

NAG Toolbox Chapter Introduction

C06 – Summation of Series

Contents

1	Scope of the Chapter	2
2	Background to the Problems	2
3	Recommendations on Choice and Use of Available Functions	7
4	Decision Trees	10
5	Functionality Index	10
6	References	12

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

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

for $k = 0, 1, \dots, n-1$. 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**:

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

for $j = 0, 1, \dots, n-1$. 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

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

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 \mathbf{x} of length n ; the corresponding nonzero imaginary parts are stored in reverse order in the remaining locations of \mathbf{x} . To clarify, if \mathbf{x} is declared with bounds $(0 : n - 1)$ in your calling function, 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 \mathbf{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

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

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

Index of \mathbf{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

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

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 \mathbf{x} of length $n + 2$. If \mathbf{x} is declared with bounds $(0 : n + 1)$ in your calling function, 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 \mathbf{x}	0	1	2	3	...	$n-2$	$n-1$	n	$n+1$
Stored values	a_0	$b_0 = 0$	a_1	b_1	...	$a_{n/2-1}$	$b_{n/2-1}$	$a_{n/2}$	$b_{n/2} = 0$

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

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

Index of \mathbf{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

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

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 $\hat{z}_{k_1 k_2}$ is the complex conjugate of $\hat{z}_{(n_1-k_1)k_2}$. It is more convenient for transformed data of two or more dimensions to be stored as a complex sequence of length $(n_1/2 + 1) \times n_2 \times \dots \times n_d$ 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 $(n_1 \times n_2 \times \dots \times n_d)$.

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

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

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

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

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

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

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

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

and so

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

for $k = 0, 1, \dots, n-1$, 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 $\text{Re}(s) > \alpha$.

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 functions 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 (1962).

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 $\{s_n\}$, for $n = m, \dots, m + 2l$, 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 k th 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

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

and to consider the sequence of values

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

where the integrals are evaluated using standard quadrature methods. In choosing the values of the a_k , it is worth bearing in mind that `nag_sum_accelerate` (c06ba) 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 Functions

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 functions 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 function 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 function.

3.1.1 Real and Hermitian data

`nag_sum_fft_realherm_1d` (c06pa) 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. `nag_sum_fft_realherm_1d` (c06pa) also performs the inverse transform using the representation of Hermitian data and transforming back to a real data sequence.

Alternatively, the two-dimensional function `nag_sum_fft_real_2d` (c06pv) 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. `nag_sum_fft_hermitian_2d` (c06pw) 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

`nag_sum_fft_complex_1d` (c06pc) transforms a single complex sequence in-place; it also performs the inverse transform. `nag_sum_fft_complex_1d_multi_col` (c06ps) transforms multiple complex sequences, each stored sequentially; it also performs the inverse transform on multiple complex sequences. This function 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 functions and you are concerned about efficiency, you are advised to conduct your own timing tests.

3.2 Half- and Quarter-wave Transforms

Four functions are provided for computing fast Fourier transforms (FFTs) of real symmetric sequences. `nag_sum_fft_sine` (c06re) computes multiple Fourier sine transforms, `nag_sum_fft_cosine` (c06rf) computes multiple Fourier cosine transforms, `nag_sum_fft_qtrsine` (c06rg) computes multiple quarter-wave Fourier sine transforms, and `nag_sum_fft_qtrcosine` (c06rh) 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.

`nag_sum_fft_sine` (c06re) may be used to solve equations where the solution is specified along the boundary.

`nag_sum_fft_cosine` (c06rf) may be used to solve equations where the derivative of the solution is specified along the boundary.

`nag_sum_fft_qtrsine` (c06rg) 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.

`nag_sum_fft_qtrcosine` (c06rh) 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 `nag_sum_fft_realherm_1d` (c06pa) are appropriate.

3.4 Multidimensional Fourier Transforms

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

	double	Hermitian	complex
2 dimensions	c06pv	c06pw	c06pu
3 dimensions	c06py	c06pz	c06px
any number of dimensions			c06pj

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.

`nag_sum_fft_complex_2d` (c06pu) and `nag_sum_fft_complex_3d` (c06px) should be used in preference to `nag_sum_fft_complex_multid` (c06pj) 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 functions are provided for performing forward and inverse transforms.

`nag_sum_fft_real_2d` (c06pv) performs the forward two-dimensional transform while `nag_sum_fft_hermitian_2d` (c06pw) performs the inverse of this transform.

`nag_sum_fft_real_3d` (c06py) performs the forward three-dimensional transform while `nag_sum_fft_hermitian_3d` (c06pz) performs the inverse of this transform.

The complex sequences computed by `nag_sum_fft_real_2d` (c06pv) and `nag_sum_fft_real_3d` (c06py) 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 $k_1 = 0, 1, \dots, n_1/2$, and $k_2 = 0, 1, \dots, n_2 - 1$.

3.5 Convolution and Correlation

`nag_sum_convcorr_real` (c06fk) computes either the discrete convolution or the discrete correlation of two real vectors.

`nag_sum_convcorr_complex` (c06pk) computes either the discrete convolution or the discrete correlation of two complex vectors.

3.6 Inverse Laplace Transforms

Two methods are provided: Weeks' method (`nag_sum_invlaplace_weeks` (c06lb)) and Crump's method (`nag_sum_invlaplace_crump` (c06la)). Both require the function $F(s)$ to be evaluated for complex values of s . If in doubt which method to use, try Weeks' method (`nag_sum_invlaplace_weeks` (c06lb)) 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. `nag_sum_invlaplace_crump` (c06la) 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 `nag_sum_invlaplace_crump` (c06la) 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 $t = k$. `nag_sum_invlaplace_crump` (c06la), 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 function available is `nag_sum_chebyshev` (c06dc), which sums a finite Chebyshev series

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

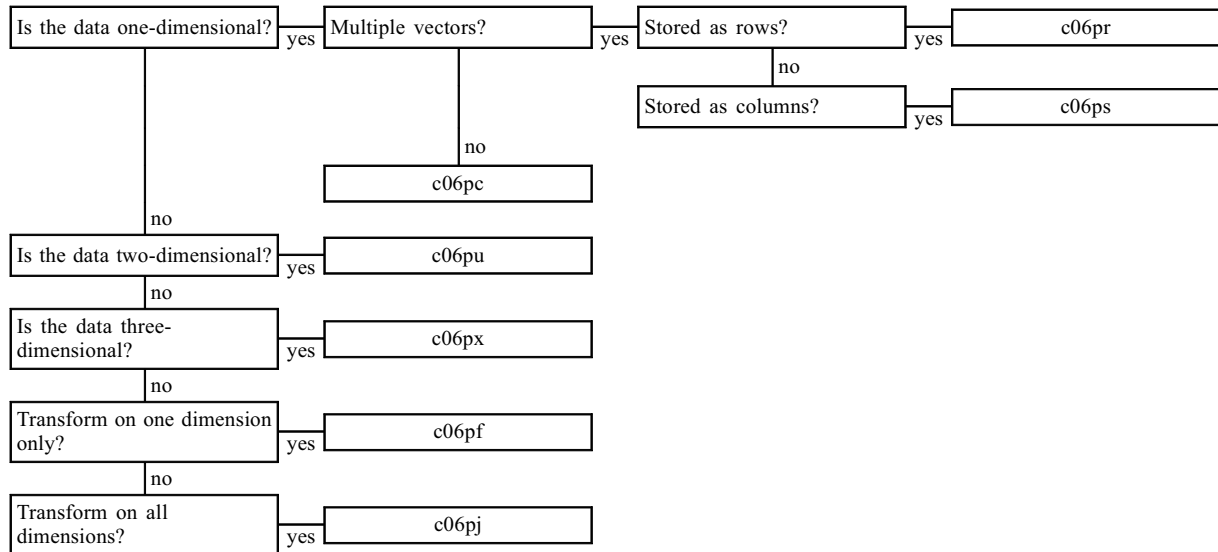
depending on the choice of a argument.

3.8 Acceleration of Convergence

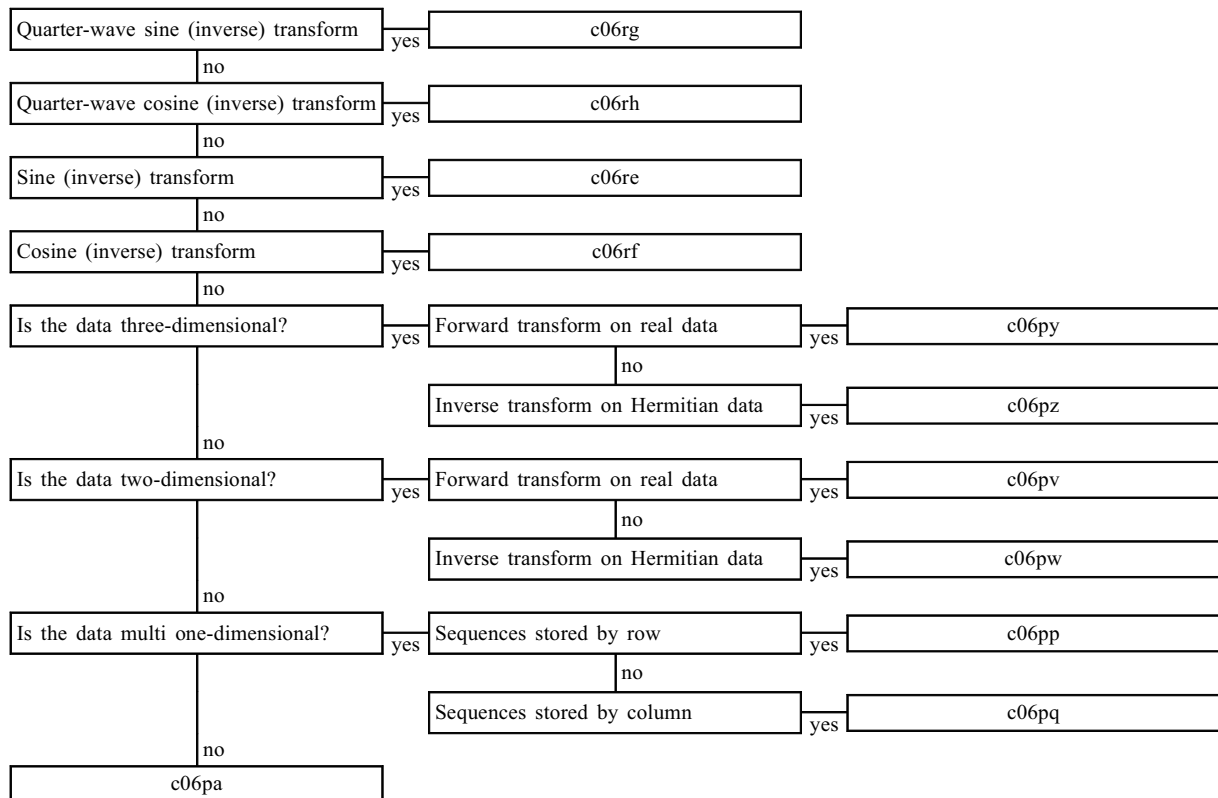
The only function available is `nag_sum_accelerate` (c06ba).

4 Decision Trees

Tree 1: Fourier Transform of Discrete Complex Data



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



5 Functionality Index

Acceleration of convergence	c06ba
Convolution or Correlation, complex vectors	c06pk
real vectors, time-saving.....	c06fk

Discrete Fourier Transform,	
multidimensional,	
complex sequence,	
complex storage	c06pj
real storage	c06fj
multiple half- and quarter-wave transforms,	
Fourier cosine transforms,	
simple use	c06rb
Fourier cosine transforms, simple use.....	c06rf
Fourier sine transforms,	
simple use	c06ra
Fourier sine transforms, simple use	c06re
quarter-wave cosine transforms,	
simple use	c06rd
quarter-wave cosine transforms, simple use.....	c06rh
quarter-wave sine transforms,	
simple use	c06rc
quarter-wave sine transforms, simple use	c06rg
one-dimensional,	
multiple transforms,	
complex sequence,	
complex storage by columns.....	c06ps
complex storage by rows	c06pr
Hermitian/real sequence,	
complex storage by columns.....	c06pq
complex storage by rows	c06pp
Hermitian sequence,	
real storage by rows	c06fq
real sequence,	
real storage by rows	c06fp
multi-variable,	
complex sequence,	
complex storage	c06pf
real storage	c06ff
single transforms,	
complex sequence,	
time-saving,	
complex storage	c06pc
real storage	c06fc
Hermitian/real sequence,	
time-saving,	
complex storage	c06pa
Hermitian sequence,	
time-saving,	
real storage	c06fb
real sequence,	
time-saving,	
real storage	c06fa
three-dimensional,	
complex sequence,	
complex storage	c06px
real storage	c06fx
Hermitian/real sequence,	
complex-to-real	c06pz
real-to-complex	c06py
two-dimensional,	
complex sequence,	
complex storage	c06pu

Hermitian/real sequence,	
complex-to-real	c06pw
real-to-complex	c06pv
Inverse Laplace Transform,	
Crump's method	c06la
Weeks' method,	
compute coefficients of solution	c06lb
evaluate solution	c06lc
Summation of Chebyshev series	c06dc

6 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
- 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
-