Package: RandomFieldsUtils (via r-universe)

October 10, 2024

Version 1.2.5

Title Utilities for the Simulation and Analysis of Random Fields and Genetic Data

Author Martin Schlather [aut, cre], Alexander FreudenBerg [aut], Reinhard Furrer [ctb], Martin Kroll [ctb], Brian D. Ripley [ctb], John W. Ratcliff et al. (cph)

Maintainer Martin Schlather <schlather@math.uni-mannheim.de>

Depends R (>= 3.0)

Imports utils, methods, parallel

Suggests spam, RandomFields

Description Various utilities are provided that might be used in spatial statistics and elsewhere. It delivers a method for solving linear equations that checks the sparsity of the matrix before any algorithm is used.

Copyright MIT licence on sse2neon.H

License GPL (>= 3)

NeedsCompilation yes

Date/Publication 2022-04-19 11:32:32 UTC

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

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

RemoteRef HEAD

RemoteSha b754335b4635fb1668e78ea3b473f0bc205939d9

Contents

`Cholesky` *Cholesky Decomposition of Positive Definite Matrices*

Description

This function calculates the Cholesky decomposition of a matrix.

Usage

```
cholx(a)
chol2mv(C, n)
tcholRHS(C, RHS)
```

Arguments

`a`	a square real-valued positive definite matrix
`C`	a (pivoted) Cholesky decomposition calculated by `cholx`
`n`	integer. Number of realisations of the multivariate normal distribution
`RHS`	vector

Details

If the matrix is diagonal direct calculations are performed.

Else the Cholesky decomposition is tried.

Value

`cholx` returns a matrix containing the Cholesky decomposition (in its upper part).

`chol2mv` takes the Cholesky decomposition and returns a `n` realisations of a multivariate normal distribution with mean 0 and covariance function `a`

`tcholRHS` multiplies the vector `RHS` from the right to *lower* triangular matrix of the Cholesky decomposition. See examples below.

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de>, https://www.wim.uni-mannheim.de/schlather/

References

Harbrecht, H., Peters, M., Schneider, R. (2012) On the low-rank approximation by the pivoted Cholesky decomposition. *Appl. Num. Math.* **62**, 428–440.

Examples

```
##########################
## Example showing the use of chol2mv and tcholRHS
n <- 10
M <- matrix(nc=n, runif(n^2))
M <- M %*% t(M) + diag(n)
C <- cholx(M)
set.seed(0)
v1 <- chol2mv(C, 1)
set.seed(0)
v2 <- tcholRHS(C, rnorm(n))
stopifnot(all(v1 == v2))


##########################
## The following example shows pivoted Cholesky can be used
## and the pivotation permutation can be transferred to
## subsequent Cholesky decompositions
set.seed(0)
n <- if (interactive()) 1000 else 100
x <- 1:n
y <- runif(n)
M <- x %*% t(x) + rev(x) %*% t(rev(x)) + y %*% t(y)

## do pivoting
RFoptions(pivot = PIVOT_DO, la_mode=LA_INTERN)
print(system.time(C <- cholx(M)))
print(range(crossprod(C) - M))
str(C)

## use the same pivoted decomposition as in the previous decomposition
M2 <- M +  n * diag(1:n)
RFoptions(pivot = PIVOT_IDX, la_mode=LA_INTERN,
          pivot_idx = attr(C, "pivot_idx"),
          pivot_actual_size = attr(C, "pivot_actual_size"))
print(system.time(C2 <- cholx(M2)))
print(range(crossprod(C2) - M2))
range((crossprod(C2) - M2) / M2)
str(C2)

RFoptions(pivot = PIVOT_AUTO, la_mode = LA_AUTO)
```

confirm *Test if Two Objects are (Nearly) Equal*

Description

`confirm(x, y)` is a utility to compare R objects `x` and `y` testing 'near equality' base on `all.equal`. It is written too allow different behaviour on different operating systems

Usage

```
confirm(x, y, ...)
```

Arguments

`x, y, ...`	see `all.equal`

Value

Only `TRUE` or error in linux-gnu. Otherwise logical.

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de>, https://www.wim.uni-mannheim.de/schlather/

Examples

```
x <- 3
confirm(gauss(x), exp(-x^2))
```

dbinorm *Density of a bivariate normal distribution*

Description

The function calculates the value of a bivariate normal distribution with mean 0.

Usage

```
dbinorm (x, S)
```

Arguments

x	a matrix containing the x values and the y values in the first and second row respectively. Or it is a list of two vectors.
S	the covariance matrix; currently only diagonal matrix possible

Value

a vector according to the size of x

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de>, https://www.wim.uni-mannheim.de/schlather/

Examples

```
x <- matrix(1:6, nc=2) + 0.0
C <- diag(c(1,2))
dbinorm(x, C)
```

FileExists *Files*

Description

The function FileExists checks whether a file or a lock-file exists

The function LockRemove removes a lock-file

Usage

```
FileExists(file, printlevel=RFoptions()$basic$printlevel)
LockFile(file, printlevel=RFoptions()$basic$printlevel)
LockRemove(file)
WaitOthers(file, i, cores, ideal.processes=ceiling(cores * 1.25),
                   max.processes=ceiling(cores * 1.5),
                   distance=5, time=5, path="./")
```

Arguments

file	name of the data file
printlevel	if PrintLevel<=1 no messages are displayed
i	integer; current value of process, usually the number of a loop index
cores	the number of cores on the machine
ideal.processes, max.processes, distance	integer. See Details
time	in minutes a process waits until it rechecks its environment
path	the current path of file

Details

FileExists checks whether file or file.lock exists. If none of them exists file.lock is created and hostname and PID are written into file.lock. This is useful if several processes use the same directory. Further, it is checked whether another process has tried to create the same file in the same instance. In this case FileExists returns for at least one of the processes that file.lock has already been created.

LockFile is the same as FileExists except that it does not check whether file already exists.

WaitOthers waits for others if more than ideal.processes processes have their value is less than i or if more than cores processes have their value is less than i-distance. It also waits if there are alreay max.processes are active. Note that WaitOthers write a file with ending '.wait', which is also deleted be LockRemove.

Value

FileExists returns

1	if file already exists
2	if file.lock already exists
3	if file.lock was tried to be created, but another process inferred and got priority
0	otherwise, file and file.lock did not exist and file.lock has been created

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de>, https://www.wim.uni-mannheim.de/schlather/

Examples

```
## Not run:
## the next command checks whether the file 'data.rda'
## or the file 'data.rda.lock' exists. If so, a positive
## value is returned. If not, the file 'data.rda.lock'
## is created and the value 0 returned.
FileExists("data.rda")

## the next command deletes the file 'data.rda.lock'
LockRemove("data.rda")

## End(Not run)
```

gauss *Gaussian Covariance Model*

Description

gauss is a stationary isotropic covariance model. The corresponding covariance function only depends on the distance $r \geq 0$ between two points and is given by

$$C(r) = e^{-r^2}$$

Usage

```
gauss(x, derivative=0)
```

Arguments

x	numerical vector; for negative values the modulus is used
derivative	value in 0:4.

Value

If derivative=0, the function value is returned, otherwise the derivativeth derivative.

A vector of length(x) is returned; nu is recycled; scaling is recycled if numerical.

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de>, https://www.wim.uni-mannheim.de/schlather/

References

Gelfand, A. E., Diggle, P., Fuentes, M. and Guttorp, P. (eds.) (2010) *Handbook of Spatial Statistics.* Boca Raton: Chapman & Hall/CRL.

Stein, M. L. (1999) *Interpolation of Spatial Data.* New York: Springer-Verlag

Examples

```
x <- 3
confirm(gauss(x), exp(-x^2))
```

host *System calls*

Description

The functions `hostname` and `pid` return the host name and the PID, respectively.

Usage

```
hostname()

pid()
```

Details

If R runs on a unix platform the host name and the PID are returned, otherwise the empty string and naught, respectively.

Value

`hostname`	returns a string
`pid`	returns an unsigned integer

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de>, https://www.wim.uni-mannheim.de/schlather/

Examples

```
cat("The name of your computer is '", hostname(),
    "'. Your R program has current pid ", pid(), ".\n", sep="")
```

Instruction Set *CPU instruction set*

Description

The function checks whether a certain instruction is used (missed) under the current compilation of a package.

Usage

```
uses.simd.instruction(which=NULL, pkgs=NULL)
misses.simd.instruction(which=NULL, pkgs=NULL)
```

Arguments

`which`	character vector with values in `"SSE2"`, `"SSSE3"`, `"AVX"`, `"AVX2"`, `"CUDA"`
`pkgs`	character vector or missing.

Value

logical vector of length `which` or matrix with number of rows equal to the length of `which`. An element is `TRUE` if the instruction set is used (missed) by the package.

If an arguments is `NULL` all available information is given.

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de>, https://www.wim.uni-mannheim.de/schlather/

Examples

```
uses.simd.instruction()
misses.simd.instruction()
```

`Internal functions` *Internal functions*

Description

These functions are internal and should not be used.

Usage

```
checkExamples(exclude = NULL, include=1:length(.fct.list),
               ask=FALSE, echo=TRUE, halt=FALSE, ignore.all = FALSE,
               path=package, package = "RandomFields",
               read.rd.files=TRUE, local = FALSE, libpath = NULL,
               single.runs = FALSE, reset, catcherror=TRUE)

Dependencies(pkgs = all.pkgs, dir = "Dependencies",
                         install = FALSE, check=TRUE, reverse=FALSE,
      package="RandomFields")

debugging_level()
```

Arguments

`exclude`, `include`, `ask`, `echo`, `halt`, `ignore.all`, `path`, `package`, `read.rd.files`, `local`, `libpath`, `single.runs`, `reset`, `catcherror`
: internal; ignore.all refers to the 'all' export statement in the namespace – whether this should be ignored. If `read.rf.files` is `TRUE` or a path to the Rd files, then the man pages are analysed to get all examples; `ignore.all` is then ignored. If `FALSE` only examples of functions (which are searched in the environments) are run.

`pkgs`, `dir`, `install`, `check`, `reverse`
: internal

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de>, https://www.wim.uni-mannheim.de/schlather/

Examples

```
## internal function: no examples given
```

`matern` *Whittle-Matern Model*

Description

`matern` calculates the Whittle-Matern covariance function (Soboloev kernel).

The Whittle model is given by

$$C(r) = W_\nu(r) = 2^{1-\nu}\Gamma(\nu)^{-1} r^\nu K_\nu(r)$$

where $\nu > 0$ and K_ν is the modified Bessel function of second kind.

The Matern model is given by

$$C(r) = \frac{2^{1-\nu}}{\Gamma(\nu)} (\sqrt{2\nu} r)^\nu K_\nu(\sqrt{2\nu} r)$$

The Handcock-Wallis parametrisation equals

$$C(r) = \frac{2^{1-\nu}}{\Gamma(\nu)} (2\sqrt{\nu} r)^\nu K_\nu(2\sqrt{\nu} r)$$

Usage

```
whittle(x, nu, derivative=0,
       scaling=c("whittle", "matern", "handcockwallis"))
matern(x, nu, derivative=0,
       scaling=c("matern", "whittle", "handcockwallis"))
```

Arguments

x	numerical vector; for negative values the modulus is used
nu	numerical vector with positive entries
derivative	value in 0:4.
scaling	numerical vector of positive values or character; see Details.

Value

If derivative=0, the function value is returned, otherwise the derivativeth derivative.

A vector of length(x) is returned; nu is recycled; scaling is recycled if numerical.

If scaling has a numerical values s, the covariance model equals

$$C(r) = \frac{2^{1-\nu}}{\Gamma(\nu)}(s\sqrt{\nu}r)^{\nu}K_{\nu}(s\sqrt{\nu}r)$$

The function values are rather precise even for large values of nu.

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de>, https://www.wim.uni-mannheim.de/schlather/

References

Covariance function

- Chiles, J.-P. and Delfiner, P. (1999) *Geostatistics. Modeling Spatial Uncertainty.* New York: Wiley.
- Gelfand, A. E., Diggle, P., Fuentes, M. and Guttorp, P. (eds.) (2010) *Handbook of Spatial Statistics.* Boca Raton: Chapman & Hall/CRL.
- Guttorp, P. and Gneiting, T. (2006) Studies in the history of probability and statistics. XLIX. On the Matern correlation family. *Biometrika* **93**, 989–995.
- Handcock, M. S. and Wallis, J. R. (1994) An approach to statistical spatio-temporal modeling of meteorological fields. *JASA* **89**, 368–378.
- Stein, M. L. (1999) *Interpolation of Spatial Data – Some Theory for Kriging.* New York: Springer.

See Also

nonstwm

Examples

```
x <- 3
confirm(matern(x, 0.5), exp(-x))
confirm(matern(x, Inf), gauss(x/sqrt(2)))
confirm(matern(1:2, c(0.5, Inf)), exp(-(1:2)))
```

---

nonstwm *nonstwm*

Description

The non-stationary Whittle-Matern model C is given by

$$C(x,y) = \Gamma(\mu)\Gamma(\nu(x))^{-1/2}\Gamma(\nu(y))^{-1/2}W_\mu(f(\mu)|x-y|)$$

where $\mu = [\nu(x)+\nu(y)]/2$, and ν must a positive function.

W_μ is the covariance function `whittle`.

The function f takes the following values

`scaling = "whittle"` : $f(\mu) = 1$

`scaling = "matern"` : $f(\mu) = \sqrt{2\nu}$

`scaling = "handcockwallis"` : $f(\mu) = 2\sqrt{\nu}$

`scaling` **= s, numerical :** $f(\mu) = s * \sqrt{nu}$

Usage

```
nonstwm(x, y, nu, log=FALSE,
        scaling=c("whittle", "matern", "handcockwallis"))
```

Arguments

x, y	numerical vectors of the same length
nu	positive value or a function with positive values and x as argument
log	logical. If TRUE the logirithm of the covariance function is returned.
scaling	positive value or character; see Details.

Value

A single value is returned.

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de>, https://www.wim.uni-mannheim.de/schlather/

References

- Stein, M. (2005) Nonstationary Spatial Covariance Functions. Tech. Rep., 2005

See Also

matern.

Examples

```
nonstwm(2, 1, sin)
```

---

orderx *Ordering Permutation*

Description

orderx has the same functionality as order, except that orderx(..., from=from, to=to) is the same as order[from:to]

Usage

```
orderx(x, from=1, to=length(x), decreasing=FALSE, na.last = NA)
```

Arguments

x	an atomic vector
from, to	order(..., from=from, to=to) equals order(...)[from:to]
decreasing	logical. Should the sort order be increasing or decreasing?
na.last	for controlling the treatment of NAs. If TRUE, missing values in the data are put last; if FALSE, they are put first; if NA, they are removed (see the Notes in order)

Details

The smaller the difference to-from is compared to the length of x, the faster is orderx compared to order.

Particularly, orderx(..., from=k, to=k) is much faster than order(...)[k].

orderx is never really slower than order.

For further details see order.

Value

integer vector of length to-from+1.

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de>, https://www.wim.uni-mannheim.de/schlather/

See Also

sortx

Examples

```
x <- runif(10^6)
k <- 10
system.time(y<-order(x)[1:k])
system.time(z<-orderx(x, from=1, to=k)) ## much faster
stopifnot(all(x[y ]== x[z])) ## same result
```

`Print` *Print method returning also the names automatically*

Description

prints variable names and the values

Usage

```
Print(..., digits = 6, empty.lines = 2)
```

Arguments

`...`	any object that can be `print`-ed
`digits`	see `print`
`empty.lines`	number of leading empty lines

Value

prints the names and the values; for vectors `cat` is used and for lists `str`

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de>, https://www.wim.uni-mannheim.de/schlather/

Examples

```
a <- 4
b <- list(c=5, g=7)
m <- matrix(1:4, nc=2)
Print(a, b, m)
```

RFoptions *Setting control arguments*

Description

RFoptions sets and returns control arguments for the analysis and the simulation of random fields

Usage

```
RFoptions(..., no.class=FALSE, install.control=NULL)
```

Arguments

...	arguments in tag = value form, or a list of tagged values. See 'Details' for options in package **RandomFieldsUtils**.
no.class	logical. If TRUE the list is returned without class specification.
install.control	list. See Details, Part 2

.

Details

The subsections below comment on
1. basic**: Basic options**
2. install.control
3. installNrun**: Options for installation and running**
4. solve**: Options for solving linear systems**
5. Reserved words

1. Basic options

asList logical. Lists of arguments are treated slightly different from non-lists. If asList=FALSE they are treated the same way as non-lists. This options being set to FALSE after calling RFoptions it should be set as first element of a list.

Default: TRUE

cores Number of cores for multicore algorithms; currently only used for the Cholesky decomposition.

Default : 1 if the package has been compiled with standard flags of CRAN and 0.75 * cores() + 0.25 * cpus() else.

Note that cores has not effect if set locally in this package or in package miraculix.

cPrintlevel cPrintlevel is automatically set to printlevel when printlevel is changed. Standard users will never use a value higher than 3.

0 : no messages
1 : messages and warnings when the user's input looks odd

2 : messages (and internal errors) documenting the choice of the simulation method
3 : further user relevant informations
4 : information on recursive function calls
5 : function flow information of central functions
6 : errors that are internally treated
7 : details on building up the covariance structure
8 : details on taking the square root of the covariance matrix
9 : details on intermediate calculations
10 : further details on intermediate calculations

Note that `printlevel` works on the R level whereas `cPrintlevel` works on the C level. `cPrintlevel` should be changed only globally.

Default: 1

efficient logical. If `TRUE` then always the most time efficient code is used.

Default: `TRUE`. It is strongly recommended to retain this value.

`helpinfo` logical. If `TRUE` then additional information is printed for more efficient programming in R.

Default: `TRUE`

`printlevel` If `printlevel`≤ 0 there is not any output on the screen. The higher the number the more tracing information is given. Standard users will never use a value higher than 3.

0 : no messages
1 : important (error) messages and warnings
2 : less important messages
3 : details, but still for the user
4 : recursive call tracing
5 : function flow information of large functions
6 : errors that are internally treated
7 : details on intermediate calculations
8 : further details on intermediate calculations

Default: 1

`seed` integer (currently only used by the package RandomFields). If `NULL` or `NA` `set.seed` is **not** called. Otherwise, `set.seed(seed)` is set before any simulations are performed.

If the argument is set locally, i.e., within a function, it has the usual local effect. If it is set globally, i.e. by `RFoptions` the `seed` is fixed for **all subsequent** calls.

If the number of simulations `n` is greater than one and if `RFoptions(seed=seed)` is set, the ith simulation is started with the seed '`seed`$+i-1$'.

`skipchecks` logical. If `TRUE`, several checks whether the given parameter values and the dimension are within the allowed range is skipped. Do not change the value of this variable except you really know what you do.

Default: `FALSE`

`verbose` logical. If `FALSE` it identical to `printlevel = 1` else to `printlevel = 2`.

`bigendian` logical. Read only.

2. `install.control`**: Details on argument** `install.control` `install.control` may contain any argument of `install.packages` except `type`. **This options is currently tailored for MS and Linux on Intel machines, only.** The argument `configure.args` may not contain `'CXX_FLAGS'` which should be passed as an extra argument with the list.

Note that if this argument is given (even with value `NULL`), an immediate installation takes place. In case the user tries to force to install 0 packages, an overview over the packages is given. If the user is asked whether re-installation shall take place, user can pass arguments to install.packages, e.g., `"quiet=FALSE"`.

If `install.control` is given, no further argument may be passed to `RFoptions`.

Additional components of `install.control` and special behaviours are:

`path` the path to the locally saved tar balls

`verbose`**,** `quiet` They affect also the behaviour of `RFoptions`.

`force` `TRUE` reinstallation of all attached libraries based on and including **RandomFieldsUtils**. I.e., `RFoptions(install.control=list(force=TRUE))` is the strongest form of forcing reinstallation.

`FALSE` In case some packages have to be re-installed the user will be asked.

not given reinstallation of the attached libraries based on and including **RandomFieldsUtils** that have not been tried yet in the current session.

`pkgs=NULL` brief overview over the installed packages based on `RandomFieldsUtils`

`CROSS` logical or character. `CROSS` is passed to 'configure.ac'.

`"noflag"` No extra compiler flag is set with respect to SIMD. This is the default.

`TRUE` each file is compiled with its specific SIMD/AVX compiler flags; this guarantees the compatiblity on a plattform with different sets of kernels. No SIND/AVX flag should be given by the user. Cross-compilation supported; no check is performed whether the code would run on the compiling CPU.

`"nosimd"` It is assumed that no SIMD is available and the flag "-no-sse2" is set (if possible).

`"sse2"` Same behaviour as `TRUE`, but all CPUs have at least `"sse2"` available.

`"sse3"`**,** `"ssse3"`**,** `"sse41"`**,** `"avx"`**,** `"avx2"` Alternatives to `"sse2"`. Giving the highest guaranteed SIMD recognition leads to the most efficient code.

`FALSE` each file is compiled with all SIMD/AVX flags recognized by both the CPU and the compiler (no cross-compilation); users may add their own SIMD/AVX flags. This might lead to faster code, but which is not downwards compatible.

`NA` Same as `FALSE` except that the flag `-mno-sse2` is set when no SIMD is needed or used.

This option can be set to `"avx"` interactively if `install="ask"`.

`CC_FLAGS` character. Flags passed to 'configure.ac'.

`SIMD_FLAGS` character. A subset of `"sse2"`, `"sse3"`, `"ssse3"`, `"sse41"`, `"avx"`, `"avx2"`, `"arch=xxx"`, etc. which will be tried instead of default flags. `SIMD_FLAGS` is passed to 'configure.ac'.

`LOCAL_ONLY` logical. If `TRUE`, the web is not searched for the latest version of the package.

`MEM_IS_ALIGNED` logical. If `TRUE`, then the memory is assumed to be aligned. If `FALSE` then the SIMD load commands `_mm_*load_*` are replaced by `_mm_*loadu_*`. If given, then `force` is set to `TRUE`.

`USE_GPU` logical. Force or hinder the compilation for the GPU

3. `installNrun`**: Options for installing and for determining basic behaviour**

`install` character. Only used by linux systems and alike including macOS The default by CRAN is that SIMD/AVX cannot be used at full extend. `install` determines what the action if the compiled version does not use the full CPU capacities. Since the use of GPU is heavily hardware dependent, its auto-recompilation is only performed in tow line of an AVX recompilation. The users usually use

`"no"` no re-installation

`"ask"` asks whether the library should be reinstalled, using the full capacity of the CPU according to the package.

`"install"` performs the auto-recompilation without asking. Note that only the command `RFoptions(install.control=list(force=TRUE))` forces re-compilation of the currently loaded packages that are based on **RandomFieldsUtils**.

Note that, in each session, a package can be reinstalled only. This feature avoids trying to run jobs that cannot be done (e.g.\ due to missing programs of the OS). See argument `install.control` for overwriting this behaviour.

Default: at starting point it is `"ask"` or `"no"`, but the value may change during the session.

`installPackages` logical. Read only. Indicates whether packages are left to be re-installed. `RFoptions(install="no")` sets it to `FALSE`. `RFoptions(install="no", install="ask")` sets it to `TRUE`.

`kahanCorrection` obsolete. logical. If `TRUE`, the Kahan summation algorithm is used for calculating scalar products.

Default: false

`la_mode` determines

LA_AUTO, `"auto"` If a graphic card is recognized, `LA_GPU` is used. In all other cases the default is primarily `LA_R`. Only on linux systems, the package peforms a simple speed test and takes `LA_INTERN` if it is faster than LA_R; the time, hence the choice, depends also on the number of cores used.

LA_INTERN, `"intern"` mostly own algorithms, often based on SIMD/AVX. This option is of interest only if no advanced BLAS/LAPACK has been compiled into R

LA_R, `"R"` BLAS/LAPACK implementation used by R

LA_GPU, `"GPU"` This option is available when the package has been compiled with nvcc.

LA_QUERY, `"query"` Request on currently used set-up

Default: `LA_AUTO`

`mem_is_alignedlogical`. Read only. See `MEM_IS_ALIGNED` in `install.control`.

`warn_parallel` Logical. **RandomFieldsUtils** and packages using it, such as **RandomFields** and **miraculix**, should now be prepared for parallelization using package `parallel`, for instance. Internal OMP parallelization of **RandomFieldsUtils** is done, but only at a view points of the subsequent packages.

As a few parts cannot be in parallel technically or from a logical point of view, a hint or a warning is given, if such a point is not accessed adequately. These messages can be turned off by `warn_parallel = FALSE`.

Default: `TRUE`.

`warn_unknown_option` integer.

`0,1,-1` Unknown options are all ignored. If the value is positive, a hint is delivered whenever an unknown option is ignored.

`-2,2` Unknown options that start with a capital letter are ignored. All others lead to an error. (Note that all `RFoptions` start with a minor letter.) If the value is positive, a hint is delivered whenever an unknown option is ignored.

`3,-3` Unknown options that consists of a single capital letter are ignored. All others lead to an error. (Note that all `RFoptions` start with a minor letter.) If the value is positive, a hint is delivered whenever an unknown option is ignored.

`4` (and other values) Any unknown option leads to an error.

Default for **RandomFieldsUtils**: `3`

Default for **RandomFields**: `1`

4. `solve`: Options for solving linear systems

`det_as_log`

`eigen2zero` When the svd or eigen decomposition is calculated, all values with modulus less than or equal to `eigen2zero` are set to zero.

Default: `1e-12`

`max_chol` integer. Maximum number of rows of a matrix in a Cholesky decomposition

Default: 16384

`max_svd` integer. Maximum number of rows of a matrix in a svd decomposition

Default: 10000

`pivot_partialdet` logical. If `TRUE` then in case of low-rank matrices the determinant is calculated only in the part with positive eigenvalues

`pivot` Type of pivoting for the Cholesky decomposition. Possible values are

PIVOT_NONE, `"no"` No pivoting.

PIVOT_AUTO, `"auto"` If the matrix has a size greater than 3x3 and Choleskey fails without pivoting, privoting is done. For matrices of size less than 4x4, no pivoting and no checks are performed. See also `PIVOT_DO`

PIVOT_DO, `"do"` Do always pivoting. NOTE: privoted Cholesky decomposition yields only very approximately an upper triangular matrix L, but still L^t L = M holds true.

PIVOT_IDX, `"idx"` uses the same pivoting as in the previous pivoted decomposition. This option becomes relevant only when simulations with different parameters or different models shall be performed with the same seed so that also the pivoting must be coupled.

Default: `PIVOT_NONE`

`pivot_actual_size` integer. Genuine dimension of the linear mapping given by a matrix in cholx. This is a very rarely used option when pivoting with `pivot=PIVOT_IDX`.

`pivot_check` logical. Only used in pivoted Cholesky decomposition. If `TRUE` and a numerically zero diagonal element is detected, it is checked whether the offdiagonal elements are numerically zero as well. (See also `pivot_max_deviation` and `pivot_max_reldeviation`.) If `NA` then only a warning is given.

Default: `TRUE`

`pivot_idx` vector of integer. Sequence of pivoting indices in pivoted Cholesky decomposition. Note that `pivot_idx[1]` gives the number of indices that will be used. The vector must have at least the length `pivot_idx[1] + 1`.

Default: `NULL`

`pivot_relerror` positive number. Tolerance for (numerically) negative eigenvalues and for (numerically) overdetermined systems appearing in the pivoted Cholesky decomposition.

Default: `1e-11`

`pivot_max_deviation` positive number. Together with `pivot_max_reldeviation` it determines when the rest of the matrix (eigenvalues) in the pivoted Cholesky decomposition are considered as zero.

Default: `1e-10`

`pseudoinverse` logical. In case of a singular matrix M, shall the pseudo inverse be returned for `solvex(M)`?

Default: `FALSE`

`pivot_max_reldeviation` positive number. Together with `pivot_max_deviation` it determines when the rest of the matrix (eigenvalues) in the pivoted Cholesky decomposition are considered as zero.

Default: `1e-10`

`solve_method` vector of at most 3 integers that gives the sequence of methods in order to inverse a matrix or to calculate its square root: `"cholesky"`, `"svd"`, `"lu"`, `"eigen"` `"sparse"`, `"method undefined"`. In the latter case, the algorithm decides which method might suit best.

Note that if `use_spam` is not `false` the algorithm checks whether a sparse matrix algorithm should be used and which is then tried first.

Default: `"method undefined"`.

`spam_factor` integer. See argument `spam_sample_n`.

Default: 4294967

`spam_min_n` integer vector of size 2. The minimal size for a matrix to apply a sparse matrix algorithms automatically. The second value is used in case the GPU is activated.

Default: `c(400, 4000)`

`spam_min_p` (`spam_min_p`) a numbers in $(0, 1)$ giving the proportion of zero above which an sparse matrix algorithm is used. The second value is used in case the GPU is activated.

Default: `0.8` (`0.9`)

`spam_pivot` integer. Pivoting algorithm for sparse matrices:

PIVOT_NONE No pivoting

PIVOTSPARSE_MMD

PIVOTSPARSE_RCM

See package `spam` for details.

Default: PIVOTSPARSE_MMD

`spam_sample_n` (`spam_sample_n_GPU`) Whether a matrix is sparse or not is tested by a 'random' sample of size `spam_sample_n`; The selection of the sample is iteratively obtained by multiplying the index by `spam_factor` modulo the size of the matrix.

Default: 500 (10000).

`spam_tol` largest absolute value being considered as zero. Default: `DBL_EPSILON`

`svdtol` Internal. When the svd decomposition is used for calculating the square root of a matrix then the absolute componentwise difference between this matrix and the square of the square root must be less than `svdtol`. No check is performed if `svdtol` is not positive.

Default: `0`

`use_spam` Should the package `spam` (sparse matrices) be used for matrix calculations? If `TRUE` **spam** is always used. If `FALSE`, it is never used. If `NA` its use is determined by the size and the sparsity of the matrix.

Default: `NA`.

5. Reserved Words

`list_` `list_` usually equals the output of `RFoptions()`. This argument is used to reset the RFoptions. Some of the options behave differently if passed through `list_`. E.g. a warning counter is not reset. The argument `list_` cannot be combined with any other arguments.

`getoptions_` string vector of prefixes that indicate classes of options. In this package they can be `"basic"` and `"solve"`. (E.g. package **RandomFields** has many more classes of options.) The given classes of options are then returned by `RFoptions()`. Note that the values are the previous values.

`getoptions_` must always be the very first argument.

`saveoptions_` string vector of prefixes. Same as for `getoptions_`, except that important classes are always returned and thus should not be given. Hence `saveoptions_` is often a convenient short cut for `getoptions_`. The class always included in this package is `"basic"`, in package **RandomFields** these are the two classes `"basic"` and `"general"`.

`saveoptions_` must always be the very first argument. In particular, it may not given at the same time with `getoptions_`.

`local_` logical. This options is allowed only when advanced packages are used, see **RandomFields**.

`warnUnknown_` integer. Same as option `warn_unknown_option`, except that its value overwrites the value of `warn_unknown_option` in the current command `RFoptions`. This options must be placed between `CODE` and `getoptions_`, if the latter are used.

Value

`NULL` if any argument is given, and the full list of arguments, otherwise.

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de>, https://www.wim.uni-mannheim.de/schlather/

Examples

```
n <- 10
M <- matrix(1, ncol=n, nrow=n)

## Not run:
try(chol(M)) ## error, as M is not strictly positive definite
```

```
try(cholx(M)) ## also fails

## End(Not run)


RFoptions(la_mode=LA_INTERN, pivot=PIVOT_AUTO)
cholx(M) ## works
RFoptions(la_mode=LA_R)


RFoptions(solve_method="svd", pseudoinverse=TRUE)
solvex(M)
RFoptions(solve_method="method undefined", pseudoinverse=FALSE)
```

`rowMeansx` *Some Further Row and Column Functions*

Description

The function `rowMeansx` returns weighted row means;
the function `colMax` returns column maxima;
the function `rowProd` returns the product of each row;
the function `quadratic` calculates a quadratic form
the function `SelfDivByRow` devides each column by a scalar;
the function `dotXV` calculates columnwise the dot product;
the function `crossprodx` calculates the cross product (using AVX);
the function `scalarx` calculates the scalar product (using AVX);

Usage

```
rowMeansx(x, weight=NULL)
colMax(x)
rowProd(x)
SelfDivByRow(x, v)
quadratic(x, v)
dotXV(x, w)
crossprodx(x,y,mode=-1)
scalarx(x, y, mode=0)
```

Arguments

`x`	numerical (or logical) matrix
`v`	vector whose length equals the number of columns of `x`
`w`	vector whose length equals the number of rows of `x`
`weight`	numerical or logical vector of length `nrow(x)`

y	numerical matrix
mode	integer between 0 and 8 or negative, indicating that the default value should be used. Determine the algorithm how the scalar product is calculated. These values are experimental and may change their meaning.

Details

`quadratic(x, v)` calculates the quadratic form $v^\top xv$; The matrix `x` must be squared.

Value

`rowMeansx` returns a vector of length`nrow(x)`.

`colMax` returns a vector of length `ncol(x)`.

`rowProd` returns a vector of length `nrow(x)`.

`quadratic` returns a scalar.

`SelfDivByRow` returns a matrix of same size as `x`.

`dotXV` returns a matrix of same size as `x`.

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de>, https://www.wim.uni-mannheim.de/schlather/

Examples

```
c <- if (interactive()) 10000 else 10
r <- if (interactive()) 20000 else 20
M <- matrix(nr=r, 1:(c * r))

## unweighted means, compare to rowMeans
print(system.time(m1 <- rowMeans(M)))
print(system.time(m2 <- rowMeansx(M)))
stopifnot(all.equal(m1, m2))

## weighted row means, compare to rowMeans
W <- 1 / (ncol(M) : 1)
print(system.time({M0 <- t(W * t(M)); m1 <- rowMeans(M0)}))
print(system.time(m2 <- rowMeansx(M, W)))
stopifnot(all.equal(m1, m2))

print(system.time(m1 <- apply(M, 2, max)))
print(system.time(m2 <- colMax(M)))
stopifnot(m1 == m2)
```

sleep.milli *Sleep*

Description

Process sleeps for a given amount of time

Usage

```
sleep.milli(n)
sleep.micro(n)
```

Arguments

n	integer. sleeping time units

Value

No value is returned.

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de>, https://www.wim.uni-mannheim.de/schlather/

Examples

```
## next command waits half a second before returning
sleep.milli(500)
```

solve *Solve a System of Equations for Positive Definite Matrices*

Description

This function solves the equality $ax = b$ for x where a is a **positive definite** matrix and b is a vector or a matrix. It is slightly faster than the inversion by the cholesky decomposition and clearly faster than solve. It also returns the logarithm of the determinant at no additional computational costs.

Usage

```
solvex(a, b=NULL, logdeterminant=FALSE)
```

Arguments

a	a square real-valued matrix containing the coefficients of the linear system. Logical matrices are coerced to numeric.
b	a numeric or complex vector or matrix giving the right-hand side(s) of the linear system. If missing, b is taken to be an identity matrix and solvex will return the inverse of a.
logdeterminant	logical. whether the logarithm of the determinant should also be returned

Details

If the matrix is diagonal direct calculations are performed.

Else if the matrix is sparse the package **spam** is used.

Else the Cholesky decomposition is tried. Note that with RFoptions(pivot=) pivoting can be enabled. Pivoting is about 30% slower.

If it fails, the eigen value decomposition is tried.

Value

If logdeterminant=FALSE the function returns a vector or a matrix, depending on b which is the solution to the linear equation. Else the function returns a list containing both the solution to the linear equation and the logarithm of the determinant of a.

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de>, https://www.wim.uni-mannheim.de/schlather/

References

See chol.spam of the package **spam**.

Examples

```
RFoptions(solve_method = "cholesky", printlevel=1)
set.seed(1)
n <- 1000
x <- 1:n
y <- runif(n)


## FIRST EXAMPLE: full rank matrix
M <- exp(-as.matrix(dist(x) / n))
b0 <- matrix(nr=n, runif(n * 5))
b <- M %*% b0 + runif(n)

## standard with 'solve'
print(system.time(z <- zR <- solve(M, b)))
print(range(b - M %*% z))
stopifnot(all(abs((b - M %*% z)) < 2e-11))
```

```
## using exactly the algorithm used in R
RFoptions(pivot=PIVOT_NONE, la_mode=LA_R) ## (default)
print(system.time(z <- solvex(M, b)))
print(range(b - M %*% z))
stopifnot(all(z == zR))

## Without pivoting, internal code:
RFoptions(pivot=PIVOT_NONE, la_mode=LA_INTERN) ## (default)
print(system.time(z <- solvex(M, b)))
print(range(b - M %*% z))
stopifnot(all(abs((b - M %*% z)) < 2e-11))

## Pivoting is slower here:
RFoptions(pivot=PIVOT_DO, la_mode=LA_INTERN)
print(system.time(z <- solvex(M, b)))
print(range(b - M %*% z))
stopifnot(all(abs((b - M %*% z)) < 2e-11))


## SECOND EXAMPLE: low rank matrix
M <- x %*% t(x) + rev(x) %*% t(rev(x)) + y %*% t(y)
b1 <- M %*% b0

## Without pivoting, it does not work
RFoptions(pivot=PIVOT_NONE, la_mode=LA_R)
## Not run: try(solve(M, b1))
RFoptions(pivot=PIVOT_NONE, la_mode=LA_INTERN)
## Not run: try(solvex(M, b1))

## Pivoting works -- the precision however is reduced :
RFoptions(pivot=PIVOT_DO, la_mode=LA_INTERN)
print(system.time(z1 <- solvex(M, b1)))
print(range(b1 - M %*% z1))
stopifnot(all(abs((b1 - M %*% z1)) < 2e-6))

## Pivoting fails, when the equation system is not solvable:
b2 <- M + runif(n)
## Not run: try(solvex(M, b2))


RFoptions(pivot = PIVOT_AUTO, la_mode = LA_AUTO)
```

`sortx` *Sorting Vectors*

Description

`sortx` has the same functionality as `sort`, except that `sortx(..., from=from, to=to)` is the same as `sort[from:to]`

Sort a vector or factor into ascending or descending order.

Usage

```
sortx(x, from=1, to=length(x), decreasing=FALSE, na.last = NA)
```

Arguments

x	an atomic vector
from, to	sort(..., from=from, to=to) equals sort(...)[from:to]
decreasing	logical. Should the sort sort be increasing or decreasing?
na.last	for controlling the treatment of NAs. If TRUE, missing values in the data are put last; if FALSE, they are put first; if NA, they are removed (see the Notes in sort)

Details

The smaller the difference to-from is compared to the length of x, the faster is sortx compared to sort.

Particularly, sortx(..., from=k, to=k) is much faster than sort(...)[k].

For further details see sort.

Value

vector of length to-from+1.

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de>

See Also

orderx

Examples

```
x <- runif(10^6)
k <- 10
system.time(y<-sort(x)[1:k])
system.time(z<-sortx(x, from=1, to=k)) ## much faster
stopifnot(all(y == z)) ## same result
```

Struve *Modified Struve functions and related functions*

Description

These functions return the values of the modified Struve functions and related functions

Usage

```
struveH(x, nu)
struveL(x, nu, expon.scaled=FALSE)
I0L0(x)
```

Arguments

x	non-negative numeric vector
nu	numeric vector
expon.scaled	logical; if TRUE, the results are exponentially scaled in order to avoid overflow or underflow respectively.

Details

I0L0 returns besselI(nu=0). minus struveL(nu=0).

Value

Numeric vector with the (scaled, if expon.scaled = TRUE) values of the corresponding function.

The length of the result is the maximum of the lengths of the arguments x and nu. The two arguments are recycled to that length.

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de>, https://www.wim.uni-mannheim.de/schlather/

References

- MacLeod, A.J. (1993) Chebyshev expansions for modified Struve and related functions, *Mathematics of Computation*, **60**, 735-747
- Abramowitz, M., and Stegun, I.A. (1984) *Pocketbook of Mathematical Functions*, Verlag Harry Deutsch

See Also

besselI

Examples

```
if (FALSE) {

x <- seq(1, 2, 0.1)
struveH(x, 0)
struveH(x, 1)

I0L0(x) - (besselI(x, nu=0) - struveL(x, 0))
besselI(x, nu=1) - struveL(x, 1) ## cf. Abramovitz & Stegun, table 12.1

}
```

Index