Package 'numops'

September 8, 2026

Title Lightweight Numerical Operations

Version 1.0.0

URL https://github.com/Macosso/numops

BugReports https://github.com/Macosso/numops/issues

Description Provides dependency-free helpers for recurring numerical tasks on vectors, matrices, and arrays. Operations cover bounds, interpolation, remapping, division, Euclidean norms, normalization, and adjacent differences. Multi-input operations use strict scalar recycling, reject incompatible lengths, and preserve names, dimensions, and dimension names where applicable. Explicit handling of invalid intervals, zero denominators, and zero norms gives consistent behavior for common edge cases.

License GPL-3

Encoding UTF-8

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

NeedsCompilation no

Author Joao Claudio Macosso [aut, cre] (ORCID: <https://orcid.org/0009-0006-5051-9312>)

Maintainer Joao Claudio Macosso <joaoclaudiomacosso@gmail.com>

Repository CRAN

Date/Publication 2026-09-08 13:40:18 UTC

Contents

`adjacent_difference` *Adjacent differences with preserved length*

Description

Keeps the first value and replaces every later value with its change from the preceding value.

Usage

```
adjacent_difference(x)
```

Arguments

`x`	A numeric vector in the order in which differences are required.

Details

For nonempty `x`, the result satisfies `result[1] = x[1]` and `result[i] = x[i] - x[i - 1]` for later positions. Unlike `diff()`, the first value is retained, so the input can be recovered with `cumsum()`. An empty input is returned unchanged.

Value

A numeric vector with the same length and names as `x`.

Examples

```
adjacent_difference(c(10, 13, 12))
```

clamp *Clamp values to an interval*

Description

Replaces values below `lower` with `lower` and values above `upper` with `upper`.

Usage

```
clamp(x, lower, upper)
```

Arguments

`x`	A numeric vector, matrix, or array containing values to restrict.
`lower`	A numeric lower bound giving the smallest permitted value.
`upper`	A numeric upper bound giving the greatest permitted value.

Details

Each result is computed as `min(max(x, lower), upper)`. Missing values in `x` are preserved. Bounds may be infinite, but must not be missing or have `lower > upper`.

Value

A numeric vector, matrix, or array of the shared length. It takes names, dimensions, and dimnames from the first input already having that length.

Recycling

`x`, `lower`, and `upper` must each have length one or a shared length. Length-one inputs are recycled; other length combinations are errors. Names, dimensions, and dimnames come from the first input with the shared length.

Examples

```
clamp(c(-1, 0.5, 2), 0, 1)
```

clamp01 *Clamp values to the unit interval*

Description

Replaces values below zero with zero and values above one with one.

Usage

```
clamp01(x)
```

Arguments

x	A numeric vector, matrix, or array containing values to restrict.

Details

This is equivalent to `clamp(x, 0, 1)`, or element-wise to `min(max(x, 0), 1)`. Missing values are preserved.

Value

A numeric vector, matrix, or array with the same length, names, dimensions, and dimnames as `x`.

Examples

```
clamp01(c(-0.2, 0.4, 1.3))
```

divide_or *Divide with a fallback value*

Description

Divides corresponding values and substitutes a chosen result wherever the denominator is zero.

Usage

```
divide_or(x, y, default = NA_real_)
```

Arguments

x	A numeric numerator.
y	A numeric denominator.
default	The numeric value to return where `y == 0`. The default is `NA_real_`.

Details

The result is `x / y` where `y != 0`, and `default` where `y == 0`. Missing denominators and other non-finite results follow ordinary R division; they do not trigger `default`.

Value

A numeric vector, matrix, or array of the shared length. It takes names, dimensions, and dimnames from the first input already having that length.

Recycling

`x`, `y`, and `default` must each have length one or a shared length. Length-one inputs are recycled; other length combinations are errors. Names, dimensions, and dimnames come from the first input with the shared length.

Examples

```
divide_or(c(1, 2, 0), c(1, 0, 0), default = NA_real_)
```

`inv_lerp` *Inverse linear interpolation*

Description

Calculates how far `x` lies from `a` toward `b`.

Usage

```
inv_lerp(a, b, x)
```

Arguments

`a`	A numeric endpoint corresponding to a result of zero.
`b`	A numeric endpoint corresponding to a result of one. It must differ from `a` at every non-missing position.
`x`	A numeric vector, matrix, or array containing values to locate.

Details

Each result is `(x - a) / (b - a)`, calculated to avoid unnecessary overflow for widely separated endpoints. Results outside `[0, 1]` indicate that `x` lies outside the endpoints. Infinite endpoints are not supported, and missing values are propagated.

Value

A numeric vector, matrix, or array of the shared length. It takes names, dimensions, and dimnames from the first input already having that length.

Recycling

a, b, and x must each have length one or a shared length. Length-one inputs are recycled; other length combinations are errors. Names, dimensions, and dimnames come from the first input with the shared length.

Examples

```
inv_lerp(10, 20, c(10, 15, 20))
```

---

in_range *Test whether values are in an interval*

Description

Tests whether each value falls in the closed interval from lower to upper.

Usage

```
in_range(x, lower, upper)
```

Arguments

x	A numeric vector, matrix, or array containing values to test.
lower	A numeric inclusive lower bound.
upper	A numeric inclusive upper bound.

Details

Each result is computed as x >= lower & x <= upper. Missing values in x produce missing results. Bounds may be infinite, but must not be missing or have lower > upper.

Value

A logical vector, matrix, or array of the shared length. It takes names, dimensions, and dimnames from the first input already having that length.

Recycling

x, lower, and upper must each have length one or a shared length. Length-one inputs are recycled; other length combinations are errors. Names, dimensions, and dimnames come from the first input with the shared length.

Examples

```
in_range(1:5, 2, 4)
```

l2_norm *Euclidean norm*

Description

Computes Euclidean lengths for a complete object or for slices selected by `margin`.

Usage

```
l2_norm(x, margin = NULL)
```

Arguments

`x`	A numeric vector, matrix, or array.
`margin`	An integer vector naming the dimensions that index separate slices, or `NULL` to treat all elements as one vector. For a matrix, `margin = 1` computes row norms and `margin = 2` computes column norms.

Details

For a slice with values `x[i]`, the L2 norm is `sqrt(sum(x[i]^2))`. The calculation is scaled to avoid unnecessary overflow and underflow. An empty slice has norm zero, an infinite value produces an infinite norm, and a missing value produces a missing norm.

Value

If `margin` is `NULL`, one numeric value. Otherwise, a numeric vector or array indexed by `dim(x)[margin]`, with the corresponding dimnames.

Examples

```
l2_norm(c(3, 4))
l2_norm(matrix(1:6, nrow = 2), margin = 1)
```

lerp *Linear interpolation*

Description

Computes the value a proportion `t` of the way from `a` to `b`.

Usage

```
lerp(a, b, t)
```

Arguments

a	A numeric endpoint returned when t = 0.
b	A numeric endpoint returned when t = 1.
t	A numeric interpolation proportion, usually between zero and one.

Details

Each result is computed as a + t * (b - a), using a calculation that avoids unnecessary overflow when a and b have opposite signs. Values of t outside [0, 1] extrapolate. The endpoints are returned exactly when t is zero or one.

Value

A numeric vector, matrix, or array of the shared length. It takes names, dimensions, and dimnames from the first input already having that length.

Recycling

a, b, and t must each have length one or a shared length. Length-one inputs are recycled; other length combinations are errors. Names, dimensions, and dimnames come from the first input with the shared length.

Examples

```
lerp(10, 20, c(0, 0.25, 1))
```

---

midpoint *Midpoint between values*

Description

Computes the value halfway between corresponding values in x and y.

Usage

```
midpoint(x, y)
```

Arguments

x	The first numeric endpoint.
y	The second numeric endpoint.

Details

The mathematical result is (x + y) / 2. The implementation uses equivalent forms chosen to avoid unnecessary overflow for finite values. Missing and infinite values follow ordinary R arithmetic.

Value

A numeric vector, matrix, or array of the shared length. It takes names, dimensions, and dimnames from the first input already having that length.

Recycling

`x` and `y` must each have length one or a shared length. Length-one inputs are recycled; other length combinations are errors. Names, dimensions, and dimnames come from the first input with the shared length.

Examples

```
midpoint(c(0, 10), c(10, 20))
```

`normalize_l2` *Normalize to unit Euclidean length*

Description

Divides a numeric object, or each selected slice, by its Euclidean norm.

Usage

```
normalize_l2(x, margin = NULL, zero = c("keep", "na", "error"))
```

Arguments

`x`	A numeric vector, matrix, or array to normalize.
`margin`	An integer vector naming the dimensions that index separate slices, or `NULL` to normalize all elements together. For a matrix, `margin = 1` normalizes rows and `margin = 2` normalizes columns.
`zero`	How to handle a slice whose norm is zero. `"keep"` leaves the slice unchanged, `"na"` replaces it with missing values, and `"error"` stops the calculation.

Details

Each slice `s` is transformed to `s / l2_norm(s)`. Nonzero finite slices therefore have an L2 norm of one. A missing value makes its entire slice missing. Infinite values follow ordinary division by an infinite norm.

Value

A numeric vector, matrix, or array with the same length, names, dimensions, and dimnames as `x`.

Examples

```
normalize_l2(c(3, 4))
normalize_l2(matrix(1:6, nrow = 2), margin = 1)
```

`remap` *Remap values between intervals*

Description

Linearly maps values from the interval `from` to the interval `to`.

Usage

```
remap(x, from, to)
```

Arguments

`x`	A numeric vector, matrix, or array containing values to map.
`from`	A finite numeric vector of length two giving the input endpoints.
`to`	A finite numeric vector of length two giving the output endpoints.

Details

The result is `to[1] + (x - from[1]) / (from[2] - from[1]) * (to[2] - to[1])`. Values outside `from` are extrapolated. Either interval may be reversed, but the endpoints of `from` must differ.

Value

A numeric vector, matrix, or array with the same length, names, dimensions, and dimnames as `x`.

Examples

```
remap(c(0, 5, 10), c(0, 10), c(-1, 1))
```

wrap *Wrap values to a periodic interval*

Description

Periodically maps values to the half-open interval `[lower, upper)`.

Usage

```
wrap(x, lower, upper)
```

Arguments

`x`	A numeric vector, matrix, or array containing values to wrap.
`lower`	A finite numeric lower boundary included in the result.
`upper`	A finite numeric upper boundary excluded from the result.

Details

The operation is equivalent to `lower + (x - lower) %% (upper - lower)`, using an overflow-resistant calculation. Bounds must be finite with `lower < upper`. A value equal to `upper` maps to `lower`; infinite values in `x` produce `NaN`.

Value

A numeric vector, matrix, or array of the shared length. It takes names, dimensions, and dimnames from the first input already having that length.

Recycling

`x`, `lower`, and `upper` must each have length one or a shared length. Length-one inputs are recycled; other length combinations are errors. Names, dimensions, and dimnames come from the first input with the shared length.

Examples

```
wrap(c(-10, 0, 370), 0, 360)
```

Index