pblas_wrapper

NAME

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

SYNOPSIS

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

WRAPPER FUNCTIONS

void swap (const `sliced_blockcyclic_vector<T>`& v1,
const `sliced_blockcyclic_vector<T>`& v2)
void copy (const `sliced_blockcyclic_vector<T>`& v1,
const `sliced_blockcyclic_vector<T>`& v2)
void scal (const `sliced_blockcyclic_vector<T>`& v,
T al)
void axpy (const `sliced_blockcyclic_vector<T>`& v1,
const `sliced_blockcyclic_vector<T>`& v2,
T al = 1.0)
T dot (const `sliced_blockcyclic_vector<T>`& v1,
const `sliced_blockcyclic_vector<T>`& v2)
T nrm2 (const `sliced_blockcyclic_vector<T>`& v)
void gemv (const `sliced_blockcyclic_matrix<T>`& m,
const `sliced_blockcyclic_vector<T>`& v1,
const `sliced_blockcyclic_vector<T>`& v2,
char trans = 'N',
T al = 1.0,
T be = 0.0)
void ger (const `sliced_blockcyclic_vector<T>`& v1,
const `sliced_blockcyclic_vector<T>`& v2,
const `sliced_blockcyclic_matrix<T>`& m,
T al = 1.0)
void gemm (const `sliced_blockcyclic_matrix<T>`& m1,
const `sliced_blockcyclic_matrix<T>`& m2,
const `sliced_blockcyclic_matrix<T>`& m3,
char trans_m1 = 'N',
char trans_m2 = 'N',
T al = 1.0,
T be = 0.0)

void geadd (const `sliced_blockcyclic_matrix<T>`& m1,
const `sliced_blockcyclic_matrix<T>`& m2,
char trans = 'N',
T al = 1.0,
T be = 1.0)

OVERLOADED OPERATORS

`blockcyclic_matrix<T>`
operator* (const `sliced_blockcyclic_matrix<T>`& m1,
const `sliced_blockcyclic_matrix<T>`& m2)
`blockcyclic_matrix<T>`
operator+ (const `sliced_blockcyclic_matrix<T>`& m1,
const `sliced_blockcyclic_matrix<T>`& m2)
`blockcyclic_matrix<T>`
operator~ (const `sliced_blockcyclic_matrix<T>`& m1)

DESCRIPTION

PBLAS is a high-performance external library written in Fortran language. It provides rich set of functionalities on vectors and matrices. The computation loads of these functionalities 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 PBLAS 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 PBLAS routine can also be performed using Frovedis PBLAS 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 vector, they accept "`blockcyclic_matrix<T>` with single column" or "`sliced_blockcyclic_vector<T>`". And 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

swap (v1, v2)

Parameters
v1: A `blockcyclic_matrix<T>` with single column or a `sliced_blockcyclic_vector<T>` (inout)
v2: A `blockcyclic_matrix<T>` with single column or a `sliced_blockcyclic_vector<T>` (inout)

Purpose
It will swap the contents of v1 and v2, if they are semantically valid and are of same length.

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

copy (v1, v2)

Parameters
v1: A `blockcyclic_matrix<T>` with single column or a `sliced_blockcyclic_vector<T>` (input)
v2: A `blockcyclic_matrix<T>` with single column or a `sliced_blockcyclic_vector<T>` (output)

Purpose
It will copy the contents of v1 in v2 (v2 = v1), if they are semantically valid and are of same length.

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

scal (v, al)

Parameters
v: A `blockcyclic_matrix<T>` with single column or a `sliced_blockcyclic_vector<T>` (inout)
al: A "T" typed argument (float or double) to specify the value to which the input vector needs to be scaled. (input)

Purpose
It will scale the input vector with the provided "al" value, if it is semantically valid. On success, input vector "v" would be updated (in-place scaling).

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

axpy (v1, v2, al=1.0)

Parameters
v1: A `blockcyclic_matrix<T>` with single column or a `sliced_blockcyclic_vector<T>` (input)
v2: A `blockcyclic_matrix<T>` with single column or a `sliced_blockcyclic_vector<T>` (inout)
al: A "T" typed argument (float or double) to specify the value to which "v1" needs to be scaled (not in-place scaling) [Default: 1.0] (input/optional)

Purpose
It will solve the expression v2 = al*v1 + v2, if the input vectors are semantically valid and are of same length. On success, "v2" will be updated with desired result, but "v1" would remain unchanged.

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

dot (v1, v2)

Parameters
v1: A `blockcyclic_matrix<T>` with single column or a `sliced_blockcyclic_vector<T>` (input)
v2: A `blockcyclic_matrix<T>` with single column or a `sliced_blockcyclic_vector<T>` (input)

Purpose
It will perform dot product of the input vectors, if they are semantically valid and are of same length. Input vectors would not get modified during the operation.

Return Value
On success, it returns the dot product result of the type "float" or "double". If any error occurs, it throws an exception.

nrm2 (v)

Parameters
v: A `blockcyclic_matrix<T>` with single column or a `sliced_blockcyclic_vector<T>` (input)

Purpose
It will calculate the norm of the input vector, if it is semantically valid. Input vector would not get modified during the operation.

Return Value
On success, it returns the norm value of the type "float" or "double". If any error occurs, it throws an exception.

gemv (m, v1, v2, trans='N', al=1.0, be=0.0)

Parameters
m: A `blockcyclic_matrix<T>` or a `sliced_blockcyclic_matrix<T>` (input)
v1: A `blockcyclic_matrix<T>` with single column or a `sliced_blockcyclic_vector<T>` (input)
v2: A `blockcyclic_matrix<T>` with single column or a `sliced_blockcyclic_vector<T>` (inout)
trans: A character value can be either 'N' or 'T' [Default: 'N'] (input/optional)
al: A "T" typed (float or double) scalar value [Default: 1.0] (input/optional)
be: A "T" typed (float or double) scalar value [Default: 0.0] (input/optional)

Purpose
The primary aim of this routine is to perform simple matrix-vector multiplication.
But it can also be used to perform any of the below operations:

```
(1) v2 = al*m*v1 + be*v2
(2) v2 = al*transpose(m)*v1 + be*v2
```

If trans='N', then expression (1) is solved. In that case, the size of "v1" must be at least the number of columns in "m" and the size of "v2" must be at least the number of rows in "m".
If trans='T', then expression (2) is solved. In that case, the size of "v1" must be at least the number of rows in "m" and the size of "v2" must be at least the number of columns in "m".

Since "v2" is used as input-output both, memory must be allocated for this vector before calling this routine, even if simple matrix-vector multiplication is required. Otherwise, this routine will throw an exception.

For simple matrix-vector multiplication, no need to specify values for the input parameters "trans", "al" and "be" (leave them at their default values).

On success, "v2" will be overwritten with the desired output. But "m" and "v1" would remain unchanged.

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

ger (v1, v2, m, al=1.0)

Parameters
v1: A `blockcyclic_matrix<T>` with single column or a `sliced_blockcyclic_vector<T>` (input)
v2: A `blockcyclic_matrix<T>` with single column or a `sliced_blockcyclic_vector<T>` (input)
m: A `blockcyclic_matrix<T>` or a `sliced_blockcyclic_matrix<T>` (inout)
al: A "T" typed (float or double) scalar value [Default: 1.0] (input/optional)

Purpose
The primary aim of this routine is to perform simple vector-vector multiplication of the sizes "a" and "b" respectively to form an axb matrix. But it can also be used to perform the below operations:

```
m = al*v1*v2' + m
```

This operation can only be performed if the inputs are semantically valid and the size of "v1" is at least the number of rows in matrix "m" and the size of "v2" is at least the number of columns in matrix "m".

Since "m" is used as input-output both, memory must be allocated for this matrix before calling this routine, even if simple vector-vector multiplication is required. Otherwise it will throw an exception.

For simple vector-vector multiplication, no need to specify the value for the input parameter "al" (leave it at its default value).

On success, "m" will be overwritten with the desired output. But "v1" and "v2" will remain unchanged.

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

gemm (m1, m2, m3, trans_m1='N', trans_m2='N', al=1.0, be=0.0)

Parameters
m1: A `blockcyclic_matrix<T>` or a `sliced_blockcyclic_matrix<T>` (input)
m2: A `blockcyclic_matrix<T>` or a `sliced_blockcyclic_matrix<T>` (input)
m3: A `blockcyclic_matrix<T>` or a `sliced_blockcyclic_matrix<T>` (inout)
trans_m1: A character value can be either 'N' or 'T' [Default: 'N'] (input/optional)
trans_m2: A character value can be either 'N' or 'T' [Default: 'N'] (input/optional)
al: A "T" typed (float or double) scalar value [Default: 1.0] (input/optional)
be: A "T" typed (float or double) scalar value [Default: 0.0] (input/optional)

Purpose
The primary aim of this routine is to perform simple matrix-matrix multiplication.
But it can also be used to perform any of the below operations:

```
(1) m3 = al*m1*m2 + be*m3
(2) m3 = al*transpose(m1)*m2 + be*m3
(3) m3 = al*m1*transpose(m2) + be*m3
(4) m3 = al*transpose(m1)*transpose(m2) + be*m3
```

(1) will be performed, if both "trans_m1" and "trans_m2" are 'N'.

(2) will be performed, if trans_m1='T' and trans_m2 = 'N'.

(3) will be performed, if trans_m1='N' and trans_m2 = 'T'.

(4) will be performed, if both "trans_m1" and "trans_m2" are 'T'.

If we have four variables nrowa, nrowb, ncola, ncolb defined as follows:

```
if(trans_m1 == 'N') {
  nrowa = number of rows in m1
  ncola = number of columns in m1
}
else if(trans_m1 == 'T') {
  nrowa = number of columns in m1
  ncola = number of rows in m1
}
```

```
if(trans_m2 == 'N') {
  nrowb = number of rows in m2
  ncolb = number of columns in m2
}
else if(trans_m2 == 'T') {
  nrowb = number of columns in m2
  ncolb = number of rows in m2
}
```

Then this function can be executed successfully, if the below conditions are all true:

```
(a) "ncola" is equal to "nrowb"
(b) number of rows in "m3" is equal to or greater than "nrowa"
(b) number of columns in "m3" is equal to or greater than "ncolb"
```

Since "m3" is used as input-output both, memory must be allocated for this matrix before calling this routine, even if simple matrix-matrix multiplication is required. Otherwise it will throw an exception.

For simple matrix-matrix multiplication, no need to specify the value for the input parameters "trans_m1", "trans_m2", "al", "be" (leave them at their default values).

On success, "m3" will be overwritten with the desired output. But "m1" and "m2" will remain unchanged.

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

geadd (m1, m2, trans='N', al=1.0, be=1.0)

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 value can be either 'N' or 'T' [Default: 'N'] (input/optional)
al: A "T" typed (float or double) scalar value [Default: 1.0] (input/optional)
be: A "T" typed (float or double) scalar value [Default: 1.0] (input/optional)

Purpose
The primary aim of this routine is to perform simple matrix-matrix addition. But it can also be used to perform any of the below operations:

```
(1) m2 = al*m1 + be*m2
(2) m2 = al*transpose(m1) + be*m2
```

If trans='N', then expression (1) is solved. In that case, the number of rows and the number of columns in "m1" should be equal to the number of rows and the number of columns in "m2" respectively.
If trans='T', then expression (2) is solved. In that case, the number of columns and the number of rows in "m1" should be equal to the number of rows and the number of columns in "m2" respectively.

If it is needed to scale the input matrices before the addition, corresponding "al" and "be" values can be provided. But for simple matrix-matrix addition, no need to specify values for the input parameters "trans", "al" and "be" (leave them at their default values).

On success, "m2" will be overwritten with the desired output. But "m1" would remain unchanged.

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

operator* (m1, m2)

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

Purpose
This operator operates on two input matrices and returns the resultant matrix after successful multiplication. Both the input matrices remain unchanged.

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

operator+ (m1, m2)

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

Purpose
This operator operates on two input matrices and returns the resultant matrix after successful addition. Both the input matrices remain unchanged.

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

operator~ (m1)

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

Purpose
This operator operates on single input matrix and returns its transposed matrix. E.g., if "m" is a matrix of type `blockcyclic_matrix<T>`, then "~m" will return transposed of matrix "m" of the type `blockcyclic_matrix<T>`.

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

SEE ALSO

sliced_blockcyclic_matrix_local, sliced_blockcyclic_vector_local, blas_wrapper