pMatlab v0.7 Function Reference

Hahn Kim, Nadya Travinin
{hgk, nt}@ll.mit.edu

Table of Contents

This work is sponsored by the Department of the Air Force under Air Force contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government

Introduction

This document is meant to be a reference for functions that are of use to pMatlab *users*, i.e. application developers There are a number of additional functions included in pMatlab, but those functions are used internally by pMatlab and should **not** be called by pMatlab applications.

The functions described in this reference are divided into sections that approximately match Mathwork's own categorization of MATLAB© functions. Most sections describe overloaded MATLAB functions; some sections contain additional functions that are unique to pMatlab, but are related to the overloaded MATLAB functions in that section.

- **pMatlab** describes general pMatlab functions required by all pMatlab applications.
- **Distributed matrices and matrix manipulation** describes functions related to creating and obtaining information about distributed matrices. Because this section contains a large number of overloaded MATLAB functions and a number of new pMatlab functions, it is further divided into subsections.
    - **Elementary distributed matrices** describes functions related to the creation of distributed matrices.
    - **Basic array information** and **Distributed array information** describe functions that obtain information about a distributed matrix.
    - **Matrix manipulation** and **Distributed matrix manipulation** describe functions used to manipulate distributed matrices.
- **Elementary math functions**, **Operators and special characters**, **Sparse matrices**, and **Data analysis and Fourier transforms** describe functions and operators that have been overloaded in pMatlab.

Most functions are *class* functions. Consequently, the names of these functions have been prefaced with the name of class they are a part of. For example, the `fft` function for the `dmat` class is listed as `dmat/fft`. Help for every function can be obtained from the MATLAB command prompt by running `help class/function`. For example, to get the help documentation for the `fft` function overloaded for `dmat`, run:

```
help dmat/fft.
```

For some overloaded functions, it may be useful to refer to the help documentation for the original MATLAB function by running `help function` at the MATLAB command prompt. To get the help documentation for the original MATLAB `fft` function, run:

```
help fft.
```

pMatlab

---

pMATLAB

Data structure created by `pMatlab_Init`. Contains information necessary for communication. See `pMatlab_Init` for more details.

---

pMatlab_Init

Initializes pMatlab environment.

Syntax

```
pMatlab_Init
```

Description

Initializes variables required by the pMatlab library, such as number of processors, current processor's rank and which processor is the leader. All of the variables necessary for communication are stored in the `pMATLAB` structure.

Fields of the `pMATLAB` structure:

- `comm` - contains the MatlabMPI communicator
- `comm_size` - size of communicator, i.e. number of processors
- `my_rank` - rank of the local processor
- `leader` - indicates which rank is the leader, by default set to 0
- `pList` - list of ranks of participating processors
- `tag` - current message tag
- `tag_num` - number of messages sent; synchronized across all processors in `pList`

Fields to be potentially added in the future:

- `num_tasks` - number of tasks (scopes) created from the beginning of the program
- `curr_task` - current task (scope)
- `scopes` - contains a cell array of communication scopes; each entry is a struct with the current fields of the `pMATLAB` structure plus the `task_num` field

pMatlab_Finalize

Terminates pMatlab environment.

Syntax

```
pMatlab_Finalize
```

Description

Terminates pMatlab environment, i.e. exits non-leader MATLAB processes. This ensures that MATLAB processes are not orphaned on remote machines while leaving the leader process running.

---

pMatlab_ver

Display version number for pMatlab

Syntax

```
v = pMatlab_ver
```

Description

`v = pMatlab_ver` returns a string `v` containing the pMatlab version.

MPI_Abort

Aborts any currently running pMatlab or MatlabMPI program and blocks returning until all processes have ended.

Syntax

```
MPI_Abort
```

Description

Will abort any currently running pMatlab/MatlabMPI program by looking for leftover MATLAB processes and killing them. Cannot be used after `MatMPI_Delete_all`. Must be run in the directory from which the pMatlab/MatlabMPI programs was launched.

MatMPI_Delete_all

Deletes the MatMPI directory and its contents.

Syntax

```
MatMPI_Delete_all
```

MPI_Run

Launches a pMatlab or MatlabMPI program.

Syntax

```
eval(MPI_Run(mfile, Ncpus, cpus))
```

Description

`mfile` is a string that contains the name of the pMatlab/MatlabMPI program to be launched, without the `.m` suffix.

`Ncpus` is an integer that specifies the number of processors to launch `mfile` onto.

`cpus` specifies what machines to launch `mfile` onto:

- `cpus = {};` Run all MATLAB processes on the local machine.
- `cpus = {'machine1' 'machine2' ...};` Specify names of machines on which to run. To run interactively, `machine1` must be the name of local machine.
- `cpus = {'machine1:dir1' 'machine2:dir2' ...};` Specify machines names and which directory to use for communication on each machine. Directories must be visible to both machines, i.e. crossmounted. Directories should be located on the local disk of their respective machines.
- `cpus = {'machine1:type' 'machine2:type'};` Specify machine names and the type of each machine. `type` can be either `'unix'` or `'pc'`. Default is `'unix'` (can be changed in `MatMPI_Comm_settings.m`)
- `cpus = {'machine1:type:dir1' 'machine2:type:dir2'};` Specify machine names, communication directories, and the type of each machine.

Distributed matrices and matrix manipulation

Elementary distributed matrices

---

map/map

Map class constructor.

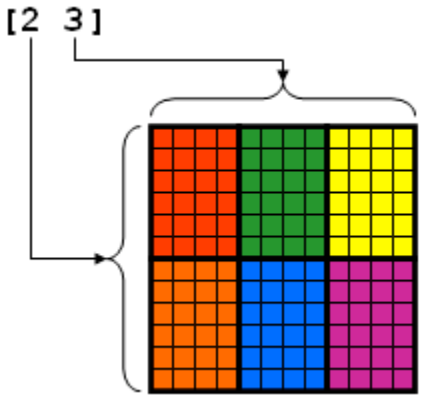
Syntax

```
p = map(GRID_SPEC, DIST_SPEC, PROC_LIST)
p = map(GRID_SPEC, DIST_SPEC, PROC_LIST, OVERLAP_SPEC)
```

Description

`map(GRID_SPEC, DIST_SPEC, PROC_LIST, OVERLAP_SPEC)` constructs a map object to be used as an input to a `dmat` constructor.

- `GRID_SPEC`: Vector of integers specifying how each dimension of a `dmat` is broken up. For example, if `GRID_SPEC = [2 3]`, the first dimension is broken up between 2 processors and the second dimension is broken up between 3 processors. The following figure illustrates how this grid example would break up a `dmat` given 6 processors using a block distribution.



  The length of `GRID_SPEC` can be 2, 3, or 4 and must match the number of dimensions in the `dmat`.

- `DIST_SPEC`: Array of structures with two possible fields, `dist` and `b_size`, specifying the `dmat` distribution.

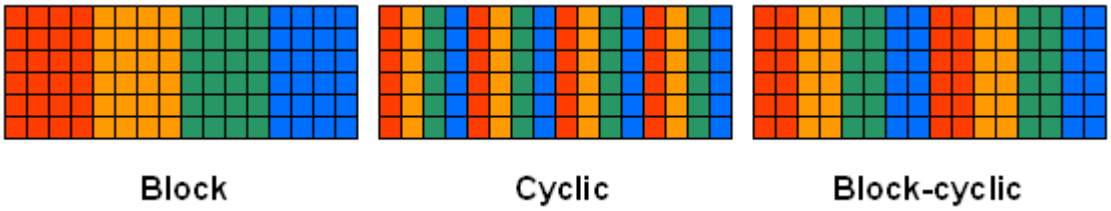
  `DIST_SPEC.dist` is a string specifying the type of data distribution the `dmat` should use. Each entry in the array must have the `dist` field defined. The `dist` field can have three possible values:

- `'b'`: block
- `'c'`: cyclic
- `'bc'`: block-cyclic

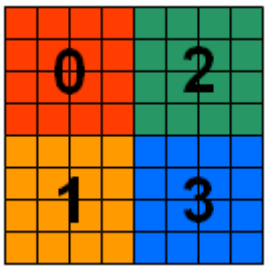
Setting `DIST_SPEC` to `{}` uses block distribution for all dimensions.

`DIST_SPEC.b_size` specifies the block size for block-cyclic distributions. If `DIST_SPEC.dist` is set to `'bc'`, then `DIST_SPEC.b_size` must also be defined. If `DIST_SPEC.dist` is set to `'b'` or `'c'`, then `DIST_SPEC.b_size` does not have to be defined.

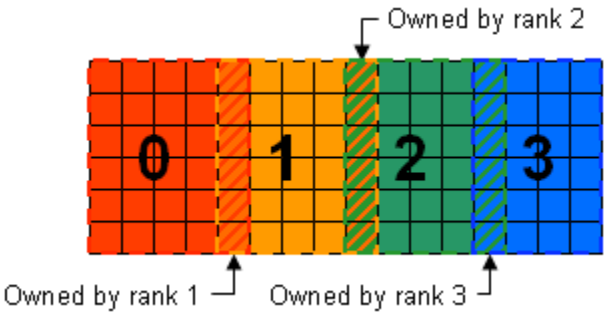
The following figure shows an example of the same `dmat` distributed over 4 processors using each of the three types of data distributions:



- `PROC_LIST`: Array of processor ranks specifying on which ranks the object should be distributed. Ranks are assigned column-wise (top-down, then left-right) to grid locations in sequential order.



- `OVERLAP_SPEC`: Optional. Vector of integers specifying amount of overlap between processors for each dimension. The following figure shows an example of a `dmat` distributed across four processors with 1 column of overlap between adjacent processors.



The length of `OVERLAP_SPEC` can be 2, 3, or 4 and must match the number of dimensions in the `dmat`. Only block distributions can have overlap.

`map` returns a data structure `p` which contains the following fields:

- `DIM`: the number of dimensions of the map (must equal the dimension of the `dmat`)
- `PROC_LIST`: the list of processor ranks on which the object should be distributed
- `DIST_SPEC`: the distribution specification for each dimension
- `GRID`: array of length `DIM` specifying how the object should be distributed

Examples

2D map, 2x2 grid, block-cyclic along rows and columns, block size 2 along rows, block size 3 along columns:

```
grid1 = [2 2];           % 2x2 grid
dist1(1).dist = 'bc';    % block-cyclic along dim 1 (rows)
dist1(1).b_size = 2;     % block size 2 along dim 1 (rows)
dist1(2).dist = 'bc';    % block-cyclic along dim 2 (columns)
dist1(2).b_size = 3;     % block size 3 along dim 2 (columns)
proc1 = [0:3];           % list of ranks 0 through 3
map1 = map(grid1, dist1, proc1);
```

2D map, 2x3 grid, cyclic along both rows and columns:

```
grid2 = [2 3];           % 2x3 grid
dist2(1).dist = 'c';     % cyclic along dim 1 (rows)
dist2(2).dist = 'c';     % cyclic along dim 2 (columns)
proc2 = [0:5];           % list of ranks 0 through 5
map2 = map(grid2, dist2, proc2);
```

2D map, 1x2 grid, block along rows, cyclic along columns:

```
grid3 = [1 2];           % 1x2 grid
dist3(1).dist = 'b';     % block along dim 1 (rows)
dist3(2).dist = 'c';     % cyclic along dim 2 (columns)
proc3 = [0:1];           % list of ranks 0 and 1
map3 = map(grid3, dist3, proc3);
```

3D map, 2x3x2 grid, block-cyclic along rows and columns with block size 2, cyclic along third dimension:

```
grid4 = [2 3 2];         % 2x3x2 grid
dist4(1).dist = 'bc';    % block-cyclic along dim 1 (rows)
dist4(1).b_size = 2;     % block size 2 along dim 1 (rows)
dist4(2).dist = 'bc';    % block-cyclic along dim 2 (columns)
dist4(2).b_size = 2;     % block size 2 along dim 2 (columns)
dist4(3).dist = 'c';     % cyclic along dim 3
proc4 = [0:11];          % list of ranks 0 through 12
map4 = map(grid4, dist4, proc4);
```

2D map, 1x4 grid, block along rows, cyclic along columns:

```
grid5 = [1 4];           % 1x4 grid
dist5(1).dist = 'b';     % block along dim 1 (rows)
dist5(2).dist = 'c';     % cyclic along dim 2 (columns)
proc5 = [0:3];           % list of ranks 0 through 3
map5 = map(grid5, dist5, proc5);
```

2D map, block along both dimensions, overlap in the column dimension of size 1 (1 column overlap):

```
grid6 = [2 2];           % 2x2 grid
dist6 = {};              % block along all dimensions
proc6 = [0 1];           % list of ranks 0 and 1
overlap6 = [0 1];        % overlap of 0 along dim 1 (rows)
                         % overlap of 1 along dim 2 (columns)
map6 = map(grid6, dist6, proc6, overlap6);
```

These examples show only how to create `map` objects. Refer to `dmat/ones`, `dmat/rand`, and `dmat/zeros` on how to create `dmat` objects using `map` objects.

map/zeros

Create a `dmat` of zeros.

Syntax

```
Y = zeros(N, P)
Y = zeros(M, N, P)
Y = zeros(M, N, Q, P)
Y = zeros(M, N, Q, R, P)
```

Description

`zeros(N, P)` returns an `N`-by-`N` `dmat` of zeros mapped according to the map specified by `P`.

`zeros(M, N, P)` returns an `M`-by-`N` `dmat` of zeros mapped according to the map specified by `P`.

`zeros(M, N, Q, P)` returns an `M`-by-`N`-by-`Q` `dmat` of zeros mapped according to the map specified by `P`.

`zeros(M, N, Q, R, P)` returns an `M`-by-`N`-by-`Q`-by-`R` `dmat` of zeros mapped according to the map specified by `P`.

Remarks

Dimension of the `dmat` must be consistent with the dimension of the map's grid.

map/ones

Create a `dmat` of all ones

Syntax

```
Y = ones(N, P)
Y = ones(M, N, P)
Y = ones(M, N, Q, P)
Y = ones(M, N, Q, R, P)
```

Description

`ones(N, P)` returns an `N`-by-`N` `dmat` of ones mapped according to the map specified by P.

`ones(M, N, P)` returns an `M`-by-`N` `dmat` of ones mapped according to the map specified by `P`.

`ones(M, N, Q, P)` returns an `M`-by-`N`-by-`Q` `dmat` of ones mapped according to the map specified by `P`.

`ones(M, N, Q, R, P)` returns an `M`-by-`N`-by-`Q`-by-`R` `dmat` of ones mapped according to the map specified by `P`.

Remarks

Dimension of the `dmat` must be consistent with the dimension of the map's grid.

map/rand

Create a `dmat` of uniformly distributed random numbers.

Syntax

```
Y = rand(N, P)
Y = rand(M, N, P)
Y = rand(M, N, Q, P)
Y = rand(M, N, Q, R, P)
```

Description

The `rand` function generates `dmat`s of random numbers between 0 and 1 distributed uniformly.

`rand(N, P)` returns `N`-by-`N` `dmat` of random numbers mapped according to the map specified by `P`.

`rand(M, N, P)` returns `M`-by-`N` `dmat` of random numbers mapped according to the map specified by `P`.

`rand(M, N, Q, P)` returns an `M`-by-`N`-by-`Q` `dmat` of random numbers mapped according to the map specified by `P`.

`rand(M, N, Q, R, P)` returns an `M`-by-`N`-by-`Q`-by-`R` `dmat` of random numbers mapped according to the map specified by `P`.

Remarks

Dimension of the `dmat` must be consistent with the dimension of the map's grid.

Calls the MATLAB `rand` function to create each local part of the `dmat`. Thus, the resulting array will **not** be the same as a `double` random array of the same dimensions.

Basic array information

---

dmat/size

Size of the `dmat`.

Syntax

```
d = size(X)
[m, n] = size(X)
[d1, d2, d3, ..., dn] = size(X)
```

Description

`d = size(X)` returns the size of each dimension of `dmat` `X` in vector `d`.

`[m,n] = size(X)` returns the size of `dmat` `X` in separate variables `m` and `n`.

`[d1,d2,d3,...,dn] = size(X)` returns the sizes of each dimension of `X` in separate variables.

Remarks

If `A = zeros(m, n, q, p1)` and `B = zeros(m, n, q, p2)`, where `p1` and `p2` are different maps, `size(A)` and `size(B)` return the same results.

---

dmat/ndims

Number of dimension of the dmat.

Syntax

```
n = ndims(A)
```

Description

`n = ndims(A)` returns the number of dimensions in the dmat `A`. The number of dimensions in a dmat is always greater than or equal to 2.

Remarks

`ndims(A)` is `length(size(A))`.

dmat/display

Display `dmat`.

Syntax

```
display(D)
```

Description

`display(D)` aggregates the `D` onto the leader process and displays the entire contents of `D` on the leader process. On remote processes, `display(D)` displays only the local portion of `D`.

`display(D)` is also called for `D` when a semicolon is not used to terminate a statement.

Remarks

Note that `display` incurs communication overhead to aggregate `D` onto the leader processor.

map/display

Display map object.

Syntax

```
display(M)
```

Description

`display(M)` displays the contents of the map object.

Remarks

`display(M)` is also called for `M` when a semicolon is not used to terminate a statement.

dmat/global_block_range

Returns the ranges of global indices local to the current processor for a given `dmat`.

Syntax

```
I = global_block_range(D, DIM)
[I1, I2, ..., IN] = global_block_range(D)
```

Description

`I = global_block_range(D, DIM)` Returns the global index range of the `dmat D` local to the current processor in the specified dimension, `DIM`.

`[I1, I2, ..., IN] = global_block_range(D)` Returns the global index range of the `dmat D` local to the current processor for all `N` dimensions of `D`.

The global index range for each dimension is returned as a 2-element vector. The first element in the vector represents the starting global index and the second element represents the ending index.

Examples

Let `Ncpus` be 4:

```
P = map([1 Ncpus], {}, 0:Ncpus-1);
D = zeros(100, 100, P);
[I1, I2] = global_block_range(D);
```

For each rank, `I1` contains:

Rank	I1(1)	I1(2)
0	1	50
1	51	100
2	1	50
3	51	100

For each rank, `I2` contains:

Rank	I2(1)	I2(2)
0	1	50
1	1	50
2	51	100
3	51	100

dmat/global_block_ranges

Returns the ranges of global indices for all processors in the map of `dmat D`.

Syntax

```
I = global_block_ranges(D, DIM)
[I1, I2, ..., IN] = global_block_ranges(D)
```

Description

`I = global_block_ranges(D, DIM)` Returns the global index ranges of the `dmat D` for all processors in the specified dimension, `DIM`.

`[I1, I2, ..., IN] = global_block_ranges(D)` Returns the global index range of the `dmat D` for all processors in all dimensions of `D`.

For each dimension, the indices are returned as a matrix `I` of size `NUM_PROCS_IN_GRIDx`3. Each line of the returned matrix, `I(i,:)` contains the following information:

```
[PROCESSOR_RANK   START_INDEX   END_INDEX]
```

Examples

Let `Ncpus` be 4:

```
P = map([1 Ncpus], {}, 0:Ncpus-1);
D = zeros(100, 100, P);
[I1, I2] = global_block_ranges(D);
```

On every rank, `I1` contains:

I1(1)	I1(2)	I1(3)
0	1	50
2	1	50
1	51	100
3	51	100

On every rank, `I2` contains:

I2(1)	I2(2)	I2(3)
0	1	50
2	51	100
1	1	51
3	51	100

Remarks

The difference between `global_block_range` and `global_block_ranges` is subtle, but important. `global_block_range` returns a single vector containing the index range for *only* that particular processor. `global_block_ranges` returns a matrix that contains the index ranges for *every* processor.

dmat/global_ind

Returns the global indices local to the current processor.

Syntax

```
I = global_ind(D, DIM)
[I1, I2, ..., IN] = global_ind(D)
```

Description

`I = global_ind(D, DIM)` Returns the global indices of the `dmat D` local to the current processor in the specified dimension, `DIM`.

`[I1, I2, ..., IN] = global_ind(D)` Returns the global indices of the `dmat D` local to the current processor in all dimensions of `D`.

The global indices for each dimension are returned as a vector.

Examples

Let `Ncpus` be 4:

```
P = map([1 Ncpus], {}, 0:Ncpus-1);
D = zeros(100, 100, P);
[I1, I2] = global_ind(D);
```

For each rank, `I1` contains:

Rank	`I1(:)`
0	1 2 3 … 49 50
1	51 52 53 … 99 100
2	1 2 3 … 49 50
3	51 52 53 … 99 100

For each rank, `I2` contains:

Rank	`I2(:)`
0	1 2 3 … 49 50
1	1 2 3 … 49 50
2	51 52 53 … 99 100
3	51 52 53 … 99 100

dmat/global_inds

Returns the global indices for all processors in the map of `dmat D`.

Syntax

```
I = global_inds(D, DIM)
[I1, I2, ..., IN] = global_inds(D)
```

Description

`global_inds(D, DIM)` Returns global indices of the `dmat D` for all processors in the specified dimension, `DIM`.

`global_inds(D)` Returns global indices of the `dmat D` for all processors in all dimensions of `D`.

For each dimension, the indices are returned as a matrix `I` of size `NUM_PROCS_IN_GRIDXMAX_LOCAL_INDS`. Each line of the returned matrix `I`, `I(i,:)`, contains the following information:

```
[PROCESSOR_RANK   IND1   IND2   ...   INDn]
```

To ensure that all rows in the return index are the same, the indices matrix is appended with extra zeros where there are not enough indices.

Examples

Let `Ncpus` be 4:

```
P = map([1 Ncpus], {}, 0:Ncpus-1);
D = zeros(100, 100, P);
[I1, I2] = global_ind(D);
```

On every rank, `I1` contains:

`I1(1)`	`I1(2:end)`
0	1 2 3 … 49 50
2	1 2 3 … 49 50
1	51 52 53 … 99 100
3	51 52 53 … 99 100

On every rank, `I2` contains:

`I2(1)`	`I2(2:end)`
0	1 2 3 … 49 50
2	51 52 53 … 99 100
1	1 2 3 … 49 50
3	51 52 53 … 99 100

Remarks

The difference between `global_ind` and `global_inds` is subtle, but important. `global_ind` returns a single vector containing the indices for only that particular processor. `global_ind` returns a matrix that contains the indices for every processor.

dmat/global_range

Returns the ranges of global indices `dmat D` of local to the current processor. Returns the same range as `global_block_range` if `D` is block distributed, returns subranges for block-cyclic and cyclic distributions.

Syntax

```
I = global_range(D, DIM)
[I1, I2, ..., IN] = global_range(D)
```

Description

`I = global_range(D, DIM)` Returns the global index range of the `dmat D` local to the current processor in the specified dimension, `DIM`.

`[I1, I2, ..., IN] = global_range(D)` Returns the global index range of the `dmat D` local to the current processor in all dimensions of `D`.

For each dimension, the indices are returned as a matrix `I`. Each line of the returned matrix, `I(i,:)`, contains the following information:

```
[START_INDEX_1 END_INDEX_1 START_INDEX_2 END_INDEX_2 ...]
```

Examples

Let `Ncpus` be 4:

```
dist(1).dist = 'b';
dist(2).dist = 'b';
P = map([Ncpus/2 Ncpus/2], dist, 0:Ncpus-1);
D = zeros(100, 100, P);
[I1, I2] = global_range(D);
```

For each rank, `I1` contains:

Rank	`I1`
0	[1 50]
1	[51 100]
2	[1 50]
3	[51 100]

For each rank, `I2` contains:

Rank	`I2`
0	[1 50]
1	[1 50]
2	[51 100]
3	[51 100]

Let `Ncpus` be 4:

```
dist(1).dist = 'c';
dist(2).dist = 'b';
P = map([Ncpus/2 Ncpus/2], dist, 0:Ncpus-1);
D = zeros(100, 100, P);
[I1, I2] = global_range(D);
```

For each rank, `I1` contains:

Rank	`I1`
0	[1 1 3 3 5 5 … 97 97 99 99]
1	[2 2 4 4 6 6 … 98 98 100 100]
2	[1 1 3 3 5 5 … 97 97 99 99]
3	[2 2 4 4 6 6 … 98 98 100 100]

For each rank, `I2` contains:

Rank	`I2`
0	[1 50]
1	[1 50]
2	[51 100]
3	[51 100]

Let `Ncpus` be 4:

```
dist(1).dist = 'bc';
dist(1).b_size = 4;
dist(2).dist = 'b';
P = map([Ncpus/2 Ncpus/2], dist, 0:Ncpus-1);
D = zeros(100, 100, P);
[I1, I2] = global_range(D);
```

For each rank, `I1` contains:

Rank	`I1`
0	[1 4 9 12 … 89 92 97 100]
1	[5 8 13 16 … 93 96]
2	[1 4 9 12 … 89 92 97 100]
3	[5 8 13 16 … 93 96]

For each rank, `I2` contains:

Rank	`I2`
0	[1 50]
1	[1 50]
2	[51 100]
3	[51 100]

dmat/global_ranges

Returns the ranges of global indices for all processors in the map of `dmat D`. Returns the same range as `global_block_ranges` if `D` is block distributed, returns subranges for block-cyclic and cyclic distributions.

Syntax

```
I = global_ranges(D, DIM)
[I1, I2, ..., IN] = global_ranges(D)
```

Description

`I = global_ranges(D, DIM)` Returns the global index ranges of the `dmat D` for all processors in the specified dimension, `DIM`.

`[I1, I2, ..., IN] = global_ranges(D)` Returns the global index range of the `dmat D` for all processors in all dimensions of `D`.

For each dimension, the indices are returned as a matrix `I` of size

`NUM_PROCS_IN_GRIDxNUM_BLOCK_BOUNDARIES`. Each line of the returned matrix, `I(i,:)`, contains the following information：

`[PROCESSOR_RANK START_INDEX_1 END_INDEX_1 START_INDEX_2 END_INDEX_2 ...]`

Examples

Let `Ncpus` be 4:

```
dist(1).dist = 'b';
dist(2).dist = 'b';
P = map([Ncpus/2 Ncpus/2], dist, 0:Ncpus-1);
D = zeros(100, 100, P);
[I1, I2] = global_ranges(D);
```

On every rank, `I1` contains:

I1(1)	I1(2:end)
0	1 50
2	1 50
1	51 100
3	51 100

On every rank, `I2` contains:

I2(1)	I2(2:end)
0	1 50
2	51 100
1	1 50
3	51 100

Let `Ncpus` be 4:

```
dist(1).dist = 'c';
dist(2).dist = 'b';
P = map([Ncpus/2 Ncpus/2], dist, 0:Ncpus-1);
D = zeros(100, 100, P);
[I1, I2] = global_ranges(D);
```

On every rank, `I1` contains:

`I1(1)`	`I1(2:end)`
0	1 1 3 3 5 5 … 97 97 99 99
2	1 1 3 3 5 5 … 97 97 99 99
1	2 2 4 4 6 6 … 98 98 100 100
3	2 2 4 4 6 6 … 98 98 100 100

On every rank, `I2` contains:

`I2(1)`	`I2(2:end)`
0	1 50
2	51 100
1	1 50
3	51 100

Let `Ncpus` be 4:

```
dist(1).dist = 'bc';
dist(1).b_size = 4;
dist(2).dist = 'b';
P = map([Ncpus/2 Ncpus/2], dist, 0:Ncpus-1);
D = zeros(100, 100, P);
[I1, I2] = global_ranges(D);
```

On every rank, `I1` contains:

`I1(1)`	`I1(2:end)`
0	1 4 9 12 … 89 92 97 100
2	1 4 9 12 … 89 92 97 100
1	5 8 13 16 … 93 96 0 0
3	5 8 13 16 … 93 96 0 0

On every rank, `I2` contains:

`I2(1)`	`I2(2:end)`
0	1 50
2	51 100
1	1 50
3	51 100

Remarks

If processors in the same dimension have different number of blocks, the block boundaries are padded with zeros for the processors that have fewer blocks.

map/inmap

Checks if a processor is in the map.

Syntax

```
b = inmap(m, r)
```

Description

`b = inmap(m, r)` checks if processor rank `r` is in map `m`. Returns TRUE for Boolean if rank `r` is in the map, FALSE otherwise.

dmat/find

Find indices of nonzero elements in a `dmat`.

Syntax

```
[I, J] = find(X)
```

Description

`[I, J] = find(X)` returns the row and column indices of nonzero elements of the `dmat` `X`.

Remarks

Currently supports only `[I, J] = find(X)` calling convention. Only works on 2D `dmats`.

`find` requires every processor to send its results to every other processor, thus can incur a significant amount of communication overhead.

dmat/agg

Aggregates the parts of a `dmat` on the leader processor.

Syntax

```
A = agg(D)
```

Description

`A = agg(D)` aggregates the parts of a `dmat` `D` into a whole and returns it as a regular `double` matrix, `A`. If the current processor is the leader, returns the aggregated matrix. Otherwise, returns the local part of `D`.

Remarks

Currently, it doesn't matter if the leader is in the map – the global matrix is returned on the leader, regardless.

Note that `agg` incurs communication overhead to aggregate `D` onto the leader processor.

Since `A` on the leader processor contains the entire contents of `D` but on all other processors contains only the local portion of `D`, `A` will have different values and sizes on each processor. Thus, `agg` should be used with caution.

dmat/agg_all

Aggregates the parts of a `dmat` onto all processors.

Syntax

```
A = agg_all(D)
```

Description

`A = agg_all(D)` aggregates the parts of a `dmat D` onto all processors in the map of `D` and returns a regular `double` matrix `A`.

Remarks

Unlike `agg`, `agg_all` creates a result that is consistent in size and values across all processors . However, because `agg_all` causes all processors to communicate with all processors, `agg_all` can incur a significant amount of communication and should be used with caution.

---

dmat/local

Returns the local part of the `dmat`.

Syntax

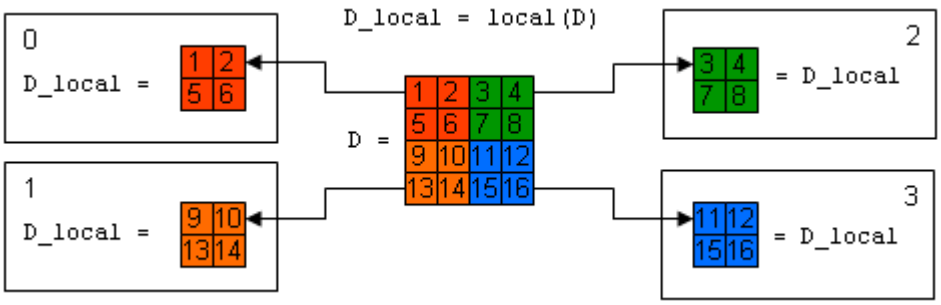
```
D_local = local(D)
```

Description

`D_local = local(D)` Returns the local part of the `dmat D` on the current processor.

Examples

The following diagram shows four processors obtaining their respective local parts of the `dmat`, `D`, and copying the contents into a local variable, `D_local`. Note that `D_local` exists on each processor but contains different data.



---

dmat/put_local

Assigns new data to the local part of the `dmat`.

Syntax

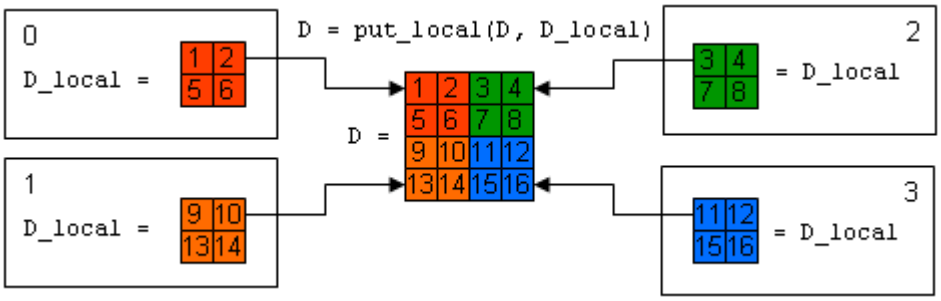
```
D = put_local(D, D_LOCAL)
```

Description

`D = put_local(D, D_LOCAL)` assigns `D_LOCAL` to the local part of the `dmat` `D`.

Examples

The following diagram shows four processors each writing a local matrix, `D_local`, into their respective parts of a `dmat`, `D`. Note that `D_local` on each processor must be the same size as the local portion of their respective parts of `D`.



dmat/synch

Synchronize the overlapped data in a `dmat`.

Syntax

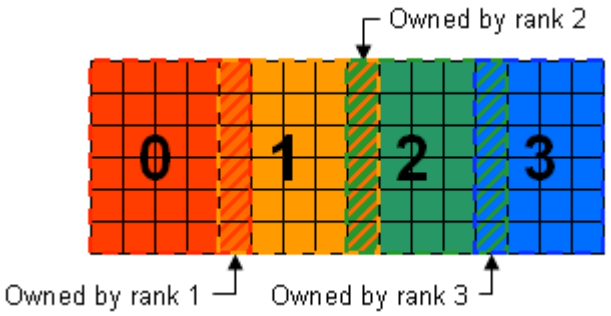
```
D = synch(D)
```

Description

`D = synch(D)` If overlap is present, the owner processor of the overlapping data sends its data to the processor that has a copy of the overlapping data. No-op if there is no overlap.

Remarks

The owner is the processor with the higher index in the grid in the corresponding dimension. For example, if the overlap is in the second dimension the owner is the processor in the column of the grid with the higher index.



---

remap

Remaps a `dmat` with a new map.

Syntax

```
remap(X, NEW_MAP)
```

Description

`remap(X, NEW_MAP)` takes a `dmat` `X` and redistributes it according to the specified `map NEW_MAP`.

dmat/subsasgn

Subscripted assignment to a distributed object. Overloaded method for `A(I)=B`. Should not be called directly.

Syntax

```
A = subsasgn(A, S, B)
```

Description

`A = subsasgn(A, S, B)` Subscripted assignment of `B` (right hand side) to `A` (left hand side). `A` is of type `dmat`. `B` can be of type `dmat` or `double`. `S` is a structure array with the fields:

- `type`: String containing `'()'`, `'{}'`, or `'.'` specifying the subscript type. Currently only supports `'()'` .
- `subs`: Cell array or string containing the actual subscripts.

Remarks

In cases where `A(I)` and/or `B` are distributed across multiple processors, `subsasgn` will automatically transfer the appropriate data between processors.

dmat/subsref

Subscripted reference. Overloaded method for `A(I)`. Should not be called directly.

Syntax

```
B = subsref(A, S)
```

Description

`B = subsref(A, S)` Subscripted reference on a `dmat` A. S is a structure array with the fields:

- `type`: String containing `'()'`, `'{}'`, or `'.'` specifying the subscript type.
- `subs`: Cell array or string containing the actual subscripts.

Remarks:

Currently, `subsref` will only return a standalone `dmat` in the following cases:

- `A(:,:)` – Refers to the entire contents of the `dmat A`
- `A(i,j)` – Refers to a single element in the `dmat A`. Returns a new dmat containing the value at (i,j) stored on the processor which contained that element in `A`.

In all other cases, `subsref` will not produce a "standalone" `dmat`, i.e. the resulting `dmat` can not be directly used as an input to any pMatlab function, with the exception of `local`.

map/subsasgn

Subscripted assignment. Should not be called directly.

Syntax

```
A = subsasgn(A, S, B)
```

Description

`A = subsasgn(A, S, B)` Subscripted assignment of `B` (right hand side) to `A` (left hand side).

`A.FIELD = B` allows the fields of a `map` object `A` to be assigned using the `'.'` notation (complies with structure behavior).

Remarks

This functionality might be deprecated from the final API, to limit control the user has of private members of the MAP object.

---

map/subsref

Subscripted reference. Should not be called directly.

Syntax

```
B = subsref(A, S)
```

Description

`A = subsref (A, S, B)` Subscripted assignment of `A` (right hand side) to `B` (left hand side).

`B = A.FIELD` allows the fields of a `map` object `A` to be referenced using the `'.'` notation (complies with structure behavior).

Remarks

This functionality might be deprecated from the final API, to limit control the user has of private members of the MAP object. `subsref` might be replaced by `get` functions.

Elementary math functions

---

dmat/abs

Absolute value.

Syntax

```
Y = abs(X)
```

Description

`abs(X)` returns a `dmat` `Y` that contains the absolute values of `X`. `Y` has the same mapping as `X`.

---

dmat/complex

Construct complex `dmat` from real `dmat`.

Syntax

```
C = complex(A)
C = complex(A, B)
```

Description

`C = complex(A)` for real `A` returns the complex `dmat` `C` with real part `A` and all zero imaginary part.

`C = complex(A, B)` returns the complex `dmat` `A + Bi`. `A` and `B` must have the same mapping.

Operators and special characters

---

dmat/plus

`+` Plus.

Syntax

```
R = P + Q
```

Description

`R = P + Q` adds two matrices together. `P` and/or `Q` can be a distributed matrix with any type of distribution. If both `P` and `Q` are of type `dmat`, then `R` will have `P`'s map. Otherwise, `R` will have the map of the `dmat` input.

---

dmat/mtimes

`*` Matrix multiply.

Syntax

```
C = A * B
```

Description

`C = A * B` multiples two matrices together. `A` and/or `B` may be a dmat with any type of distribution. If both `A` and `B` are of type `dmat`, then `C` will have `A`'s map. Otherwise `C` will have the map of the `dmat` input.

Remarks

Overlaps have not been tested.

dmat/times

`.*` Element times

Syntax

```
A .* B
```

Description

`A .* B` element-wise multiplies the local part of the `dmat A` by a non-`dmat B`. ALWAYS returns a `dmat`. `A` and `B` must have the same dimensions unless one is a scalar. The scalar may be either distributed or non-distributed.

Remarks

`times` currently only supports the following (reverse orders of operands are also supported):

- scalar `double .*` scalar `dmat`
- scalar `double .*` non-scalar `dmat`
- scalar `dmat .*` scalar `dmat`
- scalar `dmat .*` non-scalar `dmat`
- scalar `dmat .*` non-scalar `double`
- non-scalar `dmat .*` non-scalar `dmat` (if maps are equal)

If multiplying a non-scalar `dmat` by a scalar (`double` or `dmat`), the product's map is equal to the non-scalar `dmat`'s.

If multiplying a non-scalar `double` by a scalar `dmat`, the product uses a single-processor map with block distribution on the same processor as the scalar `dmat`.

dmat/eq

== Equal.

Syntax

```
A == B
```

Description

`A == B` compares the dimensions, maps, sizes and data of `A` and `B`.

If `A` and `B`'s maps, dimensions, and sizes agree then the output is a `dmat` with 0 where elements are not equal and 1 where elements are equal (similar to serial MATLAB).

If the maps are not equal, then a 0 is returned regardless of any other properties.

If the maps agree but the dimensions and/or sizes are not equal, then an error is thrown (analogous to serial MATLAB behavior).

---

map/eq

== Equal.

Syntax

```
A == B
```

Description

`A == B` compares member variables of maps `A` and `B`. If all are the same then TRUE is returned, otherwise FALSE is returned.

dmat/gt

> Greater than.

Syntax

```
A > B
```

Description

`A > B` compares each element of `dmat A` to scalar `B`. Returns a `dmat` with each entry equal to 0 if original entry was < `B`, and 1 otherwise. Calls the MATLAB `gt` on the local part of A.

---

map/ne

~= Not equal.

Syntax

```
A ~= B
```

Description

`A ~= B` Returns FALSE if two maps are equal, TRUE otherwise. Two maps are equal if their grids are equal.

Sparse matrices

map/sparse

Create a sparse `dmat`.

Syntax

```
S = sparse([],[],[],M,N,NZMAX,P)
S = sparse([],[],[],M,N,P)
S = sparse([],[],[],P)
S = sparse(M,N,P)
```

Description

`S = sparse(I,J,S,M,N,NZMAX,P)` generates an `M`-by-`N` sparse `dmat` distributed according to map `P` with space allocated for `NZMAX` nonzeros (note that `NZMAX` applies to the overall `dmat`, not to individual processors. `NZMAX` will be distributed as evenly as possible over all processors).

The rows of `[I,J,S]` are intended to be used to initialize the non-zero values of the matrix. However, `sparse` currently does not support the use of `I,J,` and `S` in pMatlab; they are kept to remain consistent with the `sparse` function built into MATLAB.

There are four ways that `sparse` can be called:

- `S = sparse([],[],[],M,N,NZMAX,P)`
- `S = sparse([],[],[],M,N,P)` uses `NZMAX = 0`.
- `S = sparse([],[],[],P)` uses `M = 0` and `N = 0`. This generates the ultimate sparse matrix, an `M`-by-`N` all zero matrix.
- `S = sparse(M,N,P)` abbreviates `sparse([],[],[],M,N,0,P)`. This also generates an `M`-by-`N` all zero matrix.

Remarks

The recommended method of creating a sparse `dmat` is with `spalloc`.

map/spalloc

Allocate space for a sparse `dmat`.

Syntax

```
S = spalloc(M, N, NZMAX, P)
```

Description

`S = spalloc(M, N, NZMAX, P)` creates an `M`-by-`N` all zero sparse `dmat` with room to eventually hold `NZMAX` nonzeros.

Remarks

`NZMAX` applies to the overall `dmat`, not individual processors. `NZMAX` is evenly distributed across processors

---

dmat/sparse

Converts a `dmat` to a sparse `dmat`

Syntax

```
S = sparse(X)
```

Description

`S = sparse(X)` converts a full `dmat` to sparse form by squeezing out any zero elements. If `X` is already a sparse `dmat`, `sparse(X)` returns `X`.

Data analysis and Fourier transforms

---

dmat/conv2

Two dimensional convolution.

Syntax

```
C = conv2(A, B, 'shape')
```

Description

`C = conv2(A, B, 'shape')` performs the 2D convolution of `dmat A` and `double B`. Returns a subsection of the 2D convolution with size specified by `'shape'`.

Remarks

Only `'shape' == 'same'` is supported, which returns the central part of the convolution of the same size as `A`.

---

dmat/fft

Discrete Fourier transform on a `dmat`.

Syntax

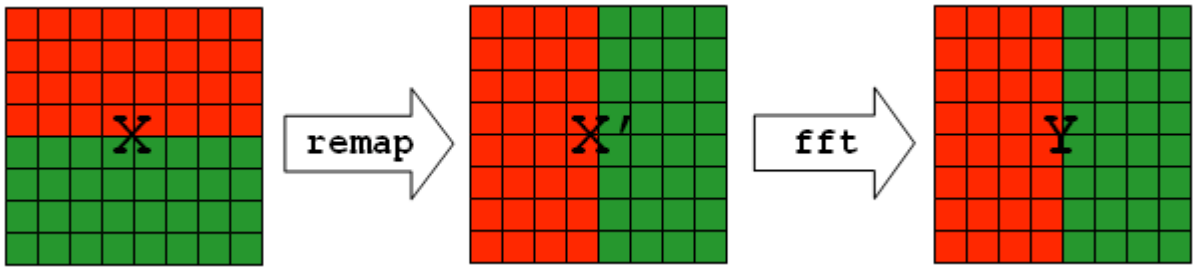
```
Y = fft(X)
Y = fft(X, [], DIM)
Y = fft(X, N, DIM)
```

Description

`fft(X)` is the discrete Fourier transform (DFT) of matrix `X`. The FFT operation is applied to each column. If the matrix `X` is row distributed, `fft` displays a warning and remaps `X`. Calls MATLAB fft on the local part.

`fft(X, [], DIM)` or `fft(X, N, DIM)` applies FFT across dimension `DIM`. `fft(X, N, DIM)` returns the `N` -point DFT. If `X` is distributed along a dimension other than dimension `DIM`, displays a warning and remaps `X` along the dimension `DIM`. Calls the MATLAB `fft` on the local part.

For example, suppose that X is distributed row-wise. Calling `Y = fft(X, [], 1)`, which will perform a FFT on each column, will perform the following remapping:



Remarks

`fft` supports 2D and 3D `dmats`.

If `fft` remaps `x`, `y` has the new map (different from the original map passed in with `x`).

Index