Package 'roll'

May 1, 2017

Type Package

Title Rolling Statistics

Version 1.0.7

Date 2017-05-01

Author Jason Foster

Maintainer Jason Foster <jason.j.foster@gmail.com>

Description Parallel functions for computing rolling statistics of time-series data.

License GPL (>= 2)

URL https://github.com/jjf234/roll

BugReports https://github.com/jjf234/roll/issues

Imports Rcpp, RcppParallel

LinkingTo Rcpp, RcppArmadillo, RcppParallel

SystemRequirements GNU make, C++11

RoxygenNote 6.0.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2017-05-01 10:58:07 UTC

R topics documented:

`roll_cor` *Rolling Correlation Matrices*

Description

A parallel function for computing rolling correlation matrices of time-series data.

Usage

```
roll_cor(data, width, weights = rep(1, width), center = TRUE,
  scale = TRUE, min_obs = width, complete_obs = TRUE,
  na_restore = FALSE, parallel_for = c("rows", "cols"))
```

Arguments

`data`	matrix or xts object. Rows are observations and columns are variables.
`width`	integer. Window size.
`weights`	vector. Weights for each observation within a window.
`center`	logical. If `TRUE` then the weighted mean of each variable is used, if `FALSE` then zero is used.
`scale`	logical. If `TRUE` then the weighted standard deviation of each variable is used, if `FALSE` then no scaling is done.
`min_obs`	integer. Minimum number of observations required to have a value within a window, otherwise result is `NA`.
`complete_obs`	logical. If `TRUE` then rows containing any missing values are removed, if `FALSE` then pairwise is used.
`na_restore`	logical. Should missing values be restored?
`parallel_for`	character. Executes a "for" loop in which iterations run in parallel by `rows` or `cols`.

Details

The numerical calculations use RcppParallel to parallelize rolling correlation matrices of time-series data. RcppParallel provides a complete toolkit for creating safe, portable, high-performance parallel algorithms, built on top of the Intel Threading Building Blocks (TBB) and TinyThread libraries.

By default, all the available cores on a machine are used for parallel algorithms. If users are either already taking advantage of parallelism or instead want to use a fixed number or proportion of threads, then set the number of threads in the RcppParallel package with the `setThreadOptions` function.

Value

A cube with each slice the rolling correlation matrix.

Examples

```
n_vars <- 10
n_obs <- 1000
data <- matrix(rnorm(n_obs * n_vars), nrow = n_obs, ncol = n_vars)

# Rolling correlation matrices
result <- roll_cor(data, 252)

# Rolling correlation matrices with exponential decay
weights <- 0.9 ^ (251:0)
result <- roll_cor(data, 252, weights)
```

---

roll_cov *Rolling Covariance Matrices*

Description

A parallel function for computing rolling covariance matrices of time-series data.

Usage

```
roll_cov(data, width, weights = rep(1, width), center = TRUE,
  scale = FALSE, min_obs = width, complete_obs = TRUE,
  na_restore = FALSE, parallel_for = c("rows", "cols"))
```

Arguments

data	matrix or xts object. Rows are observations and columns are variables.
width	integer. Window size.
weights	vector. Weights for each observation within a window.
center	logical. If TRUE then the weighted mean of each variable is used, if FALSE then zero is used.
scale	logical. If TRUE then the weighted standard deviation of each variable is used, if FALSE then no scaling is done.
min_obs	integer. Minimum number of observations required to have a value within a window, otherwise result is NA.
complete_obs	logical. If TRUE then rows containing any missing values are removed, if FALSE then pairwise is used.
na_restore	logical. Should missing values be restored?
parallel_for	character. Executes a "for" loop in which iterations run in parallel by rows or cols.

Details

The numerical calculations use RcppParallel to parallelize rolling covariance matrices of time-series data. RcppParallel provides a complete toolkit for creating safe, portable, high-performance parallel algorithms, built on top of the Intel Threading Building Blocks (TBB) and TinyThread libraries.

By default, all the available cores on a machine are used for parallel algorithms. If users are either already taking advantage of parallelism or instead want to use a fixed number or proportion of threads, then set the number of threads in the RcppParallel package with the `setThreadOptions` function.

Value

A cube with each slice the rolling covariance matrix.

Examples

```
n_vars <- 10
n_obs <- 1000
data <- matrix(rnorm(n_obs * n_vars), nrow = n_obs, ncol = n_vars)

# Rolling covariance matrices
result <- roll_cov(data, 252)

# Rolling covariance matrices with exponential decay
weights <- 0.9 ^ (251:0)
result <- roll_cov(data, 252, weights)
```

`roll_eigen` *Rolling Eigenvalues and Eigenvectors*

Description

A parallel function for computing rolling eigenvalues and eigenvectors of time-series data.

Usage

```
roll_eigen(data, width, weights = rep(1, width), center = TRUE,
  scale = FALSE, min_obs = width, complete_obs = TRUE,
  na_restore = FALSE, parallel_for = c("rows", "cols"))
```

Arguments

`data`	matrix or xts object. Rows are observations and columns are variables.
`width`	integer. Window size.
`weights`	vector. Weights for each observation within a window.
`center`	logical. If `TRUE` then the weighted mean of each variable is used, if `FALSE` then zero is used.

scale	logical. If TRUE then the weighted standard deviation of each variable is used, if FALSE then no scaling is done.
min_obs	integer. Minimum number of observations required to have a value within a window, otherwise result is NA.
complete_obs	logical. If TRUE then rows containing any missing values are removed, if FALSE then pairwise is used.
na_restore	logical. Should missing values be restored?
parallel_for	character. Executes a "for" loop in which iterations run in parallel by rows or cols.

Details

The numerical calculations use RcppParallel to parallelize rolling eigenvalues and eigenvectors of time-series data. RcppParallel provides a complete toolkit for creating safe, portable, high-performance parallel algorithms, built on top of the Intel Threading Building Blocks (TBB) and TinyThread libraries.

By default, all the available cores on a machine are used for parallel algorithms. If users are either already taking advantage of parallelism or instead want to use a fixed number or proportion of threads, then set the number of threads in the RcppParallel package with the setThreadOptions function.

Value

A list containing the following components:

values	An object of the same class and dimension as data with the rolling eigenvalues.
vectors	A cube with each slice the rolling eigenvectors.

Examples

```
n_vars <- 10
n_obs <- 1000
data <- matrix(rnorm(n_obs * n_vars), nrow = n_obs, ncol = n_vars)

# Rolling eigenvalues and eigenvectors
result <- roll_eigen(data, 252)

# Rolling eigenvalues and eigenvectors with exponential decay
weights <- 0.9 ^ (251:0)
result <- roll_eigen(data, 252, weights)
```

roll_lm *Rolling Linear Models*

Description

A parallel function for computing rolling linear models of time-series data.

Usage

```
roll_lm(x, y, width, weights = rep(1, width), intercept = TRUE,
  center = FALSE, center_x = center, center_y = center, scale = FALSE,
  scale_x = scale, scale_y = scale, min_obs = width,
  complete_obs = TRUE, na_restore = FALSE, parallel_for = c("rows",
  "cols"))
```

Arguments

x	matrix or xts object. Rows are observations and columns are the independent variables.
y	matrix or xts object. Rows are observations and columns are the dependent variables.
width	integer. Window size.
weights	vector. Weights for each observation within a window.
intercept	logical. Either TRUE to include or FALSE to remove the intercept.
center	logical. center = z is shorthand for center_x = z and center_y = z, where z is either TRUE or FALSE.
center_x	logical. If TRUE then the weighted mean of each x variable is used, if FALSE then zero is used.
center_y	logical. Analogous to center_x.
scale	logical. scale = z is shorthand for scale_x = z and scale_y = z, where z is either TRUE or FALSE.
scale_x	logical. If TRUE then the weighted standard deviation of each x variable is used, if FALSE then no scaling is done.
scale_y	logical. Analogous to scale_x.
min_obs	integer. Minimum number of observations required to have a value within a window, otherwise result is NA.
complete_obs	logical. If TRUE then rows containing any missing values are removed, if FALSE then pairwise is used.
na_restore	logical. Should missing values be restored?
parallel_for	character. Executes a "for" loop in which iterations run in parallel by rows or cols.

Details

The numerical calculations use RcppParallel to parallelize rolling linear models of time-series data. RcppParallel provides a complete toolkit for creating safe, portable, high-performance parallel algorithms, built on top of the Intel Threading Building Blocks (TBB) and TinyThread libraries.

By default, all the available cores on a machine are used for parallel algorithms. If users are either already taking advantage of parallelism or instead want to use a fixed number or proportion of threads, then set the number of threads in the RcppParallel package with the `setThreadOptions` function.

Value

A list containing the following components:

`coefficients`	A list of objects with the rolling coefficients for each `y`. An object is the same class and dimension (with an added column for the intercept) as `x`.
`r.squared`	A list of objects with the rolling r-squareds for each `y`. An object is the same class as `x`.

Examples

```
n_vars <- 10
n_obs <- 1000
x <- matrix(rnorm(n_obs * n_vars), nrow = n_obs, ncol = n_vars)
y <- matrix(rnorm(n_obs), nrow = n_obs, ncol = 1)

# Rolling regressions
result <- roll_lm(x, y, 252)

# Rolling regressions with exponential decay
weights <- 0.9 ^ (251:0)
result <- roll_lm(x, y, 252, weights)
```

---

roll_mean *Rolling Means*

Description

A parallel function for computing rolling means of time-series data.

Usage

```
roll_mean(data, width, weights = rep(1, width), min_obs = width,
  complete_obs = FALSE, na_restore = FALSE, parallel_for = c("rows",
  "cols"))
```

Arguments

data	matrix or xts object. Rows are observations and columns are variables.
width	integer. Window size.
weights	vector. Weights for each observation within a window.
min_obs	integer. Minimum number of observations required to have a value within a window, otherwise result is NA.
complete_obs	logical. If TRUE then rows containing any missing values are removed, if FALSE then each value is used.
na_restore	logical. Should missing values be restored?
parallel_for	character. Executes a "for" loop in which iterations run in parallel by rows or cols.

Details

The numerical calculations use RcppParallel to parallelize rolling means of time-series data. RcppParallel provides a complete toolkit for creating safe, portable, high-performance parallel algorithms, built on top of the Intel Threading Building Blocks (TBB) and TinyThread libraries.

By default, all the available cores on a machine are used for parallel algorithms. If users are either already taking advantage of parallelism or instead want to use a fixed number or proportion of threads, then set the number of threads in the RcppParallel package with the setThreadOptions function.

Value

An object of the same class and dimension as data with the rolling means.

Examples

```
n_vars <- 10
n_obs <- 1000
data <- matrix(rnorm(n_obs * n_vars), nrow = n_obs, ncol = n_vars)

# Rolling means
result <- roll_mean(data, 252)

# Rolling means with exponential decay
weights <- 0.9 ^ (251:0)
result <- roll_mean(data, 252, weights)
```

roll_pcr *Rolling Principal Component Regressions*

Description

A parallel function for computing rolling principal component regressions of time-series data.

Usage

```
roll_pcr(x, y, width, comps = 1:ncol(x), weights = rep(1, width),
  intercept = TRUE, center = FALSE, center_x = center,
  center_y = center, scale = FALSE, scale_x = scale, scale_y = scale,
  min_obs = width, complete_obs = TRUE, na_restore = FALSE,
  parallel_for = c("rows", "cols"))
```

Arguments

x	matrix or xts object. Rows are observations and columns are the independent variables.
y	matrix or xts object. Rows are observations and columns are the dependent variables.
width	integer. Window size.
comps	integer vector. Select a subset of principal components.
weights	vector. Weights for each observation within a window.
intercept	logical. Either TRUE to include or FALSE to remove the intercept.
center	logical. center = z is shorthand for center_x = z and center_y = z, where z is either TRUE or FALSE.
center_x	logical. If TRUE then the weighted mean of each x variable is used, if FALSE then zero is used.
center_y	logical. Analogous to center_x.
scale	logical. scale = z is shorthand for scale_x = z and scale_y = z, where z is either TRUE or FALSE.
scale_x	logical. If TRUE then the weighted standard deviation of each x variable is used, if FALSE then no scaling is done.
scale_y	logical. Analogous to scale_x.
min_obs	integer. Minimum number of observations required to have a value within a window, otherwise result is NA.
complete_obs	logical. If TRUE then rows containing any missing values are removed, if FALSE then pairwise is used.
na_restore	logical. Should missing values be restored?
parallel_for	character. Executes a "for" loop in which iterations run in parallel by rows or cols.

Details

The numerical calculations use RcppParallel to parallelize rolling principal component regressions of time-series data. RcppParallel provides a complete toolkit for creating safe, portable, high-performance parallel algorithms, built on top of the Intel Threading Building Blocks (TBB) and TinyThread libraries.

By default, all the available cores on a machine are used for parallel algorithms. If users are either already taking advantage of parallelism or instead want to use a fixed number or proportion of threads, then set the number of threads in the RcppParallel package with the `setThreadOptions` function.

Value

A list containing the following components:

`coefficients`	A list of objects with the rolling coefficients for each `y`. An object is the same class and dimension (with an added column for the intercept) as `x`.
`r.squared`	A list of objects with the rolling r-squareds for each `y`. An object is the same class as `x`.

Examples

```
n_vars <- 10
n_obs <- 1000
x <- matrix(rnorm(n_obs * n_vars), nrow = n_obs, ncol = n_vars)
y <- matrix(rnorm(n_obs), nrow = n_obs, ncol = 1)

# Rolling principal component regressions
result <- roll_pcr(x, y, 252, comps = 1)

# Rolling principal component regressions with exponential decay
weights <- 0.9 ^ (251:0)
result <- roll_pcr(x, y, 252, comps = 1, weights)
```

---

roll_prod *Rolling Products*

Description

A parallel function for computing rolling products of time-series data.

Usage

```
roll_prod(data, width, weights = rep(1, width), min_obs = width,
  complete_obs = FALSE, na_restore = FALSE, parallel_for = c("rows",
  "cols"))
```

Arguments

data	matrix or xts object. Rows are observations and columns are variables.
width	integer. Window size.
weights	vector. Weights for each observation within a window.
min_obs	integer. Minimum number of observations required to have a value within a window, otherwise result is NA.
complete_obs	logical. If TRUE then rows containing any missing values are removed, if FALSE then each value is used.
na_restore	logical. Should missing values be restored?
parallel_for	character. Executes a "for" loop in which iterations run in parallel by rows or cols.

Details

The numerical calculations use RcppParallel to parallelize rolling products of time-series data. RcppParallel provides a complete toolkit for creating safe, portable, high-performance parallel algorithms, built on top of the Intel Threading Building Blocks (TBB) and TinyThread libraries.

By default, all the available cores on a machine are used for parallel algorithms. If users are either already taking advantage of parallelism or instead want to use a fixed number or proportion of threads, then set the number of threads in the RcppParallel package with the setThreadOptions function.

Value

An object of the same class and dimension as data with the rolling products.

Examples

```
n_vars <- 10
n_obs <- 1000
data <- matrix(rnorm(n_obs * n_vars), nrow = n_obs, ncol = n_vars)

# Rolling products
result <- roll_prod(data, 252)

# Rolling products with exponential decay
weights <- 0.9 ^ (251:0)
result <- roll_prod(data, 252, weights)
```

`roll_scale` *Rolling Scaling and Centering*

Description

A parallel function for computing rolling scaling and centering of time-series data.

Usage

```
roll_scale(data, width, weights = rep(1, width), center = TRUE,
  scale = TRUE, min_obs = width, complete_obs = FALSE,
  na_restore = FALSE, parallel_for = c("rows", "cols"))
```

Arguments

`data`	matrix or xts object. Rows are observations and columns are variables.
`width`	integer. Window size.
`weights`	vector. Weights for each observation within a window.
`center`	logical. If `TRUE` then the weighted mean of each variable is used, if `FALSE` then zero is used.
`scale`	logical. If `TRUE` then the weighted standard deviation of each variable is used, if `FALSE` then no scaling is done.
`min_obs`	integer. Minimum number of observations required to have a value within a window, otherwise result is `NA`.
`complete_obs`	logical. If `TRUE` then rows containing any missing values are removed, if `FALSE` then each value is used.
`na_restore`	logical. Should missing values be restored?
`parallel_for`	character. Executes a "for" loop in which iterations run in parallel by `rows` or `cols`.

Details

The numerical calculations use RcppParallel to parallelize rolling scaling and centering of time-series data. RcppParallel provides a complete toolkit for creating safe, portable, high-performance parallel algorithms, built on top of the Intel Threading Building Blocks (TBB) and TinyThread libraries.

By default, all the available cores on a machine are used for parallel algorithms. If users are either already taking advantage of parallelism or instead want to use a fixed number or proportion of threads, then set the number of threads in the RcppParallel package with the `setThreadOptions` function.

Value

An object of the same class and dimension as `data` with the rolling scaling and centering.

Examples

```
n_vars <- 10
n_obs <- 1000
data <- matrix(rnorm(n_obs * n_vars), nrow = n_obs, ncol = n_vars)

# Rolling z-scores
result <- roll_scale(data, 252)

# Rolling z-scores with exponential decay
weights <- 0.9 ^ (251:0)
result <- roll_scale(data, 252, weights)
```

---

roll_sd *Rolling Standard Deviations*

Description

A parallel function for computing rolling standard deviations of time-series data.

Usage

```
roll_sd(data, width, weights = rep(1, width), center = TRUE,
  min_obs = width, complete_obs = FALSE, na_restore = FALSE,
  parallel_for = c("rows", "cols"))
```

Arguments

data	matrix or xts object. Rows are observations and columns are variables.
width	integer. Window size.
weights	vector. Weights for each observation within a window.
center	logical. If TRUE then the weighted mean of each variable is used, if FALSE then zero is used.
min_obs	integer. Minimum number of observations required to have a value within a window, otherwise result is NA.
complete_obs	logical. If TRUE then rows containing any missing values are removed, if FALSE then each value is used.
na_restore	logical. Should missing values be restored?
parallel_for	character. Executes a "for" loop in which iterations run in parallel by rows or cols.

Details

The numerical calculations use RcppParallel to parallelize rolling standard deviations of time-series data. RcppParallel provides a complete toolkit for creating safe, portable, high-performance parallel algorithms, built on top of the Intel Threading Building Blocks (TBB) and TinyThread libraries.

By default, all the available cores on a machine are used for parallel algorithms. If users are either already taking advantage of parallelism or instead want to use a fixed number or proportion of threads, then set the number of threads in the RcppParallel package with the `setThreadOptions` function.

Value

An object of the same class and dimension as `data` with the rolling standard deviations.

Examples

```
n_vars <- 10
n_obs <- 1000
data <- matrix(rnorm(n_obs * n_vars), nrow = n_obs, ncol = n_vars)

# Rolling standard deviations
result <- roll_sd(data, 252)

# Rolling standard deviations with exponential decay
weights <- 0.9 ^ (251:0)
result <- roll_sd(data, 252, weights)
```

---

roll_sum *Rolling Sums*

Description

A parallel function for computing rolling sums of time-series data.

Usage

```
roll_sum(data, width, weights = rep(1, width), min_obs = width,
  complete_obs = FALSE, na_restore = FALSE, parallel_for = c("rows",
  "cols"))
```

Arguments

`data`	matrix or xts object. Rows are observations and columns are variables.
`width`	integer. Window size.
`weights`	vector. Weights for each observation within a window.
`min_obs`	integer. Minimum number of observations required to have a value within a window, otherwise result is `NA`.

complete_obs	logical. If TRUE then rows containing any missing values are removed, if FALSE then each value is used.
na_restore	logical. Should missing values be restored?
parallel_for	character. Executes a "for" loop in which iterations run in parallel by rows or cols.

Details

The numerical calculations use RcppParallel to parallelize rolling sums of time-series data. RcppParallel provides a complete toolkit for creating safe, portable, high-performance parallel algorithms, built on top of the Intel Threading Building Blocks (TBB) and TinyThread libraries.

By default, all the available cores on a machine are used for parallel algorithms. If users are either already taking advantage of parallelism or instead want to use a fixed number or proportion of threads, then set the number of threads in the RcppParallel package with the setThreadOptions function.

Value

An object of the same class and dimension as data with the rolling sums.

Examples

```
n_vars <- 10
n_obs <- 1000
data <- matrix(rnorm(n_obs * n_vars), nrow = n_obs, ncol = n_vars)

# Rolling sums
result <- roll_sum(data, 252)

# Rolling sums with exponential decay
weights <- 0.9 ^ (251:0)
result <- roll_sum(data, 252, weights)
```

---

roll_var *Rolling Variances*

Description

A parallel function for computing rolling variances of time-series data.

Usage

```
roll_var(data, width, weights = rep(1, width), center = TRUE,
  min_obs = width, complete_obs = FALSE, na_restore = FALSE,
  parallel_for = c("rows", "cols"))
```

Arguments

data	matrix or xts object. Rows are observations and columns are variables.
width	integer. Window size.
weights	vector. Weights for each observation within a window.
center	logical. If TRUE then the weighted mean of each variable is used, if FALSE then zero is used.
min_obs	integer. Minimum number of observations required to have a value within a window, otherwise result is NA.
complete_obs	logical. If TRUE then rows containing any missing values are removed, if FALSE then each value is used.
na_restore	logical. Should missing values be restored?
parallel_for	character. Executes a "for" loop in which iterations run in parallel by rows or cols.

Details

The numerical calculations use RcppParallel to parallelize rolling variances of time-series data. RcppParallel provides a complete toolkit for creating safe, portable, high-performance parallel algorithms, built on top of the Intel Threading Building Blocks (TBB) and TinyThread libraries.

By default, all the available cores on a machine are used for parallel algorithms. If users are either already taking advantage of parallelism or instead want to use a fixed number or proportion of threads, then set the number of threads in the RcppParallel package with the setThreadOptions function.

Value

An object of the same class and dimension as data with the rolling variances.

Examples

```
n_vars <- 10
n_obs <- 1000
data <- matrix(rnorm(n_obs * n_vars), nrow = n_obs, ncol = n_vars)

# Rolling variances
result <- roll_var(data, 252)

# Rolling variances with exponential decay
weights <- 0.9 ^ (251:0)
result <- roll_var(data, 252, weights)
```

roll_vif *Rolling Variance Inflation Factors*

Description

A parallel function for computing rolling variance inflation factors of time-series data.

Usage

```
roll_vif(data, width, weights = rep(1, width), center = FALSE,
  scale = FALSE, min_obs = width, complete_obs = TRUE,
  na_restore = FALSE, parallel_for = c("rows", "cols"))
```

Arguments

data	matrix or xts object. Rows are observations and columns are variables.
width	integer. Window size.
weights	vector. Weights for each observation within a window.
center	logical. If TRUE then the weighted mean of each variable is used, if FALSE then zero is used.
scale	logical. If TRUE then the weighted standard deviation of each variable is used, if FALSE then no scaling is done.
min_obs	integer. Minimum number of observations required to have a value within a window, otherwise result is NA.
complete_obs	logical. If TRUE then rows containing any missing values are removed, if FALSE then pairwise is used.
na_restore	logical. Should missing values be restored?
parallel_for	character. Executes a "for" loop in which iterations run in parallel by rows or cols.

Details

The numerical calculations use RcppParallel to parallelize rolling variance inflation factors of time-series data. RcppParallel provides a complete toolkit for creating safe, portable, high-performance parallel algorithms, built on top of the Intel Threading Building Blocks (TBB) and TinyThread libraries.

By default, all the available cores on a machine are used for parallel algorithms. If users are either already taking advantage of parallelism or instead want to use a fixed number or proportion of threads, then set the number of threads in the RcppParallel package with the setThreadOptions function.

Value

An object of the same class and dimension as data with the rolling variance inflation factors.

Examples

```
n_vars <- 10
n_obs <- 1000
data <- matrix(rnorm(n_obs * n_vars), nrow = n_obs, ncol = n_vars)

# Rolling variance inflation factors
result <- roll_vif(data, 252)

# Rolling variance inflation factors with exponential decay
weights <- 0.9 ^ (251:0)
result <- roll_vif(data, 252, weights)
```

Index