Annex A

Appendix

This appendix contains overall notation, definitions, and implementation details for the chapters of the BLAS Technical Forum Standard.

A.1 Vector Norms

There are a variety of ways to define the norm of a vector, in particular for vectors of complex numbers, several of which have been used in the existing Level 1 BLAS and in various LAPACK auxiliary routines. Our definitions include all of these in a systematic way.

Data Type	Name	Notation	Definition
Real	one-norm	$\|x\|_1$	$\sum_i \|x_i\|$
	two-norm	$\|x\|_2$	$\sqrt{\sum_i x_i^2}$
	infinity-norm	$\|x\|_\infty$	$\max_i \|x_i\|$
Complex	one-norm	$\|x\|_1$	$\sum_i \|x_i\|$ $= \sum_i (Re(x_i)^2 + Im(x_i)^2)^{1/2}$
	real one-norm	$\|x\|_{1R}$	$\sum_i (\|Re(x_i)\| + \|Im(x_i)\|)$
	two-norm	$\|x\|_2$	$\sqrt{\sum_i \|x_i\|^2}$ $= (\sum_i (Re(x_i)^2 + Im(x_i)^2))^{1/2}$
	infinity-norm	$\|x\|_\infty$	$\max_i \|x_i\|$ $= \max_i (Re(x_i)^2 + Im(x_i)^2)^{1/2}$
	real infinity-norm	$\|x\|_{\infty R}$	$\max_i (\|Re(x_i)\| + \|Im(x_i)\|)$

Table A.1: Vector Norms

Rationale. The reason for the two extra norms of complex vectors, the real one-norm and real infinity-norm, is to avoid the expense of up to n square roots, where n is the length of the vector x. The two-norm only requires one square root, so a real version is not needed. The infinity norm only requires one square root in principle, but this would require tests and branches, making it more complicated and slower than the real infinity-norm. When x is real, the one-norm and real one-norm are identical, as are the infinity-norm and real infinity-norm. We note that the Level 1 BLAS routine ICAMAX, which finds the largest entry of a complex vector, finds the largest value of $|Re(x_i)| + |Im(x_i)|$. (*End of rationale.*)

Computing the two-norm or Frobenius-norm of a vector is equivalent. However, this is not the case for computing matrix norms. For consistency of notation between vector and matrix norms, both norms are available.

A.2 Matrix Norms

Analogously to vector norms as discussed in Section A.1, there are a variety of ways to define the norm of a matrix, in particular for matrices of complex numbers. Our definitions include all of these in a systematic way.

Data Type	Name	Notation	Definition
Real	one-norm	$\|A\|_1$	$\max_j \sum_i \|a_{ij}\|$
	Frobenius-norm	$\|A\|_F$	$\sqrt{\sum_i \sum_j a_{ij}^2}$
	infinity-norm	$\|A\|_\infty$	$\max_i \sum_j \|a_{ij}\|$
	max-norm	$\|A\|_{\max}$	$\max_i \max_j \|a_{ij}\|$
Complex	one-norm	$\|A\|_1$	$\max_j \sum_i \|a_{ij}\|$
			$= \max_j \sum_i (Re(a_{ij})^2 + Im(a_{ij})^2)^{1/2}$
	real one-norm	$\|A\|_{1R}$	$\max_j \sum_i (\|Re(a_{ij})\| + \|Im(a_{ij})\|)$
	Frobenius-norm	$\|A\|_F$	$\sqrt{\sum_i \sum_j \|a_{ij}\|^2}$
			$= (\sum_i \sum_j (Re(a_{ij})^2 + Im(a_{ij})^2))^{1/2}$
	infinity-norm	$\|A\|_\infty$	$\max_i \sum_j \|a_{ij}\|$
			$= \max_i \sum_j (Re(a_{ij})^2 + Im(a_{ij})^2)^{1/2}$
	real infinity-norm	$\|A\|_{\infty R}$	$\max_i \sum_j (\|Re(a_{ij})\| + \|Im(a_{ij})\|)$
	max-norm	$\|A\|_{\max}$	$\max_i \max_j \|a_{ij}\|$
			$= \max_i \max_j (Re(a_{ij})^2 + Im(a_{ij})^2)^{1/2}$
	real max-norm	$\|A\|_{\max R}$	$= \max_i \max_j (\|Re(a_{ij})\| + \|Im(a_{ij})\|)$

Table A.2: Matrix Norms

In contrast to computing vector norms, computing the two-norm and Frobenius-norm of a matrix are not equivalent. If the user asks for the two-norm of a matrix, where the matrix is 2-by-2 or larger, an error flag is raised. The one exception occurs when the matrix is a single column or a single row. In this case, the two-norm is requested and the Frobenius-norm is returned.

A.3 Operator Arguments

The following table lists the operator arguments and their associated named constants. For complete details of the meanings of the operator prec, refer to section 4.3.1.
Example: Consider the matrix-vector products $x = Ax$, $x = A^T x$ and $x = A^H x$. It is convenient to use the `trans` operator and define $op(A)$ as being A, A^T or A^H depending on the value of the `trans` operator argument. Again, the specification of the type and the valid values such an operator should have will be defined in the language-dependent section and may vary from one language binding to another.

It is worthwhile noticing that in some rare cases, the meaning of the `trans` operator argument is extended to a function of the matrix to which it applies. Consider for example the symmetric

operator argument	named constant	meaning
norm	blas_one_norm	1-norm
	blas_real_one_norm	real 1-norm
	blas_two_norm	2-norm
	blas_frobenius_norm	Frobenius-norm
	blas_inf_norm	infinity-norm
	blas_real_inf_norm	real infinity-norm
	blas_max_norm	max-norm
	blas_real_max_norm	real max-norm
sort	blas_increasing_order	sort in increasing order
	blas_decreasing_order	sort in decreasing order
side	blas_left_side	operate on the left-hand side
	blas_right_side	operate on the right-hand side
uplo	blas_upper	reference upper triangle only
	blas_lower	reference lower triangle only
trans$x$	blas_no_trans	operate with x
	blas_trans	operate with x^T
	blas_conj_trans	operate with x^H
conj	blas_conj	operate with $\bar{x}$
	blas_no_conj	operate with x
diag	blas_non_unit_diag	non-unit triangular
	blas_unit_diag	unit triangular
jrot	blas_jrot_inner	inner rotation $c \geq \frac{1}{\sqrt{2}}$
	blas_jrot_outer	outer rotation $0 \leq c \leq \frac{1}{\sqrt{2}}$
	blas_jrot_sorted	sorted rotation $abs(a) \geq abs(b)$
order	blas_colmajor	assume column-major ordering
	blas_rowmajor	assume row-major ordering
index_base	blas_zero_base	assumes zero-based indexing
	blas_one_base	assumes one-based indexing
prec	blas_prec_single	internal computation performed in single precision
	blas_prec_double	internal computation performed in double precision
	blas_prec_indigenous	internal computation performed in the widest hardware-supported format available
	blas_prec_extra	internal computation performed in format wider than 80-bits

Table A.3: Operator Arguments

rank-k update operations, $C \leftarrow C + AA^T$ and $C \leftarrow C + A^TA$ where C is a symmetric matrix. The value of the trans operator refers to the product AA^T. It follows that these operations can be specified by $C \leftarrow C + op(AA^T)$ where $op(AA^T)$ is AA^T or A^TA depending on the input value of the trans argument.

All possible values of the operator argument trans are not always meaningful. For example, in the symmetric rank-k update operations defined above, when the matrix C is complex symmetric, the only valid values of $op(AA^T)$ are AA^T or A^TA. Similarly, when the matrix C is complex Hermitian, the only valid values of $op(AA^H)$ are AA^H or A^HA. Such restrictions are detailed for each dense and banded BLAS function to which they apply.

Some BLAS routines have more than one trans operator argument because such an argument is needed for each matrix to which it applies. For example, a general matrix-multiply operation can be specified as $C \leftarrow op(A)op(B)$ where A, B and C are general matrices. A trans argument is needed for each of the input matrices A and B; by convention we denote those formal arguments transA and transB.

> *Rationale.* As mentioned above, section (1.4) does not specify how the objects manipulated by the BLAS routines are stored. This important aspect of the interface specification is deferred to the language-dependent specification sections. In particular, the operator arguments **do not** indicate whether only half or all entries of triangular, symmetric and Hermitian matrices are stored, or even how these entries are stored. The intent is to provide each language binding with the opportunity to choose the appropriate data structures for each object. Note that a given language binding specification may provide multiple functions performing the same operation on operands stored differently. For example, triangular matrices may be stored within conventional two-dimensional arrays or in packed storage, where the triangle may be packed by rows or columns. Consequently, a BLAS routine specified in the functionality tables may induce multiple functions in a particular language binding, say for instance, to provide the user with the same operation on objects that are stored differently. (*End of rationale.*)

It follows that, in general, a mathematical operation involving a matrix A, where A could be general or banded, triangular, symmetric or Hermitian, induces the language-independent specification of multiple routines. However, this language-independent section ignores the fact that a given language binding may choose to provide multiple storage schemes for some specific classes of matrices, such as triangular matrices.

A.4 Fortran 95 Modules

Several Fortran 95 modules are provided, allowing for the flexible inclusion of only select portions of the document. The modules `blas_dense`, `blas_sparse`, and `blas_extended`, are provided for Chapters 2, 3, and 4, respectively.

```
http://www.netlib.org/blas/blast-forum/blas_dense.f90
http://www.netlib.org/blas/blast-forum/blas_sparse.f90
http://www.netlib.org/blas/blast-forum/blas_extended.f90
```

Each of these modules in turn contains a `USE` statement to include the module of operator arguments (`blas_operator_arguments` for Chapters 2 and 4, and `blas_sparse_namedconstants` for Chapter 3), and the respective module(s) of explicit interfaces for that chapter.

For Chapters 2 and 4, one derived type is specified for each category of operator arguments (such as trans) and some parameters are defined of this type (for the different settings). For consistency, the suffix `_type` is used to name all of the derived types. This suffix is needed in some cases to differentiate between the type and one of the parameters (for example, `blas_trans_type` is a type and `blas_trans` is a parameter of this type). The Sparse BLAS chapter represents its operator arguments and a list of matrix properties (see section 3.5.1) as named constants.

Advice to implementors. For Chapter 2, all the entities (derived types, named constants and BLAS procedures) must be accessible to the user via the module `blas_dense`.

There are many ways to create this module. However the following three conditions **MUST** be adhered to:

- all entities can be accessed by the module
- the generic names must be the same as in the Fortran 95 bindings
- the specific name must be standard. The standard that we recommend is "suffix _d, _z, _s and _c" for double precision, double complex, real and complex.

For example the Fortran 95 bindings gives the generic name `gemm`. This is a generic procedure for the following 12 specific procedures:

gemm_d	corresponds to BLAS_DGEMM (legacy DGEMM)
gemm_z	corresponds to BLAS_ZGEMM (legacy ZGEMM)
gemm_s	corresponds to BLAS_SGEMM (legacy SGEMM)
gemm_c	corresponds to BLAS_CGEMM (legacy CGEMM)
gemv_d	corresponds to BLAS_DGEMV (legacy DGEMV)
gemv_z	corresponds to BLAS_ZGEMV (legacy ZGEMV)
gemv_s	corresponds to BLAS_SGEMV (legacy SGEMV)
gemv_c	corresponds to BLAS_CGEMV (legacy CGEMV)
ger_d	corresponds to BLAS_DGER (legacy DGER)
ger_z	corresponds to BLAS_ZGER (legacy SGER)
ger_s	corresponds to BLAS_SGER (legacy ZGERU, ZGERC)
ger_c	corresponds to BLAS_CGER (legacy CGERU, CGERC)

A specific procedure could be an external procedure or a module procedure.

One approach for creating the module `blas_dense` is to:

- create one file for each procedure
- create the interface blocks for the generic names using one or more modules
- create the module `blas_dense` from the modules in the last step and other modules such as `blas_operator_arguments`

Assuming we are using external procedures, the following files could be used as templates to create the module `blas_dense`. The interface blocks are grouped according to the grouping in section 2.8.1. The files are:

- `http://www.netlib.org/blas/blast-forum/blas_operator_arguments.f90`
  file containing the module blas_operator_arguments
- `http://www.netlib.org/blas/blast-forum/blas_precision.f90`
  file containing the module used to specify the precision (not visible to the user)
- `http://www.netlib.org/blas/blast-forum/blas_dense_red_op.f90`
  file containing the interface blocks for the reduction operations (section 2.8.2)
- `http://www.netlib.org/blas/blast-forum/blas_dense_gen_trans.f90`
  file containing the interface blocks for the generate transformations procedures (section 2.8.3)

- `http://www.netlib.org/blas/blast-forum/blas_dense_vec_op.f90`
  file containing the interface blocks for the vector operations (section 2.8.4)
- `http://www.netlib.org/blas/blast-forum/blas_dense_vec_mov.f90`
  file containing the interface blocks for the data movement with vectors (section 2.8.5)
- `http://www.netlib.org/blas/blast-forum/blas_dense_mat_vec_op.f90`
  file containing the interface blocks for the matrix_vector operations (section 2.8.6)
- `http://www.netlib.org/blas/blast-forum/blas_dense_mat_op.f90`
  file containing the interface blocks for the matrix operations (section 2.8.7)
- `http://www.netlib.org/blas/blast-forum/blas_dense_mat_mat_op.f90`
  file containing the interface blocks for the matrix_matrix operations (section 2.8.8)
- `http://www.netlib.org/blas/blast-forum/blas_dense_mat_mov.f90`
  file containing the interface blocks for the data movement with matrices (section 2.8.9)
- `http://www.netlib.org/blas/blast-forum/blas_dense_fpinfo.f90`
  file containing the interface blocks for the environmental enquiry (section 2.8.10)
- `http://www.netlib.org/blas/blast-forum/blas_dense.f90`
  file containing the module `blas_dense` that imports the information from all other modules and makes them available.

The specifications for all specific procedures MUST be as they appear in the above files. The only change is the way that the precision is specified. (*End of advice to implementors.*)

A.5 Fortran 77 Include File

One Fortran 77 include file is provided, `blas_namedconstants.h`. This include file contains the values of all named constants, and applies to Chapters 2, 3, and 4.

`http://www.netlib.org/blas/blast-forum/blas_namedconstants.h`

Operator arguments norm, sort, side, uplo, trans, conj, diag, jrot, index_base, and prec are represented in the Fortran 77 interface as INTEGERs. These operator arguments assume the named constant values as defined in section A.3. The Sparse BLAS chapter defines a list of matrix properties (see section 3.5.1) that must also be defined.

Advice to implementors. This specification is a deviation from the Legacy BLAS, where these operator arguments were defined as `CHARACTER*1`. (*End of advice to implementors.*)

A.6 C Include Files

Several C include files are provided, allowing for the flexible inclusion of only select portions of the document. The file `blas.h` contains the enumerated types and all prototypes for Chapters 2, 3, and 4. The files `blas_dense.h`, `blas_sparse.h`, and `blas_extended.h`, include the values of the operator arguments (enumerated types) and the function prototypes for the respective chapter.

```
http://www.netlib.org/blas/blast-forum/blas.h
http://www.netlib.org/blas/blast-forum/blas_dense.h
http://www.netlib.org/blas/blast-forum/blas_sparse.h
http://www.netlib.org/blas/blast-forum/blas_extended.h
```

The file `blas_enum.h` contains the values of all enumerated types, applying to all chapters. The files `blas_dense_proto.h`, `blas_sparse_proto.h`, and `blas_extended_proto.h`, contain the respective function prototypes for Chapters 2, 3, and 4.

```
http://www.netlib.org/blas/blast-forum/blas_enum.h
http://www.netlib.org/blas/blast-forum/blas_dense_proto.h
http://www.netlib.org/blas/blast-forum/blas_sparse_proto.h
http://www.netlib.org/blas/blast-forum/blas_extended_proto.h
```

All operator arguments are handled by enumerated types in the C interface. This allows for tighter error checking, and provides less opportunity for user error. In addition to the operator arguments of norm, sort, side, uplo, trans, conj, diag, jrot, index_base, and prec, this interface adds another such argument to all routines involving two dimensional arrays, order. order designates if the array elements are stored in row-major or column-major ordering. Refer to section 2.6.6 for further details. The Sparse BLAS chapter defines a list of matrix properties (see section 3.5.1) that must also be defined.