NAG Library Chapter Introduction

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 subroutine.

(d) Calculating the **fast Gauss transform** approximation to the **discrete Gauss transform**.

(e) Direct summation of orthogonal series.

(f) 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 direct 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 multidimensional transforms. For example, in two dimensions we have

$$\hat{z}_{k_1k_2} = \frac{1}{\sqrt{n_1 n_2}} \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} z_{j_1j_2} \exp\left(-i\frac{2\pi j_1 k_1}{n_1}\right) \exp\left(-i\frac{2\pi j_2 k_2}{n_2}\right). \tag{3}$$

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

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

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 (deprecated) scheme, which will be referred to as the **real storage format** for Hermitian sequences, the real parts a_k for $0 \leq k \leq n/2$ are stored in normal order in the first $n/2 + 1$ locations of an array **x** of length n; the corresponding nonzero 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 subroutine, 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	$a_1 + ib_1$	$a_2 + ib_2$	...	$a_{n/2}$	...	$a_2 - ib_2$	$a_1 - ib_1$
Stored values	a_0	a_1	a_2	...	$a_{n/2}$	...	b_2	b_1

$$\begin{aligned} \mathbf{x}(k) &= a_k, && \text{for } k = 0, 1, \ldots, n/2, \text{ and} \\ \mathbf{x}(n-k) &= b_k, && \text{for } k = 1, 2, \ldots, n/2 - 1. \end{aligned}$$

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	$a_1 + ib_1$	$a_2 + ib_2$	...	$a_s + ib_s$	$a_s - ib_s$	...	$a_2 - ib_2$	$a_1 - ib_1$
Stored values	a_0	a_1	a_2	...	a_s	b_s	...	b_2	b_1

$$\begin{aligned} \mathbf{x}(k) &= a_k, && \text{for } k = 0, 1, \ldots, s, \text{ and} \\ \mathbf{x}(n-k) &= b_k, && \text{for } k = 1, 2, \ldots, s. \end{aligned}$$

The second (recommended) 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 $0 \leq k \leq n/2$, in consecutive locations of an array **x** of length $n + 2$. If **x** is declared with bounds $(0 : n + 1)$ in your calling subroutine, 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	...	$a_{n/2-1}$	$b_{n/2-1}$	$a_{n/2}$	$b_{n/2} = 0$

$$\begin{aligned} \mathbf{x}(2 \times k) &= a_k, && \text{for } k = 0, 1, \ldots, n/2, \text{ and} \\ \mathbf{x}(2 \times k + 1) &= b_k, && \text{for } k = 0, 1, \ldots, n/2. \end{aligned}$$

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

$$\begin{aligned}\mathbf{x}(2\times k)&=a_k, &&\text{for } k=0,1,\ldots,s, \text{ and}\\ \mathbf{x}(2\times k+1)&=b_k, &&\text{for } k=0,1,\ldots,s.\end{aligned}$$

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.

For real data that is two-dimensional or higher, the symmetry in the transform persists for the leading dimension only. So, using the notation of equation (3) for the complex two-dimensional discrete transform, we have that $\hat{z}_{k_1k_2}$ is the complex conjugate of $\hat{z}_{(n_1-k_1)(n_2-k_2)}$. It is more convenient for transformed data of two or more dimensions to be stored as a complex sequence of length $(n_1/2+1)\times n_2\times\cdots\times n_d$ where d is the number of dimensions. The inverse discrete Fourier transform operating on such a complex sequence (Hermitian in the leading dimension) returns a real array of full dimension $(n_1\times n_2\times\cdots\times n_d)$.

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

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

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

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

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

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

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

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

and so

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

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 (1974)) 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 (1974)) 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 (1962); more detailed information on the fast Fourier transform algorithm can be found in Gentleman and Sande (1966) and Brigham (1974).

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 discretized 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 (1977) and Swarztrauber (1984).

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

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

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 Fast Gauss Transform

Gauss transforms have applications in areas including statistics, machine learning, and numerical solution of the heat equation. The discrete Gauss transform (DGT), $G(y)$, evaluated at a set of target points $y(j)$, for $j = 1, 2, \ldots, m \in R^d$, is defined as:

$$G(y_j) = \sum_{i=1}^{n} q_i e^{-\left\|y_j - x_i\right\|_2^2 / h_i^2}, \quad j = 1, \ldots, m$$

where x_i, for $i = 1, 2, \ldots, n \in R^d$, are the Gaussian source points, q_i, for $i = 1, 2, \ldots, n \in R^+$, are the source weights and h_i, for $i = 1, 2, \ldots, n \in R^+$, are the source standard deviations (alternatively source scales or source bandwidths).

The fast Gauss transform (FGT) algorithm presented in Raykar and Duraiswami (2005) approximates the DGT by using two Taylor series and clustering of the source points.

2.4 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 (1962).

2.5 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 $\{s_n\}$, for $n = m, \ldots, m + 2l$, then, except in certain singular cases, arguments, 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 (1955) 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

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

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

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 FFT routines in this chapter are particularly efficient if the only prime factors of n are 2, 3 or 5.

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.

3.1.1 Real and Hermitian data

c06paf transforms a single sequence of real data onto (and in-place) a representation of the transformed Hermitian sequence using the **complex storage scheme** described in Section 2.1.2. **c06paf** also performs the inverse transform using the representation of Hermitian data and transforming back to a real data sequence.

Alternatively, the two-dimensional routine **c06pvf** can be used (on setting the second dimension to 1) to transform a sequence of real data onto an Hermitian sequence whose first half is stored in a separate Complex array. The second half need not be stored since these are the complex conjugate of the first half in reverse order. **c06pwf** performs the inverse operation, transforming the the Hermitian sequence (half-)stored in a Complex array onto a separate real array.

3.1.2 Complex data

c06pcf transforms a single complex sequence in-place; it also performs the inverse transform. **c06psf** transforms multiple complex sequences, each stored sequentially; it also performs the inverse transform on multiple complex sequences. This routine is designed to perform several transforms in a single call, all with the same value of n.

If extensive use is to be made of these routines and you are concerned about efficiency, you are advised to conduct your own timing tests.

3.2 Half- and Quarter-wave Transforms

Four routines are provided for computing fast Fourier transforms (FFTs) of real symmetric sequences. **c06ref** computes multiple Fourier sine transforms, **c06rff** computes multiple Fourier cosine transforms, **c06rgf** computes multiple quarter-wave Fourier sine transforms, and **c06rhf** computes multiple quarter-wave Fourier cosine transforms.

3.3 Application to Elliptic Partial Differential Equations

As described in Section 2.1.6, Fourier transforms may be used in the solution of elliptic PDEs.

c06ref may be used to solve equations where the solution is specified along the boundary.

c06rff may be used to solve equations where the derivative of the solution is specified along the boundary.

c06rgf 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.

c06rhf 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 **c06paf** are appropriate.

3.4 Multidimensional Fourier Transforms

The following routines compute multidimensional discrete Fourier transforms of real, Hermitian and complex data stored in complex arrays:

	real	**Hermitian**	**complex**
2 dimensions	c06pvf	c06pwf	c06puf
3 dimensions	c06pyf	c06pzf	c06pxf
any number of dimensions			c06pjf

The Hermitian data, either transformed from or being transformed to real data, is compacted (due to symmetry) along its first dimension when stored in Complex arrays; thus approximately half the full Hermitian data is stored.

c06puf and **c06pxf** should be used in preference to **c06pjf** for two- and three-dimensional transforms, as they are easier to use and are likely to be more efficient.

The transform of multidimensional real data is stored as a complex sequence that is Hermitian in its leading dimension. The inverse transform takes such a complex sequence and computes the real transformed sequence. Consequently, separate routines are provided for performing forward and inverse transforms.

c06pvf performs the forward two-dimensionsal transform while **c06pwf** performs the inverse of this transform.

c06pyf performs the forward three-dimensional transform while **c06pzf** performs the inverse of this transform.

The complex sequences computed by **c06pvf** and **c06pyf** contain roughly half of the Fourier coefficients; the remainder can be reconstructed by conjugation of those computed. For example, the Fourier coefficients of the two-dimensional transform $\hat{z}_{(n_1-k_1)k_2}$ are the complex conjugate of $\hat{z}_{k_1k_2}$ for $k_1 = 0, 1, \ldots, n_1/2$, and $k_2 = 0, 1, \ldots, n_2 - 1$.

3.5 Convolution and Correlation

c06fkf computes either the discrete convolution or the discrete correlation of two real vectors.

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 (**c06lbf**) and Crump's method (**c06laf**). Both require the function $F(s)$ to be evaluated for complex values of s. If in doubt which method to use, try Weeks' method (**c06lbf**) 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 Fast Gauss Transform

The only routine available is **c06saf**. If the dimensionality of the data is low or the number of source and target points is small, however, it may be more efficient to evaluate the discrete Gauss transform directly.

3.8 Direct Summation of Orthogonal Series

The only routine available is **c06dcf**, which sums a finite Chebyshev series

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

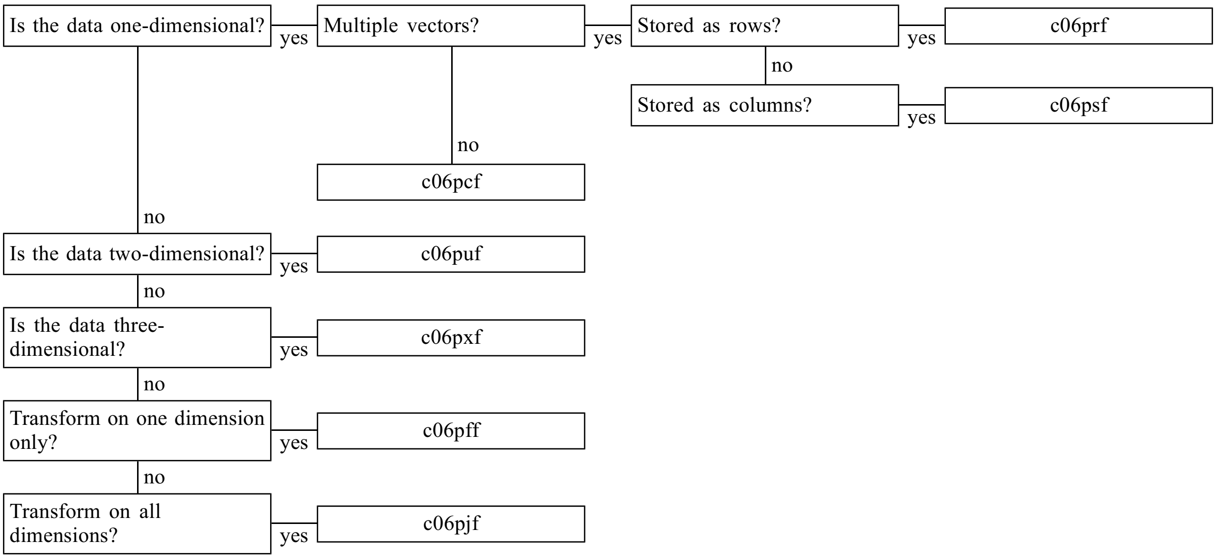
depending on the choice of argument.

3.9 Acceleration of Convergence

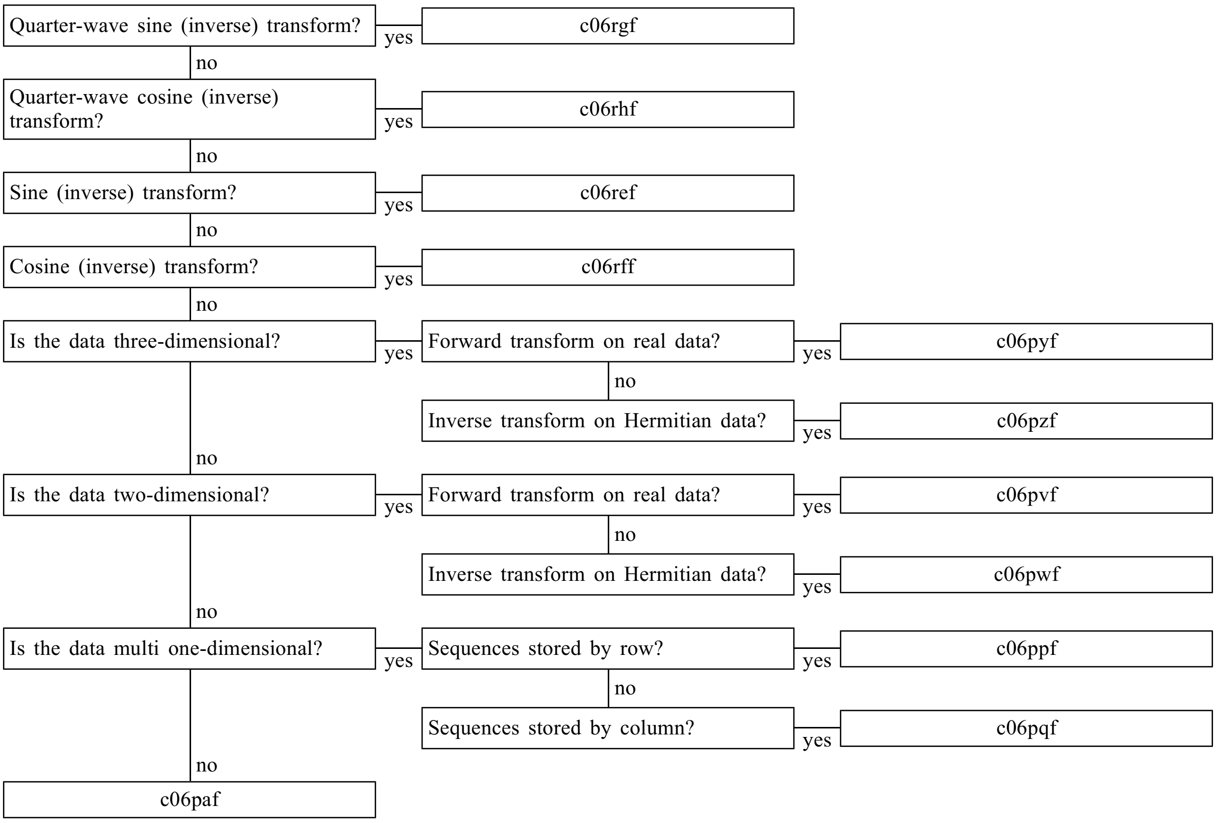
The only routine available is **c06baf**.

4 Decision Trees

Tree 1: Fourier Transform of Discrete Complex Data


Is the data one-dimensional?
yes
Multiple vectors?
yes
Stored as rows?
yes
c06prf
no
Stored as columns?
yes
c06psf
no
c06pcf
no
Is the data two-dimensional?
yes
c06puf
no
Is the data three-dimensional?
yes
c06pxf
no
Transform on one dimension only?
yes
c06pff
no
Transform on all dimensions?
yes
c06pjf

Tree 2: Fourier Transform of Real Data or Data in Complex Hermitian Form Resulting from the Transform of Real Data


Quarter-wave sine (inverse) transform?
yes
c06rgf
no
Quarter-wave cosine (inverse) transform?
yes
c06rhf
no
Sine (inverse) transform?
yes
c06ref
no
Cosine (inverse) transform?
yes
c06rff
no
Is the data three-dimensional?
yes
Forward transform on real data?
yes
c06pyf
no
Inverse transform on Hermitian data?
yes
c06pzf
no
Is the data two-dimensional?
yes
Forward transform on real data?
yes
c06pvf
no
Inverse transform on Hermitian data?
yes
c06pwf
no
Is the data multi one-dimensional?
yes
Sequences stored by row?
yes
c06ppf
no
Sequences stored by column?
yes
c06pqf
no
c06paf

5 Functionality Index

Acceleration of convergence c06baf

Convolution or Correlation,
  complex vectors c06pkf
  real vectors,
    time-saving.. c06fkf

- Discrete Fourier Transform,
  - multidimensional,
    - complex sequence,
      - complex storage ... c06pjf
      - real storage ... c06fjf
  - multiple half- and quarter-wave transforms,
    - Fourier cosine transforms,
      - simple use ... c06rbf
    - Fourier cosine transforms, simple use ... c06rff
    - Fourier sine transforms,
      - simple use ... c06raf
    - Fourier sine transforms, simple use ... c06ref
    - quarter-wave cosine transforms,
      - simple use ... c06rdf
    - quarter-wave cosine transforms, simple use ... c06rhf
    - quarter-wave sine transforms,
      - simple use ... c06rcf
    - quarter-wave sine transforms, simple use ... c06rgf
  - one-dimensional,
    - multiple transforms,
      - complex sequence,
        - complex storage by columns ... c06psf
        - complex storage by rows ... c06prf
      - Hermitian/real sequence,
        - complex storage by columns ... c06pqf
        - complex storage by rows ... c06ppf
    - multi-variable,
      - complex sequence,
        - complex storage ... c06pff
        - real storage ... c06fff
    - single transforms,
      - complex sequence,
        - time-saving,
          - complex storage ... c06pcf
          - real storage ... c06fcf
      - Hermitian/real sequence,
        - time-saving,
          - complex storage ... c06paf
      - Hermitian sequence,
        - time-saving,
          - real storage ... c06fbf
      - real sequence,
        - time-saving,
          - real storage ... c06faf
  - three-dimensional,
    - complex sequence,
      - complex storage ... c06pxf
      - real storage ... c06fxf
    - Hermitian/real sequence,
      - complex-to-real ... c06pzf
      - real-to-complex ... c06pyf
  - two-dimensional,
    - complex sequence,
      - complex storage ... c06puf
    - Hermitian/real sequence,
      - complex-to-real ... c06pwf
      - real-to-complex ... c06pvf
- Fast Gauss Transform ... c06saf

Inverse Laplace Transform,	
Crump's method	c06laf
Weeks' method,	
compute coefficients of solution	c06lbf
evaluate solution	c06lcf
Summation of Chebyshev series	c06dcf

6 Auxiliary Routines Associated with Library Routine Arguments

None.

7 Routines Withdrawn or Scheduled for Withdrawal

The following lists all those routines that have been withdrawn since Mark 19 of the Library or are scheduled for withdrawal at one of the next two marks.

Withdrawn Routine	**Mark of Withdrawal**	**Replacement Routine(s)**
c06dbf	25	c06dcf
c06eaf	26	c06paf
c06ebf	26	c06paf
c06ecf	26	c06pcf
c06ekf	26	c06fkf
c06fpf	28	c06pqf
c06fqf	28	c06pqf
c06frf	26	c06psf
c06fuf	26	c06puf
c06gbf	26	No replacement required
c06gcf	26	No replacement required
c06gqf	26	No replacement required
c06gsf	26	No replacement required
c06haf	26	c06ref
c06hbf	26	c06rff
c06hcf	26	c06rgf
c06hdf	26	c06rhf

8 References

Brigham E O (1974) *The Fast Fourier Transform* Prentice–Hall

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

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

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

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

Raykar V C and Duraiswami R (2005) Improved Fast Gauss Transform With Variable Source Scales *University of Maryland Technical Report CS-TR-4727/UMIACS-TR-2005-34*

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

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

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

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

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