of $2^{m+1} \geq 8$ real data points, using the Cooley-Tukey algorithm[1,2]. If inverse is <u>false</u>, the arrays A[0:n] and B[0:n], where $n=2^m$, are assumed to contain the first 2^m real data points $x_0, x_1, \ldots x_{n-1}$ as A[0], A[1],...A[n-1] and the remaining 2^m real data points $x_n, x_{n+1}, \ldots x_{2n-1}$ as B[0], B[1],... B[n-1]. On completion of the transform the arrays A and B contain respectively the Fourier cosine and sine coefficients a_k and b_k, computed according to the relations

$$a_k = \frac{1}{n} \sum_{j=0}^{2n-1} x_j \cos(\pi jk/n) \qquad \text{for } k=0,1,\ldots n$$

and

$$b_k = \frac{1}{n} \sum_{k=0}^{2n-1} x_j \sin(\pi jk/n) \qquad \text{for } k=0,1,\ldots n \, .$$

If inverse is <u>true</u>, the arrays A and B are assumed to contain initially n+1 cosine coefficients $a_0, a_1, \ldots a_n$ and n+1 sine coefficients $b_0, b_1, \ldots b_n$, where $b_0 = b_n = 0$. The procedure evaluates the corresponding time series $x_0, x_1, \ldots x_{2n-1}$, where

$$x_j = \frac{a_0}{2} + \sum_{k=1}^{n-1} [a_k \cos(\pi jk/n) + b_k \sin(\pi jk/n)] + \frac{a_n}{2} \, ,$$

and leaves the first n values as A[0], A[1],... A[n-1] and the remaining n values as B[0], B[1],... B[n-1].

The procedures FASTFOURIER, REVERSEFOURIER, REORDER, and REALTRAN are used by this procedure, and must also be declared;

```
begin if inverse then
    begin REALTRAN(A,B,2↑m,true,true);
        FASTFOURIER(A,B,m,1/2,true);
        REORDER(A,B,m,true);
    end else
    begin REORDER(A,B,m,true);
        REVERSEFOURIER(A,B,m,1/2↑(m+1),false);
        REALTRAN(A,B,2↑m,false,false);
    end
end REALTRANSFORM;
```

```
procedure FASTFOURIER(A,B,m,scale,negexp);
     value m,scale,negexp; integer m; real scale;
     Boolean negexp; array A,B;
```

comment Computes the finite Fourier transform of 2^m complex data points, using a modified version of the Cooley-Tukey fast transform algorithm [1]. The data is assumed to be in normal order in arrays A[0:n-1] and B[0:n-1] for the real and imaginary components respectively, where $n=2^m$ is the dimension of the transform and $m>1$ is assumed. The transformed result replaces the original data, but is arranged in reverse binary order. That is, the j^{th} value of the result, where $j = j_{m-1}2^{m-1} + j_{m-2}2^{m-2} + \ldots + j_1 2 + j_o$, is found in location $j_0 2^{m-1} + j_1 2^{m-2} + \ldots + j_{m-2}2 + j_{m-1}$ of arrays A and B. Procedure REORDER can be used to permute the result to normal ordering, if desired. If negexp is false, the Fourier transform

$$(x_j + iy_j) = \text{scale} \sum_{k=0}^{n-1} (a_k+ib_k) \exp(i2\pi jk/n) \qquad \text{for } j = 0,1,\ldots n-1$$

is computed, and if negexp is true, the corresponding complex conjugate transform is computed, using a minus sign in the exponential terms. The terms (a_k+ib_k) represent the initial values, and (x_j+iy_j) represent the transformed values;

```
begin integer j,k,kk,kb,ks,jj,n,nq,span;
     real re,im,cn,sn,rad;
     integer array C,D[0:m]; array CC,SS[0:m];
     C[0] := n := 1;
```

```
     ks := n;
     for kk := C[m-1]-1 step -1 until 0 do
     begin ks := ks-1; re := A[kk]-A[ks];
         A[kk] := scale×(A[kk]+A[ks]);
         A[ks] := scale×re; im := B[kk]-B[ks];
         B[kk] := scale×(B[kk]+B[ks]);
         B[ks] := scale×im
     end;
     jj := kb := 0;  j := m := m-2; nq := C[m];
     for k := 0 step 1 until m do D[k] := C[m-k] ;
     rad := 6.28318530718/n; go to L2;
L:   if jj≥D[j] then
     begin jj := jj-D[j]; j := j+1; go to L end
     else jj := jj+D[j];
L2:  span := C[j]; if jj<D[j] then
     begin k := span×jj;
         CC[j] := cn := sin((nq-k)×rad);
         SS[j] := sn := sin(k×rad)
     end else
     begin cn := -SS[j]; sn := CC[j] end;
     if negexp then sn := -sn;
     for kk := kb+span-1 step -1 until kb do
     begin ks := kk+span;
         re := cn×A[ks]-sn×B[ks];
         im := sn×A[ks]+cn×B[ks];
         A[ks] := A[kk]-re; A[kk] := A[kk]+re;
         B[ks] := B[kk]-im; B[kk] := B[kk]+im;
     end;
     if j>0 then
```

```
    begin j := j-1; go to L2 end;
    kb := kb+2; if kb<n then go to L;
end FASTFOURIER;
```

procedure REVERSEFOURIER(A,B,m,scale,negexp);

value m,scale,negexp; **integer** m; **real** scale;

Boolean negexp; **array** A,B;

comment Computes the finite Fourier transform of 2^m complex data points, using a modified version of the Sande-Tukey [2,3] fast transform. The data is assumed to be in reverse binary order in arrays A[0:n-1] and B[0:n-1] for the real and imaginary components respectively, where $n=2^m$ and $m>1$ are assumed. The data may be in this ordering due to an earlier transform by procedure FASTFOURIER or a permutation by procedure REORDER. The transformed result replaces the original data, and is left in normal ordering. If negexp is *false* the Fourier transform

$$(x_j+iy_j) = \text{scale} \sum_{k=0}^{n-1} (a_k+ib_k) \exp(i2\pi jk/n) \quad \text{for } j=0,1,\ldots n-1$$

is computed, and if negexp is *true*, the corresponding complex conjugate transform is computed, using a minus sign in the exponential terms. The terms (a_k+ib_k) represent the initial values, and (x_j+iy_j) the transformed values;

begin **integer** j,k,kk,kb,ks,jj,n,nh,nq,span;

real re,im,cn,sn,rad;

integer **array** C,D[0:m]; **array** CC,SS[0:m];

C[0] := n := 1;

for k := 1 **step** 1 **until** m **do** C[k] := n := n+n;

nh := C[m-1]; nq := C[m-2];

rad := 6.28318530718/n;

```
    m := m-2; jj := nh-1;
    for k := 0 step 1 until m do D[k] := nh-C[k];
    for kb := n-2 step -2 until 0 do
    begin span := 1; j := m; k := jj;
L:      if k<nq then
        begin cn := SS[j]; sn := -CC[j] end else
        begin CC[j] := cn := -sin((k-nq)×rad);
              SS[j] := sn := sin((nh-k)×rad)
        end;
        if negexp then sn := -sn;
        for kk := kb+span-1 step -1 until kb do
        begin ks := kk+span;
              re := A[kk]-A[ks]; A[kk] := A[kk]+A[ks];
              im := B[kk]-B[ks]; B[kk] := B[kk]+B[ks];
              A[ks] := cn×re-sn×im; B[ks] := sn×re+cn×im;
        end;
        if jj<D[j] then
        begin jj := jj+C[j]; j := j-1; span := span+span;
              if j<0 then go to L2; k := k+k; go to L
        end
        else jj := jj-C[j]
    end;
L2: span := nh; ks := kb := nh-1;
    for kk := 0 step 1 until kb do
    begin ks := ks+1; re := A[kk]-A[ks];
        A[kk] := scale×(A[kk]+A[ks]); A[ks] := scale×re;
        im := B[kk]-B[ks]; B[kk] := scale×(B[kk]+B[ks]);
        B[ks] := scale×im
    end;
```

```
procedure REORDER(A,B,m,reel); value m,reel;
    integer m; Boolean reel; array A,B;
```

comment If reel is false, the 2^m elements each of arrays A[0:n-1] and B[0:n-1] are permuted from normal to reverse binary ordering or from reverse binary to normal order. The pair of values in location $j = j_{m-1}2^{m-1} + j_{m-1}2^{m-2} + \ldots + j_1 2 + j_0$ is interchanged with the pair of values in location $k = j_0 2^{m-1} + j_1 2^{m-2} + \ldots + j_{m-2}2 + j_{m-1}$. Doing the permutation twice gives an identity transformation. If reel is true, it is assumed that a sequence of 2^{m+1} real values, with the first 2^m values in array A and the second 2^m values in array B, is either to be permuted in preparation for computing Fourier coefficients or is the expected final result of evaluation of a real Fourier series. The permutation made is first to interchange each even numbered entry in B with the next higher odd numbered entry in A, then to permute adjacent pairs of entries in A and B to reverse binary order. Again, doing the permutation twice gives an identity transformation. $m > 1$ is assumed;

```
begin integer i,j,jj,k,kk,kb,ks,ku,lim,n;
    real t;
    integer array C,LST[0:m];
    C[0] := n := 1;
    for k := 1 step 1 until m do C[k] := n := n+n;
    j := m-1; i := kb := 0; if reel then
    begin ku := n-2; for k := 0 step 2 until ku do
        begin t := A[k+1]; A[k+1] := B[k]; B[k] := t end
    end else
    m := m-1; lim := (m+2) ÷ 2;
```

```
L:  ku := ks := C[j]+kb; jj := C[m-j]; kk := kb+jj;
L2: k := kk+jj;
L3: t := A[kk]; A[kk] := A[ks]; A[ks] := t;
    t := B[kk]; B[kk] := B[ks]; B[ks] := t;
    kk := kk+1; ks := ks+1;
    if kk<k then go to L3;
    kk := kk+jj; ks := ks+jj;
    if kk<ku then go to L2;
    if j>lim then
    begin j := j-1; i := i+1; LST[i] := j; go to L end;
    if i>0 then
    begin j := LST[i]; i := i-1; kb := ks; go to L end;
end REORDER;
```

```
procedure REALTRAN(A,B,n,negexp,inverse);
    value n,negexp,inverse; integer n;
    Boolean negexp,inverse; array A,B;
comment If inverse is false, this procedure unscrambles the complex transform
    of the n even numbered and n odd numbered elements of a
    real sequence of length 2n, where the even numbered
    elements were originally in A and the odd numbered elements
    in B. Then it combines the two real transforms to give the
    Fourier cosine coefficients A[0], A[1],... A[n] and sine
    coefficients B[0], B[1],... B[n] for the full sequence of
    2n elements. If inverse is true, the process is reversed,
    and a set of Fourier cosine and sine coefficients is made
    ready for evaluation of the corresponding Fourier series by
    means of the fast transform. In either case, the value of
    negexp must agree with that used in procedure FASTFOURIER
    or REVERSEFOURIER with which REALTRAN is paired. Going in
    either direction, REALTRAN scales by a factor of two, which
    should be taken into account in determining the appropriate
    overall scaling;
begin integer j,k,nh;
    real aa,ab,ba,bb,re,im,cd,cn,sd,sn,rad,r;
    nh := n ÷ 2; rad := 3.14159265359/n ;
    sd := sin(rad); r := -(2×sin(0.5×rad))↑2;
    cd := -0.5×r; cn := 1; sn := 0;
    if ¬ (negexp ≡ inverse) then sd := -sd;
    if inverse then
    begin cn := -1; cd := -cd; B[0] := B[n] := 0 end else
```

```
begin A[n] := A[0]; B[n] := B[0] end;
for j := 0 step 1 until nh do
begin k := n-j;
    aa := A[j]+A[k]; ab := A[j]-A[k];
    ba := B[j]+B[k]; bb := B[j]-B[k];
    re := cn×ba+sn×ab; im := sn×ba-cn×ab;
    B[k] := im-bb; B[j] := im+bb;
    A[k] := aa-re; A[j] := aa+re;
    cd := r×cn+cd; cn := cd+cn;
    sd := r×sn+sd; sn := sd+sn;
end;
if inverse then A[n] := B[n] := 0;
end REALTRAN;
```

These procedures were written originally for use on the Burroughs B-5500 system. Because of the limitation of no more than 1023 words in a single dimensiona... ray on this system, two-dimensional data arrays are used for transforms with $m>9$. With this modification, real transforms with $m=16$ (2^{17} data points) take about ten minutes of processing time and six minutes of input-output channel time for the (automatic) transfer of array rows between disk and core storage. Several transforms of this size have been computed, while sharing the computer with other programs. Experience with a large number of transforms with $m\geq14$ (exceeding actual core capacity) has shown that multiprogramming causes little increase in running times.

REFERENCES

1. J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series", Mathematics of Computation, Vol. 19, No. 90, pp. 297-301 (April 1965).

2. C. Bingham, M. D. Godfrey, and J. W. Tukey, "Modern Techniques of Power Spectral Estimation", unpublished paper, 1966.

3. W. M. Gentleman and G. Sande, "Fast Fourier Transforms - for Fun and Profit", AFIPS Conference Proceedings (1966 FJCC), Vol. 29, pp. 563-578, Spartan Books, Washington, D.C., 1966.

```
COMMENT DRIVER PROGRAM FOR TESTING FAST TRANSFORM PROCEDURES;
BEGIN REAL SS1,SS2,RX,RY,R;
INTEGER J,K,M,N,NN,RDM;
ARRAY A,B,X,Y[0:512];
COMMENT DECLARE PROCEDURES FASTFOURIER, REVERSEFOURIER, REORDER
    AND REALTRAN;
COMMENT DECLARE PROCEDURES FASTTRANSFORM AND REALTRANSFORM;
M := 9; N := 2↑M; COMMENT DIMENSION OF PROBLEM;
RDM := 123; COMMENT INITIAL RANDOM NUMBER, ODD AND < 2↑27;
NN := N-1; FOR J := 0 STEP 1 UNTIL NN DO
BEGIN COMMENT FILL DATA ARRAYS WITH NORMAL DEVIATES, MEAN=0, S.D.=1;
LR: RDM := 3589×RDM; RDM := RDM-(RDM ÷ 134217728)×134217728;
    RX := (RDM-67108864)/67108864;
    RDM := 3589×RDM; RDM := RDM-(RDM ÷ 134217728)×134217728;
    RY := (RDM-67108864)/67108864;
    R := RX↑2+RY↑2; IF R≥1.0 THEN GO TO LR;
    R := SQRT(-2×LN(R)/R);
    A[J] := X[J] := RX×R; B[J] := Y[J] := RY×R;
END;
A[N] := B[N] := X[N] := Y[N] := 0;
FASTTRANSFORM(A,B,M,FALSE);
FASTTRANSFORM(A,B,M,TRUE);
SS1 := SS2 := 0; FOR J := 0 STEP 1 UNTIL N DO
BEGIN SS1 := (A[J]-X[J])↑2+SS1; A[J] := X[J];
    SS2 := (B[J]-Y[J])↑2+SS2; B[J] := Y[J];
END;
SS1 := SQRT(SS1/N); SS2 := SQRT(SS2/N);
COMMENT LIST ROOT-MEAN-SQUARE ERRORS FOR REAL AND IMAGINARY
```

```
   PARTS OF THE COMPLEX TRANSFORM-INVERSE PAIR;
UTREAL(1,SS1); OUTREAL(1,SS2);
EALTRANSFORM(A,B,M,FALSE);
EALTRANSFORM(A,B,M,TRUE);
S1 := SS2 := 0; FOR J := 0 STEP 1 UNTIL N DO
EGIN SS1 := (A[J]-X[J])↑2+SS1; A[J] := X[J];
   SS2 := (B[J]-Y[J])↑2+SS2; B[J] := Y[J];
ND;
S1 := SQRT((SS1+SS2)/(N+N));
OMMENT LIST ROOT-MEAN-SQUARE ERROR FOR REAL TRANSFORM-INVERSE PAIR;
UTREAL(1,SS1);
ND;
```