NAG Toolbox Chapter Introduction

F16 – Further Linear Algebra Support Routines

Contents

1 Scope of the Chapter 2

2 Background to the Problems 2

3 Recommendations on Choice and Use of Available Functions 2

4 Functionality Index 6

5 References 7

1 Scope of the Chapter

This chapter is concerned with basic linear algebra functions which perform elementary algebraic operations involving scalars, vectors and matrices. Most functions for such operations conform either to the specications of the BLAS (Basic Linear Algebra Subprograms) or to the specifications of the BLAST (Basic Linear Algebra Subprograms Technical) Forum. This chapter includes functions from the BLAST specifications. Most (BLAS) functions for such operations are available in .

2 Background to the Problems

Most of the functions in this chapter meet the specification of Basic Linear Algebra Subprograms Technical (BLAST) Forum (2001).

They are called extensively by functions in other chapters of the NAG Toolbox, especially in the linear algebra chapters. They are intended to be useful building-blocks for users of the Library who are developing their own applications. The functions fall into four main groups (following the definitions introduced by the BLAS):

Level 0: scalar operations;

Level 1: vector operations;

Level 2: matrix-vector operations and matrix operations which includes single matrix operations;

Level 3: matrix-matrix operations.

The terminology reflects the number of operations involved, so for example a Level 2 function involves $O\left(n^2\right)$ operations, for vectors and matrices of order n.

Because of the overlap of functionality with , only a subset of BLAST functions are implemented in this chapter. A full descripion of the

3 Recommendations on Choice and Use of Available Functions

3.1 Naming Scheme

3.1.1 NAG names

Table 1 shows the naming scheme for the functions in this chapter which follows the naming scheme used in .

	Level-0	Level-1	Level-2	Level-3
integer	–	F16D_F	–	–
‘real’	–	F16E_F	–	–
‘real’	–	–	F16R_F	–
‘complex’	–	F16G_F	–	–
‘complex’	–	–	F16U_F	–
‘mixed type’	–	F16J_F	–	–

Table 1

The heading ‘mixed type’ is for functions where a mixture of data types is involved, such as a function that returns the real norm of a complex vector. In future marks of the Library, functions may be included in categories that are currently empty and further categories may be introduced.

3.1.2 BLAS names

Those functions which conform to the specifications of the BLAS may be called either by their NAG names or by their BLAS names.

In many implementations of the NAG Toolbox, references to BLAS names may be linked to an efficient machine-specific implementation of the BLAS, usually provided by the vendor of the machine; Chapter F16 BLAS functions are unlikely to be provide by a vendor. Such implementations are stringently

tested before being used with the NAG Toolbox, to ensure that they correctly meet the specifications of the BLAS, and that they return the desired accuracy. Use of BLAS names is recommended for efficiency.

References to NAG function names (beginning F06- or F16-) are always linked to the code provided in the NAG Toolbox and may be significantly slower (in the case of functions) than the equivalent BLAS function.

The names of the Level-2 and Level-3 BLAS follow a simple scheme (which is similar to that used for LAPACK functions in Chapters F07 and F08). Each name has the structure **XYYZZZ**, where the components have the following meanings:

– the initial letter **X** indicates the data type (real or complex) and precision:

  S real, single precision (in Fortran, `REAL`)

  D real, double precision (in Fortran, `DOUBLE PRECISION`)

  C complex, single precision (in Fortran, `COMPLEX`)

  Z complex, double precision (in Fortran, `COMPLEX*16` or `DOUBLE COMPLEX`)

– the second and third letters **YY** indicate the type of the matrix A (and in some cases its storage scheme):

  GE general

  GB general band

  SY symmetric

  SP symmetric (packed storage)

  SB symmetric band

  HE (complex) Hermitian

  HP (complex) Hermitian (packed storage)

  HB (complex) Hermitian band

  TR triangular

  TP triangular (packed storage)

  TB triangular band

– the remaining 1, 2 or 3 letters **ZZZ** indicate the computation performed:

  MV matrix-vector product

  MM matrix-matrix product

  R rank-1 update

  R2 rank-2 update

  RK rank-k update

  R2K rank-2k update

  SV solve a system of linear equations

  SM solve a system of linear equations with a matrix of right-hand sides

Thus the function nag_blast_daxpby (f16ec) performs a sum of two real, scaled vectors in double precision; the corresponding function for complex scalars and vectors is nag_blast_zaxpby (f16gc).

The names of the Level-1 BLAS mostly follow the same convention for the initial letter (S-, C-, D- or Z-), except for a few involving data of mixed type, where the first two characters are precision-dependent.

3.2 The Level-0 Scalar Functions

The Level-0 functions perform operations on scalars or on vectors or matrices of order 2.

3.3 The Level-1 Vector Functions

The Level-1 functions perform operations either on a single vector or on a pair of vectors.

3.4 The Level-2 Matrix-vector and Matrix Functions

The Level-2 functions perform operations involving either a matrix on its own, or a matrix and one or more vectors.

3.5 The Level-3 Matrix-matrix Functions

The Level-3 functions perform operations involving matrix-matrix products.

3.6 Vector Arguments

Vector arguments (except in the Level-1 Sparse BLAS) are represented by a one-dimensional array, immediately followed by an **increment** argument whose name consists of the three characters INC followed by the name of the array. For example, a vector x is represented by the two arguments **x** and **incx**. The length of the vector, n say, is passed as a separate argument, **n**.

The increment argument is the spacing (stride) in the array between the elements of the vector. For instance, if **incx** $= 2$, then the elements of x are in locations $x(1), x(3), \ldots, x(2n-1)$ of the array **x** and the intermediate locations $x(2), x(4), \ldots, x(2n-2)$ are not referenced.

When **incx** > 0, the vector element x_i is in the array element $\mathbf{x}(1 + (i-1) \times \mathbf{incx})$. When **incx** ≤ 0, the elements are stored in the reverse order so that the vector element x_i is in the array element $\mathbf{x}(1 - (n-i) \times \mathbf{incx})$ and hence, in particular, the element x_n is in $\mathbf{x}(1)$. The declared length of the array **x** in the calling function must be at least $(1 + (\mathbf{n} - 1) \times |\mathbf{incx}|)$.

Negative increments are permitted only for:

Level-1 functions which have more than one vector argument;

Level-2 BLAS functions (but not for other Level-2 functions)

Zero increments are formally permitted for Level-1 functions with more than one argument (in which case the element $\mathbf{x}(1)$ is accessed repeatedly), but their use is strongly discouraged since the effect may be implementation-dependent. There is usually an alternative function in this chapter, with a simplified argument list, to achieve the required purpose. Zero increments are not permitted in the Level-2 BLAS.

3.7 Matrix Arguments and Storage Schemes

In this chapter the following different storage schemes are used for matrices:

– **conventional storage** in a two-dimensional array;

– **packed and RFP storage** for symmetric, Hermitian or triangular matrices;

– **band storage** for band matrices;

– storage for spiked matrices.

These storage schemes are compatible with those used in Chapters F07 and F08. (Different schemes for packed or band storage are used in a few older functions in Chapters F01, F02, F03 and F04.)

Chapter F01 provides some utility functions for conversion between storage schemes.

In the examples, $*$ indicates an array element which need not be set and is not referenced by the functions. The examples illustrate only the relevant leading rows and columns of the arrays; array arguments may of course have additional rows or columns, according to the usual rules for passing array arguments in Fortran.

3.7.1 Conventional storage

Please see Section 3.2.1 in the F07 Chapter Introduction for full details.

3.7.2 Packed storage

Please see Section 3.2.2 in the F07 Chapter Introduction for full details.

3.7.3 Rectangular Full Packed (RFP) storage

Please see Section 3.2.3 in the F07 Chapter Introduction for full details.

3.7.4 Band storage

Please see Section 3.2.4 in the F07 Chapter Introduction for full details.

3.7.5 Unit triangular matrices

Please see Section 3.2.5 in the F07 Chapter Introduction for full details.

3.7.6 Real diagonal elements of complex Hermitian matrices

Please see Section 3.2.6 in the F07 Chapter Introduction for full details.

3.8 Option Arguments

Many of the functions in this chapter have one or more **option arguments**, of type CHARACTER. The descriptions in the function documents refer only to upper-case values (for example **uplo** = 'U' or **uplo** = 'L'); however, in every case, the corresponding lower-case characters may be supplied (with the same meaning). Any other value is illegal.

The following option arguments are used in this chapter:

If **trans** = 'N', operate with the matrix (Not transposed);

if **trans** = 'T', operate with the Transpose of the matrix;

if **trans** = 'C', operate with the Conjugate transpose of the matrix.

If **uplo** = 'U', upper triangle or trapezoid of matrix;

if **uplo** = 'L', lower triangle or trapezoid of matrix.

If **diag** = 'U', unit triangular;

if **diag** = 'N', nonunit triangular.

If **side** = 'L', operate from the left-hand side;

if **side** = 'R', operate from the right-hand side.

If **norm_p** = '1' or 'O', 1-norm of a matrix;

if **norm_p** = 'I', ∞-norm of a matrix;

if **norm_p** = 'F' or 'E', Frobenius or Euclidean norm of a matrix;

if **norm_p** = 'M', maximum absolute value of the elements of a matrix (not strictly a norm).

3.8.1 Matrix norms

The option argument **norm_p** specifies different matrix norms whose definitions are given here for reference (for a general m by n matrix A):

One-norm (**norm_p** = 'O' or '1'):

$$\|A\|_1 = \max_j \sum_{i=1}^{m} |a_{ij}|;$$

Infinity-norm (**norm_p** = 'I'):

$$\|A\|_\infty = \max_i \sum_{j=1}^{n} |a_{ij}|;$$

Frobenius or Euclidean norm (**norm_p** = 'F' or 'E'):

$$\|A\|_F = \left(\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2 \right)^{1/2}.$$

If A is symmetric or Hermitian, $\|A\|_1 = \|A\|_\infty$.

The argument **norm_p** can also be used to specify the maximum absolute value $\max_{i,j} |a_{ij}|$ (if **norm_p** = 'M'), but this is not a norm in the strict mathematical sense.

3.9 Error Handling

Functions in this chapter do not use the usual NAG Toolbox error-handling mechanism, involving the argument IFAIL.

If one of the Level-2 or Level-3 BLAS functions is called with an invalid value of one of its arguments, then an error message is output on the error message unit (see nag_file_set_unit_error (x04aa)), giving the name of the function and the number of the first invalid argument, and execution of the program is terminated. The following values of arguments are invalid:

– any value of the arguments **trans**, **transa**, **transb**, **uplo**, **side** or **diag**, whose meaning is not specified;

– a negative value of any of the arguments **m**, **n**, **k**, **kl** or **ku**;

– too small a value for any of the leading dimension arguments;

– a zero value for the increment arguments **incx** and **incy**.

Zero values for the matrix dimensions **m**, **n** or **k** are considered valid.

The other functions in this chapter do not report any errors in their arguments. Normally, if called, for example, with an unspecified value for one of the option arguments, or with a negative value of one of the problem dimensions **m** or **n**, they simply do nothing and return immediately.

4 Functionality Index

Matrix-vector operations,
complex matrix and vector(s),
compute a norm or the element of largest absolute value,
band matrix f16ub
real matrix and vector(s),
compute a norm or the element of largest absolute value,
band matrix f16rb

Scalar and vector operations,
complex vector(s),
maximum absolute value and location f16js
minimum absolute value and location f16jt
sum of elements f16gl
sum of two scaled vectors f16gc
sum of two scaled vectors preserving input f16gh
integer vector(s),
maximum absolute value and location f16dq

maximum value and location f16dn
minimum absolute value and location f16dr
minimum value and location f16dp
sum of elements f16dl
real vector(s),
dot product of two vectors with optional scaling and accumulation f16ea
maximum absolute value and location f16jq
maximum value and location f16jn
minimum absolute value and location f16jr
minimum value and location f16jp
sum of elements f16el
sum of two scaled vectors f16ec
sum of two scaled vectors preserving input f16eh

5 References

Basic Linear Algebra Subprograms Technical (BLAST) Forum (2001) *Basic Linear Algebra Subprograms Technical (BLAST) Forum Standard* University of Tennessee, Knoxville, Tennessee http://www.netlib.org/blas/blast-forum/blas-report.pdf

Dodson D S and Grimes R G (1982) Remark on Algorithm 539 *ACM Trans. Math. Software* **8** 403–404

Dodson D S, Grimes R G and Lewis J G (1991) Sparse extensions to the Fortran basic linear algebra subprograms *ACM Trans. Math. Software* **17** 253–263

Dongarra J J, Du Croz J J, Duff I S and Hammarling S (1990) A set of Level 3 basic linear algebra subprograms *ACM Trans. Math. Software* **16** 1–28

Dongarra J J, Du Croz J J, Hammarling S and Hanson R J (1988) An extended set of FORTRAN basic linear algebra subprograms *ACM Trans. Math. Software* **14** 1–32

Dongarra J J, Moler C B, Bunch J R and Stewart G W (1979) *LINPACK Users' Guide* SIAM, Philadelphia

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

Lawson C L, Hanson R J, Kincaid D R and Krogh F T (1979) Basic linear algebra supbrograms for Fortran usage *ACM Trans. Math. Software* **5** 308–325