scalapack_wrapper

NAME

scalapack_wrapper - a frovedis module provides user-friendly interfaces for commonly used scalapack routines in scientific applications like machine learning algorithms.

SYNOPSIS

```
#include <frovedis/matrix/scalapack_wrapper.hpp>
```

WRAPPER FUNCTIONS

int getrf (const `sliced_blockcyclic_matrix<T>`& m,
`lvec<int>`& ipiv)
int getri (const `sliced_blockcyclic_matrix<T>`& m,
const `lvec<int>`& ipiv)
int getrs (const `sliced_blockcyclic_matrix<T>`& m1,
const `sliced_blockcyclic_matrix<T>`& m2,
const `lvec<int>`& ipiv,
char trans = 'N')
void lacpy (const `sliced_blockcyclic_matrix<T>`& m1,
const `sliced_blockcyclic_matrix<T>`& m2,
char uplo = 'A')
int gesv (const `sliced_blockcyclic_matrix<T>`& m1,
const `sliced_blockcyclic_matrix<T>`& m2)
int gesv (const `sliced_blockcyclic_matrix<T>`& m1,
const `sliced_blockcyclic_matrix<T>`& m2,
`lvec<int>`& ipiv)
int gels (const `sliced_blockcyclic_matrix<T>`& m1,
const `sliced_blockcyclic_matrix<T>`& m2,
char trans = 'N')
int gesvd (const `sliced_blockcyclic_matrix<T>`& m,
`std::vector<T>`& sval)
int gesvd (const `sliced_blockcyclic_matrix<T>`& m,
`std::vector<T>`& sval,
const `sliced_blockcyclic_matrix<T>`& svec,
char vtype = 'L')
int gesvd (const `sliced_blockcyclic_matrix<T>`& m,
`std::vector<T>`& sval,
const `sliced_blockcyclic_matrix<T>`& lsvec,
const `sliced_blockcyclic_matrix<T>`& rsvec)

SPECIAL FUNCTIONS

`blockcyclic_matrix<T>` inv (const `sliced_blockcyclic_matrix<T>`& m)

DESCRIPTION

ScaLAPACK is a high-performance external library written in Fortran language. It provides rich set of linear algebra functionalities whose computation loads are parallelized over the available processes in a system and the user interfaces of this library is very detailed and complex in nature. It requires a strong understanding on each of the input parameters, along with some distribution concepts.

Frovedis provides a wrapper module for some commonly used ScaLAPACK subroutines in scientific applications like machine learning algorithms. These wrapper interfaces are very simple and user needs not to consider all the detailed distribution parameters. Only specifying the target vectors or matrices with some other parameters (depending upon need) are fine. At the same time, all the use cases of a ScaLAPACK routine can also be performed using Frovedis ScaLAPACK wrapper of that routine.

These wrapper routines are global functions in nature. Thus they can be called easily from within the "frovedis" namespace. As a distributed input matrix, they accept "`blockcyclic_matrix<T>`" or "`sliced_blockcyclic_matrix<T>`". "T" is a template type which can be either "float" or "double". The individual detailed descriptions can be found in the subsequent sections. Please note that the term "inout", used in the below section indicates a function argument as both "input" and "output".

Detailed Description

getrf (m, ipiv)

Parameters
m: A `blockcyclic_matrix<T>` or a `sliced_blockcyclic_matrix<T>` (inout)
ipiv: An empty object of the type `frovedis::lvec<int>` (output)

Purpose
It computes an LU factorization of a general M-by-N distributed matrix, "m" using partial pivoting with row interchanges.

On successful factorization, matrix "m" is overwritten with the computed L and U factors. Along with the input matrix, this function expects user to pass an empty object of the type "`frovedis::lvec<int>`" as a second argument, named as "ipiv" which would be updated with the pivoting information associated with input matrix "m" by this function while computing factors. This "ipiv" information will be useful in computation of some other functions (like getri, getrs etc.)

Return Value
On success, it returns the exit status of the scalapack routine itself. If any error occurs, it throws an exception explaining cause of the error.

getri (m, ipiv)

Parameters
m: A `blockcyclic_matrix<T>` or a `sliced_blockcyclic_matrix<T>` (inout)
ipiv: An object of the type `frovedis::lvec<int>` (input)

Purpose
It computes the inverse of a distributed square matrix using the LU factorization computed by getrf(). So in

order to compute inverse of a matrix, first compute it's LU factor (and ipiv information) using getrf() and then pass the factored matrix, "m" along with the "ipiv" information to this function.

On success, factored matrix "m" is overwritten with the inverse (of the matrix which was passed to getrf()) matrix. "ipiv" will be internally used by this function and will remain unchanged.

Return Value
On success, it returns the exit status of the scalapack routine itself. If any error occurs, it throws an exception explaining cause of the error.

getrs (m1, m2, ipiv, trans='N')

Parameters
m1: A `blockcyclic_matrix<T>` or a `sliced_blockcyclic_matrix<T>` (input)
m2: A `blockcyclic_matrix<T>` or a `sliced_blockcyclic_matrix<T>` (inout)
ipiv: An object of the type `frovedis::lvec<int>` (input)
trans: A character containing either 'N' or 'T' [Default: 'N'] (input/optional)

Purpose
It solves a real system of distributed linear equations, AX=B with a general distributed square matrix (A) using the LU factorization computed by getrf(). Thus before calling this function, it is required to obtain the factored matrix "m1" (along with "ipiv" information) by calling getrf().

If trans='N', the linear equation AX=B is solved.
If trans='T' the linear equation transpose(A)X=B (A'X=B) is solved.

The matrix "m2" should have number of rows >= the number of rows in "m1" and at least 1 column in it.

On entry, "m2" contains the distributed right-hand-side (B) of the equation and on successful exit it is overwritten with the distributed solution matrix (X).

Return Value
On success, it returns the exit status of the scalapack routine itself. If any error occurs, it throws an exception explaining cause of the error.

lacpy (m1, m2, uplo='A')

Parameters
m1: A `blockcyclic_matrix<T>` or a `sliced_blockcyclic_matrix<T>` (input)
m2: A `blockcyclic_matrix<T>` or a `sliced_blockcyclic_matrix<T>` (output)
uplo: A character containing either 'U', 'L' or 'A' [Default: 'A'] (input/optional)

Purpose
It copies a distributed M-by-N matrix, "m1" in another distributed M-by-N matrix, 'm2' (m2=m1). No communication is performed during this copy. Only local versions are copied in each other.

If uplo='U', only upper-triangular part of "m1" will be copied in upper-triangular part of "m2".
If uplo='L', only lower-triangular part of "m1" will be copied in lower-triangular part of "m2".
And if uplo='A', all part of "m2" will be copied in "m2".

This function expects a valid M-by-N distributed matrix "m2" to be passed as second argument which will be updated with the copy of "m2" on successful exit. Thus a user is needed to allocate the memory for "m2" and pass to this function before calling it. If dimension of "m2" is not matched with dimension of "m1" or "m2" is not allocated beforehand, this function will throw an exception.

Return Value
On success, it returns void. If any error occurs, it throws an exception explaining cause of the error.

gesv (m1, m2)

Parameters
m1: A `blockcyclic_matrix<T>` or a `sliced_blockcyclic_matrix<T>` (inout)
m2: A `blockcyclic_matrix<T>` or a `sliced_blockcyclic_matrix<T>` (inout)

Purpose
It solves a real system of distributed linear equations, AX=B with a general distributed square matrix, "m1" by computing it's LU factors internally. This function internally computes the LU factors and ipiv information using getrf() and then solves the equation using getrs().

The matrix "m2" should have number of rows >= the number of rows in "m1" and at least 1 column in it.

On entry, "m1" contains the distributed left-hand-side square matrix (A), "m2" contains the distributed right-hand-side matrix (B) and on successful exit "m1" is overwritten with it's LU factors, "m2" is overwritten with the distributed solution matrix (X).

Return Value
On success, it returns the exit status of the scalapack routine itself. If any error occurs, it throws an exception explaining cause of the error.

gesv (m1, m2, ipiv)

Parameters
m1: A `blockcyclic_matrix<T>` or a `sliced_blockcyclic_matrix<T>` (inout)
m2: A `blockcyclic_matrix<T>` or a `sliced_blockcyclic_matrix<T>` (inout)
ipiv: An empty object of the type `frovedis::lvec<int>` (output)

Purpose
The function serves the same purpose as explained in above version of gesv (with two parameters). Only difference is that this version accepts an extra parameter "ipiv" of the type `lvec<int>` which would be allocated and updated with the pivoting information computed during factorization of "m1". Along with the factored matrix, it might also be needed to know the associated pivot values. In that case, this version of gesv (with three parameters) can be used.

On entry, "m1" contains the distributed left-hand-side square matrix (A), "m2" contains the distributed right-hand-side matrix (B), and "ipiv" is an empty object. On successful exit "m1" is overwritten with it's LU factors, "m2" is overwritten with the distributed solution matrix (X), and "ipiv" is updated with the pivot values associated with factored matrix, "m1".

Return Value
On success, it returns the exit status of the scalapack routine itself. If any error occurs, it throws an exception explaining cause of the error.

gels (m1, m2, trans='N')

Parameters
m1: A `blockcyclic_matrix<T>` or a `sliced_blockcyclic_matrix<T>` (input)
m2: A `blockcyclic_matrix<T>` or a `sliced_blockcyclic_matrix<T>` (inout)
trans: A character containing either 'N' or 'T' [Default: 'N'] (input/optional)

Purpose
It solves overdetermined or underdetermined real linear systems involving an M-by-N distributed matrix (A) or its transpose, using a QR or LQ factorization of (A). It is assumed that distributed matrix (A) has full rank.

If trans='N' and M >= N: it finds the least squares solution of an overdetermined system.
If trans='N' and M < N: it finds the minimum norm solution of an underdetermined system.

If trans='T' and M >= N: it finds the minimum norm solution of an underdetermined system.
If trans='T' and M < N: it finds the least squares solution of an overdetermined system.

The matrix "m2" should have number of rows >= max(M,N) and at least 1 column.

On entry, "m1" contains the distributed left-hand-side matrix (A) and "m2" contains the distributed right-hand-side matrix (B). On successful exit, "m1" is overwritten with the QR or LQ factors and "m2" is overwritten with the distributed solution matrix (X).

Return Value
On success, it returns the exit status of the scalapack routine itself. If any error occurs, it throws an exception explaining cause of the error.

gesvd (m, sval)

Parameters
m: A `blockcyclic_matrix<T>` or a `sliced_blockcyclic_matrix<T>` (inout)
sval: An empty vector of the type `std::vector<T>` (output)

Purpose
It computes the singular value decomposition (SVD) of an M-by-N distributed matrix.

On entry "m" contains the distributed matrix whose singular values are to be computed, "sval" is an empty object of the type `std::vector<T>`. And on successful exit, the contents of "m" is destroyed (internally used as workspace) and "sval" is updated with the singular values in sorted oder, so that sval(i) >= sval(i+1).

Return Value
On success, it returns the exit status of the scalapack routine itself. If any error occurs, it throws an exception explaining cause of the error.

gesvd (m, sval, svec, vtype)

Parameters
m: A `blockcyclic_matrix<T>` or a `sliced_blockcyclic_matrix<T>` (inout)
sval: An empty vector of the type `std::vector<T>` (output)
svec: A `blockcyclic_matrix<T>` or a `sliced_blockcyclic_matrix<T>` (output)
vtype: A character value containing either 'L' or 'R' [Default: 'L'] (input/optional)

Purpose
It computes the singular value decomposition (SVD) of an M-by-N distributed matrix. Additionally, it also computes *left **or** right singular vectors*.

If vtype='L', "svec" will be updated with first min(M,N) columns of left singular vectors (stored columnwise). In that case "svec" should have at least M number of rows and min(M,N) number of columns.
If vtype='R', "svec" will be updated with first min(M,N) rows of right singular vectors (stored rowwise in transposed form). In that case "svec" should have at least min(M,N) number of rows and N number of columns.

This function expects that required memory would be allocated for the output matrix "svec" beforehand. If it is not allocated, an exception will be thrown.

On entry "m" contains the distributed matrix whose singular values are to be computed, "sval" is an empty object of the type `std::vector<T>`, "svec" is a valid sized (as explained above) distributed matrix.
And on successful exit, the contents of "m" is destroyed (internally used as workspace), "sval" is updated with the singular values in sorted oder, so that sval(i) >= sval(i+1) and "svec" is updated with the desired singular vectors (as explained above).

Return Value
On success, it returns the exit status of the scalapack routine itself. If any error occurs, it throws an exception explaining cause of the error.

gesvd (m, sval, lsvec, rsvec)

Parameters
m: A `blockcyclic_matrix<T>` or a `sliced_blockcyclic_matrix<T>` (inout)
sval: An empty vector of the type `std::vector<T>` (output)
lsvec: A `blockcyclic_matrix<T>` or a `sliced_blockcyclic_matrix<T>` (output)
rsvec: A `blockcyclic_matrix<T>` or a `sliced_blockcyclic_matrix<T>` (output)

Purpose
It computes the singular value decomposition (SVD) of an M-by-N distributed matrix. Additionally, it also computes *left **and** right singular vectors*.

This function expects that required memory would be allocated for the output matrices "lsvec" and "rsvec" beforehand, to store the left and right singular vectors respectively. If they are not allocated, an exception will be thrown.

Output matrix "lsvec" will be updated with first min(M,N) columns of left singular vectors (stored columnwise). Thus, "lsvec" should have at least M number of rowsand min(M,N) number of columns.
Output matrix "rsvec" will be updated with first min(M,N) rows of right singular vectors (stored rowwise in transposed form). Thus, "rsvec" should have at least min(M,N) number of rows and N number of columns.

On entry "m" contains the distributed matrix whose singular values are to be computed, "sval" is an empty object of the type `std::vector<T>`, "lsvec" and "rsvec" are valid sized (as explained above) distributed matrices. And on successful exit, the contents of "m" is destroyed (internally used as workspace), "sval" is updated with the singular values in sorted oder, so that sval(i) >= sval(i+1), "lsvec" and "rsvec" are updated with the left and right singular vectors respectively (as explained above).

Return Value
On success, it returns the exit status of the scalapack routine itself. If any error occurs, it throws an exception explaining cause of the error.

inv (m)

Parameters
m: A `blockcyclic_matrix<T>` or a `sliced_blockcyclic_matrix<T>` (input)

Purpose
It computes the inverse of a distributed square matrix "m" by using getrf() and getri() internally. Thus it is a kind of short-cut function to obtain the inverse of a distributed matrix.

On successful exit, it returns the resultant inversed matrix. The input matrix "m" remains unchanged. Since it returns the resultant matrix, it can be used in any numerical expressions, along with other operators. E.g., if a and b are two blockcyclic matrices, then the expresion like, "a*(~b)*inv(a)" can easily be performed.

Return Value
On success, it returns the resultant matrix of the type `blockcyclic_matrix<T>`. If any error occurs, it throws an exception explaining cause of the error.

SEE ALSO

sliced_blockcyclic_matrix_local, sliced_blockcyclic_vector_local, lapack_wrapper