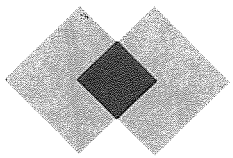

AMT

DAP Series

Digital Signal Processing Library

(man013.03)

Contents

CONTENTS

Chapter 1

Introduction

1.1 General

DSPLIB is the **D**igital **S**ignal **P**rocessing **LIB**rary. Release 2 of DSPLIB contains over 200 routines, that variously can perform:

- Fast Fourier transforms (FFT)
- Windowing
- Signal generation
- Signal format conversion

In general, for each function provided by DSPLIB there are 14 subroutines, one for each FORTRAN-PLUS data precision: INTEGER*1 to INTEGER*8 and REAL*3 to REAL*8. In the detailed specifications that follow, for each function the INTEGER*1 to INTEGER*8 variants are grouped into the generic INTEGER*n, and the REAL*3 to REAL*8 variants are grouped into the generic REAL*n.

The exception to this generality is the group of high performance FFT subroutines, which can handle input data precisions of 8 to 11 bits, and in some cases up to 16 bits.

The DAP is based on an ES by ES array of single bit processing elements, where ES is the edge size of the DAP: 32 for DAP 500 machines, 64 for DAP 600 machines, and so on. All of the subroutines in Release 2 operate on N-point arrays of matrices, treated as ES^2 1-dimensional N-point data sets. Each complete data set is manipulated by only 1 processing element and stored in the memory directly associated with that processing element. The subroutines operate on all ES^2 data sets in parallel.

Points to note:

- The number of processing elements in the DAP (the square of ES, the 'edge size', in DAP terminology) does not affect the running of any subroutine in the library. However, the routines can be tuned at software configuration time for particular sizes of DAP array memory,

to optimise subroutine performance; you should make sure that the DSPLIB supplied to you has been configured by AMT for your DAP's memory size. If you try to run the library on a larger array memory you should have no problems (other than reduced performance); if you try to run it on a smaller array memory, some of the subroutines will not work.

- Various system routines are used internally by the DSPLIB; the names of these routines start with **SPL_**, and are listed under **Auxiliary routines** in the reference section of this manual. You should make sure that none of the routines you write has the same name as any of these system routines; if any do you may get unexpected results when you use DSPLIB.

1.2 Fast Fourier transforms

Two chapters in the manual are devoted to fast Fourier transforms. Chapter 3 describes 10 groups of subroutines, that take input data from arrays X and Y, calculate the transform, and put result back in X and Y. One of the user-defined parameters to those FFTs is N, the required length of the transform – both the number of pairs of input data to be operated on to form the transform, and the number of terms in the Fourier series that are to be evaluated to form the required transform. There is a maximum permitted size to the value of N, given by the formula:

$$N_{max} = \xi\left(\frac{256\ Mem}{bytes}\right)$$

where:

N_{max} is the maximum length of the transform that can be specified in the routine

Mem is the size of the DAP array store, in Mbytes

bytes is the number of bytes in each data element being transformed

$\xi(x) = z$, where $z = 2^y$, $2^y < x < 2^{y+1}$, and y is an integer.

Chapter 4 describes 2 groups of high performance FFTs, that give an improvement in performance of the order of 3 or 4 over those described in chapter 3. They also allow you more freedom in the way you input the data to be transformed, at the expense of being more difficult to use.

The discrete Fourier transforms (DFTs), and the inverse discrete Fourier transforms, on which fast Fourier transforms (FFTs) are based, are calculated from the series:

$$\text{the } k^{th} \text{ component of the DFT: } X(k) = \sum_{n=0}^{N-1} x(n) \exp\left(-j\frac{2\pi nk}{N}\right) \quad \text{for } k = 0, 1, ..., N-1$$

and

the n^{th} component of the inverse DFT: $x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)\exp\left(j\frac{2\pi nk}{N}\right)$

for $n = 0, 1, ..., N-1$

where N is the user-specified length of the required transform.

1.3 Signal format conversion

In the section of the library concerned with signal format conversion, routines exist to convert signal format between:

- In-phase and quadrature components, and magnitude and phase angle components
- In-phase and quadrature components, and power and phase angle components

The algorithms used for the conversion are:

$$\text{Magnitude} = \sqrt{x^2 + y^2}$$

$$\text{Power} = 20\log_{10}\sqrt{x^2 + y^2}$$

$$\text{Phase angle} = \arctan\left(\frac{y}{x}\right)$$

where:

x is the in-phase component of the signal

y is the quadrature component of the signal

DSPLIB's subroutines are described in the following chapters. Chapter 2 contains a quick reference list of the routines; chapters 3 to 7 describe each subroutine in detail.

1.4 Using the library on different ranges of DAP machines

In general the signal processing software has been written for use on any of the models in the whole range of DAP machines, although some limitations to full portability do exist, mainly because of the characteristics of the various host operating systems. These limitations are discussed in the next sections.

1.5 Compilation and Linking Procedure

1.5.1 In a Sun UNIX environment

You can link the digital signal processing library into a program by using the **-l** option to either **dapa** or **dapf** (see *DAP Series: Program Development under UNIX* (man003) for more details).

For example, a FORTRAN-PLUS source program in a file **sigproc.df** can be compiled and linked with the AMT-supplied DSPLIB routines, and the object code put into a DOF file **sigproc** by executing the command:

```
dapf -o sigproc sigproc.df -l dsplib
```

If you want to port FORTRAN-PLUS or APAL code containing calls to signal processing subroutines to a DAP of different edge size, also operating under UNIX, then you have to recompile and relink the code.

The UNIX environment variable that 'knows' the size of the target DAP is DAPSIZE, and DAPSIZE defaults to 32 when you run **dapf** or **dapa**. Hence, if you want to compile and link code to run on a DAP 500, you do not have to set DAPSIZE explicitly. If you want to compile and link code to run on a DAP 600, then before you use **dapf** or **dapa** you should enter:

```
setenv  DAPSIZE  64
```

and so on for other sizes of DAP.

1.5.2 In a VAX/VMS environment

You can link the digital signal processing library into a program by including it in the list of input files to the DLINK command (see *DAP Series: Program Development Under VAX/VMS* (man004) for further details). In release 3.0V of the basic software, two versions of the graphics library are supplied, DSPLIB5 for DAP 500, and DSPLIB6 for DAP 600; when you link the signal processing library into your program you need to specify the appropriate version of DSPLIB.

For example, to compile and link the DAP program in the file **SIGPROC.DFP** to run on a DAP 600, you can use the following commands:

```
$ DFORTRAN/DAPSIZE=64 SIGPROC
$ DLINK/DAPSIZE=64 SIGPROC,SYS$LIBRARY:DSPLIB6/LIBRARY
```

To compile and link code in file **SIGCALCS.DFP** to run on a DAP 500 the above commands would be:

```
$ DFORTRAN/DAPSIZE=32 SIGCALCS
$ DLINK/DAPSIZE=32 SIGCALCS,SYS$LIBRARY:DSPLIB5/LIBRARY
```

As an alternative to specifying that the signal processing routines which the code in SIGPROC references are to be found in library SYS$LIBRARY:DSPLIB6.DLB, or the routines for SIGCALCS are in library SYS$LIBRARY:DSPLIB5.DLB you can define the logical name DAP*n*_LIBRARY by using the command:

$ DEFINE DAP*n*_LIBRARY SYS$LIBRARY:DSPLIB*n*

where *n* is 5 (for DAP 500) or 6 (for DAP 600). This will cause DLINK to search DSPLIB*n* automatically for unsatisfied external references. If you are going to use DSPLIB*n* frequently, you could insert the above DEFINE into your LOGIN.COM file. If there are several DAP users on the system, linked to a DAP 600 say, the system manager could include the command:

```
$ DEFINE/SYSTEM DAP6_LIBRARY SYS$LIBRARY:DSPLIB6
```

into the site system start-up command file which would give all users automatic access to the library.

Similarly, the command:

```
$ DEFINE/SYSTEM DAP5_LIBRARY SYS$LIBRARY:DSPLIB5
```

would achieve the same thing for a DAP 500 system.

On a system that has available both DAP 500 and DAP 600, then both DAP5_LIBRARY and DAP6_LIBRARY can be defined, and users will pick up the appropriate version of DSPLIB when they specify DAPSIZE in their DFORTRAN and DLINK commands.

Chapter 2

DSPLIB quick reference catalogue

Listed below are the groups of subroutines in DSPLIB, and the subroutines in each group. For details of each group's use, consult the relevant section in the chapters that follow.

You may find this chapter helpful in the initial selection of suitable routines for the job in hand.

Chapter 3 – Fast Fourier transforms

1 **Complex forward FFT** starts on page 12

CF_FFT1_I*n* calculates a forward fast Fourier transform (FFT) of complex input data held in an array of numbers of type INTEGER*n, where n can have any value from 1 to 8.

CF_FFT1_R*n* calculates a forward fast Fourier transform (FFT) of complex input data held in an array of numbers of type REAL*n, where n can have any value from 3 to 8.

2 **Complex inverse FFT** starts on page 16

CI_FFT1_I*n* calculates an inverse fast Fourier transform (FFT) of complex input data held in an array of numbers of type INTEGER*n, where n can have any value from 1 to 8.

CI_FFT1_R*n* calculates an inverse fast Fourier transform (FFT) of complex input data held in an array of numbers of type REAL*n, where n can have any value from 3 to 8.

3 **Real forward FFT** starts on page 20

RF_FFT1_I*n* calculates a forward fast Fourier transform (FFT) of real input data held in an array of numbers of type INTEGER*n, where n can have any value from 1 to 8.

RF_FFT1_R*n* calculates a forward fast Fourier transform (FFT) of real input data held in an array of numbers of type REAL*n, where n can have any value from 3 to 8.

4 **Real inverse FFT** starts on page 24

RI_FFT1_I*n* calculates an inverse fast Fourier transform (FFT) of complex conjugate symetric input data held in an array of numbers of type INTEGER*n, where n can have any value from 1 to 8, and produces a real output.

RI_FFT1_R*n* calculates an inverse fast Fourier transform (FFT) of complex conjugate symetric input data held in an array of numbers of type REAL*n, where n can have any value from 3 to 8, and produces a real output.

Chapter 3

Fast Fourier transforms

Contents:

3.1 Complex forward FFT

3.1.1 CF_FFT1_I*n* **release 2**

1 Purpose

CF_FFT1_I*n* calculates a forward fast Fourier transform (FFT) of complex input data held in an array of numbers of type INTEGER*n, where n can have any value from 1 to 8.

2 Specification

SUBROUTINE CF_FFT1_I*n* (X, Y, N, M, IFAIL)
INTEGER*n X(,,N), Y(,,N)
INTEGER*4 N, M, IFAIL

3 Description

CF_FFT1_I*n* simultaneously evaluates ES^2 N-point complex FFTs, where ESE is the edge size of the target DAP. X (real part) and Y (imaginary part) are both N-deep matrix arrays with data arranged sequentially 'down the PEs'. An 'in-place' FFT algorithm is used, so the output data will be stored in X and Y, and the user's input data over-written.

4 References

[1] Brigham E O

The Fast Fourier Transform: Prentice-Hall, 1974

5 Arguments

X – INTEGER*n matrix variable N-deep

Contains the real input data on entry to the subroutine, and the real spectrum on exit.

Y – INTEGER*n matrix variable N-deep

Contains the imaginary input data on entry to the subroutine, and the imaginary spectrum on exit.

N – INTEGER*4 scalar variable

Specifies the length of the transform. N must be an integer power of 2, that is, $N = 2^M$, where M is a positive integer.

M – INTEGER*4 scalar variable

M is a positive integer such that $N = 2^M$, where N is the user-specified length of the transform.

IFAIL – INTEGER*4 scalar variable

Unless the routine detects an error IFAIL will be set to zero on exit from the subroutine.

6 Error indicators

0 The subroutine function has worked correctly

1 N is not correctly specified

2 M is not correctly specified

7 Auxiliary routines

IN_FFT1_I*n* – Initialises the FFTs coefficient arrays for a given length N (the M^{th} power of 2).

System Routines

SPL_RAD2-I*n*
SPL_RAD4-I*n*
SPL_IRAD8-I*n*
SPL_LRAD8-I*n*

SPL_MR_REV_82
SPL_MR_REV_84
SPL_MR_REV_88

8 Accuracy

Details to be supplied

9 Further comments

COMMON blocks used – SPL_FFT_I*n*, SPL_MACH, SPL_TMP0_I*n* to SPL_TMP4_I*n*, SPL_TRIG

10 Keywords

FFT, fast Fourier transforms, spectral analysis

3.1.2 CF_FFT1_R*n* **release 2**

1 Purpose

CF_FFT1_R*n* calculates a forward fast Fourier transform (FFT) of complex input data held in an array of numbers of type REAL*n, where n can have any value from 3 to 8.

2 Specification

SUBROUTINE CF_FFT1_R*n* (X, Y, N, M, IFAIL)
REAL*n X(,,N), Y(,,N)
INTEGER*4 N, M, IFAIL

3 Description

CF_FFT1_R*n* simultaneously evaluates ES^2 N-point complex FFTs, where ES is the edge size of the target DAP. X (real part) and Y (imaginary part) are both N-deep matrix arrays with data arranged sequentially 'down the PEs'. An 'in-place' FFT algorithm is used, so the output spectral data will be stored in X and Y, and the user's input data over-written.

4 References

[1] Brigham E O

The Fast Fourier Transform: Prentice-Hall, 1974.

5 Arguments

X – REAL*n matrix variable N-deep

Contains the real input data on entry to the subroutine, and the real spectrum on exit.

Y – REAL*n matrix variable N-deep

Contains the imaginary input data on entry to the subroutine, and the imaginary spectrum on exit.

N – INTEGER*4 scalar variable

Specifies the length of the transform. N must be an integer power of 2, that is, $N = 2^M$, where M is a positive integer.

M – INTEGER*4 scalar variable

M is a positive integer such that $N = 2^M$, and N is the user-specified length of the transform.

IFAIL – INTEGER*4 scalar variable

Unless the routine detects an error IFAIL will be set to zero on exit from the subroutine.

6 Error indicators

0 The subroutine function has worked correctly

1 N is not correctly specified

2 M is not correctly specified

7 Auxiliary routines

IN_FFT1_R*n* – Initialises the FFT's coefficient arrays for a given length N (the M^{th} power of 2).

System Routines

SPL_RAD2-R*n*
SPL_RAD4-R*n*
SPL_IRAD8-R*n*
SPL_LRAD8-R*n*

SPL_MR_REV_82
SPL_MR_REV_84

8 Accuracy

Details to be supplied

9 Further comments

COMMON blocks used – SPL_FFT_R*n*, SPL_MACH, SPL_TMP0_I*n* to SPL_TMP4_I*n*, SPL_TRIG

10 Keywords

FFT, fast Fourier transforms, spectral analysis

3.2 Complex inverse FFT

3.2.1 CI_FFT1_I*n* release 2

1 Purpose

CI_FFT1_I*n* calculates an inverse fast Fourier transform (FFT) of complex spectral input data held in an array of numbers of type INTEGER*n, where n can have any value from 1 to 8.

2 Specification

SUBROUTINE CI_FFT1_I*n* (X, Y, N, M, IFAIL)
INTEGER*n X(,,N), Y(,,N)
INTEGER*4 N, M, IFAIL

3 Description

CI_FFT1_I*n* simultaneously evaluates ES^2 N-point complex inverse FFTs, where ES is the edge size od the target DAP. X (real part) and Y (imaginary part) are both N-deep matrix arrays with data arranged sequentially 'down the PEs'. An 'in-place' inverse FFT algorithm is used, so the output data will be stored in X and Y, and the user's input data over-written.

4 References

[1] Brigham E O

The Fast Fourier Transform: Prentice-Hall, 1974

5 Arguments

X – INTEGER*n matrix variable N-deep

Contains the real spectrum on entry to the subroutine, and real data on exit.

Y – INTEGER*n matrix variable N-deep

Contains the imaginary spectrum on entry to the subroutine, and imaginary data on exit.

N – INTEGER*4 scalar variable

Specifies the length of the transform. N must be an integer power of 2, that is, $N = 2^M$, where M is a positive integer.

M – INTEGER*4 scalar variable

M is a positive integer such that $N = 2^M$, where N is the user-specified length of the transform.

IFAIL – INTEGER*4 scalar variable

Unless the routine detects an error IFAIL will be set to zero on exit from the subroutine.

6 Error indicators

0 The subroutine function has worked correctly

1 N is not correctly specified

2 M is not correctly specified

7 Auxiliary routines

IN_FFT1_I*n* – Initialises the FFTs coefficient arrays for a given length N (the M^{th} power of 2).

System Routines

SPL_RADI2-I*n*
SPL_RADI4-I*n*
SPL_IRADI8-I*n*
SPL_LRADI8-I*n*

SPL_MR_REV_82
SPL_MR_REV_84
SPL_MR_REV_88

8 Accuracy

Details to be supplied

9 Further comments

COMMON blocks used – SPL_FFT_I*n*, SPL_MACH, SPL_TMP0_I*n* to SPL_TMP4_I*n*, SPL_TRIG

10 Keywords

FFT, fast Fourier transforms, spectral analysis

3.2.2 CI_FFT1_R*n* **release 2**

1 Purpose

CI_FFT1_R*n* calculates an inverse fast Fourier transform (FFT) of complex spectral input data held in an array of numbers of type REAL*n, where n can have any value from 3 to 8.

2 Specification

SUBROUTINE CI_FFT1_R*n* (X, Y, N, M, IFAIL)
REAL*n X(,,N), Y(,,N)
INTEGER*4 N, M, IFAIL

3 Description

CI_FFT1_R*n* simultaneously evaluates ES^2 N-point complex inverse FFTs, where ES is the edge size od the target DAP. X (real part) and Y (imaginary part) are both N-deep matrix arrays with data arranged sequentially 'down the PEs'. An 'in-place' inverse FFT algorithm is used, so the output data will be stored in X and Y, and the user's input data over-written.

4 References

[1] Brigham E O

The Fast Fourier Transform: Prentice-Hall, 1974.

5 Arguments

X – REAL*n matrix variable N-deep

Contains the real spectrum on entry to the subroutine, and real data on exit.

Y – REAL*n matrix variable N-deep

Contains the imaginary spectrum on entry to the subroutine, and imaginary data on exit.

N – INTEGER*4 scalar variable

Specifies the length of the transform. N must be an integer power of 2, that is, $N = 2^M$, where M is a positive integer.

M – INTEGER*4 scalar variable

M is a positive integer such that $N = 2^M$, where N is the user-specified length of the transform.

IFAIL – INTEGER*4 scalar variable

Unless the routine detects an error IFAIL will be set to zero on exit from the subroutine.

6 Error indicators

0 The subroutine function has worked correctly

1 N is not correctly specified

2 M is not correctly specified

7 Auxiliary routines

IN_FFT1_R*n* – Initialises the FFT's coefficient arrays for a given length N (the M^{th} power of 2).

System Routines

SPL_RADI2-R*n*
SPL_RADI4-R*n*
SPL_IRADI8-R*n*
SPL_LRADI8-R*n*

SPL_MR_REV_82
SPL_MR_REV_84
SPL_MR_REV_88

8 Further comments

COMMON blocks used – SPL_FFT_R*n*, SPL_MACH, SPL_TMP0_I*n* to SPL_TMP4_I*n*, SPL_TRIG

9 Keywords

FFT, fast Fourier transforms, spectral analysis

3.3 Real forward FFT

3.3.1 RF_FFT1_I*n* **release 2**

1 Purpose

RF_FFT1_I*n* calculates a forward fast Fourier transform (FFT) of real input data held in an array of numbers of type INTEGER*n, where n can have any value from 1 to 8.

2 Specification

SUBROUTINE RF_FFT1_I*n* (X, Y, N, M, IFAIL)
INTEGER*n X(,,N), Y(,,N)
INTEGER*4 N, M, IFAIL

3 Description

RF_FFT1_I*n* simultaneously evaluates ES^2 N-point real FFTs, where ES is the edge size od the target DAP. X (real part) and Y (set to zero) are both N-deep matrix arrays with data arranged sequentially 'down the PEs'. An 'in-place' FFT algorithm is used, so the output spectral data will be stored in X and Y, and the user's input data over-written.

4 References

[1] Brigham E O

The Fast Fourier Transform: Prentice-Hall, 1974

5 Arguments

X – INTEGER*n matrix variable N-deep

Contains the real input data on entry to the subroutine, and the real spectrum on exit.

Y – INTEGER*n matrix variable N-deep

Set to zero on entry to the subroutine, and contains the imaginary spectrum on exit.

N – INTEGER*4 scalar variable

Specifies the length of the transform. N must be an integer power of 2, that is, $N = 2^M$, where M is a positive integer.

M – INTEGER*4 scalar variable

M is a positive integer such that $N = 2^M$, where N is the user-specified length of the transform.

IFAIL – INTEGER*4 scalar variable

Unless the routine detects an error IFAIL will be set to zero on exit from the subroutine.

6 Error indicators

0 The subroutine function has worked correctly

1 N is not correctly specified

2 M is not correctly specified

7 Auxiliary routines

IN_FFT1_I*n* – Initialises the FFTs coefficient arrays for a given length N (the M^{th} power of 2).

System Routines

SPL_RRAD2-I*n*
SPL_RRAD4-I*n*
SPL_IRRAD8-I*n*
SPL_LRRAD8-I*n*

SPL_MR_REV_82
SPL_MR_REV_84
SPL_MR_REV_88

8 Accuracy

Details to be supplied

9 Further comments

COMMON blocks used – SPL_FFT_I*n*, SPL_MACH, SPL_TMP0_I*n* to SPL_TMP4_I*n*, SPL_TRIG

10 Keywords

FFT, fast Fourier transforms, spectral analysis

3.3.2 RF_FFT1_R*n* **release 2**

1 Purpose

RF_FFT1_R*n* calculates a forward fast Fourier transform (FFT) of real input data held in an array of numbers of type REAL*n, where n can have any value from 3 to 8.

2 Specification

SUBROUTINE RF_FFT1_R*n* (X, Y, N, M, IFAIL)
REAL*n X(,,N), Y(,,N)
INTEGER*4 N, M, IFAIL

3 Description

RF_FFT1_R*n* simultaneously evaluates ES^2 N-point real FFTs, where ES is the edge size od the target DAP. X (real part) and Y (set to zero) are both N-deep matrix arrays with data arranged sequentially 'down the PEs'. An 'in-place' FFT algorithm is used, so the output spectral data will be stored in X and Y, and the user's input data over-written.

4 References

[1] Brigham E O

The Fast Fourier Transform: Prentice-Hall, 1974.

5 Arguments

X – REAL*n matrix variable N-deep

Contains the real input data on entry to the subroutine, and the real spectrum on exit.

Y – REAL*n matrix variable N-deep

Set to zero on entry to the subroutine, and contains the imaginary spectrum on exit.

N – INTEGER*4 scalar variable

Specifies the length of the transform. N must be an integer power of 2, that is, $N = 2^M$, where M is a positive integer.

M – INTEGER*4 scalar variable

M is a positive integer such that $N = 2^M$, and N is the user-specified length of the transform.

IFAIL – INTEGER*4 scalar variable

Unless the routine detects an error IFAIL will be set to zero on exit from the subroutine.

6 Error indicators

0 The subroutine function has worked correctly

1 N is not correctly specified

2 M is not correctly specified

7 Auxiliary routines

IN_FFT1_R*n* – Initialises the FFT's coefficient arrays for a given length N (the M^{th} power of 2).

System Routines

SPL_RRAD2-R*n*
SPL_RRAD4-R*n*
SPL_IRRAD8-R*n*
SPL_LRRAD8-R*n*

SPL_MR_REV_82
SPL_MR_REV_84
SPL_MR_REV_88

8 Accuracy

Details to be supplied

9 Further comments

COMMON blocks used – SPL_FFT_R*n*, SPL_MACH, SPL_TMP0_I*n* to SPL_TMP4_I*n*, SPL_TRIG

10 Keywords

FFT, fast Fourier transforms, spectral analysis

3.4 Real inverse FFT

3.4.1 RI_FFT1_I*n* **release 2**

1 Purpose

RI_FFT1_I*n* calculates an inverse fast Fourier transform (FFT) of complex spectral input data held in an array of numbers of type INTEGER*n, where n can have any value from 1 to 8, and produces a real output.

2 Specification

SUBROUTINE RI_FFT1_I*n* (X, Y, N, M, IFAIL)
INTEGER*n X(,,N), Y(,,N)
INTEGER*4 N, M, IFAIL

3 Description

RI_FFT1_I*n* simultaneously evaluates ES^2 N-point real inverse FFTs, where ES is the edge size od the target DAP. X (real part) and Y (imaginary part) are both N-deep matrix arrays with data arranged sequentially 'down the PEs'. An 'in-place' inverse FFT algorithm is used, so the output data will be stored in X, and the user's input data over-written. Y will be set to zero.

4 References

[1] Brigham E O

The Fast Fourier Transform: Prentice-Hall, 1974

5 Arguments

X – INTEGER*n matrix variable N-deep

Contains the real spectral data on entry to the subroutine, and real data on exit.

Y – INTEGER*n matrix variable N-deep

Contains the imaginary spectral data on entry to the subroutine, and zero on exit.

N – INTEGER*4 scalar variable

Specifies the length of the transform. N must be an integer power of 2, that is, $N = 2^M$, where M is a positive integer.

M – INTEGER*4 scalar variable

M is a positive integer such that $N = 2^M$, where N is the user-specified length of the transform.

IFAIL – INTEGER*4 scalar variable

Unless the routine detects an error IFAIL will be set to zero on exit from the subroutine.

6 Error indicators

0 The subroutine function has worked correctly

1 N is not correctly specified

2 M is not correctly specified

7 Auxiliary routines

IN_FFT1_I*n* – Initialises the FFTs coefficient arrays for a given length N (the M^{th} power of 2).

System Routines

SPL_RRADI2-I*n*
SPL_RRADI4-I*n*
SPL_RIRADI8-I*n*
SPL_LRRADI8-I*n*

SPL_MR_REV_82
SPL_MR_REV_84
SPL_MR_REV_88

8 Accuracy

Details to be supplied

9 Further comments

COMMON blocks used – SPL_FCODE_I*n*, SPL_FFT_I*n*, SPL_MACH, SPL_TMP0_I*n* to SPL_TMP4_I*n*, SPL_TRIG

10 Keywords

FFT, fast Fourier transforms, spectral analysis

3.4.2 RI_FFT1_R*n* release 2

1 Purpose

RI_FFT1_R*n* calculates an inverse fast Fourier transform (FFT) of complex spectral input data held in an array of numbers of type REAL*n, where n can have any value from 3 to 8, and produces a real output.

2 Specification

SUBROUTINE RI_FFT1_R*n* (X, Y, N, M, IFAIL)
REAL*n X(,,N), Y(,,N)
INTEGER*4 N, M, IFAIL

3 Description

RI_FFT1_R*n* simultaneously evaluates ES^2 N-point real inverse FFTs, where ES is the edge size od the target DAP. X (real part) and Y (imaginary part) are both N-deep matrix arrays with data arranged sequentially 'down the PEs'. An 'in-place' inverse FFT algorithm is used, so the output data will be stored in X, and the user's input data over-written. Y will be set to zero.

4 References

[1] Brigham E O

The Fast Fourier Transform: Prentice-Hall, 1974.

5 Arguments

X – REAL*n matrix variable N-deep

Contains the real spectral data on entry to the subroutine, and real data on exit.

Y – REAL*n matrix variable N-deep

Contains the imaginary spectral data on entry to the subroutine, and zero on exit.

N – INTEGER*4 scalar variable

Specifies the length of the transform. N must be an integer power of 2, that is, $N = 2^M$, where M is a positive integer.

M – INTEGER*4 scalar variable

M is a positive integer such that $N = 2^M$, where N is the user-specified length of the transform.

IFAIL – INTEGER*4 scalar variable

Unless the routine detects an error IFAIL will be set to zero on exit from the subroutine.

6 Error indicators

0 The subroutine function has worked correctly

1 N is not correctly specified

2 M is not correctly specified

7 Auxiliary routines

IN_FFT1_R*n* – Initialises the FFT's arrays for a given length N (the M^{th} power of 2).

System Routines

SPL_RRADI2-R*n*
SPL_RRADI4-R*n*
SPL_IRRADI8-R*n*
SPL_LRRADI8-R*n*

SPL_MR_REV_82
SPL_MR_REV_84
SPL_MR_REV_88

8 Accuracy

Details to be supplied

9 Further comments

COMMON blocks used – SPL_FFT_R*n*, SPL_MACH, SPL_TMP0_I*n* to SPL_TMP4_I*n*, SPL_TRIG

10 Keywords

FFT, fast Fourier transforms, spectral analysis

3.5 FFT initialisation

3.5.1 IN_FFT1_I*n* **release 2**

1 Purpose

IN_FFT1_I*n* initialises coefficients and parameters for CF_FFT1_I*n*, CI_FFT1_I*n*, RF_FFT1_I*n* and RI_FFT1_I*n*, where *n* can have any value between 1 and 8.

2 Specification

SUBROUTINE IN_FFT1_I*n* (N, M, IFAIL)
INTEGER*4 N, M, IFAIL

3 Description

IN_FFT1_I*n* generates two arrays of coefficients for use in CF_FFT1_I*n*, CI_FFT1_I*n*, RF_FFT1_I*n* and RI_FFT1_I*n*; where *n* can have any value from 1 to 8.

The arrays produced – CI*n* and SI*n* – are held in the common block SPL_FFT_I*n*, and are used by the system software. You need only call IN_FFT1_I*n* once before using any of the FFT subroutines listed above and having the same value of *n*.

4 References

[1] Brigham E O

The Fast Fourier Transform: Prentice-Hall, 1974

5 Arguments

N – INTEGER*4 scalar variable

Specifies the length of the transform. N must be an integer power of 2, that is, $N = 2^M$, where M is a positive integer.

M – INTEGER*4 scalar variable

M is a positive integer such that $N = 2^M$, where N is the user-specified length of the transform.

IFAIL – INTEGER*4 scalar variable

Unless the routine detects an error IFAIL will be set to zero on exit from the subroutine.

6 Error indicators

0 The subroutine function has worked correctly

1 N is not correctly specified

2 M is not correctly specified

3 N and M are not mutually compatible

7 Auxiliary routines

None

8 Accuracy

Details to be supplied

9 Further comments

COMMON blocks used – SPL_FCODE_I*n*, SPL_FFT_I*n*, SPL_MACH, SPL_TMP0_I*n* to SPL_TMP4_I*n*, SPL_TRIG

10 Keywords

FFT, fast Fourier transforms, spectral analysis

3.5.2 IN_FFT1_R*n* **release 2**

1 Purpose

IN_FFT1_R*n* initialises coefficients and parameters for CF_FFT1_R*n*, CI_FFT1_R*n*, RF_FFT1_R*n* and RI_FFT1_R*n*, where *n* can have any value between 3 and 8.

2 Specification

SUBROUTINE IN_FFT1_R*n* (N, M, IFAIL)
INTEGER*4 N, M, IFAIL

3 Description

IN_FFT1_R*n* generates two arrays of coefficients for use in CF_FFT1_R*n*, CI_FFT1_R*n*, RF_FFT1_R*n* and RI_FFT1_R*n*; where *n* can have any value from 3 to 8.

The arrays produced – CR*n* and SR*n* – are held in the common block SPL_FFT_R*n*, and are used by the system software. You need only call IN_FFT1_R*n* once before using any of the FFT subroutines listed above and having the same value of *n*.

4 References

[1] Brigham E O

The Fast Fourier Transform: Prentice-Hall, 1974

5 Arguments

N – INTEGER*4 scalar variable

Specifies the length of the transform. N must be an integer power of 2, that is, $N = 2^M$, where M is a positive integer.

M – INTEGER*4 scalar variable

M is a positive integer such that $N = 2^M$, where N is the user-specified length of the transform.

IFAIL – INTEGER*4 scalar variable

Unless the routine detects an error IFAIL will be set to zero on exit from the subroutine.

6 Error indicators

0 The subroutine function has worked correctly

1 N is not correctly specified

2 M is not correctly specified

3 N and M are not mutually compatible

7 Auxiliary routines

None

8 Accuracy

Details to be supplied

9 Further comments

COMMON blocks used – SPL_FCODE_R*n*, SPL_FFT_R*n*, SPL_MACH, SPL_TMP0_I*n* to SPL_TMP4_I*n*, SPL_TRIG

10 Keywords

FFT, fast Fourier transforms, spectral analysis

Chapter 4

High performance FFTs

Contents:

The fast Fourier transform routines described in this chapter offer improved performance over the routines described in the previous chapter. The new routines offer you a faster way to calculate FFTs, and provide a more general way to specify your input and required output data.

4.1 Forward FFTs

4.1.1 CF_FFT_T_*n*_*len*_IN_*m* **release 2**

1 Purpose

CF_FFT_T_*n*_*len*_IN_*m* calculates with tapered arithmetic the forward fast Fourier transform of complex input data held in a data array of type INTEGER*p. n is the length of the required transform, and can take any of the values 2, 4, 8, 16, 32, 64, 128, 256 or 512; *len* is the size of the array needed to hold the input (and output) data, and can take the value **S** or **L** corresponding to a value for p of 2 or 3; the real and imaginary components of the input data are assumed to have a precision of m bits, where m takes an integer value in the range 8 to 11 or up to 16 in some cases. The routine will accept input data components that are interleaved with, or separated from each other, and which can be digit-reversed or naturally ordered.

2 Specification

SUBROUTINE CF_FFT_T_*n*_*len*_IN_*m* (DATA, INFLAG, OUTFLAG)
INTEGER*p DATA(, , $2n+2$)
INTEGER*4 INFLAG, OUTFLAG

3 Description

CF_FFT_T_*n*_*len*_IN_*m* simultaneously calculates ES^2 n-point complex FFTs, where n is the length of the FFT that is to be calculated. The same data array, DATA, is used for input and output.

The routine assumes that the real and imaginary components of the input data have a precision of m bits, with a system limit of 16 on m. The number of bits needed to store the output spectra is at least $(m+1)$, and increases with n, the length of the transform. The 'precision growth' of the output spectral data compared with the input data puts a limit on the possible precision of the input data the routine will handle.

If you specify *len* to have the value **L**, then you have to declare the matrix array DATA as INTEGER*3, so 24 bits are available to hold each data value. For an n of 512, the precision growth is 5 bits, so for input data with m of 16 bits, output data is only 21 bits long. Hence, if you declare DATA as INTEGER*3, the routine puts no n-related restrictions on the precision of your input data, and an input data limit of 16 bits applies for FFTs of any length.

You may wish to conserve array memory space as much as possible – and specify *len* to have a value of **S**, and DATA as INTEGER*2. In this case, precision growth with n does affect the maximum possible input data precision the routine can handle. The table below lists the relationship between n and m_{max}, the maximum value of m the routine will handle with an INTEGER*2 data array:

n	2	4	8	16	32	64	128	256	512
m_{max}	15	15	14	14	13	13	12	12	11

For example, if you want to calculate the FFTs of ES^2 sets of data, 32 pairs of data in each set ($n = 32$), if you specify *len* as **S** and DATA as INTEGER*2, then the maximum precision of your input data the routine will handle is 13 bits, and you might use the routine:

CF_FFT_T_32_S_IN_13

Under these conditions, if you specify m to be more than 13, perhaps by using the routine CF_FFT_T_32_S_IN_15, your program will fail at the linking stage, with an error message similar to 'Names undefined ...'.

In all cases there is a limit to the values that the components of the input data can take for a given precision. Suppose one instance of the complex input data is $a + jb$, and an input data precision of m bits is assumed; a, b and m must satisfy the relationship:

$$\sqrt{(a^2+b^2)} < 2^{m-1}-1$$

If the relationship is not satisfied, there will be overflow during the calculations, but because these routines have been written to work at maximum speed, the normal DAP error reporting features have not been used, and you may not know overflow had occurred.

The output data is 'normalised', by dividing it by $2^{[(log_2 n)-pg]}$, where pg is the precision growth for the length of transform concerned. The normalisation factors NF and precision growths for different lengths of transform are:

n	2	4	8	16	32	64	128	256	512
pg	1	1	2	2	3	3	4	4	5
NF	1	2	2	4	4	8	8	16	16

4 References

None

5 Arguments

DATA – INTEGER*p

On entry, the routine expects to find the data to be transformed in DATA, and to be right aligned (that is, in the least significant end of each element in DATA), in a mapping defined by the value in INFLAG. If you specify *len* to be **L** then you must declare DATA as INTEGER*3 (24 bits); if you specify *len* to be **S** you must declare DATA as INTEGER*2 (16 bits).

Note that you should declare DATA two matrices larger than needed to cope with the input data; this extra area is used for workspace.

On output, DATA holds the normalised spectral output of the transform, in a format defined by OUTFLAG, right-aligned, and sign-extended to the most significant end of each element of DATA, to use all of the bits in each element.

5 **Arguments** – continued

INFLAG – INTEGER*4

Specifies the assumed input data mapping, and can have any one of the following values:

0 The input data is assumed to be digit-reversed with real and imaginary parts interleaved

1 The input data is assumed to be digit-reversed with real and imaginary parts separated

2 The input data is assumed to be naturally ordered with real and imaginary parts interleaved

3 The input data is assumed to be naturally ordered with real and imaginary parts separated

The routine expects to find its input data in the first $2n$ blocks of INTEGER*p planes in DATA. If the real and imaginary parts of the complex input data are separated, the first n blocks of DATA will contain the real parts of all the input data, and second n blocks all the imaginary parts. If the real and imaginary parts are interleaved, then the real part of any given data input pair is immediately followed by the pair's imaginary part. In all cases, the last two blocks of INTEGER*p planes of DATA are used by the routine for workspace.

'Naturally ordered' and 'digit-reversed' refer to the ordering the routine assumes for the input data in DATA. The table below lists the address bits needed to address the real and imaginary components of the input data for different values of n for input data assumed to be naturally ordered. a_0 is the most significant bit of the address of the input data pair; a_z is the bit that differentiates between the real and the imaginary components of the data, so is the most significant bit of the total address for data assumed to be separated, and the least significant bit for data assumed to be interleaved. The real component of a data pair is assumed to have a lower address in DATA than the imaginary component.

Length 'n'	*Naturally ordered address bits*	
of transform	interleaved data components	separated data components
2	$a_0\ a_z$	$a_z\ a_0$
4	$a_0a_1\ a_z$	$a_z\ a_0a_1$
8	$a_0a_1a_2\ a_z$	$a_z\ a_0a_1a_2$
16	$a_0a_1a_2\ a_3\ a_z$	$a_z\ a_0a_1a_2\ a_3$
32	$a_0a_1a_2\ a_3a_4\ a_z$	$a_z\ a_0a_1a_2\ a_3a_4$
64	$a_0a_1a_2\ a_3a_4a_5\ a_z$	$a_z\ a_0a_1a_2\ a_3a_4a_5$
128	$a_0a_1a_2\ a_3a_4a_5\ a_6\ a_z$	$a_z\ a_0a_1a_2\ a_3a_4a_5\ a_6$
256	$a_0a_1a_2\ a_3a_4a_5\ a_6a_7\ a_z$	$a_z\ a_0a_1a_2\ a_3a_4a_5\ a_6a_7$
512	$a_0a_1a_2\ a_3a_4a_5\ a_6a_7a_8\ a_z$	$a_z\ a_0a_1a_2\ a_3a_4a_5\ a_6a_7a_8$

The table on the next page shows the effect of 'digit-reverse' of the ordering of address bits, using the same conventions as in the table above.

5 Arguments – continued

Length 'n' of transform	*Digit-reversed address bits* interleaved data components	separated data components
2	$a_0 \quad a_z$	$a_z \quad a_0$
4	$a_0a_1 \quad a_z$	$a_z \quad a_0a_1$
8	$a_0a_1a_2 \quad a_z$	$a_z \quad a_0a_1a_1$
16	$a_3 \quad a_0a_1a_2 \quad a_z$	$a_z \quad a_3 \quad a_0a_1a_2$
32	$a_3a_4 \quad a_0a_1a_2 \quad a_z$	$a_z \quad a_3a_4 \quad a_0a_1a_2$
64	$a_3a_4a_5 \quad a_0a_1a_2 \quad a_z$	$a_z \quad a_3a_4a_5 \quad a_0a_1a_2$
128	$a_6 \quad a_3a_4a_5 \quad a_0a_1a_2 \quad a_z$	$a_z \quad a_6 \quad a_3a_4a_5 \quad a_0a_1a_2$
256	$a_6a_7 \quad a_3a_4a_5 \quad a_0a_1a_2 \quad a_z$	$a_z \quad a_6a_7 \quad a_3a_4a_5 \quad a_0a_1a_2$
512	$a_6a_7a_8 \quad a_3a_4a_5 \quad a_0a_1a_2 \quad a_z$	$a_z \quad a_6a_7a_8 \quad a_3a_4a_5 \quad a_0a_1a_2$

For example, if n is 32, and the routine expects naturally ordered input data with real and imaginary parts interleaved (INFLAG = 2), then it would assume that the imaginary part of the fifth input data pair would be stored in DATA(,,10) (that is, would have binary address within DATA of (001 00 1) (note that the first plane is numbered 1 in FORTRAN-PLUS, but is numbered 0 as far as the actual address bits are concerned).

If naturally ordered and separated data were expected (INFLAG = 3), the address of the imaginary part of the fifth input data pair would be DATA(,,37), with a binary address of (1 001 00).

If the routine expects the input data to be digit-reversed and interleaved (INFLAG = 0), then it would assume that the imaginary part of the fifth data pair had a binary address of (00 001 1) and would be stored in DATA(,,4).

If the routine expects the input data to be digit-reversed and separated (INFLAG = 1), then it would assume that the imaginary part of the fifth data pair had a binary address of (1 00 001) and would be stored in DATA(,,34).

OUTFLAG – INTEGER*4

Specifies the required mapping of the normalised spectral data in DATA on output, and can have either of the following values:

0 The real and imaginary parts of the output spectra are interleaved, and are naturally ordered

1 The real and imaginary parts of the output spectra are separated, and are naturally ordered

As with the assumed mapping of the input data, the output data is held either interleaved or separated, in the first $2n$ blocks of DATA

6 Error indicators

None – in order to process the input data at maximum speed, no error detection is used.

7 Auxiliary routines

None

8 Accuracy

Details to be supplied

9 Further comments

None

10 Keywords

FFT, fast Fourier transforms

4.2 Inverse FFTs

4.2.1 CI_FFT_T_*n*_*len*_IN_*m* release 2

1 Purpose

CI_FFT_T_*n*_*len*_IN_*m* calculates with tapered arithmetic the inverse fast Fourier transform of complex input spectral data held in a data array of type INTEGER*p. n is the length of the required transform, and can take any of the values 2, 4, 8, 16, 32, 64, 128, 256 or 512; *len* is the size of the array needed to hold the input (and output) data, and can take the value **S** or **L** corresponding to a value for p of 2 or 3; the real and imaginary components of the input spectra are assumed to have a precision of m bits, where m takes an integer value in the range 8 to 11 or up to 16 in some cases. The routine will accept input data components that are interleaved with, or separated from each other, and which can be digit-reversed or naturally ordered.

2 Specification

SUBROUTINE CI_FFT_T_*n*_*len*_IN_*m* (DATA, INFLAG, OUTFLAG)
INTEGER*p DATA(,,$2n+2$)
INTEGER*4 INFLAG, OUTFLAG

3 Description

CI_FFT_T_*n*_*len*_IN_*m* simultaneously calculates ES^2 n-point complex inverse FFTs, where n is the length of the FFT that is to be calculated. The same data array, DATA, is used for input and output.

The routine assumes that the real and imaginary components of the input spectral data have a precision of m bits, with a system limit of 16 on m. The number of bits needed to store the output data is at least $(m+1)$, and increases with n, the length of the transform. The 'precision growth' of the output data compared with the input data puts a limit on the possible precision of the input data the routine will handle.

If you specify *len* to have the value **L**, then you have to declare the matrix array DATA as INTEGER*3, so 24 bits are available to hold each data value. For an n of 512, the precision growth is 5 bits, so for input data with m of 16 bits, output data is only 21 bits long. Hence, if you declare DATA as INTEGER*3, the routine puts no n-related restrictions on the precision of your input data, and an input data limit of 16 bits applies for FFTs of any length.

You may wish to conserve array memory space as much as possible – and specify *len* to have a value of **S**, and DATA as INTEGER*2. In this case, precision growth with n does affect the maximum possible input data precision the routine can handle. The table below lists the relationship between n and m_{max}, the maximum value of m the routine will handle with an INTEGER*2 data array:

n	2	4	8	16	32	64	128	256	512
m_{max}	15	15	14	14	13	13	12	12	11

For example, if you want to calculate the inverse FFTs of ES^2 sets of spectral data, 128 pairs of data in each set ($n = 128$), if you specify *len* as **S** and DATA as INTEGER*2, then

the maximum precision of your input data the routine will handle is 12 bits, and you might use the routine:

CI_FFT_T_32_S_IN_12

Under these conditions, if you specify m to be more than 12, perhaps by using the routine CI_FFT_T_32_S_IN_15, your program will fail at the linking stage, with an error message similar to 'Names undefined ...'.

In all cases there is a limit to the values that the components of the input data can take for a given precision. Suppose one instance of the complex input data is $a + jb$, and an input data precision of m bits is assumed; a, b and m must satisfy the relationship:

$$\sqrt{(a^2+b^2)} < 2^{m-1}-1$$

If the relationship is not satisfied, there will be overflow during the calculations, but because these routines have been written to work at maximum speed, the normal DAP error reporting features have not been used, and you may not know overflow had occurred.

The output data is 'normalised', by dividing it by $2^{[(log_2 n)-pg]}$, where pg is the precision growth for the length of transform concerned. The normalisation factors NF and precision growths for different lengths of transform are:

n	2	4	8	16	32	64	128	256	512
pg	1	1	2	2	3	3	4	4	5
NF	1	2	2	4	4	8	8	16	16

4 References

None

5 Arguments

DATA – INTEGER*p

On entry, the routine expects to find the spectral data to be transformed in DATA, and to be right aligned (that is, in the least significant end of each element in DATA), in a mapping defined by the value in INFLAG. If you specify *len* to be **L** then you must declare DATA as INTEGER*3 (24 bits); if you specify *len* to be **S** you must declare DATA as INTEGER*2 (16 bits).

Note that you should declare DATA two matrices larger than needed to cope with the input data; this extra area is used for workspace.

On output, DATA holds the normalised output of the transform, in a format defined by OUTFLAG, right-aligned, and sign-extended to the most significant end of each element of DATA, to use all of the bits in each element.

5 **Arguments** – continued

INFLAG – INTEGER*4

Specifies the assumed input data mapping, and can have any one of the following values:

0 The input data is assumed to be digit-reversed with real and imaginary parts interleaved

1 The input data is assumed to be digit-reversed with real and imaginary parts separated

2 The input data is assumed to be naturally ordered with real and imaginary parts interleaved

3 The input data is assumed to be naturally ordered with real and imaginary parts separated

The routine expects to find its input data in the first $2n$ blocks of INTEGER*p planes in DATA. If the real and imaginary parts of the complex input data are separated, the first n blocks of DATA will contain the real parts of all the input data, and second n blocks all the imaginary parts. If the real and imaginary parts are interleaved, then the real part of any given data input pair is immediately followed by the pair's imaginary part. In all cases, the last two blocks of INTEGER*p planes of DATA are used by the routine for workspace.

'Naturally ordered' and 'digit-reversed' refer to the ordering the routine assumes for the input data in DATA. The table below lists the address bits needed to address the real and imaginary components of the input data for different values of n for input data assumed to be naturally ordered. a_0 is the most significant bit of the address of the input data pair; a_z is the bit that differentiates between the real and the imaginary components of the data, so is the most significant bit of the total address for data assumed to be separated, and the least significant bit for data assumed to be interleaved. The real component of a data pair is assumed to have a lower address in DATA than the imaginary component.

Length 'n' of transform	*Naturally ordered address bits*	
	interleaved data components	separated data components
2	$a_0\ a_z$	$a_z\ a_0$
4	$a_0a_1\ a_z$	$a_z\ a_0a_1$
8	$a_0a_1a_2\ a_z$	$a_z\ a_0a_1a_2$
16	$a_0a_1a_2\ a_3\ a_z$	$a_z\ a_0a_1a_2\ a_3$
32	$a_0a_1a_2\ a_3a_4\ a_z$	$a_z\ a_0a_1a_2\ a_3a_4$
64	$a_0a_1a_2\ a_3a_4a_5\ a_z$	$a_z\ a_0a_1a_2\ a_3a_4a_5$
128	$a_0a_1a_2\ a_3a_4a_5\ a_6\ a_z$	$a_z\ a_0a_1a_2\ a_3a_4a_5\ a_6$
256	$a_0a_1a_2\ a_3a_4a_5\ a_6a_7\ a_z$	$a_z\ a_0a_1a_2\ a_3a_4a_5\ a_6a_7$
512	$a_0a_1a_2\ a_3a_4a_5\ a_6a_7a_8\ a_z$	$a_z\ a_0a_1a_2\ a_3a_4a_5\ a_6a_7a_8$

The table on the next page shows the effect of 'digit-reverse' of the ordering of address bits, using the same conventions as in the table above.

5 Arguments – continued

Length 'n' of transform	*Digit-reversed address bits* interleaved data components	separated data components
2	$a_0 \quad a_z$	$a_z \quad a_0$
4	$a_0a_1 \quad a_z$	$a_z \quad a_0a_1$
8	$a_0a_1a_2 \quad a_z$	$a_z \quad a_0a_1a_1$
16	$a_3 \quad a_0a_1a_2 \quad a_z$	$a_z \quad a_3 \quad a_0a_1a_2$
32	$a_3a_4 \quad a_0a_1a_2 \quad a_z$	$a_z \quad a_3a_4 \quad a_0a_1a_2$
64	$a_3a_4a_5 \quad a_0a_1a_2 \quad a_z$	$a_z \quad a_3a_4a_5 \quad a_0a_1a_2$
128	$a_6 \quad a_3a_4a_5 \quad a_0a_1a_2 \quad a_z$	$a_z \quad a_6 \quad a_3a_4a_5 \quad a_0a_1a_2$
256	$a_6a_7 \quad a_3a_4a_5 \quad a_0a_1a_2 \quad a_z$	$a_z \quad a_6a_7 \quad a_3a_4a_5 \quad a_0a_1a_2$
512	$a_6a_7a_8 \quad a_3a_4a_5 \quad a_0a_1a_2 \quad a_z$	$a_z \quad a_6a_7a_8 \quad a_3a_4a_5 \quad a_0a_1a_2$

For example, if n is 64, and the routine expects naturally ordered input data with real and imaginary parts interleaved (INFLAG = 2), then it would assume that the real part of the twentieth input data pair would be stored in DATA(,,39) (that is, would have binary address within DATA of (010 011 0) (note that the first plane is numbered 1 in FORTRAN-PLUS, but is numbered 0 as far as the actual address bits are concerned).

If naturally ordered and separated data were expected (INFLAG = 3), the address of the real part of the twentieth input data pair would be DATA(,,20), with a binary address of (0 010 011).

If the routine expects the input data to be digit-reversed and interleaved (INFLAG = 0), then it would assume that the real part of the twentieth data pair had a binary address of (011 010 0) and would be stored in DATA(,,53).

If the routine expects the input data to be digit-reversed and separated (INFLAG = 1), then it would assume that the real part of the twentieth data pair had a binary address of (0 011 010) and would be stored in DATA(,,27).

OUTFLAG – INTEGER*4

Specifies the required mapping of the normalised data in DATA on output, and can have either of the following values:

0 The real and imaginary parts of the output data are interleaved, and are naturally ordered

1 The real and imaginary parts of the output data are separated, and are naturally ordered

As with the assumed mapping of the input data, the output data is held either interleaved or separated, in the first $2n$ blocks of DATA

6 Error indicators

None – in order to process the input data at maximum speed, no error detection is used.

7 Auxiliary routines

None

8 Accuracy

Details to be supplied

9 Further comments

None

10 Keywords

FFT, fast Fourier transforms

Chapter 5

Windowing

Contents:

5.1 Windowing

5.1.1 WIN1_I*n* **release 2**

1 Purpose

WIN1_I*n* is a windowing subroutine for INTEGER*n real or imaginary arrays, where n can have any value from 1 to 8.

2 Specification

SUBROUTINE WIN1_I*n* (X, N, IFAIL)
INTEGER*n X(,,N)
INTEGER*4 N, IFAIL

3 Description

WIN1_I*n* windows an INTEGER*n data array with a window function held in a COMMON block, where n can have any value from 1 to 8. Scaling arithmetic is included to avoid overflows and underflows.

4 References

[1] Harris F J

On the use of windows for harmonic analysis with the discrete Fourier transform: Proc IEEE, Vol 66, No 1, January 1978, pp 51-83.

5 Arguments

X – INTEGER*n

On input the array contains the user input data, which is replaced on output by the windowed data.

N – INTEGER*4 scalar

Specifies the array length. N must be an integer power of 2, that is, $N = 2^M$, where M is a positive integer.

IFAIL – INTEGER*4 scalar

Unless the subroutine detects an error IFAIL will be set to zero on exit from the subroutine.

6 Error Indicators

0 The subroutine function has worked correctly

1 N is not correctly specified

7 Auxiliary Routines

System routine

SPL_WIN1_I*n*

8 Accuracy

Details to be supplied

9 Further Comments

COMMON blocks used – SPL_MACH, SPL_TRIG, SPL_WINF1_*In*

10 Keywords

FFT, Fast Fourier Transforms, Spectral analysis

5.1.2 WIN1_R*n* **release 2**

1 Purpose

WIN1_R*n* is a windowing subroutine for REAL*n real or imaginary arrays, where n can have any value from 3 to 8.

2 Specification

SUBROUTINE WIN1_R*n* (X, N, IFAIL)
REAL*n X(,,N)
INTEGER*4 N, IFAIL

3 Description

WIN1_R*n* windows an REAL*n data array with a window function held in a COMMON block, where n can have any value from 3 to 8. Scaling arithmetic is included to avoid overflows and underflows.

4 References

[1] Harris F J

On the use of windows for harmonic analysis with the discrete Fourier transform: Proc IEEE, Vol 66, No 1, January 1978, pp 51-83.

5 Arguments

X – REAL*n

On input the array contains the user data, which is replaced on output by the windowed data.

N – INTEGER*4 scalar

Specifies the array length. N must be an integer power of 2, that is, $N = 2^M$, where M is a positive integer.

IFAIL – INTEGER*4 scalar

Unless the subroutine detects an error IFAIL will be set to zero on exit from the subroutine.

6 Error Indicators

0 The subroutine function has worked correctly

1 N is not correctly specified

7 Auxiliary Routines

System routine

SPL_WIN1_R*n*

8 Accuracy

Details to be supplied

9 Further Comments

COMMON blocks used – SPL_MACH, SPL_TRIG SPL_WINF1_R*n*

10 Keywords

FFT, Fast Fourier Transforms, Spectral Analysis

5.2 Window initialisation

5.2.1 IN_WIN1_I*n* **release 2**

1 Purpose

IN_WIN1_I*n* is a window initialisation subroutine for INTEGER*n precision, where n can have any value from 1 to 8.

2 Specification

SUBROUTINE IN_WIN1_I*n* (FUNC, ALPHA, BETA, N, IFAIL)
REAL*4 ALPHA, BETA
INTEGER*4 FUNC, N, IFAIL

3 Description

IN_WIN1_I*n* initialises an array with a suitably scaled window function, where n can have any value from 1 to 8.

Scaling is essential because the window functions used are never greater than unity; the scale factor gives a maximum window value of half the full positive range - 1, that is, for $n = 5$, the maximum window value is 15.

4 References

[1] Harris F J

On the use of windows for harmonic analysis with the discrete Fourier transform: Proc IEEE, Vol 66, No 1, January 1978, pp 51-83.

5 Arguments

FUNC – INTEGER*4

When you call IN_WIN1_I*n* you must always give a value for each of the four parameters to the routine, even though the values you give for ALPHA or BETA or both are not used for some values of FUNC. In release 1 of DSPLIB FUNC must lie in the range 1 -7; the list below gives the various functions that can be called, details of the use made of ALPHA and BETA, and the algorithms used to calculate the window function $w(n)$:

1 – Hamming; neither parameter is used

$$w(n) = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N}\right)$$

2 – Hanning; only ALPHA is used

$$w(n) = \sin^{\alpha}\left(\frac{\pi n}{N}\right)$$

3 – Gaussian; only ALPHA is used, it must be positive

$$w(n) = \exp\left(-\frac{1}{2}\left(\alpha\frac{n}{N/2}\right)^2\right)$$

4 – Dolph-Chebyshev; only ALPHA is used, it must be positive

$$w(n) = F^{-1}(W(k))$$

where:

F^{-1} is the inverse DFT

$$W(k) = (-1)^k \frac{\cos\left(N \cos^{-1}\left(\beta \cos\left(\frac{\pi k}{N}\right)\right)\right)}{\cosh\left(N \cosh^{-1}(\beta)\right)}$$

$$\beta = \cosh\left(\frac{1}{N} \cosh^{-1}(10^{\alpha})\right)$$

α is the number of decades of sidelobe level

5 – Blackman; neither parameter is used

$$w(n) = \sum_{n=0}^{\frac{N}{2}} (-1)^m \,.\, a_m \,.\, \cos\left(\frac{2\pi nm}{N}\right)$$

where $\sum_{m=0}^{\frac{N}{2}} a_m = 1.0$

6 – Blackman-Harris; only ALPHA is used, it must take the value 3.0 or 4.0 - the number of samples in the truncated series. The window function $w(n)$ is a truncated BLACKMAN series

7 – Kaiser; only ALPHA used, it must be positive

$$w(n) = \frac{I_0\left(\pi\alpha\sqrt{1.0 - \left(\frac{2n}{N}\right)^2}\right)}{I_0[\pi\alpha]}$$

where:

$$I_0(x) = \sum_{k=0}^{\infty} \left(\frac{\left(\frac{x}{2}\right)^k}{k!}\right)^2$$

$\pi\alpha$ is the time-bandwidth product

ALPHA – REAL*4

BETA – REAL*4

N – INTEGER*4 scalar

Specifies the array length. N must be an integer power of 2, that is, $N = 2^M$, where M is a positive integer.

IFAIL – INTEGER*4 scalar

Unless the subroutine detects an error IFAIL will be set to zero on exit from the subroutine.

6 Error Indicators

0 The subroutine has worked correctly

1 N was not specified correctly

2 The value of ALPHA was not valid

3 The value of BETA was not valid

4 The function was not known

5 The function's parameters were not valid

7 Auxiliary Routines

System routines

SPL_IWIN1_I*n*
SPL_ICSH_R*m*
SPL_CSH_R*m*
SPL_ICOS_R*m*
SPL_BESSELI0_R*m*

where $m = 3$ for $n = 1, 2$ and $m = n$ for $n = 3, 4, 5, 6, 7, 8$

8 Accuracy

Details to be supplied

9 Further Comments

COMMON blocks used – SPL_MACH, SPL_TRIG, SPL_WINF1_I*n*

10 Keywords

FFT, Fast Fourier Transforms, Spectral analysis

5.2.2 IN_WIN1_R*n* release 2

1 Purpose

IN_WIN1_R*n* is a window initialisation subroutine for REAL*n precision, where n can have any value from 3 to 8.

2 Specification

```
SUBROUTINE IN_WIN1_Rn ( FUNC, ALPHA, BETA, N, IFAIL )
REAL*4  ALPHA, BETA
INTEGER*4 FUNC, N, IFAIL
```

3 Description

IN_WIN1_R*n* initialises an array with a window function, where n can have any value from 3 to 8.

4 References

[1] Harris F J

On the use of windows for harmonic analysis with the discrete Fourier transform: Proc IEEE, Vol 66, No 1, January 1978, pp 51-83.

5 Arguments

FUNC – INTEGER*4

When you call IN_WIN1_I*n* you must always give a value for each of the four parameters to the routine, even though the values you give for ALPHA or BETA or both are not used for some values of FUNC. In release 1 of DSPLIB FUNC must lie in the range 1 -7; the list below gives the various functions that can be called, details of the use made of ALPHA and BETA, and the algorithms used to calculate the window function $w(n)$:

1 – Hamming; neither parameter is used

$$w(n) = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N}\right)$$

2 – Hanning; only ALPHA is used

$$w(n) = \sin^{\alpha}\left(\frac{\pi n}{N}\right)$$

3 – Gaussian; only ALPHA is used, it must be positive

$$w(n) = \exp\left(-\frac{1}{2}\left(\alpha\frac{n}{N/2}\right)^2\right)$$

4 – Dolph-Chebyshev; only ALPHA is used, it must be positive

$$w(n) = F^{-1}(W(k))$$

where:

F^{-1} is the inverse DFT

$$W(k) = (-1)^k \frac{\cos\left(N \cos^{-1}\left(\beta \cos\left(\frac{\pi k}{N}\right)\right)\right)}{\cosh\left(N \cosh^{-1}(\beta)\right)}$$

$$\beta = \cosh\left(\frac{1}{N} \cosh^{-1}(10^{\alpha})\right)$$

α is the number of decades of sidelobe level

5 – Blackman; neither parameter is used

$$w(n) = \sum_{n=0}^{\frac{N}{2}} (-1)^m \cdot a_m \cdot \cos\left(\frac{2\pi n m}{N}\right)$$

where $\sum_{m=0}^{\frac{N}{2}} a_m = 1.0$

6 – Blackman-Harris; only ALPHA is used, it must take the value 3.0 or 4.0 - the number of samples in the truncated series. The window function $w(n)$ is a truncated BLACKMAN series

7 – Kaiser; only ALPHA used, it must be positive

$$w(n) = \frac{I_0\left(\pi\alpha\sqrt{1.0 - \left(\frac{2n}{N}\right)^2}\right)}{I_0[\pi\alpha]}$$

where:

$$I_0(x) = \sum_{k=0}^{\infty} \left(\frac{\left(\frac{x}{2}\right)^k}{k!}\right)^2$$

$\pi\alpha$ is the time-bandwidth product

ALPHA – REAL*4

BETA – REAL*4

N – INTEGER*4 scalar

Specifies the array length. N must be an integer power of 2, that is, N = 2^M, where M is a positive integer.

IFAIL – INTEGER*4 scalar

Unless the subroutine detects an error IFAIL will be set to zero on exit from the subroutine.

6 Error Indicators

0 The subroutine has worked correctly

1 N was not specified correctly

2 The value of ALPHA was not valid

3 The value of BETA was not valid

4 The function was not known

5 The function's parameters were not valid

7 Auxiliary Routines

System routines

SPL_IWIN1_R*n*
SPL_ICSH_R*n*
SPL_CSH_R*n*
SPL_ICOS_R*n*
SPL_BESSELI0_R*n*

8 Accuracy

Details to be supplied

9 Further Comments

COMMON blocks used – SPL_MACH, SPL_TRIGR, SPL_WINF1_R*n*

10 Keywords

FFT, Fast Fourier Transforms, Spectral analysis

Chapter 6

Signal generation

Contents:

6.1 Sine wave

6.1.1 CSINE_I*n*

1 Purpose

CSINE_I*n* generates a sine wave of INTEGER*n precision, where n can have any value from 1 to 8.

2 Specification

```
SUBROUTINE CSINE_In ( X, FREQ, AMPL, N, IFAIL )
INTEGER*n X(,,N)
REAL*4   FREQ, AMPL
INTEGER*4 N, IFAIL
```

3 Description

CSINE_I*n* sets a suitably scaled sine function into an array, where n can have any value from 1 to 8.

4 References

None

5 Arguments

X – INTEGER*n

A data array into which the sine function is to be put

FREQ – REAL*4

The frequency of the sine function; that is, the number of cycles set into the array

AMPL – REAL*4

The amplitude of the sine function

N – INTEGER*4 scalar

Specifies the array length. N must be an integer power of 2; that is $N = 2^M$, where M is a positive integer.

IFAIL – INTEGER*4 scalar

Unless the subroutine detects an error IFAIL will be set to zero on exit from the subroutine

6 Error Indicators

0 The subroutine has worked correctly

1 N was not specified correctly

2 The value given to FREQ was not valid

3 The value given to AMPL was not valid

7 Auxiliary Routines

System routines

SPL_CSINE_I*n*

8 Accuracy

Details to be supplied

9 Further Comments

COMMON blocks used – SPL_MACH, SPL_TRIG

10 Keywords

Sine, sine generation, sine wave generation

6.1.2 CSINE_R*n* **release 2**

1 Purpose

CSINE_R*n* generates a sine wave of REAL*n precision, where n can have any value from 3 to 8.

2 Specification

```
SUBROUTINE CSINE_Rn ( X, FREQ, AMPL, N, IFAIL )
REAL*n   X(,,N)
REAL*4   FREQ, AMPL
INTEGER*4 N, IFAIL
```

3 Description

CSINE_R*n* sets a suitably scaled sine function into an array, where n can have any value from 3 to 8.

4 References

None

5 Arguments

X – REAL*n

A data array into which the sine function is to be put

FREQ – REAL*4

The frequency of the sine function; that is, the number of cycles set into the array

AMPL – REAL*4

The amplitude of the sine function

N – INTEGER*4 scalar

Specifies the array length. N must be an integer power of 2; that is $N = 2^M$, where M is a positive integer.

IFAIL – INTEGER*4 scalar

Unless the subroutine detects an error IFAIL will be set to zero on exit from the subroutine

6 Error Indicators

0 The subroutine has worked correctly

1 N was not specified correctly

2 The value given to FREQ was not valid

3 The value given to AMPL was not valid

7 Auxiliary Routines

System routines

SPL_CSINE_R*n*

8 Accuracy

Details to be supplied

9 Further Comments

COMMON blocks used – SPL_MACH, SPL_TRIG

10 Keywords

Sine, sine generation, sine wave generation

6.2 Chirp sine wave

6.2.1 CHIRP_I*n* **release 2**

1 Purpose

CHIRP_I*n* generates a chirp sine wave with INTEGER*n precision, where n can have any value from 1 to 8.

2 Specification

SUBROUTINE CHIRP_I*n* (X, FREQS, FREQE, AMPL, N, IFAIL)
INTEGER*n X(,,N)
REAL*4 FREQS, FREQE, AMPL
INTEGER*4 N, IFAIL

3 Description

CHIRP_I*n* sets a suitably scaled chirp sine function into an array, where n can have any value from 1 to 8.

4 References

None

5 Arguments

X – INTEGER*n

The data array into which the chirp function will be put

FREQS – REAL*4

The start frequency of the chirp

FREQE – REAL*4

The end frequency of the chirp

AMPL – REAL*4

The amplitude of the chirp sine function

N – INTEGER*4 scalar

Specifies the array length. N must be an integer power of 2; that is $N = 2^M$, where M is a positive integer.

IFAIL – INTEGER*4 scalar

Unless the subroutine detects an error IFAIL will be set to zero on exit from the subroutine

6 Error Indicators

0 The subroutine worked correctly
1 N was not specified correctly
2 The values given to both FREQS and FREQE were not valid
3 The value given to FREQS was not valid
4 The value given to FREQE was not valid
5 The value given to AMPL was not valid

7 Auxiliary Routines

System routines

SPL_CHIRP_I*n*

8 Accuracy

Details to be supplied

9 Further Comments

COMMON blocks used – SPL_MACH, SPL_TRIG

10 Keywords

Chirp, chirp generation, chirp sine function

6.2.2 CHIRP_R*n* **release 2**

1 Purpose

CHIRP_R*n* generates a chirp sine wave with REAL**n* precision, where *n* can have any value from 3 to 8.

2 Specification

SUBROUTINE CHIRP_R*n* (X, FREQS, FREQE, AMPL, N, IFAIL)
REAL***n* X(, , N)
REAL*4 FREQS, FREQE, AMPL
INTEGER*4 N, IFAIL

3 Description

CHIRP_R*n* sets a suitably scaled chirp sine function into an array, where *n* can have any value from 3 to 8.

4 References

None

5 Arguments

X – REAL***n*

The data array into which the chirp function will be put

FREQS – REAL*4

The start frequency of the chirp

FREQE – REAL*4

The end frequency of the chirp

AMPL – REAL*4

The amplitude of the chirp sine function

N – INTEGER*4 scalar

Specifies the array length. N must be an integer power of 2; that is $N = 2^M$, where M is a positive integer.

IFAIL – INTEGER*4 scalar

Unless the subroutine detects an error IFAIL will be set to zero on exit from the subroutine

6 Error Indicators

0 The subroutine worked correctly
1 N was not specified correctly
2 The value given to FREQS was not valid
3 The values given to both FREQS and FREQE were not valid
4 The value given to FREQE was not valid
5 The value given to AMPL was not valid

7 Auxiliary Routines

System routines

SPL_CHIRP_R*n*

8 Accuracy

Details to be supplied

9 Further Comments

COMMON blocks used – SPL_MACH, SPL_TRIG

10 Keywords

Chirp, chirp generation, chirp sine function

6.3 Exponential decay

6.3.1 EX_DECAY_I*n* **release 2**

1 Purpose

EX_DECAY_I*n* generates an exponential decay function with INTEGER*n precision, where n can have any value from 1 to 8.

2 Specification

```
SUBROUTINE EX_DECAY_In ( X, Y, AMPL, N, IFAIL )
INTEGER*n X(,,N), Y(,,N)
REAL*4   AMPL
INTEGER*4 N, IFAIL
```

3 Description

EX_DECAY_I*n* sets a suitably scaled complex exponential decay function into X and Y, where n can have any value from 1 to 8. The $(i+1)^{th}$ matrix in each array contains values given by:

$$X_i + jY_i = \text{AMPL} \cdot Z^i \qquad \text{for } i = 0, 1, \ldots, N-1$$

where $Z = \dfrac{x + jy}{\sqrt{x^2 + y^2}}$

and x and y are initial values, input in the first matrices of the X and Y arrays respectively.

4 References

None

5 Arguments

X – INTEGER*n

On input the routine expects to find the real part of the complex initial values for the required decay function in the first matrix of the array; on exit the whole array contains the real part of the function

Y – INTEGER*n

On input the routine expects to find the imaginary part of the complex initial values for the required decay function in the first matrix of the array; on exit the whole array contains the imaginary part of the function

AMPL – REAL*4

The initial amplitude of the decay function

N – INTEGER*4 scalar

Specifies the array length. N must be a positive integer.

IFAIL – INTEGER*4 scalar

Unless the subroutine detects an error IFAIL will be set to zero on exit from the subroutine

6 Error Indicators

0 The subroutine worked correctly

1 N was not specified correctly

2 The magnitude of one or more of the input complex initial values was zero

3 The value given to AMPL was zero

7 Auxiliary Routines

SPL_CMPY_R*n*

8 Accuracy

Details to be supplied

9 Further Comments

COMMON blocks used – SPL_MACH, SPL_TRIG

10 Keywords

Exponential, decay, exponential decay function

6.3.2 EX_DECAY_R*n* **release 2**

1 Purpose

EX_DECAY_R*n* generates an exponential decay function with REAL*n precision, where n can have any value from 3 to 8.

2 Specification

```
SUBROUTINE EX_DECAY_Rn ( X, Y, AMPL, N, IFAIL )
REAL*n X(,,N), Y(,,N)
REAL*4  AMPL
INTEGER*4 N, IFAIL
```

3 Description

EX_DECAY_R*n* sets a suitably scaled complex exponential decay function into X and Y, where n can have any value from 3 to 8. The $(i+1)^{th}$ matrix in each array contains values given by:

$$X_i + jY_i = AMPL \,.\, Z^i \qquad \text{for } i = 0, 1, \ldots, N-1$$

where $Z = x + j\,y$

and x and y are initial values, input in the first matrices of the X and Y arrays respectively, and where $\sqrt{x^2 + y^2} \leq 1$ for all element pairs in those first matrices.

4 References

None

5 Arguments

X – REAL*n

On input the routine expects to find the real part of the complex initial values for the required decay function in the first matrix of the array; on exit the whole array contains the real part of the function

Y – REAL*n

On input the routine expects to find the imaginary part of the complex initial values for the required decay function in the first matrix of the array; on exit the whole array contains the imaginary part of the function

AMPL – REAL*4

The initial amplitude of the decay function

N – INTEGER*4 scalar

Specifies the array length. N must be a positive integer.

IFAIL – INTEGER*4 scalar

Unless the subroutine detects an error IFAIL will be set to zero on exit from the subroutine

6 Error Indicators

0 The subroutine worked correctly

1 N was not specified correctly

2 The magnitude of one or more of the input complex initial values was greater than 1

3 The value given to AMPL was zero

7 Auxiliary Routines

SPL_CMPY_R*n*

8 Accuracy

Details to be supplied

9 Further Comments

None

10 Keywords

Exponential, decay, exponential decay function

Chapter 7

Signal format conversion

Contents:

7.1 In-phase and quadature to power and phase

7.1.1 IQ_PWR_I*n* **release 2**

1 Purpose

IQ_PWR_I*n* converts signal data from the in-phase and quadrature form to the 'power' and phase angle form, with INTEGER*n precision, where n can have any value form 1 to 8.

2 Specification

SUBROUTINE IQ_PWR_I*n* (X, Y, N, IFAIL)
INTEGER*n X(,,N), Y(,,N)
INTEGER*4 N, IFAIL

3 Description

IQ_PWR_I*n* converts data representing signal in-phase and quadrature amplitudes to the equivalent 'power' and phase angle form, with INTEGER*n precision, where n can have any value form 1 to 8. The angle is given in degrees. The output 'power' data is presented in decibels relative to unity 'power'.

The subroutine places the output data in the arrays initially holding the input data, hence the initial data is overwritten.

4 References

None

5 Arguments

X – INTEGER*n

The array from which the subroutine takes the in-phase component of the data to be converted, and into which the 'power' component is placed

Y – INTEGER*n

The array from which the subroutine takes the quadrature component of the data to be converted, and into which the phase angle component, expressed in degrees, is placed

N – INTEGER*4 scalar

Specifies the array length. N must be an integer power of 2, that is, $N = 2^M$, where M is a positive integer.

IFAIL – INTEGER*4 scalar

Unless the subroutine detects an error IFAIL will be set to zero on exit from the subroutine

6 Error Indicators

0 The subroutine worked correctly

1 N was not specified correctly

7 Auxiliary Routines

System routines

SPL_IQPOW_I*n*
SPL_UNWRAP_I*n*
SPL_ATAN2_I*n*

8 Accuracy

Details to be supplied

9 Further Comments

COMMON blocks used – SPL_MACH, SPL_TRIG

10 Keywords

Amplitude to power, in-phase and quadrature to power and phase angle, conversion

7.1.2 IQ_PWR_R*n* **release 2**

1 Purpose

IQ_PWR_R*n* converts signal data from the in-phase and quadrature form to the 'power' and phase angle form, with REAL**n* precision, where *n* can have any value form 3 to 8.

2 Specification

SUBROUTINE IQ_PWR_R*n* (X, Y, N, IFAIL)
REAL**n* X(,,N), Y(,,N)
INTEGER*4 N, IFAIL

3 Description

IQ_PWR_R*n* converts data representing signal in-phase and quadrature amplitudes to the equivalent 'power' and phase angle form, with REAL**n* precision, where *n* can have any value form 3 to 8. The angle is given in radians. The output 'power' data is presented in decibels relative to unity 'power'.

The subroutine places the output data in the array initially holding the input data, hence the initial data is overwritten.

4 References

None

5 Arguments

X – REAL**n*

The array from which the subroutine takes the in-phase component of the data to be converted, and into which the 'power' component is placed

Y – REAL**n*

The array from which the subroutine takes the quadrature component of the data to be converted, and into which the phase angle component, expressed in radians, is placed

N – INTEGER*4 scalar

Specifies the array length. N must be an integer power of 2, that is, $N = 2^M$, where M is a positive integer.

IFAIL – INTEGER*4 scalar

Unless the subroutine detects an error IFAIL will be set to zero on exit from the subroutine

6 Error Indicators

0 The subroutine worked correctly

1 N was not specified correctly

7 Auxiliary Routines

System routines

SPL_IQPOW_I*n*
SPL_UNWRAP_I*n*

SPL_ATAN2_I*n*

8 Accuracy

Details to be supplied

9 Further Comments

COMMON blocks used – SPL_MACH, SPL_TRIG

10 Keywords

Amplitude to power, in-phase and quadrature to power and phase angle, conversion

7.2 Power and phase to in-phase and quadrature

7.2.1 PWR_IQ_I*n* **release 2**

1 Purpose

PWR_IQ_I*n* converts signal data from the 'power' and phase angle form to the in-phase and quadrature form, with INTEGER*n precision, where n can have any value from 1 to 8.

2 Specification

```
SUBROUTINE PWR_IQ_In ( X, Y, N, IFAIL )
INTEGER*n X(,,N), Y(,,N)
INTEGER*4 N, IFAIL
```

3 Description

PWR_IQ_I*n* converts data representing signal 'power' and phase angle values to the equivalent in-phase and quadrature power form, with INTEGER*n precision, where n can have any value from 1 to 8. The routine assumes that the angles are expressed in degrees.

The subroutine places the output data in the array initially holding the input data, hence the initial data is overwritten.

4 References

None

5 Arguments

X – INTEGER*n

The array from which the subroutine takes the power component of the data to be converted, and into which the in-phase component is placed

Y – INTEGER*n

The array from which the subroutine takes the phase angle component of the data to be converted, and into which the quadrature component is placed

N – INTEGER*4 scalar

Specifies the array length. N must be an integer power of 2, that is, $N = 2^M$, where M is a positive integer.

IFAIL – INTEGER*4 scalar

Unless the subroutine detects an error IFAIL will be set to zero on exit from the subroutine

6 Error Indicators

0 The subroutine worked correctly

1 N was not specified correctly

7 Auxiliary Routines

System routines

SPL_POWIQ_I*n*

8 Accuracy

Details to be supplied

9 Further Comments

COMMON blocks used – SPL_MACH, SPL_TRIG

10 Keywords

Power to amplitude, power and phase angle to in-phase and quadrature, conversion

7.2.2 PWR_IQ_R*n* **release 2**

1 Purpose

PWR_IQ_R*n* converts signal data from the 'power' and phase angle form to the in-phase and quadrature form, with REAL*n precision, where n can have any value from 3 to 8.

2 Specification

SUBROUTINE PWR_IQ_R*n* (X, Y, N, IFAIL)
REAL*n X(,,N), Y(,,N)
INTEGER*4 N, IFAIL

3 Description

PWR_IQ_R*n* converts data representing signal 'power' and phase angle values to the equivalent in-phase and quadrature power form, with REAL*n precision, where n can have any value from 3 to 8.The routine assumes that the angles are expressed in radians.

The subroutine places the output data in the array initially holding the input data, hence the initial data is overwritten.

4 References

None

5 Arguments

X – REAL*n

The array from which the subroutine takes the power component of the data to be converted, and into which the in-phase component is placed

Y – REAL*n

The array from which the subroutine takes phase angle component of the data to be converted, and into which the quadrature component is placed

N – INTEGER*4 scalar

Specifies the array length. N must be an integer power of 2, that is, $N = 2^M$, where M is a positive integer.

IFAIL – INTEGER*4 scalar

Unless the subroutine detects an error IFAIL will be set to zero on exit from the subroutine

6 Error Indicators

0 The subroutine worked correctly

1 N was not specified correctly

7 Auxiliary Routines

System routines

SPL_POWIQ_I*n*

8 Accuracy

Details to be supplied

9 Further Comments

COMMON blocks used – SPL_MACH, SPL_TRIG

10 Keywords

Power to amplitude, power and phase angle to in-phase and quadrature, conversion

7.3 In-phase and quadrature to magnitude and phase

7.3.1 IQ_MAG_I*n* **release 2**

1 Purpose

IQ_MAG_I*n* converts signal data from the in-phase and quadrature form to the magnitude and phase angle form, with INTEGER*n precision, where n can have any value from 1 to 8.

2 Specification

SUBROUTINE IQ_MAG_I*n* (X, Y, N, IFAIL)
INTEGER*n X(,,N), Y(,,N)
INTEGER*4 N, IFAIL

3 Description

IQ_MAG_I*n* converts data representing signal in-phase and quadrature values to the equivalent magnitude and phase angle form, with INTEGER*n precision, where n can have any value from 1 to 8. The phase angle is given in degrees.

The subroutine places the output data in the array initially holding the input data, hence the initial data is overwritten.

4 References

None

5 Arguments

X – INTEGER*n

The array from which the subroutine takes the in-phase component of the data to be converted, and into which the magnitude component is placed

Y – INTEGER*n

The array from which the subroutine takes the quadrature component of the data to be converted, and into which the phase angle component, expressed in degrees, is placed

N – INTEGER*4 scalar

Specifies the array length. N must be an integer power of 2, that is, $N = 2^M$, where M is a positive integer.

IFAIL – INTEGER*4 scalar

Unless the subroutine detects an error IFAIL will be set to zero on exit from the subroutine

6 Error Indicators

0 The subroutine worked correctly

1 N was not specified correctly

8 Auxiliary Routines

System routines

SPL_IQMAG_I*n*
SPL_UNWRAP_I*n*
SPL_ATAN2_I*n*

9 Accuracy

Details to be supplied

10 Further Comments

COMMON blocks used – SPL_MACH, SPL_TRIG

11 Keywords

Amplitude to magnitude, in-phase and quadrature to magnitude and phase angle, conversion

7.3.2 IQ_MAG_R*n* **release 2**

1 Purpose

IQ_MAG_R*n* converts signal data from the in-phase and quadrature form to the magnitude and phase angle form, with REAL*n precision, where n can have any value from 3 to 8.

2 Specification

SUBROUTINE IQ_MAG_R*n* (X, Y, N, IFAIL)
REAL*n X(,,N), Y(,,N)
INTEGER*4 N, IFAIL

3 Description

IQ_MAG_R*n* converts data representing signal in-phase and quadrature values to the equivalent magnitude and phase angle form, with REAL*n precision, where n can have any value from 3 to 8. The phase angle is given in radians.

The subroutine places the output data in the array initially holding the input data, hence the initial data is overwritten.

4 References

None

5 Arguments

X – REAL*n

The array from which the subroutine takes the in-phase component of the data to be converted, and into which the magnitude component is placed

Y – REAL*n

The array from which the subroutine takes the quadrature component of the data to be converted, and into which the phase angle component, expressed in radians, is placed

N – INTEGER*4 scalar

Specifies the array length. N must be an integer power of 2, that is, $N = 2^M$, where M is a positive integer.

IFAIL – INTEGER*4 scalar

Unless the subroutine detects an error IFAIL will be set to zero on exit from the subroutine

6 Error Indicators

0 The subroutine worked correctly

1 N was not specified correctly

7 Auxiliary Routines

System routines

SPL_IQMAG_I*n*
SPL_UNWRAP_I*n*
SPL_ATAN2_I*n*

8 Accuracy

Details to be supplied

9 Further Comments

COMMON blocks used – SPL_MACH, SPL_TRIG

10 Keywords

Amplitude to magnitude, in-phase and quadrature to magnitude and phase angle, conversion

7.4.2 MAG_IQ_R*n* **release 2**

1 Purpose

MAG_IQ_R*n* converts signal data from the magnitude and phase angle form to the in-phase and quadrature form, with REAL*n precision, where n can have any value from 3 to 8.

2 Specification

SUBROUTINE MAG_IQ_R*n* (X, Y, N, IFAIL)
REAL*n X(,,N), Y(,,N)
INTEGER*4 N, IFAIL

3 Description

MAG_IQ_R*n* converts data representing signal magnitude and phase angle values to the equivalent in-phase and quadrature form, with REAL*n precision, where n can have any value from 3 to 8. The routine assumes that the phase angle is expressed in degrees.

The subroutine places the output data in the array initially holding the input data, hence the initial data is overwritten.

4 References

None

5 Arguments

X – REAL*n

The array from which the subroutine takes the magnitude component of the data to be converted, and into which the in-phase component is placed

Y – REAL*n

The array from which the subroutine takes phase angle component of the data to be converted, and into which the quadrature component is placed

N – INTEGER*4 scalar

Specifies the array length. N must be an integer power of 2, that is, $N = 2^M$, where M is a positive integer.

IFAIL – INTEGER*4 scalar

Unless the subroutine detects an error IFAIL will be set to zero on exit from the subroutine

6 Error Indicators

0 The subroutine worked correctly

1 N was not specified correctly

7 Auxiliary Routines

System routines

SPL MAGIQ_I*n*

8 Accuracy

Details to be supplied

9 Further Comments

COMMON blocks used – SPL_MACH, SPL_TRIG

10 Keywords

Magnitude to amplitude, magnitude and phase angle to in-phase and quadrature, conversion