

# Package ‘hutilscpp’

February 3, 2019

**Title** Miscellaneous Functions in C++

**Version** 0.1.0

**Description** Provides utility functions that are simply, frequently used, but may require higher performance than what can be obtained from base R. Incidentally provides support for 'reverse geocoding', such as matching a point with its nearest neighbour in another array. Used as a complement to package 'hutils' by sacrificing compilation or installation time for higher running speeds. The name is a portmanteau of the author and 'Rcpp'.

**URL** <https://github.com/hughparsonage/hutilscpp>

**BugReports** <https://github.com/hughparsonage/hutilscpp/issues>

**License** GPL-2

**Encoding** UTF-8

**LazyData** true

**LinkingTo** Rcpp

**Imports** Rcpp, data.table, hutils, utils

**RoxygenNote** 6.1.1

**Suggests** bench, testthat, TeXCheckR, covr

**NeedsCompilation** yes

**Author** Hugh Parsonage [aut, cre]

**Maintainer** Hugh Parsonage <[hugh.parsonage@gmail.com](mailto:hugh.parsonage@gmail.com)>

**Repository** CRAN

**Date/Publication** 2019-02-03 15:43:19 UTC

## R topics documented:

anyOutside . . . . .	2
AnyWhich . . . . .	3
are_even . . . . .	3
bench_system_time . . . . .	4
cumsum_reset . . . . .	4

do_pmaxC . . . . .	5
do_pmaxV . . . . .	6
do_pminC . . . . .	6
do_pminV . . . . .	7
helper . . . . .	7
match_nrst_haversine . . . . .	8
pmaxC . . . . .	9
pminC . . . . .	10
poleInaccessibility . . . . .	11
range_rcpp . . . . .	12
squish . . . . .	13
which_first . . . . .	14
which_true_onwards . . . . .	15

<b>Index</b>	<b>16</b>
--------------	-----------

---

anyOutside	<i>Are any values outside the interval specified?</i>
------------	---

---

## Description

Are any values outside the interval specified?

## Usage

```
anyOutside(x, a, b, nas_absent = NA, na_is_outside = NA)
```

## Arguments

x	A numeric vector.
a, b	Single numeric values designating the interval.
nas_absent	Are NAs <i>known</i> to be absent from x? If nas_absent = NA, the default, x will be searched for NAs; if nas_absent = TRUE, x will not be checked; if nas_absent = FALSE, the answer is NA_integer_ if na.rm = FALSE otherwise only non-NA values outside [a, b]. If nas_absent = TRUE but x has missing values then the result is unreliable.
na_is_outside	(logical, default: NA) How should NAs in x be treated? <b>If</b> NA the default, then the first value in x that is either outside [a, b] or NA is detected: if it is NA, then NA_integer_ is returned; otherwise the position of that value is returned.#? <b>If</b> FALSE then NA values are effectively skipped; the position of the first <i>known</i> value outside [a, b] is returned. <b>If</b> TRUE the position of the first value that is either outside [a, b] or NA is returned.

**Value**

0L if no values in `x` are outside `[a, b]`. Otherwise, the position of the first value of `x` outside `[a, b]`.

**Examples**

```
anyOutside(1:10, 1L, 10L)
anyOutside(1:10, 1L, 7L)

# na_is_outside = NA
anyOutside(c(1:10, NA), 1L, 7L) # Already outside before the NA
anyOutside(c(NA, 1:10, NA), 1L, 7L) # NA since it occurred first

anyOutside(c(1:7, NA), 1L, 7L, na_is_outside = FALSE)
anyOutside(c(1:7, NA), 1L, 7L, na_is_outside = TRUE)
```

---

AnyWhich

*Quickly verify (and locate) the existence of a breach.*


---

**Description**

Used when a single instance is likely to occur and be important to detect quickly (in a sufficiently large integer vector).

**Arguments**

<code>x</code>	An integer vector.
<code>a</code>	A (single) integer. That which is to be compared.
<code>gt, lt, eq</code>	Booleans, whether or not the comparison is greater than, less than, or equal to. Only <code>gt</code> and <code>lt</code> are mutually exclusive. If all FALSE, find the first instance where none are equal (i.e. does <code>x</code> have more than one distinct value).

---

are\_even

*Are even*


---

**Description**

Are even

**Usage**

```
are_even(x, check_integerish = TRUE)

which_are_even(x, check_integerish = TRUE)
```

**Arguments**

`x` An integer vector. Double vectors may also be used.

`check_integerish` (logical, default: TRUE) Should the values in `x` be checked for non-integer values if `x` is a double vector. If TRUE and values are found to be non-integer a warning is emitted.

**Value**

For `are_even`, a logical vector the same length as `x`, TRUE whenever `x` is even.

For `which_are_even` the integer positions of even values in `x`.

---

`bench_system_time` *Evaluate time of computation*

---

**Description**

(Used for examples and tests)

**Usage**

`bench_system_time(expr)`

**Arguments**

`expr` Passed to `system_time`.

---

`cumsum_reset` *Cumulative sum unless reset*

---

**Description**

Cumulative sum unless reset

**Usage**

`cumsum_reset(x, y = as.integer(x))`

**Arguments**

`x` A logical vector indicating when the sum should *continue*.

`y` Optional: a numeric vector the same length as `x` to cumulatively sum.

**Value**

If `y` is a double vector, a double vector of cumulative sums, resetting whenever `x` is `FALSE`; otherwise an integer vector.

If `length(x) == 0`, `y` is returned (i.e. `integer(0)` or `double(0)`).

**Examples**

```
cumsum_reset(c(TRUE, TRUE, FALSE, TRUE, TRUE, TRUE, FALSE))
cumsum_reset(c(TRUE, TRUE, FALSE, TRUE, TRUE, TRUE, FALSE),
              c(1000, 1000, 10000, 10, 20, 33, 0))
```

---

do\_pmaxC

*Internal pmaxC helpers*


---

**Description**

Internal functions used when the overheads of assertions would be too expensive. The `_IP_` flavours modify in place.

**Usage**

```
do_pmaxC_dbl(x, a, in_place = FALSE)
```

```
do_pmaxC_int(x, a, in_place = FALSE)
```

```
do_pmax0(x, in_place = FALSE)
```

```
do_pmaxIPnum0(x)
```

```
do_pmaxIPint0(x)
```

**Arguments**

`x`                   A numeric/integer vector.

`a`                    A single numeric/integer.

`in_place`            Modify `x` in place?

---

do_pmaxV	<i>Parallel maximum in C++</i>
----------	--------------------------------

---

**Description**

A faster pmax().

**Usage**

```
do_pmaxNumNum(x, y, in_place = FALSE)
```

```
do_pmaxIntInt(x, y, in_place = FALSE)
```

**Arguments**

x	A numeric vector.
y	A numeric vector, the same length as x.
in_place	(bool, default: false) Should the function operate on x in-place?

**Value**

The parallel maximum of the input values.

---

do_pminC	<i>Parallel maximum</i>
----------	-------------------------

---

**Description**

A faster pmin().

**Arguments**

x	A numeric vector.
a	A single numeric value.
in_place	(bool, default: false) Should the function operate on x in-place?

**Value**

The parallel minimum of the input values. The 0 versions are shortcuts for a = 0.

**Note**

This function will always be faster than pmin(x, a) when a is a single value, but can be slower than pmin.int(x, a) when x is short. Use this function when comparing a numeric vector with a single value.

---

do_pminV	<i>Parallel maximum</i>
----------	-------------------------

---

**Description**

A faster pmin().

**Usage**

```
do_pminV_dbl(x, y, in_place = FALSE)
```

```
do_pminV_int(x, y, in_place = FALSE)
```

**Arguments**

x	A numeric vector.
y	A numeric vector, the same length as x.
in_place	(bool, default: false) Modify x in-place?

**Value**

The parallel maximum of the input values.

---

helper	<i>Helper</i>
--------	---------------

---

**Description**

Helper

**Usage**

```
helper(expr)
```

**Arguments**

expr	An expression
------	---------------

**Value**

The expression evaluated.

**Examples**

```
x6 <- 1:6  
helper(x6 + 1)
```

---

match\_nrst\_haversine *Match coordinates to nearest coordinates*

---

## Description

When geocoding coordinates to known addresses, an efficient way to match the given coordinates with the known is necessary. This function provides this efficiency by using C++ and allowing approximate matching.

## Usage

```
match_nrst_haversine(lat, lon, addresses_lat, addresses_lon,
  Index = seq_along(addresses_lat), cartesian_R = NULL,
  close_enough = 10, excl_self = FALSE, as.data.table = TRUE,
  .verify_box = TRUE)
```

## Arguments

lat, lon	Coordinates to be geocoded. Numeric vectors of equal length.
addresses_lat, addresses_lon	Coordinates of known locations. Numeric vectors of equal length (likely to be a different length than the length of lat, except when excl_self = TRUE).
Index	A vector the same length as lat to encode the match between lat, lon and addresses_lat, addresses_lon. The default is to use the integer position of the nearest match to addresses_lat, addresses_lon.
cartesian_R	The maximum radius of any address from the points to be geocoded. Used to accelerate the detection of minimum distances. Note, as the argument name suggests, the distance is in cartesian coordinates, so a small number is likely.
close_enough	The distance, in metres, below which a match will be considered to have occurred. (The distance that is considered "close enough" to be a match.) For example, close_enough = 10 means the first location within ten metres will be matched, even if a more distant match occurs later. May be provided as a string to emphasize the units, e.g. close_enough = "0.25km". Only km and m are permitted.
excl_self	(bool, default: FALSE) For each $x_i$ of the first coordinates, exclude the $y_i$ -th point when determining closest match. Useful to determine the nearest neighbour within a set of coordinates, viz. match_nrst_haversine(x, y, x, y, excl_self = TRUE).
as.data.table	Return result as a data.table? If FALSE, a list is returned. TRUE by default to avoid dumping a huge list to the console.
.verify_box	Check the initial guess against other points within the box of radius $\ell^\infty$ .



**Value**

A list (or data.table if as.data.table = TRUE) with two elements, both the same length as lat, giving for point lat, lon:

pos the position (or corresponding value in Table) in addresses\_lat, addresses\_lon nearest to lat, lon.

dist the distance, in kilometres, between the two points.

**Examples**

```
lat2 <- runif(5, -38, -37.8)
lon2 <- rep(145, 5)

lat1 <- c(-37.875, -37.91)
lon1 <- c(144.96, 144.978)

match_nrst_haversine(lat1, lon1, lat2, lon2, 0L)
match_nrst_haversine(lat1, lon1, lat1, lon1, 11:12, excl_self = TRUE)
```

---

pmaxC

*Parallel maximum/minimum*


---

**Description**

Faster pmax() and pmin().

**Usage**

```
pmaxC(x, a, in_place = FALSE)

pmax0(x, in_place = FALSE)

pmaxV(x, y, in_place = FALSE)

pmax3(x, y, z, in_place = FALSE)
```

**Arguments**

x	A numeric vector.
a	A single numeric value.
in_place	(logical, default: FALSE) Should x be modified in-place.
y, z	Other numeric vectors the same length as x

**Value**

The parallel maximum/minimum of the input values. pmax0(x) is shorthand for pmaxC(x, 0), i.e. convert negative values in x to 0.

**Note**

This function will always be faster than `pmax(x, a)` when `a` is a single value, but can be slower than `pmax.int(x, a)` when `x` is short. Use this function when comparing a numeric vector with a single value.

Use `in_place = TRUE` within functions when you are sure it is safe.

**Examples**

```
pmaxC(-5:5, 2)
```

---

<code>pminC</code>	<i>Parallel minimum</i>
--------------------	-------------------------

---

**Description**

Parallel minimum

**Usage**

```
pmin0(x, in_place = FALSE)
```

```
pminV(x, y, in_place = FALSE)
```

```
pminC(x, a = 0, in_place = FALSE)
```

```
pmin3(x, y, z, in_place = FALSE)
```

**Arguments**

<code>x</code>	A numeric vector.
<code>in_place</code>	(logical, default: FALSE) Should <code>x</code> be modified in-place.
<code>y, z</code>	Other numeric vectors.
<code>a</code>	A single number.

**Details**

The type of `x` is preserved as far as possible.

**Value**

Same as `pmin(x, 0)`.

```
pmin0(x) = pmin(x, 0)
```

```
pminV(x, y) = pmin(x, y)
```

```
pminC(x, a) = pmin(x, a) for length-one a.
```

```
pmin3(x, y, z) = pmin(x, pmin(y, z)).
```

**Examples**

```
pminV(10:1, 1:10)
pmin0(-5:5)
seq_out <- function(x, y) seq(x, y, length.out = 10)
pmin3(seq_out(0, 10), seq_out(-5, 50), seq_out(20, -10))
```

---

poleInaccessibility *Find a binary pole of inaccessibility*

---

**Description**

Find a binary pole of inaccessibility

**Usage**

```
poleInaccessibility2(x = NULL, y = NULL, DT = NULL, x_range = NULL,
  y_range = NULL, copy_DT = TRUE)
```

```
poleInaccessibility3(x = NULL, y = NULL, DT = NULL, x_range = NULL,
  y_range = NULL, copy_DT = TRUE, test_both = TRUE)
```

**Arguments**

<code>x, y</code>	Coordinates.
<code>DT</code>	A data.table containing LONGITUDE and LATITUDE to define the x and y coordinates.
<code>x_range, y_range</code>	Numeric vectors of length-2; the range of x and y. Use this rather than the default when the 'vicinity' of x, y is different from the minimum closed rectangle covering the points.
<code>copy_DT</code>	(logical, default: TRUE) Run <a href="#">copy</a> on DT before proceeding. If FALSE, DT have additional columns updated by reference.
<code>test_both</code>	(logical, default: TRUE) For 3, test both stretching vertically then horizontally and horizontally then vertically.

**Value**

`poleInaccessibility2` A named vector containing the xmin, xmax and ymin, ymax coordinates of the largest rectangle of width an integer power of two that is empty.

`poleInaccessibility3` Starting with the rectangle formed by `poleInaccessibility2`, the rectangle formed by stretching it out vertically and horizontally until the edges intersect the points `x,y`

**Examples**

```

library(data.table)
library(hutils)
# A square with a 10 by 10 square of the northeast corner removed
x <- runif(1e4, 0, 100)
y <- runif(1e4, 0, 100)
DT <- data.table(x, y)
# remove the NE corner
DT_NE <- DT[implies(x > 90, y < 89)]
DT_NE[, poleInaccessibility2(x, y)]
DT_NE[, poleInaccessibility3(x, y)]

```

---

range\_rcpp

*Range C++*


---

**Description**

Range of a vector using Rcpp.

**Usage**

```
range_rcpp(x, warn_empty = TRUE, integer0_range_is_integer = FALSE)
```

**Arguments**

x	A vector for which the range is desired. Vectors with missing values are not supported and have no definite behaviour.
warn_empty	(logical, default: TRUE) If x is empty (i.e. has no length), should a warning be emitted (like <a href="#">range</a> )?
integer0_range_is_integer	(logical, default: FALSE) If x is a length-zero integer, should the result also be an integer? Set to FALSE by default in order to be compatible with <a href="#">range</a> , but can be set to TRUE if an integer result is desired, in which case <code>range_rcpp(integer())</code> is (INT_MAX, -INT_MAX).

**Value**

A length-4 vector, the first two positions give the range and the next two give the positions in x where the max and min occurred.

This is almost equivalent to `c(range(x), which.min(x), which.max(x))`. Note that the type is not strictly preserved, but no loss should occur. In particular, logical x results in an integer result, and a double x will have double values for `which.min(x)` and `which.max(x)`.

A completely empty, logical x returns `c(NA, NA, NA, NA)` as an integer vector.

**Examples**

```
x <- rnorm(1e3) # Not noticeable at this scale
bench_system_time(range_rcpp(x))
bench_system_time(range(x))
```

---

squish	<i>Squish into a range</i>
--------	----------------------------

---

**Description**

Squish into a range

**Usage**

```
squish(x, a, b, in_place = FALSE)
```

**Arguments**

x	A numeric vector.
a, b	Upper and lower bounds
in_place	(logical, default: FALSE) Should the function operate on x in place?

**Value**

A numeric/integer vector with the values of x "squished" between a and b; values above b replaced with b and values below a replaced with a.

**Examples**

```
squish(-5:5, -1L, 1L)
```

---

 which\_first

*Where does a logical expression first return TRUE?*


---

## Description

A faster and safer version of `which.max` applied to simple-to-parse logical expressions.

## Usage

```
which_first(expr, verbose = FALSE)
```

## Arguments

<code>expr</code>	An expression, such as <code>x == 2</code> .
<code>verbose</code>	(logical, default: <code>FALSE</code> ) If <code>TRUE</code> a message is emitted if <code>expr</code> could not be handled in the advertised way.

## Details

If the `expr` is of the form `LHS <operator> RHS` and `LHS` is a single symbol, `operator` is one of `==`, `!=`, `>`, `>=`, `<`, `<=`, or `%in%`. and `RHS` is a single numeric value, then `expr` is not evaluated directly; instead, each element of `LHS` is compared individually.

If `expr` is not of the above form, then `expr` is evaluated and passed to `which.max`.

Using this function can be significantly faster than the alternatives when the computation of `expr` would be expensive, though the difference is only likely to be clear when `length(x)` is much larger than 10 million. But even for smaller vectors, it has the benefit of returning `0L` if none of the values in `expr` are `TRUE`, unlike `which.max`.

Compared to [Position](#) for an appropriate choice of `f` the speed of `which_first` is not much faster when the expression is `TRUE` for some position. However, `which_first` is faster when all elements of `expr` are `FALSE`. Thus `which_first` has a smaller worst-case time than the alternatives for most `x`.

## Value

The same as `which.max(expr)` or `which(expr)[1]` but returns `0L` when `expr` has no `TRUE` values.

## Examples

```
N <- 1e5
# N <- 1e8 ## too slow for CRAN

# Two examples, from slowest to fastest,
# run with N = 1e8 elements

# seconds
x <- rep_len(runif(1e4, 0, 6), N)
```

```

bench_system_time(x > 5)
bench_system_time(which(x > 5))      # 0.8
bench_system_time(which.max(x > 5))  # 0.3
bench_system_time(which_first(x > 5)) # 0.000

## Worst case: have to check all N elements
x <- double(N)
bench_system_time(x > 0)
bench_system_time(which(x > 0))      # 1.0
bench_system_time(which.max(x > 0))  # 0.4 but returns 1, not 0
bench_system_time(which_first(x > 0)) # 0.1

x <- as.character(x)
# bench_system_time(which(x == 5))    # 2.2
bench_system_time(which.max(x == 5))  # 1.6
bench_system_time(which_first(x == 5)) # 1.3

```

---

which\_true\_onwards      *At which point are all values true onwards*

---

### Description

At which point are all values true onwards

### Usage

```
which_true_onwards(x)
```

### Arguments

x                      A logical vector. NA values are not permitted.

### Value

The position of the first TRUE value in x at which all the following values are TRUE.

### Examples

```
which_true_onwards(c(TRUE, FALSE, TRUE, TRUE, TRUE))
```

# Index

anyOutside, 2  
AnyWhich, 3  
are\_even, 3  
  
bench\_system\_time, 4  
  
copy, 11  
cumsum\_reset, 4  
  
do\_pmax0 (do\_pmaxC), 5  
do\_pmaxC, 5  
do\_pmaxC\_dbl (do\_pmaxC), 5  
do\_pmaxC\_int (do\_pmaxC), 5  
do\_pmaxIntInt (do\_pmaxV), 6  
do\_pmaxIPint0 (do\_pmaxC), 5  
do\_pmaxIPnum0 (do\_pmaxC), 5  
do\_pmaxNumNum (do\_pmaxV), 6  
do\_pmaxV, 6  
do\_pminC, 6  
do\_pminV, 7  
do\_pminV\_dbl (do\_pminV), 7  
do\_pminV\_int (do\_pminV), 7  
  
helper, 7  
  
match\_nrst\_haversine, 8  
  
pmax0 (pmaxC), 9  
pmax3 (pmaxC), 9  
pmaxC, 9  
pmaxV (pmaxC), 9  
pmin0 (pminC), 10  
pmin3 (pminC), 10  
pminC, 10  
pminV (pminC), 10  
poleInaccessibility, 11  
poleInaccessibility2  
    (poleInaccessibility), 11  
poleInaccessibility3  
    (poleInaccessibility), 11  
Position, 14  
  
range, 12  
range\_rcpp, 12  
  
squish, 13  
system\_time, 4  
  
which\_are\_even (are\_even), 3  
which\_first, 14  
which\_true\_onwards, 15