Package 'decimal'

August 24, 2026

Title Exact Arbitrary-Precision Decimal Vectors

Version 0.1.0

Description Arbitrary-precision vectors with an exact decimal representation, avoiding the rounding surprises of binary floating point. Built on the 'mpdecimal' C library, arithmetic is governed by an explicit decimal context controlling precision, rounding, and signalling, and vectors integrate with 'vctrs' for use in data frames, 'tibble' objects, summaries, and common numeric workflows. Missing values, signed zeros, infinities, and not-a-number values are supported throughout. The arithmetic model follows Cowlishaw (2009) ``General Decimal Arithmetic'' <https://speleotrove.com/decimal/decarith.html>.

License MIT + file LICENSE

Language en-US

Depends R (>= 4.2.0)

Imports methods, rlang, vctrs, withr

Suggests covr, knitr, pillar, rmarkdown, tibble, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/Needs/documentation roxygen2

Encoding UTF-8

NeedsCompilation yes

Config/roxygen2/version 8.0.0

Author Pedro Baltazar [aut, cre, cph],
Stefan Krah [ctb, cph] (Vendored 'mpdecimal' library in src/mpdecimal; see inst/COPYRIGHTS.)

Maintainer Pedro Baltazar <pedrobtz@gmail.com>

Repository CRAN

Date/Publication 2026-08-24 12:40:09 UTC

Contents

`adjusted` *Compute the adjusted exponent*

Description

`adjusted()` returns the position of the most significant digit after accounting for the stored exponent. For a finite nonzero value, this is equivalent to the base-10 order of magnitude. The representation of zero retains its exponent, so differently scaled zeros can have different adjusted exponents.

Usage

```
adjusted(x)
```

Arguments

`x`	A decimal-compatible vector.

Value

An integer vector with the same length as `x`. Infinities, NaNs, and R missing values produce `NA_integer_`.

Examples

```
adjusted(decimal(c("123", "0.01")))
```

`as_decimal` *Convert a vector to decimal*

Description

`as_decimal()` is the conversion generic for decimal vectors. Character values are parsed exactly, integer values are converted exactly, and existing decimal vectors are returned or rescaled. Promotion to a finer shared scale is exact and context-free; reduction to a coarser scale is quantized. Double values are decoded from their exact IEEE 754 representation and therefore require an explicit or globally configured scale before quantization.

Usage

```
as_decimal(x, scale = NULL)

## S3 method for class 'decimal'
as_decimal(x, scale = NULL)

## S3 method for class 'character'
as_decimal(x, scale = NULL)

## S3 method for class 'integer'
as_decimal(x, scale = NULL)

## S3 method for class 'numeric'
as_decimal(x, scale = NULL)

## Default S3 method:
as_decimal(x, scale = NULL)
```

Arguments

`x`	A decimal, character, integer, or double vector.
`scale`	An integer scalar giving the number of fractional digits to store, or `NULL` to use the global default or input-specific inference.

Details

When `scale` is `NULL`, `getOption("decimal.default_scale")` is used when set. Without that option, character input uses the largest number of fractional digits found in the input, integer input uses scale zero, and double input raises an error. Quantization uses the active `decimal_context()`.

Value

A `decimal` vector.

Examples

```
as_decimal(c("1.2", "3.45"))
```

`clear_decimal_flags` *Clear sticky decimal flags*

Description

Clear sticky decimal flags

Usage

```
clear_decimal_flags()
```

Value

The previously active sticky flags, invisibly.

Examples

```
clear_decimal_flags()
```

`decimal` *Construct decimal vectors*

Description

`decimal()` creates an immutable decimal vector backed by exact strings, at a single shared scale (number of fractional digits) for the whole vector. Character and integer inputs are parsed exactly. Promotion to a finer shared scale only appends zeros and is context-free; a requested coarser scale quantizes using the active context. Double inputs are decoded from their exact IEEE 754 binary value and require either an explicit `scale` or the `decimal.default_scale` option before quantization.

Usage

```
decimal(x = character(), scale = NULL)
```

Arguments

x	A decimal, character, integer, or double vector.
scale	An integer scalar giving the number of fractional digits to store. NULL uses the decimal.default_scale option when set, then falls back to input-specific inference. Negative values round into the integer part (e.g. scale = -2 rounds to the nearest hundred).

Details

When scale is NULL, getOption("decimal.default_scale") is used when set. Otherwise, character input infers the largest number of fractional digits present in x, and integer input uses scale zero. Because scale is a property of the vector, not the element, decimal("1.2") == decimal("1.20"), and combining them yields one uniformly scaled vector.

Value

A decimal vector.

Examples

```
decimal(c("1.20", "2.30"))
```

---

decimal_context *Create a decimal arithmetic context*

Description

Constructs a validated arithmetic context for native mpdecimal operations. The context controls precision, rounding, exponent limits, traps, sticky flags, and classification of normal versus subnormal values.

Usage

```
decimal_context(
  precision = 28L,
  rounding = "half_even",
  emax = 999999L,
  emin = -999999L,
  traps = decimal_default_traps(),
  flags = character(),
  clamp = FALSE
)
```

Arguments

precision	Integer scalar precision.
rounding	One of "up", "down", "ceiling", "floor", "half_up", "half_down", "half_even", or "05up".
emax	Integer scalar maximum exponent.
emin	Integer scalar minimum exponent.
traps	Character vector of trapped signals. Trapped signals raise an error; all other raised signals are recorded as flags and, unless options(decimal.report_flags = FALSE), reported as warnings.
flags	Character vector of sticky signal flags.
clamp	Logical scalar clamp mode.

Details

Each operation may raise one or more signals (see decimal_flags()). Their disposition depends on traps:

- A raised signal that is in traps stops the operation with an error.
- Any other raised signal is recorded as a sticky flag and, by default, also surfaced as a warning of class decimal_flags_warning.

The warnings are purely informational; sticky flags accumulate either way. Set options(decimal.report_flags = FALSE) to silence them and rely on decimal_flags() alone.

Public signal names are "clamped", "division_by_zero", "inexact", "invalid_operation", "overflow", "rounded", "subnormal", and "underflow". The standard invalid_operation condition groups lower-level invalid subconditions such as undefined division (0 / 0).

A few operations are exempt from the warning because inexact/rounded is their guaranteed, expected outcome rather than a surprise: quantize() (and round()/signif(), built on it), and sqrt(), exp(), log(), and log10(), which are irrational for nearly every input. These still accumulate sticky flags as usual.

Value

A decimal_context object.

Examples

```
decimal_context(precision = 10L)
decimal_context(precision = 3L, rounding = "floor", traps = character())
```

decimal_flags *Read sticky decimal flags*

Description

Sticky flags accumulate the signals raised by operations that were not trapped (see `decimal_context()`). They persist until `clear_decimal_flags()` is called. By default the same non-trapped signals are also reported as warnings as they occur; set `options(decimal.report_flags = FALSE)` to silence the warnings and inspect flags only through this function.

Usage

```
decimal_flags()
```

Value

A character vector of active sticky flags.

Examples

```
decimal_flags()
```

decimal_from_double *Controlled conversion from double*

Description

Decodes each IEEE 754 double to its exact decimal value, then quantizes that value to the requested scale. This differs from parsing a character literal such as `"0.1"`. Because the exact binary value of a double can require dozens of fractional digits, a scale must be given explicitly or configured with `options(decimal.default_scale = )`. Quantization uses the active context's rounding and trap settings.

Usage

```
decimal_from_double(x, scale = NULL)
```

Arguments

`x`	A double vector.
`scale`	An integer scalar giving the number of fractional digits to store, or `NULL` to use `getOption("decimal.default_scale")`.

Value

A `decimal` vector.

Examples

```
decimal_from_double(0.1, scale = 20)
decimal_from_double(c(0.5, 0.25), scale = 2)
```

---

fma *Fused multiply-add*

Description

`fma()` computes `x * y + z` with a single final rounding step. This can be more accurate than evaluating multiplication and addition separately under a limited-precision context. The arguments follow normal vctrs recycling rules.

Usage

```
fma(x, y, z)
```

Arguments

`x`, `y`, `z`	Decimal-compatible vectors.

Value

A `decimal` vector.

Examples

```
fma(decimal("2"), decimal("3"), decimal("4"))
```

---

get_decimal_context *Get the active decimal arithmetic context*

Description

Get the active decimal arithmetic context

Usage

```
get_decimal_context()
```

Value

A `decimal_context` object.

Examples

```
get_decimal_context()
```

is_decimal *Test whether an object is a decimal vector*

Description

`is_decimal()` reports whether `x` inherits from the `decimal` vector class. It does not attempt to parse or convert other objects.

Usage

```
is_decimal(x)
```

Arguments

`x`	An object to test.

Value

A single logical value.

Examples

```
is_decimal(decimal("1.5"))
```

is_normal *Identify normal decimal values*

Description

`is_normal()` reports whether each finite, nonzero value is normal under the active decimal context. Normality depends on the context's exponent limits and precision.

Usage

```
is_normal(x)
```

Arguments

`x`	A decimal-compatible vector.

Value

A logical vector with the same length as `x`.

Examples

```
is_normal(decimal(c("1", "0", "Infinity")))
```

is_qnan *Identify quiet NaN values*

Description

`is_qnan()` identifies quiet not-a-number values. Signaling NaNs and R missing values are not quiet NaNs.

Usage

```
is_qnan(x)
```

Arguments

`x`	A decimal-compatible vector.

Value

A logical vector with the same length as `x`.

Examples

```
is_qnan(decimal(c("NaN", "sNaN", "1")))
```

is_signed *Identify values with a negative sign*

Description

`is_signed()` inspects the stored sign bit rather than comparing with zero. It therefore identifies negative zero as signed.

Usage

```
is_signed(x)
```

Arguments

`x`	A decimal-compatible vector.

Value

A logical vector with the same length as `x`.

Examples

```
is_signed(decimal(c("-2", "2", "-0", "0")))
```

is_snan *Identify signaling NaN values*

Description

`is_snan()` identifies signaling not-a-number values without performing an arithmetic operation or raising the `invalid_operation` signal.

Usage

```
is_snan(x)
```

Arguments

`x`	A decimal-compatible vector.

Value

A logical vector with the same length as `x`.

Examples

```
is_snan(decimal(c("sNaN", "NaN", "1")))
```

is_subnormal *Identify subnormal decimal values*

Description

`is_subnormal()` reports whether each finite, nonzero value is subnormal under the active decimal context. A value can therefore be subnormal in one context and normal in another.

Usage

```
is_subnormal(x)
```

Arguments

`x`	A decimal-compatible vector.

Value

A logical vector with the same length as `x`.

Examples

```
x <- decimal("0.001")
with_decimal_context(
  decimal_context(precision = 3L, emin = -2L),
  is_subnormal(x)
)
```

`is_zero` *Identify decimal zeros*

Description

`is_zero()` identifies both positive and negative zero, regardless of the vector's scale.

Usage

```
is_zero(x)
```

Arguments

`x`	A decimal-compatible vector.

Value

A logical vector with the same length as `x`.

Examples

```
is_zero(decimal(c("0.00", "-0", "1")))
```

`local_decimal_context` *Install a decimal context for the current scope*

Description

Install a decimal context for the current scope

Usage

```
local_decimal_context(x, .local_envir = parent.frame())
```

Arguments

`x`	A `decimal_context` object or compatible list.
`.local_envir`	Environment whose scope should control restoration.

Value

x, invisibly.

Examples

```
f <- function() {
  local_decimal_context(decimal_context(precision = 2L, traps = character()))
  decimal("1.234") + decimal("0")
}
f()
```

---

mpdecimal_version *Report the bundled mpdecimal runtime version*

Description

Returns the version string reported by the bundled mpdecimal library that was loaded with the package DLL.

Usage

```
mpdecimal_version()
```

Value

A length-one character vector.

Examples

```
mpdecimal_version()
```

NA_decimal_ *Missing decimal scalar*

Description

A length-one missing decimal value.

Usage

```
NA_decimal_
```

Value

A length-one decimal vector containing the typed R missing value.

Examples

```
c(decimal("1"), NA_decimal_)
```

normalize *Remove unnecessary trailing zeros*

Description

`normalize()` reduces each finite value to its shortest equivalent decimal representation, then chooses the finest scale required by any element so the result remains a valid shared-scale decimal vector. Special values pass through unchanged.

Usage

```
normalize(x)
```

Arguments

`x`	A decimal-compatible vector.

Value

A normalized `decimal` vector.

Examples

```
normalize(decimal(c("1.2300", "1.2")))
```

number_class *Classify decimal values*

Description

`number_class()` returns the General Decimal Arithmetic class of each value. Possible finite classes include `"+Normal"`, `"-Normal"`, `"+Subnormal"`, `"-Subnormal"`, `"+Zero"`, and `"-Zero"`; infinities and NaNs have their corresponding class names. Normal and subnormal classes depend on the active decimal context.

Usage

```
number_class(x)
```

Arguments

`x`	A decimal-compatible vector.

Value

A character vector with the same length as `x`. R missing values produce `NA_character_`.

Examples

```
number_class(decimal(c("1", "-0", "Infinity", "NaN")))
```

`quantize` *Quantize decimal values to a scale*

Description

`quantize()` rounds each value in `x` to the scale declared by `quantum`. The operation uses the active context's rounding mode, updates sticky flags, and raises any enabled traps. The arguments follow normal vctrs recycling rules.

Usage

```
quantize(x, quantum)
```

Arguments

`x`	A decimal-compatible vector to quantize.
`quantum`	A decimal-compatible vector whose shared scale determines the result scale.

Details

Reducing the scale is the explicit purpose of `quantize()` (and of `round()` and `signif()`, both implemented on top of it), so the `inexact` and `rounded` signals this commonly raises are not reported as warnings the way other operations' signals are (see `decimal_context()`); they still accumulate as sticky flags.

Value

A `decimal` vector with the shared scale of `quantum`.

Examples

```
quantize(decimal("1.23456"), decimal("0.01"))
```

same_quantum *Compare decimal vector scales*

Description

same_quantum() tests whether x and y have the same shared vector scale. Scale is a vector-level property in this package, so every recycled element comparison receives the same result.

Usage

```
same_quantum(x, y)
```

Arguments

x, y	Decimal-compatible vectors.

Value

A logical vector with the common recycled size of x and y.

Examples

```
same_quantum(decimal("1.00"), decimal("2.0"))
```

set_decimal_context *Set the active decimal arithmetic context*

Description

Set the active decimal arithmetic context

Usage

```
set_decimal_context(x)
```

Arguments

x	A decimal_context object or compatible list.

Value

The previously active decimal_context, invisibly.

Examples

```
old <- set_decimal_context(decimal_context(precision = 5L))
set_decimal_context(old)
```

`with_decimal_context` *Use a decimal context within a block*

Description

Use a decimal context within a block

Usage

```
with_decimal_context(x, code)
```

Arguments

x	A `decimal_context` object or compatible list.
code	Code evaluated with `x` installed as the active context.

Value

The result of `code`.

Examples

```
with_decimal_context(
  decimal_context(precision = 3L, traps = character()),
  decimal("1.25") + decimal("0")
)
```

Index