Package 'cppdoubles'

July 22, 2025

Title Fast Relative Comparisons of Floating Point Numbers in 'C++'

Version 0.4.0

Description Compare double-precision floating point vectors using relative differences. All equality operations are calculated using 'cpp11'.

License MIT + file LICENSE

BugReports https://github.com/NicChr/cppdoubles/issues

Depends R (>= 3.5.0)

Suggests bench, testthat (>= 3.0.0)

LinkingTo cpp11

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation yes

Author Nick Christofides [aut, cre] (ORCID: <https://orcid.org/0000-0002-9743-7342>)

Maintainer Nick Christofides <nick.christofides.r@gmail.com>

Repository CRAN

Date/Publication 2025-06-09 14:20:02 UTC

Contents

`all_equal` *Are all values of x nearly equal (within a tolerance) to all values of y?*

Description

A memory-efficient alternative to `all.equal.numeric()`.

Usage

```
all_equal(x, y, tol = get_tolerance(), na.rm = FALSE)
```

Arguments

`x`	A double vector.
`y`	A double vector.
`tol`	A double vector of tolerances.
`na.rm`	Should `NA` values be ignored? Default is `FALSE`.

Details

`all_equal` compares each pair of double-precision floating point numbers in the same way as `double_equal`. If any numbers differ, the algorithm breaks immediately, which can offer significant speed when there are differences at the start of a vector. All arguments are recycled except `na.rm`.

Value

A logical vector of length 1.

The result should match `all(double_equal(x, y))`, including the way `NA` values are handled.

Examples

```
library(cppdoubles)
library(bench)
x <- seq(0, 1, 0.2)
y <- sqrt(x)^2

all_equal(x, y)

# Comparison to all.equal
z <- runif(10^4, 1, 100)
ones <- rep(1, length(z))
mark(base = isTRUE(all.equal(z, z)),
            cppdoubles = all_equal(z, z),
            iterations = 100)
mark(base = isTRUE(all.equal(z, ones)),
            cppdoubles = all_equal(z, ones),
            iterations = 100)
```

`rel_diff` *Absolute and relative difference*

Description

Calculate absolute differences with `abs_diff()` and relative differences with `rel_diff()`

Usage

```
rel_diff(x, y, scale = NA_real_)

abs_diff(x, y)
```

Arguments

`x`	A double vector.
`y`	A double vector.
`scale`	A double vector. When `NA`, the scale is calculated as `max(abs(x), abs(y))`.

Details

Relative difference:

The relative difference in this package is calculated as `abs_diff(x / scale, y / scale)` except in the case that both `x` and `y` are approximately 0 which results in 0.

The scale is calculated as `max(abs(x), abs(y))` by default when scale is `NA`. This has the nice property of making `rel_diff()` a commutative function in which the order of the arguments doesn't matter. You can of course supply your own scale.

For info, an R way to calculate the relative difference is as follows

```
r_rel_diff <- function(x, y){
  ax <- abs(x)
  ay <- abs(y)
  scale <- pmax(ax, ay)
  ifelse(
    ax < sqrt(.Machine$double.eps) & ay < sqrt(.Machine$double.eps),
    0,
    abs_diff(x / scale, y / scale)
  )
}
```

This is much slower than the C++ written `rel_diff`.

Comparison with `all.equal()`:

As mentioned above, unlike `base::all.equal()`, `rel_diff()` is commutative. To match the relative difference calculation used by `all.equal()`, simply set `scale = x`.

Therefore, to make a vectorised binary version of `all.equal()`, we can write for example the following:

```
all.equal2 <- \(x, y, tol = get_tolerance()) rel_diff(x, y, scale = x) < tol
```

Value

A numeric vector.

`tolerance` *Get and set package-wide tolerance*

Description

Get and set package-wide tolerance

Usage

```
get_tolerance()

set_tolerance(x)
```

Arguments

x	`[double(1)]` - Tolerance to be used across all cppdoubles functions.

Value

Either sets or gets the tolerance to be used package-wide.

`%~==%` *Relative comparison of double-precision floating point numbers*

Description

Fast and efficient methods for comparing floating point numbers using relative differences.

Usage

```
x %~==% y

x %~>=% y

x %~>% y

x %~<=% y

x %~<% y

double_equal(x, y, tol = get_tolerance())
```

```
double_gte(x, y, tol = get_tolerance())

double_gt(x, y, tol = get_tolerance())

double_lte(x, y, tol = get_tolerance())

double_lt(x, y, tol = get_tolerance())
```

Arguments

x	A double vector.
y	A double vector.
tol	A double vector of tolerances.

Details

When either `x[i]` or `y[i]` contain a number very close to zero, absolute differences are used, otherwise relative differences are used.

The output of `double_equal()` is commutative, which means the order of arguments don't matter whereas this is not the case for `all.equal.numeric()`.

The calculation is done in C++ and is quite efficient. Recycling follows the usual R rules and is done without allocating additional memory.

Value

A logical vector.

Examples

```
library(cppdoubles)

### Basic usage ###

# Standard equality operator
sqrt(2)^2 == 2

# approximate equality operator
sqrt(2)^2 %~==% 2

sqrt(2)^2 %~>=% 2
sqrt(2)^2 %~<=% 2
sqrt(2)^2 %~>% 2
sqrt(2)^2 %~<% 2

# Alternatively
double_equal(2, sqrt(2)^2)
double_gte(2, sqrt(2)^2)
double_lte(2, sqrt(2)^2)
double_gt(2, sqrt(2)^2)
double_lt(2, sqrt(2)^2)
```

```
rel_diff(1, 1 + 2e-10)
double_equal(1, 1 + 2e-10, tol = sqrt(.Machine$double.eps))
double_equal(1, 1 + 2e-10, tol = 1e-10)

# Optionally set a threshold for all comparison
options(cppdoubles.tolerance = 1e-10)
double_equal(1, 1 + 2e-10)

# Floating point errors magnified example

x1 <- 1.1 * 100 * 10^200
x2 <- 110 * 10^200

abs_diff(x1, x2) # Large absolute difference
rel_diff(x1, x2) # Very small relative difference as expected

double_equal(x1, x2)

# all.equal is not commutative but double_equal is
all.equal(10^-8, 2 * 10^-8)
all.equal(2 * 10^-8, 10^-8)

double_equal(10^-8, 2 * 10^-8)
double_equal(2 * 10^-8, 10^-8)

# All comparisons are vectorised and recycled

double_equal(sqrt(1:10),
             sqrt(1:5),
             tol = c(-Inf, 1e-10, Inf))

# One can check for whole numbers like so
whole_number <- function(x, tol = get_tolerance()){
  double_equal(x, round(x))
}
whole_number(seq(-5, 5, 0.25))
```

Index