NAG Library Routine Document

C06PWF

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details.

1 Purpose

C06PWF computes the two-dimensional inverse discrete Fourier transform of a bivariate Hermitian sequence of complex data values.

2 Specification

```
SUBROUTINE C06PWF (M, N, Y, X, IFAIL)

INTEGER                 M, N, IFAIL
REAL (KIND=nag_wp)      X(M*N)
COMPLEX (KIND=nag_wp) Y((M/2+1)*N)
```

3 Description

C06PWF computes the two-dimensional inverse discrete Fourier transform of a bivariate Hermitian sequence of complex data values $z_{j_1 j_2}$, for $j_1 = 0, 1, \ldots, m-1$ and $j_2 = 0, 1, \ldots, n-1$.

The discrete Fourier transform is here defined by

$$\hat{x}_{k_1 k_2} = \frac{1}{\sqrt{mn}} \sum_{j_1=0}^{m-1} \sum_{j_2=0}^{n-1} z_{j_1 j_2} \times \exp\left(2\pi i \left(\frac{j_1 k_1}{m} + \frac{j_2 k_2}{n}\right)\right),$$

where $k_1 = 0, 1, \ldots, m-1$ and $k_2 = 0, 1, \ldots, n-1$. (Note the scale factor of $\frac{1}{\sqrt{mn}}$ in this definition.)

Because the input data satisfies conjugate symmetry (i.e., $z_{k_1 k_2}$ is the complex conjugate of $z_{(m-k_1)k_2}$, the transformed values $\hat{x}_{k_1 k_2}$ are real.

A call of C06PVF followed by a call of C06PWF will restore the original data.

This routine calls C06PQF and C06PRF to perform multiple one-dimensional discrete Fourier transforms by the fast Fourier transform (FFT) algorithm in Brigham (1974) and Temperton (1983).

4 References

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

Temperton C (1983) Fast mixed-radix real Fourier transforms *J. Comput. Phys.* **52** 340–350

5 Parameters

1: M – INTEGER *Input*

On entry: m, the first dimension of the transform.

Constraint: M ≥ 1.

2: N – INTEGER *Input*

On entry: n, the second dimension of the transform.

Constraint: N ≥ 1.

3: Y((M/2 + 1) × N) – COMPLEX (KIND=nag_wp) array *Input*

On entry: the Hermitian sequence of complex input dataset z, where $z_{j_1 j_2}$ is stored in $\mathrm{Y}(j_2 \times (m/2+1) + j_1 + 1)$, for $j_1 = 0, 1, \ldots, m/2$ and $j_2 = 0, 1, \ldots, n-1$. That is, if Y is regarded as a two-dimensional array of dimension $(0 : \mathrm{M}/2, 0 : \mathrm{N}-1)$, then $\mathrm{Y}(j_1, j_2)$ must contain $z_{j_1 j_2}$.

4: X(M × N) – REAL (KIND=nag_wp) array *Output*

On exit: the real output dataset $\hat{x}$, where $\hat{x}_{k_1 k_2}$ is stored in $\mathrm{X}(k_2 \times m + k_1 + 1)$, for $k_1 = 0, 1, \ldots, m-1$ and $k_2 = 0, 1, \ldots, n-1$. That is, if X is regarded as a two-dimensional array of dimension $(0 : \mathrm{M}-1, 0 : \mathrm{N}-1)$, then $\mathrm{X}(k_1, k_2)$ contains $\hat{x}_{k_1 k_2}$.

5: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. If you are unfamiliar with this parameter you should refer to Section 3.3 in the Essential Introduction for details.

For environments where it might be inappropriate to halt program execution when an error is detected, the value −1 or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, if you are not familiar with this parameter, the recommended value is 0. **When the value −1 or 1 is used it is essential to test the value of IFAIL on exit.**

On exit: IFAIL = 0 unless the routine detects an error or a warning has been flagged (see Section 6).

6 Error Indicators and Warnings

If on entry IFAIL = 0 or −1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors or warnings detected by the routine:

IFAIL = 1

On entry, M = ⟨*value*⟩.
Constraint: M ≥ 1.

IFAIL = 2

On entry, N = ⟨*value*⟩.
Constraint: N ≥ 1.

IFAIL = 3

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

IFAIL = −999

Dynamic memory allocation failed.

7 Accuracy

Some indication of accuracy can be obtained by performing a forward transform using C06PVF and a backward transform using C06PWF, and comparing the results with the original sequence (in exact arithmetic they would be identical).

8 Further Comments

The time taken by C06PWF is approximately proportional to $mn\log(mn)$, but also depends on the factors of m and n. C06PWF is fastest if the only prime factors of m and n are 2, 3 and 5, and is particularly slow if m or n is a large prime, or has large prime factors.

Workspace is internally allocated by C06PWF. The total size of these arrays is approximately proportional to mn.

9 Example

See Section 9 in C06PVF.