NAG C Library Chapter Introduction

c06 – Fourier Transforms

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 data values using discrete Fourier transforms.

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 $j = 0, 1, \ldots, n-1$. 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

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

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 one scheme being used in this chapter. In this 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 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 **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

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

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

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

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

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

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

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

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 (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 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 (1977) and Swarztrauber (1984).

3 Recommendations on Choice and Use of Available Functions

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.

Two groups, each of three functions, are provided in real storage format.

	Group 1	Group 2
Real sequences	nag_fft_real (c06eac)	nag_fft_multiple_real (c06fpc)
Hermitian sequences	nag_fft_hermitian (c06ebc)	nag_fft_multiple_hermitian (c06fqc)
General complex sequences	nag_fft_complex (c06ecc)	nag_fft_multiple_complex (c06frc)

Group 1 functions each compute a single transform of length n, without requiring any extra working storage. The Group 1 functions 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 2 functions 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 functions 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 functions because of reduced overheads and because they pre-compute and store the required values of trigonometric functions. Group 2 functions 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 functions are particularly efficient if the only prime factors of n are 2, 3 or 5.

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

To compute inverse (backward) discrete Fourier transforms the functions should be used in conjunction with the utility functions nag_conjugate_hermitian (c06gbc), nag_conjugate_complex (c06gcc) and nag_multiple_conjugate_hermitian (c06gqc) which form the complex conjugate of a Hermitian or general sequence of complex data values.

3.2 Half- and Quarter-wave Transforms

Four functions are provided for computing fast Fourier transforms (FFTs) of real symmetric sequences. nag_fft_multiple_sine (c06hac) computes multiple Fourier sine transforms, nag_fft_multiple_cosine (c06hbc) computes multiple Fourier cosine transforms, nag_fft_multiple_qtr_sine (c06hcc) computes multiple quarter-wave Fourier sine transforms, and nag_fft_multiple_qtr_cosine (c06hdc) computes multiple quarter-wave Fourier cosine transforms.

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.

nag_fft_multiple_sine (c06hac) may be used to solve equations where the solution is specified along the boundary.

nag_fft_multiple_cosine (c06hbc) may be used to solve equations where the derivative of the solution is specified along the boundary.

nag_fft_multiple_qtr_sine (c06hcc) 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_fft_multiple_qtr_cosine (c06hdc) 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_fft_multiple_real (c06fpc) and nag_fft_multiple_hermitian (c06fqc) are appropriate.

3.4 Multi-dimensional Fourier Transforms

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

	Real storage	Complex storage
2 dimensions	nag_fft_2d_complex (c06fuc)	
3 dimensions		nag_fft_3d (c06pxc)
any number of dimensions		nag_fft_multid_full (c06pjc)

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

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

nag_fft_2d_complex (c06fuc) and nag_fft_3d (c06pxc) should be used in preference to nag_fft_multid_full (c06pjc) 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

nag_convolution_real (c06ekc) computes either the discrete convolution or the discrete correlation of two real vectors.

4 Index

Compute trigonometric functions ..... nag_fft_init_trig (c06gzc)
Convolution or Correlation
  real vectors,
    space-saving ..... nag_convolution_real (c06ekc)
Discrete Fourier Transform
  half- and quarter-wave transforms
    multiple Fourier cosine transforms ..... nag_fft_multiple_cosine (c06hbc)
    multiple Fourier sine transforms ..... nag_fft_multiple_sine (c06hac)
    multiple quarter-wave cosine transforms ..... nag_fft_multiple_qtr_cosine (c06hdc)
    multiple quarter-wave sine transforms ..... nag_fft_multiple_qtr_sine (c06hcc)
  multi-dimensional
    complex sequence,
      complex storage ..... nag_fft_multid_full (c06pjc)
  one-dimensional,
    multi-variable
      complex sequence,
        complex storage ..... nag_fft_multid_single (c06pfc)
    multiple transforms
      complex sequence,
        real storage by rows ..... nag_fft_multiple_complex (c06frc)
      Hermitian sequence,
        real storage by rows ..... nag_fft_multiple_hermitian (c06fqc)
      real sequence,
        real storage by rows ..... nag_fft_multiple_real (c06fpc)
    single transforms
      complex sequence,
        space saving,
          real storage ..... nag_fft_complex (c06ecc)
      Hermitian sequence,
        space-saving,
          real storage ..... nag_fft_hermitian (c06ebc)
      real sequence,
        space-saving,
          real storage ..... nag_fft_real (c06eac)
  three-dimensional
    complex sequence,
      complex storage ..... nag_fft_3d (c06pxc)
  two-dimensional
    complex sequence,
      real storage ..... nag_fft_2d_complex (c06fuc)

5 Functions Withdrawn or Scheduled for Withdrawal

None.

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