Package 'flint'

July 22, 2025

Version 0.0.5

VersionNote sync configure.ac

Date 2025-05-27

Title Fast Library for Number Theory

Description An R interface to 'FLINT' <https://flintlib.org/>, a C library for number theory. 'FLINT' extends GNU 'MPFR' <https://www.mpfr.org/> and GNU 'MP' <https://gmplib.org/> with support for operations on standard rings (the integers, the integers modulo n, finite fields, the rational, p-adic, real, and complex numbers) as well as matrices and polynomials over rings. 'FLINT' implements midpoint-radius interval arithmetic, also known as ball arithmetic, in the real and complex numbers, enabling computation in arbitrary precision with rigorous propagation of rounding errors; see Johansson (2017) <doi:10.1109/TC.2017.2690633>. Finally, 'FLINT' provides ball arithmetic implementations of many special mathematical functions, with high coverage of reference works such as the NIST Digital Library of Mathematical Functions <https://dlmf.nist.gov/>. The R interface defines S4 classes, generic functions, and methods for representation and basic operations as well as plain R functions mirroring and vectorizing entry points in the C library.

License GPL (>= 2)

URL https://github.com/jaganmn/flint

BugReports https://github.com/jaganmn/flint/issues

Depends R (>= 4.3), methods

Imports stats

SystemRequirements flint (>= 3), mpfr (>= 3.1), gmp

SystemRequirementsNote purely informational as we use configure tests

NeedsCompilation yes

Author Mikael Jagan [aut, cre] (ORCID: <https://orcid.org/0000-0002-3542-2938>)

Maintainer Mikael Jagan <jaganmn@mcmaster.ca>

Repository CRAN

Date/Publication 2025-05-27 18:50:02 UTC

Contents

`flint-package` R *Package* **flint**

Description

An R interface to FLINT, a C library for number theory.

Usage

```
flintABI()
flintIdentical(object, reference)
flintLength(object, exact = TRUE)
flintPrec(prec = NULL)
flintRnd(rnd = NULL)
flintSize(object)
flintTriple(object)
flintVersion()
```

Arguments

`object`, `reference`
: objects inheriting from virtual class `flint`. Otherwise, the behaviour is undefined.

`exact`
: a logical indicating if the length should be represented exactly as an object of class `ulong`.

`prec`
: a new default value for the precision of inexact floating-point operations, if non-`NULL`. The value should be a positive integer indicating a number of bits.

`rnd`
: a new default value for the rounding mode of inexact floating-point operations, if non-`NULL`. The value should be a character string indicating a rounding mode for signed floating types. Valid characters are '`[Uu]`' (towards positive infinity), '`[Dd]`' (towards negative infinity), '`[Zz]`' (towards zero), '`[Aa]`' (away from zero), and '`[Nn]`' (to nearest, with precedence to even significands).

Details

To report a bug or request a feature, use `bug.report(package = "flint")`.

To render the change log, use `news(package = "flint")`.

To render the index, use `help(package = "flint")`

To render a list of help topics for S4 classes, use `help.search(package = "flint", keyword = "classes")`

To render a list of help topics for special mathematical functions, use `help.search(package = "flint", keyword = "math")`

Value

`flintABI` returns the size in bits of C type `long int`, either `32` or `64`. The value is determined when package **flint** is configured. It is checked at configure time and at load time that linked C libraries were configured for the same ABI.

`flintIdentical` tests whether its arguments inherit from the same nonvirtual subclass of `flint` and have identical length, elements, and names. If the elements are recursive structures, then they are compared recursively.

`flintLength` returns a representation of the length of `object`. If `exact = TRUE`, then the return value is an object of class `ulong` representing the length exactly. Otherwise, if the length is less than or equal to `.Machine[["integer.max"]]`, then the return value is a traditional integer vector representing the length exactly. Otherwise, the return value is a traditional double vector representing the length exactly if and only if $n \leq 2^d - 1$ or $2^{d+p} \leq n < 2^{d+p+1}$ and n is divisible by 2^{p+1}, where n is the length, d is `.Machine[["double.digits"]]`, and $p = 0, 1, \ldots$. Lengths not exactly representable in double precision are rounded to the next representable number in the direction of zero. Return values not representing the length exactly have an attribute `off` preserving the rounding error (an integer in $1, \ldots, 2^p$).

`flintPrec` returns the previous default precision.

`flintRnd` returns the previous default rounding mode.

`flintSize` returns an upper bound for the number of bytes used by `object`, as an object of class `object_size` (following function `object.size` in package **utils**). If no members of the recursive

structure share memory, then the upper bound is exact. Recursion starts at the address stored by the R object, not at the address of the object itself. A corollary is that `flintSize(object)` is zero for `object` of zero length. Another corollary is that the bytes counted by `flintSize` and the bytes counted by `object.size` are disjoint.

`flintTriple` returns a character vector of length 3 containing the class of `object`, the length of `object`, and the address stored by `object`.

`flintVersion` returns a named list of numeric versions with elements:

`package`	the R package version.
`flint.h`	the FLINT header version.
`libflint`	the FLINT library version.
`mpfr.h`	the GNU MPFR header version.
`libmpfr`	the GNU MPFR library version.
`gmp.h`	the GNU MP header version.
`libgmp`	the GNU MP library version.

Header versions are determined at compile time. Library versions are determined at compile time (static linking) or at load time (dynamic linking).

Author(s)

Mikael Jagan <jaganmn@mcmaster.ca>

References

FLINT Team (2025). FLINT: Fast Library for Number Theory. https://flintlib.org/

Examples

```
flintABI()

oprec <- flintPrec()
nprec <- 100L
stopifnot(identical(flintPrec(nprec), oprec),
          identical(flintPrec(), nprec),
          identical(flintPrec(oprec), nprec),
          identical(flintPrec(), oprec))

ornd <- flintRnd()
nrnd <- "Z"
stopifnot(identical(flintRnd(nrnd), ornd),
          identical(flintRnd(), nrnd),
          identical(flintRnd(ornd), nrnd),
          identical(flintRnd(), ornd))

flintVersion()
```

acb-class *Arbitrary Precision Floating-Point Complex Numbers with Error Bounds*

Description

Class acb extends virtual class flint. It represents vectors of complex numbers with error bounds on the real and imaginary parts. Elements are specified by two pairs of mixed format floating-point numbers: an arb real part and an arb imaginary part, each specified by an arf midpoint and a mag radius.

Usage

```
## The class generator function:
.acb(...)

## Mode 1: initialize with zeros
## .acb(length = 0L)
##
## Mode 2: initialize with vector
## .acb(length = length(x), x)
##
## Mode 3: initialize by parts
## .acb(length = max0(length(real), length(imag)), real, imag)
##
## where max0 <- function(m, n) if (min(m, n)) max(m, n) else 0L
```

Arguments

...	arguments passed to methods for initialize.

Value

The class generator function returns new("acb", ...).

Slots

.xData, names inherited from virtual class flint.

Methods

Due to constraints imposed by generic functions, methods typically do *not* provide a formal argument prec allowing for a precision to be indicated in the function call. Such methods use the current default precision set by flintPrec.

! signature(x = "acb"):
equivalent to (but faster than) x == 0.

`+ signature(e1 = "acb", e2 = "missing")`:
returns a copy of the argument.

`- signature(e1 = "acb", e2 = "missing")`:
returns the negation of the argument.

`Complex signature(z = "acb")`:
mathematical functions of one argument; see `S4groupGeneric`.

`Math signature(x = "acb")`:
mathematical functions of one argument; see `S4groupGeneric`. Member functions `floor`, `ceiling`, `trunc`, `cummin`, `cummax` are not implemented.

`Math2 signature(x = "acb")`:
decimal rounding according to a second argument `digits`; see `S4groupGeneric`. There are just two member member functions: `round`, `signif`.

`Ops signature(e1 = "ANY", e2 = "acb")`:
`signature(e1 = "acb", e2 = "ANY")`:
binary arithmetic, comparison, and logical operators; see `S4groupGeneric`. The "other" operand must be atomic or inherit from virtual class `flint`. The operands are recycled and promoted as necessary.

`Summary signature(x = "acb")`:
univariate summary statistics; see `S4groupGeneric`. The return value is a logical vector of length 1 (`any`, `all`) or an `acb` vector of length 1 or 2 (`sum`, `prod`). Member functions `min`, `max`, `range` are not implemented.

`anyNA signature(x = "acb")`:
returns `TRUE` if any element of `x` has real or imaginary part with midpoint `NaN`, `FALSE` otherwise.

`as.vector signature(x = "acb")`:
returns `as.vector(y, mode)`, where `y` is a complex vector containing the result of converting the midpoints of the real and imaginary parts of `x` to the range of double, rounding if the value is not exactly representable in double precision. The rounding mode is to the nearest representable number (with precedence to even significands in case of ties), unless a midpoint exceeds `.Machine[["double.xmax"]]` in absolute value, in which case `-Inf` or `Inf` is introduced with a warning. Coercion to types `"character"`, `"symbol"` (synonym `"name"`), `"pairlist"`, `"list"`, and `"expression"`, which are not "number-like", is handled specially.

`coerce signature(from = "ANY", to = "acb")`:
returns `.acb(x = from)`. An error is signaled if the class or type of `from` is not supported by the method for `initialize`.

`format signature(x = "acb")`:
returns a character vector suitable for printing, using string format `"(m +/- r)+(m +/- r)i"` and scientific format for each `m` and `r`. Optional arguments control the output; see `format-methods`.

`initialize signature(.Object = "acb", length = "numeric")`:
returns `.Object` after setting its `.xData` slot. The new slot value is an external pointer to an array of the corresponding C type, which is newly allocated and initialized entirely with zeros (zero midpoint, zero radius in the real and imaginary parts). `.Object` is not copied, so all references are affected.

`initialize signature(.Object = "acb", length = "numeric", x = "atomic|flint")`:
as above, except that the new slot value points to an array initialized with values from `x`.

`initialize signature(.Object = "acb", length = "numeric", real = "atomic|flint", imag = "atomic|flint")`:
as above, except that the new slot value points to an array whose real and imaginary parts are initialized separately with values from `real` and `imag`, which are coerced to class `arb` and recycled as necessary.

`is.finite signature(x = "acb")`:
returns a logical vector indicating which elements of `x` do not have real or imaginary part with midpoint `NaN`, `-Inf`, or `Inf` or radius `Inf`.

`is.infinite signature(x = "acb")`:
returns a logical vector indicating which elements of `x` have real or imaginary part with midpoint `-Inf` or `Inf` or radius `Inf`.

`is.na`, `is.nan signature(x = "acb")`:
returns a logical vector indicating which elements of `x` have real or imaginary part with midpoint `NaN`.

`is.unsorted signature(x = "acb")`:
signals an error indicating that `<=` is not a total order on the range of `arb`; see `xtfrm` below.

`log signature(x = "acb")`:
returns the logarithm of the argument. The natural logarithm is computed by default (when optional argument `base` is unset).

`mean signature(x = "acb")`:
returns the arithmetic mean.

`xtfrm signature(x = "acb")`:
signals an error indicating that `<=` is not a total order on the range of `arb`: `a <= b || b <= a` is is not `TRUE` for all finite `a` and `b` of class `arb`. Thus, direct sorting of `acb`, which is based on `arb`, is not supported. Users wanting to order the *midpoints* of the real and imaginary parts should operate on `Mid(Real(x))` and `Mid(Imag(x))`.

References

The FLINT documentation of the underlying C type: https://flintlib.org/doc/acb.html

Johansson, F. (2017). Arb: efficient arbitrary-precision midpoint-radius interval arithmetic. *IEEE Transactions on Computers*, *66*(8), 1281-1292. doi:10.1109/TC.2017.2690633

See Also

Virtual class `flint`. Generic functions `Real` and `Imag` and their replacement forms for getting and setting real and imaginary parts.

Examples

```
showClass("acb")
showMethods(classes = "acb")
```

acf-class *Arbitrary Precision Floating-Point Complex Numbers*

Description

Class acf extends virtual class flint. It represents vectors of arbitrary precision floating-point complex numbers. Elements have real and imaginary parts, each with arbitrary precision significand and exponent. The underlying C type can represent NaN, -Inf, and Inf real and imaginary parts.

Note that package **stats** exports a function acf, referring to autocovariance and autocorrelation functions of time series. It returns objects of *informal* S3 class acf, for which a small number of *informal* S3 methods are registered. The *formal* S4 class and methods documented here are entirely unrelated.

Usage

```
## The class generator function:
.acf(...)

## Mode 1: initialize with zeros
## .acf(length = 0L)
##
## Mode 2: initialize with vector
## .acf(length = length(x), x)
##
## Mode 3: initialize by parts
## .acf(length = max0(length(real), length(imag)), real, imag)
##
## where max0 <- function(m, n) if (min(m, n)) max(m, n) else 0L
```

Arguments

...	arguments passed to methods for initialize.

Value

The class generator function returns new("acf", ...).

Slots

.xData, names inherited from virtual class flint.

Methods

Due to constraints imposed by generic functions, methods typically do *not* provide a formal argument prec allowing for a precision to be indicated in the function call. Such methods use the current default precision set by flintPrec.

`! signature(x = "acf")`:
equivalent to (but faster than) `x == 0`.

`+ signature(e1 = "acf", e2 = "missing")`:
returns a copy of the argument.

`- signature(e1 = "acf", e2 = "missing")`:
returns the negation of the argument.

`Complex signature(z = "acf")`:
mathematical functions of one argument; see `S4groupGeneric`.

`Math signature(x = "acf")`:
mathematical functions of one argument; see `S4groupGeneric`. Member functions `floor`, `ceiling`, `trunc`, `cummin`, `cummax` are not implemented.

`Math2 signature(x = "acf")`:
decimal rounding according to a second argument `digits`; see `S4groupGeneric`. There are just two member member functions: `round`, `signif`.

`Ops signature(e1 = "ANY", e2 = "acf")`:
`signature(e1 = "acf", e2 = "ANY")`:
binary arithmetic, comparison, and logical operators; see `S4groupGeneric`. The "other" operand must be atomic or inherit from virtual class `flint`. The operands are recycled and promoted as necessary.

`Summary signature(x = "acf")`:
univariate summary statistics; see `S4groupGeneric`. The return value is a logical vector of length 1 (`any`, `all`) or an `acf` vector of length 1 or 2 (`sum`, `prod`). Member functions `min`, `max`, `range` are not implemented.

`anyNA signature(x = "acf")`:
returns `TRUE` if any element of `x` has real or imaginary part `NaN`, `FALSE` otherwise.

`as.vector signature(x = "acf")`:
returns `as.vector(y, mode)`, where `y` is a complex vector containing the result of converting the real and imaginary parts of `x` to the range of double, rounding if the value is not exactly representable in double precision. The rounding mode is to the nearest representable number (with precedence to even significands in case of ties), unless parts exceed `.Machine[["double.xmax"]]` in absolute value, in which case `-Inf` or `Inf` is introduced with a warning. Coercion to types `"character"`, `"symbol"` (synonym `"name"`), `"pairlist"`, `"list"`, and `"expression"`, which are not "number-like", is handled specially.

`coerce signature(from = "ANY", to = "acf")`:
returns `.acf(x = from)`. An error is signaled if the class or type of `from` is not supported by the method for `initialize`.

`format signature(x = "acf")`:
returns a character vector suitable for printing, using string format `"a+bi"` and scientific format for each `a` and `b`. Optional arguments control the output; see `format-methods`.

`initialize signature(.Object = "acf", length = "numeric")`:
returns `.Object` after setting its `.xData` slot. The new slot value is an external pointer to an array of the corresponding C type, which is newly allocated and initialized entirely with zeros. `.Object` is not copied, so all references are affected.

`initialize signature(.Object = "acf", length = "numeric", x = "atomic|flint")`:
as above, except that the new slot value points to an array initialized with values from `x`.

`initialize signature(.Object = "acf", length = "numeric", real = "atomic|flint", imag = "atomic|flint")`:
as above, except that the new slot value points to an array whose real and imaginary parts are initialized separately with values from `real` and `imag`, which are coerced to class `arf` and recycled as necessary.

`is.finite signature(x = "acf")`:
returns a logical vector indicating which elements of `x` do not have real or imaginary part `NaN`, `-Inf`, or `Inf`.

`is.infinite signature(x = "acf")`:
returns a logical vector indicating which elements of `x` have real or imaginary part `-Inf` or `Inf`.

`is.na`, `is.nan signature(x = "acf")`:
returns a logical vector indicating which elements of `x` have real or imaginary part `NaN`.

`is.unsorted signature(x = "acf")`:
returns a logical indicating if `x` is not sorted in nondecreasing order (increasing order if optional argument `strictly` is set to `TRUE`) by real part then by imaginary part.

`mean signature(x = "acf")`:
returns the arithmetic mean.

`xtfrm signature(x = "acf")`:
returns a numeric vector that sorts in the same order as `x`. The permutation `order(xtfrm(x), ...)` orders `x` first by its real part then by its imaginary part, with the caveat that all `a+NaNi` and `NaN+bi` have equal precedence (for compatibility with **base**).

See Also

Virtual class `flint`. Generic functions `Real` and `Imag` and their replacement forms for getting and setting real and imaginary parts.

Examples

```
showClass("acf")
showMethods(classes = "acf")
```

---

arb-class *Arbitrary Precision Floating-Point Real Numbers with Error Bounds*

Description

Class `arb` extends virtual class `flint`. It represents vectors of arbitrary precision floating-point real numbers with error bounds. Elements are specified by a pair of mixed format floating-point numbers: an `arf` midpoint and a `mag` radius.

Arithmetic on `arb` vectors is midpoint-radius interval arithmetic, also known as ball arithmetic, enabling computation with rigorous propagation of errors. Logic and comparison involving `arb` vectors are defined as follows: unary `op(x)` is true if and only if `op` is true for all elements of the interval `x`, and binary `op(x, y)` is true if and only if `op` is true for all elements of the Cartesian product of the intervals `x` and `y`. A corollary is that the operator `<=` does not define a *total order* on the range of `arb` (that is, the set of intervals $[m - r, m + r]$), and a consequence is that methods for generic functions that necessitate a total order tend to signal an error.

Usage

```
## The class generator function:
.arb(...)

## Mode 1: initialize with zeros
## .arb(length = 0L)
##
## Mode 2: initialize with vector
## .arb(length = length(x), x)
##
## Mode 3: initialize by parts
## .arb(length = max0(length(mid), length(rad)), mid, rad)
##
## where max0 <- function(m, n) if (min(m, n)) max(m, n) else 0L
```

Arguments

`...`	arguments passed to methods for `initialize`.

Value

The class generator function returns `new("arb", ...)`.

Slots

`.xData`, `names` inherited from virtual class `flint`.

Methods

Due to constraints imposed by generic functions, methods typically do *not* provide a formal argument `prec` allowing for a precision to be indicated in the function call. Such methods use the current default precision set by `flintPrec`.

`! signature(x = "arb")`:
: equivalent to (but faster than) `x == 0`.

`+ signature(e1 = "arb", e2 = "missing")`:
: returns a copy of the argument.

`- signature(e1 = "arb", e2 = "missing")`:
: returns the negation of the argument.

`Complex signature(z = "arb")`:
: mathematical functions of one argument; see `S4groupGeneric`.

`Math signature(x = "arb")`:
: mathematical functions of one argument; see `S4groupGeneric`.

`Math2 signature(x = "arb")`:
: decimal rounding according to a second argument `digits`; see `S4groupGeneric`. There are just two member member functions: `round`, `signif`.

`Ops signature(e1 = "ANY", e2 = "arb")`:
`signature(e1 = "arb", e2 = "ANY")`:
binary arithmetic, comparison, and logical operators; see `S4groupGeneric`. The "other" operand must be atomic or inherit from virtual class `flint`. The operands are recycled and promoted as necessary.

`Summary signature(x = "arb")`:
univariate summary statistics; see `S4groupGeneric`. The return value is a logical vector of length 1 (`any`, `all`) or an `arb` vector of length 1 or 2 (`sum`, `prod`, `min`, `max`, `range`).

`anyNA signature(x = "arb")`:
returns `TRUE` if any element of `x` has midpoint `NaN`, `FALSE` otherwise.

`as.vector signature(x = "arb")`:
returns `as.vector(y, mode)`, where `y` is a double vector containing the result of converting the midpoints of `x` to the range of double, rounding if the value is not exactly representable in double precision. The rounding mode is to the nearest representable number (with precedence to even significands in case of ties), unless a midpoint exceeds `.Machine[["double.xmax"]]` in absolute value, in which case `-Inf` or `Inf` is introduced with a warning. Coercion to types `"character"`, `"symbol"` (synonym `"name"`), `"pairlist"`, `"list"`, and `"expression"`, which are not "number-like", is handled specially.

`coerce signature(from = "ANY", to = "arb")`:
returns `.arb(x = from)`. An error is signaled if the class or type of `from` is not supported by the method for `initialize`.

`format signature(x = "arb")`:
returns a character vector suitable for printing, using string format `"(m +/- r)"` and scientific format for `m` and `r`. Optional arguments control the output; see `format-methods`.

`initialize signature(.Object = "arb", length = "numeric")`:
returns `.Object` after setting its `.xData` slot. The new slot value is an external pointer to an array of the corresponding C type, which is newly allocated and initialized entirely with zeros (zero midpoint, zero radius). `.Object` is not copied, so all references are affected.

`initialize signature(.Object = "arb", length = "numeric", x = "atomic|flint")`:
as above, except that the new slot value points to an array initialized by the elements of `x`.

`initialize signature(.Object = "arb", length = "numeric", mid = "atomic|flint", rad = "atomic|flint")`:
as above, except that the new slot value points to an array whose midpoints and radii are initialized separately with values from `mid` and `rad`, which are coerced to classes `arf` and `mag` and recycled as necessary.

`is.finite signature(x = "arb")`:
returns a logical vector indicating which elements of `x` do not have midpoint `NaN`, `-Inf`, or `Inf` or radius `Inf`.

`is.infinite signature(x = "arb")`:
returns a logical vector indicating which elements of `x` have midpoint `-Inf` or `Inf` or radius `Inf`.

`is.na`, `is.nan signature(x = "arb")`:
returns a logical vector indicating which elements of `x` have midpoint `NaN`.

`is.unsorted signature(x = "arb")`:
signals an error indicating that <= is not a total order on the range of `arb`; see `xtfrm` below.

`log signature(x = "arb")`:
: returns the logarithm of the argument. The natural logarithm is computed by default (when optional argument `base` is unset).

`mean signature(x = "arb")`:
: returns the arithmetic mean.

`xtfrm signature(x = "arb")`:
: signals an error indicating that `<=` is not a total order on the range of `arb`: `a <= b || b <= a` is is not `TRUE` for all finite `a` and `b` of class `arb`. Thus, direct sorting of `arb` is not supported. Users wanting to order the *midpoints* should operate on `Mid(x)`.

References

The FLINT documentation of the underlying C type: https://flintlib.org/doc/arb.html

Johansson, F. (2017). Arb: efficient arbitrary-precision midpoint-radius interval arithmetic. *IEEE Transactions on Computers*, *66*(8), 1281-1292. doi:10.1109/TC.2017.2690633

See Also

Virtual class `flint`. Generic functions `Mid` and `Rad` and their replacement forms for getting and setting midpoints and radii.

Examples

```
showClass("arb")
showMethods(classes = "arb")
```

---

arb_dirichlet_zeta *Zeta and Related Functions*

Description

Compute the Riemann zeta function, the Hurwitz zeta function, or Lerch's transcendent. Lerch's transcendent $\Phi(z, s, a)$ is defined by

$$\sum_{k=0}^{\infty} \frac{z^k}{(k+a)^s}$$

for $|z| < 1$ and by analytic continuation elsewhere in the z-plane. The Riemann and Hurwitz zeta functions are the special cases $\zeta(s) = \Phi(1, s, 1)$ and $\zeta(s, a) = \Phi(1, s, a)$, respectively. See the references for restrictions on s and a.

Usage

```
arb_dirichlet_zeta(s, prec = flintPrec())
acb_dirichlet_zeta(s, prec = flintPrec())

arb_dirichlet_hurwitz(s, a = 1, prec = flintPrec())
acb_dirichlet_hurwitz(s, a = 1, prec = flintPrec())
```

```
## arb_dirichlet_lerch_phi(z = 1, s, a = 1, prec = flintPrec())
   acb_dirichlet_lerch_phi(z = 1, s, a = 1, prec = flintPrec())
```

Arguments

z, s, a	numeric, complex, arb, or acb vectors.
prec	a numeric or slong vector indicating the desired precision as a number of bits.

Value

An arb or acb vector storing function values with error bounds. Its length is the maximum of the lengths of the arguments or zero (zero if any argument has length zero). The arguments are recycled as necessary.

References

The FLINT documentation of the underlying C functions: https://flintlib.org/doc/acb_dirichlet.html

NIST Digital Library of Mathematical Functions: https://dlmf.nist.gov/25

See Also

Classes arb and acb.

Examples

```
dzet <- acb_dirichlet_zeta
dhur <- acb_dirichlet_hurwitz
dler <- acb_dirichlet_lerch_phi

## Somewhat famous particular values :
debugging <- tolower(Sys.getenv("R_FLINT_CHECK_EXTRA")) == "true"
     s <- .acb(x = c(   -1,    0,      2,       4))
zeta.s <- .acb(x = c(-1/12, -1/2, pi^2/6, pi^4/90))
stopifnot(all.equal(dzet(   s   ), zeta.s),
          all.equal(dhur(   s, 1), zeta.s),
          !debugging ||
          {
          print(cbind(as.complex(dler(1, s, 1)), as.complex(zeta.s)))
          all.equal(dler(1, s, 1), zeta.s) # FLINT bug, report this
          })

set.seed(0xabcdL)
r <- 10L
eps <- 0x1p-4
a    <- flint:::complex.runif(r, modulus = c(    0, 1/eps))
z.l1 <- flint:::complex.runif(r, modulus = c(    0, 1-eps))
z.g1 <- flint:::complex.runif(r, modulus = c(1+eps, 1/eps))
z <- .acb(x = c(z.l1, z.g1))
```

```
## A relation with the hypergeometric function from
##   http://dlmf.nist.gov/25.14.E3_3 :
h2f1 <- acb_hypgeom_2f1
stopifnot(all.equal(dler(z.l1, 1, a), h2f1(a, 1, a + 1, z.l1)/a))

## TODO: test values also for z[Mod(z) > 1] ...
```

---

arb_hypgeom_2f1 *Hypergeometric Functions*

Description

Computes the principal branch of the hypergeometric function ${}_2F_1(a, b, c, z)$, defined by

$$\sum_{k=0}^{\infty} \frac{(a)_k (b)_k}{(c)_k} \frac{z^k}{k!}$$

for $|z| < 1$ and by analytic continuation elsewhere in the z-plane, or the principal branch of the *regularized* hypergeometric function ${}_2F_1(a, b, c, z)/\Gamma(c)$.

Usage

```
arb_hypgeom_2f1(a, b, c, x, flags = 0L, prec = flintPrec())
acb_hypgeom_2f1(a, b, c, z, flags = 0L, prec = flintPrec())
```

Arguments

a, b, c, x, z	numeric, complex, arb, or acb vectors.
flags	an integer vector. The lowest bit of the integer element(s) indicates whether to regularize. Later bits indicate special cases for which an alternate algorithm may be used. Non-experts should use flags = 0L or 1L, leaving the later bits unset.
prec	a numeric or slong vector indicating the desired precision as a number of bits.

Value

An arb or acb vector storing function values with error bounds. Its length is the maximum of the lengths of the arguments or zero (zero if any argument has length zero). The arguments are recycled as necessary.

References

The FLINT documentation of the underlying C functions: https://flintlib.org/doc/arb_hypgeom.html, https://flintlib.org/doc/acb_hypgeom.html

NIST Digital Library of Mathematical Functions: https://dlmf.nist.gov/15

See Also

Classes arb and acb.

Examples

```
h2f1 <- acb_hypgeom_2f1

set.seed(0xbcdeL)
r <- 10L
eps <- 0x1p-4
z.l1 <- flint:::complex.runif(r, modulus = c(    0, 1-eps))
z.g1 <- flint:::complex.runif(r, modulus = c(1+eps, 1/eps))
z <- .acb(x = c(z.l1, z.g1))

## Elementary special cases from http://dlmf.nist.gov/15.4 :
stopifnot(all.equal(h2f1(1.0, 1.0, 2.0,  z  ),
                    -log(1 - z)/z),
          all.equal(h2f1(0.5, 1.0, 1.5,  z^2),
                    0.5 * (log(1 + z) - log(1 - z))/z),
          all.equal(h2f1(0.5, 1.0, 1.5, -z^2),
                    atan(z)/z))
## [ see more in ../tests/hypgeom.R ]
```

---

arb_hypgeom_gamma *Gamma and Related Functions*

Description

Compute the gamma function, the reciprocal gamma function, the logarithm of the absolute value of the gamma function, the polygamma function, or the beta function. The gamma function $\Gamma(z)$ is defined by

$$\int_0^\infty t^{z-1} e^{-t} \mathrm{d}t$$

for $\Re(z) > 0$ and by analytic continuation elsewhere in the z-plane, excluding poles at $z = 0, -1, \ldots$. The beta function $B(a, b)$ is defined by

$$\int_0^1 t^{a-1} (1-t)^{b-1} \mathrm{d}t$$

for $\Re(a), \Re(b) > 0$ and by analytic continuation to all other (a, b).

Usage

```
arb_hypgeom_gamma(x, prec = flintPrec())
acb_hypgeom_gamma(z, prec = flintPrec())

arb_hypgeom_rgamma(x, prec = flintPrec())
acb_hypgeom_rgamma(z, prec = flintPrec())

arb_hypgeom_lgamma(x, prec = flintPrec())
acb_hypgeom_lgamma(z, prec = flintPrec())
```

```
## arb_hypgeom_polygamma(s = 0, z, prec = flintPrec())
   acb_hypgeom_polygamma(s = 0, z, prec = flintPrec())

arb_hypgeom_beta(a, b, prec = flintPrec())
acb_hypgeom_beta(a, b, prec = flintPrec())
```

Arguments

x, z, s, a, b	numeric, complex, arb, or acb vectors.
prec	a numeric or slong vector indicating the desired precision as a number of bits.

Details

acb_hypgeom_polygamma(s, z) evaluates the polygamma function of order s at z. The order s can be any complex number. For nonnegative integers m, s = m corresponds to the derivative of order m of the digamma function $\psi(z) = \Gamma'(z)/\Gamma(z)$. Use acb_hypgeom_polygamma(0, z) to evaluate the digamma function at z.

Value

An arb or acb vector storing function values with error bounds. Its length is the maximum of the lengths of the arguments or zero (zero if any argument has length zero). The arguments are recycled as necessary.

References

The FLINT documentation of the underlying C functions: https://flintlib.org/doc/arb_hypgeom.html, https://flintlib.org/doc/acb_hypgeom.html

NIST Digital Library of Mathematical Functions: https://dlmf.nist.gov/5

See Also

Classes arb and acb; arb_hypgeom_gamma_lower and arb_hypgeom_beta_lower for the "incomplete" gamma and beta functions.

Examples

```
## TODO
```

---

arb_hypgeom_gamma_lower

Incomplete Gamma and Related Functions

Description

Compute the principal branch of the (optionally, regularized) incomplete gamma and beta functions. The lower incomplete gamma function $\gamma(s, z)$ is defined by

$$\int_0^z t^{s-1} e^{-t} \mathrm{d}t$$

for $\Re(s) > 0$ and by analytic continuation elsewhere in the s-plane, excluding poles at $s = 0, -1, \ldots$. The upper incomplete gamma function $\Gamma(s, z)$ is defined by

$$\int_z^\infty t^{s-1} e^{-t} \mathrm{d}t$$

for $\Re(s) > 0$ and by analytic continuation elsewhere in the s-plane except at $z = 0$. The incomplete beta function $B(a, b, z)$ is defined by

$$\int_0^z t^{a-1} (1-t)^{b-1} \mathrm{d}t$$

for $\Re(a), \Re(b) > 0$ and by analytic continuation to all other (a, b). It coincides with the beta function at $z = 1$. The regularized functions are $\gamma(s, z)/\Gamma(s)$, $\Gamma(s, z)/\Gamma(s)$, and $B(a, b, z)/B(a, b)$.

Usage

```
arb_hypgeom_gamma_lower(s, x, flags = 0L, prec = flintPrec())
acb_hypgeom_gamma_lower(s, z, flags = 0L, prec = flintPrec())

arb_hypgeom_gamma_upper(s, x, flags = 0L, prec = flintPrec())
acb_hypgeom_gamma_upper(s, z, flags = 0L, prec = flintPrec())

arb_hypgeom_beta_lower(a, b, x, flags = 0L, prec = flintPrec())
acb_hypgeom_beta_lower(a, b, z, flags = 0L, prec = flintPrec())
```

Arguments

x, z, s, a, b	numeric, complex, arb, or acb vectors.
flags	an integer vector with elements 0, 1, or 2 indicating unregularized, regularized, or "alternately" regularized; see the FLINT documentation.
prec	a numeric or slong vector indicating the desired precision as a number of bits.

Value

An arb or acb vector storing function values with error bounds. Its length is the maximum of the lengths of the arguments or zero (zero if any argument has length zero). The arguments are recycled as necessary.

References

The FLINT documentation of the underlying C functions: https://flintlib.org/doc/arb_hypgeom.html, https://flintlib.org/doc/acb_hypgeom.html

NIST Digital Library of Mathematical Functions: https://dlmf.nist.gov/8

See Also

Classes arb and acb; arb_hypgeom_gamma and arb_hypgeom_beta for the "complete" gamma and beta functions.

Examples

```
hg  <- acb_hypgeom_gamma
hgl <- acb_hypgeom_gamma_lower
hgu <- acb_hypgeom_gamma_upper

hb  <- acb_hypgeom_beta
hbl <- acb_hypgeom_beta_lower

set.seed(0xcdefL)
r <- 10L
eps <- 0x1p-4
a <- flint:::complex.runif(r, modulus = c(  0, 1/eps))
b <- flint:::complex.runif(r, modulus = c(  0, 1/eps))
z <- flint:::complex.runif(r, modulus = c(eps, 1/eps))

## Some trivial identities
stopifnot(# http://dlmf.nist.gov/8.2.E3
          all.equal(hgl(a, z) + hgu(a, z), hg(a), tolerance = 1e-5),
          # https://dlmf.nist.gov/8.4.E5
          all.equal(hgu(1, z), exp(-z), check.class = FALSE))

## Regularization
stopifnot(all.equal(hgl(a,    z, flags = 1L), hgl(a,    z)/hg(a   )),
          all.equal(hgu(a,    z, flags = 1L), hgu(a,    z)/hg(a   )),
          all.equal(hbl(a, b, z, flags = 1L), hbl(a, b, z)/hb(a, b)))

## A relation with the hypergeometric function from
##   https://dlmf.nist.gov/8.17.E7 :
h2f1 <- acb_hypgeom_2f1
stopifnot(all.equal(hbl(a, b, z), z^a * h2f1(a, 1 - b, a + 1, z)/a))
```

---

arb_lambertw *Lambert W function*

Description

Computes any branch W_k of the multiple-valued Lambert W function. $W(z)$ is the set of solutions w of the equation $we^w = z$.

Usage

```
arb_lambertw(x,          flags = 0L, prec = flintPrec())
acb_lambertw(z, k = 0L, flags = 0L, prec = flintPrec())
```

Arguments

x, z	numeric, complex, arb, or acb vectors.
k	an integer or fmpz vector listing indices of branches of the function. 0 indicates the principal branch.
flags	for arb_lambertw: an integer vector indicating which of the index 0 and index -1 branches is computed (0 means index 0, 1 means index -1). for acb_lambertw: an integer vector indicating how branch cuts are defined. Nonzero values are nonstandard; see the first reference.
prec	a numeric or slong vector indicating the desired precision as a number of bits.

Value

An arb or acb vector storing function values with error bounds. Its length is the maximum of the lengths of the arguments or zero (zero if any argument has length zero). The arguments are recycled as necessary.

References

The FLINT documentation of the underlying C functions: https://flintlib.org/doc/arb.html, https://flintlib.org/doc/acb.html

NIST Digital Library of Mathematical Functions: https://dlmf.nist.gov/4.13

See Also

Classes arb and acb.

Examples

```
## TODO
```

---

arf-class *Arbitrary Precision Floating-Point Real Numbers*

Description

Class arf extends virtual class flint. It represents vectors of arbitrary precision floating-point real numbers. Elements have arbitrary precision significand and exponent. The underlying C type can represent NaN, -Inf, and Inf.

Usage

```
## The class generator function:
.arf(...)

## Mode 1: initialize with zeros
## .arf(length = 0L)
##
## Mode 2: initialize with vector
## .arf(length = length(x), x)
```

Arguments

`...`	arguments passed to methods for `initialize`.

Value

The class generator function returns `new("arf", ...)`.

Slots

`.xData`, `names` inherited from virtual class `flint`.

Methods

Due to constraints imposed by generic functions, methods typically do *not* provide a formal argument `prec` allowing for a precision to be indicated in the function call. Such methods use the current default precision set by `flintPrec`.

`! signature(x = "arf")`:
: equivalent to (but faster than) `x == 0`.

`+ signature(e1 = "arf", e2 = "missing")`:
: returns a copy of the argument.

`- signature(e1 = "arf", e2 = "missing")`:
: returns the negation of the argument.

`Complex signature(z = "arf")`:
: mathematical functions of one argument; see `S4groupGeneric`.

`Math signature(x = "arf")`:
: mathematical functions of one argument; see `S4groupGeneric`. Notably, the logarithmic, exponential, (inverse) trigonometric, (inverse) hyperbolic, and gamma-related member functions are not yet implemented. Users wanting those can (for now) operate on `as(x, "arb")`.

`Math2 signature(x = "arf")`:
: decimal rounding according to a second argument `digits`; see `S4groupGeneric`. There are just two member member functions: `round`, `signif`.

`Ops signature(e1 = "ANY", e2 = "arf")`:
`signature(e1 = "arf", e2 = "ANY")`:
: binary arithmetic, comparison, and logical operators; see `S4groupGeneric`. The "other" operand must be atomic or inherit from virtual class `flint`. The operands are recycled and promoted as necessary.

`Summary signature(x = "arf")`:
: univariate summary statistics; see `S4groupGeneric`. The return value is a logical vector of length 1 (`any`, `all`) or an `arf` vector of length 1 or 2 (`sum`, `prod`, `min`, `max`, `range`).

`anyNA signature(x = "arf")`:
: returns `TRUE` if any element of `x` is `NaN`, `FALSE` otherwise.

`as.vector signature(x = "arf")`:
: returns `as.vector(y, mode)`, where `y` is a double vector containing the result of converting each element of `x` to the range of double, rounding if the value is not exactly representable in double precision. The rounding mode is to the nearest representable number (with precedence to even significands in case of ties), unless the element exceeds `.Machine[["double.xmax"]]` in absolute value, in which case `-Inf` or `Inf` is introduced with a warning. Coercion to types `"character"`, `"symbol"` (synonym `"name"`), `"pairlist"`, `"list"`, and `"expression"`, which are not "number-like", is handled specially.

`coerce signature(from = "ANY", to = "arf")`:
: returns `.arf(x = from)`. An error is signaled if the class or type of `from` is not supported by the method for `initialize`.

`format signature(x = "arf")`:
: returns a character vector suitable for printing, using scientific format. Optional arguments control the output; see `format-methods`.

`initialize signature(.Object = "arf", length = "numeric")`:
: returns `.Object` after setting its `.xData` slot. The new slot value is an external pointer to an array of the corresponding C type, which is newly allocated and initialized entirely with zeros. `.Object` is not copied, so all references are affected.

`initialize signature(.Object = "arf", length = "numeric", x = "atomic|flint")`:
: as above, except that the new slot value points to an array initialized with values from `x`.

`is.finite signature(x = "arf")`:
: returns a logical vector indicating which elements of `x` are not `NaN`, `-Inf`, or `Inf`.

`is.infinite signature(x = "arf")`:
: returns a logical vector indicating which elements of `x` are `-Inf` or `Inf`.

`is.na, is.nan signature(x = "arf")`:
: returns a logical vector indicating which elements of `x` are `NaN`.

`is.unsorted signature(x = "arf")`:
: returns a logical indicating if `x` is not sorted in nondecreasing order (increasing order if optional argument `strictly` is set to `TRUE`).

`mean signature(x = "arf")`:
: returns the arithmetic mean.

References

The FLINT documentation of the underlying C type: https://flintlib.org/doc/arf.html

Johansson, F. (2017). Arb: efficient arbitrary-precision midpoint-radius interval arithmetic. *IEEE Transactions on Computers*, *66*(8), 1281-1292. doi:10.1109/TC.2017.2690633

See Also

Virtual class `flint`.

Examples

```
showClass("arf")
showMethods(classes = "arf")
```

`c.flint` *Concatenate Vectors*

Description

Primitive function `c` is internally generic but dispatches only on its first argument. A corollary is that `c(x, ...)` does *not* dispatch the S4 method with signature `x="flint"` if `x` is not a `flint` vector, even if a `flint` vector appears later in the call as an element of `...`. An S3 method `c.flint` is registered *and exported* to support concatenation with `c.flint(x, ...)` as an alternative to `c(as(x, "flint"), ...)`.

Usage

```
## S3 method for class 'flint'
c(..., recursive = FALSE, use.names = TRUE)
```

Arguments

`...`	objects inheriting from virtual class `flint`, atomic (except character) vectors, and `NULL`.
`recursive`	a logical indicating if pairlists, lists, and expressions should be handled recursively. If `TRUE`, then the function behaves as if such arguments were replaced by their terminal nodes.
`use.names`	a logical indicating if names should be preserved.

Value

If none of the arguments is a `flint` vector, then the internal default method for `c` is dispatched.

If at least one argument is a `flint` vector, then the return value is a `flint` vector, unless `recursive = FALSE` and at least one argument is a symbol, pairlist, list, or expression, in which case the return value is a list or expression. The length is the sum of the lengths of the arguments.

If the return value is a `flint` vector, then its class is the most specific subclass of `flint` whose range contains the ranges of the classes of the arguments.

See Also

Virtual class `flint`.

Examples

```
x <- .slong(x = 2:5)
c(x, 6L)
c(1L, x) # bad
c.flint(x, 6L)
c.flint(1L, x)
```

Constants *Mathematical Constants Represented to Arbitrary Precision*

Description

Compute standard mathematical constants to arbitrary precision.

Usage

```
arb_const_pi(prec = flintPrec())
arb_const_log2(prec = flintPrec())
arb_const_log10(prec = flintPrec())
arb_const_e(prec = flintPrec())
```

Arguments

prec	a numeric or slong vector indicating the desired precision as a number of bits.

Value

An arb vector storing function values with error bounds. Its length is the length of prec, typically 1.

References

The FLINT documentation of the underlying C functions: https://flintlib.org/doc/arb.html

See Also

Class arb.

Examples

```
prec <- cumprod(rep(c(1, 2), c(1L, 15L)))
arb_const_pi(prec)
```

flint-class *Class of FLINT-Type Vectors*

Description

Class flint is a virtual class representing vectors of any FLINT C type. The C type is determined by the class attribute and interfaced exactly using R's external pointer type.

Slots

.xData an external pointer. The protected field is an integer vector of length 1 or 2 storing the object length whose size is 32 or 64 bits depending on the ABI; see flintABI. The pointer field contains the address of a block of allocated memory of size greater than or equal to the object length times the size of the FLINT C type. It is a null pointer if and only if the object length is zero.

Methods for initialize set a finalizer on .xData (see reg.finalizer) to ensure that allocated memory is freed before .xData is itself freed by the garbage collector.

names a character vector of length 0, indicating that the object is unnamed, or of length equal to the object length. A corollary is that objects whose length exceeds the maximum length of a character vector cannot have names.

Methods

[signature(x = "ANY", i = "flint", j = "missing", drop = "missing"):
signature(x = "flint", i = "ANY", j = "missing", drop = "missing"):
signature(x = "flint", i = "flint", j = "missing", drop = "missing"):
return a traditional vector or a flint vector containing the elements of x indexed by i (the "subscript"). The subscript can be missing, NULL, logical, integer, double, character, ulong, slong, fmpz, or fmpq. Methods for signatures with x = "flint" signal an error for NA and out of bounds subscripts, as the C types interfaced by flint vectors have no representation for missing values. Note that [does not perform S4 dispatch if its first positional argument is not an S4 object. If it is known that i is a flint vector and not known whether x is a flint vector, then consider programming defensively by calling [as `[`(i = i, x = x) rather than as x[i].

[<- signature(x = "ANY", i = "ANY", j = "missing", value = "flint"):
signature(x = "ANY", i = "flint", j = "missing", value = "ANY"):
signature(x = "ANY", i = "flint", j = "missing", value = "flint"):
signature(x = "flint", i = "ANY", j = "missing", value = "ANY"):
signature(x = "flint", i = "ANY", j = "missing", value = "flint"):
signature(x = "flint", i = "flint", j = "missing", value = "ANY"):
signature(x = "flint", i = "flint", j = "missing", value = "flint"):
return the traditional vector or flint vector obtained by replacing the elements of x indexed by i (the "subscript") with elements of value, which are recycled as necessary. The subscript can be missing, NULL, logical, integer, double, character, ulong, slong, fmpz, or fmpq. The class of the return value is determined following strict rules from the classes of x and value,

which are promoted to the value class as necessary. If the value class is a subclass of `flint`, then an error is signaled for `NA` and out of bounds subscripts, as the C types interfaced by `flint` vectors have no representation for missing values. Note that `[<-` does not perform S4 dispatch if its first positional argument is not an S4 object. If it is known that `i` is a `flint` vector and not known whether `x` is a `flint` vector, then consider calling `[<-` as `` `[`(i = i, x = x) <- value `` rather than as `x[i] <- value`. If it known that `value` is a `flint` vector and not known whether `x` or `i` is a `flint` vector, then consider doing something like `x <- `[<-`(value = value, x = x, i = i)`.

`[[ signature(x = "ANY", i = "flint", j = "missing")`:
`signature(x = "flint", i = "ANY", j = "missing")`:
`signature(x = "flint", i = "flint", j = "missing")`:
similar to `[`, with differences as documented in `Extract`.

`[[<- signature(x = "ANY", i = "ANY", j = "missing", value = "flint")`:
`signature(x = "ANY", i = "flint", j = "missing", value = "ANY")`:
`signature(x = "ANY", i = "flint", j = "missing", value = "flint")`:
`signature(x = "flint", i = "ANY", j = "missing", value = "ANY")`:
`signature(x = "flint", i = "ANY", j = "missing", value = "flint")`:
`signature(x = "flint", i = "flint", j = "missing", value = "ANY")`:
`signature(x = "flint", i = "flint", j = "missing", value = "flint")`:
similar to `[<-`, with differences as documented in `Extract`.

`all.equal signature(x = "ANY", y = "flint")`:
`signature(x = "flint", y = "ANY")`:
`signature(x = "flint", y = "flint")`:
returns either `TRUE`, indicating that there is no meaningful difference between `x` and `y`, or a character vector describing differences. The implementation (including optional arguments) is adapted from `all.equal.numeric`, hence see *its* documentation. Notably, comparison of objects inheriting from different subclasses of virtual class `flint` and comparison with objects (typically atomic vectors) coercible to virtual class `flint` are supported with `check.class = FALSE`. See `flintIdentical` for much stricter comparison of objects inheriting from `flint`.

`anyDuplicated signature(x = "flint")`:
returns `anyDuplicated(mtfrm(x), ...)`.

`as.raw`, **`as.logical`**, **`as.integer`**, **`as.numeric`**, **`as.complex signature(x = "flint")`**:
returns the value of `as.vector(x, mode = *)`. Methods for `as.vector` must be defined for subclasses of `flint`. Note that `as.double` dispatches internally the method for `as.numeric`, so there is no method for `as.double`; see also `as.numeric`, section 'S4 Methods'.

`as.matrix`, **`as.array`**, **`as.Date`**, **`as.POSIXct`**, **`as.POSIXlt signature(x = "flint")`**:
coerces the argument with `as.vector` and dispatches.

`as.data.frame signature(x = "flint")`:
this method is a copy of `as.data.frame.vector`. It enables the construction of data frames containing `flint` vectors using `as.data.frame` and functions that call it such as `data.frame` and `cbind.data.frame`.

`c signature(x = "flint")`:
returns `c.flint(x, ...)`, the concatenation of the arguments. Function `c.flint` is exported to work around the fact that `c(x, ...)` dispatches only on `x`.

`coerce signature(from = "ANY", to = "flint")`:
coerces atomic (except character) vectors `from` to the most specific subclass of `flint` whose range contains the range of `typeof(from)`.

`cut signature(x = "flint")`:
: returns `findInterval(x=x, vec=breaks, left.open=right, rightmost.closed=include.lowest)`, hence see below. The behaviour is consistent with the default method for `cut` with argument `labels` set to `FALSE`, provided that `breaks` is sorted and no element of `x` is out of bounds.

`duplicated signature(x = "flint")`:
: returns `duplicated(mtfrm(x), ...)`.

`findInterval` returns a `ulong` vector of length equal to the length of `x`, following the documented behaviour of the **base** function, hence see `findInterval`. A caveat is that an error is signaled if `x` contains `NaN`, because `ulong` has no representation for R's missing value `NA_integer_`.

`is.na<- signature(x = "flint")`:
: returns the value of `x` after `x[value] <-` *na*, where *na* is an `NA` of integer, double, or complex type, depending on the class of `x`.

`length signature(x = "flint")`:
: returns `flintLength(x, exact = FALSE)`.

`length<- signature(x = "flint")`:
: returns a `flint` vector of length given by the second argument `value`. The first `min(length(x), value)` elements are copied from `x` and the remaining elements are initialized to zero.

`match signature(x = "ANY", table = "flint")`:
`signature(x = "flint", table = "ANY")`:
`signature(x = "flint", table = "flint")`:
: returns an integer vector matching `x` to `table` after coercing to a common class then "match transforming" with `mtfrm`. The behaviour is parallel to that of the default method, hence see `match`.

`mtfrm signature(x = "flint")`:
: returns `format(x, base = 62L, digits = 0L)`, a character vector representing the elements of `x` exactly in base 62 (chosen over smaller bases to reduce the number of characters in the output); see also `format-methods`.

`names signature(x = "flint")`:
: returns the value of the `names` slot or `NULL`, `NULL` if the `names` slot has zero length.

`names<- signature(x = "flint", value = "NULL")`:
: returns `x` with `names` slot set to a character vector of length zero.

`names<- signature(x = "flint", value = "character")`:
: returns `x` with `names` slot set to `value`. Attributes of `value` are stripped. `NA_character_` are appended to `value` if its length is less than the length of `x`. An error is signaled if its length is greater.

`print signature(x = "flint")`:
: prints `format(x)` without quotes and returns `x` invisibly. The output has a header listing the class and length of `x` and the address stored by its `.xData` slot. If the output might be differenced by `Rdiff`, then one can set optional argument `Rdiff` to `TRUE` to indicate that the address should be formatted as `<pointer: 0x...>` rather than as `0x...`, as the longer format is recognized and ignored by `Rdiff`. The default value `NULL` is equivalent to `getOption("flint.Rdiff", FALSE)`. For greater control over output, consider doing `print(format(x, ...), ...)` instead of `print(x, ...)`.

`quantile signature(x = "flint")`:
: returns a `flint` vector containing sample quantiles computed according to additional argu-

ments probs and type; see quantile. Currently, an error is is signaled for x of length zero and x containing NaN.

rep signature(x = "flint"):
repeats x (or elements of x) according to optional arguments times, length.out, and each. The behaviour is parallel to that of the internal default method, hence see rep. One difference is that rep(*0-length* , length.out=*nonzero*) signals an error, because the underlying C types have no representation for missing values.

rep.int, rep_len signature(x = "flint"):
analogues of rep(x, times=) and rep(x, length.out=) not preserving names, faster than rep when x has names.

seq signature(... = "flint"):
generates flint vectors whose elements are equally spaced. This method is dispatched by calls to seq or seq.int in which the first positional argument is a flint vector. Accepted usage is any of

```
seq(length.out=)
seq(length.out=, by=)
seq(from=, to=)
seq(from=, to=, by=)
seq(from=, to=, length.out=)
seq(from=, by=, length.out=)
seq(to=, by=, length.out=)
```

where length.out=n and along.with=x are equivalent for x of length n. Good users name all arguments.

sequence signature(nvec = "flint"):
returns the concatenation of seq(from = from[i], by = by[i], length.out = nvec[i]) after recycling arguments nvec, from, and by to a common length.

show signature(object = "flint"):
prints format(object) and returns NULL invisibly.

summary signature(object = "flint"):
returns a flint vector containing the minimum, first quartile, median, mean, third quartile, maximum, and (if nonzero) the number of NaN, unless object is complex (inherits from acf or acb) or x has error bounds (inherits from arb or acb) or optional argument triple is TRUE, in which case the value is just flintTriple() with names.

unique signature(x = "flint"):
returns x[!duplicated(x, ...)].

Methods are on purpose *not* defined for generic functions whose default methods correctly handle objects inheriting from virtual class flint, typically by calling *other* generic functions for which methods *are* defined. Examples are as.character, as.list, diff, rev, seq.int, sort, and split.

See Also

The nonvirtual subclasses: ulong, slong, fmpz, fmpq, mag, arf, acf, arb, and acb.

Examples

```
showClass("flint")
showMethods(classes = "flint")
```

`fmpq-class` *Arbitrary Precision Rational Numbers*

Description

Class `fmpq` extends virtual class `flint`. It represents vectors of arbitrary precision rational numbers. Elements are specified by a pair of arbitrary precision signed integers: a numerator and a positive, coprime denominator. There is no representation for R's missing value `NA_integer_`.

Usage

```
## The class generator function:
.fmpq(...)

## Mode 1: initialize with zeros
## .fmpq(length = 0L)
##
## Mode 2: initialize with vector
## .fmpq(length = length(x), x)
##
## Mode 3: initialize by parts
## .fmpq(length = max0(length(num), length(den)), num, den)
##
## where max0 <- function(m, n) if (min(m, n)) max(m, n) else 0L
```

Arguments

`...`	arguments passed to methods for `initialize`.

Value

The class generator function returns `new("fmpq", ...)`.

Slots

`.xData`, `names` inherited from virtual class `flint`.

Methods

`! signature(x = "fmpq")`:
equivalent to (but faster than) `x == 0`.

`+ signature(e1 = "fmpq", e2 = "missing")`:
returns a copy of the argument.

`- signature(e1 = "fmpq", e2 = "missing")`:
returns the negation of the argument.

`Complex signature(z = "fmpq")`:
mathematical functions of one argument; see `S4groupGeneric`. Member functions requiring promotion to a floating-point type may not be implemented.

`Math signature(x = "fmpq")`:
mathematical functions of one argument; see `S4groupGeneric`. Member functions requiring promotion to a floating-point type may not be implemented.

`Math2 signature(x = "fmpq")`:
decimal rounding according to a second argument `digits`; see `S4groupGeneric`. There are just two member member functions: `round`, `signif`.

`Ops signature(e1 = "ANY", e2 = "fmpq")`:
`signature(e1 = "fmpq", e2 = "ANY")`:
binary arithmetic, comparison, and logical operators; see `S4groupGeneric`. The "other" operand must be atomic or inherit from virtual class `flint`. The operands are recycled and promoted as necessary.

`Summary signature(x = "fmpq")`:
univariate summary statistics; see `S4groupGeneric`. The return value is a logical vector of length 1 (`any`, `all`) or an `fmpq` vector of length 1 or 2 (`sum`, `prod`, `min`, `max`, `range`).

`anyNA signature(x = "fmpq")`:
returns `FALSE`, as `fmpq` has no representation for `NaN`.

`as.vector signature(x = "fmpq")`:
returns `as.vector(y, mode)`, where `y` is a double vector containing the result of converting each element of `x` to the range of double, rounding if the value is not exactly representable in double precision. The rounding mode is to the nearest representable number in the direction of zero, unless the element exceeds `.Machine[["double.xmax"]]` in absolute value, in which case `-Inf` or `Inf` is introduced with a warning. Coercion to types `"character"`, `"symbol"` (synonym `"name"`), `"pairlist"`, `"list"`, and `"expression"`, which are not "number-like", is handled specially.

`coerce signature(from = "ANY", to = "fmpq")`:
returns `.fmpq(x = from)`. An error is signaled if the class or type of `from` is not supported by the method for `initialize`.

`format signature(x = "fmpq")`:
returns a character vector suitable for printing, using string format `"p/q"`. Optional arguments control the output; see `format-methods`.

`initialize signature(.Object = "fmpq", length = "`*numeric*`")`:
returns `.Object` after setting its `.xData` slot. The new slot value is an external pointer to an array of the corresponding C type, which is newly allocated and initialized entirely with zeros (zero numerator, unit denominator). `.Object` is not copied, so all references are affected.

`initialize signature(.Object = "fmpq", length = "`*numeric*`", x = "`*atomic|flint*`")`:
as above, except that the new slot value points to an array initialized with values from `x`. An error is signaled if elements of `x` are `NaN`, `-Inf`, or `Inf`.

`initialize signature(.Object = "fmpq", length = "`*numeric*`", num = "`*atomic|flint*`", den = "`*atomic|flint*`")`:
as above, except that the new slot value points to an array whose numerators and denominators are initialized separately with values from `num` and `den`, which are coerced to class `fmpz` and

recycled as necessary. An error is signaled if elements of num or den are NaN, -Inf, or Inf or elements of den are zero.

`is.finite signature(x = "fmpq")`:
returns a logical vector whose elements are all TRUE, as fmpq has no representation for NaN, -Inf, and Inf.

`is.infinite`, `is.na`, `is.nan signature(x = "fmpq")`:
returns a logical vector whose elements are all FALSE, as fmpq has no representation for NaN, -Inf, and Inf.

`is.unsorted signature(x = "fmpq")`:
returns a logical indicating if x is not sorted in nondecreasing order (increasing order if optional argument strictly is set to TRUE).

`mean signature(x = "fmpq")`:
returns the arithmetic mean. An error is signaled if the argument length is 0, because the return type is fmpq which cannot represent the result of division by 0.

References

The FLINT documentation of the underlying C type: https://flintlib.org/doc/fmpq.html

See Also

Virtual class flint. Generic functions Num and Den and their replacement forms for getting and setting numerators and denominators.

Examples

```
showClass("fmpq")
showMethods(classes = "fmpq")
```

---

fmpz-class *Arbitrary Precision Signed Integers*

Description

Class fmpz extends virtual class flint. It represents vectors of arbitrary precision signed integers. There is no representation for R's missing value NA_integer_.

Usage

```
## The class generator function:
.fmpz(...)

## Mode 1: initialize with zeros
## .fmpz(length = 0L)
##
## Mode 2: initialize with vector
## .fmpz(length = length(x), x)
```

Arguments

`...`	arguments passed to methods for `initialize`.

Value

The class generator function returns `new("fmpz", ...)`.

Slots

`.xData`, `names` inherited from virtual class `flint`.

Methods

`! signature(x = "fmpz")`:
: equivalent to (but faster than) `x == 0`.

`+ signature(e1 = "fmpz", e2 = "missing")`:
: returns a copy of the argument.

`- signature(e1 = "fmpz", e2 = "missing")`:
: returns the negation of the argument.

`Complex signature(z = "fmpz")`:
: mathematical functions of one argument; see `S4groupGeneric`. Member functions requiring promotion to a floating-point type may not be implemented.

`Math signature(x = "fmpz")`:
: mathematical functions of one argument; see `S4groupGeneric`. Member functions requiring promotion to a floating-point type may not be implemented.

`Math2 signature(x = "fmpz")`:
: decimal rounding according to a second argument `digits`; see `S4groupGeneric`. There are just two member member functions: `round`, `signif`.

`Ops signature(e1 = "ANY", e2 = "fmpz")`:
`signature(e1 = "fmpz", e2 = "ANY")`:
: binary arithmetic, comparison, and logical operators; see `S4groupGeneric`. The "other" operand must be atomic or inherit from virtual class `flint`. The operands are recycled and promoted as necessary.

`Summary signature(x = "fmpz")`:
: univariate summary statistics; see `S4groupGeneric`. The return value is a logical vector of length 1 (`any`, `all`) or an `fmpz` vector of length 1 or 2 (`sum`, `prod`, `min`, `max`, `range`).

`anyNA signature(x = "fmpz")`:
: returns `FALSE`, as `fmpz` has no representation for `NaN`.

`as.vector signature(x = "fmpz")`:
: returns `as.vector(y, mode)`, where `y` is a double vector containing the result of converting each element of `x` to the range of double, rounding if the value is not exactly representable in double precision. The rounding mode is to the nearest representable number in the direction of zero, unless the element exceeds `.Machine[["double.xmax"]]` in absolute value, in which case `-Inf` or `Inf` is introduced with a warning. Coercion to types `"character"`, `"symbol"` (synonym `"name"`), `"pairlist"`, `"list"`, and `"expression"`, which are not "number-like", is handled specially.

`coerce signature(from = "ANY", to = "fmpz")`:
returns `.fmpz(x = from)`. An error is signaled if the class or type of `from` is not supported by the method for `initialize`.

`format signature(x = "fmpz")`:
returns a character vector suitable for printing. Optional arguments control the output; see `format-methods`.

`initialize signature(.Object = "fmpz", length = "numeric")`:
returns `.Object` after setting its `.xData` slot. The new slot value is an external pointer to an array of the corresponding C type, which is newly allocated and initialized entirely with zeros. `.Object` is not copied, so all references are affected.

`initialize signature(.Object = "fmpz", length = "numeric", x = "atomic|flint")`:
as above, except that the new slot value points to an array initialized with values from `x` truncated towards zero. An error is signaled if elements of `x` are `NaN`, `-Inf`, or `Inf`.

`is.finite` returns a logical vector whose elements are all `TRUE`, as `fmpz` has no representation for `NaN`, `-Inf`, and `Inf`.

`is.infinite`, `is.na`, `is.nan signature(x = "fmpz")`:
returns a logical vector whose elements are all `FALSE`, as `fmpz` has no representation for `NaN`, `-Inf`, and `Inf`.

`is.unsorted signature(x = "fmpz")`:
returns a logical indicating if `x` is not sorted in nondecreasing order (increasing order if optional argument `strictly` is set to `TRUE`).

`mean signature(x = "fmpz")`:
returns the arithmetic mean. An error is signaled if the argument length is 0, because the return type is `fmpq` which cannot represent the result of division by 0.

References

The FLINT documentation of the underlying C type: https://flintlib.org/doc/fmpz.html

See Also

Virtual class `flint`.

Examples

```
showClass("fmpz")
showMethods(classes = "fmpz")
```

`format-methods` *Format FLINT-type Numbers as Strings*

Description

Format a `flint` vector for pretty printing.

Usage

```
## S4 method for signature 'ulong'
format(x, base = 10L, ...)
## S4 method for signature 'slong'
format(x, base = 10L, ...)
## S4 method for signature 'fmpz'
format(x, base = 10L, ...)
## S4 method for signature 'fmpq'
format(x, base = 10L, ...)
## S4 method for signature 'mag'
format(x, base = 10L, digits = NULL, sep = NULL,
       rnd = flintRnd(), ...)
## S4 method for signature 'arf'
format(x, base = 10L, digits = NULL, sep = NULL,
       rnd = flintRnd(), ...)
## S4 method for signature 'acf'
format(x, base = 10L, digits = NULL, sep = NULL,
       rnd = flintRnd(), ...)
## S4 method for signature 'arb'
format(x, base = 10L, digits = NULL, sep = NULL,
       rnd = flintRnd(), ...)
## S4 method for signature 'acb'
format(x, base = 10L, digits = NULL, sep = NULL,
       rnd = flintRnd(), ...)
```

Arguments

x	a flint vector.
base	an integer from 2 to 62 indicating a base for output. Values 2, 10, and 16 correspond to binary, decimal, and hexadecimal output. Digits are represented by characters '[0-9A-Za-z]', in that significance order, hence the maximum 10+26+26=62.
digits	an integer indicating how many digits of the significand are reported when formatting floating type vectors. When more than one digit is printed, a radix point inserted after the first digit. Value 0 is equivalent to the minimum integer d such that all elements of x are represented exactly by d digits in the specified base. The default value NULL is equivalent to getOption("digits", 99999L).
sep	a nonempty character string used to separate the significand from the exponent. The default value NULL is a equivalent to "e" for base equal to 10 and to "@" for all other bases.
rnd	a nonempty character string whose first character indicates a rounding mode. Methods for arb and acb require rnd of length 2, specifying rounding modes separately for midpoints and radii. See flintRnd for information about valid character strings.
...	further optional arguments, though these are currently unused.

Value

A character vector containing ASCII strings of equal length.

Examples

```
q <- .fmpq(num = c(-1L, 1L) * 0:5, den = 1:6)
for (b in 2:8) {
    cat("base = ", b, ":\n", sep = "")
    print(format(q, base = b), quote = FALSE, width = 12L)
}

z <- .acb(real = .arb(mid = pi, rad = 0.5 * pi))
format(z)
format(z, base = 62L, sep = "*[62]^")
strsplit(format(Re(z), digits = 80L), "[( )]")[[1L]][c(FALSE, TRUE)]
```

---

mag-class *Fixed Precision Magnitude (Error) Bounds*

Description

Class `mag` extends virtual class `flint`. It represents vectors of fixed precision error bounds. Elements are unsigned floating-point numbers with a 30-bit significand and an arbitary precision exponent. The underlying C type can represent `Inf` but not `NaN`.

Usage

```
## The class generator function:
.mag(...)

## Mode 1: initialize with zeros
## .mag(length = 0L)
##
## Mode 2: initialize with vector
## .mag(length = length(x), x)
```

Arguments

`...`	arguments passed to methods for `initialize`.

Value

The class generator function returns `new("mag", ...)`.

Slots

`.xData`, `names` inherited from virtual class `flint`.

Methods

`! signature(x = "mag")`:
: equivalent to (but faster than) `x == 0`.

`+ signature(e1 = "mag", e2 = "missing")`:
: returns a copy of the argument.

`- signature(e1 = "mag", e2 = "missing")`:
: returns a copy of the argument, to be consistent with the binary operation which returns an upper bound for the absolute value of the difference.

`Complex signature(z = "mag")`:
: mathematical functions of one argument; see `S4groupGeneric`. The return value is an upper bound for the absolute value of the exact answer.

`Math signature(x = "mag")`:
: mathematical functions of one argument; see `S4groupGeneric`. The return value is an upper bound for the absolute value of the exact answer. Notably, the (inverse) trigonometric, (inverse) hyperbolic, and gamma-related member functions are not yet implemented. Users wanting those can (for now) operate on `as(x, "arb")`.

`Math2 signature(x = "mag")`:
: decimal rounding according to a second argument `digits`; see `S4groupGeneric`. There are just two member functions: `round`, `signif`. The return value is an upper bound for the exact answer.

`Ops signature(e1 = "ANY", e2 = "mag")`:
`signature(e1 = "mag", e2 = "ANY")`:
: binary arithmetic, comparison, and logical operators; see `S4groupGeneric`. The "other" operand must be atomic or inherit from virtual class `flint`. The operands are recycled and promoted as necessary. For arithmetic, the return value is a `mag` vector only if both operands are `mag` vectors. In that case, the return value is an upper bound for the absolute value of the exact answer. Users wanting "standard" floating-point arithmetic must ensure that at least one operand is not a `mag` vector.

`Summary signature(x = "mag")`:
: univariate summary statistics; see `S4groupGeneric`. The return value is a logical vector of length 1 (`any`, `all`) or an `mag` vector of length 1 or 2 (`sum`, `prod`, `min`, `max`, `range`). For `sum` and `prod`, the return value is an upper bound for the exact answer.

`anyNA signature(x = "mag")`:
: returns `FALSE`, as `mag` has no representation for `NaN`.

`as.vector signature(x = "mag")`:
: returns `as.vector(y, mode)`, where `y` is a double vector containing the result of converting each element of `x` to the range of double, rounding in the direction of `Inf`, not always to nearest. Coercion to types `"character"`, `"symbol"` (synonym `"name"`), `"pairlist"`, `"list"`, and `"expression"`, which are not "number-like", is handled specially.

`coerce signature(from = "ANY", to = "mag")`:
: returns `.mag(x = from)`. An error is signaled if the class or type of `from` is not supported by the method for `initialize`.

`format signature(x = "mag")`:
: returns a character vector suitable for printing, using scientific format. Optional arguments control the output; see `format-methods`.

`initialize signature(.Object = "mag", length = "numeric")`:
returns `.Object` after setting its `.xData` slot. The new slot value is an external pointer to an array of the corresponding C type, which is newly allocated and initialized entirely with zeros. `.Object` is not copied, so all references are affected.

`initialize signature(.Object = "mag", length = "numeric", x = "atomic|flint")`:
as above, except that the new slot value points to an array initialized with upper bounds for `abs(x)`. An error is signaled if any `x[i]` is `NaN`.

`is.finite signature(x = "mag")`:
returns a logical vector indicating which elements of `x` are not `Inf`.

`is.infinite signature(x = "mag")`:
returns a logical vector indicating which elements of `x` are `Inf`.

`is.na, is.nan signature(x = "mag")`:
returns a logical vector whose elements are all `FALSE`, as `mag` has no representation for `NaN`.

`is.unsorted signature(x = "mag")`:
returns a logical indicating if `x` is not sorted in nondecreasing order (increasing order if optional argument `strictly` is set to `TRUE`).

`log signature(x = "mag")`:
returns an upper bound for the absolute value of the logarithm of the argument. The natural logarithm is computed by default (when optional argument `base` is unset).

`mean signature(x = "mag")`:
returns an upper bound for the arithmetic mean.

References

The FLINT documentation of the underlying C type: https://flintlib.org/doc/mag.html

Johansson, F. (2017). Arb: efficient arbitrary-precision midpoint-radius interval arithmetic. *IEEE Transactions on Computers*, *66*(8), 1281-1292. doi:10.1109/TC.2017.2690633

See Also

Virtual class `flint`.

Examples

```
showClass("mag")
showMethods(classes = "mag")
```

`Part` *Get or Set One Part of a Vector*

Description

The subclasses of virtual class `flint` are interfaces to C types in the FLINT C library. For types implemented recursively as C structs, it is often very natural to get and set the struct members. The functions documented here provide support for this common operation; they are all S4 generic.

Usage

```
Num(q)
Num(q) <- value
Den(q)
Den(q) <- value

Mid(x)
Mid(x) <- value
Rad(x)
Rad(x) <- value

Real(z)
Real(z) <- value
Imag(z)
Imag(z) <- value
```

Arguments

q	a vector-like R object with elements representing quotients of numbers. Package **flint** provides methods for class fmpq.
x	a vector-like R object with elements representing balls in a metric space. Package **flint** provides methods for class arb.
z	a vector-like R object with elements representing complex numbers. Package **flint** provides methods for classes acf and acb.
value	a vector-like R object; the replacement value. Methods in package **flint** support atomic vectors and vectors inheriting from virtual class flint. They inherit recycling behaviour from methods for initialize; see, e.g., selectMethod("Num<-", "fmpq").

Details

Num and Den extract fmpz numerators and denominators from fmpq q. The replacement form of Num constructs a new fmpq vector from value (coerced to fmpz) and Den(q). The replacement form of Den constructs a new fmpq vector from Num(q) and value (coerced to fmpz).

Mid and Rad extract arf midpoints and mag radii from arb x. The replacement form of Mid constructs a new arb vector from value (coerced to arf) and Rad(x). The replacement form of Rad constructs a new arb vector from Mid(x) and value (coerced to mag).

Real and Imag extract arf real and imaginary parts from acf z and arb real and imaginary parts from acb z. The replacement form of Real constructs a new acf or acb vector from value (coerced to arf or arb) and Imag(z). The replacement form of Imag constructs a new acf or acb vector from Real(z) and value (coerced to arf or arb).

Value

Num, Den, Mid, Rad, Real, and Imag return a vector-like R object of length matching the argument. The replacement forms return a vector-like R object of length matching either the argument or the replacement value; the class of the result is, if not the class of the argument, then a superclass. See 'Details' for behaviour specific to methods in package **flint**.

See Also

Virtual class flint.

Examples

```
(q <- q. <- .fmpq(num = 1:10, den = 2L))
Num(q)
Den(q)
Num(q) <- Den(q)
q
(m <- Num(q))
(n <- Den(q))
stopifnot(m == 1L, n == 1L, q == 1L)
```

---

ulong-class *Fixed Precision Unsigned and Signed Integers*

Description

Classes ulong and slong extend virtual class flint. They represent vectors of fixed precision unsigned and signed integers, respectively. The integer size is 32 or 64 bits, depending on the ABI; see flintABI. There is no representation for R's missing value NA_integer_.

Usage

```
## The class generator functions:
.ulong(...)
.slong(...)

## Mode 1: initialize with zeros
## .ulong(length = 0L)
## .slong(length = 0L)
##
## Mode 2: initialize with vector
## .ulong(length = length(x), x)
## .slong(length = length(x), x)
```

Arguments

...	arguments passed to methods for initialize.

Value

The class generator functions return new("ulong", ...) and new("slong", ...).

Slots

.xData, names inherited from virtual class flint.

Methods

`! signature(x = "ulong")`:
`signature(x = "slong")`:
equivalent to (but faster than) `x == 0`.

`+ signature(e1 = "ulong", e2 = "missing")`:
`signature(e1 = "slong", e2 = "missing")`:
returns a copy of the argument.

`- signature(e1 = "ulong", e2 = "missing")`:
`signature(e1 = "slong", e2 = "missing")`:
returns the negation of the argument.

`Complex signature(z = "ulong")`:
`signature(z = "slong")`:
mathematical functions of one argument; see `S4groupGeneric`. Member functions requiring promotion to a floating-point type may not be implemented.

`Math signature(x = "ulong")`:
`signature(x = "slong")`:
mathematical functions of one argument; see `S4groupGeneric`. Member functions requiring promotion to a floating-point type may not be implemented.

`Math2 signature(x = "ulong")`:
`signature(x = "slong")`:
decimal rounding according to a second argument `digits`; see `S4groupGeneric`. There are just two member member functions: `round`, `signif`.

`Ops signature(e1 = "ANY", e2 = "ulong")`:
`signature(e1 = "ANY", e2 = "slong")`:
`signature(e1 = "ulong", e2 = "ANY")`:
`signature(e1 = "slong", e2 = "ANY")`:
binary arithmetic, comparison, and logical operators; see `S4groupGeneric`. The "other" operand must be atomic or inherit from virtual class `flint`. The operands are recycled and promoted as necessary.

`Summary signature(x = "ulong")`:
`signature(x = "slong")`:
univariate summary statistics; see `S4groupGeneric`. The return value is a logical vector of length 1 (`any`, `all`) or a `ulong`, `slong`, or `fmpz` vector of length 1 or 2 (`sum`, `prod`, `min`, `max`, `range`).

`anyNA signature(x = "ulong")`:
`signature(x = "slong")`:
returns `FALSE`, as `ulong` and `slong` have no representation for `NaN`.

`as.vector signature(x = "ulong")`:
`signature(x = "slong")`:
returns `as.vector(y, mode)`, where `y` is a double vector containing the result of converting each element of `x` to the range of double, rounding if the value is not exactly representable in double precision. The rounding mode is to the nearest representable number in the direction of zero. Coercion to types `"character"`, `"symbol"` (synonym `"name"`), `"pairlist"`, `"list"`, and `"expression"`, which are not "number-like", is handled specially.

`coerce signature(from = "ANY", to = "ulong")`:
`signature(from = "ANY", to = "slong")`:
returns `.ulong(x = from)` or `.slong(x = from)`. An error is signaled if the class or type of `from` is not supported by the methods for `initialize`.

`format signature(x = "ulong")`:
`signature(x = "slong")`:
returns a character vector suitable for printing. Optional arguments control the output; see `format-methods`.

`initialize signature(.Object = "ulong", length = "numeric")`:
`signature(.Object = "slong", length = "numeric")`:
returns `.Object` after setting its `.xData` slot. The new slot value is an external pointer to an array of the corresponding C type, which is newly allocated and initialized entirely with zeros. `.Object` is not copied, so all references are affected.

`initialize signature(.Object = "ulong", length = "numeric", x = "atomic|flint")`:
`signature(.Object = "slong", length = "numeric", x = "atomic|flint")`:
as above, except that the new slot value points to an array initialized with values from `x` truncated towards zero. An error is signaled if elements of `x` are not in the range of the C type, in particular if elements are `NaN`. The range is $(-1, 2^n)$ for `ulong` and $(-2^{n-1} - 1, 2^{n-1})$ for `slong`, where n is the value of `flintABI()`.

`is.finite signature(x = "ulong")`:
`signature(x = "slong")`:
returns a logical vector whose elements are all `TRUE`, as `ulong` and `slong` have no representation for `NaN`, `-Inf`, and `Inf`.

`is.infinite`, `is.na`, `is.nan signature(x = "ulong")`:
`signature(x = "slong")`:
returns a logical vector whose elements are all `FALSE`, as `ulong` and `slong` have no representation for `NaN`, `-Inf`, and `Inf`.

`is.unsorted signature(x = "ulong")`:
`signature(x = "slong")`:
returns a logical indicating if `x` is not sorted in nondecreasing order (increasing order if optional argument `strictly` is set to `TRUE`).

`mean signature(x = "ulong")`:
`signature(x = "slong")`:
returns the arithmetic mean. An error is signaled if the argument length is 0, because the return type is `fmpq` which cannot represent the result of division by 0.

References

The FLINT documentation of the underlying C types: https://flintlib.org/doc/flint.html

See Also

Virtual class `flint`.

Examples

```
showClass("ulong")
```

```
showClass("slong")
showMethods(classes = c("ulong", "slong"))
```

Index