Parallel-SLICOT Implementation and Documentation Standards [1]

Ignacio Blanquer[2], David Guerrero[2], Vicente Hernández[2], Enrique S. Quintana-Ortí[3], Pedro A. Ruiz[2]

September 1998

[1]This document presents research results of the European Community BRITE-EURAM III Thematic Networks Programme NICONET (contract number BRRT-CT97-5040) and is distributed by the Working Group on Software WGS. *WGS secretariat*: Mrs. Ida Tassens, ESAT - Katholieke Universiteit Leuven, K. Mercierlaan 94, 3001-Leuven-Heverlee, BELGIUM. This report is also available by anonymous ftp from *wgs.esat.kuleuven.ac.be* in the directory *pub/WGS/REPORTS/SLWN1998-1.ps.Z*
[2]Departamento de Sistemas Informáticos y Computación (DSIC). Universidad Politécnica de Valencia (UPV). Valencia. Spain. {*iblanque, dguerrer, vhernand, pruiz*}*@dsic.upv.es.*
[3]Departamento de Informática, Universitat Jaume I. Castellón. Spain. *quintana@inf.uji.es.*

Contents

1 Introduction

1.1 Introduction to the Parallel Standards

This paper presents the **P-SLICOT** (Parallel Subroutine Library in Control and Systems Theory) **Implementation and Documentation Standards**. Here we propose some useful guidelines for those who want to contribute to the parallel version of SLICOT. The main goal of these rules is to facilitate the work of obtaining a portable, reliable, and easily maintainable code.

For those who have read the *sequential* **"SLICOT Implementation and Documentation Standards"** [13], it must be noted that the parallel standards are strongly based on that document. More precisely, here we present the same guidelines that appear in that document, with the addition of some points applicable to parallel versions.

The development of efficient and portable sequential numerical algorithms relies on the use of standard libraries. Optimised versions of LAPACK and BLAS are currently available for most computers.

This should be the same criterium for parallel algorithms. Designing parallel algorithms from scratch may lead to poor performance when porting algorithms from shared-memory to distributed-memory computers or vice-versa, or simply when using different operating systems.

Several libraries have been developed with parallel implementations of numerical computing kernels, like PLAPACK, ScaLAPACK, PBLAS, ... Currently, the numerical computing community is putting most of the effort in ScaLAPACK and PBLAS, issuing new and more complete versions regularly. On the other hand, the performance that ScaLAPACK and PBLAS deliver is higher than their competitors, and it surely will improve in new versions (e.g. PBLAS 2.0 allows to determine subblocks without alignment restrictions, which will reduce communication overhead; PBLAS 2.0 is in α-version and will be incorporated in ScaLAPACK 1.7).

However, ScaLAPACK lacks several routines. Not all the functionality of LAPACK has already been transferred to ScaLAPACK.

ScaLAPACK 1.6 contains drivers for Linear System Solvers, Linear Least Squares Problems, Symmetric eigenvalues/vectors and Singular Values. It has expert drivers for linear equations and Symmetric Eigenvalues problems. Schur factorizations and non symmetric eigenvalues have already been developed and will be included soon.

Thus, these standards are influenced by the use of PBLAS, ScaLAPACK, and the BLACS libraries. These libraries are the base for computation and communication in P-SLICOT. It may be useful to read the documentation for these libraries ([2] [3] [5]) to find more information about parallel data structures or communications. P-SLICOT has been oriented to the use of PBLAS and ScaLAPACK in the same way as the sequential SLICOT uses BLAS and LAPACK. However, it must be noted that under some circumstances it may be useful to call directly BLAS or LAPACK routines from a P-SLICOT routine, rather than the PBLAS or ScaLAPACK parallel versions in order to achieve better performance.

Following this similarity between LAPACK – ScaLAPACK and SLICOT – P-SLICOT, we propose to prepend the character 'P' at routines of P-SLICOT in order to differentiate the parallel from the sequential routines, in the same way as ScaLAPACK does (however, it conflicts with standard FORTRAN77, which has the limitation of 6 characters for subroutine names).

1.2 Parallel Computation Specific Parameters

We have chosen as reference for the SLICOT parallel programming model the ScaLAPACK standard. Due to the nature of the parallel programming, new considerations must be chosen in the programming standards. Before commenting these new aspects, we first give a brief description of the parallel structure of the routines.

On a distributed memory computer, the application programmer is responsible for decomposing the data over the processes of the computer. There are different schemes for distributing data on a parallel distributed memory computer, such as row-wise, block-row wise, column-wise, row-cyclic, ... In order to be effective, the block-cyclic decomposition has been chosen, as in general, this scheme gives the best results. Note that block-cyclic includes all block, row or column schemes, after selecting the appropriate parameters. The block cyclic scheme allows the routines to achieve scalability, present well balanced computations and minimize synchronization costs. The set of processes is mapped to a logical grid, where each process is naturally identified by its coordinates in a $P \times Q$ grid.

$$
\begin{array}{cc|cc|cc|cc|c}
a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} & a_{18} & a_{19} \\
a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} & a_{28} & a_{29} \\
\hline
a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} & a_{38} & a_{39} \\
a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} & a_{48} & a_{49} \\
\hline
a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} & a_{58} & a_{59} \\
a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} & a_{68} & a_{69} \\
\hline
a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} & a_{77} & a_{78} & a_{79} \\
a_{81} & a_{82} & a_{83} & a_{84} & a_{85} & a_{86} & a_{87} & a_{88} & a_{89} \\
\hline
a_{91} & a_{92} & a_{93} & a_{94} & a_{95} & a_{96} & a_{97} & a_{98} & a_{99}
\end{array}
$$

9×9 matrix partitioned in 2×2 blocks

Figure 1: Matrix partitioning

Assuming a two-dimensional block cyclic data distribution, an M by N matrix is first decomposed into `MB` by `NB` blocks starting at its upper left corner. These blocks are then uniformly distributed across the process grid. Thus each process owns a collection of blocks, which are locally and contiguously stored in a two dimensional “column major” array. We present in Fig. 1 the partitioning of a 9×9 matrix into 2×2 blocks. Then in Fig. 2 we show how these 2×2 blocks are mapped onto a 2×3 process grid, i.e., $M = N = 9$ and `MB=NB=2`. The local entries of each matrix column are contiguously stored in the processes' memories.

It follows that a general M by N distributed matrix is defined by its dimensions, the size of the elementary `MB` by `NB` block used for its decomposition and the coordinates of the process having in its local memory the first matrix entry $(RSRC, CSRC)$. Finally, a local leading dimension LLD, is associated with the local memory address pointing to the data structure used for the local storage of this distributed matrix. In Fig. 2, we choose for illustration purposes $RSRC = CSRC = 0$. In addition, the local arrays in process row 0 must have a leading dimension LLD, greater than or equal to 5, and greater than or equal to 4 in the process row 1.

1.2.1 The block size

Determining optimal, or near optimal block sizes for different environments is a difficult task because it depends on many factors such as:

- The machine architecture.
- Performance of the different BLAS levels.
- The latency and bandwidth of message passing.
- The number of processors available.

	0				1			2	
0	a_{11}	a_{12}	a_{17}	a_{18}	a_{13}	a_{14}	a_{19}	a_{15}	a_{16}
	a_{21}	a_{22}	a_{27}	a_{28}	a_{23}	a_{24}	a_{29}	a_{25}	a_{26}
	a_{51}	a_{52}	a_{57}	a_{58}	a_{53}	a_{54}	a_{59}	a_{55}	a_{56}
	a_{61}	a_{62}	a_{67}	a_{68}	a_{63}	a_{64}	a_{69}	a_{65}	a_{66}
	a_{91}	a_{92}	a_{97}	a_{98}	a_{93}	a_{94}	a_{99}	a_{95}	a_{96}
1	a_{31}	a_{32}	a_{37}	a_{38}	a_{33}	a_{34}	a_{39}	a_{35}	a_{36}
	a_{41}	a_{42}	a_{47}	a_{48}	a_{43}	a_{44}	a_{49}	a_{45}	a_{46}
	a_{71}	a_{72}	a_{77}	a_{78}	a_{73}	a_{74}	a_{79}	a_{75}	a_{76}
	a_{81}	a_{82}	a_{87}	a_{88}	a_{83}	a_{84}	a_{89}	a_{85}	a_{86}

2×3 process grid point of view

Figure 2: Mapping of matrix onto process grid (`NPROW=2, NPCOL=3`)

- The dimensions of the process grid.
- The dimension of the problem, and so on.

Moreover, in several parallel routines, the work distribution becomes uneven as the computation progresses. A larger block size results then, in greater load imbalance, though it generally reduces the frequency of communication between processes. There is, therefore, a trade-off between load imbalance and communication cost, which can be controlled by varying the block size.

However, since the computation cost ultimately dominates, the influence of the block size on the overall communication start-up cost and loop and index computation overhead decreases very rapidly with problem size for a given grid of processes. Even with the imbalance routines, the computation of the PSLICOT routines should be performed in a blocked fashion using Level 3 BLAS, as it is done in SLICOT. The computational blocking factor must be chosen to be the same as the distribution block size. Therefore, smaller distribution block sizes increase the loop and index computation overhead.

Consequently, if the previous rule is used for selecting the block size, the performance of the ScaLAPACK library is not very sensitive to the block size, as long as the extreme cases are avoided. A very small block size leads to BLAS 2 operations and poorer performance. A very large block size leads to computational imbalance.

However, the chosen block size impacts the amount of workspace needed on every process. This amount of workspace is typically large enough to contain a block of columns or a block of rows of the distributed matrix. Therefore, the larger the block size, the greater the necessary workspace, i.e. the smaller the largest solvable problem on a given grid of processes. For Level 3 BLAS block-partitioned algorithms, the smallest possible block operands are of size `MB`$\times$`NB` (which is the block dimensions). Therefore, it is good practice to choose the block size to be the problem size for which the PBLAS matrix multiply PGEMM routine achieves the best performance. Experimentally, good block sizes range from 20 to 50.

1.2.2 The grid structure

The grid size and shape does really affect performance more than the block size. Determining automatically the best shape and size for any problem and architecture is not possible. The best choice depends on the number of available processors, the topology of the network and the problem itself. For example, in the case of one link networks, it is more suitable to use a one-dimension processor array, which although it penalises performance, the communication overhead is reduced.

As a guideline for the users, it is recommended to indicate the topology that fits best to each parallel algorithm. Although it could give better performance when using other topologies in other machines, it could be a good starting point to ease the implementing process.

2 Software Implementation Standards

2.1 General Rules

2.1.1 Subprograms must be written in standard FORTRAN77 as specified in ANSI X3.9-1978, ISO 1539-1980(E) with a number of additional restrictions (see Section 2.3). If possible, they should be verified by passing them through a FORTRAN77 verifier (e.g. [10]). The only extensions to the FORTRAN77 standard are the use of a double precision complex data type (COMPLEX∗16) in some routines (there is no provision for this data type in the standard) and the prepending of the character 'P' to the names of the parallel routines (the standard restricts the length of subroutine names up to 6 characters, which is exceeded by these names).

2.1.2 Subprograms must be independent of the problem size and should be capable of being called by a program which is designed to handle problems of different sizes.

2.1.3 In general subprograms must not carry out internally any input or output of data to external devices, such as terminals, printers, discfiles, etc. Exceptions to this rule are: error routines (they should write on a specified file) and I/O routines meant for interactive usage.

2.1.4 Every effort must be made to avoid program failure within subprograms due to bad data, etc. The preferred response to failure conditions is an orderly return of control to the calling program together with the setting of an appropriate well defined error flag.

2.2 User Interface Standards

2.2.1 All input data required by the subprogram and all output data to be returned to the calling program must be transferred via the argument list. No data essential to the operation of the subprogram may be transferred via COMMON statements. Specific constants, such as characteristics of the computational environment should be provided for via an appropriate function or subroutine call within the routine.

2.2.2 The order of the parameters in the argument list in every user-callable routine must be as follows:

- mode parameters;
- user-supplied functions;
- problem dimensions;
- scalar arguments (except for leading dimensions) defining the input data (some of them may be overwritten by results);
- array arguments (directly followed by their leading first dimension and second dimension if applicable or by their ScaLAPACK description vector in the case of distributed matrices) defining the input data (some of them may be overwritten by results);

Note: An array descriptor (description vector) is associated with each global array. This array stores the information required to establish the mapping between each global array entry and its corresponding process and memory location. The notations x_ used in the entries of the array descriptor denote the attributes of a global array. For example, M_ denotes the number of rows and M_A specifically denotes the number of rows in global matrix A. These descriptors assume different storage for the global data. The length of the array descriptor is specified by DLEN_ and varies according to the descriptor type DTYPE_. For example, the array descriptor for dense matrices has the following structure:

DESC_()	Symbolic Name	Scope	Definition
1	DTYPE_A	(global)	Descriptor type DTYPE_A=1 for dense matrices
2	CTXT_A	(global)	The BLACS context handle, indicating the BLACS process grid A is distributed over. The context itself is global, but the handle (the integer value) may vary.
3	M_A	(global)	The number of rows in the global array A.
4	N_A	(global)	The number of columns in the global array A.
5	MB_A	(global)	The blocking factor used to distribute the rows of the array.
6	NB_A	(global)	The blocking factor used to distribute the columns of the array.
7	RSRC_A	(global)	The process row over which the first row of the array A is distributed.
8	CSRC_A	(global)	The process column over which the first column of the array A is distributed.
9	LLD_A	(local)	The leading dimension of the local array. LLD_A $\geq$ MAX(1,LOC_r(M_A)).

- other scalar arguments returning results;
- other array arguments returning results (directly followed by their leading first dimension and second dimension if applicable or by their ScaLAPACK description vector in the case of distributed matrices);
- arguments defining tolerances;
- work arrays (and associated array dimensions);
- warning indicator IWARN;
- error indicator INFO.

Note: input parameters may also be output parameters.

2.2.3 There should be no parameter in the argument list which is a function of other arguments.

2.2.4 Mode parameters, problem dimensions, leading dimensions, description vectors, and tolerances must not be overwritten by output data.

2.2.5 Mode parameters are used to specify which option of a subprogram is required by the calling program. They should be usually of type CHARACTER∗1.

The values of mode parameters should be meaningful letters, e.g., 'C' or 'c' if a routine for Lyapunov equations should solve the continuous-time equation and 'D' or 'd' if the discrete-time equation is to be solved. Capitals and the corresponding small letters must have the same meaning.

Where appropriate, mode parameters could also be of type LOGICAL or INTEGER.

2.2.6 Problem dimensions equal to zero should be allowed, where appropriate.

2.2.7 All one-dimensional arrays (unless of fixed size) must be declared either with assumed-size dimension (∗) or with adjustable dimension. The assumed-size dimension (∗) must be used if the actual dimension is allowed to be zero (empty actual array) or if the dimension is not a simple integer expression in terms of parameters in the argument list. Similar remarks also apply to the last dimension of two-dimensional or three-dimensional arrays (see below).

Each two-dimensional array must have its own leading dimension, to be passed as a separate integer parameter, directly following the array-name in the argument list. For this integer parameter a name of the form LD<array-name> must be used.

For example, the two-dimensional array A, which is supposed to contain an n-by-n matrix ($n \geq 1$), is declared as:

```
SUBROUTINE        XXXXXX (N, A, LDA, ...)
INTEGER           N, LDA
DOUBLE PRECISION  A(LDA, N)
```

Each three-dimensional array must have its own leading dimensions, to be passed as separate integer parameters, directly following the array-name in the argument list. For these integer parameters names of the form LD<array-name>1 and LD<array-name>2 must be used.

E.g., the three-dimensional array B of which the last dimension is at least max(M,N) is declared as:

```
SUBROUTINE        XXXXXX (N, M, B, LDB1, LDB2, ...)
INTEGER           N, M, LDB1, LDB2
DOUBLE PRECISION  B(LDB1, LDB2, *)
```

Each distributed submatrix (local two-dimensional array) must be specified using the global indexes over the matrix to which it belongs and a description vector (ScaLAPACK array descriptor) with information about distribution and local leading dimension. These parameters are passed to the subroutine directly following the array-name in the argument list. Names of the form I<array-name>,J<array-name> and DESC<array-name> must be used.

E.g., the submatrix of size m-by-n of a matrix C distributed according to a ScaLAPACK descriptor is declared as:

```
SUBROUTINE        PXXXXXX (M, N, C, IC, JC, DESCC, ...)
INTEGER           M, N, IC, JC, DESCC( * )
DOUBLE PRECISION  C( * )
```

2.2.8 Arrays with more than three dimensions must not be used and three-dimensional arrays should be avoided wherever possible.

2.2.9 Tolerances are used to control numerical error. If more than one tolerance is required, then the use of the names TOL1, TOL2, TOL3, ... is recommended. Otherwise, the name TOL should be used. Where appropriate, other names could be used, e.g., RCOND for the minimal value of the reciprocal condition number, below which a matrix is considered to be singular.

All tolerances must be reset internally by the routine if the accuracy requirements specified by the user cannot be achieved, e.g., if TOL < 0.0 on entry, then the tolerance might be taken as EPS (machine precision) instead.

2.2.10 Only one work array of each type may occur in the argument list of every user-callable routine, where the following names must be used:

IWORK for integer workspace;

SWORK for single precision workspace;

DWORK for double precision workspace;

CWORK for complex single precision workspace;

ZWORK for complex double precision workspace;

BWORK for logical workspace.

Work arrays must appear in the above order in the argument list of the routine.

If several work arrays of the same type are required in the body of a subprogram, then the single work array may be partitioned via a call to a lower-level routine.

Workspace is provided in either of the following forms:

Fixed size workspace, which is applied when the size of the workspace is a simple function of the problem dimensions and the mode parameters.

Variable size workspace, which is applied when the size of the workspace is a complicated function of the problem dimensions and the mode parameters or when the performance of the routine can be improved by increasing the size of workspace. Mostly, the latter is the case when the routine is an implementation of a block algorithm or when it calls a blocked routine.

In the argument list the work array xWORK (where x is one of the capitals I, S, D, C, Z, B) must be followed by an input integer parameter LxWORK, which contains the size of the workspace available. In case of a blocked routine the optimal value of LxWORK is a function of the problem dimensions, the mode parameters, and the optimal block size NB, returned by the LAPACK routine ILAENV. For further information, see [1] Sections 5.2 and 6.2. In case of a parallel routine the optimal LxWORK is usually a function of the local number of rows and columns of the submatrices involved in computation (so it may be different among the various processes).

On exit, if possible, xWORK(1) should contain the optimal size of workspace.

See also 3.2.1 for examples.

2.2.11 The optional warning indicator IWARN is an integer variable used by the routine to indicate that either the normal situation (IWARN $= 0$) or an exceptional situation (IWARN $\neq 0$) occurs. Unless the routine generates a warning, IWARN must contain 0 on exit. Other values of IWARN may be used to indicate that a non-fatal "error" has occurred (that is, one which does not effect an immediate return to the calling program), e.g., lowering the rank of a matrix. Note that any results produced by the routine must still be correct even if IWARN $\neq 0$ on exit.

2.2.12 The error indicator INFO must be included as the final parameter in the argument list of every user-callable routine. Unless the routine detects an error, INFO must contain 0 on exit. Other values of INFO may be used to indicate that a fatal error has occurred (that is, one which effects an immediate return to the calling program), e.g., violation of the conditions of an input parameter or attempted division by zero.

If a subroutine detects an error in the i-th parameter of the argument list before the actual computation starts and this parameter is not a descriptor, it must return INFO $= -i$. If the error is in the j-th entry of a descriptor array which is the i-th argument, then the routine must return INFO $= -(100 * i + j)$. Otherwise, if an error is detected in the course of the computation, INFO must contain a positive value (see also 3.2.1).

2.2.13 All input data must be unaltered on return from the subprogram unless this is unlogical due to the nature of the parameter or impractical due to excessive storage requirements.

A possible change of the contents of input parameters must be emphasized in the documentation and in-line comments of the subprogram.

2.3 Programming Standards

2.3.1 Subprograms should be well structured. The structure should essentially consist of a subdivision into linearly ordered modules, where each module itself may consist of a set of submodules. Wherever possible, use should be made of modules from accepted infrastructural packages (e.g., BLAS, LAPACK, PBLAS, ScaLAPACK or BLACS) or of modules already included in the Library.

2.3.2 Comment statements must be inserted where needed to clarify the execution and data flow in the program. Comments must be in English and written in normal printed text format (not all upper case).

2.3.3 Every subprogram (user-callable and lower-level) must contain the WGS copyright note and an in-line documentation in its heading. The in-line documentation consists of the following sections:

- Purpose;
- Arguments;
- Method;
- References;
- Contributors;
- Revisions;
- Keywords.

For a detailed description of the first four points see 3.2.1. See also Appendix A for an example.

2.3.4 The following restrictions, additional to those imposed by the FORTRAN77 standard, are regarded as mandatory by WGS:

- Do not use COMMON, except for communication between library routines. In any case avoid the use of COMMON if possible.
- Do not use EQUIVALENCE.
- Do not use ASSIGN, assigned GO TO, computed GO TO.
- Do not use BLOCK DATA.
- Do not use REAL or DOUBLE PRECISION variables to control DO-loops.
- Do not use ENTRY and alternate RETURN.
- Do not use FORTRAN keywords or intrinsic functions as names of variables or subprograms.

2.3.5 Meaningful variable names should be used. Where a routine is based on referenced published material, variable names should match as closely as possible the nomenclature of the published material. Ensure that variables and subprograms have distinct names.

2.3.6 All arguments must be explicitly typed, i.e., appear in a type statement.

2.3.7 All local variables, i.e., those variables in the body of the routine which are not in the argument list, should be specified in type statements within the routine separately from the arguments.

For development purpose the use of the (non-standard) statement IMPLICIT NONE, if available, is recommended on top of the specification of the local variables.

2.3.8 To give a name to values which are truly constant, PARAMETER statements rather than DATA statements should be used, e.g.,

```
DOUBLE PRECISION  ZERO, ONE
PARAMETER         (ZERO=0.0D0, ONE=1.0D0)
```

DATA statements can be used to assign values to array-elements, or to assign initial values to scalar variables whose values may subsequently be changed.

2.3.9 REAL or DOUBLE PRECISION constants should be defined with at least 20 significant digits (and with no leading zeros) unless their values can be exactly represented in fewer digits (e.g., 1.5) or it is not necessary for the values to be accurate to full machine precision. Rational values should be expressed as rational fractions (e.g., one-third as 1.0/3.0).

2.3.10 Before starting the actual computation the subroutine must check the input parameters for errors and zero dimensions.

The following types of errors in input parameters should be taken into account:

- negative problem dimensions;
- illegal values of mode parameters;
- insufficient size of leading dimensions with respect to the problem dimensions, or in the case of a parallel routine a check of the descriptors must be done (the routine CHK1MAT from ScaLAPACK may be useful);
- insufficient value of workspace size parameter LxWORK (where x is one of the capitals I, S, D, C, Z, L) in case of variable size workspace (see 2.2.10);
- other errors in input parameters arising from the context, e.g., an impossible combination of the values of mode parameters.

If zero dimensions are encountered, which lead to immediate termination, the subroutine must set INFO = 0 and return to the calling routine.

2.3.11 DO statements must have an associated CONTINUE statement to indicate the extent of the resultant loop, and the body of the statements in the loop should be indented.

2.3.12 DO statements should be used to execute a repetition for a given number of times, i.e., the index variable should pass through all values of the list and no GO TO statement should be used to jump out of the DO loop.

Repetitions with an unforeseen number of executions should be implemented by a WHILE construction e.g.,

```
      I = 1
C     WHILE (I <= MAXI and A(I) > 0.0) DO
   10 IF (I .LE. MAXI) THEN
        IF (A(I) .GT. ZERO) THEN
          ...
          I = I + 1
          GO TO 10
        END IF
      END IF
C     END WHILE 10
```

2.3.13 The use of GO TO statements should be avoided as far as possible. Arithmetic IF statements should not be used at all. The use of IF THEN, ELSE, and ELSE IF statements is recommended instead. The body of these statements should be indented.

2.3.14 Statement labels must appear in ascending order right adjusted in the label field.

2.3.15 Computations on arrays should be programmed 'by columns' wherever possible, in order to reduce the amount of paging in virtual-memory systems, and to increase efficiency on some vector-processing systems.
Also to facilitate automatic vectorization, innermost DO loops should be kept simple and, if possible, should not contain IF statements (this is especially important in computationally intensive parts of the code). Moreover, computations in parallel routines should be programmed 'block oriented' in order to take profit of the locality of data due to the distribution of the matrices (care must be taken to minimize communications as far as possible).

2.3.16 Routines should be coded so that the algorithms adapt to characteristics of the computational environment in which they are being executed. Relevant characteristics (e.g., machine base, relative machine precision, largest/smallest machine representable number, safe range parameter) are provided by the LAPACK routine DLAMCH.

2.3.17 The use of level 2 or 3 BLAS and its parallel version in PBLAS is strongly recommended because of efficiency and portability reasons.

2.3.18 It is recommended that the penultimate line of the source code contains a comment statement of the form:

`C *** Last line of` < **subprogram name** > `***`

3 Documentation Standards

3.1 General Rules

3.1.1 Every user-callable subprogram must be documented by a Library Routine Document.

3.1.2 Documentation must provide complete information to enable a user of the subprogram to proceed with a high expectation of successful application.

In particular, limitations or constraints on the use of a subprogram should be emphasized and if possible advice given for overcoming the limitations by the use of alternative subprograms, choice of mode of use, etc..

3.1.3 For every user-callable routine, an example program demonstrating some of the capabilities of the routine must be provided. Note that example programs, together with their data files (if any) and results files, are reproduced in their entirety in the manual. Neither the example program, nor the input data nor the results should be any longer than is absolutely necessary.

3.2 Subprogram Documentation Standards

3.2.1 The Library Routine Document for a user-callable routine must consist of the following numbered sections:

1 Purpose

This section must contain a brief description outlining the purpose of the routine.

2 Specification

This section must contain the heading of the subprogram with the list of all the parameters in the argument list along with a type declaration for each parameter, e.g.,

```
      SUBROUTINE PAB01ND( JOBZ, N, M, A, IA, JA, DESCA, B, IB, JB,
     $                    DESCB, NCONT, INDCON, NBLK, Z, IZ, JZ,
     $                    DESCZ, TAU, TOL, IWORK, DWORK, LDWORK, INFO )
C     .. Scalar Arguments ..
      CHARACTER*1       JOBZ
      INTEGER           INDCON, INFO, LDWORK, M, N, NCONT, IA, JA, IB,
     $                  JB, IZ, JZ
      DOUBLE PRECISION  TOL
C     .. Array Arguments ..
      DOUBLE PRECISION  A(*), B(*), DWORK(*), TAU(*), Z(*)
      INTEGER           IWORK(*), NBLK(*), DESCA(*), DESCB(*), DESCZ(*)
```

Consistently, '$' should be used as the line continuation character.

3 Arguments

This section must contain a list of all parameters along with their description. It is subdivided into the Subsections:

- Mode Parameters;
- Input/Output Parameters;
- Tolerances;
- Workspace;
- Warning Indicator;
- Error Indicator.

The order of the entries in this section must accord with the order of the parameters in the calling sequence of the subroutine (see 2.2.2).

Each data entry must have the form:

< **name** > < **I/O-note** > < **type** > < **array-dimensions** >
< **description** > < **restrictions** >

where the synonyms have the following meaning:

name: The name of the parameter.

I/O-note: If the parameter is a local/global input/output parameter (except for leading dimensions) it must contain the phrase:

(input) in case of an input parameter, the contents of which is not overwritten on exit;

(output) in case of an output parameter, the input contents of which does not affect any output data of the subroutine;

(input/output) in case of an input parameter, the contents of which may be overwritten on exit, possibly by useless data.

In the case of a parallel routine, these phrases are prepended by the word '**local**' or '**global**' depending on whether the sense of the input/output is local or global. An input/output parameter is said to be local (global) if its value is expected to be entered/returned differently (same) in the various processes.

type: The type of the parameter.

array-dimensions: In case of an array argument the array dimensions.

description: A concise description of the parameter.

restrictions: Restrictions imposed on the parameter (if there are any).

In the sequel, some comments and examples for each subsection are provided.

3.1 Mode Parameters
Example:

STAGES CHARACTER*1
Specifies the reduction stages to be performed as follows:
= 'F': Perform the forward stage only;
= 'B': Perform the backward stage only;

= 'A': Perform both (all) stages.

3.2 Input/Output Parameters

Example:

A	(local input/local output) DOUBLE PRECISION array, local dimension (LLD_A, LOCc(JA+N-1)) On entry, the local pieces of the state transition submatrix of A of size n-by-n to be transformed. ... On exit, the leading n-by-n part of this distributed array contains the transformed state transition matrix $U^T \mathrm{sub}(A) U$. ...
IA	(global input) INTEGER The row index in the global array A indicating the first row of the submatrix of A.
JA	(global input) INTEGER The column index in the global array A indicating the first column of the submatrix of A.
DESCA	(global and local input) INTEGER array of dimension DLEN_A. The array descriptor for the distributed matrix A.

It must be mentioned explicitly in the description if an output array argument contains no useful information on exit, e.g.,

On exit, this array contains no useful information.

3.3 Tolerances

The description of a tolerance parameter must contain an outline of how the default tolerance may be invoked (if applicable), e.g.,

TOL	DOUBLE PRECISION The tolerance to be used in rank determination when transforming (A, B). If the user sets TOL > 0, then the given value of TOL is used as a lower bound for the reciprocal condition number ...

3.4 Workspace

Workspace is provided either as fixed size or variable size workspace (see 2.2.10). In case of variable size workspace the description should contain an expression for the optimal size of workspace or a note on how to get the optimal or at least a good value for LxWORK (where x is one of the capitals I, S, D, C, Z, B)

Example for fixed size workspace:

DWORK	DOUBLE PRECISION array, dimension (N)

Example for variable size workspace:

DWORK	DOUBLE PRECISION array, dimension (LDWORK) On exit, if INFO $= 0$, DWORK(1) returns the optimal value of LDWORK.

LDWORK	INTEGER The dimension of the array DWORK. LDWORK is local input and must be at least LDWORK $\geq$ NB_A * (Mp0 + Nq0 + NB_A), where IROFF = MOD(IA-1, MB_A), ICOFF = MOD(JA-1, NB_A), IAROW = INDXG2P(IA, MB_A, MYROW, RSRC_A, NPROW), IACOL = INDXG2P(JA, NB_A, MYCOL, CSRC_A, NPCOL), Mp0 = NUMROC(M+IROFF, MB_A, MYROW, IAROW, NPROW), Nq0 = NUMROC(N+ICOFF, NB_A, MYCOL, IACOL, NPCOL), and NUMROC, INDXG2P are ScaLAPACK tool functions; MYROW, MYCOL, NPROW and NPCOL can be determined by calling the subroutine BLACS_GRIDINFO.

Note that assumed size $(*)$ dimensions are not to be used.

3.5 Warning Indicator

For example:

IWARN INTEGER

- =0: No warning;
- =1: The matrix A is rank deficient.

3.6 Error Indicator

The description of the diagnostic argument must have the form:

INFO INTEGER

- =0: successful exit;
- <0: if the i^{th} argument is an array and the j^{th} entry had an illegal value, then INFO $= -(100 * i + j)$, if the i^{th} argument is a scalar and had an illegal value, then INFO $= -i$;
- =1: . . . ;

  ⋮
- =n: . . . .

where the lines 1 to 3 are mandatory.

4 Method

This section must contain a description of the algorithm used by the routine.

5 References

This section must contain a list of references which the user may consult for further information. Note that theses and internal reports should not be referenced if a reference in the open literature is available.

6 Numerical Aspects

In this section information about the algorithm concerning accuracy, numerical stability, and operations count (if applicable) can be given.

7 Parallel Execution Recomendations

- Block size, if it cannot be automatically computed

- Performance estimation (if possible)
- Topology recommended

Other considerations for the execution in parallel.

8 Further Comments

This section is meant for additional remarks about the routine which may be of interest to users, e.g.,

> This routine may be BASE (machine base) dependant.

9 Example

This section must contain a brief description of the example problem to be solved. See Appendix B for an example.

9.1 Program Text

A simple program to show how to use the subprogram.

9.2 Program Data

Input data of a small well conditioned problem, for which the results are reproducible on different machines.

9.3 Program Results

Output data produced by the test program with the given input data.

See Appendix B for a complete example.

3.2.2 In the Library Routine Document empty (sub)sections must be included in the text with the description 'None.'.

3.2.3 In the Library Routine Document mathematical variables must be designated by slanted letters and FORTRAN variables by Roman letters, e.g.,

> The array A contains the matrix A.

3.2.4 In the Library Routine Document matrix dimensions must be written in the form $<$ dimension-1 $>$-by-$<$ dimension-2 $>$, e.g.,

> A is an m-by-n matrix...

where m and n correspond to the FORTRAN variables M and N, respectively.

3.2.5 The end of a Library Routine Document must be marked by a horizontal line.

3.2.6 Every subprogram (user-callable and lower-level) must have in-line documentation inserted in the source code between the heading and the first executable statement. This documentation must contain the items which are listed in 2.3.3. See 3.2.1 for a detailed description and Appendix A for a complete example.

3.2.7 In in-line documentations the character `'` should be used to denote the (conjugate) transposed matrix, e.g., `... the matrix U' * A * U ...` .

A Subprogram Description Example

```
      SUBROUTINE PAB01ND( JOBZ, N, M, A, IA, JA, DESCA, B, IB, JB,
     $                    DESCB, NCONT, INDCON, NBLK, Z, IZ, JZ,
     $                    DESCZ, TAU, TOL, IWORK, DWORK, LDWORK, INFO )
C
C     RELEASE 1.0.
C
C     PURPOSE
C
C     To find a controllable realization for the linear time-invariant
C     multi-input system either in the continous
C
C             dX/dt = sub( A ) * X + sub( B ) * U,
C
C     or discrete form
C
C             Xk+1 = sub( A ) * Xk + sub( B ) * Uk,
C
C     where sub( A ) = A(IA:IA+N-1,JA:JA+N-1) and
C     sub( B ) = B(IB:IB+N-1,JB:JB+M-1) are N-by-N and N-by-M distribu-
C     ted matrices, respectively, which are reduced by this routine to
C     orthogonal canonical form using (and optionally accumulating)
C     orthogonal similarity transformations.  Specifically, the pair
C     ( sub( A ), sub( B ) ) is reduced to the pair (Ac, Bc),
C     Ac = sub( Z )' * sub( A ) * sub( Z ),  Bc = sub( Z )' * sub( B ),
C     given by
C
C             [ Acont    *    ]         [ Bcont ]
C        Ac = [               ],   Bc = [       ],
C             [   0   Auncont ]         [   0   ]
C
C        and
C
C                [ A11 A12  . . .  A1,p-1 A1p ]         [ B1 ]
C                [ A21 A22  . . .  A2,p-1 A2p ]         [ 0  ]
C                [  0  A32  . . .  A3,p-1 A3p ]         [ 0  ]
C        Acont = [  .   .   . . .    .     .  ],   Bc = [ .  ],
C                [  .   .     . .    .     .  ]         [ .  ]
C                [  .   .       .    .     .  ]         [ .  ]
C                [  0   0   . . .  Ap,p-1 App ]         [ 0  ]
C
C     where the blocks  B1, A21, ..., Ap,p-1  have full row ranks and
C     p is the controllability index of the pair.  The size of the
C     block  Auncont is equal to the dimension of the uncontrollable
```

```
C     subspace of the pair ( sub( A ), sub( B ) ).
C
C     ARGUMENTS
C
C     Mode Parameters
C
C     JOBZ    CHARACTER*1
C             Indicates whether the user wishes to accumulate in a
C             distributed submatrix sub( Z ) = Z(IZ:IZ+N-1,JZ:JZ+N-1)
C             the orthogonal similarity transformations for reducing the
C             system, as follows:
C             = 'N':  Do not form sub( Z ) and do not store the
C                     orthogonal transformations;
C             = 'F':  Do not form sub( Z ), but store the orthogonal
C                     transformations in the factored form;
C             = 'I':  sub( Z ) is initialized to the unit matrix and
C                     the orthogonal transformation matrix sub( Z ) is
C                     returned.
C
C     Input/Output Parameters
C
C     N       (global input) INTEGER
C             The order of the original state-space representation,
C             i.e. the order of the distributed submatrix sub( A ).
C             N >= 0.
C
C     M       (global input) INTEGER
C             The number of system inputs, or of columns of the
C             distributed submatrix sub( B ).  M >= 0.
C
C     A       (local input/local output) DOUBLE PRECISION pointer into
C             the local memory to an array of dimension
C             (LLD_A, LOCc(JA+N-1))
C             On entry, the local pieces of the N-by-N distributed
C             matrix sub( A ) must contain the original state dynamics
C             matrix A.
C             On exit, the leading NCONT-by-NCONT part contains the
C             upper block Hessenberg state dynamics matrix Acont in Ac,
C             given by sub( Z )' * sub( A ) * sub( Z ), of a
C             controllable realization for the original system. The
C             elements below the first block-subdiagonal are set to
C             zero.
C
C     IA      (global input) INTEGER
C             The row index in the global array A indicating the first
```

```
C             row of sub( A ).
C
C     JA      (global input) INTEGER
C             The column index in the global array A indicating the
C             first column of sub( A ).
C
C     DESCA   (global and local input) INTEGER array of dimension DLEN_
C             The array descriptor for the distributed matrix A.
C
C     B       (local input/local output) DOUBLE PRECISION pointer into
C             the local memory to an array of dimension
C             (LLD_B, LOCc(JB+M-1))
C             On entry, the local pieces of the N-by-M distributed
C             matrix sub( B ) must contain the input matrix B.
C             On exit, the leading NCONT-by-M part of this array
C             contains the transformed input matrix Bcont in Bc, given
C             by sub( Z )' * sub( B ), with all elements but the first
C             block set to zero.
C
C     IB      (global input) INTEGER
C             The row index in the global array B indicating the first
C             row of sub( B ).
C
C     JB      (global input) INTEGER
C             The column index in the global array B indicating the
C             first column of sub( B ).
C
C     DESCB   (global and local input) INTEGER array of dimension DLEN_
C             The array descriptor for the distributed matrix B.
C
C     NCONT   (global output) INTEGER
C             The order of the controllable state-space representation.
C
C     INDCON  (global output) INTEGER
C             The controllability index of the controllable part of the
C             system representation.
C
C     NBLK    (global output) INTEGER array, dimension (N)
C             The leading INDCON elements of this array contain
C             the orders of the diagonal blocks of Acont.
C
C     Z       (local output) DOUBLE PRECISION pointer into the local
C             memory to an array of dimension (LLD_Z, LOCc(JZ+N-1))
C             If JOBZ = 'I', then the local pieces of the N-by-N
C             distributed matrix sub( Z ) contains the matrix of
```

```
C            accumulated orthogonal similarity transformations which
C            reduces the given system to orthogonal canonical form.
C            If JOBZ = 'F', the elements below the diagonal, with the
C            array TAU, represent the orthogonal transformation matrix
C            as a product of elementary reflectors. The transformation
C            matrix can then be obtained by calling the ScaLAPACK
C            library routine PDORGQR.
C            If JOBZ = 'N', the distributed matrix sub( Z ) is not
C            referenced.
C
C    IZ      (global input) INTEGER
C            The row index in the global array Z indicating the first
C            row of sub( Z ).
C
C    JZ      (global input) INTEGER
C            The column index in the global array Z indicating the
C            first column of sub( Z ).
C
C    DESCZ   (global and local input) INTEGER array of dimension DLEN_
C            The array descriptor for the distributed matrix Z.
C
C    TAU     (local output) DOUBLE PRECISION array, dimension
C            LOCc(JA+N-1)
C            This array contains the scalar factors
C            TAU of the elementary reflectors. TAU is tied to the
C            distributed matrix A.
C
C    Tolerances
C
C    TOL     DOUBLE PRECISION
C            The tolerance to be used in rank determination when
C            transforming (sub( A ), sub( B )).
C            If the user sets TOL > 0, then the given value of TOL is
C            used as a lower bound for the reciprocal condition number
C            (see the description of the argument RCOND in the P-SLICOT
C            routine PMB03OD); a (sub)matrix whose estimated condition
C            number is less than 1/TOL is considered to be of full
C            rank.
C            If the user sets TOL <= 0, then an implicitly computed,
C            default tolerance, defined by
C            TOLDEF = N*N*EPS*MAX( NORM(sub( A )), NORM(sub( B )) ), is
C            used instead, where EPS is the machine precision (see
C            ScaLAPACK Library routine PDLAMCH).
C
C    Workspace
```

```
C
C     IWORK   INTEGER array, dimension MAX( LOCc(JA+N-1), LOCc(JB+M-1) )
C
C     DWORK   DOUBLE PRECISION array, dimension (LDWORK)
C             On exit, if INFO = 0, DWORK(1) returns the optimal value
C             of LDWORK.
C
C     LDWORK  INTEGER
C             The length of the array DWORK.
C             LDWORK >= MAX( (NB_A*(NB_A-1))/2, ( NqA0 + MAX( NpB0 +
C                     NUMROC( NUMROC( N+ICOFFA, NB_B, 0, 0, NPCOL ),
C                             NB_B, 0, 0, LCMQ ), MpA0 ) )*NB_B ) +
C                     NB_B * NB_B + NpB0 * MqB0
C
C             where LCMQ = LCM / NPCOL with LCM = ICLM( NPROW, NPCOL ),
C
C             IROFFB = MOD( IB-1, MB_B ), ICOFFB = MOD( JB-1, NB_B ),
C             IBROW = INDXG2P( IB, MB_B, MYROW, RSRC_B, NPROW ),
C             IBCOL = INDXG2P( JB, NB_B, MYCOL, CSRC_B, NPCOL ),
C             NpB0 = NUMROC( N+IROFFB, MB_B, MYROW, IBROW, NPROW ),
C             MqB0 = NUMROC( M+ICOFFB, MB_B, MYCOL, IBCOL, NPROW ),
C
C             IROFFA = MOD( IA-1, MB_A ), ICOFFA = MOD( JA-1, NB_A ),
C             IAROW = INDXG2P( IA, MB_A, MYROW, RSRC_A, NPROW ),
C             IACOL = INDXG2P( JA, NB_A, MYCOL, CSRC_A, NPCOL ),
C             NpA0 = NUMROC( N+IROFFA, MB_A, MYROW, IAROW, NPROW ),
C             NqA0 = NUMROC( N+ICOFFA, NB_A, MYCOL, IACOL, NPCOL ),
C
C             ILCM, INDXG2P and NUMROC are ScaLAPACK tool functions;
C             MYROW, MYCOL, NPROW and NPCOL can be determined by calling
C             the subroutine BLACS_GRIDINFO.
C
C             If LDWORK=-1, then LDWORK is global input and a workspace
C             query is assumed; the routine only calculates the minimum
C             and optimal size for all work arrays. Each of these
C             values is returned in the first entry of the corresponding
C             work array, and no error message is issued by PXERBLA.
C
C     Error Indicator
C
C     INFO    INTEGER
C             = 0:  successful exit;
C             < 0:  If the i-th argument is an array and the j-entry had
C                   an illegal value, then INFO = -(100*i+j), if the
C                   i-th argument is a scalar and had an illegal value,
```

```
C                 then INFO = -i.
C
C     METHOD
C
C     Matrix sub( B ) is first QR-decomposed and the appropriate
C     orthogonal similarity transformation applied to the matrix
C     sub( A ). Leaving the first rank( sub( B ) ) states unchanged, the
C     remaining lower left block of sub( A ) is then QR-decomposed and
C     the new orthogonal matrix, Q1, is also applied to the right of
C     sub( A ) to complete the similarity transformation. By continuing
C     in this manner, a completely controllable state-space pair
C     (Acont, Bcont) is found for the given (sub( A ), sub( B )), where
C     Acont is upper block Hessenberg with each subdiagonal block of
C     full row, and Bcont is zero apart from its (independent) first
C     rank(sub( B )) rows. NOTE that the system controllability indices
C     are easily calculated from the dimensions of the blocks of Acont.
C
C     REFERENCES
C
C     [1] Van Dooren, P.M.
C         The computation of Kronecker's canonical form of a singular
C         pencil.
C         Linear Algebra & Its Appl., Vol. 27, pp. 103-141, 1979.
C
C     [2] Quintana-Orti, E.S., Quintana, G. and Hernandez, V.
C         Parallel Algorithms for the Block Hessenberg Form and
C         Applications.
C         Preprints of the 4th IFAC Workshop on Algorithms and
C         Architectures for Real-Time Control, Vilamoura, Portugal,
C         pp. 353-358, 9-11 April 1997.
C
C     [3] Konstantinov, M.M., Petkov, P.Hr. and Christov, N.D.
C         Orthogonal Invariants and Canonical Forms for Linear
C         Controllable Systems.
C         Proc. 8th IFAC World Congress, Kyoto, 1, pp. 49-54, 1981.
C
C     [4] Alvarruiz, F., Hernandez, V., Ruiz, P.A. and Vidal, A.M.
C         Computing Minimal Realizations of Control Linear Systems.
C         An Approach Based on Distributed Memory Algorithms
C         3rd Portuguese Conference on Automatic Control, September 1998.
C
C     NUMERICAL ASPECTS
C                                   3
C     The algorithm requires 0(N ) operations and is backward stable.
C
```

```
C     PARALLEL EXECUTION RECOMENDATIONS
C
C     Due to the actual matrix alignment restrictions of some
C     subroutines of ScaLAPACK (v1.5), it is not possible to take
C     any block size (NB). In general, it is a good solution to take a
C     divisor of M (columns of B) as NB. Anyway, it is foreseeable that
C     these restrictions disappear in new ScaLAPACK versions.
C
C     With respect to the topology, a ring seems to obtain better
C     performances.
C
C     REVISIONS
C
C     January 8, 1998.
C
C     KEYWORDS
C
C     Controllability, minimal realization, orthogonal canonical form,
C     orthogonal transformation.
C
C     ******************************************************************
C
C     .. Parameters ..
      INTEGER           BLOCK_CYCLIC_2D, DLEN_, DT_, CTXT_, M_, N_,
     $                  MB_, NB_, RSRC_, CSRC_, LLD_,
     $                  IMAX, IMIN
      PARAMETER         ( BLOCK_CYCLIC_2D = 1, DLEN_ = 9, DT_ = 1,
     $                    CTXT_ = 2, M_ = 3, N_ = 4, MB_ = 5, NB_ = 6,
     $                    RSRC_ = 7, CSRC_ = 8, LLD_ = 9,
     $                    IMAX = 1, IMIN = 2 )
      DOUBLE PRECISION  ZERO, ONE
      PARAMETER         ( ZERO=0.0D0, ONE=1.0D0 )
C     .. Scalar Arguments ..
      CHARACTER*1       JOBZ
      INTEGER           INDCON, INFO, LDWORK, M, N, NCONT, IA, JA, IB,
     $                  JB, IZ, JZ
      DOUBLE PRECISION  TOL
C     .. Array Arguments ..
      DOUBLE PRECISION  A(*), B(*), DWORK(*), TAU(*), Z(*)
      INTEGER           IWORK(*), NBLK(*), DESCA(*), DESCB(*), DESCZ(*)
C     .. Local Scalars ..
      LOGICAL           LJOBF, LJOBI, LJOBZ, LQUERY
      INTEGER           I, ITAU, J, JWORK, MCRT, MM, NBL, NCRT, NI, NJ,
     $                  LDA, RANK, NPROW, NPCOL, MYROW, MYCOL, NFIL,
     $                  NCOL, NBA, ICTXT, IDIM, AUX, IROFFA, ICOFFA,
```

```
     $                   IROFFB, ICOFFB, IAROW, IACOL, IBROW, IBCOL,
     $                   NPA0, NQA0, NPB0, MQB0, LCM, LCMQ, LWMIN
C
      DOUBLE PRECISION   ANORM, BNORM, FANORM, FBNORM, FNRM, THRESH,
     $                   TOLDEF, WRKOPT
C     .. Local Arrays ..
      DOUBLE PRECISION   SVAL(3), TAU_AUX(N)
      INTEGER            DESCW(LLD_), DESCJ(LLD_), DESCTAU(LLD_)
C     .. External Functions ..
      LOGICAL            LSAME
      DOUBLE PRECISION   PDLAMCH, PDLANGE, DLAPY2
      INTEGER            NUMROC, INDXG2P, ILCM
      EXTERNAL           PDLAMCH, PDLANGE, DLAPY2, LSAME, NUMROC,
     $                   INDXG2P, ILCM
C     .. External Subroutines ..
      EXTERNAL           PDCOPY, PDLACPY, PDLASET, PDORGQR, PDORMQR,
     $                   PMB01PD, PMB03OD, PXERBLA, BLACS_GRIDINFO,
     $                   DESCINIT, PDLAPRNT, PIELGET
C     .. Intrinsic Functions ..
      INTRINSIC          DBLE, MAX, MIN, MOD
C     ..
C     .. Executable Statements ..
C
      LJOBF = LSAME( JOBZ, 'F' )
      LJOBI = LSAME( JOBZ, 'I' )
      LJOBZ = LJOBF.OR.LJOBI
C
C     Get grid parameters
C
      ICTXT = DESCA( CTXT_ )
      CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )
C
C     Test the input parameters.
C
      INFO = 0
      IF( .NOT.LJOBZ .AND. .NOT.LSAME( JOBZ, 'N' ) ) THEN
         INFO = -1
      ELSE
         IF( NPROW.EQ.-1 ) THEN
            INFO = -(700+CTXT_)
         ELSE
            CALL CHK1MAT( N, 2, N, 2, IA, JA, DESCA, 7, INFO)
            CALL CHK1MAT( N, 2, M, 3, IB, JB, DESCB, 11, INFO)
            IF( LJOBZ ) THEN
               CALL CHK1MAT( N, 2, N, 2, IZ, JZ, DESCZ, 18, INFO)
```

```
      END IF
      IF( INFO.EQ.0 ) THEN
         IROFFA = MOD( IA-1, DESCA( MB_ ) )
         ICOFFA = MOD( JA-1, DESCA( NB_ ) )
         IROFFB = MOD( IB-1, DESCB( MB_ ) )
         ICOFFB = MOD( JB-1, DESCB( NB_ ) )
         IAROW = INDXG2P( IA, DESCA( MB_ ), MYROW, DESCA( RSRC_ ),
     $                    NPROW )
         IACOL = INDXG2P( JA, DESCA( NB_ ), MYCOL, DESCA( CSRC_ ),
     $                    NPCOL )
         IBROW = INDXG2P( IB, DESCB( MB_ ), MYROW, DESCB( RSRC_ ),
     $                    NPROW )
         IBCOL = INDXG2P( JB, DESCB( NB_ ), MYCOL, DESCB( CSRC_ ),
     $                    NPCOL )
         NPA0 = NUMROC( N+IROFFA, DESCA( MB_ ), MYROW, IAROW,
     $                  NPROW )
         NQA0 = NUMROC( N+ICOFFA, DESCA( NB_ ), MYCOL, IACOL,
     $                  NPCOL )
         NPB0 = NUMROC( N+IROFFB, DESCB( MB_ ), MYROW, IBROW,
     $                  NPROW )
         MQB0 = NUMROC( M+ICOFFB, DESCB( NB_ ), MYCOL, IBCOL,
     $                  NPCOL )
         LCM = ILCM( NPROW, NPCOL )
         LCMQ = LCM / NPCOL
         LWMIN =  MAX( ( DESCB( NB_ ) * ( DESCB( NB_ ) - 1 ) )
     $           / 2, ( NQA0 + MAX( NPB0 + NUMROC( NUMROC(
     $           N+ICOFFA, DESCA( NB_ ), 0, 0, NPCOL ),
     $           DESCA( NB_ ), 0, 0, LCMQ ), NPA0 ) ) *
     $           DESCA( NB_ ) ) + DESCA( NB_ ) * DESCA( NB_ ) +
     $           NPB0 * MQB0
         DWORK( 1 ) = DBLE( LWMIN )
         LQUERY = ( LDWORK.EQ.-1 )
         IF( LDWORK.LT.LWMIN .AND. .NOT.LQUERY ) THEN
            INFO = -23
         ELSE IF( IA.NE.1) THEN
            INFO = -5
         ELSE IF( JA.NE.1) THEN
            INFO = -6
         ELSE IF( IB.NE.1) THEN
            INFO = -9
         ELSE IF( JB.NE.1) THEN
            INFO = -10
         ELSE IF( IZ.NE.1) THEN
            INFO = -16
         ELSE IF( JZ.NE.1) THEN
```

```
                  INFO = -17
               END IF
            END IF
         END IF
      END IF
C
      IF ( INFO.NE.0 ) THEN
C
C        Error return.
C
         CALL PXERBLA(ICTXT, 'PAB01ND', -INFO )
         RETURN
      ELSE IF( LQUERY ) THEN
C
C        LDWORK size query.
C
         RETURN
      END IF
C
      NCONT  = 0
      INDCON = 0
C
C     Quick return if possible.
C
      IF ( MIN( N, M ).EQ.0 )
     $   RETURN
C
C     Calculate the absolute norms of sub( A ) and sub( B ) (used for
C     scaling).
C
      ANORM = PDLANGE( 'M', N, N, A, IA, JA, DESCA, DWORK )
      BNORM = PDLANGE( 'M', N, M, B, IB, JB, DESCB, DWORK )
C
C     Return if matrix sub( B ) is zero.
C
      IF( BNORM.EQ.ZERO )
     $   RETURN
C
C     Scale (if needed) the submatrices sub( A ) and sub( B ).
C
      CALL PMB01PD( 'Scale', 'G', N, N, 0, 0, ANORM, 0, NBLK, A, IA, JA,
     $              DESCA, INFO )
      CALL PMB01PD( 'Scale', 'G', N, M, 0, 0, BNORM, 0, NBLK, B, IB, JB,
     $              DESCB, INFO )
C
```

```
C     Calculate the Frobenius norms of sub( A ) and sub( B ) (used for
C      rank determination).
C
      FANORM = PDLANGE( 'F', N, N, A, IA, JA, DESCA, DWORK )
      FBNORM = PDLANGE( 'F', N, M, B, IB, JB, DESCB, DWORK )
C
      FNRM = DLAPY2( FANORM, FBNORM )
C
      TOLDEF = TOL
      IF ( TOLDEF.LE.ZERO ) THEN
C
C        Use the default tolerance in controllability determination.
C
         THRESH = DBLE( N*N )*PDLAMCH( ICTXT, 'EPSILON' )
         TOLDEF = THRESH*MAX( FANORM, FBNORM )
      END IF
C
      WRKOPT = ONE
      NI = IA-1
      ITAU = 1
      NCRT = N
      MCRT = M
C
C     Workspace: N*M.
C     (Note: Comments in the code beginning "Workspace:" describe the
C     minimal amount of real workspace needed at that point in the
C     code).
C
      LDA = MAX( 1, NUMROC(NCRT, DESCA( MB_ ), MYROW, 0, NPROW))
      CALL DESCINIT( DESCW, NCRT, MCRT, DESCA( MB_ ), DESCA( NB_ ), 0,
     $               0, ICTXT, LDA, INFO )

      CALL PDLACPY( 'G', N, M, B, IB, JB, DESCB, DWORK, 1, 1, DESCW )
      CALL PDLASET( 'G', N, M, ZERO, ZERO, B, IB, JB, DESCB )

      IDIM = NUMROC( M+N, DESCA(NB_), MYCOL, 0, NPCOL )
      CALL DESCINIT( DESCJ, 1, N+M, 1, DESCA( NB_ ), 0, 0, ICTXT, 1,
     $               INFO )
      CALL DESCINIT( DESCTAU, 1, N, 1, DESCA( NB_ ), 0, 0, ICTXT, 1,
     $               INFO )
C
   10 CONTINUE
C
         NFIL = NUMROC( NCRT, DESCA( NB_ ), MYROW, DESCA( RSRC_ ),
     $                  NPROW)
```

```
      NCOL = NUMROC( MCRT, DESCA( MB_ ), MYCOL, DESCA( CSRC_ ),
     $               NPCOL )
      JWORK = NFIL * NCOL + 1

C
C     Rank-revealing QR decomposition with column pivoting.
C     Workspace: 3*(MAX(3,Mp0 + Nq0) + LOCc(JA+N-1)+Nq0).
C
      DO 20 I = 1, IDIM
         IWORK(I) = 0
   20 CONTINUE
C

      CALL PMB03OD( 'QR', NCRT, MCRT, DWORK, 1, 1, DESCW, IWORK,
     $              TOLDEF, FNRM, TAU_AUX, RANK, SVAL, DWORK(JWORK),
     $              LDWORK-JWORK+1, INFO )
C
      IF ( RANK.NE.0 ) THEN
         MM = MIN( NCRT, MCRT )
         NJ = NI + JA -1
         NI = NCONT
         NCONT = NCONT + RANK
         INDCON = INDCON + 1
         NBLK(INDCON) = RANK
C
C        Premultiply and postmultiply the appropriate block row
C        and block column of A by Q' and Q, respectively.
C        Workspace: MAX( (NB_DW*(NB_DW-1))/2, (NqA0 + MpA0)*NB_DW )+
C                   NB_DW * NB_DW
C
         CALL PDORMQR( 'Left', 'Transpose', NCRT, NCRT, MM, DWORK,
     $                 1, 1, DESCW, TAU_AUX, A, NI+1, NI+1, DESCA,
     $                 DWORK(JWORK), LDWORK-JWORK+1, INFO )
         WRKOPT = MAX( WRKOPT, DWORK(JWORK) )
C
C        Workspace: MAX( (NB_DW*(NB_DW-1))/2, ( NqA0 + MAX( NpDW0 +
C                   NUMROC( NUMROC( N+ICOFFA, NB_DW, 0, 0, NPCOL ),
C                   NB_DW, 0, 0, LCMQ ), MpA0 ) )*NB_DW ) +
C                   NB_DW * NB_DW
C

         CALL PDORMQR( 'Right', 'No transpose', N, NCRT, MM,
     $                 DWORK, 1, 1, DESCW, TAU_AUX, A, 1, NI+1,
     $                 DESCA, DWORK(JWORK), LDWORK-JWORK+1, INFO )
         WRKOPT = MAX( WRKOPT, DWORK(JWORK) )
```

```
C
C           Save scalar factors of the elementary reflectors
C
C            IF( MYROW.EQ.0)
C     $         CALL PDCOPY( MCRT, TAU_AUX, 1, 1, DESCTAU, 1, TAU, 1,
C     $                      ITAU, DESCTAU, 1 )
C
C           If required, save transformations.
C
            IF ( LJOBZ.AND.MAX( NCRT, MM ).GT.1 ) THEN
               CALL PDLACPY( 'L', NCRT-1, MM, DWORK, 2, 1, DESCW,
     $                       Z, NI+2, ITAU, DESCZ )
            END IF
C
C           Zero the subdiagonal elements of the current matrix.
C
            IF (RANK.GT.1 ) CALL PDLASET( 'L', RANK-1, RANK-1, ZERO,
     $                                    ZERO, DWORK, 2, 1, DESCW )
C
C           Backward permutation of the columns of B or A.
C
            IF ( INDCON.EQ.1 ) THEN
               DO 30 J = 1, M
                  CALL PIELGET( 'ALL', ' ', AUX, IWORK, 1, J, DESCJ )
                  CALL PDCOPY( RANK, DWORK, 1, J, DESCW, 1,
     $                         B, 1, AUX, DESCB, 1 )
   30          CONTINUE
            ELSE
               DO 40 J = 1, MCRT
                  CALL PIELGET( 'ALL', ' ', AUX, IWORK, 1, J, DESCJ )
                  CALL PDCOPY( RANK, DWORK, 1, J, DESCW, 1,
     $                         A, NI+1, NJ+AUX, DESCA, 1 )
   40          CONTINUE
C
            END IF
C
            IF ( RANK.NE.NCRT ) THEN
               MCRT = RANK
               NCRT = NCRT - RANK
               NBA = DESCA( NB_)
               LDA = MAX( 1, NUMROC( NCRT, DESCA( NB_ ), MYROW, 0,
     $                  NPROW ) )
               CALL DESCINIT( DESCW, NCRT, MCRT, DESCA( MB_ ),
     $                        DESCA( NB_ ), 0, 0, ICTXT, LDA, INFO )
               CALL PDLACPY( 'G', NCRT, MCRT, A, NCONT+1, NI+1, DESCA,
```

```
     $                        DWORK, 1, 1, DESCW )
               CALL PDLASET( 'G', NCRT, MCRT, ZERO, ZERO, A, NCONT+1,
     $                        NI+1, DESCA )
               ITAU = ITAU + MM
               GO TO 10
            END IF
         END IF
C
C     If required, accumulate transformations.
C     Workspace: NB_Z * ( NqZ0 + MpZ0 + NB_Z )
C
      IF ( LJOBI ) THEN
         CALL PDORGQR( N, N, N, Z, IZ, JZ, DESCZ, TAU, DWORK, LDWORK,
     $                  INFO )
         WRKOPT = MAX( WRKOPT, DWORK(1) )
      END IF
C
C     Annihilate the trailing elements of TAU, if JOBZ = 'F'.
C
C      IF ( LJOBF ) THEN
C         DO 50 J = ITAU+1, N
C            TAU(J) = ZERO
C 50      CONTINUE
C      END IF
C
C     Undo scaling of A and B.
C
      IF ( INDCON.LT.N ) THEN
         NBL = INDCON + 1
         NBLK(NBL) = N - NCONT
      ELSE
         NBL = 0
      END IF
      CALL PMB01PD( 'Undo', 'H', N, N, 0, 0, ANORM, NBL, NBLK, A,
     $              IA, JA, DESCA, INFO )
      CALL PMB01PD( 'Undo', 'G', NBLK(1), M, 0, 0, BNORM, 0, NBLK, B,
     $              IB, JB, DESCB, INFO )
C
C     Set minimal workspace dimension.
C
      DWORK(1) = WRKOPT
      DWORK(2) = LWMIN
      RETURN
C *** Last line of PAB01ND ***
      END
```

B Subprogram Documentation Example

PAB01ND – P-SLICOT Library Routine Document

1 Purpose

To find a controllable realization for the linear time-invariant multi-input system

$$\frac{dX}{dt} = sub(A) * X + sub(B) * U,$$

where sub(A) = A(IA:IA+N-1,JA:JA+N-1) and sub(B) = B(IB:IB+N-1,JB:JB+M-1) are n-by-n and n-by-m distributed matrices, respectively, which are reduced by this routine to orthogonal canonical form using (and optionally accumulating) orthogonal similarity transformations. Specifically, the pair (sub(A), sub(B)) is reduced to the pair (A_c, B_c), A_c = sub(Z)^T * sub(A) * sub(Z), B_c = sub(Z)^T * sub(B), given by

$$A_c = \begin{pmatrix} A_{cont} & * \\ 0 & A_{uncont} \end{pmatrix}, B_c = \begin{pmatrix} B_{cont} \\ 0 \end{pmatrix},$$

and

$$A_{cont} = \begin{pmatrix} A_{11} & A_{12} & \dots & A_{1,p-1} & A_{1p} \\ A_{21} & A_{22} & \dots & A_{2,p-1} & A_{2p} \\ 0 & A_{32} & \dots & A_{3,p-1} & A_{3p} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & A_{p,p-1} & A_{pp} \end{pmatrix}, B_c = \begin{pmatrix} B_1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

where the blocks $B_1, A_{21}, \dots, A_{p,p-1}$ have full row rank and p is the controllability index of the pair. The size of the block A_{uncont} is equal to the dimension of the uncontrollable subspace of the pair (sub(A), sub(B)).

2 Specification

```
      SUBROUTINE PAB01ND( JOBZ, N, M, A, IA, JA, DESCA, B, IB, JB,
     $                    DESCB, NCONT, INDCON, NBLK, Z, IZ, JZ,
     $                    DESCZ, TAU, TOL, IWORK, DWORK, LDWORK, INFO )
C     .. Scalar Arguments ..
      CHARACTER*1       JOBZ
      INTEGER           INDCON, INFO, LDWORK, M, N, NCONT, IA, JA, IB,
     $                  JB, IZ, JZ
      DOUBLE PRECISION  TOL
C     .. Array Arguments ..
      DOUBLE PRECISION  A(*), B(*), DWORK(*), TAU(*), Z(*)
      INTEGER           IWORK(*), NBLK(*), DESCA(*), DESCB(*), DESCZ(*)
```

3 Arguments

3.1 Mode Parameters

JOBZ CHARACTER*1
Indicates whether the user wishes to accumulate in a distributed submatrix sub(Z) = Z(IZ:IZ+N-1,JZ:JZ+N-1) the orthogonal similarity transformations for reducing the system, as follows:
= 'N': Do not form sub(Z) and do not store the orthogonal transformations;
= 'F': Do not form sub(Z), but store the orthogonal transformations in the factored form;
= 'I': sub(Z) is initialized to the unit matrix and the orthogonal transformation matrix sub(Z) is returned.

3.2 Input/Output Parameters

N (global input) INTEGER
The order of the original state-space representation, i.e. the order of the distributed submatrix sub(A). N $\geq$ 0.

M (global input) INTEGER
The number of system inputs, or of columns of the distributed submatrix sub(B). M $\geq$ 0.

A (local input/local output) DOUBLE PRECISION pointer into the local memory to an array of dimension (LLD_A, LOCc(JA+N-1))
On entry, the local pieces of the n-by-n distributed matrix sub(A) must contain the original state dynamics matrix A.
On exit, the leading n_{cont}-by-n_{cont} part contains the upper block Hessenberg state dynamics matrix A_{cont} in A_c, given by sub(Z)^T * sub(A) * sub(Z), of a controllable realization for the original system. The elements below the first block-subdiagonal are set to zero.

IA (global input) INTEGER
The row index in the global array A indicating the first row of sub(A).

JA (global input) INTEGER
The column index in the global array A indicating the first column of sub(A).

DESCA (global and local input) INTEGER array of dimension DLEN_
The array descriptor for the distributed matrix A.

B (local input/local output) DOUBLE PRECISION pointer into the local memory to an array of dimension (LLD_B, LOCc(JB+M-1))
On entry, the local pieces of the n-by-m distributed matrix sub(B) must contain the input matrix B.
On exit, the leading n_{cont}-by-m part of this array contains the transformed input matrix B_{cont} in B_c, given by sub(Z)^T * sub(B), with all elements but the first block set to zero.

IB (global input) INTEGER
The row index in the global array B indicating the first row of sub(B).

JB (global input) INTEGER
The column index in the global array B indicating the first column of sub(B).

DESCB (global and local input) INTEGER array of dimension DLEN_
The array descriptor for the distributed matrix B.

NCONT (global output) INTEGER
The order of the controllable state-space representation.

INDCON (global output) INTEGER
The controllability index of the controllable part of the system representation.

NBLK (global output) INTEGER array, dimension (N)
The leading INDCON elements of this array contain the orders of the diagonal blocks of A_{cont}.

Z (local output) DOUBLE PRECISION pointer into the local memory to an array of dimension (LLD_Z, LOCc(JZ+N-1))
If JOBZ = 'I', then the local pieces of the n-by-n distributed matrix sub(Z) contains the matrix of accumulated orthogonal similarity transformations which reduces the given system to orthogonal canonical form. If JOBZ = 'F', the elements below the diagonal, with the array TAU, represent the orthogonal transformation matrix as a product of elementary reflectors. The transformation matrix can then be obtained by calling the ScaLAPACK library routine PDORGQR. If JOBZ = 'N', the distributed matrix sub(Z) is not referenced.

IZ (global input) INTEGER
The row index in the global array Z indicating the first row of sub(Z).

JZ (global input) INTEGER
The column index in the global array Z indicating the first column of sub(Z).

DESCZ (global and local input) INTEGER array of dimension DLEN_
The array descriptor for the distributed matrix Z.

TAU (local output) DOUBLE PRECISION array, dimension LOCc(JA+N-1)
This array contains the scalar factors TAU of the elementary reflectors. TAU is tied to the distributed matrix A.

3.3 Tolerances

TOL DOUBLE PRECISION
The tolerance to be used in rank determination when transforming (sub(A), sub(B)). If the user sets TOL > 0, then the given value of TOL is used as a lower bound for the reciprocal condition number (see the description of the argument RCOND in the P-SLICOT routine PMB03OD); a (sub)matrix whose estimated condition number is less than 1/TOL is considered to be of full rank. If the user sets TOL ≤ 0, then an implicitly computed, default tolerance, defined by

TOLDEF = $n * n*$EPS$*$max(norm(sub(A)), norm(sub(B))), is used instead, where EPS is the machine precision (see ScaLAPACK Library routine PDLAMCH).

3.4 Workspace

IWORK INTEGER array, dimension max(LOCc(JA+N-1), LOCc(JB+M-1))

DWORK DOUBLE PRECISION array, dimension (LDWORK)
On exit, if INFO = 0, DWORK(1) returns the optimal value of LDWORK.

LDWORK INTEGER
The length of the array DWORK.
LDWORK $\geq$ max((NB_A*(NB_A-1))/2, (NqA0 + max(NpB0 + NUMROC(NUMROC(N+ICOFFA, NB_B, 0, 0, NPCOL), NB_B, 0, 0, LCMQ), MpA0))*NB_B) + NB_B * NB_B + NpB0 * MqB0
where LCMQ = LCM / NPCOL with LCM = ICLM(NPROW, NPCOL),
IROFFB = MOD(IB-1, MB_B), ICOFFB = MOD(JB-1, NB_B),
IBROW = INDXG2P(IB, MB_B, MYROW, RSRC_B, NPROW),
IBCOL = INDXG2P(JB, NB_B, MYCOL, CSRC_B, NPCOL),
NpB0 = NUMROC(N+IROFFB, MB_B, MYROW, IBROW, NPROW),
MqB0 = NUMROC(M+ICOFFB, MB_B, MYCOL, IBCOL, NPROW),

IROFFA = MOD(IA-1, MB_A), ICOFFA = MOD(JA-1, NB_A),
IAROW = INDXG2P(IA, MB_A, MYROW, RSRC_A, NPROW),
IACOL = INDXG2P(JA, NB_A, MYCOL, CSRC_A, NPCOL),
NpA0 = NUMROC(N+IROFFA, MB_A, MYROW, IAROW, NPROW),
NqA0 = NUMROC(N+ICOFFA, NB_A, MYCOL, IACOL, NPCOL),

ILCM, INDXG2P and NUMROC are ScaLAPACK tool functions; MYROW, MYCOL, NPROW and NPCOL can be determined by calling the subroutine BLACS_GRIDINFO.
If LDWORK=-1, then LDWORK is global input and a workspace query is assumed; the routine only calculates the minimum and optimal size for all work arrays. Each of these values is returned in the first entry of the corresponding work array, and no error message is issued by PXERBLA.

3.5 Warning Indicator

None.

3.6 Error Indicator

INFO INTEGER
=0: successful exit;

<0: If the i^{th} argument is an array and the j^{th} entry had an illegal value, then INFO = -(100*i+j), if the i^{th} argument is a scalar and had an illegal value, then INFO = -i.

4 Method

Matrix sub(B) is first QR-decomposed and the appropriate orthogonal similarity transformation applied to the matrix sub(A). Leaving the first rank(sub(B)) states unchanged, the remaining lower left block of sub(A) is then QR-decomposed and the new orthogonal matrix, Q_1, is also applied to the right of sub(A) to complete the similarity transformation. By continuing in this manner, a completely controllable state-space pair (A_{cont}, B_{cont}) is found for the given (sub(A), sub(B)), where A_{cont} is upper block Hessenberg with each subdiagonal block of full row, and B_{cont} is zero apart from its (independent) first rank(sub(B)) rows. Note that the system controllability indices are easily calculated from the dimensions of the blocks of A_{cont}.

5 References

[1] Van Dooren, P.M.
The computation of Kronecker's canonical form of a singular pencil.
Linear Algebra & Its Appl., Vol. 27, pp. 103-141, 1979.

[2] Quintana-Ortí, E.S., Quintana, G. and Hernández, V.
Parallel Algorithms for the Block Hessenberg Form and Applications.
Preprints of the 4th IFAC Workshop on Algorithms and Architectures for Real-Time Control, Vilamoura, Portugal, pp. 353-358, 9-11 April 1997.

[3] Konstantinov, M.M., Petkov, P.Hr. and Christov, N.D.
Orthogonal Invariants and Canonical Forms for Linear Controllable Systems.
Proc. 8th IFAC World Congress, Kyoto, 1, pp. 49-54, 1981.

[4] Alvarruiz, F., Hernández, V., Ruiz, P.A. and Vidal, A.M.
Computing Minimal Realizations of Control Linear Systems. An Approach Based on Distributed Memory Algorithms
3rd Portuguese Conference on Automatic Control, September 1998.

6 Numerical Aspects

The sequential algorithm requires $O(n^3)$ operations and the transformations that are applied are backward stable. This parallel version requires $O(n^3)/p$ operations per process and it has a communication cost of $O(n^2)$.

7 Parallel Execution Recomendations

Due to the actual matrix alignment restrictions of some subroutines of ScaLAPACK (v1.5), it is not possible to take any block size (NB). In general, it is a good solution to take a divisor of M (columns of B) as NB. Anyway, it is foreseeable that these restrictions disappear in new ScaLAPACK versions.

For moderate sizes, two-dimensional landscape distributions give the better performance. As the size of the problem increases, square decompositions become more efficient.

8 Further Comments

None.

9 Example

9.1 Program Text

```
*     PAB01ND EXAMPLE PROGRAM TEXT
*     RELEASE 1.0.
*
*     .. Parameters ..
      INTEGER          NIN, NOUT, NMAX, MMAX, LDAUX, LIWORK, LDWORK_MAX
      PARAMETER        ( NIN = 5, NOUT = 6, NMAX = 20, MMAX = 20,
     $                   LDAUX = NMAX, LIWORK = MMAX,
     $                   LDWORK_MAX = 3*NMAX*MMAX )
      INTEGER          DLEN_
      PARAMETER        ( DLEN_ = 9 )
*     .. Local Scalars ..
      DOUBLE PRECISION TOL
      INTEGER          I, INFO, INDCON, J, M, N, NCONT, ICTXT, NPROW,
     $                 NPCOL, MYROW, MYCOL, NPROCS, IAM, IERROR, NB, LD,
     $                 LDWORK
      LOGICAL          FIRSTP
      CHARACTER*1      JOBZ
*     .. Local Arrays ..
      DOUBLE PRECISION A(NMAX*NMAX), B(NMAX*MMAX), DWORK(LDWORK_MAX),
     $                 TAU(NMAX), Z(NMAX*NMAX), AUX(LDAUX,NMAX)
      INTEGER          IWORK(LIWORK), NBLK(NMAX), COM(4), DESCA(DLEN_),
     $                 DESCB(DLEN_), DESCZ(DLEN_), DESCAUX(DLEN_)
*     .. External Functions ..
      LOGICAL           LSAME
      INTEGER           NUMROC
      EXTERNAL          LSAME, NUMROC
*     .. External Subroutines ..
      EXTERNAL         PAB01ND, PDORGQR, MPI_INIT, BLACS_PINFO, IGEBS2D,
     $                 BLACS_GET, BLACS_GRIDINFO, BLACS_GRIDINIT,
     $                 MPI_FINALIZE, DGEBS2D, IGEBR2D, DGEBR2D, DESCINIT
*     .. Intrinsic Functions ..
      INTRINSIC        MAX, CHAR, ICHAR, REAL, INT
*     .. Executable Statements ..
*
*     Initialize MPI and obtain the number of process and my process
*     number.
*
      CALL MPI_INIT( IERROR )
      CALL BLACS_PINFO( IAM, NPROCS )
*
*     Set up process grid that is as close to square as possible
*
```

```
      NPROW = INT( SQRT( REAL(NPROCS) ) )
      NPCOL = NPROCS / NPROW
*
*     Get default system context and define grid
*
      CALL BLACS_GET( -1, 0, ICTXT )
      CALL BLACS_GRIDINIT( ICTXT, "Row", NPROW, NPCOL )
      CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )
      IF( MYROW.LT.0 ) GO TO 80
      FIRSTP = ( MYROW.EQ.0 .AND. MYCOL.EQ.0 )
*
*     Only first process obtain data
*
      IF( FIRSTP ) THEN
         WRITE ( NOUT, FMT = 99999 )
*        Skip the heading in the data file and read the data.
         READ ( NIN, FMT = '()' )
         READ ( NIN, FMT = * ) N, M, NB, TOL, JOBZ
         IF ( N.LE.0 .OR. N.GT.NMAX ) THEN
            WRITE ( NOUT, FMT = 99990 ) N
            CALL MPI_FINALIZE( IERROR )
            STOP
         END IF
         IF ( M.LE.0 .OR. M.GT.MMAX ) THEN
            WRITE ( NOUT, FMT = 99989 ) M
            CALL MPI_FINALIZE( IERROR )
            STOP
         END IF
      END IF
*
*     Distribute data
*
      IF( FIRSTP ) THEN
         COM(1) = N
         COM(2) = M
         COM(3) = NB
         COM(4) = ICHAR( JOBZ )
         CALL IGEBS2D( ICTXT, "All", " ", 1, 4, COM, 1 )
         CALL DGEBS2D( ICTXT, "All", " ", 1, 1, TOL, 1 )
      ELSE
         CALL IGEBR2D( ICTXT, "All", " ", 1, 4, COM, 1, 0, 0 )
         N = COM(1)
         M = COM(2)
         NB = COM(3)
         JOBZ = CHAR( COM(4) )
```

```
         CALL DGEBR2D( ICTXT, "All", " ", 1, 1, TOL, 1, 0, 0 )
      END IF
*
*     Initialize the array descriptors for matrices A, B and Z
*
      LD = NUMROC( N, NB, MYROW, 0, NPROW )
      CALL DESCINIT( DESCA, N, N, NB, NB, 0, 0, ICTXT, LD, INFO )
      CALL DESCINIT( DESCB, N, M, NB, NB, 0, 0, ICTXT, LD, INFO )
      CALL DESCINIT( DESCZ, N, N, NB, NB, 0, 0, ICTXT, LD, INFO )
*
*     Read and distribute matrix A.
*
      IF( FIRSTP )
     $   READ ( NIN, * ) ( ( AUX(I,J), J = 1,N ), I = 1,N )
      CALL DESCINIT( DESCAUX, N, N, N, N, 0, 0, ICTXT, LDAUX, INFO )
      CALL PDGEMR2D( N, N, AUX, 1, 1, DESCAUX, A, 1, 1, DESCA, ICTXT )
*
*     Read and distribute matrix B.
*
      IF( FIRSTP )
     $   READ ( NIN, * ) ( ( AUX(I,J), I = 1,N ), J = 1,M )
      CALL DESCINIT( DESCAUX, N, M, N, M, 0, 0, ICTXT, LDAUX, INFO )
      CALL PDGEMR2D( N, M, AUX, 1, 1, DESCAUX, B, 1, 1, DESCB, ICTXT )
*
*     Obtain workspace required (LDWORK = -1)
*
      LDWORK = -1
      CALL PAB01ND( JOBZ, N, M, A, 1, 1, DESCA, B, 1, 1,
     $              DESCB, NCONT, INDCON, NBLK, Z, 1, 1,
     $              DESCZ, TAU, TOL, IWORK, DWORK, LDWORK, INFO )
      LDWORK = DWORK(1)
      IF( LDWORK.GT.LDWORK_MAX ) THEN
         WRITE ( NOUT, FMT = 99988 )
         CALL MPI_FINALIZE( IERROR )
         STOP
      END IF
*
*     Find a controllable ssr for the given system.
*
      CALL PAB01ND( JOBZ, N, M, A, 1, 1, DESCA, B, 1, 1,
     $              DESCB, NCONT, INDCON, NBLK, Z, 1, 1,
     $              DESCZ, TAU, TOL, IWORK, DWORK, LDWORK, INFO )
*
*    Reduce and print matrix A
*
```

```
      CALL DESCINIT( DESCAUX, N, N, N, N, 0, 0, ICTXT, LDAUX, INFO )
      CALL PDGEMR2D( N, N, A, 1, 1, DESCA, AUX, 1, 1, DESCAUX, ICTXT )
      IF( FIRSTP ) THEN
         IF ( INFO.NE.0 ) THEN
            WRITE ( NOUT, FMT = 99998 ) INFO
         ELSE
            WRITE ( NOUT, FMT = 99997 ) NCONT
            WRITE ( NOUT, FMT = 99996 )
            DO 20 I = 1, NCONT
               WRITE ( NOUT, FMT = 99995 ) ( AUX(I,J), J = 1,NCONT )
   20       CONTINUE
            WRITE ( NOUT, FMT = 99994 ) ( NBLK(I), I = 1,INDCON )
            WRITE ( NOUT, FMT = 99993 )
         END IF
      END IF
*
*     Reduce and print matrix B
*
      CALL DESCINIT( DESCAUX, N, M, N, M, 0, 0, ICTXT, LDAUX, INFO )
      CALL PDGEMR2D( N, M, B, 1, 1, DESCB, AUX, 1, 1, DESCAUX, ICTXT )
      IF( FIRSTP ) THEN
         IF ( INFO.NE.0 ) THEN
            WRITE ( NOUT, FMT = 99998 ) INFO
         ELSE
            DO 40 I = 1, NCONT
               WRITE ( NOUT, FMT = 99995 ) ( AUX(I,J), J = 1,M )
   40       CONTINUE
            WRITE ( NOUT, FMT = 99992 ) INDCON
         END IF
      END IF
*
*     Reduce and print matrix Z
*
      IF ( LSAME( JOBZ, 'F' ) )
     $   CALL PDORGQR( N, N, N, Z, 1, 1, DESCZ, TAU, DWORK, LDWORK,
     $                 INFO )
      IF ( LSAME( JOBZ, 'F' ).OR.LSAME( JOBZ, 'I' ) ) THEN
         CALL DESCINIT( DESCAUX, N, N, N, N, 0, 0, ICTXT, LDAUX, INFO )
         CALL PDGEMR2D( N, N, Z, 1, 1, DESCZ, AUX, 1, 1, DESCAUX,
     $                  ICTXT )
         IF( FIRSTP ) THEN
            WRITE ( NOUT, FMT = 99991 )
            DO 60 I = 1, N
               WRITE ( NOUT, FMT = 99995 ) ( AUX(I,J), J = 1,N )
   60       CONTINUE
```

```
         END IF
      END IF
   80 CALL MPI_FINALIZE( IERROR )
      STOP
*
99999 FORMAT (' PAB01ND EXAMPLE PROGRAM RESULTS',/1X)
99998 FORMAT (' INFO on exit from PAB01ND = ',I2)
99997 FORMAT (' The order of the controllable state-space representati',
     $       'on = ',I2)
99996 FORMAT (/' The transformed state dynamics matrix of a controllab',
     $       'le realization is ')
99995 FORMAT (20(1X,F9.4))
99994 FORMAT (/' and the dimensions of its diagonal blocks are ',
     $       /20(1X,I2))
99993 FORMAT (/' The transformed input/state matrix B of a controllabl',
     $       'e realization is ')
99992 FORMAT (/' The controllability index of the transformed system r',
     $       'epresentation = ',I2)
99991 FORMAT (/' The similarity transformation matrix Z is ')
99990 FORMAT (/' N is out of range.',/' N = ',I5)
99989 FORMAT (/' M is out of range.',/' M = ',I5)
99988 FORMAT (/' Not enought workspace ')
      END
```

9.2 Program Data
Example 1

```
 PAB01ND EXAMPLE PROGRAM
   3     2     2     0.0     I
  -1.0   0.0   0.0
  -2.0  -2.0  -2.0
  -1.0   0.0  -3.0
   1.0   0.0   0.0
   0.0   2.0   1.0
```

Example 2
Soon available at ftp://www-copa.dsic.upv.es/PSLICOT/Examples/example2.program

9.3 Program Results
Example 1

```
 PAB01ND EXAMPLE PROGRAM RESULTS

 The order of the controllable state-space representation =  2
```

```
The transformed state dynamics matrix of a controllable realization is
  -3.0000    2.2361
    .0000   -1.0000

and the dimensions of its diagonal blocks are
 2

The transformed input/state matrix B of a controllable realization is
    .0000   -2.2361
   1.0000     .0000

The controllability index of the transformed system representation =  1

The similarity transformation matrix Z is
    .0000    1.0000     .0000
   -.8944     .0000    -.4472
   -.4472     .0000     .8944
```

Example 2
Soon available at ftp://www-copa.dsic.upv.es/PSLICOT/Examples/example2.results

---

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, D. Sorenson, *LAPACK Users' Guide*, 2nd ed., Philadelphia, PA, 1995.

[2] J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, *A Proposal for a Set of Parallel Basic Linear Algebra Subprograms*, Lecture Notes in Computer Science, volume 1041, pp. 107–, 1996.

[3] J. Choi, J.J. Dongarra, R. Pozo, D.W. Walker, *ScaLAPACK: a scalable linear algebra library for distributed memory concurrent computers*, Mathematical Sciences Section, Oak Ridge National Laboratory, 1992.

[4] M.J. Denham, C.J. Benson, *Implementation and Documentation Standards for the Software Library in Control Engineering (SLICE)*, Kingston Polytechnic, Control Systems Research Group, Internal report 81/3, November 1981.

[5] J.J. Dongarra, R. Clint Whaley, *LAPACK Working Note 94: A User's Guide to the BLACS v1.0*, Department of Computer Science, University of Tennessee, UT-CS-95-281, 1995.

[6] J.J. Dongarra, J. Du Croz, S. Hammarling, R.J. Hanson, *An Extended Set of Fortran Basic Linear Algebra Subprograms*, ACM Transactions on Mathematical Software, 14, pp. 1–32, 1988.

[7] J.J. Dongarra, J. Du Croz, I. Duff, S. Hammarling, *A Set of Level 3 Basic Linear Algebra Subprograms*, ACM Transactions on Mathematical Software, 16, pp. 1–28, 1990.

[8] J. Du Croz, *Draft Guidelines for Library Development*, NAG Central Office, Oxford.

[9] C.L. Lawson, R.J. Hanson, D.R. Kincaid, F.T. Krogh, *Basic Linear Algebra Subprograms for Fortran Usage*, ACM Transaction on Mathematical Software, pp. 308 - 323, 1979.

[10] Numerical Algorithms Group (NAG), *NAGWare f77 Tools Unix 2.0*, NP2419.

[11] Numerical Algorithms Group (NAG), *NAGWare Gateway Generator 2.0*, NP2770.

[12] Working Group on Software (WGS), *SLICOT Implementation and Documentation Standards (version 1.0)*, WGS-Report 90-1, Eindhoven University of Technology, May 1990.

[13] Working Group on Software (WGS), *SLICOT Implementation and Documentation Standards*, WGS-Report 96-1, Eindhoven University of Technology, [12] updated: February 1998, ftp://wgs.esat.kuleuven.ac.be/pub/WGS/REPORTS/rep96-1.ps.Z.