Package 'MMPR'

August 6, 2023

Type Package

Title Multi- and Mixed- Precision Support in R

Version 1.0

Date 2022-12-08

Author David Helmy <david.helmy@brightskiesinc.com>

Maintainer Sameh Abdulah <sameh.abdulah@kaust.edu.sa>

Description The MMPR package provides a new data-structure support for multi- and mixed-precision for R users. The package supports 16-bit, 32-bit, and 64-bit operations with the ability to perform mixed-precision operations through a newly defined tile-based data structure.

License GPL (>= 2) + file LICENSE

Imports methods, Rcpp (>= 1.0.9)

RoxygenNote 7.2.3

R topics documented:

`Arithmetic` *Binary arithmetic numeric/MMPR objects.*

Description

Binary arithmetic for numeric/MMPR objects.

Usage

```
## S4 method for signature 'Rcpp_MMPR,Rcpp_MMPR'
e1 + e2

## S4 method for signature 'Rcpp_MMPR,Rcpp_MMPR'
e1 - e2

## S4 method for signature 'Rcpp_MMPR,Rcpp_MMPR'
e1 * e2

## S4 method for signature 'Rcpp_MMPR,Rcpp_MMPR'
e1 / e2
```

```
## S4 method for signature 'Rcpp_MMPR,Rcpp_MMPR'
e1 ^ e2

## S4 method for signature 'Rcpp_MMPR,BaseLinAlg'
e1 + e2

## S4 method for signature 'Rcpp_MMPR,BaseLinAlg'
e1 * e2

## S4 method for signature 'Rcpp_MMPR,BaseLinAlg'
e1 - e2

## S4 method for signature 'Rcpp_MMPR,BaseLinAlg'
e1 / e2

## S4 method for signature 'Rcpp_MMPR,BaseLinAlg'
e1 ^ e2
```

Arguments

`e1,e2`	Numeric/MMPR objects.

Value

An MMPR object, matching the data type of the highest precision input.

Examples

```
## Not run:
library(MMPR)
s1 <- as.MMPR(1:20,nrow=2,ncol=10,"single")
s2 <- as.MMPR(21:40,nrow=2,ncol=10,"double")

x <- s1 + s2
typeof(x) # A 64-bit precision (double) MMPR matrix.

s3 <- as.MMPR(1:20,nrow=2,ncol=10,"single")
x <- s1 + s3
typeof(x) # A 32-bit precision (single) MMPR matrix.

## End(Not run)
```

`Back/Forward solve` *Back/Forward solve*

Description

Solves a system of linear equations where the coefficient matrix is upper or lower triangular. The function solves the equation A X = B, where A is the coefficient matrix, X is the solution vector, and B is the right-hand side vector.

Usage

```
 ## S4 method for signature 'Rcpp_MMPR,Rcpp_MMPR'
backsolve(r, x, k = ncol(r), upper.tri = TRUE, transpose = FALSE)

 ## S4 method for signature 'Rcpp_MMPR,Rcpp_MMPR'
forwardsolve(l, x, k = ncol(l), upper.tri = FALSE, transpose = FALSE)
```

Arguments

l	An MMPR object.
r	An MMPR object.
x	An MMPR object whose columns give the right-hand sides for the equations.
k	The number of columns of r and rows of x to use.
upper.tri	logical; if TRUE, the upper triangular part of r is used. Otherwise, the lower one.
transpose	logical; if TRUE, solve for t(l , r) %*% output == x.

Value

An MMPR object represents the solution to the system of linear equations.

Examples

```
## Not run:
  library(MMPR)
  a <- matrix(c(2, 0, 0, 3), nrow = 2)
  b <- matrix(c(1, 2), nrow = 2)
  a_mmpr <- as.MMPR(a,2,2,"single")
  b_mmpr <- as.MMPR(b,2,1,"double")
  x <- forwardsolve(a_mmpr, b_mmpr)
  x

## End(Not run)
```

Bind *rbind*

Description

rbind() and cbind() for MMPR objects.

Usage

```
## S3 method for class 'Rcpp_MMPR'
MMPR.rbind(x,y)

## S3 method for class 'Rcpp_MMPR'
MMPR.cbind(x,y)
```

Arguments

x	An MMPR object.
y	An MMPR object.

Value

An MMPR object, matching the data type of the highest precision input.

Examples

```
  ## Not run:
library(MMPR)
# create 2 MMPR matrix a,b
x <- rbind(a,b)
y <- cbind(a,b)

## End(Not run)
```

Check precision *Metadata functions*

Description

Checks the precision of a given MMPR object.

Usage

```
## S4 method for signature 'Rcpp_MMPR'
MMPR.is.single(x)
## S4 method for signature 'Rcpp_MMPR'
MMPR.is.half(x)
## S4 method for signature 'Rcpp_MMPR'
MMPR.is.double(x)
## S4 method for signature 'Rcpp_MMPR'
MMPR.is.float(x)
```

Arguments

x	An MMPR object.

Value

Boolean indicates the precision of the object according to the used function.

Examples

```
  ## Not run:
  library(MMPR)

  x <- as.MMPR(1:20,precision="double")
  MMPR.is.double(x) #TRUE
  MMPR.is.single(x) #FALSE

## End(Not run)
```

Cholesky decomposition

cholesky decomposition

Description

Performs the Cholesky factorization of a positive definite MMPR matrix x.

Usage

```
## S4 method for signature 'Rcpp_MMPR'
chol(x,upper_triangle=TRUE)
```

Arguments

x	An MMPR matrix.
upper_triangle	Boolean to check on which triangle the cholesky decomposition should be applied.

Value

An MMPR matrix.

Examples

```
  ## Not run:
  library(MMPR)
  # x <- as.MMPR(vals,nrow,ncol,precision)
  chol_out <- chol(x)

## End(Not run)
```

`Cholesky inverse` *cholesky inverse*

Description

Performs the inverse of the original matrix using the Cholesky factorization of an MMPR matrix x.

Usage

```
## S4 method for signature 'Rcpp_MMPR'
chol2inv(x, size = NCOL(x))
```

Arguments

`x`	An MMPR object.
`size`	The number of columns to use.

Value

An MMPR object.

Examples

```
  ## Not run:
  library(MMPR)
  # x <- as.MMPR(vals,nrow,ncol,precision)
  chol_out <- chol(x)
  chol <- chol2inv(chol_out)

## End(Not run)
```

`Comparison` *Binary comparison operators for numeric/MMPR objects.*

Description

Binary comparison operators for numeric/MMPR objects.

Usage

```
## S4 method for signature 'Rcpp_MMPR,Rcpp_MMPR'
e1 < e2

## S4 method for signature 'Rcpp_MMPR,Rcpp_MMPR'
e1 <= e2

## S4 method for signature 'Rcpp_MMPR,Rcpp_MMPR'
e1 == e2

## S4 method for signature 'Rcpp_MMPR,Rcpp_MMPR'
e1 != e2

## S4 method for signature 'Rcpp_MMPR,Rcpp_MMPR'
e1 > e2

## S4 method for signature 'Rcpp_MMPR,Rcpp_MMPR'
e1 >= e2

## S4 method for signature 'Rcpp_MMPR,BaseLinAlg'
e1 < e2

## S4 method for signature 'Rcpp_MMPR,BaseLinAlg'
e1 <= e2

## S4 method for signature 'Rcpp_MMPR,BaseLinAlg'
e1 == e2

## S4 method for signature 'Rcpp_MMPR,BaseLinAlg'
e1 != e2

## S4 method for signature 'Rcpp_MMPR,BaseLinAlg'
e1 > e2

## S4 method for signature 'Rcpp_MMPR,BaseLinAlg'
e1 >= e2
```

Arguments

e1,e2	Numeric/MMPR objects.

Value

A vector/matrix of logicals.

Examples

```
  ## Not run:
  library(MMPR)
  s1 <- as.MMPR(1:20,nrow=2,ncol=10,"single")
  s2 <- as.MMPR(21:40,nrow=2,ncol=10,"double")

  x <- s1 > s2

## End(Not run)
```

Concatenate *concatenate*

Description

`c()` function for MMPR objects.

Usage

```
## S3 method for class 'List of MMPR objects'
MMPR.Concatenate(x)
```

Arguments

x	List of MMPR objects.

Value

MMPR object containing values from all objects in the list.

Examples

```
  ## Not run:
library(MMPR)
x <- as.MMPR(1:20,precision"single")
y <- as.MMPR(1:20,precision"single")
list <- c(x,y)
new_obj <- MMPR.Concatenate(list)

## End(Not run)
```

Converters *Converters*

MMPR Converter

Convert R object to MMPR object.

MMPR converters:
`as.MMPR(data,nrow = 0,ncol = 0,precision)`: Converts R object to MMPR object.

`data` R matrix/vector.

`nrow` Number of rows of the new MMPR matrix, **default = zero** which means a vector will be created.

`ncol` Number of cols of the new MMPR matrix, **default = zero** which means a vector will be created.

`precision` String indicates the precision of the new MMPR object (half, single, or double).

R Converter

Convert an MMPR object to R object.

R vector converter:
`MMPR.ToNumericVector(x)`: Converts an MMPR object to a numeric R vector.

`x` MMPR object.

R matrix converter:
`MMPR.ToNumericMatrix(x)`: Converts an MMPR object to a numeric R matrix.

`x` MMPR object.

Examples

```
 ## Not run:

  # Example usage of the class and its methods
  a <- matrix(1:36, 6, 6)
  mmpr_matrix <- as.MMPR(a,nrow=6,ncol=6,precision="single")
  r_vector <- MMPR.ToNumericVector(mmpr_matrix)
  r_vector
  r_matrix <- MMPR.ToNumericMatrix(mmpr_matrix)
  r_matrix

## End(Not run)
```

Copy *copy*

MMPR deep copy

Create a copy of an MMPR object. Typically, using 'equal' creates a new pointer for the object, resulting in any modifications made to object one affecting object two as well.

copy:
`MMPR.copy(x)`: Create a new copy of an MMPR object.

`x` MMPR object.

MMPRTile deep copy

Create a duplicate of an MMPRTile object. Usually, using 'equal' creates a new pointer for the object, causing any modifications made to object one to affect object two as well.

copy:
`MMPRTile.copy(x)`: Create a new copy of an MMPRTile matrix.

`x` MMPRTile matrix.

Examples

```
## Not run:
 # Example usage of the class and its methods
 a <- matrix(1:36, 6, 6)
 mmpr_matrix <- as.MMPR(a,nrow=6,ncol=6,precision="single")

 # Normal equal '=' will create a new pointer of the object, so any change in object A
 # will affect object B
 temp_mmpr_matrix = mmpr_matrix
 temp_mmpr_matrix[2,2] <- 500
 mmpr_matrix[2,2]           #500


 mmpr_matrix_copy <- MMPR.copy(mmpr_matrix)
 mmpr_matrix[2,2] <-100
 mmpr_matrix_copy[2,2] <- 200

 mmpr_matrix[2,2]           #100
 mmpr_matrix_copy[2,2]      #200

## End(Not run)
```

Crossprod *crossprod*

Description

Calculates the cross product of two MMPR matrices. It uses BLAS routine `gemm()` for **A** X **B** operations and `syrk()` for **A** X **A^T** operations.

Usage

```
## S4 method for signature 'Rcpp_MMPR'
crossprod(x, y = NULL)

## S4 method for signature 'Rcpp_MMPR'
tcrossprod(x, y = NULL)
```

Arguments

`x`	An MMPR object.
`y`	Either `NULL`, or an MMPR matrix.

Details

Calculates cross product of two MMPR matrices performs:
x %*% y , t(x) %*% x
This function uses blas routine `gemm()` for **A** X **B** operations & `syrk()` for **A** X **A^T** operations.

Value

An MMPR matrix.

Examples

```
    ## Not run:
    library(MMPR)
    x <- as.MMPR(1:16,4,4,"single")
    y <- as.MMPR(1:20,4,5,"double")

    z <- crossprod(x)     # t(x) x
    z <- tcrossprod(x)    # x t(x)
    z <- crossprod(x,y)   # x y
    z <- x %*% y          # x y

## End(Not run)
```

Diagonal *diag*

Description

Returns the diagonal of an MMPR matrix.

Usage

```
## S4 method for signature 'Rcpp_MMPR'
diag(x)
```

Arguments

x	An MMPR matrix.

Value

An MMPR vector contains the main diagonal of the matrix.

Examples

```
  ## Not run:
    library(MMPR)
    x <- as.MMPR(1:16,4,4,"single")
    diag_vals <- diag(x)

## End(Not run)
```

Dimensions *dimensions*

Description

Returns the number of rows or cols in an MMPR object.

Usage

```
## S4 method for signature 'Rcpp_MMPR'
nrow(x)

## S4 method for signature 'Rcpp_MMPR'
ncol(x)
```

Arguments

x	An MMPR object.

Value

The number of rows/cols in an MMPR object.

Examples

```
  ## Not run:
    library(MMPR)
    x <- as.MMPR(1:16,4,4,"single")
    y <- as.MMPR(1:20,4,5,"double")
    rows_x <- nrow(x)
    cols_y <- ncol(y)

## End(Not run)
```

Eigen decomposition

eigen decomposition

Description

Solves a system of equations or invert an MMPR matrix, using lapack routine `syevr()`

Usage

```
## S4 method for signature 'Rcpp_MMPR'
eigen(x, only.values = FALSE)
```

Arguments

x	An MMPR object.
only.values	(TRUE/FALSE)?

Value

A list contains MMPR objects describing the values and optionally vectors.

Examples

```
## Not run:
library(MMPR)
x <- as.MMPR("your_data",nrow,ncol,precision)
y <- eigen(x)

## End(Not run)
```

`Extract-Replace` *Extract or replace elements from an MMPR object.*

Description

Extract or replace elements from an MMPR object using the '[', '[[', '[<-', and '[[<-' operators. When extracting values, they will be converted to double precision. However, if you update a single object, the double value will be cast down to match the precision. If the MMPR object is a matrix and you access it using the 'i' index, the operation is assumed to be performed in column-major order, or using 'i' and 'j' index.

Usage

```
  ## S3 method for class 'Rcpp_MMPR'
x[i, j, drop = TRUE]
  ## S3 method for class 'Rcpp_MMPR'
x [<- i, j, ..., value
  ## S3 method for class 'Rcpp_MMPR'
x[[i, drop = TRUE]]
  ## S3 method for class 'Rcpp_MMPR'
x [[<- i, ..., value
```

Arguments

`x`	An MMPR object.
`i`	Row index or indices.
`j`	Column index or indices.
`drop`	ignored.
`value`	A value to replace the selected elements with.

Examples

```
  ## Not run:
  library(MMPR)
    x <-as.MMPR(1:50,"single")
    ext <- x[5]
    x[5] <- 0
    x$ToMatrix(5,10)
    x[2,5]
    x[3,5] <- 100

## End(Not run)
```

Extremes *Min-Max Functions*

Description

Min-Max functions for MMPR objects values and indices, all NA values are disregarded.

Usage

```
## S4 method for signature 'Rcpp_MMPR'
min(x)

## S4 method for signature 'Rcpp_MMPR'
max(x)

## S4 method for signature 'Rcpp_MMPR'
which.min(x)

## S4 method for signature 'Rcpp_MMPR'
which.max(x)
```

Arguments

x	An MMPR object.

Value

Min/max value/index.

Examples

```
  ## Not run:
  library(MMPR)

  x <- as.MMPR(1:20,precision="double")
  min <-min(x)
  min_idx <-which.min(x)

## End(Not run)
```

Hyperbolic *Hyperbolic functions*

Description

These functions give the obvious hyperbolic functions. They respectively compute the hyperbolic cosine, sine, tangent, and their inverses, arc-cosine, arc-sine, arc-tangent (or 'area cosine', etc).

Usage

```
## S4 method for signature 'Rcpp_MMPR'
sinh(x)

## S4 method for signature 'Rcpp_MMPR'
cosh(x)

## S4 method for signature 'Rcpp_MMPR'
tanh(x)

## S4 method for signature 'Rcpp_MMPR'
asinh(x)

## S4 method for signature 'Rcpp_MMPR'
acosh(x)

## S4 method for signature 'Rcpp_MMPR'
atanh(x)
```

Arguments

x	An MMPR object.

Value

An MMPR object of the same dimensions as the input.

Examples

```
## Not run:
library(MMPR)

mppr_matrix <- as.MMPR(1:20,nrow=2,ncol=10,precision="single")
x <- sinh(mppr_matrix)

## End(Not run)
```

Log *Logarithms and Exponentials*

Description

exp/log functions.

Usage

```
## S4 method for signature 'Rcpp_MMPR'
exp(x)

## S4 method for signature 'Rcpp_MMPR'
expm1(x)

## S4 method for signature 'Rcpp_MMPR'
log(x, base = 1)

## S4 method for signature 'Rcpp_MMPR'
log10(x)

## S4 method for signature 'Rcpp_MMPR'
log2(x)
```

Arguments

x	An MMPR object.
base	The logarithm base. If base = 1, exp(1) is assumed, only base 1,2, and 10 available.

Value

An MMPR object of the same dimensions as the input.

Examples

```
## Not run:
library(MMPR)

x <- as.MMPR(1:20,precision="double")
log(x)

## End(Not run)
```

Mathis *Finite, infinite, and NaNs*

Description

Finite, infinite, and NaNs.

Usage

```
## S4 method for signature 'Rcpp_MMPR'
is.finite(x)

## S4 method for signature 'Rcpp_MMPR'
is.infinite(x)

## S4 method for signature 'Rcpp_MMPR'
is.nan(x)
```

Arguments

x	An MMPR object.

Value

A bool vector/matrix of the same dimensions as the input.

Examples

```
  ## Not run:
  library(MMPR)

  x <- as.MMPR(1:20,precision="double")
  is.nan(sqrt(x))

## End(Not run)
```

Metadata *Metadata functions*

Description

Metadata functions.

Usage

```
## S4 method for signature 'Rcpp_MMPR'
storage.mode(x)
## S4 method for signature 'Rcpp_MMPR'
typeof(x)
## S4 method for signature 'Rcpp_MMPR'
MMPR.object.size(x)
## S4 method for signature 'Rcpp_MMPR'
MMPR.ChangePrecision(x,precision)
```

Arguments

x	An MMPR object.
precision	String with the required precision.

Value

Prints/change metadata about an MMPR object.

Examples

```
  ## Not run:
  library(MMPR)

  x <- as.MMPR(1:20,precision="double")
  typeof(x)
  MMPR.ChangePrecision(x,"single")

## End(Not run)
```

Miscmath *Miscellaneous mathematical functions*

Description

Miscellaneous mathematical functions.

Usage

```
## S4 method for signature 'Rcpp_MMPR'
abs(x)

## S4 method for signature 'Rcpp_MMPR'
sqrt(x)
```

Arguments

x	An MMPR object.

Value

An MMPR object of the same dimensions as the input.

Examples

```
## Not run:
library(MMPR)

x <- as.MMPR(1:20,precision="double")
sqrt(x)

## End(Not run)
```

MMPR *MMPR S4 Class*

Description

MMPR is a multi-precision vector/matrix, that enables the creation of vector/matrix with three different precisions (16-bit (half), 32-bit(single), and 64-bit(double)).

Constructor

`new` Creates a new instance of zero values of the `MMPR` class. `new(MMPR,size,"precision")`

`size` The total number of values for which memory needs to be allocated.

`precision` String to indicate the precision of MMPR object ("half","single", or "double").

Accessors

The following accessors can be used to get the values of the slots:

`IsMatrix` Boolean to indicate whether the MMPR object is a vector or matrix.

`Size` Total number of elements inside the object, (row*col) in the case of matrix, and number of elements in the case of vector.

`Row` Number of rows.

`Col` Number of cols.

Methods

The following methods are available for objects of class `MMPR`:

PrintValues: `PrintValues()`: Prints all the values stored in the matrix or vector, along with metadata about the object.

ToMatrix: `ToMatrix(row,col)`: Changes the object representation to match the new dimensions, no memory overhead.

ToVector: `ToVector()`: Changes the MMPR matrix to vector, no memory overhead.

Examples

```
 ## Not run:
   # Example usage of the class and its methods
   mmpr_object <- new(MMPR,50,"single")

   mmpr_object$ToMatrix(5,10)
   mmpr_object$Row       #5
   mmpr_object$Col       #10
   mmpr_object$Size      #50
   mmpr_object$IsMatrix  #TRUE

   mmpr_object$PrintValues()
   mmpr_object$ToVector()

   mmpr_object

## End(Not run)
```

`MMPR GEMM` *MMPR GEMM (Matrix-Matrix Multiplication)*

Description

Performs matrix-matrix multiplication of two given MMPR matrices to performs:
C = alpha A * B + beta C
C = alpha A A^T + beta C

Usage

```
## S4 method for signature 'Rcpp_MMPR'
MMPR.gemm(a,b = NULL,c,transpose_a= FALSE,transpose_b=FALSE,alpha=1,beta=0)
```

Arguments

a	An MMPR matrix A.
b	An MMPR matrix B, if NULL, the function will perform syrk operation from blas.
c	Input/Output MMPR matrix C.
transpose_a	A flag to indicate whether transpose matrix A should be used, if B is NULL and transpose_a =TRUE The function will perform the following operation: **C=alphaA^TXA+betaC**.
transpose_b	A flag to indicate whether transpose matrix B should be used.
alpha	Specifies the scalar alpha.
beta	Specifies the scalar beta.

Value

An MMPR matrix.

Examples

```
   ## Not run:
   library(MMPR)
   # create 3 MMPR matrices a,b,c
   print(c)
   MMPR.gemm(a,b,c,transpose_a=false,transpose_b=TRUE,alpha=1,beta=1)
   print(c)

## End(Not run)
```

MMPR TRSM *MMPR TRSM (Triangular Solve)*

Description

Solves a triangular matrix equation.
performs:
op(A)*X=alpha*B
X*op(A)=alpha*B

Usage

```
## S4 method for signature 'Rcpp_MMPR'
MMPR.trsm(a,b,upper_triangle,transpose,side = 'L',alpha =1)
```

Arguments

a	MMPR Matrix A.
b	MMPR Matrix B.
upper_triangle	If the value is TRUE, the referenced part of matrix A corresponds to the upper triangle, with the opposite triangle assumed to contain zeros.
transpose	If TRUE, the transpose of A is used.
side	'R for Right side, 'L' for Left side.
alpha	Factor used for A, If alpha is zero, A is not accessed.

Value

An MMPR Matrix.

Examples

```
## Not run:
library(MMPR)
# create 2 MMPR matrices a,b
c <- MMPR.trsm(a,b,upper_triangle=TRUE,transpose=FALSE,side='L',alpha=1)
print(c)

## End(Not run)
```

MMPRTile *MMPRTile S4 Class*

Description

MMPRTile is a data structure for tile matrices with mixed precision, where each tile possesses a specific precision level.

Constructor

`new` creates a new instance of Tile-Matrix `MMPRTile` class.
`new(MMPRTile,rows,cols,rows_per_tile,cols_per_tile,values,precisions)`

`rows` Number of rows in the matrix.

`cols` Number of cols in the matrix.

`rows_per_tile` Number of rows in each tile.

`cols_per_tile` Number of cols in each tile.

`values` R matrix or vector containing all the values that should be in the matrix.

`precisions` R matrix or vector of strings, containing precision type of each tile.

Accessors

The following accessors can be used to get the values of the slots:

`Size` Total number of elements inside the Matrix.

`Row` Number of rows.

`Col` Number of cols.

`TileRow` Number of rows in each tile.

`TileCol` Number of cols in each tile.

`TileSize` Total number of elements in each tile.

Methods

The following methods are available for objects of class `MMPRTile`:

PrintTile:
`PrintTile(tile_row_idx,tile_col_idx)`: Prints all the values stored inside a specific tile plus meta-data about the tile.

`tile_row_idx` Row index of the tile.

`tile_col_idx` Col index of the tile.

ChangeTilePrecision:
`ChangeTilePrecision(tile_row_idx,tile_col_idx,precision)`: Change the precision of specific tile, this function will need to copy all the values to cast them to the new precision.

`tile_row_idx` Row index of the tile.

`tile_col_idx` Col index of the tile.

`precision` Required new precision as a string.

FillSquareTriangle:
`FillSquareTriangle(value,upper.tri,precision)`: Fills upper or lower triangle with a given value and precision, new tiles will be created, replacing the old tiles. **Note:** The input must be a square matrix

`value` A value used during matrix filling.

`upper.tri` A flag to indicate what triangle to fill. if TRUE, the upper triangle will be filled, otherwise the lower triangle.

`precision` The precision of the tiles created during matrix filling, in case it's not a diagonal tile.

Sum: `Sum()`: Get the sum of all elements in all tiles in MMPRTile Matrix.

Prod: `Prod()`: Get the product of all elements in all tiles in MMPRTile Matrix.

Examples

```
## Not run:
  # Example usage of the class and its methods
 a <- matrix(1:36, 6, 6)
 b <- c("double", "double", "single", "double",
        "half", "double", "half", "double",
        "single")

 tile_mat <- new(MMPRTile, 6, 6, 2, 2, a, b)
 tile_mat
 sum <- tile_mat$Sum()
 prod <- tile_mat$Prod()
 tile_mat$PrintTile(1,1)
 tile_mat$ChangeTilePrecision(1,1,"single")

 n_rows <- tile_mat$Row
 n_cols <- tile_mat$Col
 total_size <- tile_mat$Size
 rows_per_tile <- tile_mat$TileRow
 cols_per_tile <- tile_mat$TileCol

## End(Not run)
```

MMPRTile Cholesky decomposition

MMPRTile Chol (Cholesky decomposition)

Description

Tile-based Cholesky decomposition of a positive definite tile-based symmetric matrix.

Usage

```
## S4 method for signature 'Rcpp_MMPR'
chol(x, overwrite_input = TRUE, num_threads = 1)
```

Arguments

`x`	An MMPR tile matrix.
`overwrite_input`	A flag to determine whether to overwrite the input (TRUE), or return a new MMPR tile matrix.
`num_threads`	An integer to determine number if thread to run using openmp, default = 1 (serial with no parallelization).

Value

An MMPR tile matrix.

Examples

```
  ## Not run:
  x <- chol(y,overwrite_input=FALSE,num_threads=8)
  x <- chol(x)

## End(Not run)
```

MMPRTile GEMM *MMPRTile GEMM (Matrix-Matrix Multiplication)*

Description

Tile-based matrix-matrix multiplication of two given MMPR tiled matrices to **perform:**
C = **alpha*A** X **B** + **beta*C**

Usage

```
## S4 method for signature 'Rcpp_MMPRTile'
MMPRTile.gemm(a,b,c,transpose_a= FALSE,transpose_b=FALSE,alpha=1,beta=0)
```

Arguments

a	An MMPR tile matrix A.
b	An MMPR tile matrix B.
c	Input/Output MMPR tile matrix C.
transpose_a	A flag to indicate whether transpose matrix A should be used.
transpose_b	A flag to indicate whether transpose matrix B should be used.
alpha	Specifies the scalar alpha.
beta	Specifies the scalar beta.
num_threads	An integer to determine number if thread to run using openmp, default = 1 (serial with no parallelization).

Value

An MMPR tile matrix C.

Examples

```
  ## Not run:
  library(MMPR)
  # create 3 MMPR Tile matrices a,b,c
  print(c)
  MMPRTile.gemm(a,b,c,transpose_a=false,transpose_b=TRUE,alpha=1,beta=1,num_threads = 8)
  print(c)

## End(Not run)
```

MMPRTile TRSM *MMPRTile TRSM (Triangular Solve)*

Description

Tile-based algorithm to solve a triangular matrix equation for MMPR tiled matrices.
performs:
op(A)*X=alpha*B
X*op(A)=alpha*B

Usage

```
## S4 method for signature 'Rcpp_MMPRTile'
MMPRTile.trsm(a,b,side,upper_triangle,transpose,alpha)
```

Arguments

a	An MMPR tile matrix A.
b	An MMPR tile matrix B, X after returning.
side	'R' for right side, 'L' for left side.
upper_triangle	What part of the matrix A is referenced (if TRUE upper triangle is referenced), the opposite triangle being assumed to be zero.
transpose	If TRUE, the transpose of A is used.
alpha	Factor used for A, If alpha is zero, A is not accessed.

Value

An MMPR Tile Matrix B ->(X).

Examples

```
## Not run:
library(MMPR)
# create 2 MMPR Tile matrices a,b
print(b)
MMPRTile.trsm(a,b,side='L',upper_triangle=TRUE,transpose=FALSE,alpha=1)
print(b)

## End(Not run)
```

NA's *NA's*

Description

is.na() ,na.omit(), and na.exclude() for MMPR objects.

Usage

```
## S3 method for class 'Rcpp_MMPR'
MMPR.is.na(object,index=-1)
## S3 method for class 'Rcpp_MMPR'
MMPR.na.exclude(object,value)
## S3 method for class 'Rcpp_MMPR'
MMPR.na.omit(object)
```

Arguments

object	MMPR object.
index	If a particular index in the MMPR matrix/vector is specified, it will be checked. If no index is provided, all elements will be checked.
value	Value to replace all NAN with.

Value

MMPR.is.na will return matrix/vector/bool according to input of the function.
MMPR.na.exclude & MMPR.na.omit will not return anything.

Examples

```
  ## Not run:
library(MMPR)
x <- as.MMPR(1:20,precision"single")
x[1] <- NAN
MMPR.is.na(x,index=1) #TRUE
MMPR.na.exclude(x,50)
x[1]  #50

## End(Not run)
```

Norm *norm*

Description

Compute norm.

Usage

```
## S4 method for signature 'Rcpp_MMPR'
norm(x, type = "O")
```

Arguments

x	An MMPR object.
type	"O"-ne, "I"-nfinity, "F"-robenius, "M"-ax modulus, and "1" norms.

Value

An MMPR object.

Examples

```
## Not run:
library(MMPR)

x <- as.MMPR(1:20,precision="double")
norm(x, type="O")

## End(Not run)
```

Print *print*

Description

Prints the precision and type of the object, and print will print the meta data of the object without printing the values. Function x$PrintValues() should be used to print the values."

Usage

```
## S4 method for signature 'Rcpp_MMPR'
print(x)

## S4 method for signature 'Rcpp_MMPR'
show(object)
```

Arguments

`x, object`	An MMPR objects.

Details

Prints metadata about the object and some values.

Examples

```
  ## Not run:
    library(MMPR)
    x <- as.MMPR(1:16,4,4,"single")
    y <- as.MMPR(1:20,4,5,"double")
    x
    print(y)

## End(Not run)
```

`QR decomposition` *QR decomposition*

Description

QR factorization and related functions.

Usage

```
## S4 method for signature 'Rcpp_MMPR'
qr(x, tol = 1e-07)

## S4 method for signature 'ANY'
qr.Q(qr, complete = FALSE, Dvec)

## S4 method for signature 'ANY'
qr.R(qr, complete = FALSE)
```

Arguments

`x`	An MMPR matrix.
`qr`	QR decomposition MMPR object.
`tol`	The tolerance for determining numerical column rank.
`complete`	Should the complete or truncated factor be returned?
`Dvec`	Vector of diagonals to use when re-constructing Q (**default is 1's**).

Details

The factorization is performed by the LAPACK routine `geqp3()`. This should be similar to calling `qr()` on an ordinary R matrix with the argument `LAPACK=TRUE`.

Value

`qr`	Output of `qr()`.

Examples

```
## Not run:

library(MMPR)


qr_input <-as.MMPR( c(1, 2, 3, 2, 4, 6, 3, 3, 3),3,3,"single")
qr_out <- qr(qr_input)
qr_out
qr_out[["qr"]]$PrintValues()
qr_out[["qraux"]]$PrintValues()
qr_out[["pivot"]]$PrintValues()
qr_out[["rank"]]$PrintValues()

qr_q <- qr.Q(qr_out)
qr_q

## End(Not run)
```

Reciprocal condition

reciprocal condition

Description

Compute matrix norm.

Usage

```
## S4 method for signature 'Rcpp_MMPR'
rcond(x, norm = "O", useInv = FALSE)
```

Arguments

`x`	An MMPR object.
`norm`	"O"-ne or "I"-nfinity norm.
`useInv`	TRUE to use the lower triangle only.

Value

An MMPR Object.

Examples

```
## Not run:
library(MMPR)

x <- as.MMPR(1:20,precision="double")
rcond(x)

## End(Not run)
```

`Replicate` *replicate*

Description

Replicates the given input number of times according to count/len , only one should be set at a time, and in case both values are given, only the len value will have effect.

Usage

```
## S4 method for signature 'Rcpp_MMPR'
rep(x,count=0,len=0)
```

Arguments

`x`	An MMPR object.
`count`	Value to determine how many times the input value will be replicated.
`len`	Value to determine the required output size, the input will be replicated until it matches the output len size.

Value

MMPR vector containing the replicated values.

Examples

```
  ## Not run:
    library(MMPR)
    x <- as.MMPR(1:16,4,4,"single")
    rep_vals_1 <- rep(x,count=2)  #output size will be 16*2
    rep_vals_2 <- rep(x,len=2)  #output size will be 2

## End(Not run)
```

Round *Rounding functions*

Description

Rounding functions.

Usage

```
## S4 method for signature 'Rcpp_MMPR'
ceiling(x)

## S4 method for signature 'Rcpp_MMPR'
floor(x)

## S4 method for signature 'Rcpp_MMPR'
trunc(x)

## S4 method for signature 'Rcpp_MMPR'
round(x, digits = 0)
```

Arguments

x	An MMPR object.
digits	The number of digits to use in rounding.

Value

An MMPR object of the same dimensions as the input.

Examples

```
  ## Not run:
  library(MMPR)

  x <- as.MMPR(1:20,precision="double")
  floor(x)

## End(Not run)
```

Scale *scale*

Description

Center or scale an MMPR object.

Usage

```
## S4 method for signature 'Rcpp_MMPR'
scale(x, center, scale)
```

Arguments

x	An MMPR object.
center, scale	Logical or MMPR objects.

Value

An MMPR matrix.

Examples

```
  ## Not run:
  library(MMPR)
    x <-as.MMPR(1:50,"single")
    x$ToMatrix(5, 10)
    temp_center_scale <- new(1:10,precision="double")
    z <- scale(x=temp_scale, center=FALSE, scale=temp_center_scale)

## End(Not run)
```

Singular value decomposition *SVD*

Description

SVD factorization.

Usage

```
## S4 method for signature 'Rcpp_MMPR'
La.svd(x, nu = min(n, p), nv = min(n, p))

## S4 method for signature 'Rcpp_MMPR'
svd(x, nu = min(n, p), nv = min(n, p))
```

Arguments

x	An MMPR matrix.
nu, nv	The number of left/right singular vectors to return.

Details

The factorization is performed by the LAPACK routine gesdd().

Examples

```
## Not run:
library(MMPR)
x <- as.MMPR("your_data",nrow,ncol,precision)
y <- svd(x)

## End(Not run)
```

Solve *solve*

Description

Solve a system of equations or invert an MMPR matrix.

Usage

```
## S4 method for signature 'Rcpp_MMPR'
solve(a, b = NULL, ...)
```

Arguments

a, b	An MMPR objects.
...	Ignored.

Value

Solves the equation AX=B .and if B=NULL t(A) will be used.

Examples

```
## Not run:
library(MMPR)

x <- as.MMPR(1:20,4,5,"double")
y <- crossprod(x)
solve(y)

## End(Not run)
```

`Special Math` *Special mathematical functions.*

Description

Special mathematical functions.

Usage

```
## S4 method for signature 'Rcpp_MMPR'
gamma(x)

## S4 method for signature 'Rcpp_MMPR'
lgamma(x)
```

Arguments

`x`	An MMPR object.

Value

An MMPR object of the same dimensions as the input.

Examples

```
## Not run:
library(MMPR)

x <- as.MMPR(1:20,precision="double")
lgamma(x)

## End(Not run)
```

`Sweep` *sweep*

Description

Sweep an MMPR vector through an MMPR matrix.

Usage

```
## S4 method for signature 'Rcpp_MMPR'
sweep(x,stat,margin,FUN)
```

Arguments

x	An MMPR object.
stat	MMPR vector containing the value(s) that should be used in the operation.
margin	1 means row; otherwise means column.
FUN	Sweeping function; must be one of "+", "−", "*", "/", or "^".

Value

An MMPR matrix of the same type as the highest precision input.

Examples

```
## Not run:
library(MMPR)
x <- as.MMPR(1:20,10,2,"single")
y <- as.MMPR(1:5,precision="double")
sweep_out <- sweep(x, stat=y, margin=1, FUN="+")
MMPR.is.double(sweep_out)  #TRUE

## End(Not run)
```

Symmetric *isSymmetric*

Description

Check if a given MMPR matrix is symmetric.

Usage

```
## S4 method for signature 'Rcpp_MMPR'
isSymmetric(object, ...)
```

Arguments

object	An MMPR matrix.
...	Ignored.

Value

A logical value.

Examples

```
  ## Not run:
library(MMPR)

x <- as.MPPR(1:200,100,2,"Single")
isSymmetric(x)                      #false

crossprod_output<-crossprod(x)
isSymmetric(crossprod_output)       #true

## End(Not run)
```

Transpose *transpose*

Description

Transpose an MMPR object.

Usage

```
## S4 method for signature 'Rcpp_MMPR'
t(x)
```

Arguments

x	An MMPR object.

Value

An MMPR object.

Examples

```
  ## Not run:
  library(MMPR)
  a <- matrix(1:20, nrow = 2)
  a_mmpr <- as.MMPR(a,2,10,"double")
  a_mmpr_transpose <- t(a_mmpr)

## End(Not run)
```

Trig *Trigonometric functions*

Description

Basic trig functions.

Usage

```
## S4 method for signature 'Rcpp_MMPR'
sin(x)

## S4 method for signature 'Rcpp_MMPR'
cos(x)

## S4 method for signature 'Rcpp_MMPR'
tan(x)

## S4 method for signature 'Rcpp_MMPR'
asin(x)

## S4 method for signature 'Rcpp_MMPR'
acos(x)

## S4 method for signature 'Rcpp_MMPR'
atan(x)
```

Arguments

x	An MMPR object.

Value

An MMPR object of the same dimensions as the input.

Examples

```
## Not run:
library(MMPR)

mppr_matrix <- as.MMPR(1:20,nrow=2,ncol=10,"single")
x <- sin(mppr_matrix)

## End(Not run)
```