Chapter C06

Summation of Series

Contents

1 Scope of the Chapter

This chapter is concerned with the following tasks.

(a) Calculating the **discrete Fourier transform** of a sequence of real or complex data values.
(b) Calculating the **discrete convolution** or the **discrete correlation** of two sequences of real or complex data values using discrete Fourier transforms.
(c) Calculating the **inverse Laplace transform** of a user-supplied function.
(d) Direct summation of orthogonal series.
(e) Acceleration of convergence of a sequence of real values.

2 Background to the Problems

2.1 Discrete Fourier Transforms

2.1.1 Complex transforms

Most of the routines in this chapter calculate the finite **discrete Fourier transform** (DFT) of a sequence of n complex numbers z_j, for $j = 0, 1, \ldots, n-1$. The transform is defined by

$$\hat{z}_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} z_j \exp\left(-i\frac{2\pi jk}{n}\right) \tag{1}$$

for $k = 0, 1, \ldots, n-1$. Note that equation (1) makes sense for all integral k and with this extension $\hat{z}_k$ is periodic with period n, i.e., $\hat{z}_k = \hat{z}_{k\pm n}$, and in particular $\hat{z}_{-k} = \hat{z}_{n-k}$. Note also that the scale-factor of $\frac{1}{\sqrt{n}}$ may be omitted in the definition of the DFT, and replaced by $\frac{1}{n}$ in the definition of the inverse.

If we write $z_j = x_j + iy_j$ and $\hat{z}_k = a_k + ib_k$, then the definition of $\hat{z}_k$ may be written in terms of sines and cosines as

$$a_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} \left(x_j \cos\left(\frac{2\pi jk}{n}\right) + y_j \sin\left(\frac{2\pi jk}{n}\right)\right)$$

$$b_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} \left(y_j \cos\left(\frac{2\pi jk}{n}\right) - x_j \sin\left(\frac{2\pi jk}{n}\right)\right).$$

The original data values z_j may conversely be recovered from the transform $\hat{z}_k$ by an **inverse discrete Fourier transform**:

$$z_j = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} \hat{z}_k \exp\left(+i\frac{2\pi jk}{n}\right) \tag{2}$$

for $j = 0, 1, \ldots, n-1$. If we take the complex conjugate of (2), we find that the sequence $\bar{z}_j$ is the DFT of the sequence $\bar{\hat{z}}_k$. Hence the inverse DFT of the sequence $\hat{z}_k$ may be obtained by taking the complex conjugates of the $\hat{z}_k$; performing a DFT; and taking the complex conjugates of the result. (Note that the terms **forward** transform and **backward** transform are also used to mean the direct and inverse transforms respectively.)

The definition (1) of a one-dimensional transform can easily be extended to multi-dimensional transforms. For example, in two dimensions we have

$$\hat{z}_{k_1k_2} = \frac{1}{\sqrt{n_1 n_2}} \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} z_{j_1j_2} \exp\left(-i\frac{2\pi j_1k_1}{n_1}\right) \exp\left(-i\frac{2\pi j_2k_2}{n_2}\right).$$

Note. Definitions of the discrete Fourier transform vary. Sometimes (2) is used as the definition of the DFT, and (1) as the definition of the inverse.

2.1.2 Real transforms

If the original sequence is purely real valued, i.e., $z_j = x_j$, then

$$\hat{z}_k = a_k + ib_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j \exp\left(-i\frac{2\pi jk}{n}\right)$$

and $\hat{z}_{n-k}$ is the complex conjugate of $\hat{z}_k$. Thus the DFT of a real sequence is a particular type of complex sequence, called a **Hermitian** sequence, or **half-complex** or **conjugate symmetric**, with the properties

$$a_{n-k} = a_k \qquad b_{n-k} = -b_k \qquad b_0 = 0$$

and, if n is even, $b_{n/2} = 0$.

Thus a Hermitian sequence of n complex data values can be represented by only n, rather than $2n$, independent real values. This can obviously lead to economies in storage, with two schemes being used in this chapter. In the first scheme, which will be referred to as the **real storage format** for Hermitian sequences, the real parts a_k for $0 \le k \le n/2$ are stored in normal order in the first $n/2+1$ locations of an array X of length n; the corresponding non-zero imaginary parts are stored in reverse order in the remaining locations of X. To clarify, if X is declared with bounds $(0{:}n-1)$ in your calling (sub)program, the following two tables illustrate the storage of the real and imaginary parts of $\hat{z}_k$ for the two cases: n even and n odd.

If n is even then the sequence has two purely real elements and is stored as follows:

Index of X	0	1	2	...	$n/2$	...	$n-2$	$n-1$
Sequence	a_0	$a_1 + \imath b_1$	$a_2 + \imath b_2$	...	$a_{n/2}$	...	$a_2 - \imath b_2$	$a_1 - \imath b_1$
Stored values	a_0	a_1	a_2	...	$a_{n/2}$	...	b_2	b_1

$\mathrm{X}(k) = a_k$, for $k = 0, 1, \ldots, n/2$, and
$\mathrm{X}(n-k) = b_k$, for $k = 1, 2, \ldots, n/2 - 1$.

If n is odd then the sequence has one purely real element and, letting $n = 2s+1$, is stored as follows:

Index of X	0	1	2	...	s	$s+1$	...	$n-2$	$n-1$
Sequence	a_0	$a_1 + \imath b_1$	$a_2 + \imath b_2$	...	$a_s + \imath b_s$	$a_s - \imath b_s$	...	$a_2 - \imath b_2$	$a_1 - \imath b_1$
Stored values	a_0	a_1	a_2	...	a_s	b_s	...	b_2	b_1

$\mathrm{X}(k) = a_k$, for $k = 0, 1, \ldots, s$, and
$\mathrm{X}(n-k) = b_k$, for $k = 1, 2, \ldots, s$.

The second storage scheme, referred to in this chapter as the **complex storage format** for Hermitian sequences, stores the real and imaginary parts a_k, b_k, for $0 \le k \le n/2$, in consecutive locations of an array X of length $n+2$. If X is declared with bounds $(0{:}n+1)$ in your calling (sub)program, the following two tables illustrate the storage of the real and imaginary parts of $\hat{z}_k$ for the two cases: n even and n odd.

If n is even then the sequence has two purely real elements and is stored as follows:

Index of X	0	1	2	3	...	$n-2$	$n-1$	n	$n+1$
Stored values	a_0	$b_0 = 0$	a_1	b_1	...	$a_{n/2-1}$	$b_{n/2-1}$	$a_{n/2}$	$b_{n/2} = 0$

$\mathrm{X}(2 * k) = a_k$, for $k = 0, 1, \ldots, n/2$, and
$\mathrm{X}(2 * k + 1) = b_k$, for $k = 0, 1, \ldots, n/2$.

If n is odd then the sequence has one purely real element and, letting $n = 2s + 1$, is stored as follows:

Index of X	0	1	2	3	...	$n-2$	$n-1$	n	$n+1$
Stored values	a_0	$b_0 = 0$	a_1	b_1	...	b_{s-1}	a_s	b_s	0

$\mathrm{X}(2 * k) = a_k$, for $k = 0, 1, \ldots, s$, and
$\mathrm{X}(2 * k + 1) = b_k$, for $k = 0, 1, \ldots, s$.

Also, given a Hermitian sequence, the inverse (or backward) discrete transform produces a real sequence. That is,

$$x_j = \frac{1}{\sqrt{n}} \left(a_0 + 2 \sum_{k=1}^{n/2-1} \left(a_k \cos\left(\frac{2\pi jk}{n}\right) - b_k \sin\left(\frac{2\pi jk}{n}\right)\right) + a_{n/2} \right)$$

where $a_{n/2} = 0$ if n is odd.

2.1.3 Real symmetric transforms

In many applications the sequence x_j will not only be real, but may also possess additional symmetries which we may exploit to reduce further the computing time and storage requirements. For example, if the sequence x_j is **odd**, $(x_j = -x_{n-j})$, then the discrete Fourier transform of x_j contains only sine terms. Rather than compute the transform of an odd sequence, we define the **sine transform** of a real sequence by

$$\hat{x}_k = \sqrt{\frac{2}{n}} \sum_{j=1}^{n-1} x_j \sin\left(\frac{\pi jk}{n}\right),$$

which could have been computed using the Fourier transform of a real odd sequence of length $2n$. In this case the x_j are arbitrary, and the symmetry only becomes apparent when the sequence is extended. Similarly we define the **cosine transform** of a real sequence by

$$\hat{x}_k = \sqrt{\frac{2}{n}} \left(\frac{1}{2}x_0 + \sum_{j=1}^{n-1} x_j \cos\left(\frac{\pi jk}{n}\right) + \frac{1}{2}(-1)^k x_n \right)$$

which could have been computed using the Fourier transform of a real **even** sequence of length $2n$.

In addition to these 'half-wave' symmetries described above, sequences arise in practice with 'quarter-wave' symmetries. We define the **quarter-wave sine transform** by

$$\hat{x}_k = \frac{1}{\sqrt{n}} \left(\sum_{j=1}^{n-1} x_j \sin\left(\frac{\pi j(2k-1)}{2n}\right) + \frac{1}{2}(-1)^{k-1} x_n \right)$$

which could have been computed using the Fourier transform of a real sequence of length $4n$ of the form

$$(0, x_1, \ldots, x_n, x_{n-1}, \ldots, x_1, 0, -x_1, \ldots, -x_n, -x_{n-1}, \ldots, -x_1).$$

Similarly we may define the **quarter-wave cosine transform** by

$$\hat{x}_k = \frac{1}{\sqrt{n}} \left(\frac{1}{2}x_0 + \sum_{j=1}^{n-1} x_j \cos\left(\frac{\pi j(2k-1)}{2n}\right) \right)$$

which could have been computed using the Fourier transform of a real sequence of length $4n$ of the form

$$(x_0, x_1, \ldots, x_{n-1}, 0, -x_{n-1}, \ldots, -x_0, -x_1, \ldots, -x_{n-1}, 0, x_{n-1}, \ldots, x_1).$$

2.1.4 Fourier integral transforms

The usual application of the discrete Fourier transform is that of obtaining an approximation of the **Fourier integral transform**

$$F(s) = \int_{-\infty}^{\infty} f(t) \exp(-i2\pi st)\, dt$$

when $f(t)$ is negligible outside some region $(0, c)$. Dividing the region into n equal intervals we have

$$F(s) \cong \frac{c}{n} \sum_{j=0}^{n-1} f_j \exp(-i2\pi sjc/n)$$

and so

$$F_k \cong \frac{c}{n} \sum_{j=0}^{n-1} f_j \exp(-i2\pi jk/n)$$

for $k = 0, 1, \ldots, n-1$, where $f_j = f(jc/n)$ and $F_k = F(k/c)$.

Hence the discrete Fourier transform gives an approximation to the Fourier integral transform in the region $s = 0$ to $s = n/c$.

If the function $f(t)$ is defined over some more general interval (a, b), then the integral transform can still be approximated by the discrete transform provided a shift is applied to move the point a to the origin.

2.1.5 Convolutions and correlations

One of the most important applications of the discrete Fourier transform is to the computation of the discrete **convolution** or **correlation** of two vectors x and y defined (as in Brigham [1]) by

$$\text{convolution: } z_k = \sum_{j=0}^{n-1} x_j y_{k-j}$$

$$\text{correlation: } w_k = \sum_{j=0}^{n-1} \bar{x}_j y_{k+j}$$

(Here x and y are assumed to be periodic with period n.)

Under certain circumstances (see Brigham [1]) these can be used as approximations to the convolution or correlation integrals defined by

$$z(s) = \int_{-\infty}^{\infty} x(t) y(s-t)\, dt$$

and

$$w(s) = \int_{-\infty}^{\infty} \bar{x}(t) y(s+t)\, dt, \quad -\infty < s < \infty.$$

For more general advice on the use of Fourier transforms, see Hamming [5]; more detailed information on the fast Fourier transform algorithm can be found in Gentleman and Sande [4] and Brigham [1].

2.1.6 Applications to solving partial differential equations (PDEs)

A further application of the fast Fourier transform, and in particular of the Fourier transforms of symmetric sequences, is in the solution of elliptic PDEs. If an equation is discretised using finite differences, then it is possible to reduce the problem of solving the resulting large system of linear equations to that of solving a number of tridiagonal systems of linear equations. This is accomplished by uncoupling the equations using Fourier transforms, where the nature of the boundary conditions determines the choice of transforms − see Section 3.3. Full details of the Fourier method for the solution of PDEs may be found in Swarztrauber [7], [8].

2.2 Inverse Laplace Transforms

Let $f(t)$ be a real function of t, with $f(t) = 0$ for $t < 0$, and be piecewise continuous and of exponential order α, i.e.,

$$|f(t)| \leq Me^{\alpha t}$$

for large t, where α is the minimal such exponent.

The Laplace transform of $f(t)$ is given by

$$F(s) = \int_0^\infty e^{-st} f(t)\, dt, \quad t > 0$$

where $F(s)$ is defined for $\mathrm{Re}(s) > \alpha$.

The inverse transform is defined by the Bromwich integral

$$f(t) = \frac{1}{2\pi i} \int_{a-i\infty}^{a+i\infty} e^{st} F(s)\, ds, \;\; t > 0.$$

The integration is performed along the line $s = a$ in the complex plane, where $a > \alpha$. This is equivalent to saying that the line $s = a$ lies to the right of all singularities of $F(s)$. For this reason, the value of α is crucial to the correct evaluation of the inverse. It is not essential to know α exactly, but an upper bound must be known.

The problem of determining an inverse Laplace transform may be classified according to whether (a) $F(s)$ is known for real values only, or (b) $F(s)$ is known in functional form and can therefore be calculated for complex values of s. Problem (a) is very ill-defined and no routines are provided. Two methods are provided for problem (b).

2.3 Direct Summation of Orthogonal Series

For any series of functions ϕ_i which satisfy a recurrence

$$\phi_{r+1}(x) + \alpha_r(x)\phi_r(x) + \beta_r(x)\phi_{r-1}(x) = 0$$

the sum

$$\sum_{r=0}^{n} a_r \phi_r(x)$$

is given by

$$\sum_{r=0}^{n} a_r \phi_r(x) = b_0(x)\phi_0(x) + b_1(x)(\phi_1(x) + \alpha_0(x)\phi_0(x))$$

where

$$b_r(x) + \alpha_r(x) b_{r+1}(x) + \beta_{r+1}(x) b_{r+2}(x) = a_r b_{n+1}(x) = b_{n+2}(x) = 0.$$

This may be used to compute the sum of the series. For further reading, see Hamming [5].

2.4 Acceleration of Convergence

This device has applications in a large number of fields, such as summation of series, calculation of integrals with oscillatory integrands (including, for example, Hankel transforms), and root-finding. The mathematical description is as follows. Given a sequence of values $\{s_n\}$, $n = m, m+1, m+2, \ldots, m+2l$ then, except in certain singular cases, parameters, a, b_i, c_i may be determined such that

$$s_n = a + \sum_{i=1}^{l} b_i c_i^n.$$

If the sequence $\{s_n\}$ converges, then a may be taken as an estimate of the limit. The method will also find a pseudo-limit of certain divergent sequences – see Shanks [6] for details.

To use the method to sum a series, the terms s_n of the sequence should be the partial sums of the series, e.g.,, $s_n = \sum_{k=1}^{n} t_k$, where t_k is the kth term of the series. The algorithm can also be used to some

advantage to evaluate integrals with oscillatory integrands; one approach is to write the integral (in this case over a semi-infinite interval) as

$$\int_0^\infty f(x)\,dx = \int_0^{a_1} f(x)\,dx + \int_{a_1}^{a_2} f(x)\,dx + \int_{a_2}^{a_3} f(x)\,dx + \ldots$$

and to consider the sequence of values

$$s_1 = \int_0^{a_1} f(x)\,dx;\ s_2 = \int_0^{a_2} f(x)\,dx = s_1 + \int_{a_1}^{a_2} f(x)\,dx, \text{etc},$$

where the integrals are evaluated using standard quadrature methods. In choosing the values of the a_k, it is worth bearing in mind that C06BAF converges much more rapidly for sequences whose values oscillate about a limit. The a_k should thus be chosen to be (close to) the zeros of $f(x)$, so that successive contributions to the integral are of opposite sign. As an example, consider the case where $f(x) = M(x)\sin x$ and $M(x) > 0$: convergence will be much improved if $a_k = k\pi$ rather than $a_k = 2k\pi$.

3 Recommendations on Choice and Use of Available Routines

Note. Refer to the Users' Note for your implementation to check that a routine is available.

3.1 One-dimensional Fourier Transforms

The choice of routine is determined first of all by whether the data values constitute a real, Hermitian or general complex sequence. It is wasteful of time and storage to use an inappropriate routine. The choice is next determined by the users preferred storage format; where it is preferred for complex sequences to be stored in two separate real arrays or for Hermitian sequences to be stored in real storage format (see Section 2.1.2) then a real storage format routine should be used; where it is preferred for complex data to be stored in complex arrays or for Hermitian sequences to be stored in complex storage format then a complex storage format routine should be used.

Note also that the complex storage format routines have a reduced parameter list: there are no INIT or TRIG parameters.

Three groups, each of three routines, are provided in real storage format and three groups of two routines are provided in complex storage format.

	Group 1	Group 2	Group 3	Group 4
Real storage format				
Real sequences	C06EAF	C06FAF	C06FPF	
Hermitian sequences	C06EBF	C06FBF	C06FQF	
General complex sequences	C06ECF	C06FCF	C06FRF	
Complex storage format				
Real/Hermitian sequences		C06PAF	C06PPF	C06PQF
General complex sequences		C06PCF	C06PRF	C06PSF

Group 1 routines each compute a single transform of length n, without requiring any extra working storage. Group 2 routines also compute a single transform of length n, but require one additional ***real*** (***complex*** for C06PCF) work-array. For some values of n — when n has unpaired prime factors — Group 1 routines are particularly slow and the Group 2 routines are much more efficient. The Group 1 and some Group 2 routines (C06FAF, C06FBF and C06FCF) impose some restrictions on the value of n, namely that no prime factor of n may exceed 19 and the total number of prime factors (including repetitions) may not exceed 20 (though the latter restriction only becomes relevant when $n > 10^6$).

Group 3 and Group 4 routines are all designed to perform several transforms in a single call, all with the same value of n. They are designed to be much faster than the Group 1 and Group 2 routines on vector-processing machines. They do however require more working storage. Even on scalar processors, they may be somewhat faster than repeated calls to Group 1 or Group 2 routines because of reduced overheads and because they pre-compute and store the required values of trigonometric functions. Group 3 and Group 4 routines differ in the way sequences are stored: Group 3 routines store sequences as rows of a two-dimensional array while Group 4 routines store sequences as columns of a two-dimensional array.

Group 3 and Group 4 routines impose no practical restrictions on the value of n; however, the fast Fourier transform algorithm ceases to be 'fast' if applied to values of n which cannot be expressed as a product of small prime factors. All the above routines are particularly efficient if the only prime factors of n are 2, 3 or 5.

If extensive use is to be made of these routines, users who are concerned about efficiency are advised to conduct their own timing tests.

To compute inverse (backward) discrete Fourier transforms the real storage format routines should be used in conjunction with the utility routines C06GBF, C06GCF and C06GQF which form the complex conjugate of a Hermitian or general sequence of complex data values. In the case of complex storage format routines, there is a **direction** parameter which determines the direction of the transform; a call to such a routine in the forward direction followed by a call in the backward direction reproduces the original data.

3.2 Half- and Quarter-wave Transforms

Eight routines are provided for computing fast Fourier transforms (FFTs) of real symmetric sequences. C06HAF and C06RAF compute multiple Fourier sine transforms, C06HBF and C06RBF compute multiple Fourier cosine transforms, C06HCF and C06RCF compute multiple quarter-wave Fourier sine transforms, and C06HDF and C06RDF compute multiple quarter-wave Fourier cosine transforms. There are two routines for each type of transform; the routines C06RAF, C06RBF, C06RCF and C06RDF have shorter parameter lists than their counterparts and are therefore simpler to use.

3.3 Application to Elliptic Partial Differential Equations

As described in Section 2.1, Fourier transforms may be used in the solution of elliptic PDEs.

C06HAF and C06RAF may be used to solve equations where the solution is specified along the boundary.

C06HBF and C06RBF may be used to solve equations where the derivative of the solution is specified along the boundary.

C06HCF and C06RCF may be used to solve equations where the solution is specified on the lower boundary, and the derivative of the solution is specified on the upper boundary.

C06HDF and C06RDF may be used to solve equations where the derivative of the solution is specified on the lower boundary, and the solution is specified on the upper boundary.

For equations with periodic boundary conditions the full-range Fourier transforms computed by C06FPF and C06FQF are appropriate.

3.4 Multi-dimensional Fourier Transforms

The following routines compute multi-dimensional discrete Fourier transforms of complex data:

	Real storage	Complex storage
2 dimensions	C06FUF	C06PUF
3 dimensions	C06FXF	C06PXF
any number of dimensions	C06FJF	C06PJF

The real storage format routines store sequences of complex data in two ***real*** arrays containing the real and imaginary parts of the sequence respectively. The complex storage format routines store the sequences in ***complex*** arrays.

Note that complex storage format routines have a reduced parameter list, having no INIT or TRIG parameters.

C06FUF (C06PUF) and C06FXF (C06PXF) should be used in preference to C06FJF (C06PJF) for two- and three-dimensional transforms, as they are easier to use and are likely to be more efficient, especially on vector processors.

3.5 Convolution and Correlation

C06EKF and C06FKF each compute either the discrete convolution or the discrete correlation of two real vectors. The distinction between these two routines is the same as that between the C06E- and C06F-routines described in Section 3.1. C06PKF computes either the discrete convolution or the discrete correlation of two complex vectors.

3.6 Inverse Laplace Transforms

Two methods are provided: Weeks' method and Crump's method. Both require the function $F(s)$ to be evaluated for complex values of s. If in doubt which method to use, try Weeks' method first; when it is suitable, it is usually much faster.

Typically the inversion of a Laplace transform becomes harder as t increases so that all numerical methods tend to have a limit on the range of t for which the inverse $f(t)$ can be computed. C06LAF is useful for small and moderate values of t.

It is often convenient or necessary to scale a problem so that α is close to 0. For this purpose it is useful to remember that the inverse of $F(s+k)$ is $\exp(-kt)f(t)$. The method used by C06LAF is not so satisfactory when $f(t)$ is close to zero, in which case a term may be added to $F(s)$, e.g.,, $k/s + F(s)$ has the inverse $k + f(t)$.

Singularities in the inverse function $f(t)$ generally cause numerical methods to perform less well. The positions of singularities can often be identified by examination of $F(s)$. If $F(s)$ contains a term of the form $\exp(-ks)/s$ then a finite discontinuity may be expected in the inverse at $t = k$. C06LAF, for example, is capable of estimating a discontinuous inverse but, as the approximation used is continuous, Gibbs' phenomena (overshoots around the discontinuity) result. If possible, such singularities of $F(s)$ should be removed before computing the inverse.

3.7 Direct Summation of Orthogonal Series

The only routine available is, C06DBF, which sums a finite Chebyshev series

$$\sum_{j=0}^{n} c_j T_j(x), \quad \sum_{j=0}^{n} c_j T_{2j}(x) \quad \text{or} \quad \sum_{j=0}^{n} c_j T_{2j+1}(x)$$

depending on the choice of a parameter.

3.8 Acceleration of Convergence

The only routine available is, C06BAF.

4 Index

complex sequence, complex storage	C06PUF
three-dimensional	
complex sequence, real storage	C06FXF
complex sequence, complex storage	C06PXF
one-dimensional, multi-variable	
complex sequence, real storage	C06FFF
complex sequence, complex storage	C06PFF
one-dimensional, multiple transforms	
complex sequence, real storage by rows	C06FRF
complex sequence, complex storage by rows	C06PRF
complex sequence, complex storage by columns	C06PSF
Hermitian sequence, real storage by rows	C06FQF
real sequence, real storage by rows	C06FPF
Hermitian/real sequence, complex storage by rows	C06PPF
Hermitian/real sequence, complex storage by columns	C06PQF
one-dimensional, single transforms	
complex sequence, space saving, real storage	C06ECF
complex sequence, time-saving, real storage	C06FCF
complex sequence, time-saving, complex storage	C06PCF
Hermitian sequence, space-saving, real storage	C06EBF
Hermitian sequence, time-saving, real storage	C06FBF
real sequence, space-saving, real storage	C06EAF
real sequence, time-saving, real storage	C06FAF
Hermitian/real sequence, time-saving, complex storage	C06PAF
half- and quarter-wave transforms	
multiple Fourier sine transforms	C06HAF
multiple Fourier sine transforms, simple use	C06RAF
multiple Fourier cosine transforms	C06HBF
multiple Fourier cosine transforms, simple use	C06RBF
multiple quarter-wave sine transforms	C06HCF
multiple quarter-wave sine transforms, simple use	C06RCF
multiple quarter-wave cosine transforms	C06HDF
multiple quarter-wave cosine transforms, simple use	C06RDF
Inverse Laplace Transform	
Crump's method	C06LAF
Weeks' method	
compute coefficients of solution	C06LBF
evaluate solution	C06LCF
Summation of Chebyshev series	C06DBF

5 Routines Withdrawn or Scheduled for Withdrawal

None since Mark 13.

6 References

[1] Brigham E O (1973) *The Fast Fourier Transform* Prentice–Hall

[2] Davies S B and Martin B (1979) Numerical inversion of the Laplace transform: A survey and comparison of methods *J. Comput. Phys.* **33** 1–32

[3] Fox L and Parker I B (1968) *Chebyshev Polynomials in Numerical Analysis* Oxford University Press

[4] Gentleman W S and Sande G (1966) Fast Fourier transforms for fun and profit *Proc. Joint Computer Conference, AFIPS* **29** 563–578

[5] Hamming R W (1962) *Numerical Methods for Scientists and Engineers* McGraw–Hill

[6] Shanks D (1955) Nonlinear transformations of divergent and slowly convergent sequences *J. Math. Phys.* **34** 1–42

[7] Swarztrauber P N (1977) The methods of cyclic reduction, Fourier analysis and the FACR algorithm for the discrete solution of Poisson's equation on a rectangle *SIAM Rev.* **19 (3)** 490–501

[8] Swarztrauber P N (1984) Fast Poisson solvers *Studies in Numerical Analysis* (ed G H Golub) Mathematical Association of America

[9] Swarztrauber P N (1986) Symmetric FFT's *Math. Comput.* **47 (175)** 323–346

[10] Wynn P (1956) On a device for computing the $e_m(S_n)$ transformation *Math. Tables Aids Comput.* **10** 91–96