THE FAST FOURIER TRANSFORM ALGORITHM: PROGRAMMING CONSIDERATIONS IN THE CALCULATION OF SINE, COSINE AND LAPLACE TRANSFORMS†

J. W. COOLEY, P. A. W. LEWIS‡ AND P. D. WELCH

IBM Corporation, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, N.Y., U.S.A.

(Received 22 August 1969)

In the organization of programming packages for computing Fourier and Laplace transforms, it is useful, both for conceptual understanding and for operational efficiency to consider the discrete complex Fourier transform as a kind of nucleus around which programming for special applications is performed. An advantage of these procedures is that the basic complex Fourier transform algorithm is systematic and can relatively easily be implemented in efficient subroutines, micro-programs and special hardware devices. Once this is done, programming for special properties of the data can efficiently be left to the user to implement on a general purpose computer.

The problem of establishing the correspondence between the discrete transforms and the continuous functions with which one is usually dealing is described. The application of these results and the above-mentioned subroutines to the calculation and inversion of Laplace transforms is given with formulas and empirical results displaying the effect of optimal parameters on computational efficiency and accuracy.

1. INTRODUCTION

The fast Fourier transform (FFT) method is a computational algorithm which, on its rediscovery [1, 2, 3] greatly increased the speed with which Fourier transforms can be computed on digital devices. As a consequence, digital application of Fourier methods has been widely utilized and can be expected to be used even more as the FFT method is incorporated in efficient computer subroutines, microprograms, and even special hardware. Therefore, it seems useful to investigate further the essential properties of the discrete Fourier transform, its correspondence with integral transforms, and various algorithms for using the discrete Fourier transform in special circumstances.

The discrete complex Fourier series is a one-to-one mapping of any sequence $A(n)$, $n = 0, 1, 2, \ldots, N - 1$, of $N$ complex numbers onto another sequence defined by

$$X(j) = \sum_{n=0}^{N-1} A(n) W_n^j, \quad j = 0, 1, \ldots, N - 1,$$

where $i = \sqrt{-1}$ and where $W_n = \exp(2\pi i/N)$ is the principal $N$th complex root of unity. This is also referred to as the inverse discrete Fourier transform (IDFT) since, in the literature, the formula for $A(n)$ in terms of $X(j)$,

$$A(n) = \frac{1}{N} \sum_{j=0}^{N-1} X(j) W_n^{-j},$$

† Presented at the lecture series on “Applications and methods of random data analysis” on 8 to 11 July 1969, at Southampton University.
‡ At present on leave at Department of Mathematics, Imperial College of Science and Technology, London, England.

is known as the discrete Fourier transform (DFT). A direct calculation of (1) as an accumulated sum of products for each \( j \) would take \( N^2 \) operations. The fast Fourier transform method (FFT) is an algorithm for computing (1) or (2) in \( N \log N \) operations, where "operation" means a complex multiplication and addition. The FFT algorithm has been described and programmed, in most cases, as a calculation of the operation defined by (1) on complex numbers. However, in actual practice, there is a wide variety of special conditions which one actually wants. For example, the data may be real rather than truly complex, and it may be even or odd so that cosine or sine transforms may be used. Of course, these transforms can be done directly with the complex DFT, but this would be inefficient since it would require redundant storage and computation. The purpose of some of the algorithms given below will be to avoid this. It will also be shown here how one can, by suitably manipulating the data, compute Laplace transforms in terms of the DFT. One could, of course, use the basic ideas of the FFT method and write special programs for each of these special cases. However, the logically simple and systematic nature of the FFT method for complex data makes it more practical to program it in a very efficient subroutine or special hardware and to obtain the above-mentioned special transforms by manipulating the data and using the complex FFT subroutine. The purpose of section 2 of this paper is to derive some of the special algorithms for doing this when the data is real and even or odd.

In section 3, a correspondence between the DFT and the Fourier integral transform will be shown. This gives some insight into the errors introduced by the discrete sampling of a function for only a finite interval of time. This will be used to develop an error analysis and procedures for selecting optimal parameters.

Finally, in section 4, it will be shown how one can use the DFT, computed by the FFT method, to invert Laplace transforms. Two methods are described: one of these computes Laguerre expansion coefficients; the other computes point values of the solution. For these, the data being supplied to the FFT subroutines contains some redundancy which can be used to reduce the computation. An additional parameter for the user to choose appears in the Laplace transform problem. This is \( c \), the real part of the transform variable. It is shown here how the discussion of aliasing in section 3 leads to formulas for an optimal choice of \( c \).

2. THE CALCULATION OF SINE AND COSINE TRANSFORMS OF REAL OR IMAGINARY DATA

The purpose of this section is to show how a subroutine or special hardware unit capable of computing the discrete complex Fourier series

\[
X(j) = \sum_{n=0}^{N-1} A(n) W_N^{nj}
\]

(3)

can, with some pre- and post-processing of the data, compute a variety of special transforms. The variety actually consists in using various types of redundancy in the data to reduce the size of the complex array on which the subroutine is to perform.

Before proceeding, some definitions and theorems will be summarized without proofs. Further details which are useful in computing special cases, have been published elsewhere [5].

Definition. If the sequence \( A(n), n = 0, 1, \ldots, N - 1 \) is the DFT of \( X(j) \), we write

\[
X(j) \leftrightarrow A(n).
\]

(4)

Functions related to each other by an integral Fourier transform will also be connected by the double-headed arrow. In this case the function on the right is the Fourier integral transform of the one on the left. The sequence appearing on the right in (4) is usually a function of frequency and will be referred to as the discrete Fourier transform (DFT) of the sequence on
the left. The sequence appearing on the left in (4) is usually a function of time, space, or simply a sequential index and will be referred to here as a Fourier series or as the inverse discrete Fourier transform (IDFT) of the sequence on the right. It may also be referred to as “data”.

An important property, used in the following sections, is that the indices of \( X(j) \) and \( A(n) \) are to be interpreted modulo \( N \). Therefore, \( X(j) \) with \( 0 \leq j < N - 1 \) is understood to mean \( X(N - j) \).

A sequence is even if
\[
X(j) = X(-j),
\]
and odd if
\[
X(j) = -X(-j). 
\]

A sequence is conjugate even or conjugate odd if
\[
X(j) = \bar{X}(-j) \quad \text{or} \quad X(j) = -\bar{X}(j),
\]
respectively, where "\( \sim \)" denotes the complex conjugate.

Some useful theorems follow.

**Theorem 1.** The DFT is linear. If
\[
X_1(j) \leftrightarrow A_1(n), \quad n, j = 0, 1, \ldots, N - 1,
\]
\[
X_2(j) \leftrightarrow A_2(n),
\]
and
\[
X(j) = aX_1(j) + bX_2(j),
\]
then
\[
A(n) = aA_1(n) + bA_2(n),
\]
where \( X(j) \leftrightarrow A(n) \).

**Theorem 2.**
\[
X(-j) \leftrightarrow A(-n). 
\]

**Corollary 2.1.** \( X(j) \) is even/odd if and only if \( A(n) \) is even/odd.

**Theorem 3.**
\[
\bar{X}(j) \leftrightarrow \bar{A}(-n),
\]
\[
\bar{X}(-j) \leftrightarrow \bar{A}(n). 
\]

**Corollary 3.1.** \( X(j) \) is real/imaginary if and only if \( A(n) \) is conjugate even/odd, respectively; i.e. \( A(n) = \pm \bar{A}(-n) \). \( A(n) \) is real-imaginary if and only if \( X(j) \) is even/odd, respectively; i.e. \( X(j) = \pm \bar{X}(-j) \).

**Corollary 3.2.** \( X(j) \) is real and even if and only if \( A(n) \) is real and even.
\( X(j) \) is real and odd if and only if \( A(n) \) is imaginary and odd.
\( X(j) \) is imaginary and even if and only if \( A(n) \) is imaginary and even.
\( X(j) \) is imaginary and odd if and only if \( A(n) \) real and odd.

**Theorem 4.**
\[
X(j - k) = W_{\frac{N}{m}}^{-nk} A(n), \quad (10a)
\]
\[
W_{\frac{N}{m}}^{-m} X(j) = A(n - m). \quad (10b)
\]

**Theorem 5.**
\[
\delta(j) \leftrightarrow 1/N, 
\]
\[
1 \leftrightarrow \delta(n), \quad (11b)
\]
where \( \delta(0) = 1, \delta(j) = 0 \) for \( j \neq 0 \).
Theorem 6.

\[ X(0) = \sum_{n=0}^{N-1} A(n), \]  
\[ A(0) = 1/N \sum_{j=0}^{N-1} X(j). \]  

(12)  
(13)

Theorem 7. Convolution theorem. If

\[ X_1(j) \leftrightarrow A_1(n), \]
\[ X_2(j) \leftrightarrow A_2(n), \]

then

\[ 1/N \sum_{k=0}^{N-1} X_1(k) X_2(j-k) \leftrightarrow A_1(n) A_2(n), \]  
and

\[ X_1(j) X_2(j) \leftrightarrow \sum_{m=0}^{N-1} A_1(m) A_2(n-m). \]  

(14)  
(15)

Corollary 7.1.

\[ 1/N \sum_{j=0}^{N-1} |X(j)|^2 = \sum_{n=0}^{N-1} |A(n)|^2. \]  

(16)

Theorem 8. If

\[ \text{stretch}_k\{j; X\} = X(0), 0, \ldots, 0, X(1), 0, \ldots, 0, X(2), \ldots, X(N-1), 0, \ldots, 0, \]  
where \( K - 1 \) zeros have been inserted between the \( X(j) \)’s, then

\[ \text{stretch}_k\{j; X\} \leftrightarrow A(n)/K, \quad n = 0, 1, \ldots, NK - 1. \]  

(17)  
(18)

Theorem 9. If

\[ \text{sample}_k\{j, X\} = X(0), X(K), X(2K), \ldots, X(N - K), \]  
then

\[ \text{sample}_k\{j, X\} \leftrightarrow \sum_{m=0}^{K-1} A \left( n + m \frac{N}{K} \right). \]  

(19)  
(20)

Let us imagine that we have a Fourier series subroutine whose input is the complex sequence \( A(n), n = 0, 1, \ldots, N - 1 \) and whose output is the Fourier series (or IDFT),

\[ X(j) = \sum_{n=0}^{N-1} A(n) W_n^j, \quad j = 0, 1, \ldots, N - 1. \]  

(21)

We now describe procedures all of which are based on this single Fourier series subroutine which, we assume, embodies the fast Fourier transform algorithm.

2.1. PROCEDURE 1. THE FOURIER TRANSFORM

It can easily be seen that letting \( \bar{X}(n)/N \) replace \( A(n) \) as input and \( \bar{X}(j) \) replace \( X(j) \) as output, the above subroutine will be capable of computing the discrete Fourier transform (DFT)

\[ A(n) = 1/N \sum_{j=0}^{N-1} X(j) W_n^{-jn}. \]  

(22)

2.2. PROCEDURE 2. THE FOURIER TRANSFORM OF TWO SETS OF REAL DATA IN ONE PASS THROUGH A DFT SUBROUTINE

Using the linearity property, we see that if \( X_1(j) \) and \( X_2(j) \) are real sequences such that

\[ X_1(j) \leftrightarrow A_1(n), \]
\[ X_2(j) \leftrightarrow A_2(n), \]

2.74
and
\[ X(j) = X_1(j) + iX_2(j), \]  
then \( X(j) \) has the transform
\[ A(n) = A_1(n) + iA_2(n). \]
Replacing \( n \) by \( N - n \), taking complex conjugates of both sides, and applying corollary 3.1 gives
\[ \bar{A}(N - n) = A_1(n) - iA_2(n). \]
Solving (24) and (25) for \( A_1(n) \) and \( A_2(n) \), we get
\[ A_1(n) = \frac{1}{2} [\bar{A}(-n) + A(n)], \]
\[ A_2(n) = \frac{i}{2} [\bar{A}(-n) - A(n)]. \]
Hence, procedure 2 is to
(a) form \( X(j) \) as defined by (23);
(b) compute \( A(n) \) by means of the DFT subroutine;
(c) compute \( A_1(n) \) and \( A_2(n) \) according to (26)
for \( n = 0, 1, 2, \ldots, N/2 \).

2.3. PROCEDURE 3. DOUBLING ALGORITHM—COMPUTING THE FOURIER TRANSFORM OF 2\( N \) POINTS FROM THE TRANSFORMS OF TWO \( N \)-POINT SEQUENCES

Given the \( 2N \) data points \( Y(j) \) with \( Y(j) \leftrightarrow C(n), n, j = 0, 1, \ldots, 2N - 1 \), suppose the two \( N \)-point sequences
\[ X_1(j) = Y(2j), \]
\[ X_2(j) = Y(2j + 1), \quad j = 0, 1, 2, \ldots, N - 1, \]
have the \( N \)-point transforms \( A_1(n) \) and \( A_2(n) \),
\[ X_1(j) \leftrightarrow A_1(n), \]
\[ X_2(j) \leftrightarrow A_2(n). \]
Separating even- and odd-indexed points in the series for \( C(n) \), we have
\[ C(n) = \frac{1}{2N} \sum_{j=0}^{2N-1} Y(j) W_{2N}^{-nj} \]
\[ = \frac{1}{2N} \left\{ \sum_{j=0}^{N-1} Y(2j) W_{2N}^{-2nj} + \sum_{j=0}^{N-1} Y(2j+1) W_{2N}^{-2n(j+1)} \right\}. \]
Since
\[ W_{2N}^2 = W_N, \]
\[ C(n) = \frac{1}{2N} \left\{ \sum_{j=0}^{N-1} Y(2j) W_N^{-nj} + \sum_{j=0}^{N-1} Y(2j+1) W_N^{-nj} W_{2N}^{-n} \right\}. \]
The sums appearing in (30) define \( A_1(n) \) and \( A_2(n) \), so (30) can be written
\[ C(n) = \frac{1}{2} \{ A_1(n) + A_2(n) W_{2N}^{-n} \}. \]
Substituting \( N + n \) for \( n \), and using the fact that \( W_{2N}^N = -1 \), we get
\[ C(N + n) = \frac{1}{2} \{ A_1(n) - A_2(n) W_{2N}^{n} \}. \]
Hence, procedure 3 is as follows:

(a) compute the two $N$-point DFT's of the sequences $X_1(j) = X(2j)$ and $X_2(j) = X(2j + 1)$;
(b) apply (31) and (32) to the resulting transforms.

This procedure, when iterated upon to produce successive doublings up to $N$, the total number of points, is the FFT algorithm with radix 2.

2.4. Procedure 4. Calculation of the DFT of Real Data

The transform of a single sequence of $2N$ real data points can be done by means of one DFT of $N$ complex points by using procedure 2 and procedure 3. First, assuming $Y(j)$, $= 0, 1, \ldots, 2N - 1$ to be the real sequence, let

$$Y(j) \leftrightarrow C(n).$$

Then, form the $N$-point sequences,

$$X_1(j) = Y(2j),$$
$$X_2(j) = Y(2j + 1),$$
$$X(j) = X_1(j) + X_2(j)i.$$  \hspace{1cm} (33)

Compute the $N$-point Fourier transform of $X(j)$ and use procedure 2, equation (26), obtain the transforms of the two real sequences $X_1(j)$, $X_2(j)$ in terms of the DFT of $X(j)$. Equation (26) is repeated here,

$$A_1(n) = \frac{1}{2} \{ \tilde{A}(-n) + A(n) \},$$ \hspace{1cm} (34)
$$A_2(n) = \frac{i}{2} \{ \tilde{A}(-n) - A(n) \}.$$

Now, having the transforms of the even- and odd-indexed points of $Y(j)$, procedure 3, the doubling algorithm (31), and the transform of the full array $Y(j)$ by using formula

$$C(n) = \frac{1}{2} \{ A_1(n) + A_2(n) W_n^{-n} \}. \hspace{1cm} (35)$$

Note that the upper half of the $C(n)$ array is redundant; i.e. $Y(j)$ real implies that $C(n) = \overline{C}(2N - n)$. Since $X_1(j)$ and $X_2(j)$ are real, $A_1(n)$ and $A_2(n)$ are conjugate even. Replacing $N$ by $N + n$ in (35), we get

$$C(N + n) = \frac{1}{2} \{ A_1(n) - A_2(n) W_n^{2n} \}. \hspace{1cm} (36)$$

It is efficient to use (35) and (36) for $n = 0, 1, 2, \ldots, N/2$.

It can easily be shown that the $C(n)$'s in

$$Y(j) = \sum_{n=-N}^{N-1} C(n) W_n^{jn}, \quad j = 0, 1, \ldots, 2N - 1 \hspace{1cm} (37)$$

can be identified with the sine-cosine coefficients of the series

$$Y(j) = a(0) + \sum_{n=1}^{N-1} \left\{ a(n) \cos \frac{n\pi j}{N} + b(n) \sin \frac{n\pi j}{N} \right\} + \frac{1}{2} (-1)^j a(N), \hspace{1cm} (38)$$

as follows:

$$a(n) = 2 \Re C(n), \quad \text{for } 0 \leq n \leq N; \hspace{1cm} (39)$$
$$b(n) = -2 \Im C(n), \quad \text{for } 0 < n < N.$$
Hence, procedure 4 is as follows:

(a) let the $2N$-point array $Y(j)$ be put into a complex $N$-point array $X(j)$ as defined by (33); in many computer programs, this means one does nothing since complex arrays are stored so that real and imaginary parts are already in alternating locations;

(b) compute the $N$-point DFT of $X(j)$;

(c) apply equation (34) for $n = 0, 1, \ldots, N/2$ to get $A_1(n)$ and $A_2(n)$;

(d) apply equations (35) and (36) for $n = 0, 1, \ldots, N/2$ to get $C(n)$ for $n = 0, 1, \ldots, N$.

Note that the $2N$ real data points $Y(j)$ are transformed to $N + 1$ complex frequency values $C(n), n = 0, 1, 2, \ldots, N$. Since $C(0)$ and $C(N)$ are real and since the remaining $C(n)$'s are complex, the sequence $C(n)$ contains $2N$ independent real numbers. One must remember that $C(n)$ is defined for $0 \leq n < 2N$, but that due to the property $C(n) = \overline{C}(-n)$, values for $N < n < 2N$ are redundant and need not be computed or stored.

2.5. PROCEDURE 5. THE CALCULATION OF FOURIER SERIES FOR REAL DATA

This is performed by reversing procedure 4. To derive the formulas, solve (35) and (36) for $A_1(n)$ and $A_2(n)$ in terms of $C(n)$ and $C(N + n)$:

$$A_1(n) = C(n) + C(N + n),$$

$$A_2(n) = [C(n) - C(N + n)] W_2^n.$$

Then, solving (34), one has

$$A(n) = A_1(n) + iA_2(n),$$

$$\overline{A}(N - n) = A_1(n) - iA_2(n).$$

Hence, procedure 5 is as follows:

(a) generate $A_1(n)$ and $A_2(n)$ for $n = 0, 1, \ldots, N/2$ according to (40) and (41);

(b) generate $A(n)$ by using (42) and (43) with $n = 0, 1, \ldots, N/2$;

(c) compute the inverse DFT of the $N$-element complex array $A(n)$; the result will be the sequence $X(j)$ whose real and imaginary parts are the real elements of $Y(j)$ as defined by (33).

Procedures 4 and 5 also permit us to transform a conjugate even data sequence $X(j)$ to a real frequency sequence $\hat{A}(n)$ and vice versa. This is easily seen to be accomplished by letting $\hat{X}(n)/N$ replace $A(n)$ and $\overline{A}(j)$ replace $X(j)$ in procedures 4 and 5, thereby switching the roles of "frequency" and "data".

2.6. PROCEDURE 6. THE CALCULATION OF COSINE SERIES FOR REAL DATA

It can easily be demonstrated that $Y(j)$ being real and even implies that its IDFT will be a cosine series. Thus, if $Y(j), j = 0, 1, 2, \ldots, 2N - 1$, is real and $Y(j) = Y(2N - j)$,

$$Y(j) = \sum_{n=0}^{2N-1} C(n) W_{2N}^{nj},$$

or

$$Y(j) = \frac{1}{2}a(0) + \sum_{n=1}^{N-1} a(n) \cos \frac{\pi j n}{N} + \frac{1}{2}(-1)^j a(N),$$

where the $a(n)$'s are real and

$$a(n) = 2C(n).$$
It also follows that the DFT of \( Y(j) \) can be expressed as a cosine series:

\[
a(n) = \frac{2}{N} \left[ \frac{1}{2} Y(0) + \sum_{j=1}^{N-1} Y(j) \cos \frac{\pi j n}{N} + \frac{1}{2} Y(N) \right].
\]  

(45)

We derive the procedure for computing a cosine transform of real even data \( Y(j) \), \( j = 0, 1, 2, \ldots, 2N - 1 \), where

\[
Y(2N - j) = Y(j).
\]

(Actually only \( Y(0), \ldots, Y(N) \) need be given.) First define the complex sequence

\[
X(j) = Y(2j) + [Y(2j + 1) - Y(2j - 1)] i
\]

(46)

for \( j = 0, 1, 2, \ldots, N - 1 \). (Only terms with \( j = 0, 1, \ldots, N/2 \) have to be formed.) This is a complex conjugate even sequence and, therefore, its transform, \( A(n) \), must be real. Procedure 5 gives an efficient way to transform a conjugate even sequence to a real sequence. We had in mind that the former was a frequency function and the latter was data. Here these must reverse roles. Letting the conjugate even sequence \( \tilde{X}(j)/N \) be the input to procedure 5, the output will be \( A(n) \), the real DFT of \( X(j) \).

Having \( A(n) \), procedure 2 can be used to obtain the transforms of the real and imaginary parts of \( X(j) \). Let us define

\[
Y(2j) \leftrightarrow A_1(n),
\]

\[
Y(2j + 1) \leftrightarrow A_2(n),
\]

\[
Y'(2j) = Y(2j + 1) - Y(2j - 1) \leftrightarrow A'_2(n).
\]

(48)

From procedure 2, equation (26),

\[
A_1(n) = \frac{1}{2}[A(n) + A(-n)],
\]

(49)

\[
A'_2(n) = \frac{1}{2i}[A(n) - A(-n)],
\]

(50)

where, it is to be remembered, \( A(n) \) is real. From theorem 4, we derive the fact that

\[
Y(2j - 1) \leftrightarrow W_n^{2n} A_2(n),
\]

and, therefore,

\[
A'_2(n) = A_2(n) - W_n^{2n} A_2(n),
\]

(51)

giving

\[
A_2(n) = A'_2(n)/(1 - W_n^{2n}).
\]

(52)

A special calculation must be made for \( n = 0 \) and \( n = N \). For this, we must compute

\[
A_2(0) = \frac{1}{N} \sum_{j=0}^{N-1} Y(2j + 1).
\]

(53)

Finally, procedure 3, the doubling algorithm, takes us from \( A_1(n) \) and \( A_2(n) \) to \( C(n) \), the DFT of \( Y(j) \). Substituting (49) and (51) in equation (31) of procedure 3, we get

\[
a(n) = 2C(n) = \frac{1}{2} \{[A(n) + A(-n)] - [A(n) - A(-n)]/[2 \sin \left( \pi n/N \right)] \}
\]

(54)

for \( n = 1, 2, \ldots, N - 1 \). For \( n = 0 \) and \( n = N \) we must use

\[
C(0) = \frac{1}{2} \{A_1(0) + A_2(0)\},
\]

\[
C(N) = \frac{1}{2} \{A_1(0) - A_2(0)\}.
\]

(55)
FAST FOURIER TRANSFORM ALGORITHM

Summarizing, procedure 6 for the cosine transform is:
(a) given the real even sequence \( Y(j), j = 0, 1, \ldots, 2N - 1 \), define \( X(j) \) according to (46);
(b) let \( \hat{X}(j)/N \) be the input to procedure 5; the output will be \( \hat{A}(n) = A(n) \);
(c) compute \( C(n) \) using (54) for \( n = 1, 2, \ldots, N - 1 \); for \( n = 0 \) and \( n = N \), let \( A_1(0) = A(0) \) and compute \( A_2(0) \) with (53); then, use (55) to obtain \( C(0) \) and \( C(N) \); finally, let \( a(n) = 2C(n) \) for \( n = 0, 1, \ldots, N \).

2.7. PROCEDURE 7. THE CALCULATION OF SINE SERIES FOR REAL DATA

It can be easily demonstrated that if \( Y(j), j = 0, 1, \ldots, 2N - 1 \) is real and odd, it is expressible as a sine series,

\[
Y(j) = \sum_{n=0}^{2N-1} C(n) \frac{W_{2N}^{nj}}{2N} = \sum_{n=1}^{N-1} b(n) \sin \left( \frac{n\pi j}{N} \right), \tag{56}
\]

where the \( b(n) \)'s are real and

\[
b(n) = 2iC(n).
\]

Note that one must have \( Y(0) = Y(N) = 0 \). It also follows that

\[
b(n) = \frac{2}{N} \sum_{j=1}^{N-1} Y(j) \sin \left( \frac{n\pi j}{N} \right). \tag{57}
\]

For the calculation, we make use of the fact that if \( Y(j) \) is real and odd, then \( iY(j) \) is conjugate even and corollary 3.1 then implies that its transform is real. Thus,

\[
Y(j) \leftrightarrow C(n) = b(n)/2i
\]

and

\[
+iY(j) \leftrightarrow +iC(n) = \frac{1}{2}b(n).
\]

Therefore, letting

\[
X(j) = -[Y(2j+1) - Y(2j-1)] + Y(2j)i, \tag{58}
\]

and, following a derivation similar to that for equation (54) in procedure 6, one arrives at formula (59) below for the coefficients of the cosine series.

Summarizing, procedure 7 is as follows:
(a) given the values \( Y(j), j = 1, \ldots, N - 1 \) of a real odd sequence \( Y(j), j = 0, 1, \ldots, 2N - 1 \), form \( X(j) \) according to (58);
(b) let \( \hat{X}(j)/N \) be input to procedure 5; the output will be \( \hat{A}(n) = A(n) \), where \( X(j) \leftrightarrow A(n) \);
(c) compute

\[
b(n) = 2iC(n) = \frac{1}{2} \{A(n) - A(-n) + [A(n) + A(-n)]/[2 \sin (\pi n/N)]\} \tag{59}
\]

for \( n = 1, 2, \ldots, N - 1 \).

Use of procedures 6 and 7 yields a fourfold decrease in computation and storage requirements since only half the real input data arrays need be supplied. (Actually, \( N/2 + 1 \) data for the cosine transform and \( N/2 - 1 \) for the sine transform.) The complex data \( X(j) \) to be transformed contains only \( N/2 \) terms rather than the \( 2N \) required if one supplies the full array \( Y(j) \) to a complex DFT subroutine.
3. THE CORRESPONDENCE BETWEEN DISCRETE AND INTEGRAL FOURIER TRANSFORMS

In the application of digital computers to Fourier methods with continuous functions, one must necessarily treat a discrete set of sampled values over a finite interval of time or distance. Furthermore, one has effects due to “rounding” errors resulting from representing and computing all quantities with finite numbers of digits. Having the theory of discrete Fourier transforms and a set of subroutines to perform the transforms described in the previous section, one is presented with the problem of knowing how to sample the given data and prepare a sequence as input to a subroutine, and then how to interpret the output of the subroutine in terms of the desired solution.

The present section briefly reviews a theorem [4] which relates the Fourier integral transform to the DFT in such a way as to give insight into the effects of sampling rates and intervals. From this, one hopes to be able to estimate optimal parameters and produce results with prescribed error bounds.

Consider the integral Fourier transform of a function \( x(t) \):

\[
a(f) = \int_{-\infty}^{\infty} x(t) e^{-2\pi i f t} \, dt \tag{60}
\]

and its inverse

\[
x(t) = \int_{-\infty}^{\infty} a(f) e^{2\pi i f t} \, df. \tag{61}
\]

This relationship will be denoted

\[x(t) \leftrightarrow a(f).\tag{62}\]

To observe the effect of sampling at finite intervals, express (61), evaluated at the points \( x_j = j \Delta t, j = 0, \pm 1, \pm 2, \ldots \), with \( F = 1/\Delta t \),

\[
x(t_j) = \int_{-\infty}^{\infty} a(f) e^{2\pi i j f / F} \, df = \sum_{k=-\infty}^{\infty} \int_{kF}^{(k+1)F} a(f) e^{2\pi i j f / F} \, df. \tag{63}
\]

The exponential function \( e^{2\pi i j f / F} \) is periodic in \( f \) with period \( F \), so this can be written

\[
x(j \Delta t) = \int_{0}^{F} a_{\Delta}(f) e^{2\pi i j f / F} \, df, \tag{64}
\]

where

\[
a_{\Delta}(f) = \sum_{k=-\infty}^{\infty} a(f + kF). \tag{65}
\]

Thus, knowing \( x(t) \) only as sampling points, the best one can do about obtaining \( a(f) \) is to compute \( a_{\Delta}(f) \). The latter, it is noted, differs from \( a(f) \) by the sum of the \( a(f)'s \) displaced by all multiples of \( F \). This error is referred to as “aliasing” and, if \( a(f) \) is monotonically decreasing as a function of \( |f| \), there will be an error of roughly 100% at \( f = F/2 \), the Nyquist frequency.

Since \( a_{\Delta}(f) \) is periodic with period \( F \), Fourier’s theorem says it has a series expansion in powers of \( e^{2\pi i j f / F} \) whose coefficients are given by \( (1/F) x(j. \Delta t) \). Thus,

\[
a_{\Delta}(f) = \frac{1}{F} \sum_{j=-\infty}^{\infty} x(j. \Delta t) e^{-2\pi i j f / F}. \tag{66}
\]
FAST FOURIER TRANSFORM ALGORITHM

To consider the effect of sampling in the frequency domain at points \( f_n = n \Delta f \), \( n = 0, \pm 1, \pm 2, \ldots \), where \( \Delta f = 1/T \), write (66),

\[
a_p(n, \Delta f) = \frac{1}{F} \sum_{j=-\infty}^{\infty} x(j, \Delta t) e^{-2\pi i j n / T F},
\]

(67)

Taking \( T F = N \) to be an integer, we use the fact that \( e^{-2\pi i j n / N} \) is a periodic function of \( j \), with period \( N \), to put (67) in the form

\[
a_p(n, \Delta f) = \frac{1}{F} \sum_{j=0}^{N-1} x_p(j \Delta t) e^{-2\pi i j n / N},
\]

(68)

where

\[
x_p(j \Delta t) = \sum_{k=-\infty}^{\infty} x((j + k N) \Delta t).
\]

(69)

The sum in (69) gives the sampled values of

\[
x_p(t) = \sum_{k=-\infty}^{\infty} x(t + k T),
\]

(70)

a periodic function of \( t \), with period \( T \), which is formed from \( x(t) \) in exactly the same way as \( a_p(f) \) is formed from \( a(f) \). Substituting \( \Delta t = T/N \) for \( 1/F \), (68) can be written

\[
a_p(n, \Delta f) = \frac{1}{N} \sum_{j=0}^{N-1} T x_p(j \Delta t) e^{-2\pi i j n / N}.
\]

(71)

This is of the same form as the DFT defined by (2) if we let

\[
X(j) = Tx_p(j \Delta t),
\]

(72)

\[
A(n) = a_p(n, \Delta f).
\]

(73)

This "proves" the central theorem of this section. (A rigorous proof under rigorous conditions is beyond the scope of this paper.)

Theorem 10. If

\[
x(t) \leftrightarrow a(f),
\]

(74)

then

\[
T \cdot x_p(j \Delta t) \leftrightarrow a_p(n, \Delta f), \quad j, n = 0, 1, 2, \ldots, N - 1,
\]

(75)

where \( N = 1/\Delta t \cdot \Delta f \).

An example of the use of this theorem is as follows. Suppose \( x(t) \) is given and one wishes to compute \( a(f) \), where it is assumed that \( x(t) \) and \( a(f) \) are related by (60) and that the integrals (60) and (61) exist. One may select \( F \), a frequency interval, and \( N \), a number of points. The parameters,

\[
\Delta f = F/N,
\]

\[
T = 1/\Delta f,
\]

\[
\Delta t = 1/F,
\]

are all determined. One then generates sampled values, \( x_p(t_j) \) at \( t_j = j \Delta t, j = 0, 1, \ldots, N - 1 \) and lets \( X(j) = Tx_p(t_j) \) be the input sequence to a DFT subroutine. The computed result will be \( A(n) = a_p(f_n) \), the sampled values of \( a_p(f) \) at sampling points \( f_n = n \Delta f \), where \( n = 0, 1, \ldots, N - 1 \).
As an example, the results of a calculation of the Fourier transform of the function
\[ x(t) = e^{-t} \quad \text{for } t > 0, \]
\[ x(t) = 0 \quad \text{for } t < 0, \] (76)
is shown in Figure 1. The correct result is
\[ a(f) = 1/(1 + 2\pi if). \] (77)

With \( F = 2 \) and \( N = 16 \), the remaining parameters are \( T = 8, \Delta t = 1/2, \) and \( \Delta f = 1/8 \). The point values of \( x(t) \), used for the input vector, are indicated by the dots on the upper curve. Aliasing is negligible, since \( x(t) \) is insignificant at \( t = 8 \).

![Figure 1. The calculation of the discrete Fourier transform of the function \( x(t) = 0 \) for \( t < 0 \), \( x(t) = e^{-t} \) for \( t > 0 \) using \( T = 8, N = 16, \Delta t = \frac{1}{2}, \Delta f = \frac{1}{8} \) and \( F = 4 \). Here \( a(f) = 1/(1 + 2\pi if) \).](image)

The two lower solid curves are the real and imaginary parts of \( a(f) \), and the computed results are indicated by the dots. For small \( f \), the dots are quite close to the curve, but they deviate for larger \( f \). At \( f \) near \( F = 2 \), the dots trace out a curve looking like the negative half of the \( a(f) \) curve. This is the aliasing present in the \( a_p(f) \) function. It is clear what remedy one can use. It is merely to take a larger \( F \), meaning a smaller \( \Delta t \). In Figure 2, the same calculation, with \( N = 32 \) is shown. Here, \( F = 4 \) so that aliasing is reduced and a reasonable approximation to \( a(f) \) is obtained over a wider range of \( f \).

The procedure for computing \( a(f) \) when given \( x(t) \) can be derived from the above example. However, a word of caution about the formation of \( a_p(f) \) is in order. A direct summation of
\[ a_p(f) = \sum_{k=-\infty}^{\infty} a(f + kF) \] (78)
may well fail to converge for a function like
\[ a(f) = 1/(1 + 2\pi if). \] (79)

More will be said about this in the next section where Laplace transforms are discussed.
The coefficients of the Fourier series

\[ x(t) = \sum_{n=-\infty}^{\infty} C(n) e^{2\pi i n t / T}, \tag{80} \]

where \( x(t) \) is defined for \( 0 < t < T \), can be obtained in terms of the DFT of the sequence \( x(t_j), t_j = j \cdot \Delta t, j = 0, 1, \ldots, N - 1, \Delta t = T / N \). This can be derived by appealing to theorem 10, but it may be simpler to obtain the results as follows. Consider

\[ x(t_j) = \sum_{n=-\infty}^{\infty} C(n) e^{2\pi i n j / N}. \tag{81} \]

![Graph](image1)

Figure 2. The same function and parameters as Figure 1 except that \( N = 32 \). This halves \( \Delta t \) (i.e. \( \Delta t = \frac{1}{2} \)) and doubles \( F \) (i.e. \( F = 4 \)) so that aliasing errors in the solution are reduced.

Since \( e^{2\pi i j / N} \) is a periodic function of \( j \) with period \( N \), the sum (81) can be expressed

\[ x(t_j) = \sum_{n=0}^{N-1} \sum_{k=-\infty}^{\infty} C(n + kN) e^{2\pi i n j / N}, \]

giving

\[ x(t_j) = \sum_{n=0}^{N-1} C_p(n) e^{2\pi i n j / N}, \tag{82} \]

where \( C_p(n) \) is the now familiar periodized function

\[ C_p(n) = \sum_{k=-\infty}^{\infty} C(n + kN). \tag{83} \]

Consequently, we obtain the following theorem.

**Theorem 11.** Given \( T \) and \( N \), and letting \( t_j = j \cdot \Delta t, \Delta t = T / N \), we have

\[ x(t_j) \leftrightarrow C_p(n). \tag{84} \]
It is seen again that sampling in one domain causes aliasing in the other; sampling \( x(t) \) at \( N \) points in the fixed interval \( 0 < t < T \) gives
\[
C_p(n) = C(n) + \text{error},
\]
where
\[
\text{error} = \sum_{k=1}^{\infty} [C(n + kN) + C(n - kN)].
\]
Thus, there will be aliasing at \( n = kN \) for all \( k \neq 0 \). Two procedures are then suggested: (i) when computing \( C(n) \) from \( x(t) \), one must take \( N \) so large that the error (86) is insignificant; (ii) when computing \( x(t) \) from known values of \( C(n) \), one first takes \( N \) to be the number of points desired, and then computes \( C_p(n) \) according to equation (83). Then one computes its IDFT.

4. THE CALCULATION OF LAPLACE TRANSFORMS

The problem of inverting the Laplace transform, i.e. of determining a time function \( g(t) \) from its Laplace transform,
\[
G(s) = \int_0^\infty g(t) e^{-st} \, dt,
\]
can often be solved analytically by applying a partial fraction expansion or an integration along some contour in the complex \( s \)-plane. When this proves to be too difficult or impossible due to the complexity of the formula for \( G(s) \), or when \( G(s) \) is known only in terms of its numerical values in the complex plane, numerical methods may be necessary. The inherent nature of the problem makes it impossible to suggest a unified approach to the problem. Rather, it happens that methods must be selected which only apply to some classes of functions \( G(s) \), but not all. (For general references, see references 6 and 7.)

There have been two fairly general numerical approaches to the problem. One is to expand the unknown function as a series in a complete set of orthogonal functions. In the case where Laguerre functions are used, it has been shown by Weeks [8] that the expansion coefficients can be expressed as inverse Fourier transforms. Then Wing [9] modified the technique by using the FFT method to compute the inverse Fourier transform. This procedure is, of course, restricted in usefulness to functions which are representable as rapidly convergent series of Laguerre functions. A more general method has been suggested by Dubner and Abate [10]. They expressed the Laplace transform as a Fourier cosine transform whose inverse is expressible as a Fourier cosine series. Both methods will be described here as applications of theorems 10 and 11. The error analysis follows in terms of the aliasing mentioned there. Dubner and Abate neglected to mention the effect of rounding in choosing an optimal parameter \( c \), to be defined below. Here, a formula for an optimal \( c \) is derived which balances aliasing and rounding errors.

In both of the methods described here, it is necessary to use values of \( G(s) \) with the complex variable \( s = c + i\omega \) on a vertical line in the complex plane. This is necessary so that one can use the periodicity in \( \omega \) of the exponential function.

The method devised by Papoulis [6] uses an expansion
\[
g(t) = e^{-ct} \sum_{n=0}^{\infty} C(n) L_n(t)
\]
in Laguerre functions \( L_n(t) \). Substituting this in (87), one gets
\[
G(s) = \sum_{n=0}^{\infty} C(n) \frac{s^n}{(s+1)^{n+1}},
\]
in Laguerre functions \( L_n(t) \). Substituting this in (87), one gets
\[
G(s) = \sum_{n=0}^{\infty} C(n) \frac{s^n}{(s+1)^{n+1}},
\]
for which Papoulis gives simple recursion formulas for the $C(n)$'s in terms of the values of the derivatives of $G(s)$ at $s = 0$.

Weeks' [8] modification of this consisted in letting $s = c + i\omega$ and using the expansion

$$g(t) = e^{(c-1/2)\pi t} \sum_{n=0}^{\infty} C(n) L_n(t/T).$$  \hspace{1cm} (90)

This gives

$$G(s) = \sum_{n=0}^{\infty} C(n) \left( \frac{\omega i - T/2}{\omega i + T/2} \right)^n.$$  \hspace{1cm} (91)

The factor $(\omega i - T/2)/(\omega i + T/2)$ has modulus 1 so one can define $\theta$ such that

$$e^{i\theta} = \frac{\omega i - T/2}{\omega i + T/2},$$  \hspace{1cm} (92)

and obtain a Fourier series

$$\sum_{n=0}^{\infty} C(n) e^{i\theta n}.$$  \hspace{1cm} (93)

Solving (92) one gets $\omega$ in terms of $\theta$;

$$\omega = (T/2) \cot \theta / 2.$$  \hspace{1cm} (94)

O. Wing [9] introduced the FFT algorithm to this method to solve (93) for the $C(n)$'s in terms of the values of the left side of (93).

Letting

$$x(\theta) = (\omega i + T/2) G(c + i\omega),$$  \hspace{1cm} (95)

where $\omega$ is defined by (94), we note first that $x(\theta)$ is periodic in $\theta$ and that theorem 11 applies. However, the Fourier transform of $x(\theta)$, which is the sequence $C(n)$, is going to have aliasing about $n = N$, where $N$ is the number of $\theta$-intervals used. It can be seen that $x(\theta) = \hat{x}(\theta)$ implies that real coefficients $C_p(n)$ will result. Weeks [8] used the even function $x_e(\theta) = x(\theta) + x(-\theta)$ to express the transform as a cosine transform of real sequences. This would permit one to use the cosine-transform-of-real-data form of the FFT algorithm as given in procedure 6 of section 2, but would require doubling $N$ to keep aliasing errors within limits; therefore, procedure 3 giving the transform of conjugate symmetric data to real data would be recommended here. Next, there is the additional problem of choosing the optimal parameters $T$ and $c$. Weeks [8] gives, as an empirical rule, the optimal parameter $T$ by the formula

$$T = \frac{t_{\text{max}}}{N},$$  \hspace{1cm} (96)

where $t_{\text{max}}$ is the maximum value of $t$ for which a solution is desired. For the choice of optimal $c$, he suggests, on the basis of empirical evidence, the value

$$c = \max \left( c_0 + \frac{1}{t_{\text{max}}}, 0 \right),$$  \hspace{1cm} (97)

where $c_0$ is the real part of the rightmost pole of $G(s)$ in the complex plane.

To summarize, the procedure for using the FFT method to obtain values of an inverse Fourier transform in terms of an expansion in Laguerre functions is as follows.

1. Select a scale factor $T$, a constant $c$, and an $N$.
2. Let

$$\Delta \theta = 2\pi / N$$  \hspace{1cm} (98)

and let

$$\theta_j = j \cdot \Delta \theta, \quad j = 0, 1, 2, \ldots, N - 1.$$
(3) Compute

\[ \omega_j = (T/2) \cot \theta_j/2, \quad j = 0, 1, \ldots, N - 1. \]  

(99)

(4) Compute

\[ X_j = (\omega_j i + T/2) G(c + i\omega_j) \quad \text{for } j = 0, 1, \ldots, N - 1. \]  

(100)

(5) Go to the FFT algorithm to solve,

\[ X_j = \sum_{n=0}^{N-1} C_n(n) \exp(2\pi inj/N) \]  

(101)

for the \( C_n(n) \)'s, \( n = 0, 1, 2, \ldots, N - 1 \).

(6) For each value of \( t \) at which the inverse Laplace transform is desired, assume that \( C(n) \approx C_n(n) \) for \( n = 0, 1, \ldots, N - 1 \) and evaluate the series,

\[ g(t) = e^{(c-T/2)t} \sum_{n=0}^{N-1} C_n(n) L_n(t/T). \]  

(102)

The second method for inverting Laplace transforms is almost a direct application of theorem 10. In fact, in examples 1 and 2 of section 3, it was mentioned that the function being calculated could be regarded as a Laplace transform.

As before, we write the Laplace transform of a real function of time \( g(t) \) as

\[ G(s) = \int_0^\infty g(t) e^{-st} \, dt, \]  

(103)

and let \( s = c + i\omega \), where \( c \) and \( \omega \) are real. Defining \( f \) by \( \omega = 2\pi f \), we have

\[ G(c + 2\pi if) = \int_0^\infty g(t) e^{-ct} e^{-2\pi ift} \, dt. \]  

(104)

This is an integral Fourier transform pair where the functions involved are

\[ a(f) = G(c + 2\pi if) \]  

(105)

and

\[ x(t) = g(t) e^{-ct} \quad \text{if } t > 0, \]  

(106)

\[ = 0 \quad \text{if } t < 0. \]

The entire procedure for computing one of these functions when knowing values of the other function is as given before in section 3, examples 1 and 2. Aliasing and error considerations are the same. However, there is one new parameter \( c \), which may be chosen by the user to optimize the calculation. Theoretically, \( c \) need only be taken larger than \( c_0 \), the maximum of the real parts of the poles of \( G(s) \). However, there is a finite range of values of \( c \) which are useful in terms of numerical accuracy and computational effort. To see that the range of useful \( c \)-values is finite, we may observe first that the effect of large \( c \) is to attenuate \( x(t) \), having the favorable effect of diminishing the aliasing in \( x(t) \). The unfavorable consequence of using a large \( c \) is that for \( t \) also large, multiplying \( x(t) \) by \( e^{-ct} \) to get \( g(t) \) increases the effect of rounding errors in \( x(t) \).

Before making a quantitative analysis of this error, we will first describe another source of error. This is in the calculation of a \( a_s(f) \), the periodized \( a(f) \) function. For example, consider the common situation where the given Laplace transform behaves asymptotically like \( 1/s \) for large \( s \). In this case, the infinite sum in the evaluation of \( a_s(f) \) may have unbounded errors as more terms are taken. To deal with this problem, a slight variation on the above procedure can be made.
FAST FOURIER TRANSFORM ALGORITHM

To do this, substitute (106) into (103) to obtain \( a(f) \) as a Fourier transform

\[
a(f) = \int_0^\infty x(t) e^{-2\pi i ft} \, dt. \tag{107}
\]

Using the fact that \( x(t) \) is real, the conjugate of \( a(f) \) is

\[
\tilde{a}(f) = \int_0^\infty x(t) e^{2\pi i ft} \, dt, \tag{108}
\]

which, added to \( a(f) \), gives

\[
\text{Re} \, a(f) = \int_0^\infty x(t) \cos 2\pi ft \, dt. \tag{109}
\]

Subtracting, we can also get

\[
\text{Im} \, a(f) = \int_0^\infty x(t) \sin 2\pi ft \, dt. \tag{110}
\]

Procedures 5 and 6 of section 2 describe how one can compute the cosine and sine transforms. Manipulation of these integrals yields the complex integral transforms in the forms

\[
\text{Re} \, a(f) = \int_{-\infty}^{\infty} x_e(t) e^{-2\pi i ft} \, dt, \tag{111}
\]

\[
\text{Im} \, a(f) = \int_{-\infty}^{\infty} x_o(t) e^{-2\pi i ft} \, dt, \tag{112}
\]

where

\[
x_e(t) = \frac{1}{2} [x(t) + x(-t)] \tag{113}
\]

and

\[
x_o(t) = \frac{1}{2} [x(t) - x(-t)] \tag{114}
\]

are the even and odd parts of \( x(t) \), respectively. Examples 1 and 2 of section 3 treated the case where \( c = 1 \) in the function

\[
G(s) = \frac{1}{s}, \quad s = c + i\omega, \quad \omega = 2\pi f.
\]

This can be written

\[
G(s) = \frac{c}{c^2 + \omega^2} - i \frac{\omega}{c^2 + \omega^2}, \tag{115}
\]

giving

\[
\text{Re} \, a(f) = \frac{c}{c^2 + (2\pi f)^2} \tag{116}
\]

and

\[
\text{Im} \, a(f) = -\frac{2\pi f}{c^2 + (2\pi f)^2}. \tag{117}
\]

The sum

\[
\text{Re} \, a_p(f) = \sum_{k=-\infty}^{\infty} \text{Re} \, a(f + kF) \tag{117}
\]

\[
= \sum_{k=-\infty}^{\infty} \frac{1}{c^2 + (2\pi)^2 (f + kF)^2}
\]

287
FAST FOURIER TRANSFORM ALGORITHM

To do this, substitute (106) into (103) to obtain \( a(f) \) as a Fourier transform

\[
a(f) = \int_0^\infty x(t) e^{-2\pi ift} \, dt. \tag{107}
\]

Using the fact that \( x(t) \) is real, the conjugate of \( a(f) \) is

\[
\tilde{a}(f) = \int_0^\infty x(t) e^{2\pi ift} \, dt, \tag{108}
\]

which, added to \( a(f) \), gives

\[
\text{Re} \, a(f) = \int_0^\infty x(t) \cos 2\pi ft \, dt. \tag{109}
\]

Subtracting, we can also get

\[
\text{Im} \, a(f) = \int_0^\infty x(t) \sin 2\pi ft \, dt. \tag{110}
\]

Procedures 5 and 6 of section 2 describe how one can compute the cosine and sine transforms. Manipulation of these integrals yields the complex integral transforms in the forms

\[
\text{Re} \, a(f) = \int_{-\infty}^\infty x_e(t) e^{-2\pi ift} \, dt, \tag{111}
\]

\[
\text{Im} \, a(f) = \int_{-\infty}^\infty x_o(t) e^{-2\pi ift} \, dt, \tag{112}
\]

where

\[
x_e(t) = \frac{1}{2} [x(t) + x(-t)] \tag{113}
\]

and

\[
x_o(t) = \frac{1}{2} [x(t) - x(-t)] \tag{114}
\]

are the even and odd parts of \( x(t) \), respectively.

Examples 1 and 2 of section 3 treated the case where \( c = 1 \) in the function

\[
G(s) = 1/s, \quad s = c + i\omega, \quad \omega = 2pf.
\]

This can be written

\[
G(s) = \frac{c}{c^2 + \omega^2} - i \frac{\omega}{c^2 + \omega^2}, \tag{115}
\]

giving

\[
\text{Re} \, a(f) = \frac{c}{c^2 + (2pf)^2} \tag{116}
\]

and

\[
\text{Im} \, a(f) = -\frac{2\pi f}{c^2 + (2pf)^2}.
\]

The sum

\[
\text{Re} \, a\nu(f) = \sum_{k=-\infty}^{\infty} \text{Re} \, a(f + kF) \tag{117}
\]

\[
= \sum_{k=-\infty}^{\infty} \frac{1}{c^2 + (2\pi k)^2(f + kF)^2}
\]
FAST FOURIER TRANSFORM ALGORITHM

We will next consider the error due to aliasing in \( x(t) \) and how it affects the optimal choice of parameters. Since the function \( x_e(t) \) is even, the periodized function \( x_{ep}(t) \) will be redundant in its upper half with \( x_{ep}(t) = x_{ep}(T - t) \) for \( T/2 \leq t < T \). Thus, we consider only \( 0 \leq t < T/2 \). We first assume that \( T \) is sufficiently large so that aliasing in \( x_e(t) \) in the interval \( 0 \leq t < T/2 \) is due only to the \( k = -1 \) term of the series for \( x_{ep}(t) \). Thus,

\[
x_{ep}(t) \approx x_e(t) + x_e(t - T),
\]

which, in the interval \( 0 \leq t < T/2 \), is

\[
x_{ep}(t) \approx \frac{1}{2} [x(t) + x(T - t)].
\]

When we multiply this by \( 2 e^{\tau t} \), we get

\[
2 e^{\tau t} x_{ep}(t) \approx e^{\tau t} x(t) + e^{\tau t} x(T - t)
\]

\[
= g(t) + e^{-\tau (T - 2t)} g(T - t).
\]

Thus, the aliasing error in the computed result is

\[
E_d(t) = e^{-\tau (T - 2t)} g(T - t).
\]

This is a decreasing function of \( \tau \) and, with a given error criterion, \( \epsilon \), can be used to obtain a lower bound on \( \tau \). On the other hand, an upper bound is imposed on \( \tau \) by the fact that the computed \( x(t) \), with its rounding error \( r(t) \), will have to be multiplied by \( e^{\tau t} \) to obtain \( g(t) \). Thus, the rounding error in \( g(t) \) will be

\[
E_r(t) = e^{\tau t} r(t),
\]

an increasing function of \( \tau \). The optimal \( \tau \), therefore, can be taken to be the value at which the maxima of the two errors, on an interval \( 0 \leq t < \tau \), are equal. Let

\[
\epsilon_1 = \max |E_d(t)| = e^{-\tau (T - 2\tau)} \hat{g},
\]

\[
\epsilon_2 = \max |E_r(t)| = e^{\tau \hat{r}},
\]

where

\[
\hat{g} = \max g(T - t)
\]

and

\[
\hat{r} = \max r(t),
\]

and where all maxima refer to the interval \( 0 < t < \tau \). Equating (132) and (133), we solve for \( \tau \), getting

\[
\tau = \frac{\ln(\hat{g}/\hat{r})}{T - \tau}.
\]

Substituting this expression for \( \tau \) in (103) we obtain

\[
\epsilon = \epsilon_1 + \epsilon_2 = 2\epsilon_1 = 2\epsilon_2 = 2\hat{g}^{1/3} \hat{r}^{2/3} T^{(T - 2\tau)/(T - \tau)}.
\]

This error assumes the values,

\[
\epsilon = 2\hat{g}^{1/3} \hat{r}^{2/3} \quad \text{at } \tau = 0,
\]

\[
\epsilon = 2\hat{g}^{1/3} \hat{r}^{2/3} \quad \text{at } \tau = T/4,
\]

\[
\epsilon = 2\hat{g} \quad \text{at } \tau = T/2.
\]

This shows that at \( T/2 \), the error is as large as \( g(t) \) itself for all \( \tau \). At smaller \( t \)-values, the error contains two factors: \( \hat{g} \), arising from aliasing; and \( \hat{r} \), arising from rounding. Therefore, the
This is probably because the errors due to rounding are subject to random fluctuations and the value of \( c \) is not too critical within ranges of values as large as 0.2.

### Table 2

Errors in the calculation of \( g(t) = J_0(t) \), where \( G(s) = 1/\sqrt{1 + s^2} \)

<table>
<thead>
<tr>
<th>( t )</th>
<th>( g(t) )</th>
<th>( c = 0.3 )</th>
<th>( c = 0.5 )</th>
<th>( c = 0.7 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00000</td>
<td>15.1</td>
<td>25.4</td>
<td>35.5</td>
</tr>
<tr>
<td>1</td>
<td>0.76520</td>
<td>0.0</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>2</td>
<td>0.2389</td>
<td>0.2</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>3</td>
<td>-0.26005</td>
<td>0.7</td>
<td>0.1</td>
<td>0.2</td>
</tr>
<tr>
<td>4</td>
<td>-0.39715</td>
<td>0.5</td>
<td>0.1</td>
<td>0.4</td>
</tr>
<tr>
<td>5</td>
<td>-0.17760</td>
<td>1.1</td>
<td>0.2</td>
<td>0.9</td>
</tr>
<tr>
<td>6</td>
<td>0.15064</td>
<td>4.0</td>
<td>0.3</td>
<td>1.8</td>
</tr>
<tr>
<td>7</td>
<td>0.30008</td>
<td>4.3</td>
<td>0.5</td>
<td>3.8</td>
</tr>
<tr>
<td>8</td>
<td>0.17165</td>
<td>4.8</td>
<td>1.3</td>
<td>7.6</td>
</tr>
</tbody>
</table>

Equations (154) and (153) predict an error bound of \( \epsilon = 1.4 \times 10^{-4} \) at the optimal \( c = 0.48 \). Values of \( G(s) \) with \( f \) up to \( F = 20 \) were used.

### Table 3

Errors in the calculation of \( g(t) = t \), where \( G(s) = 1/s^2 \)

<table>
<thead>
<tr>
<th>( t )</th>
<th>( g(t) )</th>
<th>( c = 0.4 )</th>
<th>( c = 0.6 )</th>
<th>( c = 0.8 )</th>
<th>( c = 1.0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>52.5</td>
<td>50.7</td>
<td>50.7</td>
<td>50.7</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2.9</td>
<td>0.0</td>
<td>0.0</td>
<td>0.1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>9.9</td>
<td>0.2</td>
<td>0.3</td>
<td>0.8</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>20.1</td>
<td>0.5</td>
<td>0.8</td>
<td>2.1</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>41.9</td>
<td>1.1</td>
<td>1.7</td>
<td>6.0</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>88.6</td>
<td>3.0</td>
<td>4.4</td>
<td>16.3</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>188.3</td>
<td>7.5</td>
<td>10.5</td>
<td>42.9</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>400.9</td>
<td>21.0</td>
<td>24.6</td>
<td>116.9</td>
</tr>
</tbody>
</table>

Equations (156) and (155) predict an error bound of \( \epsilon = 17.2 \times 10^4 \) at the optimal \( c = 0.79 \). Values of \( G(s) \) with \( f \) up to \( F = 20 \) were used.

### 5. SUMMARY

The fast Fourier transform method (FFT) is an algorithm for computing, systematically and rapidly, the complex Fourier series. Programs using the basic idea of this algorithm for special situations where the data is real and/or cosine or sine transforms are required can be written. Instead, however, it is suggested here that a highly optimized subroutine or special hardware device should be made to compute the complex Fourier series only. This can be used with pre- and post-processing of the data to compute any of a variety of transforms, including cosine and sine transforms.
error criterion should be applied to a smaller range of \( t \). A reasonable choice is to optimize the solution in the range \( 0 < t < T/4 \). For this, one can take the \( c \) given by (134) for \( \tau = T/4 \),

\[
c = (4/3T) \ln \left( \frac{\hat{g}}{\hat{\tau}} \right), \tag{139}
\]
to obtain a solution with an error less than

\[
\epsilon = 2\hat{g}^{1/3} \hat{\tau}^{2/3}. \tag{140}
\]

As an application of formulas (139) and (140), let us first consider the problem in section 2 which treated the Laplace transform pair

\[
G(s) = 1/s, \quad g(t) = u(t), \tag{141}
\]

where here, and in what follows, \( u(t) \) is the step function

\[
u(t) = \begin{cases} 
0 & t < 0 \\
0.5 & t = 0 \\
1 & t > 0.
\end{cases} \tag{142}
\]

In section 3 it was mentioned that this problem is equivalent to computing the Laplace transform of \( u(t) \) with \( s = 1 + j\omega \).

The numerical examples chosen here are designed to display the effects of the two sources of error. First, to determine an optimal \( c \), we use the empirical result that errors in the inverse Fourier transform, \( X(j) \), in single precision on the IBM 360 mod 67 are bounded by \( N \times 10^{-6} \). The computed IDFT gives values of

\[
X(j) = T x_t(t_j) = T/2. x(t_j). \tag{143}
\]

Consequently, the error in \( x(t) \) must be bounded by

\[
\hat{\tau} \approx (2N/T) 10^{-6} = 2F. 10^{-6}. \tag{144}
\]

The parameters of the second part of the example in section 3 are \( T = 8, N = 32 \). These determine the parameters \( \Delta t = 1/4, F = 4, \Delta f = 1/8 \). In this example, \( \hat{g} = 1 \), and (144) gives \( \hat{\tau} \approx 8 \times 10^{-6} \), resulting in the estimate of \( \epsilon \), from (140),

\[
\epsilon = 2(8 \times 10^{-6})^{2/3} = 8 \times 10^{-4}. \tag{145}
\]

From equation (139), we obtain

\[
c = (4/3T) \ln (10^6/8) = 15.7/T, \tag{146}
\]

which, with \( T = 8 \) gives \( c \approx 2 \).

Thus, for \( G(s) = 1/s \), using \( c = 2, T = 8 \), and \( N = 32 \) should give the inverse Laplace transform in the range \( 0 < t < T/4 = 2 \) with an error of no more than eight in the fourth digit.

To display further the effect of the choice of \( c \), numerical calculations were carried out with the following transform pairs:

\[
g(t) = u(t - 4) \leftrightarrow G(s) = \exp(-4s)/s; \tag{147}
\]

\[
g(t) = J_0(t) \leftrightarrow G(s) = 1/\sqrt{1 + s^2}; \tag{148}
\]

\[
g(t) = t \leftrightarrow G(s) = 1/s^2. \tag{149}
\]

All three cases are run with \( T = 32 \) and \( N = 32 \). These determine the values

\[
\Delta t = 1, \quad F = 1, \quad \Delta f = 1/32 \tag{150}
\]

and, from (139),

\[
\hat{\tau} = 2F \times 10^{-6} = 2 \times 10^{-6}.
\]
FAST FOURIER TRANSFORM ALGORITHM

For the first function (147), we have \( \hat{g} = 1 \) and obtain, from (139),

\[
c = (4/3T) \ln(\hat{g}/\hat{f}) = 0.55,
\]

(151)

and, from (140),

\[
e = 2\hat{g}^{1/3} \rho^{2/3} = 2.6 \times 10^{-4}.
\]

(152)

The errors in the calculated values are given in Table 1 for \( c = 0.4, 0.55, \) and \( 0.7 \). The best results are obtained with the predicted value of \( c = 0.55 \); for the other values of \( c \), errors accumulate for large \( t \).

**Table 1**

*Errors in the calculation of \( g(t) = u(t - 4) \), where \( G(s) = \exp(-4s)/s \), using \( T = 32 \), \( N = 32 \), and \( c = 0.4, 0.55, \) and \( 0.7 \)*

<table>
<thead>
<tr>
<th>( t )</th>
<th>( g(t) )</th>
<th>( c = 0.4 )</th>
<th>( c = 0.55 )</th>
<th>( c = 0.7 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.4</td>
<td>0.2</td>
<td>0.1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.6</td>
<td>0.3</td>
<td>0.2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1.1</td>
<td>0.7</td>
<td>0.5</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2.7</td>
<td>2.0</td>
<td>1.9</td>
</tr>
<tr>
<td>4</td>
<td>0.5</td>
<td>1.2</td>
<td>0.0</td>
<td>0.5</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>2.9</td>
<td>5.2</td>
<td>6.5</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0.3</td>
<td>4.1</td>
<td>6.0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>4.7</td>
<td>3.9</td>
<td>7.1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>13.7</td>
<td>3.6</td>
<td>10.7</td>
</tr>
</tbody>
</table>

Equations (152) and (151) predict an error bound of \( e = 2.6 \times 10^{-4} \) at the optimal \( c = 0.55 \). Values of \( G(s) \) with \( f \) up to \( F = 160 \) were used.

For the second function (148), the aliasing error will be smaller since \( J_0(t) \) is a decreasing function of \( t \). The bound \( |J_0(t)| < \sqrt{2/\pi t} \approx 0.8/\sqrt{t} \) is applied to the range \( T/2 < t < T \) giving

\[
\hat{g} = 0.8/\sqrt{T/2} = 0.2.
\]

This gives

\[
c = (4/96) \ln(0.2) + 0.55 = 0.48
\]

(153)

and

\[
e = 0.2^{1/3} \times 2.6 \times 10^{-4} = 1.4 \times 10^{-4}.
\]

(154)

The computed values of \( g(t) \) in this case are seen in Table 2 to lie within the predicted error bound for the optimal \( c \) for all \( t \) except \( t = 0 \). The discrepancy at \( t = 0 \) is due to the truncation of the series for \( a_0(f) \). The error in \( a_0(f) \) is almost a constant function of \( f \) and, therefore, affects almost exclusively, the value of \( g(t) \) at \( t = 0 \) and has little effect on other values of \( t \).

In the third case, (149), the fact that \( g(t) \) is increasing means that aliasing will be more serious and a larger \( c \) will be required to attenuate it. Taking the bound \( \hat{g} = 32 \), we have

\[
c = (4/96) \ln(32) + 0.55 = 0.79
\]

(155)

and

\[
e = 32^{1/3} \times 2.6 \times 10^{-4} = 17.2 \times 10^{-4}.
\]

(156)

The table of errors for \( g(t) = t \) are given in Table 3. Here, the results obtained with \( c = 0.6 \) are a little better than those obtained with \( c = 0.8 \) although the predicted optimal value is
FAST FOURIER TRANSFORM ALGORITHM

With a set of subroutines capable of computing sine, cosine, and Fourier transforms, one is then able, in a fairly straightforward manner, to compute and invert Laplace transforms. However, the method has three factors which limit its accuracy: one is due to the necessity of truncating the infinite sum (78) which defines $a_n(f)$; another results from rounding and depends on the number of significant digits carried by the computer being used, and the third comes from the aliasing due to the use of sampled values of function. The present analysis deals with each of these problems and results in a formula for selecting an optimal value of the parameter $c = \text{Re}(s)$, where $s$ is the Laplace transform variable.

It is hoped that the results obtained will permit computer programs automatically to select optimal parameters and operate efficiently on a wide class of functions. Such programs would provide important subroutines in larger programs for such problems as circuit analysis and the solution of systems of differential equations.

REFERENCES