Package 'RcppRoll'

July 2, 2014

Type Package

Title Fast rolling functions through Rcpp and RcppArmadillo

Version 0.1.0

Date 2013-01-10

Author Kevin Ushey

Maintainer Kevin Ushey <kevinushey@gmail.com>

Description RcppRoll supplies fast functions for 'roll'ing over
vectors and matrices, e.g. rolling means, medians and
variances. It also provides the utility functions 'rollit' and
'rollit_raw' as an interface for generating your own C++ backed rolling functions.

License GPL (>= 2)

Depends R (>= 2.15.1), Rcpp (>= 0.10.2), RcppArmadillo (>= 0.3.6.1)

Suggests zoo, microbenchmark

LinkingTo Rcpp, RcppArmadillo

Collate 'package_description.R' 'RcppExports.R' 'roll_mean.R'
'roll_sum.R' 'roll_var.R' 'roll_sd.R' 'roll_prod.R'
'roll_median.R' 'roll_max.R' 'roll_min.R' 'get_rollit_source.R'
'rollit_example.R' 'rollit_raw.R' 'rollit.R'

Repository CRAN

Date/Publication 2013-01-15 07:36:24

NeedsCompilation yes

R topics documented:

`get_rollit_source` *Get and Edit Source File Associated with User-Generated 'rollit' Function*

Description

This function returns and opens the source file associated with a particular 'rollit' function for debugging. We use R's `file.edit` interface.

Usage

```
get_rollit_source(fun, edit = TRUE, RStudio = FALSE, ...)
```

Arguments

`fun`	The generated 'rollit' function.
`edit`	boolean; open the C++ source file in a text editor?
`RStudio`	boolean; open the C++ source file in RStudio?
`...`	Optional arguments passed to `file.edit`.

RcppRoll *RcppRoll*

Description

This package implements a number of 'roll'-ing functions for R vectors and matrices.

Details

Currently, the exported functions are:

- roll_max
- roll_mean
- roll_median
- roll_min
- roll_prod
- roll_sd
- roll_sum
- roll_var

See Also

rollit for 'roll'-ing your own custom functions.

rollit *Generate your own Weighted C++ Roll Function*

Description

Using this interface, you can define a function that you would like to be called on each sub-vector you are rolling over. The generated code is compiled and exposed via sourceCpp.

Usage

```
rollit(fun = "x", vector = FALSE, const_vars = NULL,
  combine = "+", final_trans = NULL, includes = NULL,
  depends = NULL, inline = TRUE, name = NULL, ...)
```

Arguments

fun	A character string defining the function call. The function must be in terms of x. The function will be applied individually to each element being 'roll'ed over, unless vector is TRUE.
vector	boolean; if TRUE, then fun is a scalar-valued function of a vector, rather than a function to apply to each element individually.
const_vars	Constant variables that will live within the sourced C++ function. Format is a named list; e.g, you could pass list(e=exp(1)) to have e as a constant variable available in the function being called. Note that the variable N is fixed to be the number of non-zero weights passed, to facilitate the use of 0 weights in terms of skipping elements.
combine	character; typically one of "+", "-", "*", "/", but any operator usable as a C++ compound assignment operator is accepted. This specifies how each element should be combined in the rolling function. Only used when vector is FALSE.
final_trans	A final transformation to perform after either 'rolling' over each element in the vector x (with vector=FALSE), or after applying a scalar-valued function of a vector (with vector=TRUE). Must be in terms of x.
includes	Other C++ libraries to include. For example, to include boost/math.hpp, you would pass c("<boost/math.hpp>").
depends	Other libraries to link to. Linking is done through Rcpp attributes.
inline	boolean; mark the function generated as inline? This may or may not increase execution speed.
name	string; a name to internally assign to your generated C++ functions.
...	Additional arguments passed to sourceCpp.

Details

By default, we include <Rcpp.h> in each file; however, you can include your own libraries with the includes call.

Value

A wrapper R function to compiled C++ files, as generated through sourceCpp. See rollit_example for more information on the functions generated by rollit.

Note

All functions generated use Rcpp's NumericVector and NumericMatrix to interface with R vectors and matrices. Elements within these vectors are translated as doubles so any function that receives a double will work fine.

If you want to just write your own C++ function to wrap into a 'rolling' interface, see rollit_raw.

See Also

rollit_example for an example of the function signature for functions generated with rollit, sourceCpp for information on how Rcpp compiles your functions, get_rollit_source for inspection of the generated C++ code, and rollit_raw for wrapping in your own C++ code.

Examples

```
## Not run:
x <- matrix(1:16, nrow=4)

## the squared rolling sum -- we square the sum of our rolled results
rolling_sqsum <- rollit( final_trans="x*x" )

rolling_sqsum( x, 4 )
rolling_sqsum( x, 4, by.column=FALSE )
cbind( as.vector(rolling_sqsum(x, 4)), apply(x, 2, function(x) { sum(x)^2 } ) )

## implement your own variance function
## we can use the sugar function 'mean' to get
## the mean of x

const_vars <- list(m = "mean(x)")
var_fun <- "( (x-m) * (x-m) )/(N-1)"
rolling_var <- rollit( var_fun, const_vars=const_vars )

x <- c(1, 5, 10, 15)
cbind( rolling_var(x, 2), roll_var(x, 2) )

## use a function from cmath

rolling_log10 <- rollit( "log10(x)" )
rolling_log10( 10^(1:5), 2 )

## rolling product
rolling_prod <- rollit( combine="*" )
rolling_prod( 1:10, 2 )

## using weights to specify something like a 'by' argument
rolling_prod( 1:10, 3, weights=c(1,0,1) )

## a benchmark

if( require("microbenchmark") && require("zoo") ) {
  x <- rnorm(1E4)
  microbenchmark(
    rolling_var(x, 100),
    roll_var(x, 100),
    rollapply(x, 100, var),
    times=10
    )
  }
## End(Not run)
```

`rollit_example` *'rollit' Output – Example Function*

Description

This presents the function signature for the output of `rollit`.

Usage

```
rollit_example(x, n, by.column, weights,
  normalize = FALSE)
```

Arguments

`x`	A vector/matrix of numeric type.
`n`	integer; the window / subset size to roll over.
`by.column`	boolean; if `TRUE` we loop over columns, otherwise we loop over rows.
`weights`	a vector of length `n`; specify the weighting to assign to each element in the window.
`normalize`	boolean; if `TRUE` we normalize the weights to sum to 1.

Value

This function does not return anything; it merely exists as a skeleton to provide documentation for your own `rollit` generated functions.

Note

Elements in `weights` equal to 0 are skipped, so that e.g. setting `weights = c(1,0,1,0,1)` would skip every 2nd and 4th element in each length-5 window.

See Also

`rollit`

rollit_raw *Generate your own Weighted C++ Roll Function*

Description

Using this, you can write and wrap in your own C++ function.

Usage

```
rollit_raw(fun, depends = NULL, includes = NULL,
  inline = TRUE, name = NULL, additional = NULL, ...)
```

Arguments

fun	A character string defining the function call. See examples for usage.
includes	Other C++ libraries to include. For example, to include boost/math.hpp, you would pass c("<boost/math.hpp>").
depends	Other libraries to link to. Linking is done through Rcpp attributes.
inline	boolean; mark this function as inline? This may or may not increase execution speed.
name	string; a name to internally assign to your generated C++ functions.
additional	Other C++ code you want to include; e.g. helper functions. This code will be inserted as-is above the code in fun.
...	Optional arguments passed to sourceCpp.

Details

The signature of fun is fixed as:

```
double <name>( NumericVector& x, NumericVector&  weights, const int& n, const int& N, const int& ind)
```

where

- X_SUB is a #define macro that expands to the sub-vector being rolled over,
- X(i) is a #define macro that expands to the current element of X_SUB in a loop being rolled over,
- x is a reference to the **entire** vector (not just the sub-vector being rolled over),
- weights are the weights,
- n is the window size,
- N is the number of non-zero weights passed,
- ind is the current position along vector x.

Because the variables are being passed by reference, you should **not** modify them, unless you're prepared for strange behavior. See examples for a potential usage.

Examples

```
## Not run:
## implement a weighted rolling 'sum of squares'
fun <- "
double out = 0;
const double m = mean( X_SUB );
for( int i=0; i < n; i++ ) {
  out += weights[i] * ( (X(i)-m) * (X(i)-m) ) / (N-1);
  }
return out;
"

rolling_var <- rollit_raw( fun )
x <- 1:5
rolling_var( x, 5 ) == var(x)

## a (slow-ish) implementation of rolling kurtosis
fun <- "
double numerator = 0;
double denominator = 0;
const double m = mean( X_SUB );
for( int i=0; i < n; i++ ) {
  double tmp = ( X(i) - m ) * ( X(i) - m );
  numerator += tmp * tmp;
  denominator += tmp;
}
return N * numerator / ( denominator * denominator );
"

rolling_kurt <- rollit_raw( fun )
x <- rnorm(100)
rolling_kurt(x, 20)

## End(Not run)
```

`roll_max` *Rolling Max*

Description

This function implements a rolling max with C++/Rcpp.

Usage

```
roll_max(x, n, by.column = TRUE, weights = rep(1, n),
  normalize = FALSE)
```

Arguments

x	An R object of form: numeric vector, numeric matrix.
n	integer; the window / subset size to roll over.
by.column	boolean; if TRUE we loop over columns, otherwise we loop over rows.
weights	A numeric vector of weights of same length as n.
normalize	boolean; if TRUE, we normalize the weight of vectors supplied.

Note

Elements in weights equal to 0 are skipped.

---

roll_mean *Rolling Mean*

Description

This function implements a rolling mean with C++/Rcpp.

Usage

```
roll_mean(x, n, by.column = TRUE, weights = rep(1, n),
  normalize = FALSE)
```

Arguments

x	An R object of form: numeric vector, numeric matrix.
n	integer; the window / subset size to roll over.
by.column	boolean; if TRUE we loop over columns, otherwise we loop over rows.
weights	A numeric vector of weights of same length as n.
normalize	boolean; if TRUE, we normalize the weight of vectors supplied.

Note

Elements in weights equal to 0 are skipped.

roll_median *Rolling Median*

Description

This function implements a rolling median with C++/Rcpp.

Usage

```
roll_median(x, n, by.column = TRUE)
```

Arguments

x	An R object of form: numeric vector, numeric matrix.
n	integer; the window / subset size to roll over.
by.column	boolean; if TRUE we loop over columns, otherwise we loop over rows.

roll_min *Rolling Min*

Description

This function implements a rolling min with C++/Rcpp.

Usage

```
roll_min(x, n, by.column = TRUE, weights = rep(1, n),
  normalize = FALSE)
```

Arguments

x	An R object of form: numeric vector, numeric matrix.
n	integer; the window / subset size to roll over.
by.column	boolean; if TRUE we loop over columns, otherwise we loop over rows.
weights	A numeric vector of weights of same length as n.
normalize	boolean; if TRUE, we normalize the weight of vectors supplied.

Note

Elements in weights equal to 0 are skipped.

roll_prod *Rolling Prod*

Description

This function implements a rolling prod with C++/Rcpp.

Usage

```
roll_prod(x, n, by.column = TRUE, weights = rep(1, n),
  normalize = FALSE)
```

Arguments

x	An R object of form: numeric vector, numeric matrix.
n	integer; the window / subset size to roll over.
by.column	boolean; if TRUE we loop over columns, otherwise we loop over rows.
weights	A numeric vector of weights of same length as n.
normalize	boolean; if TRUE, we normalize the weight of vectors supplied.

Note

Elements in weights equal to 0 are skipped.

roll_sd *Rolling Sd*

Description

This function implements a rolling sd with C++/Rcpp.

Usage

```
roll_sd(x, n, by.column = TRUE, weights = rep(1, n),
  normalize = FALSE)
```

Arguments

x	An R object of form: numeric vector, numeric matrix.
n	integer; the window / subset size to roll over.
by.column	boolean; if TRUE we loop over columns, otherwise we loop over rows.
weights	A numeric vector of weights of same length as n.
normalize	boolean; if TRUE, we normalize the weight of vectors supplied.

Note

Elements in `weights` equal to 0 are skipped.

`roll_sum` *Rolling Sum*

Description

This function implements a rolling sum with C++/Rcpp.

Usage

```
roll_sum(x, n, by.column = TRUE, weights = rep(1, n),
  normalize = FALSE)
```

Arguments

`x`	An R object of form: numeric vector, numeric matrix.
`n`	integer; the window / subset size to roll over.
`by.column`	boolean; if `TRUE` we loop over columns, otherwise we loop over rows.
`weights`	A numeric vector of weights of same length as `n`.
`normalize`	boolean; if `TRUE`, we normalize the weight of vectors supplied.

Note

Elements in `weights` equal to 0 are skipped.

`roll_var` *Rolling Var*

Description

This function implements a rolling var with C++/Rcpp.

Usage

```
roll_var(x, n, by.column = TRUE, weights = rep(1, n),
  normalize = FALSE)
```

Arguments

`x`	An R object of form: numeric vector, numeric matrix.
`n`	integer; the window / subset size to roll over.
`by.column`	boolean; if `TRUE` we loop over columns, otherwise we loop over rows.
`weights`	A numeric vector of weights of same length as `n`.
`normalize`	boolean; if `TRUE`, we normalize the weight of vectors supplied.

Note

Elements in `weights` equal to 0 are skipped.

Index