Package: GPUmatrix (via r-universe)

January 13, 2025

Type Package

Title Basic Linear Algebra with GPU

Version 1.0.2

Description GPUs are great resources for data analysis, especially in statistics and linear algebra. Unfortunately, very few packages connect R to the GPU, and none of them are transparent enough to run the computations on the GPU without substantial changes to the code. The maintenance of these packages is cumbersome: several of the earlier attempts have been removed from their respective repositories. It would be desirable to have a properly maintained R package that takes advantage of the GPU with minimal changes to the existing code. We have developed the GPUmatrix package (available on CRAN). GPUmatrix mimics the behavior of the Matrix package and extends R to use the GPU for computations. It includes single(FP32) and double(FP64) precision data types, and provides support for sparse matrices. It is easy to learn, and requires very few code changes to perform the operations on the GPU. GPUmatrix relies on either the Torch or Tensorflow R packages to perform the GPU operations. We have demonstrated its usefulness for several statistical applications and machine learning applications: non-negative matrix factorization, logistic regression and general linear models. We have also included a comparison of GPU and CPU performance on different matrix operations.

Depends R (>= 4.1)

Imports stats, methods

Suggests torch, tensorflow, Matrix, matrixStats, float, MASS, knitr, rmarkdown

VignetteBuilder knitr

License Artistic-2.0

RoxygenNote 7.2.1

Encoding UTF-8

NeedsCompilation no

Author Cesar Lobato-Fernandez [aut, cre], Juan A.Ferrer-Bonsoms [aut], Angel Rubio [aut, ctb]

Maintainer Cesar Lobato-Fernandez <clobatofern@unav.es>

Date/Publication 2024-03-01 09:02:36 UTC

Repository https://clobatofern95.r-universe.dev

RemoteUrl https://github.com/cran/GPUmatrix

RemoteRef HEAD

RemoteSha 99aa24df83ffabf9400e71078e56bca086a1c537

Contents

aperm *Array Transposition*

Description

t returns the transpose of a gpu.matrix-class object.

Usage

```
## S4 method for signature 'gpu.matrix.tensorflow'
t(x)
## S4 method for signature 'gpu.matrix.torch'
t(x)
```

Arguments

x	a gpu.matrix to be transposed.

Value

It returns a transposed version of a. The output is also a gpu.matrix class object.

See Also

For more information: t.

Examples

```
## Not run:

  a <- gpu.matrix(1:9,nrow=3,ncol=3)
  t(a) #transpose of a.


## End(Not run)
```

apply *Apply Functions over 'gpu.matrix-class' margins*

Description

This function mimics the 'base' function 'apply' to operate on gpu.matrix-class objects: It returns a vector or a list of values obtained by applying a function to margins of a GPUmatrix.

Usage

```
## S4 method for signature 'gpu.matrix.tensorflow'
apply(X, MARGIN, FUN, ..., simplify)
## S4 method for signature 'gpu.matrix.torch'
apply(X, MARGIN, FUN, ..., simplify)
```

Arguments

X	a gpu.matrix object.
MARGIN	1 for rows and 2 for columns.
FUN	function to be applied in the operation.
...	general additional parameters. Optional arguments to FUN.
simplify	a logical indicating whether results should be simplified if possible. Note that some methods that can be simplified when working with 'matrix' objects may not always be simplified for gpu.matrix objects. See details.

Details

FUN is found by a call to match.fun as done in the base function apply. Internally, apply will use the functions implemented to work with objects from the GPUmatrix library. If the input gpu.matrix-class object(s) are stored on the GPU, then the operations will be performed on the GPU. See gpu.matrix.

As in apply, the arguments in ... cannot have the same name as any of the other arguments to ensure possible errors.

The parameter simplify indicates wheter the result should be simplified if possible. If the called FUN returns a gpu.matrix-class object, the result cannot be simplified. In these cases, the parameter simplify will work as if it was set to FALSE and the following warning message will be returned: "If the function applied to the GPU matrix returns a tensor or another GPU matrix, then the 'simplify' argument will always be FALSE."

Value

The results of mimics the base function apply.

Each call to FUN will return a vector of length n. If simplify is TRUE and the result can be simplified, then apply will return a numeric vector of dimension c(n,dim(x)[MARGIN]) if n > 1. If n = 1, apply will return a numeric vector of length dim(x)[MARGIN].

If simplify is FALSE, apply will return a list of length `dim(x)[MARGIN]`.

Note that if `simplify` is TRUE and the result of `FUN` is an object of class gpu.matrix, then the result cannot be simplified, so it will return a list of length `dim(x)[MARGIN]` and each element of this list will be of class gpu.matrix.

For more details see `apply`

See Also

For more information see: `apply`

Examples

```
if(installTorch()){

  a <- gpu.matrix(rnorm(9),3,3)

  apply(a, 1, mean) #computes the mean of each row
  apply(a, 2, mean) #computes the mean of each column

}
```

`as_methods` *as_methods*

Description

These functions mimic the 'base' functions of R that have the same name to operate on gpu.matrix-class objects:

Function `as.matrix` attempts to turn its argument into a matrix. Function `as.list` attempts to turn its argument into a list. Function `as.numeric` attempts to turn its argument into a numeric. Function `as.array` attempts to turn its argument into an array. Function `as.vector` attempts to turn its argument into a vector. Function `is.numeric` is a general test of an object being interpretable as numbers.

Usage

```
## S4 method for signature 'gpu.matrix.tensorflow'
as.array(x,...)
## S4 method for signature 'gpu.matrix.torch'
as.array(x,...)
## S4 method for signature 'gpu.matrix.tensorflow'
as.list(x,...)
## S4 method for signature 'gpu.matrix.torch'
```

```
as.list(x,...)
## S4 method for signature 'gpu.matrix.tensorflow'
as.matrix(x,...)
## S4 method for signature 'gpu.matrix.torch'
as.matrix(x,...)
## S4 method for signature 'gpu.matrix.tensorflow'
as.numeric(x,...)
## S4 method for signature 'gpu.matrix.torch'
as.numeric(x,...)
## S4 method for signature 'gpu.matrix.tensorflow'
as.vector(x,mode)
## S4 method for signature 'gpu.matrix.torch'
as.vector(x,mode)
## S4 method for signature 'gpu.matrix.torch'
is.numeric(x)
## S4 method for signature 'gpu.matrix.tensorflow'
is.numeric(x)
```

Arguments

x	a gpu.matrix object.
...	(generalized) vectors or matrices. These can be given as named arguments.
mode	Argument for as.vector. It mimics the argument of the same name of the base function as.vector: character string naming an atomic mode or "list" or "expression" or (except for vector) "any".

Details

Note that, if the input is a gpu.matrix with complex numbers: the function is.numeric will return FALSE, and the function as.numeric will only returns the real part and the following warning message: "In asMethod(object) : imaginary parts discarded in coercion".

The parameter mode of the function as.vector determines the storage mode of the result. For more details see typeof.

Value

Given a gpu.matrix-class object:

Function as.matrix turns the input gpu.matrix to a 'matrix' object.

Function as.list turns the input gpu.matrix into a list.

Function as.numeric turns the input gpu.matrix into a numeric vector.

Function as.array turns the input gpu.matrix into an array (Since the gpu.matrix objects are always two-dimensional, this function is equivalent to as.matrix.).

Function as.vector turns the input gpu.matrix into a vector.

Function is.numeric returns TRUE or FAALSE if input can be interpretable as numbers.

See Also

numeric, array, list, matrix,

Examples

```
## Not run:
a <- gpu.matrix(c(rnorm(8),2+1i),nrow=3,ncol=3)
as.array(a)
as.list(a)
as.matrix(a)
as.numeric(a)
is.numeric(a)
as.character(a)
as.vector(a,mode = "list")
as.vector(a,mode = "character")
as.vector(a,mode = "logical")
as.vector(a,mode = "integer")
as.vector(a,mode = "double")
as.vector(a,mode = "complex")
as.vector(a,mode = "raw")



## End(Not run)
```

---

cbind_rbind_methods *cbind_rbind_methods*

Description

Mimics the 'base' functions 'cbind' and 'rbind' to operate on gpu.matrix objects. The 'base' functions 'cbind' and 'rbind' internally call the methods cbind2 and rbind2.

Therefore, ss done in cbind2 of the package 'Matrix', we have defined in 'GPUmatrix' the methods cbind2 and rbind2 to operate on gpu.matrix objects too.

Usage

```
## S4 method for signature 'ANY,gpu.matrix.tensorflow'
cbind2(x,y)
## S4 method for signature 'ANY,gpu.matrix.torch'
rbind2(x,y)
## S4 method for signature 'gpu.matrix.tensorflow,ANY'
cbind2(x,y,...)
## S4 method for signature 'gpu.matrix.torch,ANY'
rbind2(x,y)
```

Arguments

x, y	a gpu.matrix object or any other matrix class.
...	(generalized) vectors or matrices. These can be given as named arguments.

Value

The result of using these functions is equivalent to using the basic cbind and rbind functions. For more details see cbind.

Note that if one of the input values is a gpu.matrix-class object, then the output will also be a gpu.matrix-class object.

The data type of the values of the resulting gpu.matrix-class object (corresponding to the dtype parameter of the gpu.matrix function) is the one that allows the integration of all input values. That is, if you call cbind(a,b) where a is a gpu.matrix-class object with values of "int32" and b is a gpu.matrix-class with values of "float64", the result will be a gpu.matrix-class with values of "float64".

See Also

cbind, rbind, cbind2

Examples

```
## Not run:

a <- gpu.matrix(1:9,nrow=3,ncol=3)

#add new row
newrow <- c(1,2,3)
a <- rbind2(a,newrow)

#add new column
newcolumn <- c(1,2,3,4)
a <- cbind(a,newcolumn)

#add new rows from other gpu.marix
b <- gpu.matrix(1:16,nrow=4,ncol=4)
d <- rbind(a,b)

#add new columns from other gpu.marix
b <- gpu.matrix(1:16,nrow=4,ncol=4)
d <- cbind(a,b)



## End(Not run)
```

concatenate_gpu.matrix

concatenate_gpu.matrix

Description

Mimics the 'base' function `'c'` to operate on `gpu.matrix` objects: function which "combines its arguments to form a vector. All arguments are coerced to a common type which is the type of the returned value." In most of the cases, the returned object is of type 'numeric'.

Usage

```
## S4 method for signature 'gpu.matrix.tensorflow'
c(x,...,recursive)
## S4 method for signature 'gpu.matrix.torch'
c(x,...,recursive)
## S4 method for signature 'numMatrixLike'
c(x,...,recursive)
```

Arguments

`x`	A `gpu.matrix` object
`...`	objects to be concatenated.
`recursive`	The same as `c`: Logical. If recursive = TRUE, the function recursively descends through lists (and pairlists) combining all their elements into a vector.

Value

It will return a vector of type 'numeric' with the combined values.

See Also

See also: `c`.

Examples

```
## Not run:

#add new value
a <- gpu.matrix(1:5,nrow=1,ncol=5)
c(a,3)

#add other vector
c(a,a)

#add value to a gpu.matrix
a <- gpu.matrix(1:9,nrow=3,ncol=3)
```

```
c(a,a)
#it will return a vector as in original c function.


## End(Not run)
```

---

cor_cov *Correlation, Variance and Covariance for 'GPUmatrix' objects*

Description

These functions mimic the stats functions cov and cor to compute on gpu.matrix objects: "cov and cor compute the covariance and correlation of x and y if these are vectors. If x and y are matrices then the covariances (or correlations) between the columns of x and the columns of y are computed."

cov2cor scales a covariance matrix into the corresponding correlation matrix efficiently.

Usage

```
## S4 method for signature 'gpu.matrix.tensorflow,ANY,ANY,ANY'
cor(x,y)
## S4 method for signature 'gpu.matrix.tensorflow,ANY,missing,character'
cor(x,y,method)
## S4 method for signature 'gpu.matrix.tensorflow,missing,missing,character'
cor(x,y,method)
## S4 method for signature 'ANY,gpu.matrix.tensorflow,ANY,ANY'
cor(x,y)
## S4 method for signature 'gpu.matrix.tensorflow,missing,ANY,ANY'
cor(x,y)

## S4 method for signature 'ANY,gpu.matrix.torch,ANY,ANY'
cor(x,y)
## S4 method for signature 'gpu.matrix.torch,ANY,ANY,ANY'
cor(x,y)
## S4 method for signature 'gpu.matrix.torch,ANY,missing,character'
cor(x,y,method)
## S4 method for signature 'gpu.matrix.torch,missing,missing,character'
cor(x,y,method)
## S4 method for signature 'gpu.matrix.torch,missing,missing,missing'
cor(x,y)
## S4 method for signature 'gpu.matrix.torch,missing,ANY,ANY'
cor(x,y)
```

```
## S4 method for signature 'gpu.matrix.tensorflow'
cov(x,y)
## S4 method for signature 'ANY,gpu.matrix.tensorflow'
cov(x,y)
## S4 method for signature 'gpu.matrix.tensorflow,ANY'
cov(x,y)
## S4 method for signature 'gpu.matrix.tensorflow,missing'
cov(x,y)


## S4 method for signature 'gpu.matrix.torch'
cov(x,y)
## S4 method for signature 'ANY,gpu.matrix.torch'
cov(x,y)
## S4 method for signature 'gpu.matrix.torch,ANY'
cov(x,y)
## S4 method for signature 'gpu.matrix.torch,missing'
cov(x,y)

## S4 method for signature 'gpu.matrix.tensorflow'
cov2cor(V)
## S4 method for signature 'gpu.matrix.torch'
cov2cor(V)
```

Arguments

x	a gpu.matrix.
y	NULL (default) or a vector, matrix, data frame or gpu.matrix with compatible dimensions to x.
method	a character string indicating which correlation coefficient (or covariance) is to be computed. One of "pearson" (default) or "spearman".
V	symmetric numeric gpu.matrix, usually positive definite such as a covariance matrix.

Details

These functions work in the same way as their counterparts in the 'stats' library. Note that the 'Kendal' method (implemented in the 'stats' library) is not available for working with gpu.matrix-class objects.

Notice that the inputs can be either an object of class 'matrix', 'Matrix' or 'gpu.matrix'. User must be sure that the input values must be numeric.

If the input gpu.matrix-class object(s) are stored on the GPU, then the operations will be performed on the GPU. See gpu.matrix. The result will be a gpu.matrix object.

For more details see cor and cov2cor.

Value

The result obtained by applying these functions will be a gpu.matrix object. For each function the result will be:

- cor correlation between x and y (when two vectors are the input) or the correlation between the columns of x and y if x and y are a gpu.matrix class object. If y is empty, is equivalent to y=x.

- cov the same as cor but compute the covariance.

- cov2cor scales a covariance matrix into the corresponding correlation matrix efficiently.

See Also

For more information: cor, cov, cov2cor,

Examples

```
## Not run:
a <- gpu.matrix(rnorm(10))
b <- gpu.matrix(rnorm(10))
cor(a,b)

#example taken from stats corresponding help page:
longley_matrix <- as.matrix(longley)
longley_gpu <- as.gpu.matrix(longley_matrix)
C1 <- cor(longley_gpu)
cov(longley_gpu)
cov2cor(cov(longley_gpu))



## End(Not run)
```

---

density *Kernel Density Estimation and Histograms*

Description

The function density mimics the function density of the library stats to operate on gpu.matrix-class objects: "It computes kernel density estimates. Its default method does so with the given kernel and bandwidth for univariate observations."

The function 'hist' mimics the function 'hist' of the library 'graphics' to operate on gpu.matrix-class objects: "It computes a histogram of the given data values."

Usage

```
## S4 method for signature 'gpu.matrix.tensorflow'
density(x)
## S4 method for signature 'gpu.matrix.torch'
density(x)
## S4 method for signature 'gpu.matrix.tensorflow'
hist(x,...)
## S4 method for signature 'gpu.matrix.torch'
hist(x,...)
```

Arguments

x	the gpu.matrix object from which the estimate density is to be computed or the histogram is desired.
...	further arguments and graphical parameters.

Details

The two functions (density and hist) have been programmed to call their corresponding counterpart functions with their default parameters. Therefore, the internal operations to obtain each graph are computed by the CPU, regardless of whether the input value is stored in the GPU.

For more information on these functions see density, and hist.

Value

The function density returns the same output as its counterpart function density from the library stats: It returns "an object with class 'density' whose underlying structure is a list containing the following components.

x	the n coordinates of the points where the density is estimated.
y	the estimated density values. These will be non-negative, but can be zero.
bw	the bandwidth used.
n	the sample size after elimination of missing values.
call	the call which produced the result.
data.name	the deparsed name of the x argument.
has.na	logical, for compatibility (always FALSE).

The print method reports summary values on the x and y components." (taken from density).

On the other hand, the function hist returns the same output as its counterpart function hist from the library graphics: It returns "an object of class 'histogram' which is a list with components:

breaks	the n+1n+1 cell boundaries (= breaks if that was a vector). These are the nominal breaks, not with the boundary fuzz.
counts	n integers; for each cell, the number of x[] inside.

density	values $\hat{f}(x_i)$, as estimated density values. If all(diff(breaks) == 1), the are the relative frequencies counts/n and in general satisfy $\sum_i \hat{f}(x_i)(b_{i+1} - b_i) = 1$, where b_i = breaks[i]
.	
mids	the n cell midpoints.
xname	a character string with the actual x argument name.
equidist	logical, indicating if the distances between breaks are all the same."

(Taken from hist)

See Also

For more information see: density, and hist

Examples

```
if(installTorch()){

  a <- gpu.matrix(rnorm(20*100),20,100)

  density(a[1,]) #density information
  plot(density(a[1,])) #plot the estimated density function

  hist(a[1,]) #plot the histogram

}
```

---

det *Calculate the Determinant of a 'GPUMatrix'*

Description

These functions mimic the 'base' functions det and determinant to operate on gpu.matrix-class objects: "det calculates the determinant of a matrix. determinant is a generic function that returns separately the modulus of the determinant, optionally on the logarithm scale, and the sign of the determinant."

Usage

```
## S4 method for signature 'gpu.matrix.tensorflow,logical'
determinant(x,logarithm,...)
## S4 method for signature 'gpu.matrix.tensorflow,missing'
determinant(x,logarithm,...)
## S4 method for signature 'gpu.matrix.torch,logical'
determinant(x,logarithm,...)
```

```
## S4 method for signature 'gpu.matrix.torch,missing'
determinant(x,logarithm,...)

## S4 method for signature 'gpu.matrix.tensorflow'
det(x,...)
## S4 method for signature 'gpu.matrix.torch'
det(x,...)
```

Arguments

x	a gpu.matrix object.
...	Optional parameters. For more details seee det
logarithm	logical; if TRUE (default) return the logarithm of the modulus of the determinant.

Details

The function det and determinant internally call the corresponding function of the library torch or tensorflow (depending on the type of input gpu.matrix-class).

If the input gpu.matrix-class object(s) are stored on the GPU, then the operations will be performed on the GPU. See gpu.matrix.

Value

det returns the same output corresponding to the base function det, which is the determinant of x. The returned value is a object of class numeric sotred in the cpu.

determinant returns the corresponding output of the base function determinant, which is an object of class det, that contains the following components:

modulus	a numeric value. The modulus (absolute value) of the determinant if logarithm is FALSE; otherwise the logarithm of the modulus.
sign	integer; either +1 or -1 according to whether the determinant is positive or negative.

See Also

For more information see: det.

Examples

```
## Not run:

x <- gpu.matrix(1:4,nrow=2, ncol = 2)
determinant(x) #modulus of the determinant.
det(x)#the determinant.


## End(Not run)
```

diag *diag*

Description

This function mimics the base function 'diag' to operate on gpu.matrix-class objects: "extract or replace the diagonal of a matrix, or constructs a diagonal matrix."

Usage

```
## S4 method for signature 'gpu.matrix.tensorflow'
diag(x)
## S4 method for signature 'gpu.matrix.torch'
diag(x)
## S4 replacement method for signature 'gpu.matrix.tensorflow,numeric'
diag(x) <- value
## S4 replacement method for signature 'gpu.matrix.torch,numeric'
diag(x) <- value
```

Arguments

x	a gpu.matrix.
value	either a single value or a vector of length equal to that of the current diagonal.

Value

Output corresponding to the base function diag: If input x is a gpu.matrix-class object then diag{x} returns a numeric object with the diagonal of the matrix x (this output is not a gpu.matrix-class object).

The replacement form diag(x) <- value sets the diagonal of the matrix x to the given value(s).

See Also

For more information see: diag

Examples

```
if(installTorch()){

  a <- gpu.matrix(rnorm(9),nrow=3,ncol=3)

  diag(a) #shows the diagonal of matrix a

  diag(a) <- c(10,0,100) #set the diagonal of matrix a
  a

}
```

dim_and_names *Number of rows and columns and its corresponding names*

Description

These functions mimic the 'base' functions `rownames`, `colnames`, `dimnames`, `dim`, `length`, `ncol`, `nrow` to operate on gpu.matrix-class objects.

The "`dim` family functions" set or get the dimension of a gpu.matrix-class object.

The "`rownames` and `colnames` family functions" set or get the corresponding names of rows and columns of a gpu.matrix-class object.

Usage

```
## S4 method for signature 'gpu.matrix.tensorflow'
rownames(x)
## S4 method for signature 'gpu.matrix.torch'
rownames(x)
## S4 method for signature 'gpu.matrix.tensorflow'
colnames(x)
## S4 method for signature 'gpu.matrix.torch'
colnames(x)
## S4 method for signature 'gpu.matrix.tensorflow'
dim(x)
## S4 method for signature 'gpu.matrix.torch'
dim(x)
## S4 method for signature 'gpu.matrix.tensorflow'
dimnames(x)
## S4 method for signature 'gpu.matrix.torch'
dimnames(x)
## S4 method for signature 'gpu.matrix.tensorflow'
length(x)
## S4 method for signature 'gpu.matrix.torch'
length(x)
## S4 method for signature 'gpu.matrix.tensorflow'
ncol(x)
## S4 method for signature 'gpu.matrix.torch'
ncol(x)
## S4 method for signature 'gpu.matrix.tensorflow'
nrow(x)
## S4 method for signature 'gpu.matrix.torch'
nrow(x)
```

```
## S4 replacement method for signature 'gpu.matrix.tensorflow,vector'
dim(x) <- value
## S4 replacement method for signature 'gpu.matrix.torch,vector'
dim(x) <- value
## S4 replacement method for signature 'gpu.matrix.tensorflow,vector'
dimnames(x) <- value
## S4 replacement method for signature 'gpu.matrix.torch,vector'
dimnames(x) <- value
```

Arguments

x	a gpu.matrix.
value	For dim a numeric vector of length 2 with the number of rows and number of columns. For dimnames a character or numeric vector of length 2 with the names of the rows and names of the columns.

Value

rownames returns the names of the rows of a gpu.matrix-class object. colnames returns the names of the columns of a gpu.matrix-class object.

dim returns the number of rows and columns of a gpu.matrix-class object and dim <- sets the number of rows and columns of a gpu.matrix-class object.

dimnames returns the names of the rows and columns of a gpu.matrix-class object and dimnames <- sets the names of the rows and columns of a gpu.matrix-class object.

length returns the length (ncol*nrow) of a gpu.matrix-class object.

ncol returns the number of columns of a gpu.matrix-class object.

nrow returns the number of rows of a gpu.matrix-class object.

See Also

For more information: rownames, colnames, dim, dim<-, dimnames, dimnames<-, length, ncol, nrow.

Examples

```
## Not run:

a <- gpu.matrix(rnorm(9))

dim(a) <- c(3,3) #sets the number of rows and columns.
dim(a) #shows the number of rows and the number of columns
ncol(a) #shows the number of columns
nrow(a) #shows the number of rows
length(a) #shows the lenght of the matrix (nrow*ncol)
```

```
dimnames(a) <- list(c("r1","r2","r3"),c("c1","c2","c3")) #sets rows and column names
dimnames(a) #shows both the row and the col names

#these functions are equivalent to the following:
rownames(a) <- c("r1","r2","r3") #adds rownames to a.
colnames(a) <- c("c1","c2","c3") #adds colnames to a.
rownames(a) #shows rownames.
colnames(a) #shows colnames.


## End(Not run)
```

`dist` *Distance Matrix Computation with GPU*

Description

This function mimics the 'stats' function `dist`: 'computes and returns the distance matrix computed by using the specified distance measure to compute the distances between the rows of a data matrix.'

Usage

```
dist(x, method = "euclidean", diag = FALSE,
    upper = FALSE, p = 2)

## S4 method for signature 'gpu.matrix.torch'
dist(x,method,diag,upper,p)
```

Arguments

`x`	a `gpu.matrix`.
`method`	the same as the 'stats' function `dist`: the distance measure to be used. Could be "euclidean", "maximum", "manhattan" or "minkowski". Note that the "canberra", "binary" methods are not included.
`diag`	the same as the 'stats' function `dist`: logical value indicating if the diagonal of the distances should be printed. It is set to TRUE and cannot be changed.
`upper`	the same as the 'stats' function `dist`: logical value indicating whether the upper triangle of the distance matrix should be printed. It is set to TRUE and cannot be changed.
`p`	the same as the 'stats' function `dist`: The power of the Minkowski distance.

Details

The function mimics the 'stat' function `dist`. The distance measures used are (taken from `dist`):

euclidean: $\sqrt{\sum_i (x_i - y_i)^2}$

maximum: Maximum distance between two components of `x` and `y`.

manhattan: Absolute distance between the tow vectors.

minkowski: the `p` norm: the ppth root of the sum of the ppth powers of the differences of the components.

For more details see `dist`.

The function `dist` internally calls the corresponding function of the library torch or tensorflow (depending on the type of input gpu.matrix-class).

If the input gpu.matrix-class object is stored on the GPU, then the operations will be performed on the GPU. See `gpu.matrix`.

Value

The function returns a gpu.matrix-class object with the corresponding distances between the rows of the input gpu.matrix object.

See Also

For more information see: `dist`, and `torch_cdist`.

Examples

```
## Not run:

#the example compare the results with the
#'stats' function 'dist':

x <- matrix(rnorm(100), nrow = 5)

dist(x,diag = TRUE,upper = TRUE,method = "euclidean")
dist(x = as.gpu.matrix(x),method = "euclidean")

dist(x,diag = TRUE,upper = TRUE,method = "maximum")
dist(x = as.gpu.matrix(x),method = "maximum")

dist(x,diag = TRUE,upper = TRUE,method = "manhattan")
dist(x = as.gpu.matrix(x),method = "manhattan")

dist(x,diag = TRUE,upper = TRUE,method = "minkowski")
dist(x = as.gpu.matrix(x),method = "minkowski")

dist(x,diag = TRUE,upper = TRUE,method = "minkowski",p = 23)
dist(x = as.gpu.matrix(x),method = "minkowski",p = 23)


## End(Not run)
```

expmGPU *'GPUmatrix' Exponential*

Description

This function mimics the function expm of the library Matrix to operate on gpu.matrix-class objects: It "computes the exponential of a matrix."

Usage

```
expmGPU(x)
## S4 method for signature 'gpu.matrix.tensorflow'
expmGPU(x)
## S4 method for signature 'gpu.matrix.torch'
expmGPU(x)
```

Arguments

x	a gpu.matrix.

Details

The exponential of a matrix is computed as: $\sum_{k=0}^{\infty} 1/k! X^k$.

The function expmGPU internally calls the corresponding function of the library torch or tensorflow (depending on the type of input gpu.matrix-class).

If the input gpu.matrix-class object(s) are stored on the GPU, then the operations will be performed on the GPU. See gpu.matrix.

Please note that this function works with float numbers (either float32 or float64). If the data type of x is integer, this function will not work. An example is shown below.

Value

The matrix exponential of x as gpu.matrix class.

See Also

For more information see expm, and torch_matrix_exp.

Examples

```
  ## Not run:
#build with a matrix that contains int number. It will  not work.
x <- gpu.matrix(1:9,nrow=3,ncol = 3,dtype = "int")
x
try(expmGPU(x))

#need to be float and not int
```

```
x <- gpu.matrix(1:9,nrow=3,ncol = 3,dtype = "float64")
expmGPU(x)


## End(Not run)
```

---

extract_gpu.matrix *extract_gpu.matrix*

Description

These operators mimic the base operators `[`, `[<-`, `[[`, and `[[<-` to compute on gpu.matrix-class objects.

Usage

```
## S4 method for signature 'gpu.matrix.tensorflow,missing'
e1 - e2
## S4 method for signature 'gpu.matrix.torch,missing'
e1 - e2
## S4 method for signature 'gpu.matrix.tensorflow,index,index'
x[i,j]
## S4 method for signature 'gpu.matrix.tensorflow,index,missing'
x[i,j,...,drop = TRUE]
## S4 method for signature 'gpu.matrix.tensorflow,matrix,missing'
x[i,j,...,drop = TRUE]
## S4 method for signature 'gpu.matrix.tensorflow,missing,index'
x[i,j]
## S4 method for signature 'gpu.matrix.torch,index,index'
x[i,j]
## S4 method for signature 'gpu.matrix.torch,index,missing'
x[i,j,...,drop = TRUE]
## S4 method for signature 'gpu.matrix.torch,matrix,missing'
x[i,j,...,drop = TRUE]
## S4 method for signature 'gpu.matrix.torch,missing,index'
x[i,j]
## S4 replacement method for signature 'gpu.matrix.tensorflow,index,index'
x[i,j] <- value
## S4 replacement method for signature 'gpu.matrix.tensorflow,index,missing'
x[i,j] <- value
## S4 replacement method for signature 'gpu.matrix.tensorflow,matrix,missing'
x[i,j] <- value
## S4 replacement method for signature 'gpu.matrix.tensorflow,missing,index'
x[i,j] <- value
## S4 replacement method for signature 'gpu.matrix.torch,index,index'
```

```
x[i,j] <- value
## S4 replacement method for signature 'gpu.matrix.torch,index,missing'
x[i,j] <- value
## S4 replacement method for signature 'gpu.matrix.torch,matrix,missing'
x[i,j] <- value
## S4 replacement method for signature 'gpu.matrix.torch,missing,index'
x[i,j] <- value
## S4 method for signature 'gpu.matrix.tensorflow,index'
x[[i,j,...]]
## S4 method for signature 'gpu.matrix.torch,index'
x[[i,j,...]]
## S4 replacement method for signature 'gpu.matrix.tensorflow,index'
x[[i]] <- value
## S4 replacement method for signature 'gpu.matrix.torch,index'
x[[i]] <- value
```

Arguments

e1	a gpu.matrix.
e2	a gpu.matrix.
x	a gpu.matrix object from which extract element(s) or in which to replace element(s)
i, j, ...	indices specifying elements to extract or replace.
value	typically an array-like R object of a similar class as x.
drop	For matrices and arrays. If TRUE the result is coerced to the lowest possible dimension

Details

When replacing a value or values in a gpu.matrix, the gpu.matrix will not change its datatype (corresponding to the parameter dtype of the function gpu.matrix) based on the datatype in value. For example, the code x[1,1] <- value where the array x is a gpu.matrix with integer values and value has 'double' values, only the integer part of value will be stored in x[1,1].

See Also

See Also Extract.

Examples

```
## Not run:

a <- gpu.matrix(1:9,nrow=3,ncol=3)
rownames(a) <- c("R1","R2","R3")
colnames(a) <- c("C1","C2","C3")

#return
```

```
a[3,3] # the element row 3 and column 3
a[6] # the 6th element
a[1,] # the first row
a[c(1,2),] # the first and second row
a[c(1,1),] # the first row twice
a[,1] # the first column
a[,c(1,2)] # the first and second column
a[,c(1,1)] # the first column twice

#replace
a[3,3] <- 100 # replace the element 3,3
a[1,] <- c(1,2,1) # replace the first row
a[,2] <- c(0,0,0) # replace the second column
a[c(1,2),] <- matrix(1:6,nrow = 2) # replace the first and second row




## End(Not run)
```

`fft` *Fast Discrete Fourier Transform (FFT)*

Description

The function `fft` mimics the function `fft` of the library 'stats' to compute on gpu.matrix-class objects: it "Computes the Discrete Fourier Transform (DFT) of an array with a fast algorithm, the 'Fast Fourier Transform' (FFT)."

The function `mvfft` mimics the function `mvfft` of the library 'stats' which: "takes a real or complex matrix as argument, and returns a similar shaped matrix, but with each column replaced by its discrete Fourier transform".

Usage

```
## S4 method for signature 'gpu.matrix.tensorflow'
fft(z)
## S4 method for signature 'gpu.matrix.torch'
fft(z)
## S4 method for signature 'gpu.matrix.torch,logical'
fft(z,inverse)

## S4 method for signature 'gpu.matrix.torch'
mvfft(z)
## S4 method for signature 'gpu.matrix.tensorflow'
mvfft(z)
## S4 method for signature 'gpu.matrix.torch,logical'
mvfft(z,inverse)
```

Arguments

z	a gpu.matrix object containing the values to be transformed.
inverse	the same as in the library 'stats': "if TRUE, the unnormalized inverse transform is computed (the inverse has a +in the exponent of e, but here, we do not divide by 1/length(x))". By default is FALSE. Plea Note that this parameter only work for torch.

Details

The function fft mimics the function fft to operate on gpu.matrix-class objects of one dimension. If the input gpu.matrix z has tow dimensions the function will not work, as the method for two dimensions is not implemented yet for gpu.matrix-class objects. In this case the function will display the following error message: "FFT in gpu.matrix with 2 dimensions is not allowed yet".

The function mvfft mimics the function mvfft to operate on gpu.matrix-class objects. This function will apply the discrete Fourier transform to each column of the input z matrix.

Note that the inverse parameter only works for 'torch' for both fft and mvfft functions.

The functions fft and mvfft internally call the corresponding function of the library torch or tensorflow (depending on the type of input gpu.matrix-class).

If the input gpu.matrix-class object(s) are stored on the GPU, then the operations will be performed on the GPU. See gpu.matrix.

Value

It returns a gpu.matrix-class object with the transformed values. To access the real and imaginary information use the function Re() for teh rea part and Im() for the imaginary part. Furthermore, the following code can be used: output@gm$real for the real part and output@gm$imag for the imaginary part.

See Also

For more information see: fft, torch_fft_ifft, and torch_fft_fft.

Examples

```
if(installTorch()){

x <- gpu.matrix(1:4,ncol = 1)
output_gpu <- fft(x)
output_matrix <- fft(z = as.matrix(x))

#check results:
Re(output_gpu)
Re(output_matrix)
Im(output_gpu)
Im(output_matrix)


x <- gpu.matrix(1:12,ncol = 3)
```

```
output_gpu <- mvfft(x)
output_matrix <- mvfft(as.matrix(x))

#check results:
Re(output_gpu)
Re(output_matrix)
Im(output_gpu)
Im(output_matrix)
}
```

---

gpu.matrix *create and store a matrix in the GPU*

Description

Mimic the base 'matrix' function to create a gpu.matrix-class object, that could be of class gpu.matrix.torch or gpu.matrix.tensorflow depending on the system installed in the computer.

The matrix created will be stored in the GPU (by default) or in the CPU. The example section explains how to be sure where the matrix is stored.

This function also mimics the function Matrix of the library 'Matrix'.

Usage

```
gpu.matrix(data = NULL, nrow = NULL, ncol = NULL,
           byrow = FALSE,dimnames = NULL,
           dtype=NULL, sparse=NULL,colnames=c(),
           rownames=c(),device=NULL,type=NULL)

as.gpu.matrix(x,...)
## S4 method for signature 'ANY'
as.gpu.matrix(x,...)
```

Arguments

data, x	a scalar, vector or matrix (both matrix or Matrix class).
nrow	Number of rows of the matrix. By default the number of rows of data if data is an object of class matrix or Matrix.
ncol	Number of columns of the matrix. By default the number of columns of data if data is an object of class matrix or Matrix.
byrow	The same as function matrix: "logical. If FALSE (the default) the matrix is filled by columns, otherwise the matrix is filled by rows."

dimnames	The same as in function matrix: "A dimnames attribute for the matrix: NULL or a list of length 2 giving the row and column names respectively. An empty list is treated as NULL, and a list of length one as row names. The list can be named, and the list names will be used as names for the dimensions."
dtype	data type. User can indicate "float64", "float32" or "int" for "int64". if not specified, dtype will correspond to the input data type.
sparse	The same as in function Matrix of the library 'Matrix': "logical or NULL, specifying if the result should be sparse or not. By default, it is made sparse when more than half of the entries are 0."
colnames	A vector with the column names.
rownames	A vector with the row names.
type	If the gpu.matrix is 'torch' or "tensorflow". If it is NULL, gpu.matrix will try to create a gpu.matrix.torch object.
device	It indicates the device to load cuda. If not indicated, 'device' will be set to 'cuda' if it is available.
...	additional arguments to be passed to or from methods.

Details

The gpu.matrix function mimics the Matrix function of the 'Matrix' library and the basic matrix function. If tensorflow and/or torch are properly installed and the device parameter is set to "cuda" (by default), then the created gpu.matrix object will be stored on the GPU. The example shows how to check this.

The user can apply to the created gpu.matrix-class object -using the same operators- the basic functions that can be applied to a object of class 'matrix' and/or class 'Matrix'.

It can also work with sparse matrices as the 'Matrix' library.

Value

Returns a GPUmatrix object that can be either "gpu.matrix.tensorflow" or "gpu.matrix.torch". For both torch and tensorflow the functions to be applied to a matrix are the same.

If the gpu.matrix-class object is not sparse it will show on the console the matrix as it is. If the gpu.matrix is sparse, it will return to the console the position where there are number different from zero. The internal values of the matrix can be seen using the operator "@".

If the gpu.matrix-class object contains complex numbers, to access the real and imaginary information use the function Re() for teh rea part and Im() for the imaginary part. Furthermore, the following code can be used: output@gm$real for the real part and output@gm$imag for the imaginary part.

Even if the gpu.matrix-class object is sparse or not, both kind of matrices works equally with all functions.

Author(s)

Cesar Lobato and Angel Rubio.

See Also

See gpu.matrix, Matrix, and matrix.

For more details about the parameter dtype visit dtype

Examples

```
## Not run:
## create a gpu.matrix.torch and check it is stored in the GPU.
a <- gpu.matrix(1:9,nrow=3,ncol=3)
class(a)
a@gm$is_cuda

# the output of class(a) should be:
#[1] "gpu.matrix.torch"
#attr(,"package")
#[1] "GPUmatrix"

#the output of a@gm@device should have a similar shape:
#[1] TRUE

## create a gpu.matrix.torch and check it is stored in the CPU.
a <- gpu.matrix(1:9,nrow=3,ncol=3, device="cpu")
class(a)
a@gm$is_cuda

# the output of class(a) should be:
#[1] "gpu.matrix.torch"
#attr(,"package")
#[1] "GPUmatrix"

#the output of a@gm@device should have a similar shape:
#[1] FALSE

## create a gpu.matrix.tensorflow and check it is stored in the GPU.
a <- gpu.matrix(1:9,nrow=3,ncol=3,type="tensorflow")
class(a)
a@gm$device

# the output of class(a) should be:
#[1] "gpu.matrix.tensorflow"
#attr(,"package")
#[1] "GPUmatrix"

#the output of a@gm@device should have a similar shape:
#[1] "/job:localhost/replica:0/task:0/device:GPU:0"

#create a sparse
a <- gpu.matrix(data=c(0,1,1,0,1,0),nrow = 3,ncol = 2,sparse = T)
a

#create a complex gpu.matrix
```

```
a <- gpu.matrix(data=c(0+1i,1i,1,0,1,0),nrow = 3,ncol = 2)
a




## End(Not run)
```

---

gpu.matrix-class *Class 'gpu.matrix' for matrix stored in GPU*

Description

GPU computational power is a great resource for computational biology specifically in statistics and linear algebra. the gpu.matrix-class is a class of the GPUmatrix package, that store a matrix in the GPU.

The GPUmatrix package is based on S4 objects in R and we have created a constructor function that acts similarly to the default `matrix` constructor in R for CPU matrices. The constructor function is `gpu.matrix` and accepts the same parameters as `matrix`.

Slots

Use the `@` operator to access the different slots:

rownames the row names of the gpu.matrix

colnames the colunm names of the gpu.matrix

gm the corresponding tensor

sparse Logical: indicates if the gpu.matrix is sparse or not

type If it is tensorflow or torch

See Also

See Also `gpu.matrix`, `Matrix`, and `matrix`..

GPUglm *Fitting Generalized Linear Models using GPUmatrix objects*

Description

These functions mimic the functions speedglm and speedglm.wfit of the library 'speedglm' to compute on gpu.matrix-class objects. At the same time, these functions mimic the functions glm, and glm.fit from the library 'stats' to compute on large data sets.

Usage

```
glm.fit.GPU(x, y, intercept = TRUE, weights = NULL, family =
                   gaussian(), start = NULL, etastart = NULL, mustart =
                   NULL, offset = NULL, acc = 1e-08, maxit = 25, k = 2,
                   sparse = NULL, trace = FALSE, dtype = "float64", device =
                   NULL, type = NULL, ...)

GPUglm(...)
```

Arguments

As mentioned in the description, these functions mimic speedglm, so almost every parameter does too. There is only three new parameters explained below.

The common parameters with speedglm:

x	the same as speedglm: the design matrix of dimension n*p where n is the number of observations and p is the number of features. x can be either a 'matrix', 'Matrix' or 'gpu.matrix-class' object.
y	the same as speedglm: a vector of n observations. y can be either a 'matrix', 'Matrix' or 'gpu.matrix-class' object.
intercept	the same as speedglm: Logical. If first column of x should be consider as 'intercept' (default) or not. Notice that seting this parameter TRUE or FALSE will not change the design matrix used to fit the model.
weights	the same as speedglm: an optional vector of 'prior weights' to be used in the fitting process. Should be NULL (default) or a numeric vector.
family	the same as speedglm: a description of the error distribution and link function to be used in the model. For glm.fit.GPU this can be a character string naming a family function, a family function or the result of a call to a family function. (See family for details of family functions.)
start	the same as speedglm: starting values for the parameters in the linear prediction.
etastart	the same as speedglm: starting values for the linear predictor.
mustart	the same as speedglm: starting values for the vector of means.

offset	the same as speedglm: this can be used to specify an a priori known component to be included in the linear predictor during fitting. This should be NULL or a numeric vector of length equal to the number of cases. One or more offset terms can be included in the formula instead or as well, and if more than one is specified their sum is used. See model.offset.
acc	the same as speedglm: tolerance to be used for the estimation (by default equal to: 1e-08).
maxit	the same as speedglm: maximum number of iterations.
k	the same as speedglm: numeric, the penalty per parameter to be used; the default k = 2 is the classical AIC.
sparse	if matrix x is desired to be treated as sparse. Not yet implemented.
trace	If the user wants to see the development of the iterations. By default FALSE
...	For GPUglm: arguments to be used to form the default control argument if it is not supplied directly.

The glm.fit.GPU function internally initialises matrices of the 'GPUmatrix' class by calling the gpu.matrix function. The following parameters correspond to this function:

dtype	parameter of the function gpu.matrix: "data type. User can indicate "float64", "float32" or "int" for "int64"." By default it is set to 'float64'.
device	parameter of the function gpu.matrix:"It indicates the device to load cuda. If not indicated, 'device' will be set to 'cuda' if it is available."
type	parameter of the function gpu.matrix: "If gpu.matrix is 'torch' (by default if type is NULL) or "tensorflow"."

Details

The GPUglm function internally calls the glm function by selecting glm.fit.GPU as the method. The input parameters of the GPUglm function are equivalent to those of the glm function.

If the gpu.matrix-class object(s) are stored on the GPU, then the operations will be performed on the GPU. See gpu.matrix.

Value

Both glmGPU, and glm.fit.GPU returns an object of class "GPUglm". This object can be treated as a list. This object mimics the output of the function speedglm:

coefficients	the estimated coefficients.
logLik	the log likelihood of the fitted model.
iter	the number of iterations of IWLS used.
tol	the maximal value of tolerance reached.
family	the maximal value of tolerance reached.
link	the link function used.
df	the degrees of freedom of the model.
XTX	the product X'X (weighted, if the case).

dispersion	the estimated dispersion parameter of the model.
ok	the set of column indeces of the model matrix where the model has been fitted.
rank	the rank of the model matrix.
RSS	the estimated residual sum of squares of the fitted model.
method	TODO
aic	the estimated Akaike Information Criterion.
offset	he model offset.
sparse	a logical value which indicates if the model matrix is sparse.
deviance	the estimated deviance of the fitted model.
nulldf	the degrees of freedom of the null model.
nulldev	the estimated deviance of the null model.
ngoodobs	the number of non-zero weighted observations.
n	the number of observations.
intercept	a logical value which indicates if an intercept has been used.
convergence	a logical value which indicates if convergence was reached.
terms	the terms object used.
call	the matched call.
xlevels	(where relevant) a record of the levels of the factors used in fitting.

See Also

See also: speedglm and glm.

Also of interest may be the function LR_GradientConjugate_gpumatrix for logistic regression.

Examples

```
## Not run:
require(MASS,quietly = TRUE)
require(stats,quietly = TRUE)

# linear model (example taken from 'glm'):

utils::data(anorexia, package = "MASS")
anorex_glm <- glm(Postwt ~ Prewt + Treat + offset(Prewt),
                  family = gaussian(), data = anorexia)
summary(anorex_glm)

#Using GPUglm:
anorex_GPUglm <- GPUglm(Postwt ~ Prewt + Treat + offset(Prewt),
                        family = gaussian, data = anorexia)
summary(anorex_GPUglm)

#linear model using glm.fit.gpu
x <- model.matrix(~Treat+Prewt,data=anorexia)
```

```
y <- as.matrix(anorexia$Postwt)
s1_glm <- glm.fit(x=x,y=y)
s1_gpu <- glm.fit.GPU(x=x,y=y)

s1_glm$coefficients
s1_gpu$coefficients


# poisson (example taken from 'glm'):
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,9)
treatment <- gl(3,3)
glm.D93 <- glm(counts ~ outcome + treatment, family = poisson())
summary(glm.D93)

gpu.glm.D93 <- GPUglm(counts ~ outcome + treatment, family = poisson())
summary(gpu.glm.D93)

#logistic:
data(menarche)
glm.out <- glm(cbind(Menarche, Total-Menarche) ~ Age, family=binomial(), data=menarche)
summary(glm.out)

glm.out_gpu <- GPUglm(cbind(Menarche, Total-Menarche) ~ Age, family=binomial(), data=menarche)
summary(glm.out_gpu)

#can be also called using glm.fit.gpu:
new_menarche <- data.frame(Age=rep(menarche$Age,menarche$Total))
observations <- c()
for(i in 1:nrow(menarche)){
  observations <- c(observations,rep(c(0,1),c(menarche$Total[i]-menarche$Menarche[i],
                                              menarche$Menarche[i])))
}
new_menarche$observations <- observations
x <- model.matrix(~Age,data=new_menarche)
head(new_menarche)
glm.fit_gpu <- glm.fit.GPU(x=x,y=new_menarche$observations, family=binomial())
summary(glm.fit_gpu)

#GPUmatrix package also include the function 'LR_GradientConjugate_gpumatrix'
lr_gran_sol <- LR_GradientConjugate_gpumatrix(X = x,y = observations)

#check results
glm.out$coefficients
glm.out_gpu$coefficients
glm.fit_gpu$coefficients
lr_gran_sol

## End(Not run)
```

installTorch *installTorch*

Description

This function checks that the torch package is installed correctly.

Usage

```
installTorch
```

kroneker *kroneker Products*

Description

Kroneker product of two gpu.matrix-class objects. This function mimics the 'base' function `'kronecker'` to operate on gpu.matrix-class objects.

Usage

```
## S4 method for signature 'ANY,gpu.matrix.tensorflow'
X %x% Y
## S4 method for signature 'ANY,gpu.matrix.torch'
X %x% Y
## S4 method for signature 'gpu.matrix.tensorflow,ANY'
X %x% Y
## S4 method for signature 'gpu.matrix.torch,ANY'
X %x% Y
```

Arguments

X	A gpu.matrix.
Y	A gpu.matrix or a matrix or a numeric variable.

Details

The function %x% internally calls the corresponding function of the library torch or tensorflow (depending on the type of input gpu.matrix-class).

If the input gpu.matrix-class object(s) are stored on the GPU, then the operations will be performed on the GPU. See gpu.matrix.

See Also

See Also kronecker and torch_kron.

Examples

```
## Not run:

a <- gpu.matrix(1:9,nrow=3,ncol=3)
a %x% diag(1,3)



## End(Not run)
```

`LR_GradientConjugate_gpumatrix` *Logistic Regression with Conjugate Gradient method*

Description

The developed function performs the logistic regression using the Conjugate Gradient method. This method has shown to be very effective for logistic regression of big models [1]. The code is general enough to accommodate standard R matrices, sparse matrices from the 'Matrix' package and, more interestingly, gpu.matrix-class objects from the GPUmatrix package.

Usage

```
LR_GradientConjugate_gpumatrix(X, y, beta = NULL,
                               lambda = 0, iterations = 100,
                               tol = 1e-08)
```

Arguments

`X`	the design matrix. Could be either a object of class `gpu.matrix`, `matrix`, or `Matrix`.
`y`	vector of observations.
`beta`	initial solution.
`lambda`	numeric. Penalty factor also known as the L2 norm or L2 penalty, which is computed as the sum of the squared coefficients: $\lambda \|\|\beta_i\|\|_2^2$
`iterations`	maximum number of iterations.
`tol`	tolerance to be used for the estimation.

Details

If the input gpu.matrix-class object(s) are stored on the GPU, then the operations will be performed on the GPU. See `gpu.matrix`.

Value

The function returns a vector containing the values of the coefficients. This returned vector will be a 'matrix', 'Matrix' or 'gpu.matrix-class' object depending on the class of the object X.

Author(s)

Angel Rubio and Cesar Lobato.

References

[1] Minka TP (2003). "A comparison of numerical optimizers for logistic regression." URL: https://tminka.github.io/papers/lo
logreg.pdf.

See Also

See also: GPUglm

Examples

```
## Not run:

#toy example:
set.seed(123)
m <- 1000
n <- 100
x <- matrix(runif(m*n),m,n)
sol <- rnorm(n)
y <- rbinom(m, 1, prob = plogis(x%*%sol))
s2_granConj <- LR_GradientConjugate_gpumatrix(X = x,y = y)

#the following compares LR_GradientConjugate_gpumatrix
# with glm and GPUglm:

require(MASS)
require(stats,quietly = TRUE)
#logistic:
data(menarche)
glm.out <- glm(cbind(Menarche, Total-Menarche) ~ Age, family=binomial(), data=menarche)
summary(glm.out)

glm.out_gpu <- GPUglm(cbind(Menarche, Total-Menarche) ~ Age, family=binomial(), data=menarche)
summary(glm.out_gpu)

#can be also called using glm.fit.gpu:
new_menarche <- data.frame(Age=rep(menarche$Age,menarche$Total))
observations <- c()
for(i in 1:nrow(menarche)){
  observations <- c(observations,rep(c(0,1),c(menarche$Total[i]-menarche$Menarche[i],
                                              menarche$Menarche[i])))
}
new_menarche$observations <- observations
x <- model.matrix(~Age,data=new_menarche)
```

```
head(new_menarche)
glm.fit_gpu <- glm.fit.GPU(x=x,y=new_menarche$observations, family=binomial())
summary(glm.fit_gpu)

#GPUmatrix package also include the function 'LR_GradientConjugate_gpumatrix'
lr_gran_sol <- LR_GradientConjugate_gpumatrix(X = x,y = observations)

#check results
glm.out$coefficients
glm.out_gpu$coefficients
glm.fit_gpu$coefficients
lr_gran_sol


## End(Not run)
```

`matrix-product` *Matrix Products*

Description

Mimic of the 'base' functions `%*%`, `crossprod`, `tcrossprod` to operate on gpu.matrix-class objects.

Usage

```
## S4 method for signature 'gpu.matrix.tensorflow,ANY'
x %*% y
## S4 method for signature 'gpu.matrix.torch,ANY'
x %*% y

## S4 method for signature 'gpu.matrix.tensorflow,ANY'
crossprod(x, y,...)
## S4 method for signature 'gpu.matrix.tensorflow,missing'
crossprod(x, y = NULL,...)

## S4 method for signature 'gpu.matrix.tensorflow,ANY'
tcrossprod(x, y,...)
## S4 method for signature 'gpu.matrix.tensorflow,missing'
tcrossprod(x, y = NULL,...)


## S4 method for signature 'gpu.matrix.torch,ANY'
crossprod(x, y,...)
## S4 method for signature 'gpu.matrix.torch,missing'
crossprod(x, y = NULL,...)

## S4 method for signature 'gpu.matrix.torch,ANY'
```

```
tcrossprod(x, y,...)
## S4 method for signature 'gpu.matrix.torch,missing'
tcrossprod(x, y = NULL,...)
```

Arguments

x	a gpu.matrix.
y	a gpu.matrix, 'matrix' or 'Matrix' object. For the functions tcrossprod and crossprod is NULL (by default), that is equivalent to x=y.
...	potentially more arguments passed to and from methods.

Details

Internally, these functions call the appropriate tensorflow or torch function to perform the matrix product (depending on the type of input gpu.matrix-class).

If the input gpu.matrix-class object(s) are stored on the GPU, then the operations will be performed on the GPU. See gpu.matrix.

Value

A gpu.matrix-class object with the result of the matrix product.

Methods

%*% signature(x = "gpu.matrix.tensorflow", y = "ANY"): Matrix multiplication

crossprod signature(x = "gpu.matrix.tensorflow", y = "ANY"): Matrix multiplication

crossprod signature(x = "gpu.matrix.tensorflow", y = "missing"): Matrix multiplication

tcrossprod signature(x = "gpu.matrix.tensorflow", y = "ANY"): Matrix multiplication

tcrossprod signature(x = "gpu.matrix.tensorflow", y = "missing"): Matrix multiplication

%*% signature(x = "gpu.matrix.torch", y = "ANY"): Matrix multiplication

crossprod signature(x = "gpu.matrix.torch", y = "ANY"): Matrix multiplication

crossprod signature(x = "gpu.matrix.torch", y = "missing"): Matrix multiplication

tcrossprod signature(x = "gpu.matrix.torch", y = "ANY"): Matrix multiplication

tcrossprod signature(x = "gpu.matrix.torch", y = "missing"): Matrix multiplication

See Also

tcrossprod in R's base, and crossprod and %*%. **Matrix** package %&% for boolean matrix product methods. Also see torch_matmul

Examples

```
## Not run:
a <- gpu.matrix(rnorm(12),nrow=4,ncol=3)
b <- t(a)
b
crossprod(a,a)

b <- a
b
tcrossprod(a)



## End(Not run)
```

`matrix_decomposition` *Decomposition of a matrix with GPU*

Description

These functions mimic the functions `eigen` ,`svd`,`chol` to operate on gpu.matrix-class objects:

`'eigen'` mimics the base function `'eigen'` that "computes the eigenvalues and eigenvectors of a numeric (double, integer, logical) or complex matrix."

`'svd'` mimics the base function `'svd'` that "computes the singular-value decomposition of a rectangular matrix."

`'chol'` mimics the base function `'chol'` that "computes Compute the Cholesky factorization of a real symmetric positive-definite square matrix."

Usage

```
## S4 method for signature 'gpu.matrix.tensorflow'
eigen(x)
## S4 method for signature 'gpu.matrix.torch'
eigen(x)

## S4 method for signature 'gpu.matrix.tensorflow'
svd(x)
## S4 method for signature 'gpu.matrix.torch'
svd(x)

## S4 method for signature 'gpu.matrix.tensorflow'
chol(x)
## S4 method for signature 'gpu.matrix.torch'
chol(x)
```

Arguments

x	a gpu.matrix. X must fulfil certain characteristics depending on the function to be called (see details).

Details

These functions mimic the behaviour of their respective 'base' functions.

In the case of the eigen function, the input value can be a numeric or complex gpu.matrix class.

For svd function, the input value could be a numeric or complex gpu.matrix-class object.

For chol function, the input must be a positive-definite squere matrix.

Internally, these functions call its corresponding function of the tensorflow or torch library depending on the type of input gpu.matrix-class.

If the input gpu.matrix-class object(s) are stored on the GPU, then the operations will be performed on the GPU. See gpu.matrix.

Value

The output of these functions correspond to their equivalent base functions:

eigen mimics the base function eigen that computes the eigenvalues and eigenvectors of a numeric (double, integer, logical) or complex matrix. It returns a list with the following items:

values	a vector with the P eigenvalues of x
vectors	the eigenvectors of x

svd mimics the base function svd that computes the singular-value decomposition of a rectangular matrix. It returns a list with the following items:

d	a vector containing the singular values of x
u	a matrix whose columns contain the left singular vectors of x
v	a matrix whose columns contain the right singular vectors of x

chol mimics the base function chol that computes Compute the Cholesky factorization of a real symmetric positive-definite square matrix. It returns a gpu.matrix-class object with The upper triangular factor of the Cholesky decomposition, i.e., the matrix R such that $R'R = X$.

See Also

For more information see: eigen, svd, chol, linalg_eig, torch_svd, and torch_cholesky.

chol function is called by the function chol_solve.

For the qr decomposition see qr.

Examples

```
## Not run:
a <- gpu.matrix(rnorm(9),3,3)
ein <- eigen(a) #eigenvalues and eigenvectors
svd_return <- svd(a) #svd of gpu.matrix a

ata <- tcrossprod(a)
#ata is a real symmetric positive-definite square matrix.
chol(ata) #cholesky decomposition.



## End(Not run)
```

`matrix_general_operators_methods`
Return the first or last part of a GPUmatrix object

Description

`head` and `tail` mimic the functions `head` and `tail` from `utils` to operate on gpu.matrix-class objects. By default `head` shows the first 6 rows of a matrix or first 6 elements of a vector or list. `tail` shows the last 6 rows of a matrix or last 6 elements of a vector or list.

The function `show` mimics the function `show` of `methods` to compute on gpu.matrix-class objects: "It display the object, by printing, plotting or whatever suits its class."

The function `print` mimics the base function `print` to operate on gpu.matrix-class objects.

Usage

```
## S4 method for signature 'gpu.matrix.tensorflow'
tail(x,...)
## S4 method for signature 'gpu.matrix.torch'
tail(x,...)
## S4 method for signature 'gpu.matrix.tensorflow'
show(object)
## S4 method for signature 'gpu.matrix.torch'
show(object)
## S4 method for signature 'gpu.matrix.tensorflow'
head(x,...)
## S4 method for signature 'gpu.matrix.torch'
head(x,...)

## S4 method for signature 'gpu.matrix.torch'
print(x)
```

Arguments

x, object	a gpu.matrix.
...	arguments to be passed to or from other methods.

See Also

For more information see: head, tail, and show.

Examples

```
## Not run:
a <- gpu.matrix(rnorm(20*5),20,5)
head(a) #shows the first six row of every column
tail(a) #shows the las six row of every column

show(a) #show all the object
a #equivalente to run the function show.

## End(Not run)
```

---

matrix_ranges *Get different statistics for a gpu.matrix-class.*

Description

Functions to summarise different values of a gpu.matrix-class object by rows or columns. Specifically: the maximum value, the index of the maximum value, the minimum value, the index of the minimum value, the mean, the variance, the sum of the values and the rank of the values.

These functions mimic the corresponding function of 'base', 'matrixStats' and 'Matrix' libraries.

Usage

```
## S4 method for signature 'gpu.matrix.tensorflow'
rowMaxs(x)
## S4 method for signature 'gpu.matrix.torch'
rowMaxs(x)
## S4 method for signature 'gpu.matrix.tensorflow'
colMaxs(x)
## S4 method for signature 'gpu.matrix.torch'
colMaxs(x)
## S4 method for signature 'gpu.matrix.tensorflow'
max(x)
## S4 method for signature 'gpu.matrix.torch'
max(x)
```

```
## S4 method for signature 'gpu.matrix.tensorflow'
rowMins(x)
## S4 method for signature 'gpu.matrix.torch'
rowMins(x)
## S4 method for signature 'gpu.matrix.tensorflow'
colMins(x)
## S4 method for signature 'gpu.matrix.torch'
colMins(x)
## S4 method for signature 'gpu.matrix.tensorflow'
min(x)
## S4 method for signature 'gpu.matrix.torch'
min(x)

## S4 method for signature 'gpu.matrix.tensorflow'
rowMeans(x)
## S4 method for signature 'gpu.matrix.torch'
rowMeans(x)
## S4 method for signature 'gpu.matrix.tensorflow'
colMeans(x)
## S4 method for signature 'gpu.matrix.torch'
colMeans(x)
## S4 method for signature 'gpu.matrix.tensorflow'
mean(x)
## S4 method for signature 'gpu.matrix.torch'
mean(x)

## S4 method for signature 'gpu.matrix.tensorflow'
rowVars(x)
## S4 method for signature 'gpu.matrix.torch'
rowVars(x)
## S4 method for signature 'gpu.matrix.tensorflow'
colVars(x)
## S4 method for signature 'gpu.matrix.torch'
colVars(x)

## S4 method for signature 'gpu.matrix.tensorflow'
rowRanks(x)
## S4 method for signature 'gpu.matrix.torch'
rowRanks(x)
## S4 method for signature 'gpu.matrix.tensorflow'
colRanks(x)
## S4 method for signature 'gpu.matrix.torch'
colRanks(x)


## S4 method for signature 'gpu.matrix.tensorflow'
rowSums(x)
## S4 method for signature 'gpu.matrix.torch'
```

```
rowSums(x)
## S4 method for signature 'gpu.matrix.tensorflow'
colSums(x)
## S4 method for signature 'gpu.matrix.torch'
colSums(x)
## S4 method for signature 'gpu.matrix.tensorflow'
sum(x)
## S4 method for signature 'gpu.matrix.torch'
sum(x)
```

Arguments

x	a gpu.matrix.

Details

The value returned by almost each function is a numeric vector stored in the CPU. Only the function rowRanks, colRanks, and sum return a gpu.matrix-class object.

These functions internally calls the corresponding function of the library torch or tensorflow (depending on the type of input gpu.matrix-class). If the input gpu.matrix-class object is stored on the GPU, then the operations will be performed on the GPU. See gpu.matrix.

Value

max, rowMaxs, colMaxs calculate the maximum value of a gpu.matrix-class object, of each row and of each column respectively. which.max determines the location of the maximum value.

min, rowMins, colMins calculate the minimum value of a gpu.matrix-class object, of each row and of each column respectively. which.min determines the location of the minimum value.

mean, rowMeans, colMeans calculate the mean (average) value of a gpu.matrix-class object, of each row and of each column respectively.

rowVars, colVars calculate the variance of each row and of each column of a gpu.matrix-class object respectively.

rowRanks, colRanks: given a gpu.matrix-class object, these functions return a gpu.matrix which rearranges each row and each column into ascending respectively.

rowSums, colSums, sum sum the value of a a gpu.matrix-class object, of each row and of each column respectively.

See Also

For more information:

rowMaxs, colMaxs, max, which.max, and torch_max.

rowMins, colMins, min, which.min, and torch_min.

rowMeans, colMeans, mean, and torch_mean.

rowVars, colVars, and torch_var.

rowRanks, colRanks, and torch_argsort.

rowSums, colSums, sum, and torch_sum.

Examples

```
## Not run:
a <- gpu.matrix(rnorm(9),3,3)

#the maximum value of a:
max(a)

#maximum of value in each row of a:
rowMaxs(a)

#maximum value in each column of a:
colMaxs(a)

#index of the maximum value of a:
which.max(a)

#minimum value of a:
min(a)

#minimum value in each row of a:
rowMins(a)

#minimum value in each column of a:
colMins(a)

#index of the minimum value in a:
which.min(a)

#mean of a:
mean(a)

#mean of each row of a:
rowMeans(a)

#mean of each column of a:
colMeans(a)

#variance of each row of a:
rowVars(a)

#variance of each column of a:
colVars(a)

#sum of all values of a:
sum(a)

#sum of each fow of a:
rowSums(a)

#sum of each column of a:
colSums(a)
```

```
#ranking of each row of a:
rowRanks(a)

#ranking of each columna of a:
colRanks(a)

## End(Not run)
```

NMFgpumatrix *Non negative factorization of a matrix*

Description

The non-negative factorization (NMF) of a matrix is an approximate factorization were an initial matrix V is approximated by the product of two matrix W and H so that,

$V \approx WH$

This function operates in the same way with the 'base' `matrix` objects as with gpu.matrix-class objects, and it does not require any additional changes beyond initializing the input matrix as a gpu.matrix-class object.

Usage

```
NMFgpumatrix(V, k = 10, Winit = NULL,
             Hinit = NULL, tol = 1e-06,
             niter = 100)
```

Arguments

V	a gpu.matrix. Values in V must be ≥ 0.
k	The inner dimension of the product of the matrices W and H. That is, it corresponds to the number of columns in W and the number of rows in H.
Winit	Initial value for matrix W. Initial values for W must be ≥ 0.
Hinit	Initial value for matrix H. Initial values for H must be ≥ 0.
tol	tolerance to be used for the estimation.
niter	maximum number of iterations.

Details

We have implemented our own non-negative matrix factorization (NMF) function using Lee and Seung[1] multiplicative update rule:

$W^{n+1}_{[i,j]} \leftarrow W^{n}_{[i,j]} \frac{(V(H^{n+1})^T)_{[i,j]}}{(W^n H^{n+1}(H^{n+1})^T)_{[i,j]}}$

and

$H^{n+1}_{[i,j]} \leftarrow H^{n}_{[i,j]} \frac{((W^n)^T V)_{[i,j]}}{((W^n)^T W^n H^n)_{[i,j]}}$

to update the W and H respectively.

Note that the values of V must be positive. If any value of V is negative, it will be set to 0. If this happens, the following warning message will be displayed: "The values of V must be positive. Negative values in V are set to 0.

If the user decides to initialise the values of W and H, they must also be positive. If there are negative values, they will be set to 0. The following warning message will be displayed: "The Winit values must be positive. Negative values in Winit are set to 0" if Winit has negative values or "The values of Hinit must be positive. Negative values in Hinit are set to 0" if Winit has negative values.

In addition, Winit and Hinit must also have the correct dimensions. Winit must fulfil two conditions: `nrow(Winit) == nrow(V)` and `ncol(Winit) == k`. If not, the function will stop with the following error message: "The dimensions of the Winit matrix are incorrect. Please check that `nrow(Winit) == nrow(V)` and that `ncol(Winit) == k`". On the other hand, Hinit must fulfil two conditions: `nrow(Winit) == nrow(V)` and that `ncol(Winit) == k`. If not, the function will stop with the following error message: "The dimensions of the Winit matrix are incorrect. Please check that `nrow(Winit) == nrow(V)` and that `ncol(Winit) == k`".

If the input gpu.matrix-class object is stored on the GPU, then the operations will be performed on the GPU. See `gpu.matrix`.

Value

The function returns a list that contains the corresponding matrix W and H. If the input `V` matrix is a gpu.matrix-class object, then both W and H are also gpu.matrix-class objects.

Author(s)

Angel Rubio and Cesar Lobato.

References

[1] Lee, D., Seung, H. Learning the parts of objects by non-negative matrix factorization. Nature 401, 788–791 (1999). https://doi.org/10.1038/44565

Examples

```
## Not run:
library(Matrix)
set.seed(1)
a1 <- gpu.matrix(runif(90),nrow=30,ncol=3)
a2 <- gpu.matrix(runif(30),nrow=3,ncol=10)
V <- a1 %*% a2
b <- NMFgpumatrix(V = V, k=3, tol = 1e-6)

#check result:
image(Matrix(as.matrix(V)))
image(Matrix(as.matrix(b$W %*% b$H)))


## End(Not run)
```

power_of_a_matrix *Compute the kth power of a matrix.*

Description

Comput the kth power of a squere matrix, i.e., multiply the gpu.matrix-class object by itself as many times as user indicates.

Usage

```
## S4 method for signature 'gpu.matrix.tensorflow,numeric'
x %^% k
## S4 method for signature 'gpu.matrix.torch,numeric'
x %^% k
```

Arguments

x	a gpu.matrix.
k	the power of the matrix.

Details

The input x gpu.matrix-class needs to be squere. This function internally call the method %*% as many times as required. If the input gpu.matrix-class object is stored on the GPU, then the operations will be performed on the GPU. See gpu.matrix.

Value

the nth power of the input gpu.matrix-class object. The returned matrix is also a gpu.matrix-class object.

See Also

See also: %*%.

Examples

```
## Not run:

a <- gpu.matrix(rnorm(9),nrow=3,ncol=3)
a %^% 5



## End(Not run)
```

qr_decomposition *The QR Decomposition of a GPUmatrix object*

Description

These functions mimic the base qr family functions to operate on gpu.matrix-class objects.

Usage

```
## S4 method for signature 'gpu.matrix.tensorflow'
qr(x,...)
## S4 method for signature 'gpu.matrix.torch'
qr(x,...)

## S4 method for signature 'list'
qr.Q(qr,complete,Dvec)
## S4 method for signature 'list'
qr.R(qr,complete)
## S4 method for signature 'list'
qr.X(qr,complete)

## S4 method for signature 'list'
qr.coef(qr,y)
## S4 method for signature 'list'
qr.qy(qr,y)
## S4 method for signature 'list'
qr.qty(qr,y)
## S4 method for signature 'list'
qr.resid(qr,y)
## S4 method for signature 'ANY,gpu.matrix.tensorflow'
qr.solve(a,b)
## S4 method for signature 'ANY,gpu.matrix.torch'
qr.solve(a,b)
## S4 method for signature 'gpu.matrix.tensorflow,ANY'
qr.solve(a,b)
## S4 method for signature 'gpu.matrix.tensorflow,gpu.matrix.tensorflow'
qr.solve(a,b)
## S4 method for signature 'gpu.matrix.torch,ANY'
qr.solve(a,b)
## S4 method for signature 'gpu.matrix.torch,gpu.matrix.torch'
qr.solve(a,b)
## S4 method for signature 'list,ANY'
qr.solve(a,b)
```

Arguments

x	a gpu.matrix.

y, b	a gpu.matrix corresponding to the right-hand side of equations ax=b or ax=y.
...	further arguments passed to or from other methods.
qr	a list resulting from the application of the function qr.
complete	The same as in 'base' function qr.Q, and qr.X
Dvec	The same as in 'base' function qr.Q
a	a gpu.matrix corresponding to the left-hand side of equations ax=b or ax=y.

Details

The function qr internally calls the corresponding function of the library torch or tensorflow (depending on the type of input gpu.matrix-class).

The QR decomposition can be used to solve the equation Ax=b for a given matrix A, and a vector of observations b. In this context, the functions qr.coef, and qr.resid return the coefficients, and residuals values. Moreover, the functions qr.qy, and qr.qty returns Q %*% y and Q %*% t(y). Note that if parameter complete is TRUE then an arbitrary orthogonal completion of the **X** and **Q** matrix or wheter the **R** matrix is to be completed by binding zero-value rows beneath the square upper triangle.

The function solve.qr solves the system of equations Ax=b via the QR decomposition. This function internally calls the corresponding function of the library torch or tensorflow (depending on the type of input gpu.matrix-class).

If the input gpu.matrix-class object(s) are stored on the GPU, then the operations will be performed on the GPU. See gpu.matrix.

Value

The function qr returns a list with the following items:

q	The corresponding complete matrix Q resulting from the application of the QR decomposition to a. It is a gpu.matrix-class object.
r	The corresponding complete matrix R resulting from the application of the QR decomposition to a. It is a gpu.matrix-class object.
x	The matrix a. It is a gpu.matrix-class object.

Please note that the output returned by this function is different from the 'base' function qr, which returns an object of the 'qr' class.

After performing a QR decomposition on a matrix A, given the resulting object, the functions qr.X, qr.Q, and qr.R return the original matrix A, the matrix Q, and the matrix R respectively. The returned matrices are gpu.matrix-class objects.

The functions qr.coef and qr.resid return the coefficients and residuals when fitting the equation Ax=b. In this context, the functions qr.qy, and qr.qty returns Q %*% y and Q %*% t(y). The resulting vectors are objects of the class gpu.matrix.

The function qr.solve returns a gpu.matrix-class object containing the coefficients of the solution of the system of equations Ax=b by QR decomposition.

See Also

See qr, linalg_qr, torch_triangular_solve

Examples

```
## Not run:
## overdetermined system
A <- gpu.matrix(runif(12),nrow =  4)
b <- gpu.matrix(rnorm(4),ncol=1)
qr.solve(a = A, b)
qr_gpu <- qr(A)
qr.solve(a=qr_gpu,b)
qr.coef(qr = qr_gpu,b)
qr.resid(qr = qr_gpu,b)
qr.qty(qr = qr_gpu,b)
qr.qy(qr = qr_gpu,b)
qr.X(qr = qr_gpu,complete = T)
qr.Q(qr = qr_gpu,complete = T)
qr.R(qr = qr_gpu,complete = T)


## underdetermined system
A <- gpu.matrix(runif(12),nrow =  3)
b <- gpu.matrix(rnorm(3),ncol=1)
qr.solve(a = A, b)
qr_gpu <- qr(A)
qr.solve(a=qr_gpu,b)
qr.coef(qr = qr_gpu,b)
qr.resid(qr = qr_gpu,b)
qr.qty(qr = qr_gpu,b)
qr.qy(qr = qr_gpu,b)
qr.X(qr = qr_gpu,complete = T)
qr.Q(qr = qr_gpu,complete = T)
qr.R(qr = qr_gpu,complete = T)

## End(Not run)
```

---

round *rounding of numers*

Description

It mimics the base function 'round' to operate on gpu.matrix-class objects. This function rounds the values in its first argument to the specified number of decimal places (default 0).

Usage

```
## S4 method for signature 'gpu.matrix.tensorflow,ANY'
round(x)
## S4 method for signature 'gpu.matrix.torch,missing'
round(x,digits)
## S4 method for signature 'gpu.matrix.torch,numeric'
```

```
round(x,digits)
## S4 method for signature 'gpu.matrix.tensorflow,missing'
round(x,digits)
## S4 method for signature 'gpu.matrix.tensorflow,numeric'
round(x,digits)
```

Arguments

x	a gpu.matrix.
digits	integer indicating the number of decimal places (round) or significant digits (signif) to be used.

Details

The function round internally calls the corresponding function of the library torch or tensorflow (depending on the type of input gpu.matrix-class).

The behaveour of the function mimics the 'base' function round. Note that for rounding off a 5, the function will consider "go to the even digit". Therefore, round(2.5) = 2 and round(3.5) = 4. For more details see round, and torch_round.

If the input gpu.matrix-class object is stored on the GPU, then the operations will be performed on the GPU. See gpu.matrix.

Value

The function will return a gpu.matrix-class object with the rounded values.

See Also

round, and torch_round.

Examples

```
## Not run:

a <- gpu.matrix(rnorm(9),3,3)
round(a,digits = 3) #round to the third digit



## End(Not run)
```

solve_gpu.matrix *Solve a System of Equations*

Description

The function solve mimics of the 'base' function solve to operate on gpu.matrix-class objects: it "solves the equation a %*% x = b."

The function ginv mimics the function ginv of package 'MASS' to operate on gpu.matrix-class objects: it "Calculates the Moore-Penrose generalized inverse of a matrix X."

The function chol_solve is a GPUmatrix own function. This function uses the Cholesky decomposition to solve a system of equations.

Usage

```
## S4 method for signature 'ANY,gpu.matrix.tensorflow'
solve(a,b)
## S4 method for signature 'ANY,gpu.matrix.torch'
solve(a,b)
## S4 method for signature 'gpu.matrix.tensorflow,ANY'
solve(a,b)
## S4 method for signature 'gpu.matrix.tensorflow,missing'
solve(a)
## S4 method for signature 'gpu.matrix.torch,ANY'
solve(a,b)
## S4 method for signature 'gpu.matrix.torch,missing'
solve(a)

## S4 method for signature 'gpu.matrix.torch'
ginv(X,tol)
## S4 method for signature 'gpu.matrix.tensorflow'
ginv(X,tol)

## S4 method for signature 'ANY,gpu.matrix.torch'
chol_solve(x,y)
## S4 method for signature 'ANY,gpu.matrix.tensorflow'
chol_solve(x,y)
## S4 method for signature 'gpu.matrix.torch,ANY'
chol_solve(x,y)
## S4 method for signature 'gpu.matrix.tensorflow,ANY'
chol_solve(x,y)
```

Arguments

These inputs correspond to the solve function:

a	a square numeric or complex gpu.matrix containing the coefficients of the linear system.
b	a numeric or complex vector or matrix giving the right-hand side(s) of the linear system. If b missing, solve will return the inverse of a.

These inputs correspond to the chol_solve function:

x	Given the equation Ax=b, x must be the transponsed of the cholesky decomposition of matrix A if A is a real symmetric positive-definite square matrix.
y	a numeric or complex vector or matrix giving the right-hand side(s) of the linear system.

These inputs correspond to the ginv function:

X	Matrix for which the Moore-Penrose inverse is required.
tol	A relative tolerance to detect zero singular values.

Details

The functions solve, and ginv internally calls the corresponding function of the library torch or tensorflow (depending on the type of input gpu.matrix-class).

If the input gpu.matrix-class object(s) are stored on the GPU, then the operations will be performed on the GPU. See gpu.matrix.

Value

The result of these functions is an object of the class gpu.matrix.

See Also

See also solve, ginv, torch_inverse, and torch_pinverse.

For cholesky decomposition see chol from base or matrix_decomposition from GPUmatrix.

Also see qr.solve.

Examples

```
## Not run:

#solve a system of equations:
a <- gpu.matrix(rnorm(9),nrow=3,ncol=3)
b <- c(1,1,1)
betas <- solve(a,b)
a %*% betas

#the inverse matrix
inv <- solve(a)
a %*% inv

#inverse using ginv
```

```
inv_2 <- ginv(a)
a %*% inv_2


#chol_solve: it can be applies only if
# in the equation Ax=b A is real symmetric positive-definite square matrix.
a <- gpu.matrix(rnorm(9),3,3)
A <- tcrossprod(a) #A is symmetrix positive-definite
b <- gpu.matrix(rnorm(3))

x_solve <- solve(A,b) #using solve to compare results
x_chol_solve <- chol_solve(t(chol(A)),b) #using chol_solve
#NOTE: notice that the input for chol_solve is the Cholesky decomposition
# of matrix A.


## End(Not run)
```

`sort` *sort*

Description

This function mimics the 'base' function `sort` to operate on gpu.matrix-class objects. This function sort the input matrix into ascending or descending order.

Usage

```
## S4 method for signature 'gpu.matrix.tensorflow,logical'
sort(x,decreasing)
## S4 method for signature 'gpu.matrix.torch,logical'
sort(x,decreasing)
```

Arguments

`x`	a `gpu.matrix`.
`decreasing`	Logical. Should the sort be increasing or decreasing?

Details

The function internally calls the corresponding function of the library torch or tensorflow (depending on the type of input gpu.matrix-class).

If the input gpu.matrix-class object(s) are stored on the GPU, then the operations will be performed on the GPU. See `gpu.matrix`.

Value

Returns a gpu.matrix-class object that is a vector (or a matrix with one column) with the values sorted.

See Also

sort, torch_sort.

Examples

```
## Not run:
a <- gpu.matrix(rnorm(9),nrow=3,ncol=3)
sort(a) #returns a vector with the data sorted.

## End(Not run)
```

---

type of gpu.matrix *Spicify type of 'GPUmatrix'*

Description

dtype and dtype<- are functions that show or set the number of bits to use to store the number. The possible options are "float64" for float64 (default), "float32" for float32 and "int" for int64. float64 uses 64 bits, that means that float64's take up twice as much memory thatn float32, thus doing operations on them may be slower in some machine architectures. However, float64's can represent numbers much more accurately than 32 bit floats. They also allow much larger numbers to be stored.

to_dense is a function that transforms a sparse matrix to a dense matrix. On the other hand, to_sparse transforms a dense matrix to a sparse matrix.

Usage

```
## S4 method for signature 'gpu.matrix.torch'
to_dense(x)
## S4 method for signature 'gpu.matrix.tensorflow'
to_dense(x)
## S4 method for signature 'gpu.matrix.torch'
to_sparse(x)
## S4 method for signature 'gpu.matrix.tensorflow'
to_sparse(x)


## S4 method for signature 'gpu.matrix.torch'
dtype(x)
## S4 method for signature 'gpu.matrix.tensorflow'
dtype(x)
```

```
## S4 replacement method for signature 'gpu.matrix.torch'
dtype(x) <- value
## S4 replacement method for signature 'gpu.matrix.tensorflow'
dtype(x) <- value
```

Arguments

x	a gpu.matrix.
value	type of gpu.matrix object

Value

dtype and dtype <- show or set the number of bits to use to store the number.

to_dense returns a dense gpu.matrix-class object while the function to_sparse returns a sparse gpu.matrix-class object.

See Also

See also gpu.matrix.

Examples

```
## Not run:

a <- gpu.matrix(rnorm(9),3,3)

dtype(a) #bits used to store the numbers: it is float64 by default.

b <- a
dtype(b) <- "float32" #change to float32
b

b <- a
dtype(b) <- "int" #change to integer64 (int64)
b

#sparse or dense matrices
A <- gpu.matrix(data=c(1,1,1,0,0,1,0,1,0),3,3)
A #A is a dense gpu.matrix

A_sparse <- to_sparse(A) #transform A to a sparse matrix.
A_sparse #this matrix stores the where number different to 0 were placed.

to_dense(A_sparse) #transform A_sparse to a dense matrix and we obtain the orginal matrix A:
A


## End(Not run)
```

Index