Package 'RobustArithmetic'

September 16, 2026

Type Package

Title Verified Interval Arithmetic with Correctly Rounded Kernels

Version 0.2.0

Copyright See file inst/COPYRIGHTS.

Description Verified interval arithmetic for R, in the inf-sup (endpoint) representation of the set-based flavor of the interval standard. Every operation returns an enclosure that provably contains the exact result: outward rounding is obtained from the predecessor and successor formulas of Rump, Zimmermann, Boldo and Melquiond (2009) <doi:10.1007/s10543-009-0218-z>, which are valid under round-to-nearest and therefore need no change to the floating-point rounding mode. That mode is not reachable from R, and changing it would not be a local act: it is per-thread state of the processor, so it would govern every floating-point operation executed afterwards on that thread, in this package or anywhere else. Elementary functions are provided at two levels: a fast level over the included correctly rounded binary64 implementation, comprising fifteen kernels from CORE-MATH <doi:10.1109/ARITH54963.2022.00014> and the hardware square root, widened by the pre-registered slack of two outward steps; and a rigorous level over 'Rmpfr' with a directed-rounding bridge, reached by an escalation ladder of precisions when a verdict would otherwise fall inside the slack. Fast-level enclosures retain measured provenance because correct rounding of the included software is verified numerically rather than established here as a theorem for every kernel.
On top of the kernel the package builds natural and centered interval extensions of expressions, a monotonicity test, the Hansen-Sengupta interval Newton operator with extended division and epsilon-inflated candidate verification, and a subdivision (paving) engine whose only failure mode is a named abstention with its budget printed. Conformance with IEEE Std 1788.1-2017 <doi:10.1109/IEEESTD.2018.8277144> is not claimed, and the reason is the standard's own: its subclause 1.5 makes conformance a list of requirements that an implementation shall satisfy, with no partial grade to claim. What this package follows, measured one requirement at a time and stated in the package documentation, is the interval type and the decoration system of clause 5, 22 of the 39 arithmetic operations of Table 4.1, and the seven

numeric functions of Table 4.3. What it does not provide is the cancellative operations, the interval comparison relations, the text input and output of subclause 6.8, the interchange representation of subclause 7.3, and the tightest accuracy that subclause 6.5.2 requires of the basic operations, which here are one unit in the last place wider at each end.

License GPL (>= 3)

Encoding UTF-8

Depends R (>= 4.1)

Imports stats

Suggests Rmpfr, testthat (>= 3.0.0), knitr, rmarkdown

VignetteBuilder knitr, rmarkdown

Config/testthat/edition 3

URL https://github.com/IsadoreNabi/RobustArithmetic

BugReports https://github.com/IsadoreNabi/RobustArithmetic/issues

Config/roxygen2/version 8.0.0

RoxygenNote 8.0.0

NeedsCompilation yes

Author Jose Mauricio Gomez Julian [aut, cre] (ORCID:
<https://orcid.org/0009-0000-2412-3150>),
Alexei Sibidanov [ctb, cph],
Paul Zimmermann [ctb, cph],
Tom Hubrecht [ctb, cph],
Cyprien Peignier [ctb, cph],
CERN [cph],
INRIA [cph],
Szabolcs Nagy [ctb, cph]

Maintainer Jose Mauricio Gomez Julian <isadore.nabi@pm.me>

Repository CRAN

Date/Publication 2026-09-16 06:50:19 UTC

Contents

`as.data.frame.ra_certificate` *Coerce a fixed point certificate to a data frame*

Description

Returns the certificate as a single row.

Usage

```
## S3 method for class 'ra_certificate'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	An object of class `ra_certificate`.
`row.names`	Passed to the data frame constructor.
`optional`	Passed to the data frame constructor.
`...`	Ignored, present for consistency with the generic.

Details

The row carries the verdicts, the route, the radius and the error bound, the three bounds used, the norm label and the provenance, so that a batch of certificates can be stacked and read as a table.

Value

A data frame of one row.

References

Granas, A., & Dugundji, J. (2003). Fixed point theory. Springer. https://doi.org/10.1007/978-0-387-21593-8

See Also

[ra_ball_certificate()].

Examples

```
as.data.frame(ra_ball_certificate(0.2, 0.5))
```

`format.ra_certificate` *Format a fixed point certificate*

Description

Renders the certificate as the lines of its card.

Usage

```
## S3 method for class 'ra_certificate'
format(x, ...)
```

Arguments

`x`	An object of class `ra_certificate`.
`...`	Ignored, present for consistency with the generic.

Details

The card names both words and never lets one stand for the other. A confined ball that is not a contraction says so, and says that what happens inside is not claimed, which is the doctrinal boundary of the certificate.

Value

A character vector, one element per line.

References

Granas, A., & Dugundji, J. (2003). Fixed point theory. Springer. https://doi.org/10.1007/978-0-387-21593-8

See Also

[ra_ball_certificate()].

Examples

```
format(ra_ball_certificate(0.2, 0.5))
```

`format.ra_summary_certificate` *Format the summary of a fixed point certificate*

Description

Renders the summary as its lines.

Usage

```
## S3 method for class 'ra_summary_certificate'
format(x, ...)
```

Arguments

`x`	An object of class `ra_summary_certificate`.
`...`	Ignored, present for consistency with the generic.

Details

The summary prints the two words in the same order the card does, so that a reader who has seen one recognises the other.

Value

A character vector, one element per line.

References

Granas, A., & Dugundji, J. (2003). Fixed point theory. Springer. https://doi.org/10.1007/978-0-387-21593-8

See Also

[summary.ra_certificate()].

Examples

```
format(summary(ra_ball_certificate(0.2, 0.5)))
```

`Ops.ra_ivl` *Interval arithmetic through the ordinary operators*

Description

The four arithmetic operators and unary negation, applied to intervals, returning enclosures of the exact set of results.

Usage

```
## S3 method for class 'ra_ivl'
Ops(e1, e2)
```

Arguments

`e1`, `e2`	Objects of class `ra_ivl`, or numeric vectors, which are promoted to intervals of zero width. Lengths are recycled.

Details

Addition, subtraction and multiplication are computed from the endpoints and then widened by one rounding step in each direction, which is what makes the result contain the exact set. Multiplication takes the four endpoint products and keeps their extremes; a zero endpoint meeting an infinite one contributes zero rather than an indeterminate form, because an infinity here is a bound and never a member of the interval.

Division follows the set-based definition of the standard: the result is the tightest interval containing every quotient that is defined. When the divisor contains zero the set of quotients is generally not an interval, and the result is its hull, which is valid but wide, and carries the `trv` decoration to say that nothing was asserted about the function. The operation that keeps the two pieces apart instead of hulling them is [ra_div_extended()], and the one that refuses rather than widening is `ra_div(x, y, zero = "error")`.

An empty argument gives an empty result, and Not-an-Interval propagates through everything.

Value

An object of class `ra_ivl`.

Methodological notes

The excess width of these operations is one-sided and that is what makes them usable. Evaluating an expression in which a variable occurs more than once treats the occurrences as if they could take different values, so the result is wider than the true range; it is never narrower. A conclusion drawn from an enclosure that excludes zero therefore holds, while a failure to exclude zero proves nothing, and the package never reads it as if it did.

The operations are subclause 4.5.2 and Table 4.1 of IEEE Std 1788.1-2017; chapters 2 and 4 of Moore et al. (2009) are the textbook treatment.

Dependencies

Base R only.

References

Institute of Electrical and Electronics Engineers. (2018). IEEE standard for interval arithmetic (simplified) (IEEE Std 1788.1-2017). https://doi.org/10.1109/IEEESTD.2018.8277144

Moore, R. E., Kearfott, R. B., & Cloud, M. J. (2009). Introduction to interval analysis. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9780898717716

See Also

[ra_div_extended()], [ra_sqr()], [ra_pown()].

Examples

```
x <- ra_interval(1, 2)
y <- ra_interval(3, 4)
x + y
x * y
x - x
x / y
-x
```

---

print.ra_certificate *Print a fixed point certificate*

Description

Prints the card of the certificate.

Usage

```
## S3 method for class 'ra_certificate'
print(x, ...)
```

Arguments

x	An object of class ra_certificate.
...	Ignored, present for consistency with the generic.

Details

Printing is the card and nothing else; the numbers behind it are the elements of the list and [as.data.frame()] gives them in one row.

Value

x, invisibly.

References

Granas, A., & Dugundji, J. (2003). Fixed point theory. Springer. https://doi.org/10.1007/978-0-387-21593-8

See Also

[format.ra_certificate()].

Examples

```
print(ra_ball_certificate(0.2, 0.5))
```

---

print.ra_summary_certificate

Print the summary of a fixed point certificate

Description

Prints the lines of the summary.

Usage

```
## S3 method for class 'ra_summary_certificate'
print(x, ...)
```

Arguments

x	An object of class ra_summary_certificate.
...	Ignored, present for consistency with the generic.

Details

Printing is the summary and nothing else.

Value

x, invisibly.

References

Granas, A., & Dugundji, J. (2003). Fixed point theory. Springer. https://doi.org/10.1007/978-0-387-21593-8

See Also

[format.ra_summary_certificate()].

Examples

```
print(summary(ra_ball_certificate(0.2, 0.5)))
```

`ra_arith` *The named arithmetic operations*

Description

The same operations as the operators, available under names, so that they can be passed to other functions and so that division can be asked to behave differently when the divisor contains zero.

Usage

```
ra_neg(x)

ra_add(x, y)

ra_sub(x, y)

ra_mul(x, y)

ra_div(x, y, zero = c("hull", "error"))
```

Arguments

`x`, `y`	Objects of class `ra_ivl`, or numeric vectors.
`zero`	A character scalar saying what division should do when the divisor contains zero: `"hull"`, the default and the behaviour of the standard, returns the tightest interval containing every defined quotient; `"error"` raises `ra_division_straddles_zero`.

Details

The `"error"` option exists because in some settings a divisor straddling zero is a sign that the caller has lost track of a domain condition, and returning the whole line lets the run continue on an answer that is valid and empty of content. In the paving engine the opposite is true, and the default is the total behaviour of the standard.

Value

An object of class `ra_ivl`.

Methodological notes

The two behaviours differ in what they treat as a failure, not in what they treat as true. Neither ever returns an interval that fails to contain the exact result.

Table 4.1 of IEEE Std 1788.1-2017 gives the division its domain: the plane minus the axis where the divisor vanishes.

Dependencies

Base R only.

References

Institute of Electrical and Electronics Engineers. (2018). IEEE standard for interval arithmetic (simplified) (IEEE Std 1788.1-2017). https://doi.org/10.1109/IEEESTD.2018.8277144

See Also

[Ops.ra_ivl()].

Examples

```
ra_add(ra_interval(1, 2), ra_interval(3, 4))
ra_div(ra_interval(1, 2), ra_interval(-1, 1))
tryCatch(ra_div(ra_interval(1, 2), ra_interval(-1, 1), zero = "error"),
         ra_division_straddles_zero = function(cnd) "refused")
```

---

ra_ball_certificate *Certify a fixed point over a ball in a norm*

Description

Certifies, from three scalar upper bounds, that a continuous map sends a closed ball into itself (confinement, and hence existence of a fixed point by Brouwer) and, separately, that it is a contraction there (uniqueness and convergence by Banach).

Usage

```
ra_ball_certificate(
  rho,
  lipschitz = NULL,
  radius = NULL,
  image_radius = NULL,
  norm = "unspecified"
)
```

Arguments

rho	An upper bound on the residual at the centre, `||Phi(c) - c||`, as a numeric of length one or an `ra_ivl`.
lipschitz	An upper bound on a Lipschitz constant of `Phi` over the ball, in the same norm. Optional: without it there is no contraction verdict and no minimal radius, and `radius` together with `image_radius` must be supplied.
radius	The radius of the ball to verify. When `NULL` and `lipschitz` is below one, the minimal certifiable radius `rho / (1 - L)` is computed instead.
image_radius	An optional upper bound on `sup ||Phi(x) - c||` over the ball: the direct route to confinement, which needs no Lipschitz constant and is genuinely weaker than contraction.
norm	A character label naming the norm the three bounds are measured in. It is carried, never checked: see the details.

Details

The three bounds must all be measured in the SAME norm, and the package cannot check that: the label travels with the object and the responsibility is the caller's. Mixing norms silently would be a mode of failure indistinguishable from a valid certificate at run time.

The Lipschitz constant must hold over the WHOLE ball and not at its centre. A caller who bounds the derivative at the centre alone receives a certificate that means nothing, and no arithmetic can detect it. In one dimension the package closes that loop itself: [ra_certify_fixed_point()] derives both bounds over the ball from the expression.

Confinement is reached by either of two theorems, and the object says which. Through the direct route, an image radius no larger than the ball radius is confinement by definition. Through the Lipschitz route, the triangle inequality gives `||Phi(x) - c|| <= L r + rho`, so `L r + rho <= r` suffices; note that with a positive residual this forces `L < 1`, which is why the direct route is not redundant.

Every comparison is evaluated with the rounding that makes an affirmative verdict provable: the left side of the confinement test is rounded up and the right side down, and the minimal radius divides an upward-rounded residual by a downward-rounded `1 - L`. When `L` is within an ulp of one that denominator can round to zero; the certificate is then withheld with its reason, never returned as an infinite radius.

Value

An object of class `ra_certificate`, a list with the logical verdicts `confined` and `contraction`, the route that decided the confinement, the certified `radius`, the `error_bound` on the centre, the bounds used, the norm label, the provenance and, when a verdict is withheld, the `reason`.

Non-objectives

A refusal certifies nothing. The hypotheses are sufficient and not necessary, and there is no test of exclusion here, so this file has no word of demonstrated absence. Nothing is claimed about the basin of attraction outside the ball, about periodic orbits, or about what happens inside a confined ball that is not a contraction.

References

Granas, A., & Dugundji, J. (2003). Fixed point theory. Springer. https://doi.org/10.1007/978-0-387-21593-8

Neumaier, A. (1990). Interval methods for systems of equations. Cambridge University Press. https://doi.org/10.1017/CBO9780511526473

Rump, S. M. (2010). Verification methods: Rigorous results using floating-point arithmetic. Acta Numerica, 19, 287-449. https://doi.org/10.1017/S096249291000005X

See Also

[ra_certify_fixed_point()] for the univariate door that derives the bounds itself, [ra_gershgorin()] and [ra_spectral_sum()] for assembling them in the matrix case.

Examples

```
ra_ball_certificate(0.2, 0.5)
ra_ball_certificate(0.24, 1.6, radius = 0.4, image_radius = 0.4)
```

`ra_certify_fixed_point` *Certify a fixed point of a univariate expression over a ball*

Description

Derives both bounds the certificate needs – the residual at the centre and a Lipschitz constant over the whole ball – by interval evaluation of the expression and of its derivative, and then certifies.

Usage

```
ra_certify_fixed_point(
  e,
  centre,
  radius,
  var = "x",
  env = list(),
  level = c("fast", "rigorous"),
  precision = ra_precision_ladder()[1L]
)
```

Arguments

`e`	A call or expression over the closed operator table, in one variable.
`centre`	A numeric of length one: the centre of the ball.
`radius`	A numeric of length one: the radius of the ball.
`var`	A character scalar naming the variable of `e`.

env	A named list of values for the other symbols of e.
level	The evaluation level, as in [ra_elem()].
precision	The precision of the rigorous level, in bits.

Details

This is the door that closes the loop: the Lipschitz constant is the supremum of the absolute value of the enclosed derivative over the ball, so a caller cannot certify on a constant measured at the centre. The image radius is enclosed directly as well, so the confinement can be certified by the direct route when the map is not a contraction.

In one variable the ball is the interval [c - r, c + r], and the three bounds are exactly what [ra_enclose_expr()] returns for the expression and for stats::D of the expression.

Value

An object of class ra_certificate, as [ra_ball_certificate()], with the norm labelled "abs".

References

Granas, A., & Dugundji, J. (2003). Fixed point theory. Springer. https://doi.org/10.1007/978-0-387-21593-8

See Also

[ra_ball_certificate()] for the general case, [ra_enclose_expr()] for the enclosure underneath.

Examples

```
ra_certify_fixed_point(quote(x / 2 + 1), centre = 1.9, radius = 0.5)
```

---

ra_check_expr *Check that an expression is written with admitted symbols*

Description

Walks an expression and refuses, by name and with the reason, any symbol outside the closed table of this version.

Usage

```
ra_check_expr(e)
```

Arguments

e	A call, a name or a number, as returned by quote() or str2lang().

Details

The check is separate from the evaluation so that a caller can find out whether an expression is admissible before committing to a computation over it, and so that the refusal names the symbol rather than arriving as a failure deep inside a recursion.

Value

Invisibly, a character vector of the variables the expression uses. Raises `ra_symbol_not_in_table` if a head symbol is refused.

Methodological notes

The table is closed by enumeration and not open by default. An unknown symbol is refused rather than evaluated on the assumption that whatever R does with it is an interval extension of it, which is the assumption that would silently turn an enclosure into an estimate.

Dependencies

Base R only.

References

Moore, R. E., Kearfott, R. B., & Cloud, M. J. (2009). Introduction to interval analysis. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9780898717716

See Also

[ra_operator_table()] and [ra_eval_natural()].

Examples

```
ra_check_expr(quote(exp(x) - x^2))
tryCatch(ra_check_expr(quote(besselJ(x, 1))),
         ra_symbol_not_in_table = function(cnd) "refused")
```

`ra_check_sentinels` *Verify the included evaluator against independent sentinels*

Description

Evaluate every stored difficult case through the same binary64 evaluator as the fast level and compare each result bit for bit with its independent correctly rounded value.

Usage

```
ra_check_sentinels()
```

Details

The comparison preserves distinctions hidden by ordinary numeric equality, including the sign of zero. Any error while evaluating the table makes the load-time verification unavailable, which degrades the fast level rather than certifying an unchecked evaluator.

Value

A data frame with columns `fun`, `x`, `role`, `source`, `source_index`, `expected`, `observed`, `expected_bits`, `observed_bits` and `ok`. Each row of `ok` is `TRUE` only when the two binary64 bit patterns are identical.

Methodological notes

Version agreement cannot establish that compiled mathematics behaves as intended. This check asks the installed evaluator itself to reproduce independently generated difficult cases. Its control is exercised by replacing that evaluator with a one-neighbour perturbation in the scenario harness and requiring the verification to fail.

Dependencies

Base R and the package's registered native routines.

References

Sibidanov, A., Zimmermann, P., & Glondu, S. (2022). The CORE-MATH project. In *2022 IEEE 29th Symposium on Computer Arithmetic (ARITH)* (pp. 26-34). IEEE. https://doi.org/10.1109/ARITH54963.2022.00014

See Also

`ra_sentinels()` for the stored cases and `ra_fast_level_status()` for the session-level consequence.

Examples

```
check <- ra_check_sentinels()
all(check$ok)
```

---

ra_constants *The two constants of the outward-rounding theorem*

Description

Returns the unit roundoff of binary64, the widening factor of the predecessor and successor formulas, and the smallest positive subnormal, as the package holds them.

Usage

```
ra_constants()
```

Details

The three are exposed because a reader who wants to check the package against the paper should be able to read the constants rather than take them on trust. `phi` is the successor of `u` in binary64, which is what makes the formulas valid at the points where the naive factor `u` fails; those points are not exotic, and `x = 1` is one of them.

Value

A named numeric vector of length three, with elements `u`, `phi` and `eta`.

Methodological notes

The naive factor is the positive control of the rounding tests: the same formulas evaluated with `phi = u` must violate validity, and the test suite fails if they do not. A positive control that cannot fire is a broken harness, so the naive factor is kept and exercised rather than deleted.

The validity and exactness claims are Theorems 2.1 and 2.2 of Rump et al. (2009), whose proofs are machine-checked in Coq.

Dependencies

Base R only.

References

Rump, S. M., Zimmermann, P., Boldo, S., & Melquiond, G. (2009). Computing predecessor and successor in rounding to nearest. BIT Numerical Mathematics, 49(2), 419-431. https://doi.org/10.1007/s10543-009-0218-z

See Also

[ra_succ()] and [ra_pred()], which use them.

Examples

```
k <- ra_constants()
k["phi"] > k["u"]
identical(k["phi"], 2^-53 * (1 + 2^-52))
```

`ra_div_extended` *Extended division, keeping the two pieces apart*

Description

Divides by an interval that may contain zero and returns the result as one or two intervals instead of hulling them together, which is what makes the gap in the middle usable.

Usage

```
ra_div_extended(x, y)
```

Arguments

x, y	Objects of class ra_ivl, or numeric vectors. y may contain zero.

Details

When the divisor contains zero in its interior and the numerator does not contain zero, the set of quotients is the union of two half-lines with a gap between them. Hulling them gives the whole real line and throws away the only piece of information the division produced, which is the gap. Keeping them apart is what allows the Newton step to split a box into two at exactly the place where the derivative vanishes, which is the place a bisection at the midpoint would have to find by luck.

When the divisor touches zero only at one endpoint the result is a single half-line; when the numerator also contains zero nothing is excluded and the result is the whole line in one piece; when the divisor is exactly the zero interval no quotient is defined, and the result is reported as the whole line rather than as the empty set, so that nobody reads a vacuous division as a proof of exclusion.

Value

A list with two elements, first and second, both of class ra_ivl and of the length of the recycled arguments. Where the result has only one piece, the corresponding element of second is the empty interval.

Methodological notes

This is the operation that separates the Hansen-Sengupta operator from the Krawczyk operator, and it is why the design chose the former. Neumaier's argument is that the iteration with extended division is never weaker than Krawczyk's and splits the box exactly where the derivative contains zero, which is the multiple root that motivates the whole path.

The case table implemented here is equations (9.2.3) and (9.2.4) in section 9.2 of Hansen and Walster (2004); the comparison with the Krawczyk operator is section 5.1 of Neumaier (1990).

Dependencies

Base R only.

References

Hansen, E., & Walster, G. W. (2004). Global optimization using interval analysis (2nd ed.). Marcel Dekker.

Neumaier, A. (1990). Interval methods for systems of equations. Cambridge University Press. https://doi.org/10.1017/CBO9780511526473

See Also

[ra_div()].

Examples

```
ra_div_extended(ra_interval(1, 1), ra_interval(-1, 2))
ra_div_extended(ra_interval(1, 1), ra_interval(0, 2))
```

`ra_elem` *Interval extension of an elementary function*

Description

Encloses the set of values an admitted function takes over an interval, at either of the two levels the package provides.

Usage

```
ra_elem(
  fun,
  x,
  level = c("fast", "rigorous"),
  precision = ra_precision_ladder()[1L]
)
```

Arguments

`fun`	A character scalar naming a function of the closed table returned by [ra_operator_table()].
`x`	An object of class `ra_ivl`, or a numeric vector, which is promoted to intervals of zero width.
`level`	A character scalar, `"fast"` for the included correctly rounded binary64 implementation widened by the declared slack, or `"rigorous"` for a multiprecision evaluation. Defaults to `"fast"`.
`precision`	An integer scalar, the working precision in bits of the rigorous level. Ignored at the fast level. Defaults to the first rung of [ra_precision_ladder()].

Details

The value returned encloses `{f(t) : t in x, t in dom(f)}`. The enclosure may be wider than that set and is never narrower, so a point proved to lie outside it is proved to be outside the range.

The endpoints are chosen by the shape of the function on its domain rather than by evaluating both ends and sorting. A monotone function is read at its ends; the hyperbolic cosine has its minimum in the interior of any box containing zero, and the value there is exactly one; the sine and cosine have interior extrema whose positions are found by reduction, and the value at an attained extremum is exactly one in absolute value. Those exact values receive no slack, because they are the mathematics of the function and not an evaluation of a library.

The result is finally intersected with the range of the function. This matters at the ends of a range that the slack can step past: the square root of zero is zero, and widening it downward would report that the square root of a nonnegative box might be negative.

An argument box that leaves the domain is intersected with it. The part outside is dropped, the range of the part inside is returned, and the decoration falls to `trv`, which is the standard saying that the claim of being defined everywhere on the box has failed. A box entirely outside the domain gives the empty interval. Neither case raises and neither produces a missing value.

Value

An object of class `ra_ivl` of the length of `x`.

Methodological notes

The fast implementation consists of fifteen CORE-MATH software kernels and the binary64 hardware square root. Its declared error bound is one half of a unit in the last place, so the pre-registered formula `ceiling(2 * e + 1)` applies two outward steps to every function. The result retains measured provenance because the correctly rounded claim of the included software was verified numerically and is not established here as a theorem for every kernel.

The reduction that locates the interior extrema is performed in interval arithmetic against an enclosure of pi, and the resulting interval of indices is asked whether it contains an integer. That interval is a superset of the true one, so the test can report an extremum that is not there but can never miss one that is; the first widens the answer and the second would break the guarantee. When the box is wide enough that the index interval contains an integer regardless, the answer is the full range, which is valid and, above a magnitude that [ra_periodic_frontier()] measures, also tight: adjacent doubles of that size are more than a period apart.

The tangent is read as monotone only when the index interval of its poles contains no integer. When it may contain one, the range is genuinely unbounded and the answer is the whole line with `trv`. A point argument is never a pole, because every pole is irrational and every double is not, so the reduction is skipped there and the library value is used.

Dependencies

The fast level uses the package's registered native routines and has no external package dependency. The rigorous level requires 'Rmpfr', which is in `Suggests`, and raises `ra_no_mpfr` when it is absent, since the caller asked for it by name.

References

Moore, R. E., Kearfott, R. B., & Cloud, M. J. (2009). Introduction to interval analysis. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9780898717716

Institute of Electrical and Electronics Engineers. (2018). IEEE standard for interval arithmetic (simplified) (IEEE Std 1788.1-2017). https://doi.org/10.1109/IEEESTD.2018.8277144

Sibidanov, A., Zimmermann, P., & Glondu, S. (2022). The CORE-MATH project. In 2022 IEEE 29th Symposium on Computer Arithmetic (ARITH) (pp. 26-34). IEEE. https://doi.org/10.1109/ARITH54963.2022.00014

See Also

[ra_operator_table()] for what is admitted, [ra_slack()] for the widening of the fast level, and [ra_elem_escalate()] for the ladder.

Examples

```
ra_elem("exp", ra_interval(0, 1))
ra_elem("sin", ra_interval(0, pi))
ra_elem("log", ra_interval(-1, 2))
ra_elem("tan", ra_interval(1, 2))
```

---

ra_elem_escalate *Climb the precision ladder until a question is settled*

Description

Evaluates an elementary function at the fast level and, if the enclosure does not settle the question the caller asked, again at each rung of the precision ladder, stopping at the first rung that does or reporting the budget it exhausted.

Usage

```
ra_elem_escalate(fun, x, decide, ladder = ra_precision_ladder())
```

Arguments

`fun`	A character scalar naming a function of the closed table.
`x`	An object of class `ra_ivl` of length one, or a numeric scalar.
`decide`	A function of one argument, the enclosure, returning a logical scalar that is `TRUE` when the enclosure settles the question.
`ladder`	An integer vector of precisions in bits, in increasing order. Defaults to [ra_precision_ladder()].

Details

The abstention is the point of this function. A question that the available budget cannot settle produces an object saying so, with the number of bits printed in the sentence; it does not return a verdict chosen to break the tie, and it does not fall silent. When the backend is absent the fast level answers alone and the object carries the sentence of [ra_message_no_mpfr()], because a missing optional package is not a failure of the caller.

Value

An object of class `ra_escalation`.

Methodological notes

The deciding criterion belongs to the caller and is never loosened here. The ladder is the mechanism for resolving rather than relaxing: a verdict lying closer to a boundary than the declared slack of the fast level is not declared by widening the tolerance until it fits, it is recomputed with more bits until the enclosure separates from the boundary or the budget runs out. Which of the two happened is recorded in the object rather than inferred from the answer.

Dependencies

Base R; uses 'Rmpfr' from `Suggests` for every rung above the fast level, and degrades to the fast level alone without it.

References

Revol, N., & Rouillier, F. (2005). Motivations for an arbitrary precision interval arithmetic and the MPFI library. Reliable Computing, 11(4), 275-290. https://doi.org/10.1007/s11155-005-6891-y

Rump, S. M. (2010). Verification methods: Rigorous results using floating-point arithmetic. Acta Numerica, 19, 287-449. https://doi.org/10.1017/S096249291000005X

See Also

[ra_elem()] and [ra_precision_ladder()].

Examples

```
if (ra_has_mpfr()) {
  ra_elem_escalate("sin", ra_interval(1, 1),
                   function(iv) ra_wid(iv) < 2^-52)
}
```

`ra_enclose_expr` *Enclose the range of a univariate expression, by three routes at once*

Description

Computes the range of an expression over an interval by the natural extension, by the centered form and, when the derivative allows it, by monotonicity, and returns the intersection of whichever routes applied.

Usage

```
ra_enclose_expr(
  e,
  x,
  var = "x",
  env = list(),
  methods = c("natural", "mean_value", "monotonic"),
  level = c("fast", "rigorous"),
  precision = ra_precision_ladder()[1L]
)
```

Arguments

e	A call, a name or a number.
x	An object of class ra_ivl of length one, or a numeric vector of length one or two giving the endpoints.
var	A character scalar naming the variable that ranges over x. Defaults to "x".
env	A named list giving values to the other variables of e. Defaults to an empty list.
methods	A character vector choosing the routes to attempt, any of "natural", "mean_value" and "monotonic". Defaults to all three.
level	A character scalar, "fast" or "rigorous".
precision	An integer scalar, the working precision of the rigorous level.

Details

The three routes are three theorems and the answer is their intersection, which is valid because each of them alone is.

The natural extension evaluates the expression with interval operations. It always applies and is usually the widest.

The centered form is f(c) + df(x) * (x - c), with df the derivative and c the midpoint. It is the mean value theorem read as an enclosure. It needs the derivative, which comes from the differentiation engine of R, and it is narrower than the natural extension on a narrow box and wider on a wide one, which is why both are computed rather than one chosen in advance.

The monotonicity route applies when the enclosure of the derivative excludes zero. The function is then monotone on the box, its range is the hull of the values at the two endpoints, and that is the exact range up to the enclosure of the endpoint evaluations themselves. It is the cheapest and the tightest and it is tried first in the sense that it is reported when it applies.

Value

An object of class ra_ivl of length one, carrying the attribute "ra_routes", a character vector naming the routes that contributed.

Methodological notes

Intersecting rather than choosing is deliberate. A rule that picked the narrower of two enclosures by comparing their widths would be making a decision the caller cannot audit; the intersection needs no decision and is never wider than either. Where a route does not apply it contributes nothing, and which ones contributed is recorded in the returned attribute rather than left to be inferred from the width.

Dependencies

Uses stats::D for the derivative. 'Rmpfr' from Suggests at the rigorous level.

References

Moore, R. E., Kearfott, R. B., & Cloud, M. J. (2009). Introduction to interval analysis. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9780898717716

Neumaier, A. (1990). Interval methods for systems of equations. Cambridge University Press. https://doi.org/10.1017/CBO9780511526473

See Also

[ra_eval_natural()].

Examples

```
ra_enclose_expr(quote(x - x), ra_interval(-1, 1))
ra_enclose_expr(quote(exp(x)), ra_interval(0, 1))
ra_enclose_expr(quote(r - x^2), ra_interval(-1, 1), env = list(r = 0.5))
```

---

ra_enclose_mpfr *Enclose a multiprecision value in a pair of doubles*

Description

Returns the tightest pair of binary64 endpoints this package can place around a multiprecision value.

Usage

```
ra_enclose_mpfr(x)
```

Arguments

x	An object of class mpfr.

Details

The two endpoints are [ra_to_double()] in each direction. When the multiprecision value happens to be exactly representable the two coincide, and the enclosure is a point; otherwise they are adjacent doubles and the enclosure has the least width any pair of doubles could have.

Value

A list with elements lo and hi, both numeric vectors of the length of x.

Methodological notes

This is the only place where a result of the rigorous level becomes an ordinary interval, so it is the only place where the guarantee of the rigorous level could be lost. Concentrating the crossing here is what makes the guarantee auditable.

Dependencies

Requires 'Rmpfr', which is in `Suggests`.

References

Revol, N., & Rouillier, F. (2005). Motivations for an arbitrary precision interval arithmetic and the MPFI library. Reliable Computing, 11(4), 275-290. https://doi.org/10.1007/s11155-005-6891-y

See Also

[ra_to_double()].

Examples

```
if (ra_has_mpfr()) {
  e <- ra_enclose_mpfr(exp(Rmpfr::mpfr("1", 200L)))
  e$hi - e$lo
}
```

`ra_escalation_show` *Show the result of an escalation*

Description

Render an escalation for the console and for a data frame, with the word of its verdict and the budget it reached.

Usage

```
## S3 method for class 'ra_escalation'
format(x, ...)

## S3 method for class 'ra_escalation'
print(x, ...)

## S3 method for class 'ra_escalation'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'ra_escalation'
summary(object, ...)

## S3 method for class 'ra_escalation_summary'
print(x, ...)
```

Arguments

x	An object of class ra_escalation.
...	Further arguments, ignored, present for consistency with the generics.
row.names, optional	Arguments of the as.data.frame generic. The first is used; the second is ignored.
object	An object of class ra_escalation, for summary.

Details

The word is printed always, including when the question was settled, so that a reader never has to infer from the presence of a number whether the number was reached or exhausted.

Value

format returns a character scalar; print returns its argument invisibly; as.data.frame returns a one-row data frame with the function, the enclosure, its decoration, the budget, whether the question was settled and the word; summary returns an object of class ra_escalation_summary, which prints the same fields with the argument box added.

Methodological notes

These are written with the class rather than after it. A class whose printing and coercion arrive later spends the interval producing output that looks like a list of three fields, which is how an abstention gets read as a result.

Dependencies

Base R only.

References

Wickham, H. (2019). Advanced R (2nd ed.). Chapman & Hall/CRC. https://doi.org/10.1201/9781351201315

See Also

[ra_elem_escalate()].

Examples

```
if (ra_has_mpfr()) {
  e <- ra_elem_escalate("sin", ra_interval(1, 1),
                        function(iv) ra_wid(iv) < 2^-52)
  print(e)
  as.data.frame(e)
  summary(e)
}
```

`ra_eval_natural` *Natural interval extension of an expression*

Description

Evaluates an expression with every operation replaced by its interval counterpart, giving an enclosure of the set of values the expression takes as its variables range over their intervals.

Usage

```
ra_eval_natural(
  e,
  env,
  level = c("fast", "rigorous"),
  precision = ra_precision_ladder()[1L]
)
```

Arguments

`e`	A call, a name or a number.
`env`	A named list giving each variable of `e` an object of class `ra_ivl` or a numeric vector.
`level`	A character scalar passed to [ra_elem()] for the elementary functions, `"fast"` or `"rigorous"`. Defaults to `"fast"`.
`precision`	An integer scalar, the working precision of the rigorous level. Defaults to the first rung of [ra_precision_ladder()].

Details

Each occurrence of a variable is evaluated independently, so an expression mentioning a variable more than once is generally overestimated. The overestimate is one-sided and that is the whole guarantee: the result contains the range and may be wider, so a value proved outside it is proved outside the range.

An integer constant exponent uses the tight integer power, which handles a base spanning zero correctly; any other exponent is evaluated as the exponential of the product with the logarithm, which needs a positive base and reports a base that is not positive through the decoration.

Value

An object of class `ra_ivl`.

Methodological notes

The identity `x - x = 0` is not applied. Implementing it would make this expression exact and every expression not covered by the rewriting rule silently different from the ones that are, which is worse than a uniform overestimate a caller can reason about. The package chooses not to be clever in one place at the price of being unpredictable everywhere, and the choice has a test that fixes it: the enclosure of `x - x` over `[-1, 1]` is asserted to be `[-2, 2]`.

Dependencies

Base R at the fast level; 'Rmpfr' from `Suggests` at the rigorous one.

References

Moore, R. E., Kearfott, R. B., & Cloud, M. J. (2009). Introduction to interval analysis. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9780898717716

Neumaier, A. (1990). Interval methods for systems of equations. Cambridge University Press. https://doi.org/10.1017/CBO9780511526473

See Also

[ra_enclose_expr()], which combines this with the centered form.

Examples

```
x <- ra_interval(-1, 1)
ra_eval_natural(quote(x - x), list(x = x))
ra_eval_natural(quote(exp(x) + x^2), list(x = ra_interval(0, 1)))
```

`ra_fast_level_status` *Status of the fast level in the current session*

Description

Report whether the fast level is degraded, why it is degraded, and the two verifications that determine its session state.

Usage

```
ra_fast_level_status()
```

Details

The status is read-only. A failed load-time sentinel verification degrades the level immediately. With `Rmpfr` available, the first fast evaluation also runs `ra_measure_library_error()` and degrades the level if an observed error exceeds its declared slack. The first result affected by a degraded state emits one warning of class `ra_fast_level_unsafe` and escalates to the rigorous level; later results remain rigorous without repeating the warning. Without `Rmpfr`, every affected evaluation refuses with an error of the same class because no rigorous result can replace it.

Value

A list with four fields: `degraded`, a logical scalar; `reason`, an empty character vector while healthy or a character scalar naming every active cause; `sentinels`, the data frame returned by `ra_check_sentinels()` at load time or `NULL` if that verification could not run; and `audit`, the first-use comparison against MPFR or `NULL` until that audit runs.

Methodological notes

Session state and result provenance answer different questions. This function records whether the fast route is available and why. `ra_prov()` records the guarantee carried by one returned interval, so an interval escalated after degradation carries `theorem` without changing its class.

Dependencies

Base R. The first-use audit requires `Rmpfr`; reading the status does not.

References

Fousse, L., Hanrot, G., Lefevre, V., Pelissier, P., & Zimmermann, P. (2007). MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Transactions on Mathematical Software, 33*(2), Article 13. https://doi.org/10.1145/1236463.1236468

See Also

`ra_check_sentinels()`, `ra_measure_library_error()` and `ra_prov()`.

Examples

```
status <- ra_fast_level_status()
status$degraded
all(status$sentinels$ok)
```

`ra_gershgorin` *Enclose the spectrum of a symmetric matrix by Gershgorin discs*

Description

Returns an interval containing every eigenvalue of a real symmetric matrix, computed from its entries alone and without any factorisation.

Usage

```
ra_gershgorin(A, hi = NULL, discs = FALSE)
```

Arguments

`A`	A real symmetric matrix, or the matrix of lower endpoints when `hi` is given.
`hi`	An optional matrix of upper endpoints, for entries known only to an interval.
`discs`	Logical: when `TRUE` the per-row discs are returned as an attribute `ra_discs`.

Details

Every eigenvalue of a symmetric `A` lies in the union of the discs centred at `a_ii` with radius the sum of the absolute values of the off-diagonal entries of row `i`. The enclosure returned is the hull of those discs, computed with outward rounding, so it is a superset of the spectrum: it can widen, it cannot lose an eigenvalue.

On a diagonal matrix the radii are zero and the enclosure is exactly the extreme diagonal entries, which is the case where the bound is tight.

Value

An object of class `ra_ivl` of length one enclosing the whole spectrum.

References

Horn, R. A., & Johnson, C. R. (2013). Matrix analysis (2nd ed.). Cambridge University Press.

See Also

[ra_spectral_sum()] for combining two spectra, [ra_ball_certificate()] for what these bounds are usually assembled for.

Examples

```
ra_gershgorin(matrix(c(4, 1, 1, 3), 2, 2))
```

`ra_has_mpfr` *Is the rigorous level available*

Description

Reports whether the optional multiprecision backend is installed, without loading it and without raising if it is not.

Usage

```
ra_has_mpfr()
```

Details

The probe is called at the point of use rather than when the package loads, which is what allows the package to be installed, checked and used without the backend. Every function that would escalate calls this first and, when it returns `FALSE`, answers at the fast level and says so with the sentence of [ra_message_no_mpfr()].

Value

A logical scalar.

Methodological notes

The fast level is not a degraded mode. Its enclosures are valid enclosures and its verdicts are true verdicts; what the rigorous level adds is the ability to narrow a case whose distance to the decision boundary is smaller than the declared slack. Absent the backend, such a case is reported as undecided at the available budget, which is a word, not a silence.

Dependencies

Probes 'Rmpfr', which is in `Suggests`.

References

Maechler, M. (2024). Rmpfr: R MPFR - multiple precision floating-point reliable (Version 1.1-2) [Software]. Comprehensive R Archive Network. https://doi.org/10.32614/CRAN.package.Rmpfr

See Also

[ra_to_double()], the bridge that needs it, and [ra_precision_ladder()].

Examples

```
ra_has_mpfr()
```

`ra_interval` *Build an interval*

Description

Creates a decorated interval from its endpoints, checking the invariant that makes it an interval at all, and attaching the decoration the standard prescribes when none is named.

Usage

```
ra_interval(lo, hi = lo, decoration = NULL)
```

Arguments

`lo`	A numeric vector of lower endpoints. `-Inf` is allowed as a bound; `+Inf` is not, because no nonempty interval has it as a lower bound.
`hi`	A numeric vector of upper endpoints, recycled against `lo`. `+Inf` is allowed as a bound and `-Inf` is not.
`decoration`	A character vector of decorations, one of `"com"`, `"dac"`, `"def"`, `"trv"` or `"ill"`, recycled. When left as `NULL`, the initial decoration of the standard is used: `"com"` for a nonempty bounded interval, `"dac"` for an unbounded one and `"trv"` for the empty one.

Details

The invariant `lo <= hi` is checked here and never again. Every operation of the package preserves it as a theorem, and checking it downstream would be checking arithmetic that was already proved; what it would catch is a defect in this package, which is what the test suite is for and not what a user's run should pay for.

Infinite endpoints are bounds and never members: `[-Inf, Inf]` is the whole real line and not the extended real line, so `Inf` is not in it. This is why `[Inf, Inf]` is not an interval and is refused rather than silently turned into something else.

A missing endpoint produces Not-an-Interval rather than an error, because a missing value is a statement about knowledge and not a violated contract; an inverted pair of known endpoints is a violated contract and raises.

Value

An object of class `ra_ivl`.

Methodological notes

The constructor raises where subclause 6.7.5 of IEEE Std 1788.1-2017 has the decorated constructor return Not-an-Interval. This is a declared divergence and it is deliberate. Note first what the subclause asks: a failed constructor both returns a datum *and* signals the `UndefinedOperation` exception, so what the standard forbids is failing in silence, not failing loudly. In R the loud form of that is a typed condition, and the quiet form – returning a value that looks like an interval and carries a defect inside it – is the one this package will not take by default, because in a language where nothing forces the caller to inspect a decoration, a Not-an-Interval returned by default travels. The value form is one call away for anyone who wants it: build it with [ra_nai()], or catch the condition, which is typed for exactly that.

The definition of an interval is subclause 4.2 of the standard; the decorations are its subclause 5.2, the permitted combinations 5.4, and the initial decoration 5.5.1.

Dependencies

Base R only.

References

Institute of Electrical and Electronics Engineers. (2018). IEEE standard for interval arithmetic (simplified) (IEEE Std 1788.1-2017). https://doi.org/10.1109/IEEESTD.2018.8277144

See Also

[ra_empty()], [ra_entire()], [ra_nai()] for the special values, and [ra_inf()] for the accessors.

Examples

```
ra_interval(1, 2)
ra_interval(c(-1, 0), c(1, 3))
ra_interval(-Inf, Inf)
tryCatch(ra_interval(2, 1), ra_bad_endpoints = function(cnd) "refused")
```

`ra_interval_to_text` *The text output of the interval standard*

Description

Converts intervals to strings that are valid interval literals and that contain the interval they came from, which is what subclause 6.8.3 of IEEE Std 1788.1-2017 asks of `intervalToText`.

Usage

```
ra_interval_to_text(x, cs = NULL)
```

Arguments

`x`	An object of class `ra_ivl`.
`cs`	A character scalar, the conversion specifier of 6.8.3, or `NULL`. The recognised values are `"decorated"`, which is the default and appends the decoration, and `"bare"`, which omits it. Any other value is invalid and is refused rather than ignored, because 6.8.3 allows an implementation to define which specifiers it recognises but not to accept one silently and do something else.

Details

The difference from `format()` is small and entirely about syntax, because the containment is the same and comes from the same place. Infinities are written `inf` and `-inf`, which is the spelling of 6.6.1(d) and not the `Inf` of R; and the asterisk that `format()` puts on a measured enclosure is dropped, because a literal has no place to carry provenance and a string that cannot be read back is not the output of this subclause. The provenance is still readable with [ra_prov()], and a caller writing a certificate should carry it in its own field rather than in the number.

What the string means is fixed by 6.6.2: the value of the literal `[l, u]` is the *mathematical* interval `[l, u]`, the decimals read exactly. That, and not "the string reads back to the same double", is the criterion these strings satisfy.

Value

A character vector of the length of `x`.

Methodological notes

The tightness of the enclosure is left implementation-defined by 6.8.3, and what this implementation defines is stated rather than left to be measured: the value of the string is contained in `[ra_pred(lo), ra_succ(hi)]`, so converting an interval to text costs no more than one operation of the arithmetic costs.

The uncertain form of 6.6.2, `m?r`, is not emitted. It is often shorter for a narrow interval, and a later version may take a specifier for it; nothing here depends on it, and a form that is not emitted is better absent than half-written.

Dependencies

Base R only.

References

Institute of Electrical and Electronics Engineers. (2018). IEEE standard for interval arithmetic (simplified) (IEEE Std 1788.1-2017). https://doi.org/10.1109/IEEESTD.2018.8277144

See Also

[format.ra_ivl()] for the console rendering, which encloses in the same way but spells the infinities as R does.

Examples

```
ra_interval_to_text(ra_interval(1/3, 1/3 + 1e-12))
ra_interval_to_text(ra_entire(), cs = "bare")
```

---

ra_measures *Derived quantities of an interval*

Description

The width, midpoint, radius, magnitude and mignitude, computed so that each one is itself an outward bound of the quantity it names.

Usage

```
ra_wid(x)

ra_mid(x)

ra_rad(x)

ra_mag(x)

ra_mig(x)
```

Arguments

x	An object of class ra_ivl.

Details

The width is rounded upward and the radius is rounded upward, so that neither can be reported smaller than it is; a width reported too small is what would let a stopping rule declare a box narrow enough when it is not. The midpoint is rounded to nearest and is the one quantity here that is not a bound: it is a point of the interval, chosen for expansion, and the operations that use it re-enclose whatever they compute from it.

The magnitude is the largest absolute value attained on the interval and the mignitude is the smallest; the mignitude is zero exactly when the interval contains zero, which is how the division and the monotonicity test ask that question.

Value

A numeric vector of the length of `x`. The empty interval gives `NaN` for all five, and Not-an-Interval gives `NaN`.

Methodological notes

Reporting the midpoint without an outward step is safe only because no guarantee rests on it. The Newton operator may expand about any point of the box and remains valid; the choice of the midpoint is an efficiency decision, and its rounding error is absorbed by the enclosure of the function value computed there.

The definitions of the midpoint, radius, magnitude and mignitude are section 1.6 of Neumaier (1990).

Dependencies

Base R only.

References

Neumaier, A. (1990). Interval methods for systems of equations. Cambridge University Press. https://doi.org/10.1017/CBO9780511526473

See Also

[ra_inf()].

Examples

```
x <- ra_interval(c(-2, 1), c(3, 4))
ra_wid(x)
ra_mid(x)
ra_mag(x)
ra_mig(x)
```

`ra_measure_library_error` *Measure the error of the evaluation path the fast level actually uses*

Description

Evaluates each admitted function over a mesh of its domain, both through the included fast implementation and through the multiprecision backend, and reports the largest disagreement in units in the last place beside the bound declared for the included implementation.

Usage

```
ra_measure_library_error(n = 20000L, seed = NULL, bits = 300L)
```

Arguments

`n`	An integer scalar, how many arguments to try per function. Defaults to 20000.
`seed`	An integer scalar seeding the arguments, or `NULL`, the default, in which case the state of the random number generator is left exactly as the caller had it and no seed is set inside the function.
`bits`	An integer scalar, the working precision of the referent. Defaults to 300.

Details

The slack of the fast level is `ceiling(2 * e + 1)` units in the last place over the declared bound `e = 0.5`. The fifteen included CORE-MATH kernels are correctly rounded under their contract, and the binary64 hardware square root is correctly rounded under IEEE 754. This function measures the installed package's path against an independent multiprecision referent.

The error is computed in multiprecision and not in binary64. Computed in binary64 it quantises to whole units in the last place, which cannot resolve whether an observed disagreement lies below the half-unit bound.

Value

A data frame with one row per admitted function and the columns `fun`, `n`, `observed`, `published`, `slack` and `used`, the last being the observed error as a fraction of the slack.

Methodological notes

The figure returned is a lower bound found by sampling. It is reported and never used to set the slack: a slack fitted to an observed error would be a calibration wearing the clothes of a convention, and the pre-registration of this phase forbids widening a declared number to accommodate a measurement. The measurement instead checks that the installed route stays within its declared half-unit bound and reports how much of the two-step slack is being spent.

Dependencies

Requires 'Rmpfr', which is in `Suggests`, and raises `ra_no_mpfr` without it, since there is no referent to measure against.

References

Sibidanov, A., Zimmermann, P., & Glondu, S. (2022). The CORE-MATH project. In 2022 IEEE 29th Symposium on Computer Arithmetic (ARITH) (pp. 26-34). IEEE. https://doi.org/10.1109/ARITH54963.2022.00014

See Also

[ra_slack()] and [ra_elem()].

Examples

```
if (ra_has_mpfr()) {
  head(ra_measure_library_error(n = 500L, seed = 80L), 4)
}
```

`ra_message_no_mpfr` *Report what the rigorous level would give and why it is unavailable*

Description

Produces the sentence the package uses when a computation that would have been resolved at the rigorous level cannot be, because the optional backend is not installed. The sentence names the backend, names what is lost, and names what still works without it.

Usage

```
ra_message_no_mpfr(what)
```

Arguments

`what`	A character scalar naming the operation that was being attempted.

Details

The wording follows the clean-planting pattern: a missing suggested package produces a statement of consequence, never a silent degradation and never an error. The fast level remains fully available and its enclosures remain valid; what is lost is the escalation ladder that would narrow a verdict whose distance to the decision boundary is smaller than the declared slack of the fast level.

Value

A character scalar holding the message. It is returned rather than signalled, because a missing optional backend is not a violated contract: the fast level answers, and the caller is told what a rigorous level would have added.

Methodological notes

'Rmpfr' sits in `Suggests` and is probed with `requireNamespace()` at the point of use, never at load time, so the package installs and runs without it.

Dependencies

Names 'Rmpfr', which is in `Suggests`.

References

Maechler, M. (2024). Rmpfr: R MPFR - multiple precision floating-point reliable (Version 1.1-2) [Software]. Comprehensive R Archive Network. https://doi.org/10.32614/CRAN.package.Rmpfr

Fousse, L., Hanrot, G., Lefevre, V., Pelissier, P., & Zimmermann, P. (2007). MPFR: A multiple-precision binary floating-point library with correct rounding. ACM Transactions on Mathematical Software, 33(2), Article 13. https://doi.org/10.1145/1236463.1236468

See Also

[ra_has_mpfr()], the probe this message accompanies.

Examples

```
cat(ra_message_no_mpfr("the escalation of exp() to 212 bits"), "\n")
```

`ra_newton_step` *One step of the interval Newton operator*

Description

Applies one Hansen-Sengupta step to a box: the midpoint is evaluated, the derivative is enclosed over the box, and the Newton image through extended division is intersected with the box.

Usage

```
ra_newton_step(
  e,
  x,
  var = "x",
  env = list(),
  level = c("fast", "rigorous"),
  precision = ra_precision_ladder()[1L]
)
```

Arguments

e	A call or expression over the closed operator table.
x	An object of class ra_ivl of length one: the box.
var	A character scalar naming the variable of e.
env	A named list of values, interval or numeric, for the other symbols of e.
level	The evaluation level, as in [ra_elem()].
precision	The precision of the rigorous level, in bits.

Details

The step is the Hansen-Sengupta form: the extended division returns one or two pieces when the derivative straddles zero, and each piece is intersected with the box. A single piece strictly interior to the box, together with a derivative enclosure excluding zero, is the certificate that the box holds exactly one root: existence by the fixed-point argument, uniqueness by monotonicity.

Value

An object of class ra_ivl with zero, one or two elements: the pieces of the Newton image intersected with the box. Zero pieces demonstrate the box holds no root. The attribute ra_contracted is TRUE when there is a single piece strictly interior to the box, and the attribute ra_dspan is TRUE when the enclosed derivative contains zero.

Methodological notes

The uniqueness hypothesis stated here is sufficient and not minimal: the reference theorem derives the regularity of the derivative from the strict interior inclusion rather than assuming it. In one dimension the two formulations are operationally equivalent, and the package asserts the stronger antecedent because it is the one the certificate consumer can read off the returned attributes.

Dependencies

Base R at the fast level; the rigorous level requires 'Rmpfr'.

References

Hansen, E., & Walster, G. W. (2004). Global optimization using interval analysis (2nd ed.). Marcel Dekker.

Neumaier, A. (1990). Interval methods for systems of equations. Cambridge University Press. https://doi.org/10.1017/CBO9780511526473

See Also

[ra_solve()] for the paving that drives the step, [ra_div_extended()] for the division underneath it.

Examples

```
ra_newton_step(quote(exp(x) - 2), ra_interval(0, 1))
```

`ra_operator_table` *The closed table of symbols an expression may use*

Description

Returns the symbols this version of the package admits inside an expression it is asked to enclose, together with the error bound declared by the included fast implementation and the slack the fast level adds on top of it.

Usage

```
ra_operator_table(what = c("functions", "operators", "excluded", "all"))
```

Arguments

`what`	A character scalar choosing what to return: `"functions"` for the unary functions with their errors and slacks, `"operators"` for the operators and grouping symbols, `"excluded"` for the symbols that are refused and the criterion each one fails, or `"all"` for a list holding the three. Defaults to `"functions"`.

Details

A symbol is admitted only if it satisfies four conditions at once, all of them checked against either the included implementation or the installed software that supplies the remaining operation.

The first is that the installed `stats::D` differentiates it, since the centered form and the monotonicity test both need a derivative and the package does not carry a differentiation engine of its own.

The second is that the table is closed under differentiation: the derivative of an admitted symbol, and the derivative of that, and so on until the set stops growing, must be writable with admitted symbols. This is the condition that a table assembled by taste tends to fail, and failing it is not a nuisance but a hole: the mean-value form leaves the table on its first step and there is nothing to evaluate.

The third is that the package includes a binary64 implementation with a declared error bound in units in the last place. The fifteen CORE-MATH kernels are correctly rounded, and the hardware square root is correctly rounded by the floating-point standard, so the bound is 0.5 for each.

The fourth is that the multiprecision backend provides the function with correct rounding, since otherwise the rigorous level and the escalation ladder have nowhere to go and a verdict lying inside the slack could never be resolved.

The slack is `ceiling(2 * e + 1)` units in the last place, where `e = 0.5` is the declared correctly rounded bound. The result is two outward steps for every function. The convention is named and cited rather than tuned, and it is never widened to accommodate a failing point; a point outside the slack is investigated instead.

Value

For `"functions"`, a data frame with columns `fun`, `ulp` and `slack`. For `"operators"`, a character vector. For `"excluded"`, a data frame with columns `symbol` and `reason`. For `"all"`, a named list with all three.

Methodological notes

The entry for `sqrt` is deliberately uniform. Correct rounding of the square root is required by the floating-point standard, so one outward step would already enclose it. The table nonetheless applies the same pre-registered formula and the same two steps as to every CORE-MATH kernel; the benefit is a table with no function-specific slack rule to get wrong.

The correctly rounded contract of the software kernels is described by Sibidanov et al. (2022); the correct rounding of square root is clause 5.4.1 of IEEE Std 754-2019.

Dependencies

Uses `stats::D` to close the table, and probes 'Rmpfr' from `Suggests` for the fourth criterion.

References

Sibidanov, A., Zimmermann, P., & Glondu, S. (2022). The CORE-MATH project. In 2022 IEEE 29th Symposium on Computer Arithmetic (ARITH) (pp. 26-34). IEEE. https://doi.org/10.1109/ARITH54963.2022.00014

Institute of Electrical and Electronics Engineers. (2019). IEEE standard for floating-point arithmetic (IEEE Std 754-2019). https://doi.org/10.1109/IEEESTD.2019.8766229

See Also

[ra_slack()], which reads a single entry of the table.

Examples

```
tab <- ra_operator_table()
tab[tab$fun %in% c("exp", "tanh"), ]
ra_operator_table("operators")
head(ra_operator_table("excluded"), 3)
```

---

ra_parts *Read the parts of an interval*

Description

Extract the lower endpoint, the upper endpoint, the decoration, or the answers to the questions of whether an interval is empty, the whole line, or Not-an-Interval.

Usage

```
ra_inf(x)

ra_sup(x)

ra_dec(x)

ra_is_nai(x)
```

```
ra_is_empty(x)

ra_is_entire(x)
```

Arguments

x	An object of class ra_ivl.

Details

The endpoints of the empty interval are the reversed pair the representation uses; a caller who wants to branch on emptiness should ask [ra_is_empty()] rather than compare endpoints, so that the encoding stays an internal matter.

Value

Numeric vectors for the endpoints, a character vector for the decoration, and logical vectors for the three questions, all of the length of x.

Methodological notes

The decoration is the provenance field of a result, not a second opinion about it. An interval decorated trv is exactly as valid an enclosure as one decorated com; what differs is how much the operation was able to assert about the function it evaluated.

The operations follow subclauses 4.5.7 and 5.5.2 of IEEE Std 1788.1-2017.

Dependencies

Base R only.

References

Institute of Electrical and Electronics Engineers. (2018). IEEE standard for interval arithmetic (simplified) (IEEE Std 1788.1-2017). https://doi.org/10.1109/IEEESTD.2018.8277144

See Also

[ra_wid()] for the derived quantities.

Examples

```
x <- ra_interval(c(1, -Inf), c(2, Inf))
ra_inf(x)
ra_sup(x)
ra_dec(x)
ra_is_entire(x)
```

ra_paving_show *Show a paving of certified roots*

Description

The card of a paving: its window, its certified roots with their provenance, its abstentions, whether absence holds on the rest, and the budget spent – each verdict a word, never a silence.

Usage

```
## S3 method for class 'ra_paving'
format(x, ...)

## S3 method for class 'ra_paving'
print(x, ...)

## S3 method for class 'ra_paving'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'ra_paving'
summary(object, ...)

## S3 method for class 'ra_paving_summary'
print(x, ...)
```

Arguments

x, object	An object of class ra_paving.
row.names, optional, ...	Passed along as in the generics.

Details

The card never prints a paving without its weakest provenance. A verdict says theorem either when its fast evaluation used only the theorem-level arithmetic or when it was re-verified at the rigorous level. A verdict that still relies on a measured fast-level evaluation says measured, loudly. Absence over the remainder of the window is only claimed when the paving completed within its budget.

Value

print() returns its argument invisibly; format() a character vector; as.data.frame() a data frame with one row per verdict and columns lo, hi, verdict and prov; summary() an object of class ra_paving_summary with its own print(); both report whether every verdict has theorem provenance.

Methodological notes

These methods are written together with the class rather than added later, because a class whose vector behaviour arrives in a second pass spends the interval between the two passes producing plausible wrong numbers.

Dependencies

Base R.

References

Wickham, H. (2019). Advanced R (2nd ed.). Chapman & Hall/CRC. https://doi.org/10.1201/9781351201315

See Also

[ra_solve()].

Examples

```
print(ra_solve(quote(x^2 - 2), ra_interval(0, 3)))
```

---

ra_periodic_frontier *Where the periodic enclosures stop resolving, and why*

Description

Measures the magnitude above which an interval of a given relative width can no longer be enclosed by the sine more narrowly than the whole range, at each of the two levels, and reports it beside the magnitude at which the spacing of binary64 makes that answer the correct one.

Usage

```
ra_periodic_frontier(
  rel_width = 2^-20,
  n = 16L,
  max_exponent = 80L,
  seed = NULL
)
```

Arguments

`rel_width`	A numeric scalar, the width of the probe interval as a fraction of its magnitude. Defaults to `2^-20`.
`n`	An integer scalar, how many probe points to draw per exponent. Defaults to 16.
`max_exponent`	An integer scalar, where to stop looking. Defaults to 80.
`seed`	An integer scalar seeding the probe points, or `NULL`, the default, in which case the state of the random number generator is left exactly as the caller had it and no seed is set inside the function. The two levels probe the same points either way, because the points are drawn once and shared, not redrawn per level.

Details

The enclosure of the sine over a box degenerates to the whole range when the interval of reduction indices contains an integer, which happens once the box is about a period wide. For a box of relative width `r` and magnitude `a`, that is `a * r >= 2 * pi`, so the frontier is predicted at the exponent `ceiling(log2(2 * pi / r))`, and the measured value is reported against that prediction rather than on its own.

Value

A list with the probe width, the measured exponent at each level, the exponent predicted by the spacing of the format, and a witness magnitude below the fast frontier where the enclosure is strictly narrower than the whole range. The rigorous entries are `NA` when the backend is absent.

Methodological notes

The number this reports corrects an expectation that is easy to hold and wrong: that the frontier is set by the precision of the argument reduction, and that the rigorous level would therefore push it far out. It is not. The rounding of the reduction contributes about one unit in the last place of the index, while the box contributes its own width divided by the period, and no nondegenerate box is narrower than one unit in the last place of its magnitude. The reduction can therefore widen the index interval by a small factor and never by an order. Above magnitude `2^52 * 2 * pi`, consecutive doubles are more than a period apart and the whole range is not a loss but the exact answer, at either level.

Dependencies

Base R; probes 'Rmpfr' for the rigorous frontier and reports `NA` for it when the backend is absent.

References

Muller, J.-M., Brunie, N., de Dinechin, F., Jeannerod, C.-P., Joldes, M., Lefevre, V., Melquiond, G., Revol, N., & Torres, S. (2018). Handbook of floating-point arithmetic (2nd ed.). Birkhauser. https://doi.org/10.1007/978-3-319-76526-6

See Also

[ra_elem()].

Examples

```
fr <- ra_periodic_frontier()
c(measured = fr$fast_exponent, predicted = fr$granularity_exponent)
```

`ra_powers` *Square, absolute value and integer powers*

Description

Operations whose interval version is tighter than the same expression written with multiplication, because the argument occurs once in the mathematics even though it occurs twice in the formula.

Usage

```
ra_sqr(x)

ra_abs(x)

ra_pown(x, p)
```

Arguments

`x`	An object of class `ra_ivl`, or a numeric vector.
`p`	An integer scalar exponent, which may be negative.

Details

The square of `[-1, 2]` is `[0, 4]` and not `[-2, 4]` as `x * x` gives: multiplication cannot know that the two factors are the same number, so it allows one to be negative while the other is positive. This is the dependency problem in its smallest form, and providing the operation separately is the cheapest of its mitigations.

Integer powers are computed by repeated squaring of intervals, so that every intermediate product carries its own outward step and the result is an enclosure by construction. The alternative, raising the endpoints with the power operator of R and widening by one step, is wrong and not merely loose: for an integer exponent R multiplies repeatedly, so its error grows with the exponent and a single outward step stops covering it somewhere above the cube.

The price of doing it this way is width, and the width is bounded rather than assumed. The bound is linear in the exponent even though the number of operations is logarithmic in it, and the reason is worth stating: the relative error of each rounded operation is carried into the next and multiplied by it, so squaring one's way to the `p`-th power accumulates a relative error of about `p` times the unit roundoff, which is about `p` steps in units in the last place of the result. Measured here: one step at `p = 2`, three at `p = 3`, five at `p = 5`, nine at `p = 8` and fifteen at `p = 16`. For the exponents this package meets in practice, which are small, that is negligible against the widths involved; for a large exponent the rigorous level is the place to go.

Value

An object of class `ra_ivl`.

Methodological notes

The exponent zero returns the point interval one, including for an argument containing zero, which is the convention of the standard for the integer-power operation. It is a convention and it is stated rather than assumed: the alternative reading, that zero to the zero has no value, belongs to the two-argument power and not to this one.

The entries for the square, the absolute value and the integer power are in Table 4.1 of IEEE Std 1788.1-2017; chapter 5 of Moore et al. (2009) treats the dependency these forms avoid.

Dependencies

Base R only.

References

Institute of Electrical and Electronics Engineers. (2018). IEEE standard for interval arithmetic (simplified) (IEEE Std 1788.1-2017). https://doi.org/10.1109/IEEESTD.2018.8277144

Moore, R. E., Kearfott, R. B., & Cloud, M. J. (2009). Introduction to interval analysis. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9780898717716

See Also

[Ops.ra_ivl()].

Examples

```
ra_sqr(ra_interval(-1, 2))
ra_interval(-1, 2) * ra_interval(-1, 2)
ra_abs(ra_interval(-3, 1))
ra_pown(ra_interval(2, 3), 3L)
ra_pown(ra_interval(2, 3), -1L)
```

---

ra_precision_ladder *The ladder of working precisions*

Description

Returns the sequence of precisions, in bits, that the rigorous level climbs when a verdict cannot be reached at the current one.

Usage

```
ra_precision_ladder()
```

Details

The rungs double. A verdict that is still undecided at the top rung is reported as undecided at that budget, with the number of bits printed; the ladder is never extended silently and the deciding criterion is never loosened to manufacture a verdict.

Value

An integer vector, in increasing order.

Methodological notes

Doubling rather than incrementing is deliberate. The cost of a rung is superlinear in its precision, so a fine ladder spends most of its budget on rungs that were never going to decide anything; and a case that fails to resolve after doubling is usually a case that is genuinely on the boundary rather than one that needed a few more bits.

Dependencies

Base R only.

References

Revol, N., & Rouillier, F. (2005). Motivations for an arbitrary precision interval arithmetic and the MPFI library. Reliable Computing, 11(4), 275-290. https://doi.org/10.1007/s11155-005-6891-y

See Also

[ra_to_double()].

Examples

```
ra_precision_ladder()
```

---

ra_pred *Predecessor of a double in round-to-nearest*

Description

Returns, for every element of `x`, a double strictly smaller than it and no greater than its immediate predecessor in binary64. The computation is performed in the rounding mode already in force and does not change it.

Usage

```
ra_pred(x)
```

Arguments

`x`	A numeric vector. Integer vectors are coerced.

Details

The formula is `x - (phi * abs(x) + eta)`. Everything said in the documentation of [ra_succ()] about validity, tightness and the two arguments handled outside the formula applies here with the signs reversed: the exception is `+Inf`, where the result is the largest finite double, and the saturation is toward `-Inf`.

Value

A numeric vector of the same length as `x`. Missing values and `NaN` propagate.

Methodological notes

The pair is used in exactly one way: an arithmetic result computed in round-to-nearest lies within half a unit in the last place of the exact value, so the exact value lies between the predecessor and the successor of the computed one. That sentence is the whole outward-rounding layer of the fast level, and every enclosure the package returns is built from it.

The comparison point in the floating-point standard is the nextDown operation of IEEE Std 754-2019, clause 5.3.1.

Dependencies

Base R only.

References

Rump, S. M., Zimmermann, P., Boldo, S., & Melquiond, G. (2009). Computing predecessor and successor in rounding to nearest. BIT Numerical Mathematics, 49(2), 419-431. https://doi.org/10.1007/s10543-009-0218-z

Institute of Electrical and Electronics Engineers. (2019). IEEE standard for floating-point arithmetic (IEEE Std 754-2019). https://doi.org/10.1109/IEEESTD.2019.8766229

See Also

[ra_succ()], its mirror image, and [ra_constants()].

Examples

```
ra_pred(1) == 1 - 2^-53
ra_pred(0) == -2^-1074
ra_pred(Inf) == .Machine$double.xmax
```

`ra_prov` *The provenance of an interval's guarantee*

Description

Returns, for each element, whether its enclosure is guaranteed by theorem or by a measured convention, which is the word the fast level carries.

Usage

```
ra_prov(x)
```

Arguments

`x`	An object of class `ra_ivl`.

Details

Exact endpoints and everything the kernel derives from them by outward rounding are theorems; so is the rigorous level, whose backend rounds correctly by contract. The fast level is a measured convention: a declared slack over an error measured on the route in use. Mixture propagates the weaker word, and the printed form of an interval repeats it, so a reader cannot mistake a convention for a theorem.

Value

A character vector, one of `"theorem"` or `"measured"` per element.

Methodological notes

An object missing the field, which can only come from code written before this safeguard, reports the weaker word. Defaulting to the stronger one would turn a forgotten assignment into an overclaim, which is the exact failure mode this field exists to prevent.

Dependencies

Base R.

References

Rump, S. M. (2010). Verification methods: Rigorous results using floating-point arithmetic. Acta Numerica, 19, 287-449. https://doi.org/10.1017/S096249291000005X

See Also

[ra_elem()] for the two levels, [ra_interval()].

Examples

```
ra_prov(ra_interval(1, 2))
ra_prov(ra_elem("exp", ra_interval(0, 1)))
```

---

ra_round_out *Widen a pair of endpoints outward by one rounding step*

Description

Takes the endpoints of an interval computed in round-to-nearest and moves each one away from the other, so that the exact value of whatever produced them is enclosed.

Usage

```
ra_round_out(lo, hi)
```

Arguments

lo	A numeric vector of lower endpoints.
hi	A numeric vector of upper endpoints, of the same length as lo.

Details

The function exists so that the outward step appears once in the sources rather than at every call site, which is what keeps a missing step from being invisible. A missing outward step does not produce a wrong number: it produces a number that is right almost always and wrong at the boundary, which is the failure this package is built to prevent.

Value

A list with elements lo and hi, both numeric vectors of the length of the inputs.

Methodological notes

The step is applied unconditionally, including when the computed endpoints happen to be exact. Detecting exactness would require knowing the exact result, which is the thing not known; paying one rounding step of width for an operation that did not need it is the price of not having to know.

Dependencies

Base R only.

References

Rump, S. M. (2010). Verification methods: Rigorous results using floating-point arithmetic. Acta Numerica, 19, 287-449. https://doi.org/10.1017/S096249291000005X

See Also

[ra_pred()] and [ra_succ()].

Examples

```
w <- ra_round_out(0.1, 0.1)
w$lo < 0.1 && w$hi > 0.1
```

`ra_sentinels` *Independent sentinels of the fast level*

Description

Return the difficult binary64 cases used to verify the elementary-function evaluator included in the package whenever the namespace loads.

Usage

```
ra_sentinels()
```

Details

For the fifteen software kernels, the inputs are selected deterministically across the difficult-case files in the sealed CORE-MATH archive. Square-root inputs come from the sealed binary64 sample used by the package's independent kernel tests. `dev/gen_sentinels.R` evaluates every selected input with Rmpfr at 200 bits and converts the result once with `Rmpfr::asNumeric()`. It never obtains an expected value from the evaluator that the table guards.

Value

A data frame with eight rows for each admitted function and columns `fun`, `x`, `cr`, `role`, `source` and `source_index`. `cr` is the correctly rounded binary64 value independently computed for `x`.

Methodological notes

A self-check is informative only when its expected values do not share the implementation under examination. The generator therefore seals the input archive by its SHA-256 digest, derives the expected values through MPFR and stores those values as package data. Loading the package needs no optional dependency and does not regenerate the evidence.

Dependencies

Base R. Regeneration requires `Rmpfr`; ordinary use does not.

References

Fousse, L., Hanrot, G., Lefevre, V., Pelissier, P., & Zimmermann, P. (2007). MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Transactions on Mathematical Software, 33*(2), Article 13. https://doi.org/10.1145/1236463.1236468

Sibidanov, A., Zimmermann, P., & Glondu, S. (2022). The CORE-MATH project. In *2022 IEEE 29th Symposium on Computer Arithmetic (ARITH)* (pp. 26-34). IEEE. https://doi.org/10.1109/ARITH54963.2022.00014

See Also

`ra_check_sentinels()` for the bitwise verification and `ra_fast_level_status()` for the session state.

Examples

```
sentinels <- ra_sentinels()
table(sentinels$fun)
```

---

ra_sets *Set operations on intervals*

Description

The intersection of two intervals and the interval hull of their union.

Usage

```
ra_intersect(x, y)

ra_hull(x, y)
```

Arguments

`x`, `y`	Objects of class `ra_ivl`, or numeric vectors.

Details

Both are exact: they need no outward step, because the endpoints of the result are endpoints of the arguments and no arithmetic is performed. The intersection of two intervals that do not meet is the empty interval, which is a value and not a failure; it is how the Newton step proves that a box contains no root.

The hull of the union is not the union: the union of two disjoint intervals is not an interval, and the hull fills in the gap between them. Filling in the gap is a loss of information and never a loss of validity.

Value

An object of class `ra_ivl`.

Methodological notes

The standard classes these as operations that are not interval extensions of point functions and prescribes that their results carry the `trv` decoration, because no single decoration would be informative in every context in which they are used. The package follows that, and a caller who can justify a stronger decoration in a particular use is expected to attach it deliberately rather than receive it by default.

The classification and its decoration rule are subclauses 4.5.4 and 5.7.1 of IEEE Std 1788.1-2017.

Dependencies

Base R only.

References

Institute of Electrical and Electronics Engineers. (2018). IEEE standard for interval arithmetic (simplified) (IEEE Std 1788.1-2017). https://doi.org/10.1109/IEEESTD.2018.8277144

See Also

[ra_div_extended()], which produces the pieces these combine.

Examples

```
ra_intersect(ra_interval(0, 2), ra_interval(1, 3))
ra_is_empty(ra_intersect(ra_interval(0, 1), ra_interval(2, 3)))
ra_hull(ra_interval(0, 1), ra_interval(2, 3))
```

`ra_set_dec` *Attach a decoration to an interval*

Description

Sets the decoration of an interval, repairing the combinations the standard forbids rather than producing them or refusing.

Usage

```
ra_set_dec(x, decoration)
```

Arguments

`x`	An object of class `ra_ivl`.
`decoration`	A character vector of decorations, recycled to the length of `x`.

Details

Three combinations are contradictory and cannot be produced. The empty interval with `def`, `dac` or `com` becomes the empty interval with `trv`, because those three assert that the argument box was nonempty. An unbounded interval with `com` becomes the same interval with `dac`, because `com` asserts boundedness. Anything with `ill` becomes Not-an-Interval, whose interval part has no value.

The repairs are silent because they are definitional and not corrective: the requested decoration was not a weaker claim about the same object, it was a claim no object can carry, and the value returned is the strongest one that can be.

Value

An object of class `ra_ivl`.

Methodological notes

This is where a caller who knows something the arithmetic could not deduce attaches it. The set operations, in particular, are decorated `trv` by the standard because no single decoration is right for every use of them; a caller who can justify a stronger one in a specific context is expected to attach it here, deliberately and in one visible place.

The permitted combinations and the repairing operation are subclauses 5.4 and 5.5.2 of IEEE Std 1788.1-2017.

Dependencies

Base R only.

References

Institute of Electrical and Electronics Engineers. (2018). IEEE standard for interval arithmetic (simplified) (IEEE Std 1788.1-2017). https://doi.org/10.1109/IEEESTD.2018.8277144

See Also

[ra_interval()], [ra_dec()].

Examples

```
ra_dec(ra_set_dec(ra_empty(), "com"))
ra_dec(ra_set_dec(ra_entire(), "com"))
ra_is_nai(ra_set_dec(ra_interval(1, 2), "ill"))
```

ra_show *Show an interval*

Description

Render intervals for the console and for a data frame, with the decoration attached to each one and the special values named rather than printed as endpoint pairs.

Usage

```
## S3 method for class 'ra_ivl'
format(x, ...)

## S3 method for class 'ra_ivl'
print(x, ...)

## S3 method for class 'ra_ivl'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'ra_ivl'
summary(object, ...)

## S3 method for class 'ra_ivl_summary'
print(x, ...)
```

Arguments

x	An object of class ra_ivl.
...	Further arguments, ignored, present for consistency with the generics.
row.names, optional	Arguments of the as.data.frame generic. The first is used; the second is ignored.
object	An object of class ra_ivl, for the summary method.

Details

The empty interval prints as the word for it and not as the reversed endpoint pair that represents it, because the pair is an encoding and printing it would invite a reader to compare endpoints instead of asking. Not-an-Interval prints as its name for the same reason.

Value

format returns a character vector; print returns its argument invisibly; as.data.frame returns a data frame with columns lo, hi, dec and wid; summary returns an object of class ra_ivl_summary, which prints a count of each decoration and of each special value.

Methodological notes

The decoration is printed always and never suppressed when it is `com`. A field that appears only when it is interesting teaches a reader to stop looking for it, and the decoration is precisely the field that must be read on the results that look ordinary.

The printed pair encloses the interval, and the reader is owed that and not a tidy number. Subclause 6.8.3 of IEEE Std 1788.1-2017 asks of `intervalToText` that the string contain the interval it came from, and subclause 6.6.2 fixes what a string means: the value of the literal `[l, u]` is the *mathematical* interval `[l, u]`, that is, the decimals read exactly. Rounding each endpoint to nearest – which is what every printing default in R does – breaks that: the enclosure whose endpoints are the predecessor and the successor of one would print as `[1, 1]`, which contains neither. So the lower endpoint is rounded down and the upper endpoint up, each to the fewest digits that carry the proof.

Two consequences a reader meets immediately. An endpoint prints with as many digits as its *value* needs and not as many as its neighbour needs, so `[0.1, 0.2]` prints its endpoints at different lengths: the double nearest `0.1` is above the decimal `0.1`, so `0.1` is a true lower bound, while the double nearest `0.2` is also above the decimal `0.2`, so `0.2` is not a true upper bound and the next one is printed. And a degenerate interval prints as two different decimals, because a degenerate interval of doubles is not a real number: it is everything that rounds to one.

Numbers that are decimals exactly – `-2.5`, `3.75`, `1e-3`, every power of two – print exactly as they did, at their own length, because for them rounding down and rounding up are the number itself. Nothing is paid where nothing is owed.

What the widening costs is bounded and declared. The printed enclosure is contained in `[ra_pred(lo), ra_succ(hi)]`: printing an enclosure never costs more than one operation of the arithmetic costs, and the width on the page is the width in the object to within that.

What it costs in time is declared too, because it is not free: each endpoint is proved rather than formatted, which is of the order of a millisecond, and distinct values are the unit, so a vector whose endpoints repeat costs what its distinct endpoints cost. In the benchmark recorded for this implementation (seed 20260913, one warm-up and five timed runs), formatting 1,000 intervals with 2,000 distinct uniformly drawn endpoints took a median 1.805 seconds on an AMD Ryzen 9 5980HX. This wall-clock measurement is not a portable performance bound. Printing the handful of intervals a session looks at is imperceptible; formatting a vector of thousands is seconds, and a caller in that position wants `as.data.frame()`, which does no proving because it does no printing. The endpoints themselves are in `x$lo` and `x$hi`, and `as.data.frame()` carries them unrounded, which is where a reader who needs the doubles goes.

Dependencies

Base R only.

References

Institute of Electrical and Electronics Engineers. (2018). IEEE standard for interval arithmetic (simplified) (IEEE Std 1788.1-2017). https://doi.org/10.1109/IEEESTD.2018.8277144

See Also

[ra_interval()].

Examples

```
x <- ra_interval(c(1, -Inf), c(2, Inf))
format(x)
print(x)
as.data.frame(x)
summary(c(x, ra_empty(), ra_nai()))
```

`ra_slack` *Slack in units in the last place declared for one function*

Description

Returns the number of outward rounding steps the fast level adds to an evaluation of the named function, on top of the outward step that every rounded operation already receives.

Usage

```
ra_slack(fun)
```

Arguments

`fun`	A character scalar naming a function of the closed table.

Details

The value is `ceiling(2 * e + 1)` for the declared correctly rounded bound `e = 0.5` of the named function, hence two steps for every admitted function. Asking for a symbol outside the table raises `ra_symbol_not_in_table` rather than returning a default, because a default here would be a number chosen by nobody and applied to everything the table forgot.

Value

An integer scalar, the number of steps.

Methodological notes

The slack is a declared convention, not a calibration. It is not fitted to any observed failure and it is not adjusted when a sweep finds a point outside it; such a point is a finding about the evaluation, and the response is to investigate the point, not to enlarge the number until the point falls inside.

The bound for the fifteen software kernels follows from the correctly rounded contract described by Sibidanov et al. (2022). Square root follows the corresponding requirement of IEEE Std 754-2019.

Dependencies

Base R only.

References

Sibidanov, A., Zimmermann, P., & Glondu, S. (2022). The CORE-MATH project. In 2022 IEEE 29th Symposium on Computer Arithmetic (ARITH) (pp. 26-34). IEEE. https://doi.org/10.1109/ARITH54963.2022.00014

Institute of Electrical and Electronics Engineers. (2019). IEEE standard for floating-point arithmetic (IEEE Std 754-2019). https://doi.org/10.1109/IEEESTD.2019.8766229

See Also

[ra_operator_table()].

Examples

```
ra_slack("exp")
ra_slack("tanh")
tryCatch(ra_slack("gamma"), ra_symbol_not_in_table = function(cnd) "refused")
```

---

ra_solve *Certified enclosure of the roots of an expression over a window*

Description

Paves the window, drives the interval Newton operator over the paving, and returns every root region with one of three words: `unique`, a certificate of exactly one root; absence, demonstrated over everything excluded; or `not excludable`, the abstention.

Usage

```
ra_solve(
  e,
  over,
  var = "x",
  env = list(),
  min_width = NULL,
  max_boxes = 10000L,
  tol = 1e-10,
  precision = ra_precision_ladder()[1L]
)
```

Arguments

`e`	A call or expression over the closed operator table.
`over`	An object of class `ra_ivl` of length one: the window.
`var`	A character scalar naming the variable of `e`.
`env`	A named list of values for the other symbols of `e`.
`min_width`	The width below which a box that resists every test becomes an abstention. The default is declared, not calibrated: one millionth of the window width.

max_boxes	The budget of the paving, in boxes. At the budget the engine stops and says so; what it did not reach is reported not excludable, never dropped.
tol	The width at which a certified enclosure stops being tightened.
precision	The starting precision of the rigorous verification, in bits; it climbs the declared ladder when a verification cannot decide.

Details

The search runs at the fast level and carries the provenance of the evaluations that support each verdict. The arithmetic operators and integer powers have theorem provenance at that level; elementary functions have measured provenance. When the multiprecision backend is present, every uniqueness certificate is re-established at the rigorous level, where the enclosure is a theorem, and every excluded box is re-excluded there; a verification that cannot decide climbs the precision ladder and, at the top, moves its box to the abstention with the budget in the message. Without the backend there is no blanket downgrade: each verdict keeps the provenance of its fast evaluations, and the card labels the weakest provenance carried by the paving.

The certificate behind unique is the Hansen-Sengupta one: a Newton image strictly interior to its box with a derivative enclosure excluding zero proves exactly one root there. Boxes where no certificate and no exclusion is reached are fused, inflated so that a root sitting exactly on a box boundary passes to the interior, and certified there; without that step every simple root on a representable boundary point would remain uncertifiable forever, which is the measured lesson this design carries from its probe.

Value

An object of class ra_paving: a list with the certified roots (unique), the abstentions (not_excludable), the excluded cover (excluded), the budget spent, and the provenance of the verdicts. The logical field verified is true exactly when every verdict has theorem provenance. Its print() is the card.

Methodological notes

A multiple root has no certificate by this route: its derivative enclosure contains zero, no theorem applies, and pretending otherwise would manufacture a verdict. It stays not excludable, tight around the root. Likewise two roots closer than min_width cannot be separated and produce the abstention rather than a guess. The split of a box that resists contraction leaves the resisting candidate in the larger part, which is the finer cut of the reference algorithm.

Dependencies

Base R at the fast level. The rigorous verification requires 'Rmpfr', which is in Suggests; without it arithmetic-only verdicts retain theorem provenance and verdicts using elementary functions retain measured provenance.

References

Hansen, E., & Walster, G. W. (2004). Global optimization using interval analysis (2nd ed.). Marcel Dekker.

Neumaier, A. (1990). Interval methods for systems of equations. Cambridge University Press. https://doi.org/10.1017/CBO9780511526473

Rump, S. M. (2010). Verification methods: Rigorous results using floating-point arithmetic. Acta Numerica, 19, 287-449. https://doi.org/10.1017/S096249291000005X

See Also

[ra_newton_step()] for one step of the operator, [ra_enclose_expr()] for the enclosures underneath.

Examples

```
ra_solve(quote(x^2 - 2), ra_interval(0, 3))
```

`ra_specials` *The special intervals*

Description

Build the empty interval, the whole real line, and Not-an-Interval, each as a value of the interval class.

Usage

```
ra_empty(n = 1L)

ra_entire(n = 1L)

ra_nai(n = 1L)
```

Arguments

`n`	An integer scalar, how many to build. Defaults to one.

Details

The three are values and not errors. The empty interval is the proved answer to a question such as where a function with no zero in a box has one; the whole real line is the honest answer when nothing is known; Not-an-Interval is what an invalid construction produces when it is asked to produce a value rather than to stop.

Their decorations are fixed by the standard and are not free: the empty interval can only carry `trv`, because the decorations above it assert that the argument box is nonempty; the whole real line carries `dac` rather than `com`, because `com` asserts boundedness; and Not-an-Interval carries `ill` by definition.

Value

An object of class `ra_ivl` of length `n`.

Methodological notes

Making these values rather than exceptions is what allows the paving engine to say what it proved. A box proved to contain no root returns the empty interval, which is a theorem; a box the engine ran out of budget on returns an interval with a word attached. Neither is a silence and neither is an error.

The definitions are subclauses 4.5.1 and 5.3 of IEEE Std 1788.1-2017.

Dependencies

Base R only.

References

Institute of Electrical and Electronics Engineers. (2018). IEEE standard for interval arithmetic (simplified) (IEEE Std 1788.1-2017). https://doi.org/10.1109/IEEESTD.2018.8277144

See Also

[ra_interval()].

Examples

```
ra_empty()
ra_entire()
ra_nai()
ra_is_empty(ra_empty())
```

---

ra_spectral_sum *Bound the spectrum of a sum of symmetric matrices*

Description

Combines enclosures of the spectra of two symmetric matrices into an enclosure of the spectrum of their sum, by Weyl's inequalities.

Usage

```
ra_spectral_sum(a, b)
```

Arguments

a	An `ra_ivl` of length one enclosing the spectrum of the first matrix, or a numeric pair.
b	The same for the second matrix.

Details

Weyl's inequalities give `lambda_min(A + B) >= lambda_min(A) + lambda_min(B)` and `lambda_max(A + B) <= lambda_max(A) + lambda_max(B)`, so the sum of the enclosures encloses the spectrum of the sum. The addition is rounded outwards.

The bound is what lets a caller combine a summand whose spectrum is known in closed form with one that is only bounded, which is the usual shape of a metric built as a constant part plus a state-dependent one.

Value

An object of class `ra_ivl` of length one enclosing the spectrum of the sum.

References

Horn, R. A., & Johnson, C. R. (2013). Matrix analysis (2nd ed.). Cambridge University Press.

See Also

[ra_gershgorin()] for obtaining each enclosure.

Examples

```
ra_spectral_sum(ra_interval(1, 3), ra_interval(-1, 2))
```

---

ra_stop *Raise a typed error of the package*

Description

Builds and signals a condition whose class vector identifies the kind of contract that was violated, so that callers can catch one kind without catching the others and without matching on the message text.

Usage

```
ra_stop(kind, message, call = sys.call(-1L), ...)
```

Arguments

`kind`	A character scalar naming the kind of failure. It becomes the first element of the class vector, prefixed with `"ra_"`.
`message`	A character scalar with the message shown to the user.
`call`	The call to report as the origin of the condition. Defaults to the caller of the function that raises.
`...`	Further named fields stored in the condition object, for callers that want the offending values rather than their rendering.

Details

The three-layer class vector is what makes selective handling possible: `tryCatch(..., ra_empty_interval = )` catches exactly one case, `ra_error =` catches every failure of this package and nothing from any other, and `error =` catches it along with everything else.

Value

No return value. The function always signals a condition, so it never returns to its caller.

Methodological notes

The package distinguishes a violated contract, which raises, from an undecided verdict, which does not. An interval operation that cannot prove what was asked returns an enclosure or a verdict object carrying the word for its abstention; it does not raise and it does not fall silent. Errors are reserved for inputs that could not have a meaning at all, such as an interval whose lower endpoint exceeds its upper one.

Chapter 8 of Wickham (2019) treats the condition system and the classing of conditions for selective handling.

Dependencies

Base R only.

References

Wickham, H. (2019). Advanced R (2nd ed.). Chapman & Hall/CRC. https://doi.org/10.1201/9781351201315

See Also

[ra_interval()], which raises `ra_bad_endpoints` when its invariant is violated.

Examples

```
res <- tryCatch(
  ra_stop("empty_interval", "the interval is empty"),
  ra_empty_interval = function(cnd) class(cnd)[1L]
)
res
```

---

ra_succ *Successor of a double in round-to-nearest*

Description

Returns, for every element of `x`, a double strictly greater than it and no smaller than its immediate successor in binary64. The computation is performed in the rounding mode already in force and does not change it.

Usage

```
ra_succ(x)
```

Arguments

x	A numeric vector. Integer vectors are coerced.

Details

The formula is `x + (phi * abs(x) + eta)`, evaluated in round-to-nearest, which the cited theorem proves lies in the half-open range from the immediate successor of `x` upward, so that using it in place of the immediate successor widens an enclosure and never narrows one.

How much it widens is also a theorem and not a measurement. The second of the cited theorems states that the value returned is exactly the immediate successor for every finite argument outside the two binades around the subnormal threshold, which for binary64 are the magnitudes between `2^-1022` and `2^-1020`, and that inside that band it is one bit beyond. A sweep of 180092 arguments on this machine found the exceptions exactly where the theorem allows them and nowhere else: eight points, at `2^-1022` and `2^-1021` and their negatives.

Two arguments are handled outside the formula because the formula produces `NaN` there rather than a wrong number: at `-Inf`, where the result is the largest finite negative double, and at `NaN` itself, which propagates. At `+Inf` the formula returns `+Inf`, which is the correct saturation, and at the largest finite double it returns `+Inf`, which is a valid upper bound and is how an overflow leaves the arithmetic without pretending to a finite answer.

Value

A numeric vector of the same length as `x`. Missing values and `NaN` propagate.

Methodological notes

Changing the rounding mode of the floating-point unit, which is how directed rounding is usually obtained, is not available from R and would not be safe if it were: the evaluator, the just-in-time compiler and any linked numerical library sit between this code and the unit, and a directed mode that leaked into one of them would be a silent defect of the worst kind. This is the reason the package pays the cost of a formula instead. The declared price is that each fast-level operation overestimates by at most one unit in the last place per endpoint relative to true directed rounding.

The comparison point in the floating-point standard is the nextUp operation of IEEE Std 754-2019, clause 5.3.1.

Dependencies

Base R only.

References

Rump, S. M., Zimmermann, P., Boldo, S., & Melquiond, G. (2009). Computing predecessor and successor in rounding to nearest. BIT Numerical Mathematics, 49(2), 419-431. https://doi.org/10.1007/s10543-009-0218-z

Institute of Electrical and Electronics Engineers. (2019). IEEE standard for floating-point arithmetic (IEEE Std 754-2019). https://doi.org/10.1109/IEEESTD.2019.8766229

See Also

[ra_pred()], its mirror image, and [ra_constants()].

Examples

```
ra_succ(1) == 1 + 2^-52
ra_succ(0) == 2^-1074
ra_succ(.Machine$double.xmax)
ra_succ(-Inf) == -.Machine$double.xmax
```

---

ra_to_double *Convert a multiprecision number to a double in a chosen direction*

Description

Turns a multiprecision value into a binary64 value that is guaranteed to lie on the requested side of it, so that a rigorous evaluation can be returned as an ordinary interval without losing its guarantee at the last step.

Usage

```
ra_to_double(x, direction = c("down", "up"))
```

Arguments

x	An object of class mpfr.
direction	A character scalar, either "down" or "up".

Details

The path is the one measured on the installed backend: round the multiprecision value to the precision of binary64 in the requested direction with Rmpfr::roundMpfr(), then cross to a double with Rmpfr::toNum() in the same direction. Two roundings are needed rather than one because the first fixes the significand and the second fixes the type, and only the second knows about the limits of the destination format.

The arithmetic operators of the backend were measured and do not accept a rounding mode; they evaluate to nearest. This costs nothing, because the multiprecision library rounds every one of its functions correctly by contract, so evaluating to nearest at a precision well above binary64 and then rounding once in the wanted direction gives an enclosure whose width is at most one unit in the last place, by theorem rather than by measurement.

Value

A numeric vector of the same length as x.

Methodological notes

`as.numeric()` must not be used for this crossing, and the package contains a test whose purpose is to fail if anyone tries. It underflows to zero and overflows to infinity in silence: a positive multiprecision value below the smallest subnormal becomes zero, which is a valid lower bound and an invalid upper bound, and nothing in the returned value says which of the two it was asked to be. `toNum()` in a directed mode was measured to respect the subnormal range and to saturate at the largest finite double downward and at infinity upward, which is what an outward bound needs.

Dependencies

Requires 'Rmpfr', which is in `Suggests`. Raises `ra_no_mpfr` if it is absent, because unlike an escalation this function has no fast-level answer to fall back to: it was handed a multiprecision object, which could only have come from the backend.

References

Fousse, L., Hanrot, G., Lefevre, V., Pelissier, P., & Zimmermann, P. (2007). MPFR: A multiple-precision binary floating-point library with correct rounding. ACM Transactions on Mathematical Software, 33(2), Article 13. https://doi.org/10.1145/1236463.1236468

See Also

[ra_has_mpfr()] and [ra_precision_ladder()].

Examples

```
if (ra_has_mpfr()) {
  third <- Rmpfr::mpfr("1", 120L) / 3
  lo <- ra_to_double(third, "down")
  hi <- ra_to_double(third, "up")
  c(lo < 1 / 3 || lo == 1 / 3, hi > 1 / 3 || hi == 1 / 3)
}
```

`ra_vector` *Combine and subset intervals*

Description

The concatenation, length and subsetting a vector of intervals needs to behave as a vector of intervals rather than as the list underneath it.

Usage

```
## S3 method for class 'ra_ivl'
length(x)

## S3 method for class 'ra_ivl'
c(...)
```

```
## S3 method for class 'ra_ivl'
x[i]
```

Arguments

`x`	An object of class `ra_ivl`.
`...`	Objects of class `ra_ivl` to concatenate.
`i`	An index, of any form `[` accepts for a numeric vector.

Details

Without these, `c()` on two intervals produces a list of two lists and `length()` reports three, the number of fields. Both would be wrong in a way that only shows up far downstream.

Value

An object of class `ra_ivl`, or an integer for `length`.

Methodological notes

These are written here with the class rather than added later, because a class whose vector behaviour arrives in a second pass spends the interval between the two passes producing plausible wrong numbers.

Chapter 13 of Wickham (2019) treats S3 vector classes and the methods they must carry.

Dependencies

Base R only.

References

Wickham, H. (2019). Advanced R (2nd ed.). Chapman & Hall/CRC. https://doi.org/10.1201/9781351201315

See Also

[ra_interval()].

Examples

```
x <- c(ra_interval(1, 2), ra_interval(3, 4))
length(x)
x[2]
```

`ra_verify_monotonicity` *Verify the declared shapes against the functions themselves*

Description

Runs each admitted function over a mesh of its own domain and reports whether the shape the package declares for it is the shape it has.

Usage

```
ra_verify_monotonicity(classes = NULL, n = 4001L, span = 8)
```

Arguments

`classes`	An optional named character vector overriding the declared shape of the named functions, used to check that a wrong declaration is detected. Defaults to `NULL`, meaning the declared table.
`n`	An integer scalar, the number of mesh points per function. Defaults to 4001.
`span`	A numeric scalar, the half-width of the mesh for functions whose domain is unbounded. Defaults to 8, which spans more than two periods.

Details

The shape is read from the signs of the successive differences of the function on the mesh, with the exact ties dropped so that a saturating function such as the hyperbolic tangent is not mistaken for a non-monotone one. No sign change is a monotone function; one change from falling to rising is a valley; more than one change is either a periodic function or one with poles, and the two are told apart by the length of the shortest run, since a pole reverses the sign of the difference for a single step and a genuine half-period lasts for hundreds.

Value

A list with three elements: `observed`, a character vector of the shape each function was found to have; `disagreeing`, the names of the functions whose declared shape differs from it; and `agrees`, a logical scalar that is `TRUE` when `disagreeing` is empty.

Methodological notes

The shape table is the one part of the elementary layer that could be wrong without any arithmetic being wrong, and the failure it would cause is a silent loss of containment rather than an error. Reading only the endpoints of the hyperbolic cosine over a box containing zero returns an interval that misses the true minimum by however much the box reaches to the left, and nothing in the result says so. This function exists so that the table is measured on the machine that will use it, and its own positive control is that a deliberately wrong entry passed through `classes` must appear in `disagreeing`.

Dependencies

Base R only.

References

Neumaier, A. (1990). Interval methods for systems of equations. Cambridge University Press. https://doi.org/10.1017/CBO9780511526473

See Also

[ra_elem()], which uses the table this verifies.

Examples

```
ra_verify_monotonicity()$agrees
ra_verify_monotonicity(classes = c(cosh = "increasing"))$disagreeing
```

---

ra_verify_operator_table

Re-close the table against the installed software

Description

Runs the admission criteria that depend on installed software again, here and now, against the derivative engine and the multiprecision backend of the machine the package is running on, and reports what it found.

Usage

```
ra_verify_operator_table(include_mpfr = ra_has_mpfr())
```

Arguments

`include_mpfr`	A logical scalar saying whether to test the fourth criterion, which needs the optional backend. Defaults to whether the backend is installed.

Details

The derivative engine and multiprecision backend underneath the table can move: a release of R can add a rule to the derivative table, and a release of the backend can implement a function it did not have. The correctly rounded fast kernels travel with the package and are tested separately. This function rechecks the moving criteria so that they are audited rather than trusted by whoever runs the package.

The closure is computed over head symbols rather than over whole expressions, which is what makes it terminate. Differentiating a function whose derivative is the same function at a higher order never repeats an expression, and a walk over expressions would not stop; it repeats its head immediately. The general power is seeded explicitly, because its rule is the only one that introduces a symbol present in neither operand, and it introduces it only when the exponent contains the variable.

Value

A list with four elements. `differentiable` is a character vector of admitted functions the installed derivative engine refuses. `escaped` is a character vector of symbols that the derivatives of the admitted set reach and that the table does not contain. `unsupported` is a character vector of admitted functions the backend does not provide, or `NULL` when the fourth criterion was not tested. `closed` is a logical scalar, `TRUE` when the first three are all empty.

Methodological notes

A check that can only pass is not a check. This one can fail in three independent ways, and each failure names the symbol responsible rather than reporting that something is wrong. Failure of the second criterion is the serious one: it means the centered form of some admitted function leaves the table on its first step, and there is nothing to evaluate.

Chapter 5 of Moore et al. (2009) treats interval extensions of expressions and the role of the derivative in the centered form.

Dependencies

Uses `stats::D`. Probes 'Rmpfr' from `Suggests` for the fourth criterion, and reports `NULL` for it when the backend is absent.

References

Moore, R. E., Kearfott, R. B., & Cloud, M. J. (2009). Introduction to interval analysis. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9780898717716

See Also

[ra_operator_table()].

Examples

```
rep <- ra_verify_operator_table(include_mpfr = FALSE)
rep$closed
rep$escaped
```

`ra_warn` *Raise a typed warning of the package*

Description

Builds and signals a warning condition whose class vector identifies the kind of degradation being reported, so that callers can catch or silence one kind without touching the others.

Usage

```
ra_warn(kind, message, call = sys.call(-1L), ...)
```

Arguments

kind	A character scalar naming the kind of degradation. It becomes the first element of the class vector, prefixed with "ra_".
message	A character scalar with the message shown to the user.
call	The call to report as the origin of the condition. Defaults to the caller of the function that raises.
...	Further named fields stored in the condition object.

Details

The class vector is c("ra_<kind>", "ra_warning", "warning", "condition"), mirroring the error side. A warning here is reserved for a computation that can continue on a declared fallback and says so; a violated contract raises through [ra_stop()] instead, and an undecided verdict does neither.

Value

The function signals a warning and then returns NULL invisibly, so execution continues at the caller.

Methodological notes

The warning path exists for exactly one situation: a safeguard has degraded the fast level and the rigorous level is available to escalate to. Continuing in silence would hide the degradation, and stopping would punish a caller whose answer can still be produced with its guarantee intact. The word travels with the result.

Dependencies

Base R only.

References

Wickham, H. (2019). Advanced R (2nd ed.). Chapman & Hall/CRC. https://doi.org/10.1201/9781351201315

See Also

[ra_stop()] for the error side of the condition system.

Examples

```
res <- withCallingHandlers(
  {
    ra_warn("example_kind", "an example warning")
    "the computation continued"
  },
  ra_example_kind = function(w) invokeRestart("muffleWarning")
)
res
```

`summary.ra_certificate` *Summarise a fixed point certificate*

Description

Reduces the certificate to the two verdicts and the numbers a reader needs to act on them.

Usage

```
## S3 method for class 'ra_certificate'
summary(object, ...)
```

Arguments

`object`	An object of class `ra_certificate`.
`...`	Ignored, present for consistency with the generic.

Details

The summary carries no verdict the certificate did not carry: it is the same two words with the radius and the error bound.

Value

An object of class `ra_summary_certificate`.

References

Granas, A., & Dugundji, J. (2003). Fixed point theory. Springer. https://doi.org/10.1007/978-0-387-21593-8

See Also

[ra_ball_certificate()].

Examples

```
summary(ra_ball_certificate(0.2, 0.5))
```

Index