Package ‘LSDinterface’

May 13, 2022

Type Package

Title Interface Tools for LSD Simulation Results Files

Version 1.2.1

Date 2022-5-12

Description Interfaces R with LSD simulation models. Reads object-oriented data in results files (.res[.gz]) produced by LSD and creates appropriate multi-dimensional arrays in R. Supports multiple core parallel threads of multi-file data reading for increased performance. Also provides functions to extract basic information and statistics from data files. LSD (Laboratory for Simulation Development) is free software developed by Marco Valente and Marcelo C. Pereira (documentation and downloads available at <https://www.labsimdev.org/>).

Depends R (>= 3.2.0)

Imports stats, boot, utils, parallel, abind, TSdist

Suggests LSDsensitivity

License GPL-3

Language en-US

Encoding UTF-8

NeedsCompilation no

Author Marcelo C. Pereira [aut, cre] (<https://orcid.org/0000-0002-8069-2734>)

Maintainer Marcelo C. Pereira <mcper@unicamp.br>

Repository CRAN

Date/Publication 2022-05-13 20:30:02 UTC

R topics documented:

`LSDinterface-package` *Interface Tools for LSD Simulation Results Files*

Description

Interfaces R with LSD simulation models. Reads object-oriented data in results files (.res[.gz]) produced by LSD and creates appropriate multi-dimensional arrays in R. Supports multiple core parallel threads of multi-file data reading for increased performance. Also provides functions to extract basic information and statistics from data files. LSD (Laboratory for Simulation Development) is free software developed by Marco Valente and Marcelo C. Pereira (documentation and downloads available at <https://www.labsimdev.org/>).

Details

There are specific `read.xxx.lsd()` functions for different types of LSD data structures.

`read.raw.lsd()` simply import LSD saved data in tabular (data frame) format (variables in columns and time steps in rows). `read.single.lsd()` is appropriate to simple LSD data structures where each saved variable is single-instanced (inside an object with a single copy). `read.multi.lsd()` reads all instances of all variables from the LSD results file, renaming multi-instanced variables. `read.list.lsd()` is similar to `read.multi.lsd()` but saves multiple-instanced variables as R lists, preventing renaming.

`read.3d.lsd()` and `read.4d.lsd()` are specialized versions for extracting data from multiple LSD results files simultaneously. The files must have the same structure (selected variables and number of time steps). They are frequently used to acquire data from Monte Carlo experiments or sensitivity analysis. `read.3d.lsd()` operates like `read.single.lsd()` but add each additional results file into a separate dimension of the produced 3-dimensional array (variable x time step x file). `read.4d.lsd()` adds the ability to read each instance of a multi-instanced variable to the fourth dimension of the generated 4D array (variable x instance x time step x file).

`list.files.lsd()` is a helper function to simplify the collection of results files to be used by the other functions in this package. It can be directly used to supply the `files` argument in the `read.xxx.lsd()` family of functions.

`select.colattrs.lsd()` and `select.colnames.lsd()` provide methods to extract/summarize information from previously imported LSD data structures.

`info.xxx.lsd()` functions provide information about LSD data structures. `name.xxx.lsd()` functions offer tools for dealing with LSD variable names in R.

For a complete list of exported functions, use `library( help = "LSDinterface" )`.

Author(s)

NA

Maintainer: NA

References

LSD documentation is available at https://www.labsimdev.org/.

The latest LSD binaries and source code can be downloaded at https://github.com/marcov64/Lsd/.

`info.details.lsd` *Get detailed information from a LSD results file*

Description

This function reads, analyze and organize the information from a LSD results file (.res).

Usage

```
info.details.lsd( file )
```

Arguments

`file`	the name of the LSD results file which the data are to be read from. If it does not contain an absolute path, the file name is relative to the current working directory, `getwd()`. Tilde-expansion is performed where supported. This can be a compressed file (see file) and must include the appropriated extension (usually `.res` or `.res.gz`).

Value

Returns a data frame containing detailed description (columns) of all variables (rows) contained in the selected results file.

Author(s)

Marcelo C. Pereira

See Also

`list.files.lsd()` `info.init.lsd()`, `info.names.lsd()` `info.dimensions.lsd()`

Examples

```
# get the list of file names of example LSD results
files <- list.files.lsd( system.file( "extdata", package = "LSDinterface" ) )

# get details about all variables in first file
info.details.lsd( files[ 1 ] )
```

`info.dimensions.lsd` *Dimension information for a LSD results file*

Description

This function reads some dimension information from a LSD results file (.res): number of time steps, number of variables and the original column (variable) names.

Usage

```
info.dimensions.lsd( file )
```

Arguments

`file`	the name of the LSD results file which the data are to be read from. If it does not contain an absolute path, the file name is relative to the current working directory, `getwd()`. Tilde-expansion is performed where supported. This can be a compressed file (see file) and must include the appropriated extension (usually `.res` or `.res.gz`).

Details

The returned number of time steps does not include the initial value (t = 0) for lagged variables (the second line of a .res format file).

Value

Returns a list containing two integer values and a character vector describing the selected results file.

`tSteps`	Number of time steps in file
`nVars`	Number of variables (including duplicated instances) in file
`varNames`	Names of variables (including duplicated instances) in file, after R name conversion

Author(s)

Marcelo C. Pereira

See Also

`list.files.lsd()` `info.details.lsd()`, `info.names.lsd()`, `info.init.lsd()`

Examples

```
# get the list of file names of example LSD results
files <- list.files.lsd( system.file( "extdata", package = "LSDinterface" ) )

# get dimensions from second file
info.dimensions.lsd( files[ 2 ] )
```

`info.distance.lsd` *Compute distance measure between LSD Monte Carlo time series and a set of references*

Description

This function reads a 3 or 4-dimensional array produced by `read.3d.lsd` or `read.4d.lsd` and computes several types of distance measures between the time series from a set of Monte Carlo runs and a set of reference time series (like the Monte Carlo average or median).

Usage

```
info.distance.lsd( array, references, instance = 1,
                   distance = "euclidean", std.dist = FALSE,
                   std.val = FALSE, rank = FALSE, weights = 1,
                   seed = 1, ... )
```

Arguments

`array`	a 3D or 4D array as produced by `read.3d.lsd` and `read.4d.lsd`, where in the first dimension (rows) you have the time steps, in the second (columns), the variables and in the third/fourth dimension, the Monte Carlo experiments, and the instances in the third dimension (4D arrays only). When 4D arrays are provided, only first instances are used in the computation.
`references`	a 2D matrix containing the reference time series, time in rows and variable values in named columns, from which the distance measures are to be computed. Columns must be named for the exact match to the names of the desired variables (contained in `array`). Only variables contained in both `array` and `references` are considered in the computation. According to the `distance` measure chosen, the number of time steps in `array` and `references` must be the same (as in the default Euclidean distance).

`instance`	integer: the instance of the variable to be read, for variables that exist in more than one object (4D `array` only). The default (1) is to read first instances.
`distance`	string: the distance measure to be used. The default is to compute the Euclidean distance (`"euclidean"`). For a comprehensive list of measure options, please refer to `TSDistances`. Measure names can be abbreviated.
`std.dist`	a logical value indicating, if `TRUE`, that the computed distances must be standardized with respect of the number of time steps involved. The default, `FALSE`, is not standardizing distances. This is relevant for properly comparing the metrics of series containing `NA`s.
`std.val`	a logical value indicating, if `TRUE`, that the series values must be standardized before computing the distances. The default, `FALSE`, is not standardizing values. This is relevant for properly comparing the metrics of series for different variables which are not distributed over the same range of values.
`rank`	a logical value indicating, if `TRUE`, that the Monte Carlo runs must be ranked in terms of closeness to the `references`. The default is not computing the run ranking, as this may be computationally expensive for some `distance` measures.
`weights`	a numerical vector containing the weights to be used for each variable in `references` when `rank = TRUE`. If vector has named elements, the vector names must exactly match the names of variables in `references`, order is not important, If variable names not present in vector, the missing ones are not considered in the ranking. If the vector is not named, the order of the weights must be the same as the one used for the variables (columns) in the `references` matrix. If the length of `weigths` is smaller the number of variables and not named, the vector is recycled. The default is to use the same weight for all variables.
`seed`	a single value, interpreted as an integer to define the pseudo-random number generator state used when sampling data, or `NULL`, to re-initialize the generator as if no seed had yet been set (a new state is created from the current time and the process ID).
`...`	additional parameters required by the specific method (see `TSDistances`).

Details

This function is a front-end to the extensive TSdist package for interfacing it with LSD generated data. Please check the associated documentation for further information.

TSdist package provides many different distance measure alternatives, including many that allow for different number of time steps among runs and references.

This function may also search the Monte Carlo run which has the overall smallest (standardized) distances from the given `references`. Irrespective of the options `std.dist` and `std.val`, the search uses always standardized values and distances for computation (this does not affect the distance measure matrix values).

One typical application of distance metrics is to select runs which are closer to the Monte Carlo average or median, that is, the runs which are more representative of the Monte Carlo Experiment. As there is no single criteria to define such "closeness", multiple distance measures may help to identify the set of most interesting runs.

Value

Returns a list containing:

`dist`	a named matrix containing the distances for each Monte Carlo run (lines) and variables (columns) contained both in `array` and `references` (and `weights`, if provided)
`close`	a named matrix of Monte Carlo run (sample) names, one column per variable, sorted in increasing distance order (closest runs in first line), which can be used to index the 3D or 4D `array`
`rank`	(only if `rank = TRUE`) a named vector of weighted Monte Carlo run standardized distances, sorted in increasing distance order (closest run first)

Note

When comparing distance measures between different Monte Carlo runs and variables, it is important to standardize the distances and values to ensure consistency. For variables which may present `NA` values, setting `std.dist = TRUE` ensures distance comparability by dividing the absolute distance of each run-reference pair by the number of effective (non-`NA`) time steps. When comparing variables which are dimensionally heterogeneous, `std.val = TRUE` uses the relative measure (between 1 and the run value divided by the corresponding reference value) to compute the distances.

When setting `std.val = TRUE`, all points in which the `references`' values are equal to zero are effectively removed from calculations. This behavior is always applied when searching for the closest Monte Carlo run(s).

Author(s)

Marcelo C. Pereira

See Also

`read.3d.lsd()`, `read.4d.lsd()`, `info.stats.lsd()`

Examples

```
# get the list of file names of example LSD results
files <- list.files.lsd( system.file( "extdata", package = "LSDinterface" ) )

# read first instance of all variables from MC files (3D array)
inst1Array <- read.3d.lsd( files )

# create statistics data frames for the variables
inst1Stats <- info.stats.lsd( inst1Array )

# compute the Euclidean distance to the mean for all variables and runs
inst1dist <- info.distance.lsd( inst1Array, inst1Stats$avg )
inst1dist$dist
inst1dist$close

# the same exercise but for a 4D array and Manhattan distance to the median
# plus indicating the Monte Carlo run closest to the median
```

```
allArray <- read.4d.lsd( files )
allStats <- info.stats.lsd( allArray, median = TRUE )
allDist <- info.distance.lsd( allArray, allStats$med, distance = "manhattan",
                              rank = TRUE )
allDist$dist
allDist$close
allDist$rank
names( allDist$rank )[ 1 ]  # results file name of the closest run
```

---

info.init.lsd *Read initial conditions from a LSD results file*

Description

This function reads the initial condition values from a LSD results file (.res).

Usage

```
info.init.lsd( file )
```

Arguments

`file`	the name of the LSD results file which the data are to be read from. If it does not contain an absolute path, the file name is relative to the current working directory, `getwd()`. Tilde-expansion is performed where supported. This can be a compressed file (see file) and must include the appropriated extension (usually `.res` or `.res.gz`).

Value

Returns a 1 line matrix containing the initial conditions (row 1) of all variables contained in the selected results file.

Note

The returned matrix contains all variables in the results file, even the ones that don't have an initial condition (indicated as NA). Only variables automatically initialized automatically by LSD in t = 1 are included here.

Author(s)

Marcelo C. Pereira

See Also

`list.files.lsd()` `info.details.lsd()`, `info.names.lsd()` `info.dimensions.lsd()`

Examples

```
# get the list of file names of example LSD results
files <- list.files.lsd( system.file( "extdata", package = "LSDinterface" ) )

# get initialization data from first and second files
init1 <- info.init.lsd( files[ 1 ] )
init1[ , 4 : 8 ]

init2 <- info.init.lsd( files[ 2 ] )
init2[ , 4 : 8 ]
```

`info.names.lsd` *Read unique variable names from a LSD results file (no duplicates)*

Description

This function reads the variable names (columns) from a LSD results file (.res). The names returned are converted to the original LSD names whenever possible and duplicates are removed.

Usage

```
info.names.lsd( file )
```

Arguments

`file`	the name of the LSD results file which the data are to be read from. If it does not contain an absolute path, the file name is relative to the current working directory, `getwd()`. Tilde-expansion is performed where supported. This can be a compressed file (see file) and must include the appropriated extension (usually `.res` or `.res.gz`).

Value

Returns a character vector containing the names of all unique variables contained in the selected results file.

Note

Not all names can be automatically reconverted to the original LSD names, using LSD/C++ naming conventions.

The conversion may be incorrect if the original LSD variable is named in the format "X_...".

Author(s)

Marcelo C. Pereira

See Also

`list.files.lsd()` `info.details.lsd()`, `info.init.lsd()` `info.dimensions.lsd()`

Examples

```
# get the list of file names of example LSD results
files <- list.files.lsd( system.file( "extdata", package = "LSDinterface" ) )

# get variable names from first file
info.names.lsd( files[ 1 ] )
```

`info.stats.lsd` *Compute Monte Carlo statistics from a set of LSD runs*

Description

This function reads a 3 or 4-dimensional array produced by `read.3d.lsd` or `read.4d.lsd` and produces a list with 2D data frames containing the (Monte Carlo) mean, the standard deviation, the maximum, the minimum, and other optional statistics for each variable, at each time step.

Usage

```
info.stats.lsd( array, rows = 1, cols = 2, median = FALSE,
                ci = c( "none", "mean", "median", "auto" ),
                ci.conf = 0.95, ci.boot = NULL, boot.R = 999,
                seed = 1, na.rm = TRUE, inf.rm = TRUE )
```

Arguments

`array`	a 3D or 4D array as produced by `read.3d.lsd` and `read.4d.lsd`, where in the first dimension (rows) you have the time steps, in the second (columns), the variables and in the third/fourth dimension, the Monte Carlo experiments, and the instances in the third dimension (4D arrays only).
`rows`	an integer array dimension to be used as the rows for the statistics matrices, default is to use first array dimension.
`cols`	an integer array dimension to be used as the columns for the statistics matrices, default is to use second array dimension.
`median`	a logical value indicating if (`TRUE`) the median and the median absolute deviation should also be computed. The default (`FALSE`) is not to compute these statistics.
`ci`	a character string specifying the type of confidence interval to compute, must be one of `"none"` (default) for no confidence interval computation, `"mean"`, to compute a confidence interval for the mean, `"median"`, for the median, or `"auto"`, to use the option set for the `median` argument (above). This option can be abbreviated.
`ci.conf`	confidence level of the confidence interval.
`ci.boot`	a character string specifying the type of bootstrap confidence interval to compute, must be one of `"basic"`, `"perc"` (percentile interval), or `"bca"` (BCa - adjusted percentile interval). If set to `NULL` or an empty string, a regular asymptotic confidence interval is produced (no bootstrap), assuming normal distribution for the mean or using a non-parametric rank test for the median. Non-bootstrap percentiles are much faster to compute but generally less accurate.

`boot.R`	number of bootstrap replicates.
`seed`	a single value, interpreted as an integer to define the pseudo-random number generator state used for the bootstrap process, or `NULL`, to re-initialize the generator as if no seed had yet been set (a new state is created from the current time and the process ID).
`na.rm`	a logical value indicating whether `NA` values should be stripped before the computation proceeds.
`inf.rm`	a logical value indicating whether non-finite values should be stripped before the computation proceeds.

Value

Returns a list containing four to seven matrices, with the original size and naming of the selected 2 dimensions of the argument.

`avg`	a matrix with the mean of the MC experiments
`sd`	a matrix with the standard deviation of the MC experiments
`max`	a matrix with the maximum value of the MC experiments
`min`	a matrix with the minimum value of the MC experiments
`med`	a matrix with the median of the MC experiments (only present if argument `median = TRUE`)
`mad`	a matrix with the median absolute deviation of the MC experiments (only present if argument `median = TRUE`)
`ci.hi`	a matrix with the maximum value of the MC experiments (only present if argument `ci` is not set to `"none"`)
`ci.lo`	a matrix with the minimum value of the MC experiments (only present if argument `ci` is not set to `"none"`)
`n`	a matrix with the number of observations available for computation of statistics

Author(s)

Marcelo C. Pereira

See Also

`list.files.lsd()` `read.3d.lsd()`, `read.4d.lsd()`, `info.dimensions.lsd()`

Examples

```
# get the list of file names of example LSD results
files <- list.files.lsd( system.file( "extdata", package = "LSDinterface" ) )

# read first instance of all variables from MC files (3D array)
inst1Array <- read.3d.lsd( files )

# create statistics data frames for the variables
inst1Stats <- info.stats.lsd( inst1Array )
```

```
print( inst1Stats$avg[ 10 : 20, ] )
print( inst1Stats$sd[ 10 : 20, ] )

# organize the stats, including medians, by variable (dim=2) and file (dim=3)
inst1Stats2 <- info.stats.lsd( inst1Array, rows = 2, cols = 3, median = TRUE )
print( inst1Stats2$med[ , 1 : 2 ] )

# the same but for all instance of all variables (from a 4D array)
# and a normal (non-boostrap) confidence intervals for the means
allArray <- read.4d.lsd( files )
allStats <- info.stats.lsd( allArray, ci = "auto" )
print( allStats$ci.lo[ 3, 1 : 7 ] )
print( allStats$avg[ 3, 1 : 7 ] )
print( allStats$ci.hi[ 3, 1 : 7 ] )

# organize the stats by file (dim=4) and variable (dim=2)
# plus boostrat confidence intervals for the median
allStats2 <- info.stats.lsd( allArray, rows = 4, cols = 2, median = TRUE,
                             ci = "auto", ci.boot = "bca" )
print( allStats2$ci.lo[ , 1 : 3 ] )
print( allStats2$med[ , 1 : 3 ] )
print( allStats2$ci.hi[ , 1 : 3 ] )
```

list.files.lsd *List results files from a set of LSD runs*

Description

This function produce a character vector of the names of results files produced after the execution of LSD simulation runs. The list can be used with all function in this package requiring the argument `files`.

Usage

```
list.files.lsd( path = ".", conf.name = "",
                type = c( "res", "tot", "csv" ),
                compressed = NULL, recursive = FALSE,
                join = FALSE, full.names = FALSE,
                sensitivity = FALSE )
```

Arguments

`path`	a character vector of full or relative path name to the base directory from where to search the files; the default corresponds to the working directory, `getwd()`. Tilde expansion is performed. Alternatively, the full path and name of the corresponding LSD configuration file (including the `.lsd` extension) can be provided.

`conf.name`	the LSD configuration file name (optionally including the `.lsd` extension) used to generate the desired results files; the default is to return all results files, irrespective of the configuration file used. Alternatively, a regular expression can be supplied. This argument takes precedence of any configuration file name provided together with the `path` argument.
`type`	the type (format/extension) of LSD results files to use among the options `c( "res", "tot", "csv" )`, used to define the extension of the files to be considered. `"res"` is the default. This option can be abbreviated.
`compressed`	a logical value indicating if (`TRUE`) to look only for compressed files with `.gz` extension, or uncompressed ones otherwise (`FALSE`). The default (`NULL`) is to list files irrespective if compressed or not.
`recursive`	a logical value indicating if the listing should recurse into sub-directories of `path`. The default (`FALSE`) is to scan just the sub-directory with the same name as `conf.name` (without the `.lsd` extension or numeric tags), if present (regular expression in `conf.name` is not considered), and `path`. If `TRUE`, the entire sub-directory tree, starting at `path`, is scanned for files.
`join`	a logical value indicating if results files from multiple sub-directories should be joined together in the return list. The default (`FALSE`) is to list files from just a single sub-directory, the first one found during the search starting from `path`.
`full.names`	a logical value specifying if (`TRUE`) the file names should be expanded to absolute path names. The default (`FALSE`) is to use relative (to `path`) file names.
`sensitivity`	a logical value specifying if (`TRUE`) the target results files are part of a sensitivity analysis design of experiment (DoE), which are double numbered in a particular format (`conf.name_XXX_YYY.res[.gz]`). The default (`FALSE`) is to assume files are just single numbered, which is usually inappropriate for DoE results files. See LSDsensitivity package documentation for details.

Details

The order by which sub-directories are explored may be relevant. By default, the function scans for results files in a sub-directory named as `conf.name`, if present, in the given initial directory `path`. Next, if `conf.name` has a numeric suffix in the format `name_XXX`, where XXX is any number of algarisms, it searches the sub-directory `name`, if present. Finally, it scans the initial `path` itself. If results files are present in more than one sub-directory, function returns only the files found in first one (except if `join = TRUE`), and issues a warning message. If `recursive = TRUE`, file search starts from `path` and proceeds until it encompasses the entire sub-directory tree. In this case, if multiple sub-directories contain the desired files, only the initial `path` takes precedence, and the rest of the tree is recurred in alphabetical order.

Please note that joining files from different sub-directories (`join = TRUE`) may combine results with incompatible data which cannot be processed together by the `read.xxx.lsd()` family of functions.

Value

A character vector containing the names of the found results files in the specified (sub) directories (empty if there were no files). If a path does not exist or is not a directory or is unreadable it is skipped.

Note

File naming conventions are platform dependent. The pattern matching works with the case of file names as returned by the OS.

`path` must specify paths which can be represented in the current codepage, and files/directories below path whose names cannot be represented in that codepage will most likely not be found.

Author(s)

Marcelo C. Pereira

See Also

`read.3d.lsd()`, `read.4d.lsd()`, `read.raw.lsd()`, `read.single.lsd()`, `read.multi.lsd()`, `read.list.lsd()`, LSDsensitivity package,

Examples

```
# get the names of all files the example directory
list.files.lsd( system.file( "extdata", package = "LSDinterface" ) )

# expand search to the entire example directory tree
# for results from a configuration file named "Sim1.lsd"
# and join files found in all sub-directories conatining data
list.files.lsd( system.file( "extdata", package = "LSDinterface" ),
                "Sim1.lsd", recursive = TRUE, join = TRUE )
```

`name.check.lsd` *Check a set of LSD variables names against a LSD results file*

Description

This function checks if all variable names in a set are valid for a LSD results file (.res). If no name is provided, the function returns all the valid unique variable names in the file.

Usage

```
name.check.lsd( file, col.names = NULL, check.names = TRUE )
```

Arguments

`file`	the name of the LSD results file which the data are to be read from. If it does not contain an absolute path, the file name is relative to the current working directory, `getwd()`. This can be a compressed file (see file) and must include the appropriated extension (usually `.res` or `.res.gz`).
`col.names`	a vector of optional names for the variables. The default is to read all (unique) variable names.
`check.names`	logical. If `TRUE` then the names of the variables are checked to ensure that they are syntactically valid variable names. If necessary they are adjusted to ensure that there are no duplicates.

Value

Returns a string vector containing the (original) valid variable names contained in the results file, using LSD/C++ naming conventions.

Author(s)

Marcelo C. Pereira

See Also

`list.files.lsd()` `info.names.lsd()`,

Examples

```
# get the list of file names of example LSD results
files <- list.files.lsd( system.file( "extdata", package = "LSDinterface" ) )

# check all variable names
name.check.lsd( files[ 1 ] )

# check just two names
name.check.lsd( files[ 2 ], col.names = c( "GDP", "_growth1" ) )
```

`name.clean.lsd` *Get clean (R) variable name*

Description

This function produces a more appropriate variable name from R initial column name conversion.

Usage

```
name.clean.lsd( r.name )
```

Arguments

`r.name`	a string, a vector of strings, or an object which can be coerced to a character vector by `as.character`, from the column names produced by reading a LSD results file.

Details

The function removes the extra/ending '.' characters introduced by R and introduces a '_' between time span values.

Value

A string or a string vector with the same attributes as x (after possible coercion) and the format `NAME.POSITION.INI_END`.

Author(s)

Marcelo C. Pereira

See Also

name.var.lsd(), name.nice.lsd(), info.names.lsd()

Examples

```
name.clean.lsd( "Var1.1_1..1.100." )

name.clean.lsd( c( "Var1.1_1..1.100.", "Var2.1_2_3..50.70." ) )
```

---

name.nice.lsd *Get a nice (R) variable name*

Description

This function produces a nicer variable name from R initial column name conversion, in particular removing leading underscores.

Usage

```
name.nice.lsd( r.name )
```

Arguments

r.name	a string, a vector of strings, or an object which can be coerced to a character vector by `as.character`, from the column names produced by reading a LSD results file.

Details

The function removes the extra/ending '.' characters introduced by R and introduces a '_' between time span values and deletes leading underscores ('_'), converted to 'X_' by R.

Value

A string or a string vector with the same attributes as x (after possible coercion) and the format `NAME[.POSITION.INI_END]`.

Author(s)

Marcelo C. Pereira

See Also

name.var.lsd(), name.clean.lsd(), info.names.lsd()

Examples

```
name.nice.lsd( "X_Var1.1_1..1.100." )

name.nice.lsd( c( "_Var1.1_1..1.100.", "X_Var2.1_2_3..50.70." ) )

name.nice.lsd( c( "_Var1", "X_Var2" ) )
```

`name.r.unique.lsd` *Get valid unique R variable name*

Description

This function produces a valid and unique variable name from names produced from multi-instanced LSD variables (as in `read.raw.lsd`).

Usage

```
name.r.unique.lsd( r.name )
```

Arguments

`r.name`	a string, a vector of strings, or an object which can be coerced to a character vector by `as.character`, from the column names produced by reading a LSD results file.

Details

The function removes the trailing '.' characters, and the text between, introduced during the conversion from LSD results files, add an 'X' prefix to names started by an '_'. After this initial transformation, all repeated variable names (originated from multi-instanced variables) are removed.

The produced names are R valid variable names, similar to the original LSD/C++ variable names, but with an 'X' prepended to variables starting with an '_' (which are invalid in R).

Value

A string or a string vector of converted string(s) including only non-repeated ones.

Author(s)

Marcelo C. Pereira

See Also

`name.var.lsd()`, `name.clean.lsd()`, `name.nice.lsd()`, `info.names.lsd()`

Examples

```
name.r.unique.lsd( "Var1.1_1.1_100" )

name.r.unique.lsd( c( "Var1.1_1.1_100", "_Var2.1_1.1_100", "_Var2.1_2.50_70" ) )
```

`name.var.lsd` *Get original LSD variable name*

Description

This function generates the original LSD variable name, as it was defined in LSD and before R adjusts the name, from a R column name (with or without position or timing information appended).

Usage

```
name.var.lsd( r.name )
```

Arguments

`r.name`	a string, a vector of strings, or an object which can be coerced to a character vector by `as.character`, from the column names produced by reading a LSD results file.

Details

The conversion may be incorrect if the original LSD variable is named in the format "X_...". No checking is done to make sure the variable really exists.

Value

A string or a string vector with the same attributes as x (after possible coercion).

Author(s)

Marcelo C. Pereira

See Also

`name.clean.lsd()`, `info.names.lsd()`

Examples

```
name.var.lsd( "label" )

name.var.lsd( c( "label", "X_underlinelabel" ) )
```

read.3d.lsd — *Read one instance of LSD variables (time series) from multiple LSD results files into a 3D array*

Description

This function reads the data series associated to a specific instance of each selected variable from a set of LSD results files (.res) and saves them into a 3-dimensional array (time step x variable x file).

Usage

```
read.3d.lsd( files, col.names = NULL, nrows = -1, skip = 0,
             check.names = TRUE, instance = 1, nnodes = 1,
             posit = NULL, posit.match = c( "fixed", "glob", "regex" ) )
```

Arguments

`files`	a character vector containing the names of the LSD results files which the data are to be read from. If they do not contain an absolute path, the file names are relative to the current working directory, `getwd()`. These can be compressed files and must include the appropriated extension (usually `.res` or `.res.gz`).
`col.names`	a vector of optional names for the variables. The default is to read all variables.
`nrows`	integer: the maximum number of time steps (rows) to read in. Negative and other invalid values are ignored. The default is to read all rows.
`skip`	integer: the number of time steps (rows) of the results file to skip before beginning to read data. The default is to read from the first time step (t = 1).
`check.names`	logical. If `TRUE` the names of the variables are checked to ensure that they are syntactically valid variable names. If necessary they are adjusted (by make.names) so that they are, and also to ensure that there are no duplicates.
`instance`	integer: the instance of the variable to be read, for variables that exist in more than one object. This number is based on the position (column) of the variable in the results file. The default (1) is to read first instances.
`nnodes`	integer: the maximum number of parallel computing nodes (parallel threads) in the current computer to be used for reading the files. The default, `nnodes = 1`, means single thread processing (no parallel threads). If equal to zero, creates up to one node per CPU core. Only `PSOCK` clusters are used, to ensure compatibility with any platform. Please note that each node requires its own memory space, so memory usage increases linearly with the number of nodes.
`posit`	a string, a vector of strings or an integer vector describing the LSD object position of the variable(s) to select. If an integer vector, it should define the position of a SINGLE LSD object. If a string or vector of strings, each element should define one or more different LSD objects, so the returning matrix may contain variables from more than one object. By setting `posit.match`, globbing (wildcard), and regular expressions can be used to select multiple objects at once.

posit.match a string defining how the posit argument, if provided, should be matched against the LSD object positions. If equal to "fixed", the default, only exact matching is done. "glob" allows using simple wildcard characters ('*' and '?') in posit for matching. If posit.match="regex" interpret posit as POSIX 1003.2 extended regular expression(s). See regular expressions for details of the different types of regular expressions. Options can be abbreviated.

Details

Selection restriction arguments can be provided as needed; when not specified, all available cases are considered, but just one instance is considered.

When posit is supplied together with col.names or instance, the selection process is done in two steps. Firstly, the column names and the instance position set by col.names and instance are selected. Secondly, the instances defined by posit are selected from the first selection set.

See select.colnames.lsd and select.colattrs.lsd for examples on how to apply advanced selection options.

Value

Returns a 3D array containing data series from the selected variables.

The array dimension order is: time x variable x file.

Note

If the selected files don't have the same columns available (names and instances), after column selection, an error is produced.

Author(s)

Marcelo C. Pereira

See Also

list.files.lsd() read.4d.lsd(), read.single.lsd(), read.multi.lsd(), read.list.lsd(), read.raw.lsd()

Examples

```
# get the list of file names of example LSD results
files <- list.files.lsd( system.file( "extdata", package = "LSDinterface" ) )

# read first instance of all variables from files (one level each),
# pasting the directory where the example files are (not required if in working dir)
inst1Array <- read.3d.lsd( files )
print( inst1Array[ 5 : 10, 1 : 7, 1 ] )
print( inst1Array[ 5 : 10, 1 : 7, 2 ] )
print( inst1Array[ 5 : 10, 1 : 7, 3 ] )

# read first instance of a set of variables named _A1p and _growth1
ab1Array <- read.3d.lsd( files, c( "_A1p", "_growth1" ) )
```

```
print( ab1Array[ 20 : 25, , 1 ] )
print( ab1Array[ 20 : 25, , 2 ] )
print( ab1Array[ 20 : 25, , 3 ] )

# read instance 2 of all variables, skipping the initial 20 time steps
# and keeping up to 30 time steps (from t = 21 up to t = 30)
inst2Array21_30 <- read.3d.lsd( files, skip = 20, nrows = 30, instance = 2 )
print( inst2Array21_30[ , , "Sim1_1" ] )   # use the file name to retrieve
print( inst2Array21_30[ , , "Sim1_2" ] )

# read instance 5 of all variables in second-level objects, using up to 2 cores
inst5array2 <- read.3d.lsd( files, instance = 2, posit = "*_*",
                            posit.match = "glob", nnodes = 2 )
print( inst5array2[ 11 : 20, , 1 ] )
```

read.4d.lsd *Read multiple instances of LSD variables (time series) from a set of LSD results file into a 4D array*

Description

This function reads the data series associated to a set of instances of each selected variable from a set of LSD results files (.res) and saves them into a 4-dimensional array (time x variable x instance x file).

Usage

```
read.4d.lsd( files, col.names = NULL, nrows = -1, skip = 0,
             check.names = TRUE, pool = FALSE, nnodes = 1,
             posit = NULL, posit.match = c( "fixed", "glob", "regex" ) )
```

Arguments

`files`	a character vector containing the names of the LSD results files which the data are to be read from. If they do not contain an absolute path, the file names are relative to the current working directory, `getwd()`. These can be compressed files and must include the appropriated extension (usually `.res` or `.res.gz`).
`col.names`	a vector of optional names for the variables. The default is to read all variables.
`nrows`	integer: the maximum number of time steps (rows) to read in. Negative and other invalid values are ignored. The default is to read all rows.
`skip`	integer: the number of time steps (rows) of the results file to skip before beginning to read data. The default is to read from the first time step (t = 1).
`check.names`	logical. If `TRUE` the names of the variables are checked to ensure that they are syntactically valid variable names. If necessary they are adjusted (by make.names) so that they are, and also to ensure that there are no duplicates.

`pool`	logical. If `TRUE`, variables instances from all files are concatenated (by columns) as a single 3-dimensional array. If `FALSE` (the default), each file is saved as a separated dimension (fourth) in the array.
`nnodes`	integer: the maximum number of parallel computing nodes (parallel threads) in the current computer to be used for reading the files. The default, `nnodes = 1`, means single thread processing (no parallel threads). If equal to zero, creates up to one node per CPU core. Only `PSOCK` clusters are used, to ensure compatibility with any platform. Please note that each node requires its own memory space, so memory usage increases linearly with the number of nodes.
`posit`	a string, a vector of strings or an integer vector describing the LSD object position of the variable(s) to select. If an integer vector, it should define the position of a SINGLE LSD object. If a string or vector of strings, each element should define one or more different LSD objects, so the returning matrix will contain variables from more than one object. By setting `posit.match`, globbing (wildcard), and regular expressions can be used to select multiple objects at once; in this case, all matching objects are returned.
`posit.match`	a string defining how the `posit` argument, if provided, should be matched against the LSD object positions. If equal to `"fixed"`, the default, only exact matching is done. `"glob"` allows using simple wildcard characters ('*' and '?') in `posit` for matching. If `posit.match="regex"` interpret `posit` as POSIX 1003.2 extended regular expression(s). See `regular expressions` for details of the different types of regular expressions. Options can be abbreviated.

Details

Selection restriction arguments can be provided as needed; when not specified, all available cases are selected.

When `posit` is supplied together with `col.names`, the selection process is done in two steps. Firstly, the column names set by `col.names` are selected. Secondly, the instances defined by `posit` are selected from the first selection set.

See `select.colnames.lsd` and `select.colattrs.lsd` for examples on how to apply advanced selection options.

Value

Returns a 4D array containing data series for each instance from the selected variables.

The array dimension order is: time x variable x instance x file.

When `pool = TRUE`, the produced array is 3-dimensional. Pooling require that selected columns contains EXACTLY the same variables (number of instances may be different).

Note

If the selected files don't have the same columns available (names), after column selection, an error is produced.

When using the option `pool = TRUE`, columns from multiple files are consolidated with their original names plus the file name, to keep all column names unique. Use `name.var.lsd` to get just the LSD name of the variable corresponding to each column.

Author(s)

Marcelo C. Pereira

See Also

`list.files.lsd()` `read.3d.lsd()`, `read.single.lsd()`, `read.multi.lsd()`, `read.list.lsd()`, `read.raw.lsd()`

Examples

```
# get the list of file names of example LSD results
files <- list.files.lsd( system.file( "extdata", package = "LSDinterface" ) )

# read all instances of all variables from files,
allArray <- read.4d.lsd( files )
print( allArray[ 1 : 10, 1 : 7, 1, 1 ] ) # 1st instance of 1st file (7 vars and 10 times)
print( allArray[ 11 : 20, "X_A1p", , "Sim1_2" ] ) # all instances of _A1p in Sim1_2 (10 times)
print( allArray[ 50, 9, , ] ) # all instances of all files of 9th variable for t=50

# the same, but pooling all files into a single (3D!) array
allArrayPool <- read.4d.lsd( files, pool = TRUE )
print( allArrayPool[ 1 : 10, 8 : 9, 3 ] ) # 3rd instances of last 2 vars (10 times)
print( allArrayPool[ 11 : 20, "X_A1p", 4 : 9 ] ) # 6 instances of _A1p variable (10 times)
print( allArrayPool[ 50, 9, 4 : 9 ] ) # 6 instances of all files of 9th variable for t=50

# read instances of a set of variables named '_A1p' and '_growth1'
abArray <- read.4d.lsd( files, c( "_A1p", "_growth1" ) )
print( abArray[ 1 : 10, , 1, 2 ] ) # 1st instances of 2nd file (all vars and 10 times)
print( abArray[ 11 : 20, 2, , "Sim1_3" ] ) # all instances of 2nd variable in Sim1_3 (10 times)
print( abArray[ 50, "X_A1p", , ] ) # all instances of all files of _A1p variable for t=50

# read all variables/variables, skipping the initial 20 time steps
# and keeping up to 30 time steps (from t = 21 up to t = 30)
allArray21_30 <- read.4d.lsd( files, skip = 20, nrows = 30 )
print( allArray21_30[ , "X_growth1", , 2 ] ) # all instances of _growth1 variable in 2nd file
print( allArray21_30[ 10, 8, , ] ) # all instances of all files of 8th variable for t=30

# read all variables in second-level objects, using up to 2 cores for processing
abArray2 <- read.4d.lsd( files, posit = "*_*", posit.match = "glob", nnodes = 2 )
print( abArray2[ 11 : 20, , 5, "Sim1_1" ] ) # 5th instances in Sim1_1 file
```

`read.list.lsd` *Read one or more instances of LSD variables (time series) from a set of LSD results file into a list*

Description

This function reads the data series associated to a specific or a set of instances of each selected variable from a set of LSD results file (.res) and saves them into separated matrices (one per file).

Usage

```
read.list.lsd( files, col.names = NULL, nrows = -1, skip = 0,
               check.names = TRUE, instance = 0, pool = FALSE, nnodes = 1,
               posit = NULL, posit.match = c( "fixed", "glob", "regex" ) )
```

Arguments

`files`	a character vector containing the names of the LSD results files which the data are to be read from. If they do not contain an absolute path, the file names are relative to the current working directory, `getwd()`. These can be compressed files and must include the appropriated extension (usually `.res` or `.res.gz`).
`col.names`	a vector of optional names for the variables. The default is to read all variables.
`nrows`	integer: the maximum number of time steps (rows) to read in. Negative and other invalid values are ignored. The default is to read all rows.
`skip`	integer: the number of time steps (rows) of the results file to skip before beginning to read data. The default is to read from the first time step (t = 1).
`check.names`	logical. If `TRUE` the names of the variables are checked to ensure that they are syntactically valid variable names. If necessary they are adjusted (by make.names) so that they are, and also to ensure that there are no duplicates.
`instance`	integer: the instance of the variable to be read, for variables that exist in more than one object. This number is based on the position (column) of the variable in the results file. The default (0) is to read all instances.
`pool`	logical. If `TRUE`, variables instances from all files are concatenated (by columns) into a single matrix. If `FALSE` (the default), each file is saved as a separated matrix in a list.
`nnodes`	integer: the maximum number of parallel computing nodes (parallel threads) in the current computer to be used for reading the files. The default, `nnodes = 1`, means single thread processing (no parallel threads). If equal to zero, creates up to one node per CPU core. Only `PSOCK` clusters are used, to ensure compatibility with any platform. Please note that each node requires its own memory space, so memory usage increases linearly with the number of nodes.
`posit`	a string, a vector of strings or an integer vector describing the LSD object position of the variable(s) to select. If an integer vector, it should define the position of a SINGLE LSD object. If a string or vector of strings, each element should define one or more different LSD objects, so the returning matrix will contain variables from more than one object. By setting `posit.match`, globbing (wildcard), and regular expressions can be used to select multiple objects at once; in this case, all matching objects are returned.
`posit.match`	a string defining how the `posit` argument, if provided, should be matched against the LSD object positions. If equal to `"fixed"`, the default, only exact matching is done. `"glob"` allows using simple wildcard characters ('*' and '?') in `posit` for matching. If `posit.match="regex"` interpret `posit` as POSIX 1003.2 extended regular expression(s). See `regular expressions` for details of the different types of regular expressions. Options can be abbreviated.

Details

Selection restriction arguments can be provided as needed; when not specified, all available cases are selected.

When `posit` is supplied together with `col.names` or `instance`, the selection process is done in two steps. Firstly, the column names and instance positions set by `col.names` and `instance` are selected. Secondly, the instances defined by `posit` are selected from the first selection set.

See `select.colnames.lsd` and `select.colattrs.lsd` for examples on how to apply advanced selection options.

Value

Returns a named list of matrices with the selected variables' time series in the results files. If `pool = TRUE`, the return value is a single, consolidated matrix (column names are not unique).

The matrices dimension order is: time x variable.

Matrix column names are only "cleaned" if there are just single instanced variables selected. When multiple instanced variables are present, the column names include all the header information contained in the LSD results file. The name of the LSD variable associated to any column name can be retrieved with `name.var.lsd`.

Note

When using the option `pool = TRUE`, columns from multiple files are consolidated with their original names plus the file name, to keep all column names unique. Use `name.var.lsd` to get just the LSD name of the variable corresponding to each column.

The returned matrices may be potentially very wide, in particular if variables are not well selected(see `col.names` above) or if there is a large number of instances.

Author(s)

Marcelo C. Pereira

See Also

`list.files.lsd()` `name.var.lsd()` `read.single.lsd()`, `read.multi.lsd()`, `read.3d.lsd()`, `read.4d.lsd()`, `read.raw.lsd()`,

Examples

```
# get the list of file names of example LSD results
files <- list.files.lsd( system.file( "extdata", package = "LSDinterface" ) )

# read all instances of all variables from three files (one matrix each),
tableList <- read.list.lsd( files )
print( tableList[[ 1 ]][ 1 : 5, 1 : 7 ] )
print( tableList[[ 2 ]][ 1 : 5, 1 : 7 ] )
print( tableList[[ 3 ]][ 1 : 5, 1 : 7 ] )

# read all instances of a set of variables named '_A1p' and '_growth1'
# and pool data into a single matrix
abTable <- read.list.lsd( files, c( "_A1p", "_growth1" ), pool = TRUE )
print( abTable[ 10 : 20, 10 : 12 ] )

# read instance 4 of all variables, skipping the initial 20 time steps
# and keeping up to 30 time steps (from t = 21 up to t = 30)
inst4List21_30 <- read.list.lsd( files, skip = 20, nrows = 30, instance = 4 )
print( inst4List21_30[[ 1 ]] )
print( inst4List21_30[[ 2 ]] )

# read all variables in top-level objects, using up to 2 cores for processing
instTop <- read.list.lsd( files, posit = 1, nnodes = 2 )
print( instTop$Sim1_1[ 11 : 20, ] )   # use the file name to retrieve list item
print( instTop$Sim1_2[ 11 : 20, ] )
```

`read.multi.lsd` *Read all instances of LSD variables (time series) from a LSD results file*

Description

This function reads the data series associated to all instances of each selected variable from a LSD results file (.res).

Usage

```
read.multi.lsd( file, col.names = NULL, nrows = -1, skip = 0,
                check.names = TRUE, posit = NULL,
                posit.match = c( "fixed", "glob", "regex" ),
                posit.cols = FALSE )
```

Arguments

`file`	the name of the LSD results file which the data are to be read from. If it does not contain an absolute path, the file name is relative to the current working directory, `getwd()`. This can be a compressed file (see file) and must include the appropriated extension (usually `.res` or `.res.gz`).
`col.names`	a vector of optional names for the variables. The default is to read all variables.
`nrows`	integer: the maximum number of time steps (rows) to read in. Negative and other invalid values are ignored. The default is to read all rows.
`skip`	integer: the number of time steps (rows) of the results file to skip before beginning to read data. The default is to read from the first time step (t = 1).
`check.names`	logical. If `TRUE` the names of the variables are checked to ensure that they are syntactically valid variable names. If necessary they are adjusted (by make.names) so that they are, and also to ensure that there are no duplicates.

`posit`	a string, a vector of strings or an integer vector describing the LSD object position of the variable(s) to select. If an integer vector, it should define the position of a SINGLE LSD object. If a string or vector of strings, each element should define one or more different LSD objects, so the returning matrix will contain variables from more than one object. By setting `posit.match`, globbing (wildcard), and regular expressions can be used to select multiple objects at once; in this case, all matching objects are returned.
`posit.match`	a string defining how the `posit` argument, if provided, should be matched against the LSD object positions. If equal to `"fixed"`, the default, only exact matching is done. `"glob"` allows using simple wildcard characters ('*' and '?') in `posit` for matching. If `posit.match="regex"` interpret `posit` as POSIX 1003.2 extended regular expression(s). See `regular expressions` for details of the different types of regular expressions. Options can be abbreviated.
`posit.cols`	logical. If `TRUE` just the position information is used as the names of the columns in each variable list. If `FALSE`, the default, the column names include all the header information contained in the LSD results file (name, position and time span).

Details

Selection restriction arguments can be provided as needed; when not specified, all available cases are selected.

When `posit` is supplied together with `col.names`, the selection process is done in two steps. Firstly, the column names set by `col.names` are selected. Secondly, the instances defined by `posit` are selected from the first selection set.

See `select.colnames.lsd` and `select.colattrs.lsd` for examples on how to apply advanced selection options.

Value

Returns a named list of matrices, each containing one of the selected variables' time series from the results file.

Variable names are converted to valid R ones when defining list names. Matrix column names are not "cleaned", even for single instanced variables. The column names include all the header information contained in the LSD results file.

Note

For extracting data from multiple similar files (like sensitivity analysis results), see `read.list.lsd`.

Author(s)

Marcelo C. Pereira

See Also

`list.files.lsd()` `read.single.lsd()`, `read.list.lsd()`, `read.3d.lsd()`, `read.4d.lsd()`, `read.raw.lsd()`

Examples

```
# get the list of file names of example LSD results
files <- list.files.lsd( system.file( "extdata", package = "LSDinterface" ) )

# load first .res file into a simple matrix (all instances),
macroList <- read.multi.lsd( files[ 1 ] )
length( macroList )           # number of lists holding variables
names( macroList )            # name of each list
print( macroList[[ 1 ]][ 1 : 5, , drop = FALSE ] )
print( macroList$X_A1p[ 10 : 20, ] )

# read first instance of 2 variables, skipping the initial 20 time steps
# and keeping up to 30 time steps (from t = 21 up to t = 30), positions in cols
varsList21_30 <- read.multi.lsd( files[ 2 ], c( "_A1p", "_growth1" ),
                                 skip = 20, nrows = 30, posit.cols = TRUE )
print( varsList21_30[[ 1 ]] )
print( varsList21_30$X_growth1 )
```

read.raw.lsd — *Read LSD results file and clean variables names*

Description

This function reads all the data series in a LSD results file (.res).

Usage

```
read.raw.lsd( file, nrows = -1, skip = 0, col.names = NULL,
              check.names = TRUE, clean.names = FALSE, instance = 0,
              posit = NULL, posit.match = c( "fixed", "glob", "regex" ) )
```

Arguments

`file`	the name of the LSD results file which the data are to be read from. If it does not contain an absolute path, the file name is relative to the current working directory, `getwd()`. This can be a compressed file (see file) and must include the appropriated extension (usually `.res` or `.res.gz`).
`nrows`	integer: the maximum number of time steps (rows) to read in. Negative and other invalid values are ignored. The default is to read all rows.
`skip`	integer: the number of time steps (rows) of the results file to skip before beginning to read data. The default is to read from the first time step (t = 1).
`col.names`	a vector of optional names for the variables. The default is to read all variables. The names must to be in LSD/C++ format, without dots (".") in the name. Any dot (and trailing characters) will be automatically removed.
`check.names`	logical. If `TRUE` the names of the variables are checked to ensure that they are syntactically valid variable names. If necessary they are adjusted to ensure that there are no duplicates.

`clean.names`	logical. If `TRUE` the names of the variables in the columns are "cleaned" to remove extra information from the header in the LSD results file. This option is incompatible (and will be ignored) when multiple instances of a single variable are selected. If `FALSE`, the default, preserve extra information in the names.
`instance`	integer: the instance of the variable to be read, for variables that exist in more than one object. This number is based on the relative position (column) of the variable in the results file. The default (0) is to read all instances.
`posit`	a string, a vector of strings or an integer vector describing the LSD object position of the variable(s) to select. If an integer vector, it should define the position of a SINGLE LSD object. If a string or vector of strings, each element should define one or more different LSD objects, so the returning matrix will contain variables from more than one object. By setting `posit.match`, globbing (wildcard), and regular expressions can be used to select multiple objects at once; in this case, all matching objects are returned.
`posit.match`	a string defining how the `posit` argument, if provided, should be matched against the LSD object positions. If equal to `"fixed"`, the default, only exact matching is done. `"glob"` allows using simple wildcard characters ('*' and '?') in `posit` for matching. If `posit.match="regex"` interpret `posit` as POSIX 1003.2 extended regular expression(s). See `regular expressions` for details of the different types of regular expressions. Options can be abbreviated.

Details

Selection restriction arguments can be provided as needed; when not specified, all available cases are selected.

When `posit` is supplied together with `col.names` or `instance`, the selection process is done in two steps. Firstly, the column names and instance positions set by `col.names` and `instance` are selected. Secondly, the instances defined by `posit` are selected from the first selection set.

See `select.colnames.lsd` and `select.colattrs.lsd` for examples on how to apply advanced selection options.

Value

Returns a single matrix containing all variables' time series contained in the results file.

Note

The returned matrix may be potentially very wide. See `read.single.lsd` for more polished column names. To use multiple results files simultaneously, see `read.list.lsd` and `read.3d.lsd`.

Author(s)

Marcelo C. Pereira

See Also

`list.files.lsd()` `read.single.lsd()`, `read.multi.lsd()`, `read.list.lsd()`, `read.3d.lsd()`, `read.4d.lsd()`, `select.colattrs.lsd()`, `select.colnames.lsd()`

Examples

```
# get the list of file names of example LSD results
files <- list.files.lsd( system.file( "extdata", package = "LSDinterface" ) )

# read all instances of all variables of first file,
bigTable <- read.raw.lsd( files[ 1 ] )
print( bigTable[ 1 : 5, 1 : 7 ] )

# read all instances of all variables, skipping the initial 20 time steps
# and keeping up to 30 time steps (from t = 21 up to t = 30)
all21_30 <- read.raw.lsd( files[ 2 ], skip = 20, nrows = 30 )
print( all21_30[ , 1 : 7 ] )

# read the third instances of a set of variables named '_A1p' and '_growth1'
abTable <- read.raw.lsd( files[ 1 ], col.names = c( "_A1p", "_growth1" ),
                         instance = 3 )
print( abTable[ 10 : 20, ] )

# read instances of variable '_A1p' for the second and fourth objects under
# any top-level object (use globbing)
a24 <- read.raw.lsd( files[ 1 ], col.names = "_A1p",
                     posit = c( "*_2", "*_4" ), posit.match = "glob" )
print( a24[ 1 : 10, ] )
```

---

read.single.lsd   *Read LSD variables (time series) from a LSD results file (a single instance of each variable only)*

Description

This function reads the data series associated to one instance of each selected variable from a LSD results file (.res). Just a single instance (time series of a single LSD object) is read at each call.

Usage

```
read.single.lsd( file, col.names = NULL, nrows = -1, skip = 0,
                 check.names = TRUE, instance = 1, posit = NULL,
                 posit.match = c( "fixed", "glob", "regex" ) )
```

Arguments

`file`	the name of the LSD results file which the data are to be read from. If it does not contain an absolute path, the file name is relative to the current working directory, `getwd()`. This can be a compressed file (see file) and must include the appropriated extension (usually `.res` or `.res.gz`).
`col.names`	a vector of optional names for the variables. The default is to read all variables. The names must to be in LSD/C++ format, without dots (".") in the name. Any dot (and trailing characters) will be automatically removed.

`nrows`	integer: the maximum number of time steps (rows) to read in. Negative and other invalid values are ignored. The default is to read all rows.
`skip`	integer: the number of time steps (rows) of the results file to skip before beginning to read data. The default is to read from the first time step (t = 1).
`check.names`	logical. If `TRUE` the names of the variables are checked to ensure that they are syntactically valid variable names. If necessary they are adjusted to ensure that there are no duplicates.
`instance`	integer: the instance of the variable to be read, for variables that exist in more than one object. This number is based on the relative position (column) of the variable in the results file. The default (0) is to read all instances.
`posit`	a string, a vector of strings or an integer vector describing the LSD object position of the variable(s) to select. If an integer vector, it should define the position of a SINGLE LSD object. If a string or vector of strings, each element should define one or more different LSD objects, so the returning matrix will contain variables from more than one object. By setting `posit.match`, globbing (wildcard), and regular expressions can be used to select multiple objects at once.
`posit.match`	a string defining how the `posit` argument, if provided, should be matched against the LSD object positions. If equal to `"fixed"`, the default, only exact matching is done. `"glob"` allows using simple wildcard characters ('*' and '?') in `posit` for matching. If `posit.match="regex"` interpret `posit` as POSIX 1003.2 extended regular expression(s). See `regular expressions` for details of the different types of regular expressions. Options can be abbreviated.

Details

Selection restriction arguments can be provided as needed; when not specified, all available cases are considered, but just one `instance` is considered.

When `posit` is supplied together with `col.names` or `instance`, the selection process is done in two steps. Firstly, the column names and the instance position set by `col.names` and `instance` are selected. Secondly, the instances defined by `posit` are selected from the first selection set.

See `select.colnames.lsd` and `select.colattrs.lsd` for examples on how to apply advanced selection options.

Value

Returns a matrix containing the selected variables' time series contained in the results file.

Note

This function is useful to extract time series for variables that are single instanced, like summary statistics. For multi-instanced variables, see `read.multi.lsd`. For extracting data from multiple similar files (like sensitivity analysis results), see `read.list.lsd` (multi-instanced variables) and `read.3d.lsd` (single-instanced variables).

Author(s)

Marcelo C. Pereira

See Also

list.files.lsd() read.multi.lsd(), read.list.lsd(), read.3d.lsd(), read.4d.lsd(), read.raw.lsd()

Examples

```
# get the list of file names of example LSD results
files <- list.files.lsd( system.file( "extdata", package = "LSDinterface" ) )

# load first .res file into a simple matrix (first instances only)
macroVar <- read.single.lsd( files[ 1 ] )
print( macroVar[ 10 : 20, 5 : 9 ] )

# read second instance of a set of variables named '_A1p' and '_growth1'
ag2Table <- read.single.lsd( files[ 2 ], col.names = c( "_A1p", "_growth1" ),
                             instance = 2 )
print( ag2Table[ 10 : 15, ] )

# read first instance of all variables, skipping the initial 20 time steps
# and keeping up to 30 time steps (from t = 21 up to t = 30)
var21_30 <- read.single.lsd( files[ 3 ], skip = 20, nrows = 30 )
print( var21_30[ , 1 : 7 ] )

# read third instance of all variables at the second object level
var2_3_5 <- read.single.lsd( files[ 1 ], instance = 3, posit = "*_*",
                            posit.match = "glob" )
print( var2_3_5[ 20 : 25, ] )
```

select.colattrs.lsd   *Select a subset of a LSD results matrix (by variable attributes)*

Description

This function select a subset of a LSD results matrix (as produced by read.raw.lsd) by the variable attributes, considering the LSD object position and the time span.

Usage

```
select.colattrs.lsd( dataSet, info, col.names = NULL, init.value = NA,
                     init.time = NA, end.time = NA, posit = NULL,
                     posit.match = c( "fixed", "glob", "regex" ) )
```

Arguments

dataSet	matrix produced by the invocation of read.raw.lsd, read.single.lsd, read.multi.lsd or read.list.lsd (a single matrix a time) functions.
info	data frame produced by info.details.lsd for the same results file from where dataSet was extracted.

`col.names`	a vector of optional names for the variables to select from. The default is to select from all variables.
`init.value`	initial value attributed to the variable(s) to select.
`init.time`	initial time attributed to the variable(s) to select.
`end.time`	end time attributed to the variable(s) to select.
`posit`	a string, a vector of strings or an integer vector describing the LSD object position of the variable(s) to select. If an integer vector, it should define the position of a SINGLE LSD object. If a string or vector of strings, each element should define one or more different LSD objects, so the returning matrix will contain variables from more than one object. By setting `posit.match`, globbing (wildcard), and regular expressions can be used to select multiple objects at once; in this case, all matching objects are returned.
`posit.match`	a string defining how the `posit` argument, if provided, should be matched against the LSD object positions. If equal to `"fixed"`, the default, only exact matching is done. `"glob"` allows using simple wildcard characters ('*' and '?') in `posit` for matching. If `posit.match="regex"` interpret `posit` as POSIX 1003.2 extended regular expression(s). See `regular expressions` for details of the different types of regular expressions. Options can be abbreviated.

Details

Selection restriction arguments can be provided as needed; when not specified, all available cases are selected.

When `posit` is supplied together with other attribute filters, the selection process is done in two steps. Firstly, the column names set by otter attribute filters are selected. Secondly, the instances defined by `posit` are selected from the first selection set.

See also the `read.XXXX.lsd` functions which may select just specific `posit` object instances when loading LSD results. If only a single set of instances is required, this would be more efficient than using this function.

Value

Returns a single matrix containing the selected variables' time series contained in the original data set.

Note

If only variable names selection is needed, `select.colnames.lsd` is more efficient because information pre-processing (`info.details.lsd`) is not required.

Author(s)

Marcelo C. Pereira

See Also

`list.files.lsd()` `info.details.lsd()`, `select.colnames.lsd()`

Examples

```
# get the list of file names of example LSD results
files <- list.files.lsd( system.file( "extdata", package = "LSDinterface" ) )

# read all instances of all variables of first file
bigTable <- read.raw.lsd( files[ 1 ] )

# build the info table
info <- info.details.lsd( files[ 1 ] )

# extract specific instances of a set of variables named '_A1p' and '_growth1'
abFirst2 <- select.colattrs.lsd( bigTable, info, c( "_A1p", "_growth1" ),
                                 posit = c( "1_2", "1_5" ) )
print( abFirst2[ 50 : 60, ] )

# extract instances of variable '_A1p' that start at time step t = 1
# for the second and fourth objects under any top-level object (use globbing)
a24 <- select.colattrs.lsd( bigTable, info, "_A1p", init.time = 1,
                            posit = c( "*_2", "*_4" ), posit.match = "glob" )
print( a24[ 1 : 10, ] )

# extract all second-level object instances of all variables
aSec <- select.colattrs.lsd( bigTable, info, posit = "*_*", posit.match = "glob" )
print( aSec[ 1 : 10, ] )

# extract just top-level object instances variables
aTop <- select.colattrs.lsd( bigTable, info, posit = "^[0-9]+$",
                             posit.match = "regex" )
print( aTop[ 1 : 10, ] )
```

---

select.colnames.lsd   *Select a subset of a LSD results matrix (by column/variable names)*

Description

This function select a subset of a LSD results matrix (as produced by `read.raw.lsd`) by the column (variable) names, considering only the name part of the column labels.

Usage

```
select.colnames.lsd( dataSet, col.names = NULL, instance = 0,
                     check.names = TRUE, posit = NULL,
                     posit.match = c( "fixed", "glob", "regex" ) )
```

Arguments

`dataSet`	matrix produced by the invocation of `read.raw.lsd`, `read.single.lsd`, `read.multi.lsd` or `read.list.lsd` (a single matrix a time) functions.

`col.names`	a vector of optional names for the variables. The default is to read all variables. The names must to be in LSD/C++ format, without dots (".") in the name. Any dot (and trailing characters) will be automatically removed.
`instance`	integer: the instance of the variable to be read, for variables that exist in more than one object. This number is based on the relative position (column) of the variable in the results file. The default (0) is to read all instances.
`check.names`	logical. If `TRUE` the names of the variables are checked to ensure that they are syntactically valid variable names. If necessary they are adjusted to ensure that there are no duplicates.
`posit`	a string, a vector of strings or an integer vector describing the LSD object position of the variable(s) to select. If an integer vector, it should define the position of a SINGLE LSD object. If a string or vector of strings, each element should define one or more different LSD objects, so the returning matrix will contain variables from more than one object. By setting `posit.match`, globbing (wildcard), and regular expressions can be used to select multiple objects at once; in this case, all matching objects are returned. This option only operates if `dataSet` was generated by `read.raw.lsd` WITHOUT argument `clean.names = TRUE`.
`posit.match`	a string defining how the `posit` argument, if provided, should be matched against the LSD object positions. If equal to `"fixed"`, the default, only exact matching is done. `"glob"` allows using simple wildcard characters ('*' and '?') in `posit` for matching. If `posit.match="regex"` interpret `posit` as POSIX 1003.2 extended regular expression(s). See `regular expressions` for details of the different types of regular expressions. Options can be abbreviated.

Details

Selection restriction arguments can be provided as needed; when not specified, all available cases are selected.

The selection of specific `posit` object positions require full detail on `dataSet` column names, as produced by `read.raw.lsd` and `clean.names = TRUE` is NOT used. Other `read.XXXX.lsd` functions do NOT produce the required detail on the data matrices to do object position selection. If such datasets are used to feed this function and `posit` is set, the return value will be NULL. In this case, consider using `select.colattrs.lsd`, or specifying `posit` when calling `read.XXXX.lsd` functions.

When `posit` is supplied together with other attribute filters, the selection process is done in two steps. Firstly, the column names set by otter attribute filters are selected. Secondly, the instances defined by `posit` are selected from the first selection set.

See also the `read.XXXX.lsd` functions which may select just specific `col.names` columns, `instance` instances, or `posit` positions when loading LSD results. If only a single set of columns/instance/positions is required, this may be more efficient than using this function.

Value

Returns a single matrix containing the selected variables' time series contained in the original data set.

Note

The variable/column names must be valid R or LSD column names.

Author(s)

Marcelo C. Pereira

See Also

list.files.lsd(), select.colattrs.lsd(), read.raw.lsd()

Examples

```
# get the list of file names of example LSD results
files <- list.files.lsd( system.file( "extdata", package = "LSDinterface" ) )

# read all instances of all variables in first file
bigTable <- read.raw.lsd( files[ 1 ] )
print( bigTable[ 1 : 10, 1 : 7 ] )

# extract all instances of a set of variables named '_A1p' and '_growth1'
abTable <- select.colnames.lsd( bigTable, c( "_A1p", "_growth1" ) )
print( abTable[ 11 : 15, ] )

# extract specific instances of a set of variables named '_A1p' and '_growth1'
abFirst2 <- select.colnames.lsd( bigTable, c( "_A1p", "_growth1" ),
                                 posit = c( "1_2", "1_5" ) )
print( abFirst2[ 50 : 60, ] )

# extract all second-level object instances of all variables
aSec <- select.colnames.lsd( bigTable, posit = "*_*", posit.match = "glob" )
print( aSec[ 1 : 10, ] )

# extract just top-level object instances variables
aTop <- select.colnames.lsd( bigTable, posit = "^[0-9]+$", posit.match = "regex" )
print( aTop[ 1 : 10, ] )
```

Index