

# Package ‘roundwork’

September 25, 2024

**Title** Rounding Infrastructure

**Version** 0.0.1

**Description** Flexible rounding functions for use in error detection. They were outsourced from the 'scrutiny' package.

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.2

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

**VignetteBuilder** knitr

**Imports** cli, dplyr, magrittr, purrr, rlang, stringr, tibble

**Collate** 'round.R' 'ieee-754.R' 'utils.R' 'reround-to-fraction.R'  
'round-ceil-floor.R' 'reround.R' 'rounding-bias.R' 'unround.R'  
'utils-pipe.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** 2024-09-25 08:30:01 UTC

## Contents

fractional-rounding . . . . .	2
ieee-754 . . . . .	4
reround . . . . .	5
rounding-common . . . . .	6

rounding-uncommon . . . . .	8
rounding_bias . . . . .	10
unround . . . . .	11

<b>Index</b>	<b>14</b>
--------------	-----------

---

fractional-rounding	<i>Generalized rounding to the nearest fraction of a specified denominator</i>
---------------------	--

---

## Description

Two functions that round numbers to specific fractions, not just to the next higher decimal level. They are inspired by `janitor::round_to_fraction()` but feature all the options of `reround()`:

- `reround_to_fraction()` closely follows `janitor::round_to_fraction()` by first rounding to fractions of a whole number, then optionally rounding the result to a specific number of digits in the usual way.
- `reround_to_fraction_level()` rounds to the nearest fraction of a number at the specific decimal level (i.e., number of digits), without subsequent rounding. This is closer to conventional rounding functions.

## 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

<code>x</code>	Numeric. Vector of numbers to be rounded.
<code>denominator</code>	Numeric ( $\geq 1$ ) . <code>x</code> will be rounded to the nearest fraction of denominator. Default is 1.
<code>digits</code>	Numeric (whole numbers).

- In `reround_to_fraction()`: If `digits` is specified, the values resulting from fractional rounding will subsequently be rounded to that many decimal places. If set to "auto", it internally becomes `ceiling(log10(denominator)) + 1`, as in `janitor::round_to_fraction()`. Default is `Inf`, in which case there is no subsequent rounding.
- In `reround_to_fraction_level()`: This function will round to a fraction of the number at the decimal level specified by `digits`. Default is `0`.

rounding, threshold, symmetric

More arguments passed down to `reround()`.

## Value

Numeric vector of the same length as `x` unless rounding is either of "up\_or\_down", "up\_from\_or\_down\_from", and "ceiling\_or\_floor". In these cases, it will always have length 2.

## See Also

`reround()`, which the functions wrap, and `janitor::round_to_fraction()`, part of which they copy.

## Examples

```
#`reround_to_fraction()` rounds `0.4`
# to `0` if `denominator` is `1`, which
# is the usual integer rounding...
reround_to_fraction(0.4, denominator = 1, rounding = "even")

# ...but if `denominator` is `2`, it rounds to the nearest
# fraction of 2, which is `0.5`:
reround_to_fraction(0.4, denominator = 2, rounding = "even")

# Likewise with fractions of 3:
reround_to_fraction(0.25, denominator = 3, rounding = "even")

# The default for `rounding` is to round
# both up and down, as in `reround()`:
reround_to_fraction(0.4, denominator = 2)

# These two rounding procedures differ
# at the tie points:
reround_to_fraction(0.25, denominator = 2)

# `reround_to_fraction_level()`, in contrast,
# uses `digits` to determine some decimal level,
# and then rounds to the closest fraction at
# that 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)
```

## 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 = 0, ...)
```

```
round_ties_to_away(x, digits = 0)
```

```
round_toward_positive(x, digits = 0)
```

```
round_toward_negative(x, digits = 0)
```

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

## Arguments

- |                     |   |
|---------------------|---|
| <code>x</code>      | Numeric. The decimal number to round.   |
| <code>digits</code> | Integer. Number of digits to round <code>x</code> to. Default is 0.                     |
| <code>...</code>    | Only in <code>round_ties_to_even()</code> . Passed down to <code>base::round()</code> . |

## 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.

## Value

Numeric. `x` rounded to `digits`.

## References

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

## 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()` takes one or more intermediate reconstructed values and rounds them in some specific way – namely, the way they are supposed to have been rounded originally, in the process that generated the reported values.

This function provides an interface to all of `scrutiny`'s rounding functions as well as `base::round()`. It is used as a helper within `scrutiny::grim()`, `scrutiny::grimmer()`, and `scrutiny::debit()`; and it might find use in other places for consistency testing or reconstruction of statistical analyses.

## Usage

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

**Arguments**

<code>x</code>	Numeric. Vector of possibly original values.
<code>digits</code>	Integer. Number of decimal places in the reported key values (i.e., mean or percentage within <code>scrutiny::grim()</code> , or standard deviation within <code>scrutiny::grimmer()</code> ).
<code>rounding</code>	String. The rounding method that is supposed to have been used originally. See <code>vignette("rounding-options")</code> . Default is "up_or_down", which returns two values: <code>x</code> rounded up <i>and</i> down.
<code>threshold</code>	Integer. If rounding is set to "up_from", "down_from", or "up_from_or_down_from", threshold must be set to the number from which the reconstructed values should then be rounded up or down. Otherwise irrelevant. Default is 5.
<code>symmetric</code>	Logical. Set <code>symmetric</code> to TRUE if the rounding of negative numbers with "up_or_down", "up", "down", "up_from_or_down_from", "up_from", or "down_from" should mirror that of positive numbers so that their absolute values are always equal. Otherwise irrelevant. Default is FALSE.

**Details**

`reround()` internally calls the appropriate rounding function(s) determined by the rounding argument. See `vignette("rounding-options")` for a complete list of values that rounding can take.

For the specific rounding functions themselves, see documentation at `round_up()`, `round_ceiling()`, and `base::round()`.

**Value**

Numeric vector of length 1 or 2. (It has length 1 unless rounding is "up\_or\_down", "up\_from\_or\_down\_from", or "ceiling\_or\_floor", in which case it has length 2.)

**Examples**

```
# You can specify the rounding procedure:
reround(4.1679, digits = 2, rounding = "up")

# Default is rounding both up and down:
reround(4.1679, digits = 2)
```

---

rounding-common

*Common rounding procedures*


---

**Description**

`round_up()` rounds up from 5, `round_down()` rounds down from 5. Otherwise, both functions work like `base::round()`.

`round_up()` and `round_down()` are special cases of `round_up_from()` and `round_down_from()`, which allow users to choose custom thresholds for rounding up or down, respectively.

## 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 decimal number to round.
digits	Integer. Number of digits to round x to. Default is 0.
threshold	Integer. Only in <code>round_up_from()</code> and <code>round_down_from()</code> . Threshold for rounding up or down, respectively. Value is 5 in <code>round_up()</code> 's internal call to <code>round_up_from()</code> and in <code>round_down()</code> 's internal call to <code>round_down_from()</code> .
symmetric	Logical. Set <code>symmetric</code> to TRUE if the rounding of negative numbers should mirror that of positive numbers so that their absolute values are equal. Default is FALSE.

## Details

These functions differ from `base::round()` mainly insofar as the decision about rounding 5 up or down is not based on the integer portion of x (i.e., no "rounding to even"). Instead, in `round_up_from()`, that decision is determined by the `threshold` argument for rounding up, and likewise with `round_down_from()`. The threshold is constant at 5 for `round_up()` and `round_down()`.

As a result, these functions are more predictable and less prone to floating-point number quirks than `base::round()`. Compare `round_down()` and `base::round()` in the data frame for rounding 5 created in the Examples section below: `round_down()` yields a continuous sequence of final digits from 0 to 9, whereas `base::round()` behaves in a way that can only be explained by floating point issues.

However, this surprising behavior on the part of `base::round()` is not necessarily a flaw (see its documentation, or this vignette: <https://rpubs.com/maechler/Rounding>). In the present version of R (4.0.0 or later), `base::round()` works fine, and the functions presented here are not meant to replace it. Their main purpose as helpers within scrutiny is to reconstruct the computations of researchers who might have used different software. See `vignette("rounding-options")`.

## Value

Numeric. x rounded to digits.

## See Also

`round_ceil()` always rounds up, `round_floor()` always rounds down, `round_trunc()` always rounds toward 0, and `round_anti_trunc()` always round away from 0.

**Examples**

```

# Both `round_up()` and `round_down()` work like
# `base::round()` unless the closest digit to be
# cut off by rounding is 5:

round_up(x = 9.273, digits = 1)    # 7 cut off
round_down(x = 9.273, digits = 1)  # 7 cut off
base::round(x = 9.273, digits = 1) # 7 cut off

round_up(x = 7.584, digits = 2)    # 4 cut off
round_down(x = 7.584, digits = 2)  # 4 cut off
base::round(x = 7.584, digits = 2)  # 4 cut off

# Here is the borderline case of 5 rounded by
# `round_up()`, `round_down()`, and `base::round()`:

original <- c( # Define example values
  0.05, 0.15, 0.25, 0.35, 0.45,
  0.55, 0.65, 0.75, 0.85, 0.95
)
tibble::tibble( # Output table
  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)
)

# (Note: Defining `original` as `seq(0.05:0.95, by = 0.1)`
# would lead to wrong results unless `original` is rounded
# to 2 or so digits before it's rounded to 1.)

```

**Description**

Always round up, down, toward zero, or away from it:

- `round_ceiling()` always rounds up.
- `round_floor()` always rounds down.
- `round_trunc()` always rounds toward zero.
- `round_anti_trunc()` always rounds away from zero.
- `anti_trunc()` returns the integer further away from zero.

Despite not being widely used, they are featured here in case they are needed for reconstruction.



**Usage**

```

round_ceil(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 decimal number to round.
digits	Integer. Number of digits to round x to. Default is 0.

**Details**

`round_ceil()`, `round_floor()`, and `round_trunc()` generalize the base R functions `ceil()`, `floor()`, and `trunc()`, and include them as special cases: With the default value for `digits`, 0, these `round_*` functions are equivalent to their respective base counterparts.

The last `round_*` function, `round_anti_trunc()`, generalizes another function presented here: `anti_trunc()` works like `trunc()` except it moves away from 0, rather than towards it. That is, whereas `trunc()` minimizes the absolute value of `x` (as compared to the other rounding functions), `anti_trunc()` maximizes it. `anti_trunc(x)` is therefore equal to `trunc(x) + 1` if `x` is positive, and to `trunc(x) - 1` if `x` is negative. It only ever returns 0 if `x` is 0; as 0 does not have a sign.

`round_anti_trunc()`, then, generalizes `anti_trunc()` just as `round_ceil()` generalizes `ceil()`, etc.

Moreover, `round_trunc()` is equivalent to `round_floor()` for positive numbers and to `round_ceil()` for negative numbers. The reverse is again true for `round_anti_trunc()`: It is equivalent to `round_ceil()` for positive numbers and to `round_floor()` for negative numbers.

**Value**

Numeric. `x` rounded to `digits` (except for `anti_trunc()`, which has no `digits` argument).

**See Also**

`round_up()` and `round_down()` round up or down from 5, respectively. `round_up_from()` and `round_down_from()` allow users to specify custom thresholds for rounding up or down.

**Examples**

```

# Always round up:
round_ceil(x = 4.52, digits = 1)      # 2 cut off

# Always round down:
round_floor(x = 4.67, digits = 1)    # 7 cut off

```

```
# Always round toward 0:
round_trunc(8.439, digits = 2)      # 9 cut off
round_trunc(-8.439, digits = 2)    # 9 cut off

# Always round away from 0:
round_anti_trunc(x = 8.421, digits = 2) # 1 cut off
round_anti_trunc(x = -8.421, digits = 2) # 1 cut off
```

---

rounding\_bias

*Compute rounding bias*


---

## Description

Rounding often leads to bias, such that the mean of a rounded distribution is different from the mean of the original distribution. Call `rounding_bias()` to compute the amount of this bias.

## Usage

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

## Arguments

<code>x</code>	Numeric or string coercible to numeric.
<code>digits</code>	Integer. Number of decimal digits to which <code>x</code> will be rounded.
<code>rounding</code>	String. Rounding procedure that will be applied to <code>x</code> . See <code>vignette("rounding-options")</code> . Default is "up".
<code>threshold, symmetric</code>	Further arguments passed down to <code>reround()</code> .
<code>mean</code>	Logical. If TRUE (the default), the mean total of bias will be returned. Set <code>mean</code> to FALSE to get a vector of individual biases the length of <code>x</code> .

## Details

Bias is calculated by subtracting the original vector, `x`, from a vector rounded in the specified way.

The function passes all arguments except for `mean` down to `reround()`. Other than there, however, `rounding` is "up" by default, and it can't be set to "up\_or\_down", "up\_from\_or\_down\_from", or "ceiling\_or\_floor".

**Value**

Numeric. By default of mean, the length is 1; otherwise, it is the same length as *x*.

**Examples**

```
# Define example vector:
vec <- seq(from = 0.01, to = 0.1, by = 0.01)
vec

# The default rounds `x` up from 5:
rounding_bias(x = vec, digits = 1)

# Other rounding procedures are supported,
# such as rounding down from 5...
rounding_bias(x = vec, digits = 1, rounding = "down")

# ...or rounding to even with `base::round()`:
rounding_bias(x = vec, digits = 1, rounding = "even")
```

---

unround

*Reconstruct rounding bounds*


---

**Description**

`unround()` takes a rounded number and returns the range of the original value: lower and upper bounds for the hypothetical earlier number that was later rounded to the input number. It also displays a range with inequation signs, showing whether the bounds are inclusive or not.

By default, the presumed rounding method is rounding up (or down) from 5. See the Rounding section for other methods.

**Usage**

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

**Arguments**

<code>x</code>	String or numeric. Rounded number. <i>x</i> must be a string unless <code>digits</code> is specified (most likely by a function that uses <code>unround()</code> as a helper).
<code>rounding</code>	String. Rounding method presumably used to create <i>x</i> . Default is "up_or_down". For more, see section Rounding.
<code>threshold</code>	Integer. Number from which to round up or down. Other rounding methods are not affected. Default is 5.
<code>digits</code>	Integer. This argument is meant to make <code>unround()</code> more efficient to use as a helper function so that it doesn't need to redundantly count decimal places. Don't specify it otherwise. Default is NULL, in which case decimal places really are counted internally and <i>x</i> must be a string.

## Details

The function is vectorized over  $x$  and rounding. This can be useful to unround multiple numbers at once, or to check how a single number is unrounded with different assumed rounding methods.

If both vectors have a length greater than 1, it must be the same length. However, this will pair numbers with rounding methods, which can be confusing. It is recommended that at least one of these input vectors has length 1.

Why does  $x$  need to be a string if `digits` is not specified? In that case, `unround()` must count decimal places by itself. If  $x$  then was numeric, it wouldn't have any trailing zeros because these get dropped from numerics.

Trailing zeros are as important for reconstructing boundary values as any other trailing digits would be. Strings don't drop trailing zeros, so they are used instead.

## Value

A tibble with seven columns: `range`, `rounding`, `lower`, `incl_lower`, `x`, `incl_upper`, and `upper`. The `range` column is a handy representation of the information stored in the columns from `lower` to `upper`, in the same order.

## Rounding

Depending on how  $x$  was rounded, the boundary values can be inclusive or exclusive. The `incl_lower` and `incl_upper` columns in the resulting tibble are `TRUE` in the first case and `FALSE` in the second. The `range` column reflects this with equation and inequation signs.

However, these ranges are based on assumptions about the way  $x$  was rounded. Set `rounding` to the rounding method that hypothetically lead to  $x$ :

Value of rounding	Corresponding range
"up_or_down" (default)	$\text{lower} \leq x \leq \text{upper}$
"up"	$\text{lower} \leq x < \text{upper}$
"down"	$\text{lower} < x \leq \text{upper}$
"even"	(no fix range; NA)
"ceiling"	$\text{lower} < x = \text{upper}$
"floor"	$\text{lower} = x < \text{upper}$
"trunc" (positive $x$ )	$\text{lower} = x < \text{upper}$
"trunc" (negative $x$ )	$\text{lower} < x = \text{upper}$
"trunc" (zero $x$ )	$\text{lower} < x < \text{upper}$
"anti_trunc" (positive $x$ )	$\text{lower} < x = \text{upper}$
"anti_trunc" (negative $x$ )	$\text{lower} = x < \text{upper}$
"anti_trunc" (zero $x$ )	(undefined; error)

Base R's own `round()` (R version  $\geq 4.0.0$ ), referenced by `rounding = "even"`, is reconstructed in the same way as "up\_or\_down", but whether the boundary values are inclusive or not is hard to predict. Therefore, `unround()` checks if they are, and informs you about it.

**See Also**

For more about rounding "up", "down", or to "even", see [round\\_up\(\)](#).

For more about the less likely rounding methods, "ceiling", "floor", "trunc", and "anti\_trunc", see [round\\_ceiling\(\)](#).

**Examples**

```
# By default, the function assumes that `x`
# was either rounded up or down:
unround(x = "2.7")

# If `x` was rounded up, run this:
unround(x = "2.7", rounding = "up")

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

# ...and with `base::round()` which, broadly
# speaking, rounds to the nearest even number:
unround(x = "2.7", rounding = "even")

# Multiple input number-strings return
# multiple rows in the output data frame:
unround(x = c(3.6, "5.20", 5.174))
```

# Index

anti\_trunc (rounding-uncommon), 8  
base::round(), 4–7  
ceiling(), 9  
floor(), 9  
fractional-rounding, 2  
ieee-754, 4  
janitor::round\_to\_fraction(), 2, 3  
reround, 5  
reround(), 2, 3, 10  
reround\_to\_fraction  
    (fractional-rounding), 2  
reround\_to\_fraction\_level  
    (fractional-rounding), 2  
round\_anti\_trunc (rounding-uncommon), 8  
round\_anti\_trunc(), 7  
round\_ceiling (rounding-uncommon), 8  
round\_ceiling(), 6, 7, 13  
round\_down (rounding-common), 6  
round\_down(), 9  
round\_down\_from (rounding-common), 6  
round\_down\_from(), 9  
round\_floor (rounding-uncommon), 8  
round\_floor(), 7  
round\_ties\_to\_away (ieee-754), 4  
round\_ties\_to\_even (ieee-754), 4  
round\_toward\_negative (ieee-754), 4  
round\_toward\_positive (ieee-754), 4  
round\_toward\_zero (ieee-754), 4  
round\_trunc (rounding-uncommon), 8  
round\_trunc(), 7  
round\_up (rounding-common), 6  
round\_up(), 6, 9, 13  
round\_up\_from (rounding-common), 6  
round\_up\_from(), 9  
rounding-common, 6  
rounding-uncommon, 8  
rounding\_bias, 10  
scrutiny::debit(), 5  
scrutiny::grim(), 5, 6  
scrutiny::grimmer(), 5, 6  
trunc(), 9  
unround, 11