Package 'roundwork'

November 3, 2026

Title Rounding Infrastructure

Version 0.2.0

Description Flexible rounding functions for use in error detection, including the exact rounding bounds that consistency tests are built on. They were outsourced from the 'scrutiny' package, which depends on them since its version 1.0.0.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Suggests conflicted, dplyr, ggplot2, knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Depends R (>= 4.1.0)

Imports cli (>= 3.0.0), rlang (>= 1.1.0), tibble

Collate 'aaa.R' 'round.R' 'ieee-754.R' 'import-standalone-last-export.R' 'utils.R' 'reround-to-fraction.R' 'round-ceil-floor.R' 'reround.R' 'resolve-ties-rounding.R' 'round-ties.R' 'rounding-bias.R' 'unround.R'

URL https://lhdjung.github.io/roundwork/, https://github.com/lhdjung/roundwork/

BugReports https://github.com/lhdjung/roundwork/issues

Config/testthat/edition 3

NeedsCompilation no

Author Lukas Jung [aut, cre]

Maintainer Lukas Jung <jung-lukas@gmx.net>

Repository CRAN

Date/Publication 2026-11-03 19:00:02 UTC

Contents

bound_numerators . . . 2
fractional-rounding . . . 4
ieee-754 . . . 5
reround . . . 7
resolve_ties_rounding . . . 8
rounding-common . . . 9
rounding-ties . . . 11
rounding-uncommon . . . 13
rounding_bias . . . 14
sum_range . . . 15
unround . . . 17

Index **20**

`bound_numerators` *Rounding bounds as exact integers*

Description

`bound_numerators()` returns the same rounding bounds as `unround()`, but as exact integer numerators over a common denominator rather than as decimal numbers. It is the low-level counterpart to `unround()`, meant for building consistency tests on top of.

Most users want `unround()` instead.

Usage

```
bound_numerators(x_num, digits, rounding, threshold, symmetric)
```

Arguments

`x_num`	Numeric (length 1). The rounded number. Not checked.
`digits`	Integer (length 1). Number of decimal places in `x_num`. Not checked.
`rounding`	String (length 1). Rounding method presumably used to create `x_num`. See the `Rounding` section of `unround()`.
`threshold`	Numeric (length 1). The point within a step at which rounding switches direction, in tenths of a step, for the `"up_from"`, `"down_from"`, and `"up_from_or_down_from"` methods. See `round_up_from()`.
`symmetric`	Logical (length 1). Whether the rounding of negative numbers mirrored that of positive numbers. See `round_up()`.

Details

Deriving a set of candidate values from floating-point bounds is unsafe. If a product such as `floor(upper * n)` is mathematically an exact integer, its `double` representation can fall on either side of it, so a legitimate candidate may be silently dropped or a phantom one admitted. Either way, a consistency verdict can flip. See https://github.com/lhdjung/scrutiny/issues/86.

Every bound that `unround()` can return is `x` plus a whole number of units of `1 / 10^(digits + 1)`, and `x` itself is a whole number of such units because it has `digits` decimal places. Both bounds therefore have exact integer numerators over `10^(digits + 1)`, and a comparison of some value to a bound becomes a comparison between integers, which is exact in double precision as long as every integer involved stays below `2^53`.

A consistency test that derives its candidate range from here can never disagree with `unround()` about what the bounds of a rounded number are.

If `threshold` is fractional, the numerators and the denominator are both scaled up by the power of ten that makes them whole numbers again: `100` for a `threshold` of `9.99`. This works for up to six decimal places, a scale of `1e6`. With more, the numerators are not whole numbers, and the arithmetic is no longer exact. The scale also multiplies the integers that must stay below `2^53`.

`bound_numerators()` is a low-level primitive, meant to be called once per value set inside a consistency test, so it checks only `rounding`, `threshold`, and `symmetric`. The caller is responsible for passing a single number as `x_num` and a whole number of decimal places that fits it as `digits`. Other input may return meaningless numerators or fail with an uninformative error. `unround()` checks all of its input.

Value

A list of five elements, or `NULL` if the bounds are undefined, which is the case only for a missing or infinite `x_num` or a missing `digits`:

- `lower`, `upper`: integer numerators of the two bounds.
- `denom`: the common denominator.
- `incl_lower`, `incl_upper`: whether each bound is inclusive.

Throws an error if `rounding` is not a known method.

See Also

`unround()`, which presents the same bounds as decimal numbers in a tibble.

Examples

```
# The bounds of 2.7, rounded up or down from 5,
# are 2.65 and 2.75 -- here as 265/100 and 275/100:
bound_numerators(
  x_num = 2.7,
  digits = 1,
  rounding = "up_or_down",
  threshold = 5,
  symmetric = FALSE
)
```

```
# `unround()` shows the same bounds as decimals:
unround("2.7")
```

`fractional-rounding` *Rounding to the nearest fraction*

Description

These functions round to the nearest fraction of a denominator rather than to a decimal place. They follow `janitor::round_to_fraction()` but take all of `reround()`'s options:

- `reround_to_fraction()` rounds to fractions of a whole number, then optionally rounds the result to `digits` decimal places.
- `reround_to_fraction_level()` rounds to the nearest fraction at the decimal level given by `digits`, with no further rounding.

Usage

```
reround_to_fraction(
  x = NULL,
  denominator = 1,
  digits = Inf,
  rounding = "up_or_down",
  threshold = 5,
  symmetric = FALSE
)

reround_to_fraction_level(
  x = NULL,
  denominator = 1,
  digits = 0L,
  rounding = "up_or_down",
  threshold = 5,
  symmetric = FALSE
)
```

Arguments

`x`	Numeric. Values to round.
`denominator`	Numeric (at least 1). `x` is rounded to the nearest fraction of `denominator`. Default is `1`.
`digits`	Integer. In `reround_to_fraction()`, the number of decimal places to round the result to. `"auto"` means `ceiling(log10(denominator)) + 1`, as in janitor, and the default `Inf` means no further rounding. In `reround_to_fraction_level()`, the decimal level at which to round to a fraction; default is `0`.
`rounding, threshold, symmetric`	Passed on to `reround()`.

Value

Numeric, as long as x, except for the compound rounding methods, which return two values per element of x as in reround().

See Also

reround() and janitor::round_to_fraction().

Examples

```
# With `denominator = 1`, this is ordinary rounding...
reround_to_fraction(0.4, denominator = 1, rounding = "even")

# ...with `denominator = 2`, it rounds to the nearest half:
reround_to_fraction(0.4, denominator = 2, rounding = "even")
reround_to_fraction(0.25, denominator = 3, rounding = "even")

# The default `rounding` returns both directions, as in `reround()`:
reround_to_fraction(0.25, denominator = 2)

# `reround_to_fraction_level()` rounds to a fraction at a decimal level:
reround_to_fraction_level(0.12345, denominator = 2, digits = 0)
reround_to_fraction_level(0.12345, denominator = 2, digits = 1)
reround_to_fraction_level(0.12345, denominator = 2, digits = 2)
```

---

ieee-754 *IEEE 754 rounding standard*

Description

These functions implement the industry standard IEEE 754:

- round_ties_to_even() rounds to the nearest value. If both are at equal distance, it tends to round to the even number (see base::round() for details).
- round_ties_to_away() rounds to the nearest value. If both are at equal distance, it rounds to the number with the greater absolute value, i.e., the number that is further away from zero.
- round_toward_positive() always rounds to the greater of the two nearest values. This is like ceiling at a given number of decimal places.
- round_toward_negative() always rounds to the lesser of the two nearest values. This is like flooring at a given number of decimal places.
- round_toward_zero() always rounds to the number with the lower absolute value, i.e., the number that is closer to zero. This is like truncation at a given number of decimal places.

Usage

```
round_ties_to_even(x, digits = 0L)

round_ties_to_away(x, digits = 0L)

round_toward_positive(x, digits = 0L)

round_toward_negative(x, digits = 0L)

round_toward_zero(x, digits = 0L)
```

Arguments

x	Numeric. The number to round.
digits	Integer. Number of decimal places to round to. Default is 0. Negative values round to powers of ten: round_up(1250, digits = -2) is 1300.

Details

The function names follow the official standard except for case conventions (IEEE 2019, pp. 27f.; the Wikipedia page is more accessible but uses slightly different names).

Internally, these functions are just wrappers around other roundwork functions as well as base::round(). They are presented here for easy compliance with the IEEE 754 standard in R. Each of them has an equivalent elsewhere in the package, and the results are identical:

IEEE 754 name	Equivalent
round_ties_to_even()	base::round()
round_ties_to_away()	round_ties_away()
round_toward_positive()	round_ceiling()
round_toward_negative()	round_floor()
round_toward_zero()	round_trunc()

Note that IEEE 754 has no name for the other three tie-breaking procedures in round_ties_up() and friends, nor for rounding away from zero (round_anti_trunc()), nor for the custom thresholds of round_up_from().

Value

Numeric. x rounded to digits.

References

IEEE (2019). *IEEE Standard for Floating-Point Arithmetic.* https://doi.org/10.1109/IEEESTD.2019.8766229

See Also

round_ties_up() and friends for the same tie-breaking procedures under names that do not presuppose the standard, and reround() for reaching any of them through a rounding string.

Examples

```
# Round to the nearest value. In case of a tie,
# the result is hard to predict but tends to be even:
round_ties_to_even(1.25, digits = 1)
round_ties_to_even(-1.25, digits = 1)

# Round to the nearest value. In case of a tie,
# round away from zero:
round_ties_to_away(1.25, digits = 1)
round_ties_to_away(-1.25, digits = 1)

# Always round to the greater value:
round_toward_positive(0.721, digits = 2)
round_toward_positive(-0.721, digits = 2)

# Always round to the lesser value:
round_toward_negative(3.249, digits = 2)
round_toward_negative(-3.249, digits = 2)

# Always round toward zero:
round_toward_zero(6.38, digits = 1)
round_toward_zero(-6.38, digits = 1)
```

---

reround *General interface to reconstructing rounded numbers*

Description

`reround()` rounds a vector of reconstructed values the way they are assumed to have been rounded originally. It is the common interface to all of roundwork's rounding functions and to `base::round()`.

Usage

```
reround(
  x,
  digits = 0L,
  rounding = "up_or_down",
  threshold = 5,
  symmetric = FALSE
)
```

Arguments

`x`	Numeric. Values to round.
`digits`	Integer. Number of decimal places to round to, i.e., the number in the reported values. Length 1 or the length of `x`. Must be whole numbers; a missing value propagates to a missing result.

rounding	String (length 1). The rounding method. Default is "up_or_down", which returns two values per element of x: rounded up and rounded down.
threshold	Numeric (length 1). For "up_from", "down_from", and "up_from_or_down_from", the point within a step at which rounding switches direction, in tenths of a step; greater than 0 and less than 10. Default is 5, which makes these methods the same as "up", "down", and "up_or_down". See round_up_from().
symmetric	Logical (length 1). If TRUE, negative numbers are rounded like their absolute values under "up", "down", "up_or_down", and the "*_from" methods. Default is FALSE. Must not be TRUE with a "ties_*" method, which already fixes the direction: "ties_up" with symmetric = TRUE would be "ties_away", so write that instead.

Details

reround() calls the rounding function that rounding names. See vignette("rounding-options") for the possible values, and round_up() and round_ceiling() for the functions themselves.

Value

Numeric. One value per element of x, except for the compound methods "up_or_down", "up_from_or_down_from", and "ceiling_or_floor", which return two: the results of their two constituent procedures, next to each other, as c(up_1, down_1, up_2, down_2, ...). Each pair belongs to one element of x, so the pairs must not be pooled across elements.

---

resolve_ties_rounding *Resolve a "ties_*" rounding string*

Description

resolve_ties_rounding() translates one of the "ties_*" rounding strings into the rounding and symmetric values it stands for, such as "ties_away" into rounding = "up" and symmetric = TRUE. Any other rounding is returned unchanged, along with symmetric.

reround() and unround() resolve these strings the same way, so a function that branches on rounding itself can call this first to agree with them.

Usage

```
resolve_ties_rounding(rounding, symmetric)
```

Arguments

rounding	String (length 1). A rounding method. Not checked.
symmetric	Logical (length 1). See round_up(). Not checked.

Details

`resolve_ties_rounding()` is a low-level primitive and does not check its input. The caller is responsible for passing a single string as `rounding` and a single `TRUE` or `FALSE` as `symmetric`. In particular, a number as `rounding` would pick a `"ties_*"` method by position.

Combining a `"ties_*"` string with `symmetric = TRUE` is an error, because the string already determines how negative ties are broken.

Value

A list with the elements `rounding` and `symmetric`.

Examples

```
resolve_ties_rounding("ties_away", symmetric = FALSE)

# Other methods pass through:
resolve_ties_rounding("up", symmetric = TRUE)
```

---

rounding-common *Common rounding procedures*

Description

`round_up()` rounds ties upward, and `round_down()` rounds them downward. Otherwise, both agree with `base::round()`.

They are special cases of `round_up_from()` and `round_down_from()`, which switch direction at a `threshold` other than 5.

Usage

```
round_up_from(x, digits = 0L, threshold, symmetric = FALSE)

round_down_from(x, digits = 0L, threshold, symmetric = FALSE)

round_up(x, digits = 0L, symmetric = FALSE)

round_down(x, digits = 0L, symmetric = FALSE)
```

Arguments

`x`	Numeric. The number to round.
`digits`	Integer. Number of decimal places to round to. Default is `0`. Negative values round to powers of ten: `round_up(1250, digits = -2)` is `1300`.

`threshold` Numeric. Only in `round_up_from()` and `round_down_from()`. The point within a step at which rounding switches direction, in tenths of a step; greater than `0` and less than `10`. `round_up_from()` rounds up when the part cut off is at least `threshold` tenths of a step, and `round_down_from()` rounds down when it is at most that many, so the two only differ for a value exactly on the threshold. `round_up()` and `round_down()` are the pair at `5`.

`symmetric` Logical. If `TRUE`, negative numbers are rounded like their absolute values. Default is `FALSE`. See *Negative numbers*.

Details

These functions decide a tie by a fixed rule. `base::round()` rounds ties to even instead, and decides on the binary double rather than the printed decimal. They exist to reconstruct the output of software that rounds this way; see `vignette("rounding-options")` for which program uses which procedure. They are not meant to replace `base::round()` in original computations.

Value

Numeric. `x` rounded to `digits`.

Negative numbers

By default (`symmetric = FALSE`), `round_up()` moves a tie to the higher number on the number line, so `round_up(-2.5)` is `-2`, and `round_down(-2.5)` is `-3`. This is what Stata does.

With `symmetric = TRUE`, a negative number is rounded like its absolute value, so a tie moves away from zero: `round_up(-2.5, symmetric = TRUE)` is `-3`. This is what Excel, SAS, SPSS, and Matlab do; IEEE 754 calls it *roundTiesToAway*.

For `round_up()` and `round_down()`, `symmetric` only affects ties among negative numbers. For `round_up_from()` and `round_down_from()` with a `threshold` other than 5, it affects every negative number whose cut-off part lies between `threshold` and `10 - threshold`: `round_up_from(-4.28, 1, threshold = 9)` is `-4.3`, and `-4.2` with `symmetric = TRUE`.

Under the package-wide default `rounding = "up_or_down"`, both results are accepted, so `symmetric` cannot change a consistency verdict there.

Floating-point tolerance

Shifting a number by `digits` decimal places is not exact: `0.145 * 100` is stored as `14.499999999999998`, just below the tie it is meant to be. All of these functions therefore nudge the shifted value by about `1.5e-9` before rounding, so that `round_up(0.145, 2)` is `0.15`, as in the software being reconstructed. `unround()` reports bounds that assume the same tolerance.

The nudge is fixed, whereas representation error grows with `x * 10^digits`. Up to about `1e7` for that product, far beyond any reported summary statistic, the nudge dominates. Past it, a value exactly on a boundary may be rounded in either direction.

See Also

`round_ties_up()` and friends are the same procedures under names that state the tie rule, so that `symmetric` need not be given. `round_ceiling()`, `round_floor()`, `round_trunc()`, and `round_anti_trunc()` always round in one direction.

Examples

```
# Except at a tie, all of these agree:
   round_up(x = 9.273, digits = 1)
 round_down(x = 9.273, digits = 1)
base::round(x = 9.273, digits = 1)

# At ties, they differ:
original <- c(
  0.05, 0.15, 0.25, 0.35, 0.45,
  0.55, 0.65, 0.75, 0.85, 0.95
)
tibble::tibble(
  original,
  round_up = round_up(x = original, digits = 1),
  round_down = round_down(x = original, digits = 1),
  base_round = base::round(x = original, digits = 1)
)

# Ties in negative numbers go up on the number line
# by default, and away from zero with `symmetric`:
round_up(x = -2.5)                    # Stata
round_up(x = -2.5, symmetric = TRUE)   # Excel, SAS, SPSS, Matlab

# A custom threshold moves the point at which
# rounding switches direction:
round_up_from(x = 4.28, digits = 1, threshold = 9)   # 8 < 9, so down
round_up_from(x = 4.28, digits = 1, threshold = 1)   # 8 >= 1, so up
```

`rounding-ties` *Rounding procedures named by their tie-breaking rule*

Description

Each of these functions rounds to the nearest number with `digits` decimal places. They differ only at a tie, a value exactly halfway between two such numbers:

- `round_ties_up()` sends a tie to the higher number.
- `round_ties_down()` sends a tie to the lower number.
- `round_ties_away()` sends a tie away from zero.
- `round_ties_zero()` sends a tie toward zero.
- `round_ties_even()` sends a tie to the even neighbor.

The first four are `round_up()` and `round_down()` with and without `symmetric = TRUE`, under names that state the procedure. They take no `symmetric` argument, since the name already fixes the direction on both sides of zero. `round_ties_even()` is `base::round()`.

The matching `rounding` strings, `"ties_up"` and so on, are accepted wherever `rounding` is, and giving `symmetric = TRUE` alongside one of them is an error.

Usage

```
round_ties_up(x, digits = 0L)

round_ties_down(x, digits = 0L)

round_ties_away(x, digits = 0L)

round_ties_zero(x, digits = 0L)

round_ties_even(x, digits = 0L)
```

Arguments

x	Numeric. The number to round.
digits	Integer. Number of decimal places to round to. Default is 0.

Details

round_ties_away() is IEEE 754's *roundTiesToAway*, which Excel's ROUND(), SPSS's RND(), SAS's ROUND(), and Matlab's round() implement. round_ties_up() is what Stata's round() does. round_ties_even() is *roundTiesToEven*, the standard's default and what R, Python, and NumPy do. See vignette("rounding-options").

"Up" and "down" mean up and down on the number line, not the away-from-zero sense that "round up" carries in Excel, Java, and Python. For rounding that is always away from zero or toward it, see round_anti_trunc() and round_trunc().

Unlike the other four, round_ties_even() cannot be predicted from the printed decimal: base::round() decides on the binary double, and most decimal fractions are not exact ties in binary. Consistency tests treat both bounds under rounding = "even" as inclusive for that reason; see unround().

Value

Numeric. x rounded to digits.

See Also

round_up() for the same procedures spelled with symmetric, and for the shared details on thresholds and floating-point tolerance.

Examples

```
# All five agree except at ties:
round_ties_up(x = 1.24, digits = 1)
round_ties_down(x = 1.24, digits = 1)

# At a tie, they differ, but only by sign below zero:
round_ties_up(x = c(-2.5, 2.5))
round_ties_down(x = c(-2.5, 2.5))
round_ties_away(x = c(-2.5, 2.5))    # Excel, SPSS, SAS, Matlab
round_ties_zero(x = c(-2.5, 2.5))
```

```
# Each of the four is a spelling of `round_up()` or `round_down()`:
round_ties_away(x = -2.5)
round_up(x = -2.5, symmetric = TRUE)

# `round_ties_even()` is `base::round()`:
round_ties_even(x = c(0.5, 1.5, 2.5))
```

`rounding-uncommon` *Rounding in one direction*

Description

These functions ignore which neighbor is nearer:

- `round_ceiling()` always rounds up.
- `round_floor()` always rounds down.
- `round_trunc()` always rounds toward zero.
- `round_anti_trunc()` always rounds away from zero. A value that already sits on the rounding grid stays where it is, so `0` stays `0`. This is what Excel's `ROUNDUP()`, Java's `RoundingMode.UP`, and Python's `decimal.ROUND_UP` do.
- `anti_trunc()` is the counterpart of `trunc()`: it rounds a number to the nearest whole number away from zero. Whole numbers stay unchanged.

Usage

```
round_ceiling(x, digits = 0L)

round_floor(x, digits = 0L)

round_trunc(x, digits = 0L)

anti_trunc(x)

round_anti_trunc(x, digits = 0L)
```

Arguments

`x`	Numeric. The number to round.
`digits`	Integer. Number of decimal places to round to. Default is `0`. Negative values round to powers of ten.

Details

round_ceiling(), round_floor(), and round_trunc() generalize ceiling(), floor(), and trunc() to any number of decimal places, and round_anti_trunc() generalizes anti_trunc(). With the default digits = 0, each equals its base counterpart.

round_trunc() is round_floor() for positive and round_ceiling() for negative numbers; round_anti_trunc() is the reverse.

All of these apply the floating-point tolerance described in round_up(), so that round_ceiling(0.28, 2) is 0.28 although 0.28 * 100 is stored as 28.000000000000004.

Value

Numeric. x rounded to digits (anti_trunc() has no digits argument).

See Also

round_up() and round_down() round to the nearest number and decide ties by a fixed direction.

Examples

```
round_ceiling(x = 4.52, digits = 1)
round_floor(x = 4.67, digits = 1)

round_trunc(8.439, digits = 2)
round_trunc(-8.439, digits = 2)

round_anti_trunc(x = 8.421, digits = 2)
round_anti_trunc(x = -8.421, digits = 2)
```

---

rounding_bias *Compute rounding bias*

Description

Rounding changes the mean of a distribution. rounding_bias() computes this bias as the mean of the rounded values minus the mean of the original ones, or, with mean = FALSE, the individual differences.

Usage

```
rounding_bias(
  x,
  digits,
  rounding = "up",
  threshold = 5,
  symmetric = FALSE,
  mean = TRUE
)
```

Arguments

x	Numeric, or a string coercible to numeric.
digits	Integer. Number of decimal places to round to.
rounding	String. The rounding method. Default is "up". See vignette("rounding-options").
threshold, symmetric	Passed on to reround().
mean	Logical. If TRUE (the default), return the mean bias; otherwise, the individual biases.

Details

All arguments except mean are passed on to reround(). Unlike there, rounding defaults to "up" and cannot be a compound method ("up_or_down", "up_from_or_down_from", "ceiling_or_floor"), since those return two rounded values per input and have no single bias.

Value

Numeric. Length 1 by default, or the length of x with mean = FALSE.

Examples

```
vec <- 0.01 |> seq(0.1, by = 0.01) |> round(2)
vec

rounding_bias(x = vec, digits = 1)
rounding_bias(x = vec, digits = 1, rounding = "down")
rounding_bias(x = vec, digits = 1, rounding = "even")
```

---

sum_range *Whole-number sums behind a rounded mean*

Description

sum_range() returns the least and the greatest whole number s for which s / n_items, rounded as specified, would be reported as x_num. Every whole number between the two qualifies as well.

This is the arithmetic behind the GRIM test (Brown and Heathers 2017): a mean of n_items whole numbers is GRIM-consistent if the range is not empty. The same goes for any other share of a whole, such as a percentage of a sample.

Usage

```
sum_range(x_num, n_items, digits, rounding, threshold, symmetric = FALSE)
```

Arguments

x_num	Numeric (length 1). The rounded number. Not checked.
n_items	Numeric (length 1). The number of values averaged; a positive whole number. Not checked.
digits	Integer (length 1). Number of decimal places in x_num. Not checked.
rounding	String (length 1). Rounding method presumably used to create x_num. See the Rounding section of unround().
threshold	Numeric (length 1). The point within a step at which rounding switches direction, in tenths of a step, for the "up_from", "down_from", and "up_from_or_down_from" methods. See round_up_from().
symmetric	Logical (length 1). Whether the rounding of negative numbers mirrored that of positive numbers. See round_up().

Details

Like bound_numerators(), on which it builds, sum_range() works in exact integer arithmetic. This is exact as long as (|x_num| * 10^(digits + 1) + 10) * n_items stays below 2^53, which is far beyond any realistic input. The bounds reach up to 10 units past x_num, hence the + 10. A threshold with decimal places multiplies this by the scale described in bound_numerators().

sum_range() is a low-level primitive, meant to be called once per value set inside a consistency test, so it does not check its input. The caller is responsible for passing single numbers, a whole-number digits that fits x_num, and a whole-number n_items. A fractional n_items returns a range for that fraction rather than an error or NA. Only rounding, threshold, and symmetric are checked, by bound_numerators().

Value

A numeric vector of length 2, c(lower, upper). If lower is greater than upper, no whole-number sum is consistent with x_num. Both are NA if x_num is missing or infinite, if digits is missing, or if n_items is missing, infinite, or not positive.

References

Brown, N. J. L., & Heathers, J. A. J. (2017). The GRIM Test: A Simple Technique Detects Numerous Anomalies in the Reporting of Results in Psychology. *Social Psychological and Personality Science*, 8(4), 363-369. doi:10.1177/1948550616673876

Examples

```
# A mean of 5.18 from 28 whole numbers requires their sum to be 145:
sum_range(5.18, 28, digits = 2, rounding = "up_or_down", threshold = 5)

# No sum gives 5.19: 145 / 28 is 5.18, and 146 / 28 is 5.21:
sum_range(5.19, 28, digits = 2, rounding = "up_or_down", threshold = 5)

# 34.5% of a sample of 58 is 20 people (or 0.345 as a share):
sum_range(0.345, 58, digits = 3, rounding = "up_or_down", threshold = 5)
```

unround *Reconstruct rounding bounds*

Description

`unround()` takes a rounded number and returns the range of values that would have been rounded to it: a lower and an upper bound, and whether each is inclusive. The default rounding method is `"up_or_down"`; see *Rounding* for the others.

Usage

```
unround(
  x,
  rounding = "up_or_down",
  threshold = 5,
  digits = NULL,
  symmetric = FALSE
)
```

Arguments

`x`	String or numeric. The rounded number. Must be a string unless `digits` is specified. A missing or infinite `x` has `NA` bounds.
`rounding`	String. The rounding method assumed to have produced `x`. Default is `"up_or_down"`. See *Rounding*.
`threshold`	Numeric. For the `"*_from"` methods, the point within a step at which rounding switches direction, in tenths of a step; greater than `0` and less than `10`. Default is `5`. See `round_up_from()`.
`digits`	Integer. The number of decimal places in `x`. Default is `NULL`, in which case it is counted from the string `x`. Specify it if the number is already known, as a function that uses `unround()` as a helper would.
`symmetric`	Logical. If `TRUE`, negative numbers are assumed to have been rounded like their absolute values under `"up"`, `"down"`, and the `"*_from"` methods. Default is `FALSE`. Must not be `TRUE` with a `"ties_*"` method; see `reround()`.

Details

`x` is a string by default so that its trailing zeros can be counted, since a numeric value loses them. If the number of decimal places is known, pass it as `digits`, and `x` may be numeric. `bound_numerators()` and `sum_range()` derive their bounds from the same internal functions as `unround()`, so they always agree with it.

All arguments are vectorized. Those of length greater than 1 must have the same length and are paired element by element. Pairing `x` with `rounding` in this way gives a warning.

Value

A tibble with the columns `range`, `rounding`, `lower`, `incl_lower`, `x`, `incl_upper`, and `upper`. `range` summarizes the columns from `lower` to `upper`.

Rounding

Whether a bound is inclusive depends on the rounding method. Under `"up"`, a value on the lower bound rounds up to `x`, but a value on the upper bound rounds up past it. The `range` column shows the bounds with the appropriate signs:

Value of `rounding`	**Corresponding range**
`"up_or_down"` (default)	`lower <= x <= upper`
`"up"`, `"ties_up"`	`lower <= x < upper`
`"down"`, `"ties_down"`	`lower < x <= upper`
`"ties_away"` (positive `x`)	`lower <= x < upper`
`"ties_away"` (negative `x`)	`lower < x <= upper`
`"ties_zero"` (positive `x`)	`lower < x <= upper`
`"ties_zero"` (negative `x`)	`lower <= x < upper`
`"ties_away"`, `"ties_zero"` (zero `x`)	`lower < x < upper`
`"even"`, `"ties_even"`	`lower <= x <= upper`
`"ceiling"`	`lower < x = upper`
`"floor"`	`lower = x < upper`
`"ceiling_or_floor"`	`lower < x < upper`
`"trunc"` (positive `x`)	`lower = x < upper`
`"trunc"` (negative `x`)	`lower < x = upper`
`"trunc"` (zero `x`)	`lower < x < upper`
`"anti_trunc"` (positive `x`)	`lower < x = upper`
`"anti_trunc"` (negative `x`)	`lower = x < upper`
`"anti_trunc"` (zero `x`)	`lower = x = upper` (all `0`)
`"up_from"`	`lower <= x < upper`
`"down_from"`	`lower < x <= upper`
`"up_from_or_down_from"`	`lower <= x <= upper`

`threshold` only applies to the three `"*_from"` methods. It moves their range without widening it; each spans exactly one step at every threshold.

The `"ties_*"` methods each name a complete tie-breaking procedure, so giving `symmetric = TRUE` with them is an error; see `round_ties_up()`.

Under `"even"`, both bounds are reported as inclusive. `base::round()` decides a tie on the binary double, so whether a value on a bound rounds to `x` cannot be predicted from its decimal digits. Treating both bounds as inclusive can only widen the range, which is the safe direction for error detection.

See Also

`round_up()` for rounding up or down from 5, and `round_ceiling()` for the one-directional methods.

Examples

```
# By default, `x` may have been rounded up or down:
unround(x = "2.7")

# If it was rounded up, the upper bound is exclusive:
unround(x = "2.7", rounding = "up")

# Likewise with rounding down...
unround(x = "2.7", rounding = "down")

# ...and with `base::round()`:
unround(x = "2.7", rounding = "even")

# One row per input value:
unround(x = c(3.6, "5.20", 5.174))
```

Index

anti_trunc *(*rounding-uncommon*)*, 13

base::round(), *5–7*, *9–11*, *18*
bound_numerators, 2
bound_numerators(), *16, 17*

ceiling(), *14*

floor(), *14*
fractional-rounding, 4

ieee-754, 5

janitor::round_to_fraction(), *4, 5*

reround, 7
reround(), *4–6*, *8*, *15*, *17*
reround_to_fraction
        *(*fractional-rounding*)*, 4
reround_to_fraction_level
        *(*fractional-rounding*)*, 4
resolve_ties_rounding, 8
round_anti_trunc *(*rounding-uncommon*)*, 13
round_anti_trunc(), *6*, *10*, *12*
round_ceiling *(*rounding-uncommon*)*, 13
round_ceiling(), *6*, *8*, *10*, *18*
round_down *(*rounding-common*)*, 9
round_down(), *11*, *14*
round_down_from *(*rounding-common*)*, 9
round_floor *(*rounding-uncommon*)*, 13
round_floor(), *6*, *10*
round_ties_away *(*rounding-ties*)*, 11
round_ties_away(), *6*
round_ties_down *(*rounding-ties*)*, 11
round_ties_even *(*rounding-ties*)*, 11
round_ties_to_away *(*ieee-754*)*, 5
round_ties_to_even *(*ieee-754*)*, 5
round_ties_up *(*rounding-ties*)*, 11
round_ties_up(), *6*, *10*, *18*
round_ties_zero *(*rounding-ties*)*, 11
round_toward_negative *(*ieee-754*)*, 5
round_toward_positive *(*ieee-754*)*, 5
round_toward_zero *(*ieee-754*)*, 5
round_trunc *(*rounding-uncommon*)*, 13
round_trunc(), *6*, *10*, *12*
round_up *(*rounding-common*)*, 9
round_up(), *2*, *8*, *11, 12*, *14*, *16*, *18*
round_up_from *(*rounding-common*)*, 9
round_up_from(), *2*, *6*, *8*, *16, 17*
rounding-common, 9
rounding-ties, 11
rounding-uncommon, 13
rounding_bias, 14

sum_range, 15
sum_range(), *17*

trunc(), *13, 14*

unround, 17
unround(), *2, 3*, *8*, *10*, *12*, *16*