PSBLAS-2.0 User's guide

A reference guide for the Parallel Sparse BLAS library

by Salvatore Filippone

and Alfredo Buttari

"Tor Vergata" University of Rome. April 24, 2006

Contents

1 Introduction

The PSBLAS library, developed with the aim to facilitate the parallelization of computationally intensive scientific applications, is designed to address parallel implementation of iterative solvers for sparse linear systems through the distributed memory paradigm. It includes routines for multiplying sparse matrices by dense matrices, solving block diagonal systems with triangular diagonal entries, preprocessing sparse matrices, and contains additional routines for dense matrix operations. The current implementation of PSBLAS addresses a distributed memory execution model operating with message passing. However, the overall design does not preclude different implementation paradigms, such as those based on a shared memory model.

The PSBLAS library is internally implemented in a mixture of Fortran 77 and Fortran 95 [**?**] programming languages. A similar approach has been advocated by a number of authors, e.g. [**?**]. Moreover, the Fortran 95 facilities for dynamic memory management and interface overloading greatly enhance the usability of the PSBLAS subroutines. In this way, the library can take care of runtime memory requirements that are quite difficult or even impossible to predict at implementation or compilation time. The following presentation of the PSBLAS library follows the general structure of the proposal for serial Sparse BLAS [**?**], which in its turn is based on the proposal for BLAS on dense matrices [**?**, **?**, **?**].

The applicability of sparse iterative solvers to many different areas causes some terminology problems because the same concept may be denoted through different names depending on the application area. The PSBLAS features presented in this section will be discussed mainly in terms of finite difference discretizations of Partial Differential Equations (PDEs). However, the scope of the library is wider than that: for example, it can be applied to finite element discretizations of PDEs, and even to different classes of problems such as nonlinear optimization, for example in optimal control problems.

The design of a solver for sparse linear systems is driven by many conflicting objectives, such as limiting occupation of storage resources, exploiting regularities in the input data, exploiting hardware characteristics of the parallel platform. To achieve an optimal communication to computation ratio on distributed memory machines it is essential to keep the *data locality* as high as possible; this can be done through an appropriate data allocation strategy. The choice of the preconditioner is another very important factor that affects efficiency of the implemented application. Optimal data distribution requirements for a given preconditioner may conflict with distribution requirements of the rest of the solver. Finding the optimal trade-off may be very difficult because it is application dependent. Possible solution to these problems and other important inputs to the development of the PSBLAS software package has come from an established experience in applying the PSBLAS solvers to computational fluid dynamics applications.

2 General overview

The PSBLAS library is designed to handle the implementation of iterative solvers for sparse linear systems on distributed memory parallel computers.

The system coefficient matrix A must be square; it may be real or complex, nonsymmetric, and its sparsity pattern needs not to be symmetric. The serial computation parts are based on the serial sparse BLAS, so that any extension made to the data structures of the serial kernels is available to the parallel version. The overall design and parallelization strategy have been influenced by the structure of the ScaLAPACK parallel library [**?**]. The layered structure of the PSBLAS library is shown in figure 1 ; lower layers of the library indicate an encapsulation relationship with upper layers. The ongoing discussion focuses on the Fortran 95 layer immediately below the application layer. The serial parts of the computation on each process are executed through calls to the serial sparse BLAS subroutines. In a similar way, the inter-process message exchanges are implemented through the Basic Linear Algebra Communication Subroutines (BLACS) library [**?**] that guarantees a portable and efficient communication layer. The Message Passing Interface code is encapsulated within the BLACS layer. However, in some cases, MPI routines are directly used either to improve efficiency or to implement communication patterns for which the BLACS package doesn't provide any method.

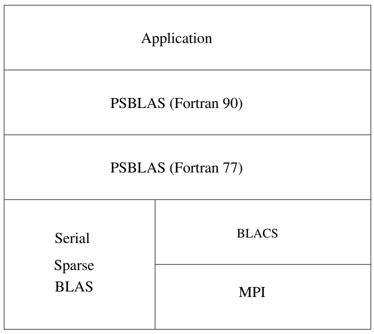

Figure 1: PSBLAS library components hierarchy.

The PSBLAS library consists of two classes of subroutines that is, the *computational routines* and the *auxiliary routines.* The computational routine set includes:

- Sparse matrix by dense matrix product;
- Sparse triangular systems solution for block diagonal matrices;
- Vector and matrix norms;
- Dense matrix sums;
- Dot products.

The auxiliary routine set includes:

- Communication descriptors allocation;
- Dense and sparse matrix allocation;
- Dense and sparse matrix build and update;

- Sparse matrix and data distribution preprocessing.

The following naming scheme has been adopted for all the symbols internally defined in the PSBLAS software package:

- all the symbols (i.e. subroutine names, data types...) are prefixed by `psb_`
- all the data type names are suffixed by `_type`
- all the constant values are suffixed by `_`
- all the subroutine names follow the rule `psb_xxname` where `xx` can be either:
  - `ge`: the routine is related to dense data,
  - `sp`: the routine is related to sparse data,
  - `cd`: the routine is related to communication descriptor (see 3).

  For example the `psb_geins`, `psb_spins` and `psb_cdins` perform the same action (see 6) on dense matrices, sparse matrices and communication descriptors respectively. Interface overloading allows the usage of the same subroutine interfaces for both real and complex data.

In the description of the subroutines, arguments or argument entries are classified as:

global For input arguments, the value must be the same on all processes participating in the subroutine call; for output arguments the value is guaranteed to be the same.

local Each process has its own value(s) independently.

3 Data Structures

In this chapter are illustrated data structures used for definition of routines interfaces. This include data structure for sparse matrix, communication descriptor and preconditioner. These data structures are used for calling PSBLAS routines in Fortran 90 language and will be used to next chapters containing these callings. Their definitions are included in the modules `psb_spmat_type`, `psb_descriptor_type` and `psb_prec_type`.

3.1 Sparse Matrix data structure

The `psb_spmat_type` data structure contains all information about local portion of the sparse matrix and its storage mode. Many of this fields are set in fully-transparent mode by PSBLAS-TOOLS routines when inserting a new sparse matrix, user must set only fields which describe matrix storage mode.
Fields contained in Sparse matrix structures are:

aspk Contains values of the local distributed sparse matrix.
Specified as: a pointer to an array of rank one of type corresponding to matrix entries type.

ia1 Holds integer information on distributed sparse matrix. Actual information will depend on data format used.
Specified as: a pointer to an integer array of rank one.

ia2 Holds integer information on distributed sparse matrix. Actual information will depend on data format used.
Specified as: a pointer to an integer array of rank one.

infoa On entry can hold auxiliary information on distributed sparse matrix. Actual information will depend on data format used.
Specified as: integer array of length `psb_ifasize_`.

fida Defines the format of the distributed sparse matrix.
Specified as: a string of length 5

descra Describe the characteristic of the distributed sparse matrix.
Specified as: array of character of length 9.

pl Specifies the local row permutation of distributed sparse matrix. If pl(1) is equal to 0, then there isn't row permutation.
Specified as: pointer to integer array of dimension equal to number of local row (matrix_data[psb_n_row_])

pr Specifies the local column permutation of distributed sparse matrix. If PR(1) is equal to 0, then there isn't columnm permutation.
Specified as: pointer to integer array of dimension equal to number of local row (matrix_data[psb_n_col_])

m Number of rows; if row indices are stored explicitly, as in Coordinate Storage, should be greater than or equal to the maximum row index actually present in the sparse matrix. Specified as: integer variable.

k Number of columns; if column indices are stored explicitly, as in Coordinate Storage or Compressed Sparse Rows, should be greater than or equal to the maximum column index actually present in the sparse matrix. Specified as: integer variable.

FORTRAN95 interface for distributed sparse matrices containing double precision real entries is defined as in figure 2.

```
type psb_dspmat_type
   integer     :: m, k
   character   :: fida(5)
   character   :: descra(10)
   integer     :: infoa(psb_ifa_size_)
   real(kind(1.d0)), pointer :: aspk(:)
   integer, pointer :: ia1(:), ia2(:), pr(:), pl(:)
end type psb_dspmat_type
```

Figure 2: The PSBLAS defined data type that contains a sparse matrix.

The following two cases are among the most commonly used:

fida="CSR" Compressed storage by rows. In this case the following should hold:

1. `ia2(i)` contains the index of the first element of row `i`; the last element of the sparse matrix is thus stored at index $ia2(m+1)-1$. It should contain `m+1` entries in nondecreasing order (strictly increasing, if there are no empty rows).
2. `ia1(j)` contains the column index and `aspk(j)` contains the corresponding coefficient value, for all $ia2(1) \leq j \leq ia2(m+1)-1$.

fida="COO" Coordinate storage. In this case the following should hold:

1. `infoa(1)` contains the number of nonzero elements in the matrix;
2. For all $1 \leq j \leq infoa(1)$, the coefficient, row index and column index are stored into `apsk(j)`, `ia1(j)` and `ia2(j)` respectively.

A sparse matrix has an associated state, which can take the following values:

Build: State entered after the first allocation, and before the first assembly; in this state it is possible to add nonzero entries.

Assembled: State entered after the assembly; computations using the sparse matrix, such as matrix-vector products, are only possible in this state;

Update: State entered after a reinitalization; this is used to handle applications in which the same sparsity pattern is used multiple times with different coefficients. In this state it is only possible to enter coefficients for already existing nonzero entries.

3.1.1 Named Constants

psb_nztotreq_ Request to fetch the total number of nonzeroes stored in a sparse matrix

psb_nzrowreq_ Request to fetch the number of nonzeroes in a given row in a sparse matrix

psb_dupl_ovwrt_ Duplicate coefficients should be overwritten (i.e. ignore duplications)

psb_dupl_add_ Duplicate coefficients should be added;

psb_dupl_err_ Duplicate coefficients should trigger an error conditino

psb_upd_dflt_ Default update strategy for matrix coefficients;

psb_upd_srch_ Update strategy based on search into the data structure;

psb_upd_perm_ Update strategy based on additional permutation data (see tools routine description).

3.2 Descriptor data structure

All the general matrix informations and elements to be exchanged among processes are stored within a data structure of the type `psb_desc_type`. Every structure of this type is associated to a sparse matrix, it contains data about general matrix informations and elements to be exchanged among processes.
It is not necessary for the user to know the internal structure of `psb_desc_type`, it is set in fully-transparent mode by PSBLAS-TOOLS routines when inserting a new sparse matrix, however the definition of the descriptor is the following.

matrix_data includes general information about matrix and BLACS grid. More precisely:

- **matrix_data[psb_dec_type_]** Identifies the decomposition type (global); the actual values are internally defined, so they should never be accessed directly.
- **matrix_data[psb_ctxt_]** Communication context as returned by the BLACS (global).
- **matrix_data[psb_m_]** Total number of equations (global).
- **matrix_data[psb_n_]** Total number of variables (global).
- **matrix_data[psb_n_row_]** Number of grid variables owned by the current process (local); equivalent to the number of local rows in the sparse coefficient matrix.
- **matrix_data[psb_n_col_]** Total number of grid variables read by the current process (local); equivalent to the number of local columns in the sparse coefficient matrix. They include the halo.

Specified as: a pointer to integer array of dimension 10.

halo_index A list of the halo and boundary elements for the current process to be exchanged with other processes; for each processes with which it is necessary to communicate:

1. Process identifier;
2. Number of points to be received;
3. Indices of points to be received;
4. Number of points to be sent;
5. Indices of points to be sent;

The list may contain an arbitrary number of groups; its end is marked by a -1.
Specified as: a pointer to an integer array of rank one.

ovrlap_index A list of the overlap elements for the current process, organized in groups like the previous vector:

1. Process identifier;
2. Number of points to be received;
3. Indices of points to be received;
4. Number of points to be sent;
5. Indices of points to be sent;

The list may contain an arbitrary number of groups; its end is marked by a -1.
Specified as: a pointer to an integer array of rank one.

ovrlap_index For all overlap points belonging to th ecurrent process:

1. Overlap point index;
2. Number of processes sharing that overlap points;

The list may contain an arbitrary number of groups; its end is marked by a -1.
Specified as: a pointer to an integer array of rank one.

loc_to_glob each element i of this array contains global identifier of the local variable i.
Specified as: a pointer to an integer array of rank one.

glob_to_loc if global variable i is read by current process then element i contains local index correpondent to global variable i; else element i contains -1 (NULL) value.
Specified as: a pointer to an integer array of rank one.

FORTRAN95 interface for `psb_desc_type` structures is therefore defined as follows:

A communication descriptor associated with a sparse matrix has a state, which can take the following values:

```
type psb_desc_type
   integer, pointer :: matrix_data(:), halo_index(:)
   integer, pointer :: overlap_elem(:), overlap_index(:)
   integer, pointer :: loc_to_glob(:), glob_to_loc(:)
end type psb_desc_type
```

Figure 3: The PSBLAS defined data type that contains the communication descriptor.

Build: State entered after the first allocation, and before the first assembly; in this state it is possible to add communication requirements among different processes.

Assembled: State entered after the assembly; computations using the associated sparse matrix, such as matrix-vector products, are only possible in this state.

3.2.1 Named Constants

psb_none_ Generic no-op;

psb_nohalo_ Do not fetch halo elements;

psb_halo_ Fetch halo elements from neighbouring processes;

psb_sum_ Sum overlapped elements

psb_avg_ Average overlapped elements

psb_dec_type_ Entry holding decomposition type (in `desc_a%matrix_data`)

psb_m_ Entry holding total number of rows

psb_n_ Entry holding total number of columns

psb_n_row_ Entry holding the number of rows stored in the current process

psb_n_col_ Entry holding the number of columns stored in the current process

psb_ctxt_ Entry holding a copy of the BLACS communication context

psb_desc_asb_ State of the descriptor: assembled, i.e. suitable for computational tasks.

psb_desc_bld_ State of the descriptor: build, must be assembled before computational use.

3.3 Preconditioner data structure

PSBLAS-2.0 offers the possibility to use many different types of preconditioning schemes. Besides the simple well known preconditioners like Diagonal Scaling or Block Jacobi (with ILU(0) incomplete factorization) also more complex preconditioning methods are implemented like the Additive Schwarz and Two-Level ones. A preconditioner is held in the `psb_prec_type` data structure which depends on the `psb_base_prec` reported in figure 4. The `psb_base_prec` data type may contain a simple preconditioning matrix with the associated communication descriptor which may be different than the system communication descriptor in the case of parallel preconditioners like the Additive Schwarz one. Then the `psb_prec_type` may contain more than one preconditioning matrix like in the case of Two-Level (in general Multi-Level) preconditioners. The user can choose the type of preconditioner to be used by means of the `psb_precset` subroutine; once the type of preconditioning method is specified, along with all the parameters that characterize it, the preconditioner data structure can be built using the `psb_precbld` subroutine. This data structure wants to be flexible enough to easily allow the implementation of new kind of preconditioners. The values contained in the `iprcparm` and `dprcparm` define tha type of preconditioner along with all the parameters related to it; thus, `iprcparm` and `dprcparm` define how the other records have to be interpreted.

```
  type psb_base_prec

    type(psb_spmat_type), pointer  :: av(:) => null()
    real(kind(1.d0)), pointer      :: d(:)  => null()
    type(psb_desc_type), pointer   :: desc_data => null()
    integer, pointer               :: iprcparm(:) => null()
    real(kind(1.d0)), pointer      :: dprcparm(:) => null()
    integer, pointer               :: perm(:)  => null()
    integer, pointer               :: mlia(:)  => null()
    integer, pointer               :: invperm(:) => null()
    integer, pointer               :: nlaggr(:)  => null()
    type(psb_spmat_type), pointer  :: aorig    => null()
    real(kind(1.d0)), pointer      :: dorig(:) => null()

 end type psb_base_prec

  type psb_prec_type
    type(psb_base_prec), pointer  :: baseprecv(:) => null()
    integer                       :: prec, base_prec
 end type psb_prec_type
```

Figure 4: The PSBLAS defined data type that contains a preconditioner.

3.3.1 Named Constants

f_ilu_n_ Incomplete LU factorization with n levels of fill-in; currently only $n = 0$ is implemented;

f_slu_ Sparse factorization using SuperLU;

f_umf_ Sparse factorization using UMFPACK;

add_ml_prec_ Additive multilevel correction;

mult_ml_prec_ Multiplicative multilevel correction;

pre_smooth_ Pre-smoothing in applying multiplicative multilevel corrections;

post_smooth_ Post-smoothing in applying multiplicative multilevel corrections;

smooth_both_ Two-sided (i.e. symmetric) smoothing in applying multiplicative multilevel corrections;

mat_distr_ Coarse matrix distributed among processes

mat_repl_ Coarse matrix replicated among processes

4 Algebraic routines

psb_geaxpby—General Dense Matrix Sum

This subroutine is an interface to the computational kernel for dense matrix sum:

$$y \leftarrow \alpha\, x + \beta y$$

Syntax

call psb_geaxpby (*alpha, x, beta, y, desc_a, info*)

x, y, α, β	**Subroutine**
Long Precision Real	psb_geaxpby
Long Precision Complex	psb_geaxpby

Table 1: Data types

On Entry

alpha the scalar α.
Scope: **global**
Type: **required**
Specified as: a number of the data type indicated in Table 1.

x the local portion of global dense matrix x.
Scope: **local**
Type: **required**
Specified as: a rank one or two array containing numbers of type specified in Table 1. The rank of x must be the same of y.

beta the scalar β.
Scope: **global**
Type: **required**
Specified as: a number of the data type indicated in Table 1.

y the local portion of the global dense matrix y.
Scope: **local**
Type: **required**
Specified as: a rank one or two array containing numbers of the type indicated in Table 1. The rank of y must be the same of x.

desc_a contains data structures for communications.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_desc_type`.

On Return

y the local portion of result submatrix y.

Scope: **local**

Type: **required**

Specified as: a rank one or two array containing numbers of the type indicated in Table 1.

info the local portion of result submatrix y.

Scope: **local**

Type: **required**

An integer value that contains an error code.

psb_gedot—Dot Product

This function computes dot product between two vectors x and y.
If x and y are double precision real vectors computes dot-product as:

$$dot \leftarrow x^T y$$

Else if x and y are double precision complex vectors then computes dot-product as:

$$dot \leftarrow x^H y$$

Syntax

psb_gedot (x, y, $desc\_a$, $info$)

dot, x, y	**Function**
Long Precision Real	psb_gedot
Long Precision Complex	psb_gedot

Table 2: Data types

On Entry

x the local portion of global dense matrix x.
Scope: **local**
Type: **required**
Specified as: an array of rank one or two containing numbers of type specified in Table 2. The rank of x must be the same of y.

y the local portion of global dense matrix y.
Scope: **local**
Type: **required**
Specified as: an array of rank one or two containing numbers of type specified in Table 2. The rank of y must be the same of x.

desc_a contains data structures for communications.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_desc_type`.

On Return

Function value is the dot product of subvectors x and y.
Scope: **global**
Specified as: a number of the data type indicated in Table 2.

info the local portion of result submatrix y.

Scope: **local**

Type: **required**

An integer value that contains an error code.

psb_gedot—Generalized Dot Product

This subroutine computes a series of dot products among the columns of two dense matrices x and y:

$$res(i) \leftarrow x(:,i)^T y(:,i)$$

If the matrices are complex, then the usual convention applies, i.e. the conjugate transpose of x is used. If x and y are of rank one, then res is a scalar, else it is a rank one array.

Syntax

psb_gedot (*res, x, y, desc_a, info*)

res, x, y	**Subroutine**
Long Precision Real	psb_gedot
Long Precision Complex	psb_gedot

Table 3: Data types

On Entry

x the local portion of global dense matrix x.
Scope: **local**
Type: **required**
Specified as: an array of rank one or two containing numbers of type specified in Table 3. The rank of x must be the same of y.

y the local portion of global dense matrix y.
Scope: **local**
Type: **required**
Specified as: an array of rank one or two containing numbers of type specified in Table 3. The rank of y must be the same of x.

desc_a contains data structures for communications.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_desc_type`.

On Return

res is the dot product of subvectors x and y.
Scope: **global**
Specified as: a number or a rank-one array of the data type indicated in Table 2.

info Scope: **local**
Type: **required**
An integer value that contains an error code.

psb_geamax—Infinity-Norm of Vector

This function computes the infinity-norm of a vector x.
If x is a double precision real vector computes infinity norm as:

$$amax \leftarrow \max_i |x_i|$$

else if x is a double precision complex vector then computes infinity-norm as:

$$amax \leftarrow \max_i (|re(x_i)| + |im(x_i)|)$$

Syntax

psb_geamax (x, *desc_a*, *info*)

$amax$	x	**Function**
Long Precision Real	Long Precision Real	psb_geamax
Long Precision Real	Long Precision Complex	psb_geamax

Table 4: Data types

On Entry

x the local portion of global dense matrix x.
Scope: **local**
Type: **required**
Specified as: a rank one or two array containing numbers of type specified in Table 4.

desc_a contains data structures for communications.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_desc_type`.

On Return

Function value is the infinity norm of subvector x.
Scope: **global**
Specified as: a long precision real number.

info Scope: **global**
Type: **required**
An integer value that contains an error code.

psb_geamax—Generalized Infinity Norm

This subroutine computes a series of infinity norms on the columns of a dense matrix x:

$$res(i) \leftarrow \max_k |x(k,i)|$$

Syntax

psb_geamax (*res, x, desc_a, info*)

res	x	**Subroutine**
Long Precision Real	Long Precision Real	psb_geamax
Long Precision Real	Long Precision Complex	psb_geamax

Table 5: Data types

On Entry

x the local portion of global dense matrix x.
Scope: **local**
Type: **required**
Specified as: a rank one or two array containing numbers of type specified in Table 5.

desc_a contains data structures for communications.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_desc_type`.

On Return

res is the infinity norm of the columns of x.
Scope: **global**
Specified as: a number or a rank-one array of long precision real numbers.

info Scope: **local**
Type: **required**
An integer value that contains an error code.

psb_geasum—1-Norm of Vector

This function computes the 1-norm of a vector x.
If x is a double precision real vector computes 1-norm as:

$$asum \leftarrow \|x_i\|$$

else if x ic double precision complex vector then computes 1-norm as:

$$asum \leftarrow \|re(x)\|_1 + \|im(x)\|_1$$

Syntax

psb_geasum (x, *desc_a*, *info*)

$asum$	x	**Function**
Long Precision Real	Long Precision Real	psb_geasum
Long Precision Real	Long Precision Complex	psb_geasum

Table 6: Data types

On Entry

x the local portion of global dense matrix x.
Scope: **local**
Type: **required**
Specified as: a rank one or two array containing numbers of type specified in Table 6.

desc_a contains data structures for communications.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_desc_type`.

On Return

Function value is the 1-norm of vector x.
Scope: **global**
Specified as: a long precision real number.

info Scope: **local**
Type: **required**
An integer value that contains an error code.

psb_genrm2—2-Norm of Vector

This function computes the 2-norm of a vector x.
If x is a double precision real vector computes 2-norm as:

$$nrm2 \leftarrow \sqrt{x^T x}$$

else if x is double precision complex vector then computes 2-norm as:

$$nrm2 \leftarrow \sqrt{x^H x}$$

$nrm2$	x	**Function**
Long Precision Real	Long Precision Real	psb_genrm2
Long Precision Real	Long Precision Complex	psb_genrm2

Table 7: Data types

Syntax

psb_genrm2 (x, *desc_a*, *info*)

On Entry

x the local portion of global dense matrix x.
Scope: **local**
Type: **required**
Specified as: a rank one or two array containing numbers of type specified in Table 7.

desc_a contains data structures for communications.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_desc_type`.

On Return

Function Value is the 2-norm of subvector x.
Scope: **global**
Type: **required**
Specified as: a long precision real number.

info Scope: **local**
Type: **required**
An integer value that contains an error code.

psb_spnrmi—Infinity Norm of Sparse Matrix

This function computes the infinity-norm of a matrix A:

$$nrmi \leftarrow \|A\|_\infty$$

where:

A represents the global matrix A

A	**Function**
Long Precision Real	psb_spnrmi
Long Precision Complex	psb_spnrmi

Table 8: Data types

Syntax

psb_spnrmi (A, $desc\_a$, $info$)

On Entry

a the local portion of the global sparse matrix A.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_spmat_type`.

desc_a contains data structures for communications.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_desc_type`.

On Return

Function value is the infinity-norm of sparse submatrix A.
Scope: **global**
Specified as: a long precision real number.

info Scope: **local**
Type: **required**
An integer value that contains an error code.

psb_spmm—Sparse Matrix by Dense Matrix Product

This subroutine computes the Sparse Matrix by Dense Matrix Product:

$$y \leftarrow \alpha P_r A P_c x + \beta y \tag{1}$$

$$y \leftarrow \alpha P_r A^T P_c x + \beta y \tag{2}$$

$$y \leftarrow \alpha P_r A^H P_c x + \beta y \tag{3}$$

where:

x is the global dense submatrix $x_{:,:}$

y is the global dense submatrix $y_{:,:}$

A is the global sparse submatrix A

P_r, P_c are the permutation matrices.

A, x, y, α, β	**Subroutine**
Long Precision Real	psb_spmm
Long Precision Complex	psb_spmm

Table 9: Data types

Syntax

CALL psb_spmm (*alpha, a, x, beta, y, desc_a, info*)

CALL psb_spmm (*alpha, a, x, beta, y,desc_a, info, trans, work*)

On Entry

alpha the scalar α.
Scope: **global**
Type: **required**
Specified as: a number of the data type indicated in Table 9.

a the local portion of the sparse matrix A.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_spmat_type`.

x the local portion of global dense matrix x.
Scope: **local**
Type: **required**
Specified as: a rank one or two array containing numbers of type specified in Table 9. The rank of x must be the same of y.

beta the scalar β.
Scope: **global**
Type: **required**
Specified as: a number of the data type indicated in Table 9.

y the local portion of global dense matrix y.
Scope: **local**
Type: **required**
Specified as: a rank one or two array containing numbers of type specified in Table 9. The rank of y must be the same of x.

desc_a contains data structures for communications.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_desc_type`.

trans indicate what kind of operation to perform.

trans = N the operation is specified by equation 1

trans = T the operation is specified by equation 2

trans = C the operation is specified by equation 3

Scope: **global**
Type: **optional**
Default: $trans = N$
Specified as: a character variable.

work work array.
Scope: **local**
Type: **optional**
Specified as: a rank one array of the same type of x and y with the TARGET attribute.

On Return

y the local portion of result submatrix y.
Scope: **local**
Type: **required**
Specified as: an array of rank one or two containing numbers of type specified in Table 9.

info Scope: **local**
Type: **required**
An integer value that contains an error code.

psb_spsm—Triangular System Solve

This subroutine computes the Triangular System Solve:

$$
\begin{array}{rcl}
y & \leftarrow & \alpha P_r T^{-1} P_c x + \beta y \\
y & \leftarrow & \alpha D P_r T^{-1} P_c x + \beta y \\
y & \leftarrow & \alpha P_r T^{-1} P_c D x + \beta y \\
y & \leftarrow & \alpha P_r T^{-T} P_c x + \beta y \\
y & \leftarrow & \alpha D P_r T^{-T} P_c x + \beta y \\
y & \leftarrow & \alpha P_r T^{-T} P_c D x + \beta y \\
y & \leftarrow & \alpha P_r T^{-H} P_c x + \beta y \\
y & \leftarrow & \alpha D P_r T^{-H} P_c x + \beta y \\
y & \leftarrow & \alpha P_r T^{-H} P_c D x + \beta y
\end{array}
$$

where:

x is the global dense submatrix $x_{:,:}$

y is the global dense submatrix $y_{:,:}$

T is the global sparse block triangular submatrix T

D is the scaling diagonal matrix.

P_r, P_c are the permutation matrices.

Syntax

CALL psb_spsm (*alpha, t, x, beta, y, desc_a, info*)

CALL psb_spsm
(*alpha, t, x, beta, y, desc_a, info, trans, unit, choice, diag, work*)

T, x, y, D, α, β	**Subroutine**
Long Precision Real	psb_spsm
Long Precision Complex	psb_spsm

Table 10: Data types

On Entry

alpha the scalar α.
Scope: **global**
Type: **required**
Specified as: a number of the data type indicated in Table 10.

t the global portion of the sparse matrix T.
Scope: **local**
Type: **required**
Specified as: a structured data type specified in § 3.

x the local portion of global dense matrix x.
Scope: **local**
Type: **required**
Specified as: a rank one or two array containing numbers of type specified in Table 10. The rank of x must be the same of y.

beta the scalar β.
Scope: **global**
Type: **required**
Specified as: a number of the data type indicated in Table 10.

y the local portion of global dense matrix y.
Scope: **local**
Type: **required**
Specified as: a rank one or two array containing numbers of type specified in Table 10. The rank of y must be the same of x.

desc_a contains data structures for communications.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_desc_type`.

trans specify with *unitd* the operation to perform.

trans = 'N' the operation is with no transposed matrix

trans = 'T' the operation is with transposed matrix.

trans = 'C' the operation is with conjugate transposed matrix.

Scope: **global**
Type: **optional**
Default: $trans = N$
Specified as: a character variable.

unitd specify with *trans* the operation to perform.

unitd = 'U' the operation is with no scaling

unitd = 'L' the operation is with left scaling

unitd = 'R' the operation is with right scaling.

Scope: **global**
Type: **optional**
Default: $unitd = U$
Specified as: a character variable.

choice specifies the update of overlap elements to be performed on exit:

`psb_none_`

`psb_sum_`

`psb_avg_`

`psb_square_root_`

Scope: **global**
Type: **optional**
Default: `psb_avg_`
Specified as: an integer variable.

diag the diagonal scaling matrix.
Scope: **local**
Type: **optional**
Default: $diag(1) = 1(noscaling)$
Specified as: a rank one array containing numbers of the type indicated in Table 10.

work a work array.
Scope: **local**
Type: **optional**
Specified as: a rank one array of the same type of x with the TARGET attribute.

On Return

y the local portion of global dense matrix y.
Scope: **local**
Type: **required**
Specified as: a pointer to array of rank one or two containing numbers of type specified in Table 10.

info Scope: **local**
Type: **required**
An integer value that contains an error code.

5 Communication routines

psb_halo—Halo Data Communication

These subroutines restore a consistent status for the halo elements, and (optionally) scale the result:

$$x \leftarrow \alpha x$$

where:

x is a global dense submatrix.

α, x	**Subroutine**
Long Precision Real	psb_halo
Long Precision Complex	psb_halo

Table 11: Data types

Syntax

CALL psb_halo (*x, desc_a, info*)

CALL psb_halo (*x, desc_a, info, alpha, work*)

On Entry

x global dense matrix x.
Scope: **local**
Type: **required**
Specified as: a rank one or two array with the TARGET attribute containing numbers of type specified in Table 11.

desc_a contains data structures for communications.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_desc_type`.

alpha the scalar α.
Scope: **global**
Type: **optional**
Default: $alpha = 1$
Specified as: a number of the data type indicated in Table 11.

work the work array.
Scope: **local**
Type: **optional**
Specified as: a rank one array of the same type of x with the POINTER attribute.

On Return

x global dense result matrix x.
Scope: **local**
Type: **required**
Returned as: a rank one or two array containing numbers of type specified in Table 11.

info the local portion of result submatrix y.
Scope: **local**
Type: **required**
An integer value that contains an error code.

psb_ovrl—Overlap Update

These subroutines restore a consistent status for the overlap elements:

$$x \leftarrow Qx$$

where:

x is the global dense submatrix x

Q is the overlap operator; it is the composition of two operators P_a and P^T.

x	**Subroutine**
Long Precision Real	psb_ovrl
Long Precision Complex	psb_ovrl

Table 12: Data types

Syntax

CALL psb_ovrl (*x, desc_a, info*)

CALL psb_ovrl (*x, desc_a, info, update=update_type, work=work*)

On Entry

x global dense matrix x.
Scope: **local**
Type: **required**
Specified as: a rank one or two array containing numbers of type specified in Table 12.

desc_a contains data structures for communications.
Scope: **local**
Type: **required**
Specified as: a structured data of type psb_desc_type.

update Update operator.

update = psb_none_ Do nothing;

update = psb_add_ Sum overlap entries, i.e. apply P^T;

update = psb_avg_ Average overlap entries, i.e. apply $P_a P^T$;

Scope: **global**
Default: *update_type = psb_avg_*
Scope: **global**
Specified as: a integer variable.

work the work array.
Scope: **local**
Type: **optional**
Specified as: a one dimensional array of the same type of x.

On Return

x global dense result matrix x.
Scope: **local**
Type: **required**
Specified as: an array of rank one or two containing numbers of type specified in Table 12.

info the local portion of result submatrix y.
Scope: **local**
Type: **required**
An integer value that contains an error code.

Usage notes

1. If there is no overlap in the data distribution, no operations are performed;

2. The operator P^T performs the reduction sum of overlap elements; it is a "prolongation" operator P^T that replicates overlap elements, accounting for the physical replication of data;

3. The operator P_a performs a scaling on the overlap elements by the amount of replication; thus, when combined with the reduction operator, it implements the average of replicated elements over all of their instances.

psb_gather—Gather Global Dense Matrix

These subroutines collect the portions of global dense matrix distributed over all process into one single array stored on one process.

$$glob\_x \leftarrow collect(loc\_x_i)$$

where:

$glob\_x$ is the global submatrix $glob\_x_{iy:iy+m-1,jy:jy+n-1}$

loc_x_i is the local portion of global dense matrix on process i.

$collect$ is the collect function.

x_i, y	**Subroutine**
Long Precision Real	psb_gather
Long Precision Complex	psb_gather

Table 13: Data types

Syntax

call psb_gather (*glob_x, loc_x, desc_a, info, root, iglobx, jglobx, ilocx, jlocx, k*)

Syntax

call psb_gather (*glob_x, loc_x, desc_a, info, root, iglobx, ilocx*)

On Entry

loc_x the local portion of global dense matrix *glob_x*.
Scope: **local**
Type: **required**
Specified as: a rank one or two array containing numbers of the type indicated in Table 13.

desc_a contains data structures for communications.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_desc_type`.

root The process that holds the global copy. If $root = -1$ all the processes will have a copy of the global vector.
Scope: **global**
Type: **optional**
Specified as: an integer variable $0 \leq ix \leq np$.

iglobx Row index to define a submatrix in glob_x into which gather the local pieces.
Scope: **global**
Type: **optional**
Specified as: an integer variable $1 \le ix \le matrix\_data(psb\_m\_)$.

jglobx Column index to define a submatrix in glob_x into which gather the local pieces.
Scope: **global**
Type: **optional**
Specified as: an integer variable.

ilocx Row index to define a submatrix in loc_x that has to be gathered into glob_x.
Scope: **local**
Type: **optional**
Specified as: an integer variable.

jlocx Columns index to define a submatrix in loc_x that has to be gathered into glob_x.
Scope: **global**
Type: **optional**
Specified as: an integer variable.

k The number of columns to gather.
Scope: **global**
Type: **optional**
Specified as: an integer variable.

On Return

glob_x The array where the local parts must be gathered.
Scope: **global**
Type: **required**
Specified as: a rank one or two array.

info the local portion of result submatrix y.
Scope: **local**
Type: **required**
An integer value that contains an error code.

psb_scatter—Scatter Global Dense Matrix

These subroutines scatters the portions of global dense matrix owned by a process to all the processes in the processes grid.

$$loc\_x_i \leftarrow scatter(glob\_x_i)$$

where:

glob_x is the global submatrix $glob\_x_{iy:iy+m-1,jy:jy+n-1}$

loc_x$_i$ is the local portion of global dense matrix on process i.

scatter is the scatter function.

x_i, y	**Subroutine**
Long Precision Real	psb_scatter
Long Precision Complex	psb_scatter

Table 14: Data types

Syntax

call psb_scatter (*glob_x, loc_x, desc_a, info, root, iglobx, jglobx, ilocx, jlocx, k*)

Syntax

call psb_scatter (*glob_x, loc_x, desc_a, info, root, iglobx, ilocx*)

On Entry

glob_x The array that must be scattered into local pieces.
Scope: **global**
Type: **required**
Specified as: a rank one or two array.

desc_a contains data structures for communications.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_desc_type`.

root The process that holds the global copy. If $root = -1$ all the processes have a copy of the global vector.
Scope: **global**
Type: **optional**
Specified as: an integer variable $0 \leq ix \leq np$.

iglobx Row index to define a submatrix in glob_x that has to be scattered into local pieces.
Scope: **global**
Type: **optional**
Specified as: an integer variable $1 \leq ix \leq matrix\_data(psb\_m\_)$.

jglobx Column index to define a submatrix in glob_x that has to be scattered into local pieces.
Scope: **global**
Type: **optional**
Specified as: an integer variable.

ilocx Row index to define a submatrix in loc_x into which scatter the local piece of glob_x.
Scope: **local**
Type: **optional**
Specified as: an integer variable.

jlocx Columns index to define a submatrix in loc_x into which scatter the local piece of glob_x.
Scope: **global**
Type: **optional**
Specified as: an integer variable.

k The number of columns to scatter.
Scope: **global**
Type: **optional**
Specified as: an integer variable.

On Return

loc_x the local portion of global dense matrix $glob\_x$.
Scope: **local**
Type: **required**
Specified as: a rank one or two array containing numbers of the type indicated in Table 14.

info the local portion of result submatrix y.
Scope: **local**
Type: **required**
An integer value that contains an error code.

6 Data management and initialization routines

psb_cdall—Allocates a communication descriptor

Syntax

call psb_cdall (*m, n, parts, icontxt, desc_a, info*)

call psb_cdall (*m, v, icontxt, desc_a, info, flag*)

This subroutine initializes the communication descriptor associated with an index space. It takes two forms depending on whether the user specifies the domain partitioning through a subroutine or through a vector

First Form: On Entry

m the number of rows of the problem.
Scope:**global**.
Type:**required**.
Specified as: an integer value.

n the number of columns of the problem.
Scope:**global**.
Type:**required**.
Specified as: an integer value. Currently constrained to be $m = n$.

parts the subroutine that defines the partitioning scheme.
Scope:**global**.
Type:**required**.
Specified as: a subroutine.

icontxt the communication context.
Scope:**global**.
Type:**required**.
Specified as: an integer value.

Second Form: On Entry

m the size of the index space.
Scope:**global**.
Type:**required**.
Specified as: an integer value $m > 0$.

v Data allocation: each index $i \in \{1 \dots m\}$ is allocated to process $v(i)$. Scope:**global**.
Type:**required**.
Specified as: an integer array of size m.

icontxt the communication context.
Scope:**global**.
Type:**required**.
Specified as: an integer value.

flag Specifies whether entries in v are zero- or one-based. Scope:**global**.
Type:**optional**.
Specified as: an integer value $0, 1$, default 0.

On Return

desc_a the communication descriptor.
Scope:**local**.
Type:**required**.
Specified as: a structured data of type `psb_desc_type`.

info Error code. Scope: **local**
Type: **required**
Specified as: an integer variable.

psb_cdins—Communication descriptor insert routine

Syntax

call psb_cdins (*nz, ia, ja, desc_a, info*)

On Entry

nz the number of points being inserted.
Scope: **local**.
Type: **required**.
Specified as: an integer value.

ia the row indices of the points being inserted.
Scope: **local**.
Type: **required**.
Specified as: an integer array of length nz.

ja the column indices of the points being inserted.
Scope: **local**.
Type: **required**.
Specified as: an integer array of length nz.

On Return

desc_a the communication descriptor to be freed.
Scope:**local**.
Type:**required**.
Specified as: a structured data of type `psb_desc_type`.

info Error code. Scope: **local**
Type: **required**
Specified as: an integer variable.

psb_cdasb—Communication descriptor assembly routine

Syntax

call psb_cdasb (*desc_a, info*)

On Entry

desc_a the communication descriptor.
Scope:**local**.
Type:**required**.
Specified as: a structured data of type `psb_desc_type`.

On Return

info Error code. Scope: **local**
Type: **required**
Specified as: an integer variable.

psb_cdcpy—Copies a communication descriptor

Syntax

call psb_cdcpy (*desc_out, desc_a, info*)

On Entry

desc_a the communication descriptor.
Scope:**local**.
Type:**required**.
Specified as: a structured data of type `psb_desc_type`.

On Return

desc_out the communication descriptor copy.
Scope:**local**.
Type:**required**.
Specified as: a structured data of type `psb_desc_type`.

info Error code. Scope: **local**
Type: **required**
Specified as: an integer variable.

psb_cdfree—Frees a communication descriptor

Syntax

call psb_cdfree (*desc_a, info*)

On Entry

desc_a the communication descriptor to be freed.
Scope:**local**.
Type:**required**.
Specified as: a structured data of type `psb_desc_type`.

On Return

info Error code. Scope: **local**
Type: **required**
Specified as: an integer variable.

psb_spall—Allocates a sparse matrix

Syntax

call psb_spall (*a, desc_a, info, nnz*)

On Entry

desc_a the communication descriptor.
Scope:**local**.
Type:**required**.
Specified as: a structured data of type `psb_desc_type`.

nnz the number of nonzeroes in the local part of the assembled matrix.
Scope: **global**.
Type: **optional**.
Specified as: an integer value. Note: a good estimate for the number of nonzeroes in the assembled matrix may substantially improve performance in the matrix build phase, as it will reduce or eliminate the need for multiple data allocation.

On Return

a the matrix to be allocated.
Scope:**local**
Type:**required**
Specified as: a structured data of type `psb_spmat_type`.

info Error code. Scope: **local**
Type: **required**
Specified as: an integer variable.

psb_spins—Insert a cloud of elements into a sparse matrix

Syntax

call psb_spins (*nz, ia, ja, val, a, desc_a, info*)

On Entry

nz the number of elements to be inserted.
Scope:**local**.
Type:**required**.
Specified as: an integer scalar.

ia the row indices of the elements to be inserted.
Scope:**local**.
Type:**required**.
Specified as: an integer array of size nz.

ja the column indices of the elements to be inserted.
Scope:**local**.
Type:**required**.
Specified as: an integer array of size nz.

val the elements to be inserted.
Scope:**local**.
Type:**required**.
Specified as: an array of size nz.

desc_a The communication descriptor.
Scope: **local**.
Type: **required**.
Specified as: a variable of type `psb_desc_type`.

On Return

a the matrix into which elements will be inserted.
Scope:**local**
Type:**required**
Specified as: a structured data of type `psb_spmat_type`.

desc_a The communication descriptor.
Scope: **local**.
Type: **required**.
Specified as: a variable of type `psb_desc_type`.

info Error code.
Scope: **local**
Type: **required**

psb_spasb—Sparse matrix assembly routine

Syntax

call psb_spasb (*a, desc_a, info, afmt, upd, dupl*)

On Entry

desc_a the communication descriptor.
Scope:**local**.
Type:**required**.
Specified as: a structured data of type `psb_desc_type`.

afmt the storage format for the sparse matrix.
Scope: **global**.
Type: **optional**.
Specified as: an array of characters. Defalt: 'CSR'.

upd Provide for updates to the matrix coefficients.
Scope: **global**.
Type: **optional**.
Specified as: integer, possible values: `psb_upd_srch_`, `psb_upd_perm_`

dupl How to handle duplicate coefficients.
Scope: **global**.
Type: **optional**.
Specified as: integer, possible values: `psb_dupl_ovwrt_`, `psb_dupl_add_`, `psb_dupl_err_`.

On Return

a the matrix to be assembled.
Scope:**local**
Type:**required**
Specified as: a structured data of type `psb_spmat_type`.

info Error code. Scope: **local**
Type: **required**
Specified as: an integer variable.

psb_spfree—Frees a sparse matrix

Syntax

call psb_spfree (*a, desc_a, info*)

On Entry

a the matrix to be freed.
Scope:**local**
Type:**required**
Specified as: a structured data of type `psb_spmat_type`.

desc_a the communication descriptor.
Scope:**local**.
Type:**required**.
Specified as: a structured data of type `psb_desc_type`.

On Return

info Error code. Scope: **local**
Type: **required**
Specified as: an integer variable.

psb_sprn—Reinit sparse matrix structure for psblas routines.

Syntax

call psb_sprn (*a, decsc_a, info*)

On Entry

a the matrix to be reinitialized.
Scope:**local**
Type:**required**
Specified as: a structured data of type `psb_spmat_type`.

desc_a the communication descriptor.
Scope:**local**.
Type:**required**.
Specified as: a structured data of type `psb_desc_type`.

On Return

info Error code. Scope: **local**
Type: **required**
Specified as: an integer variable.

psb_geall—Allocates a dense matrix

Syntax

call psb_geall (*x, desc_a, info, n*)

On Entry

desc_a The communication descriptor.
Scope: **local**
Type: **required**
Specified as: a variable of type `psb_desc_type`.

n The number of columns of the dense matrix to be allocated.
Scope: **local**
Type: **optional**
Specified as: Integer scalar, default 1. It is ignored if x is a rank-1 array.

On Return

x The dense matrix to be allocated.
Scope: **local**
Type: **required**
Specified as: a rank one or two array with the POINTER attribute, of type real, complex or integer.

info Error code. Scope: **local**
Type: **required**
Specified as: Integer scalar.

psb_geins—Dense matrix insertion routine

Syntax

call psb_geins (*m, n, blck, x, ix, jx, desc_a, info,dupl*)

call psb_geins (*m, blck, x, ix, desc_a, info,dupl*)

On Entry

m rows number of submatrix belonging to blck to be inserted..
Scope:**local**.
Type:**required**.
Specified as: an integer value.

n columns number of submatrix belonging to blck to be inserted (only when x is of rank 2).
Scope:**local**.
Type:**required**.
Specified as: an integer value.

blck the dense submatrix to be inserted.
Scope:**local**.
Type:**required**.
Specified as: a one or two dimensional array.

ix x global-row corresponding to position at which blck submatrix must be inserted.
Scope:**local**.
Type:**required**.
Specified as: an integer value.

jx x global-col corresponding to position at which blck submatrix must be inserted (only when x is of rank 2).
Scope:**local**.
Type:**required**.
Specified as: an integer value.

desc_a the communication descriptor.
Scope:**local**.
Type:**required**.
Specified as: a structured data of type `psb_desc_type`.

dupl How to handle duplicate coefficients.
Scope: **global**.
Type: **optional**.
Specified as: integer, possible values: `psb_dupl_ovwrt_`, `psb_dupl_add_`, `psb_dupl_err_`.

On Return

x the output dense matrix.
Scope: **local**
Type: **required**
Specified as: a rank one or two array with the POINTER attribute, of type real, complex or integer.

info Error code. Scope: **local**
Type: **required**
Specified as: an integer variable.

psb_geasb—Assembly a dense matrix

Syntax

call psb_geasb (*x, desc_a, info*)

On Entry

desc_a The communication descriptor.
Scope: **local**
Type: **required**
Specified as: a variable of type `psb_desc_type`.

On Return

x The dense matrix to be assembled.
Scope: **local**
Type: **required**
Specified as: a rank one or two array with the POINTER attribute, of type real, complex or integer.

info Error code.
Scope: **local**
Type: **required**
Specified as: Integer scalar.

psb_gefree—Frees a dense matrix

Syntax

call psb_gefree (*x, desc_a, info*)

On Entry

x The dense matrix to be freed.
Scope: **local**
Type: **required**
Specified as: a rank one or two array with the POINTER attribute, of type real, complex or integer.

desc_a The communication descriptor.
Scope: **local**
Type: **required**
Specified as: a variable of type `psb_desc_type`.

On Return

info Error code.
Scope: **local**
Type: **required**
Specified as: Integer scalar.

psb_gelp—Applies a left permutation to a dense matrix

Syntax

call psb_gelp (*trans, iperm, x, desc_a, info*)

On Entry

trans A character that specifies whether to permute A or A^T.
Scope: **local**
Type: **required**
Specified as: a single character with value 'N' for A or 'T' for A^T.

iperm An integer array containing permutation information.
Scope: **local**
Type: **required**
Specified as: an integer one-dimensional array.

x The dense matrix to be permuted.
Scope: **local**
Type: **required**
Specified as: a one or two dimensional array.

desc_a The communication descriptor.
Scope: **local**
Type: **required**
Specified as: a variable of type `psb_desc_type`.

On Return

info Error code.
Scope: **local**
Type: **required**
Specified as: Integer scalar.

psb_glob_to_loc—Global to local indices convertion

Syntax

call psb_glob_to_loc (x, y, $desc\_a$, $info$, $iact$)

call psb_glob_to_loc (x, $desc\_a$, $info$, $iact$)

On Entry

x An integer vector of indices to be converted.
Scope: **local**
Type: **required**
Specified as: a rank one integer array.

desc_a the communication descriptor.
Scope:**local**.
Type:**required**.
Specified as: a structured data of type `psb_desc_type`.

iact specifies action to be taken in case of range errors. Scope: **global**
Type: **optional**
Specified as: a character variable `E`, `W` or `A`.

On Return

x If y is not present, then x is overwritten with the translated integer indices.
Scope: **global**
Type: **required**
Specified as: a rank one integer array.

y If y is not present, then y is overwritten with the translated integer indices, and x is left unchanged. Scope: **global**
Type: **optional**
Specified as: a rank one integer array.

info Error code. Scope: **local**
Type: **required**
Specified as: an integer variable.

psb_loc_to_glob—Local to global indices conversion

Syntax

call psb_loc_to_glob (*x, y, desc_a, info, iact*)

call psb_loc_to_glob (*x, desc_a, info, iact*)

On Entry

x An integer vector of indices to be converted.
Scope: **local**
Type: **required**
Specified as: a rank one integer array.

desc_a the communication descriptor.
Scope:**local**.
Type:**required**.
Specified as: a structured data of type `psb_desc_type`.

iact specifies action to be taken in case of range errors. Scope: **global**
Type: **optional**
Specified as: a character variable `E`, `W` or `A`.

On Return

x If y is not present, then x is overwritten with the translated integer indices.
Scope: **global**
Type: **required**
Specified as: a rank one integer array.

y If y is not present, then y is overwritten with the translated integer indices, and x is left unchanged. Scope: **global**
Type: **optional**
Specified as: a rank one integer array.

info Error code. Scope: **local**
Type: **required**
Specified as: an integer variable.

7 Iterative Methods

In this chapter we provide routines for preconditioners and iterative methods. Their interfaces are defined in the module `psb_methd_mod`

psb_cg —CG Iterative Method

This subroutine implements the CG method with restarting. The stopping criterion is the normwise backward error, in the infinity norm, i.e. the iteration is stopped when

$$\frac{\|r\|}{(\|A\|\|x\| + \|b\|)} < eps$$

or

$$\frac{\|r_i\|}{\|b\|_2} < eps$$

according to the value passed through the istop argument (see later).

Syntax

call psb_cg (*a,prec,b,x,eps,desc_a,info,itmax,iter,err,itrace,istop*)

On Entry

a the local portion of global sparse matrix A.
Scope: **local**
Type: **required**
Specified as: a structured data of type psb_spmat_type.

prec The data structure containing the preconditioner.
Scope: **local**
Type: **required**
Specified as: a structured data of type psb_prec_type.

b The RHS vector.
Scope: **local**
Type: **required**
Specified as: a rank one array.

x The initial guess.
Scope: **local**
Type: **required**
Specified as: a rank one array.

eps The stopping tolerance.
Scope: **global**
Type: **required**
Specified as: a real number.

desc_a contains data structures for communications.
Scope: **local**
Type: **required**
Specified as: a structured data of type psb_desc_type.

itmax The maximum number of iterations to perform.
Scope: **global**
Type: **optional**
Default: $itmax = 1000$.
Specified as: an integer variable $itmax \geq 1$.

itrace If > 0 print out a convergence message every *itrace* iterations.
Scope: **global**
Type: **optional**

istop An integer specifying the stopping criterion.
Scope: **global**
Type: **optional**

On Return

x The computed solution.
Scope: **local**
Type: **required**
Specified as: a rank one array.

iter The number of iterations performed.
Scope: **global**
Type: **optional**
Returned as: an integer variable.

err The error estimate on exit.
Scope: **global**
Type: **optional**
Returned as: a real number.

info An error code.
Scope: **global**
Type: **optional**
Returned as: an integer variable.

psb_cgs —CGS Iterative Method

This subroutine implements the CGS method with restarting. The stopping criterion is the normwise backward error, in the infinity norm, i.e. the iteration is stopped when

$$\frac{\|r\|}{(\|A\|\|x\| + \|b\|)} < eps$$

or

$$\frac{\|r_i\|}{\|b\|_2} < eps$$

according to the value passed through the istop argument (see later).

Syntax

call psb_cgs (*a,prec,b,x,eps,desc_a,info,itmax,iter,err,itrace,istop*)

On Entry

a the local portion of global sparse matrix A.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_spmat_type`.

prec The data structure containing the preconditioner.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_prec_type`.

b The RHS vector.
Scope: **local**
Type: **required**
Specified as: a rank one array.

x The initial guess.
Scope: **local**
Type: **required**
Specified as: a rank one array.

eps The stopping tolerance.
Scope: **global**
Type: **required**
Specified as: a real number.

desc_a contains data structures for communications.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_desc_type`.

itmax The maximum number of iterations to perform.
Scope: **global**
Type: **optional**
Default: $itmax = 1000$.
Specified as: an integer variable $itmax \geq 1$.

itrace If > 0 print out a convergence message every *itrace* iterations.
Scope: **global**
Type: **optional**

istop An integer specifying the stopping criterion.
Scope: **global**
Type: **optional**

On Return

x The computed solution.
Scope: **local**
Type: **required**
Specified as: a rank one array.

iter The number of iterations performed.
Scope: **global**
Type: **optional**
Returned as: an integer variable.

err The error estimate on exit.
Scope: **global**
Type: **optional**
Returned as: a real number.

info An error code.
Scope: **global**
Type: **optional**
Returned as: an integer variable.

psb_bicg —BiCG Iterative Method

This subroutine implements the BiCG method with restarting. The stopping criterion is the normwise backward error, in the infinity norm, i.e. the iteration is stopped when

$$\frac{\|r\|}{(\|A\|\|x\| + \|b\|)} < eps$$

or

$$\frac{\|r_i\|}{\|b\|_2} < eps$$

according to the value passed through the istop argument (see later).

Syntax

call psb_bicg (*a,prec,b,x,eps,desc_a,info,itmax,iter,err,itrace,istop*)

On Entry

a the local portion of global sparse matrix A.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_spmat_type`.

prec The data structure containing the preconditioner.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_prec_type`.

b The RHS vector.
Scope: **local**
Type: **required**
Specified as: a rank one array.

x The initial guess.
Scope: **local**
Type: **required**
Specified as: a rank one array.

eps The stopping tolerance.
Scope: **global**
Type: **required**
Specified as: a real number.

desc_a contains data structures for communications.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_desc_type`.

itmax The maximum number of iterations to perform.
Scope: **global**
Type: **optional**
Default: $itmax = 1000$.
Specified as: an integer variable $itmax \geq 1$.

itrace If > 0 print out a convergence message every *itrace* iterations.
Scope: **global**
Type: **optional**

istop An integer specifying the stopping criterion.
Scope: **global**
Type: **optional**

On Return

x The computed solution.
Scope: **local**
Type: **required**
Specified as: a rank one array.

iter The number of iterations performed.
Scope: **global**
Type: **optional**
Returned as: an integer variable.

err The error estimate on exit.
Scope: **global**
Type: **optional**
Returned as: a real number.

info An error code.
Scope: **global**
Type: **optional**
Returned as: an integer variable.

psb_bicgstab —BiCGSTAB Iterative Method

This subroutine implements the BiCGSTAB method with restarting. The stopping criterion is the normwise backward error, in the infinity norm, i.e. the iteration is stopped when

$$\frac{\|r\|}{(\|A\|\|x\| + \|b\|)} < eps$$

or

$$\frac{\|r_i\|}{\|b\|_2} < eps$$

according to the value passed through the istop argument (see later).

Syntax

call psb_bicgstab (*a,prec,b,x,eps,desc_a,info,itmax,iter,err,itrace,istop*)

On Entry

a the local portion of global sparse matrix A.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_spmat_type`.

prec The data structure containing the preconditioner.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_prec_type`.

b The RHS vector.
Scope: **local**
Type: **required**
Specified as: a rank one array.

x The initial guess.
Scope: **local**
Type: **required**
Specified as: a rank one array.

eps The stopping tolerance.
Scope: **global**
Type: **required**
Specified as: a real number.

desc_a contains data structures for communications.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_desc_type`.

itmax The maximum number of iterations to perform.
Scope: **global**
Type: **optional**
Default: $itmax = 1000$.
Specified as: an integer variable $itmax \geq 1$.

itrace If > 0 print out a convergence message every *itrace* iterations.
Scope: **global**
Type: **optional**

istop An integer specifying the stopping criterion.
Scope: **global**
Type: **optional**

On Return

x The computed solution.
Scope: **local**
Type: **required**
Specified as: a rank one array.

iter The number of iterations performed.
Scope: **global**
Type: **optional**
Returned as: an integer variable.

err The error estimate on exit.
Scope: **global**
Type: **optional**
Returned as: a real number.

info An error code.
Scope: **global**
Type: **optional**
Returned as: an integer variable.

psb_bicgstabl —BiCGSTAB-l Iterative Method

This subroutine implements the BiCGSTAB-l method with restarting. The stopping criterion is the normwise backward error, in the infinity norm, i.e. the iteration is stopped when

$$\frac{\|r\|}{(\|A\|\|x\| + \|b\|)} < eps$$

or

$$\frac{\|r_i\|}{\|b\|_2} < eps$$

according to the value passed through the istop argument (see later).

Syntax

call psb_bicgstab (*a,prec,b,x,eps,desc_a,info,itmax,iter,err,itrace,irst,istop*)

On Entry

a the local portion of global sparse matrix A.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_spmat_type`.

prec The data structure containing the preconditioner.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_prec_type`.

b The RHS vector.
Scope: **local**
Type: **required**
Specified as: a rank one array.

x The initial guess.
Scope: **local**
Type: **required**
Specified as: a rank one array.

eps The stopping tolerance.
Scope: **global**
Type: **required**
Specified as: a real number.

desc_a contains data structures for communications.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_desc_type`.

itmax The maximum number of iterations to perform.
Scope: **global**
Type: **optional**
Default: $itmax = 1000$.
Specified as: an integer variable $itmax \geq 1$.

itrace If > 0 print out a convergence message every *itrace* iterations.
Scope: **global**
Type: **optional**

irst An integer specifying the restarting iteration.
Scope: **global**
Type: **optional**

istop An integer specifying the stopping criterion.
Scope: **global**
Type: **optional**

On Return

x The computed solution.
Scope: **local**
Type: **required**
Specified as: a rank one array.

iter The number of iterations performed.
Scope: **global**
Type: **optional**
Returned as: an integer variable.

err The error estimate on exit.
Scope: **global**
Type: **optional**
Returned as: a real number.

info An error code.
Scope: **global**
Type: **optional**
Returned as: an integer variable.

psb_gmres —GMRES Iterative Method

This subroutine implements the GMRES method with restarting. The stopping criterion is the normwise backward error, in the infinity norm, i.e. the iteration is stopped when

$$\frac{\|r\|}{(\|A\|\|x\| + \|b\|)} < eps$$

or

$$\frac{\|r_i\|}{\|b\|_2} < eps$$

according to the value passed through the istop argument (see later).

Syntax

call psb_gmres (*a,prec,b,x,eps,desc_a,info,itmax,iter,err,itrace,irst,istop*)

On Entry

a the local portion of global sparse matrix A.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_spmat_type`.

prec The data structure containing the preconditioner.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_prec_type`.

b The RHS vector.
Scope: **local**
Type: **required**
Specified as: a rank one array.

x The initial guess.
Scope: **local**
Type: **required**
Specified as: a rank one array.

eps The stopping tolerance.
Scope: **global**
Type: **required**
Specified as: a real number.

desc_a contains data structures for communications.
Scope: **local**
Type: **required**
Specified as: a structured data of type `psb_desc_type`.

itmax The maximum number of iterations to perform.
Scope: **global**
Type: **optional**
Default: $itmax = 1000$.
Specified as: an integer variable $itmax \geq 1$.

itrace If > 0 print out a convergence message every *itrace* iterations.
Scope: **global**
Type: **optional**

irst An integer specifying the restart iteration.
Scope: **global**
Type: **optional**

istop An integer specifying the stopping criterion.
Scope: **global**
Type: **optional**

On Return

x The computed solution.
Scope: **local**
Type: **required**
Specified as: a rank one array.

iter The number of iterations performed.
Scope: **global**
Type: **optional**
Returned as: an integer variable.

err The error estimate on exit.
Scope: **global**
Type: **optional**
Returned as: a real number.

info An error code.
Scope: **global**
Type: **optional**
Returned as: an integer variable.

8 Preconditioner routines

PSBLAS contains the implementation of many preconditioning techniques some of which are very flexible thanks to the presence of many parameters that is possible to adjust to fit the user's needs:

- Diagonal Scaling
- Block Jacobi with ILU(0) factorization
- Additive Schwarz with the Restricted Additive Schwarz and Additive Schwarz with Harmonic extensions;
- Two-Level Additive Schwarz; this is actually a family of preconditioners since there is the possibility to choose between many variants.

psb_precset—Sets the precodntioner type

Syntax

call psb_precset (*prec, ptype, iv, rs, rv, ierr*)

On Entry

prec Scope: **global**
Type: **required**
Specified as: e pronditioner data structure `psb_prec_type`.

ptype the type of preconditioner. Scope: **global**
Type: **required**
Specified as: a string.

iv integer parameters for the precondtioner. Scope: **global**
Type: **required**
Specified as: an integer array.

rs Scope:
Type:
Specified as: .

rv Scope:
Type:
Specified as: .

ierr Scope:
Type:
Specified as: .

psb_precbld—Builds a preconditioner

Syntax

call psb_precbld (*a, desc_a, prec, info, upd*)

On Entry

a the system sparse matrix. Scope: **global**
Type: **required**
Specified as: a sparse matrix data structure `psb_spmat_type`.

desc_a the problem communication descriptor. Scope: **global**
Type: **required**
Specified as: a communication descriptor data structure `psb_desc_type`.

upd Scope: **global**
Type: **optional**
Specified as: a character.

On Return

prec the precodntioner.
Scope: **global**
Type: **required**
Specified as: a precondtioner data structure `psb_prec_type`

info the return error code.
Scope: **local**
Type: **required**
Specified as: an integer.

psb_precaply—Preconditioner application routine

Syntax

call psb_precaply (*prec,x,y,desc_a,info,trans,work*)

Syntax

call psb_precaply (*prec,x,desc_a,info,trans*)

On Entry

prec the preconditioner. Scope: **global**
Type: **required**
Specified as: a preconditioner data structure `psb_prec_type`.

x the source vector. Scope: **global**
Type: **require**
Specified as: a double precision array.

desc_a the problem communication descriptor. Scope: **global**
Type: **required**
Specified as: a communication data structure `psb_desc_type`.

trans Scope:
Type: **optional**
Specified as: a character.

work an optional work space Scope: **local**
Type: **optional**
Specified as: a double precision array.

On Return

y the destination vector. Scope: **global**
Type: **required**
Specified as: a double precision array.

info the return error code.
Scope: **local**
Type: **required**
Specified as: an integer.

9 Error handling

The PSBLAS library error handling policy has been completely rewritten in version 2.0. The idea behind the design of this new error handling strategy is to keep error messages on a stack allowing the user to trace back up to the point where the first error message has been generated. Every routine in the PSBLAS-2.0 library has, as last non-optional argument, an integer `info` variable; whenever, inside the routine, en error is detected, this variable is set to a value corresponding to a specific error code. Then this error code is also pushed on the error stack and then either control is returned to the caller routine or the execution is aborted, depending on the users choice. At the time when the execution is aborted, an error message is printed on standard output with a level of verbosity than can be chosen by the user. If the execution is not aborted, then, the caller routine checks the value returned in the `info` variable and, if not zero, an error condition is raised. This process continues on all the levels of nested calls until the level where the user decides to abort the program execution.

Figure 5 shows the layout of a generic `psb_foo` routine with respect to the PSBLAS-2.0 error handling policy. It is possible to see how, whenever an error condition is detected, the `info` variable is set to the corresponding error code which is, then, pushed on top of the stack by means of the `psb_errpush`. An error condition may be directly detected inside a routine or indirectly checking the error code returned returned by a called routine. Whenever an error is encountered, after it has been pushed on stack, the program execution skips to a point where the error condition is handled; the error condition is handled either by returning control to the caller routine or by calling the `psb\_error` routine which prints the content of the error stack and aborts the program execution.

Figure 6 reports a sample error message generated by the PSBLAS-2.0 library. This error has been generated by the fact that the user has chosen the invalid "FOO" storage format to represent the sparse matrix. From this error message it is possible to see that the error has been detected inside the `psb_cest` subroutine called by `psb_spasb` ... by process 0 (i.e. the root process).

```
subroutine psb_foo(some args, info)
   ...
   if(error detected) then
      info=errcode1
      call psb_errpush('psb_foo', errcode1)
      goto 9999
   end if
   ...
   call psb_bar(some args, info)
   if(info .ne. zero) then
      info=errcode2
      call psb_errpush('psb_foo', errcode2)
      goto 9999
   end if
   ...
9999 continue
   if (err_act .eq. act_abort) then
     call psb_error(icontxt)
     return
   else
     return
   end if

end subroutine psb_foo
```

Figure 5: The layout of a generic `psb_foo` routine with respect to PSBLAS-2.0 error handling policy.

```
==========================================================
Process: 0.  PSBLAS Error (4010) in subroutine: df_sample
Error from call to subroutine mat dist
==========================================================
Process: 0.  PSBLAS Error (4010) in subroutine: mat_distv
Error from call to subroutine psb_spasb
==========================================================
Process: 0.  PSBLAS Error (4010) in subroutine: psb_spasb
Error from call to subroutine psb_cest
==========================================================
Process: 0.  PSBLAS Error (136) in subroutine: psb_cest
Format FOO is unknown
==========================================================
Aborting...
```

Figure 6: A sample PSBLAS-2.0 error message. Process 0 detected an error condition inside the psb_cest subroutine

psb_errpush—Pushes an error code onto the error stack

Syntax

call psb_errpush (*err_c, r_name, i_err, a_err*)

On Entry

err_c the error code
Scope: **local**
Type: **required**
Specified as: an integer.

r_name the soutine where the error has been caught.
Scope: **local**
Type: **required**
Specified as: a string.

i_err addional info for error code
Scope: **local**
Type: **optional**
Specified as: an integer array

a_err addional info for error code
Scope: **local**
Type: **optional**
Specified as: a string.

psb_error—Prints the error stack content and aborts execution

Syntax

call psb_error (*icontxt*)

On Entry

icontxt the communication context.
Scope: **global**
Type: **optional**
Specified as: an integer.

psb_set_errverbosity—Sets the verbosity of error messages.

Syntax

call psb_set_errverbosity (v)

On Entry

v the verbosity level
  Scope: **global**
  Type: **required**
  Specified as: an integer.

psb_set_erraction—Set the type of action to be taken upon error condition.

Syntax

call psb_set_erraction (*err_act*)

On Entry

err_act the type of action.
Scope: **global**
Type: **required**
Specified as: an integer.

psb_errcomm—Error communication routine

Syntax

call psb_errcomm (*icontxt, err*)

On Entry

icontxt the communication context.
Scope: **global**
Type: **required**
Specified as: an integer.

err the error code to be communicated
Scope: **global**
Type: **required**
Specified as: an integer.