Package ‘flint’

December 19, 2025

Version 0.1.4

VersionNote sync configure.ac, inst/NEWS.Rd

Date 2025-12-18

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 and other 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

Enhances Rmpfr, gmp

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>), Martin Maechler [ctb] (ORCID: <https://orcid.org/0000-0002-8685-9910>)

Maintainer Mikael Jagan <jaganmn2@gmail.com>

Repository CRAN

Date/Publication 2025-12-19 10:20:14 UTC

Contents

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

Description

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

Usage

```
flintABI()
flintBits(object)
flintBitsAccurate(object)
flintClass(object)
flintLength(object, exact = TRUE)
flintPrec(prec = NULL)
flintRnd(rnd = NULL)
flintRndMag(rnd.mag = NULL)
flintSize(object)
flintTriple(object)
flintVersion()

.initForeign(which = c("Rmpfr", "gmp"),
             where = topenv(parent.frame()))
```

Arguments

`object`	an R object, typically inheriting from virtual class `flint`.
`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 with exactly one character indicating a rounding direction. Valid characters are '`[Dd]`' (towards negative infinity), '`[Uu]`' (towards positive infinity), '`[Zz]`' (towards zero), '`[Aa]`' (away from zero), and '`[Nn]`' (to nearest, with precedence to even significands).
`rnd.mag`	as `rnd`, but used exclusively for operations returning `mag`. Note that since `mag` is unsigned, '`[Dd]`' and '`[Uu]`' are equivalent to '`[Zz]`' and '`[Aa]`', respectively. '`[Nn]`' is invalid as (by design) `mag` does not support rounding to nearest.
`which`	a character vector listing package names.
`where`	an environment for storing methods, by default the **flint** namespace.

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.

flintClass returns a character string naming the direct nonvirtual subclass of virtual class flint from which object inherits. (Hence a possible value is "ulong" but not the name of any subclass of ulong.) If object does not inherit from virtual class flint, then the return value is NA_character_.

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 $1, \ldots, 2^p$). If object does not inherit from virtual class flint, then the return value is NA_integer_.

flintPrec returns the previous default precision, where the so-called factory setting is .Machine$double.digits.

flintRnd and flintRndMag return the previous default rounding modes, where the so-called factory settings are "N" and "A", respectively.

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 length zero. Another corollary is that the bytes counted by flintSize and the bytes counted by object.size are disjoint. If object does not inherit from virtual class flint, then the return value is NA_real_ (beneath the class).

flintTriple returns a character vector of length 3 containing the class of object, the length of object, and the address stored by object. If object does not inherit from virtual class flint, then all of the elements are NA.

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).

.initForeign defines methods for coerce enabling coercion between classes in **flint** and analogous classes in the packages named by which. Packages are loaded before methods are defined to ensure that setAs is able to find class definitions. A corollary is that an error is signaled if packages are not installed in the library search path. Supported signatures:

```
which="Rmpfr":
  from="mpfr", to="arf"
  from="arf", to="mpfr"
which="gmp":
  from="bigz", to="fmpz"
  from="fmpz", to="bigz"
  from="bigq", to="fmpq"
  from="fmpq", to="bigq"
```

Note

Whether and how the global default precision and rounding modes (set by flintPrec, flintRnd, and flintRndMag) are actually used depends on conventions defined in the floating-point class documentation, hence see mag, arf, acf, arb, and acb. These conventions are partly inherited from the C library.

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

```
## Class generator functions

acb(x = 0i, length = 0L, names = NULL, real = 0, imag = 0,
    prec = NULL)

acb.array(x = 0i, dim = length(x), dimnames = NULL, real = 0, imag = 0,
          prec = NULL)
```

Arguments

x	an atomic or flint vector containing data for conversion to acb.
length	a numeric vector of length one giving the length of the return value. If that exceeds the length of x, then x is recycled. Non-integer values are rounded in the direction of zero.
names	the names slot of the return value, either NULL or a character vector of equal length. Non-character names are coerced to character.
dim	the dim slot of the return value, an integer vector of nonzero length. If the product exceeds the length of x, then x is recycled. Non-integer numeric dim are coerced to integer.
dimnames	the dimnames slot of the return value, either NULL or a list of length equal to the length of dim. The components are either NULL or character vectors of length given by dim. Non-character vector components of dimnames are coerced to character.
real, imag	atomic or flint vectors containing data for conversion to arb. Use these for initialization “by parts” (real and imaginary).
prec	the precision used for conversion of midpoints. NULL means to convert exactly if possible and to round to the global default precision otherwise; see flintPrec. By convention, rounding is always towards zero.

Details

The class generator function has six distinct usages:

```
acb()
acb(length=)
acb(x)
acb(x, length=)
acb(real=, imag=)
acb(real=, imag=, length=)
```

The first usage generates an empty vector. The second usage generates a zero vector of the indicated length. The third usage converts `x`, preserving dimensions, dimension names, and names. The fourth usage converts `x`, recycling its elements to the indicated length and discarding its dimensions, dimension names, and names. The fifth and sixth usages, in which either of `real` and `imag` can be missing, use `arb(real)` and `arb(imag)` to separately initialize the real and imaginary parts of the `acb` return value.

Attempts to recycle `real`, `imag`, or `x` of length zero to nonzero length are an error.

Usage of `acb.array` is modelled after `array`.

Value

An `acb` vector, possibly an array; see 'Details'.

Conversion

Real numbers and real and imaginary parts of complex numbers are rounded according to the precision set by `prec`, always in the direction of zero. Ball midpoints are the numbers obtained by rounding. Ball radii are upper bounds on the absolute errors incurred by rounding.

Character strings are scanned first for a real part then for an imaginary part. They can use any of three formats: "*sa*", "*tb*i", and "*satb*i", where, recursively, each of *a* and *b* have the format "(*km*+/-*r*)", defining a ball for each of the real and imaginary parts. *k* and *m* define the sign and absolute value of the signed ball midpoints, and *r* defines the unsigned ball radii. *k* can be empty if the ball midpoint is `NaN` or non-negative. *s* and *t* are unary or binary plus or minus to be reconciled with *k*; they are optional except in the third format where *t* is mandatory.

The sequences *km* and *r* are converted using function `mpfr_strtofr` from the GNU MPFR library with argument `base` set to 0; see https://www.mpfr.org/mpfr-current/mpfr.html#Assignment-Functions.

Slots

`.xData`, `dim`, `dimnames`, `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`.

`%*%`, `crossprod`, `tcrossprod` `signature(x = "acb", y = "acb")`:
: `signature(x = "acb", y = "ANY")`:
  `signature(x = "ANY", y = "acb")`:
  matrix products. The "other" operand must be atomic or inherit from virtual class `flint`. `crossprod` and `tcrossprod` behave as if `y = x` when `y` is missing or `NULL`. Operands are promoted as necessary and must be conformable (have compatible dimensions). Non-array operands of length `k` are handled as 1-by-`k` or `k`-by-1 matrices depending on the call.

`+` `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 = "acb", e2 = "acb")`:
: `signature(e1 = "acb", e2 = "ANY")`:
  `signature(e1 = "ANY", e2 = "acb")`:
  binary arithmetic, comparison, and logical operators; see `S4groupGeneric`. The "other" operand must be atomic or inherit from virtual class `flint`. Operands are promoted as necessary. Array operands must be conformable (have identical dimensions). Non-array operands are recycled.

`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. See also `asVector`.

`backsolve` `signature(r = "acb", x = "acb")`:
: `signature(r = "acb", x = "ANY")`:

signature(r = "ANY", x = "acb"):
solution of the triangular system op2(op1(r)) %*% y = x, where op1=ifelse(upper.tri, triu, tril) and op2=ifelse(transpose, t, identity) and upper.tri and transpose are optional logical arguments with default values TRUE and FALSE, respectively. The "other" operand must be atomic or inherit from virtual class flint. If x is missing, then the return value is the inverse of op2(op1(r)), as if x were the identity matrix. Operands are promoted as necessary and must be conformable (have compatible dimensions). Non-array x are handled as length(x)-by-1 matrices.

chol signature(x = "acb"):
returns the upper triangular Cholesky factor of the positive definite matrix whose upper triangular part is the upper triangular part of x (discarding imaginary parts of diagonal entries).

chol2inv signature(x = "acb"):
returns the inverse of the positive definite matrix whose upper triangular Cholesky factor is the upper triangular part of x (discarding imaginary parts of diagonal entries).

coerce signature(from = "ANY", to = "acb"):
returns the value of acb(from).

colSums, colMeans signature(x = "acb"):
returns an acb vector or array containing the column sums or means of x, defined as sums or means over dimensions 1:dims.

det signature(x = "arb"):
returns the determinant of x as an acb vector of length 1.

determinant signature(x = "acf"):
returns a list with components modulus and argument specifying the determinant of x, following the documented behaviour of the **base** function (except for the use of argument instead of sign).

diff signature(x = "acb"):
returns a vector storing lagged differences of the elements of x or (if x is a matrix) a matrix storing lagged differences of the rows of x, following the documented behaviour of the S3 default method.

diffinv signature(x = "acb"):
returns the vector or matrix y such that x = diff(y, ...), following the documented behaviour of the S3 default method.

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.

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.

`rowSums`, `rowMeans` `signature(x = "acb")`:
: returns an `acb` vector or array containing the row sums or means of `x`, defined as sums or means over dimensions `(dims+1):length(dim(x))`.

`solve` `signature(a = "acb", b = "acb")`:
: `signature(a = "acb", b = "ANY")`:
`signature(a = "ANY", b = "acb")`:
solution of the general system `a %*% x = b`. The "other" operand must be atomic or inherit from virtual class `flint`. If `b` is missing, then the return value is the inverse of `a`, as if `b` were the identity matrix. Operands are promoted as necessary and must be conformable (have compatible dimensions). Non-array `b` are handled as `length(b)`-by-1 matrices.

`xtfrm` `signature(x = "acb")`:
: signals an error indicating that <= is not a total order on the range of `arb`: `a <= b || b <= a` 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 unrelated.

The class generator functions are named ACF and ACF.array instead of acf and acf.array because an exported function named acf would mask the function in package **stats**.

Usage

```
## Class generator functions

ACF(x = 0i, length = 0L, names = NULL, real = 0, imag = 0,
    prec = NULL, rnd = NULL)

ACF.array(x = 0i, dim = length(x), dimnames = NULL, real = 0, imag = 0,
          prec = NULL, rnd = NULL)
```

Arguments

x	an atomic or flint vector containing data for conversion to acf.
length	a numeric vector of length one giving the length of the return value. If that exceeds the length of x, then x is recycled. Non-integer values are rounded in the direction of zero.
names	the names slot of the return value, either NULL or a character vector of equal length. Non-character names are coerced to character.
dim	the dim slot of the return value, an integer vector of nonzero length. If the product exceeds the length of x, then x is recycled. Non-integer numeric dim are coerced to integer.
dimnames	the dimnames slot of the return value, either NULL or a list of length equal to the length of dim. The components are either NULL or character vectors of length given by dim. Non-character vector components of dimnames are coerced to character.
real, imag	atomic or flint vectors containing data for conversion to arf. Use these instead of x for initialization “by parts” (real and imaginary).
prec	the precision used for conversion. NULL means to convert exactly if possible and to round to the global default precision otherwise; see flintPrec.
rnd	the rounding mode used for inexact conversion. NULL means to round according to the global default rounding mode; see flintRnd.

Details

The class generator function has six distinct usages:

```
acf.()
acf.(length=)
acf.(x)
acf.(x, length=)
acf.(real=, imag=)
acf.(real=, imag=, length=)
```

The first usage generates an empty vector. The second usage generates a zero vector of the indicated length. The third usage converts `x`, preserving dimensions, dimension names, and names. The fourth usage converts `x`, recycling its elements to the indicated length and discarding its dimensions, dimension names, and names. The fifth and sixth usages, in which either of `real` and `imag` can be missing, use `arf(real)` and `arf(imag)` to separately initialize the real and imaginary parts of the `acf` return value.

Attempts to recycle `real`, `imag`, or `x` of length zero to nonzero length are an error.

Usage of `acf.array` is modelled after `array`.

Value

An `acf` vector, possibly an array; see 'Details'.

Conversion

Real numbers and real and imaginary parts of complex numbers are rounded according to the precision and rounding mode set by `prec` and `rnd`.

Character strings are scanned first for a real part then for an imaginary part. They can use any of three formats: `"`*sa*`"`, `"`*tb*`i"`, and `"`*satb*`i"`, where *s* and *a* define the sign and absolute value of the real part and *t* and *b* define the sign and absolute value of the imaginary part. *s* can be empty if the real part is `NaN` or non-negative. *t* can be empty if the imaginary part is `NaN` or non-negative, but only in the second format.

The sequences *sa* and *tb* are converted using function `mpfr_strtofr` from the GNU MPFR library with argument `base` set to 0; see https://www.mpfr.org/mpfr-current/mpfr.html#Assignment-Functions.

Slots

`.xData`, `dim`, `dimnames`, `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`.

%*%, crossprod, tcrossprod `signature(x = "acf", y = "acf")`:
: `signature(x = "acf", y = "ANY")`:
`signature(x = "ANY", y = "acf")`:
matrix products. The "other" operand must be atomic or inherit from virtual class `flint`. `crossprod` and `tcrossprod` behave as if `y = x` when `y` is missing or `NULL`. Operands are promoted as necessary and must be conformable (have compatible dimensions). Non-array operands of length `k` are handled as 1-by-`k` or `k`-by-1 matrices depending on the call. The return value is approximate insofar that it may not be correctly rounded.

`+` `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 = "acf", e2 = "acf")`:
: `signature(e1 = "acf", e2 = "ANY")`:
`signature(e1 = "ANY", e2 = "acf")`:
binary arithmetic, comparison, and logical operators; see `S4groupGeneric`. The "other" operand must be atomic or inherit from virtual class `flint`. Operands are promoted as necessary. Array operands must be conformable (have identical dimensions). Non-array operands are recycled.

`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. See also `asVector`.

`backsolve` `signature(r = "acf", x = "acf")`:
: `signature(r = "acf", x = "ANY")`:
`signature(r = "ANY", x = "acf")`:
solution of the triangular system `op2(op1(r)) %*% y = x`, where `op1=ifelse(upper.tri,`

triu, tril) and op2=ifelse(transpose, t, identity) and upper.tri and transpose are optional logical arguments with default values TRUE and FALSE, respectively. The "other" operand must be atomic or inherit from virtual class flint. If x is missing, then the return value is the inverse of op2(op1(r)), as if x were the identity matrix. Operands are promoted as necessary and must be conformable (have compatible dimensions). Non-array x are handled as length(x)-by-1 matrices.

chol signature(x = "acf"):
: returns the upper triangular Cholesky factor of the positive definite matrix whose upper triangular part is the upper triangular part of x (discarding imaginary parts of diagonal entries).

chol2inv signature(x = "acf"):
: returns the inverse of the positive definite matrix whose upper triangular Cholesky factor is the upper triangular part of x (discarding imaginary parts of diagonal entries).

coerce signature(from = "ANY", to = "acf"):
: returns the value of acf.(from).

colSums, colMeans signature(x = "acf"):
: returns an acf vector or array containing the column sums or means of x, defined as sums or means over dimensions 1:dims.

det signature(x = "acf"):
: returns the determinant of x as an acf vector of length 1.

determinant signature(x = "acf"):
: returns a list with components modulus and argument specifying the determinant of x, following the documented behaviour of the **base** function (except for the use of argument instead of sign).

diff signature(x = "acf"):
: returns a vector storing lagged differences of the elements of x or (if x is a matrix) a matrix storing lagged differences of the rows of x, following the documented behaviour of the S3 default method.

diffinv signature(x = "acf"):
: returns the vector or matrix y such that x = diff(y, ...), following the documented behaviour of the S3 default method.

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.

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.

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

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

`rowSums`, `rowMeans` `signature(x = "acf")`:
: returns an `acf` vector or array containing the row sums or means of `x`, defined as sums or means over dimensions `(dims+1):length(dim(x))`.

`solve` `signature(a = "acf", b = "acf")`:
: `signature(a = "acf", b = "ANY")`:
: `signature(a = "ANY", b = "acf")`:
: solution of the general system `a %*% x = b`. The "other" operand must be atomic or inherit from virtual class `flint`. If `b` is missing, then the return value is the inverse of `a`, as if `b` were the identity matrix. Operands are promoted as necessary and must be conformable (have compatible dimensions). Non-array `b` are handled as `length(b)`-by-1 matrices.

`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

```
## Class generator functions

arb(x = 0, length = 0L, names = NULL, mid = 0, rad = 0,
    prec = NULL)

arb.array(x = 0, dim = length(x), dimnames = NULL, mid = 0, rad = 0,
          prec = NULL)
```

Arguments

`x`	an atomic or `flint` vector containing data for conversion to `arb`.
`length`	a numeric vector of length one giving the length of the return value. If that exceeds the length of `x`, then `x` is recycled. Non-integer values are rounded in the direction of zero.
`names`	the `names` slot of the return value, either `NULL` or a character vector of equal length. Non-character `names` are coerced to character.
`dim`	the `dim` slot of the return value, an integer vector of nonzero length. If the product exceeds the length of `x`, then `x` is recycled. Non-integer numeric `dim` are coerced to integer.
`dimnames`	the `dimnames` slot of the return value, either `NULL` or a list of length equal to the length of `dim`. The components are either `NULL` or character vectors of length given by `dim`. Non-character vector components of `dimnames` are coerced to character.
`mid, rad`	atomic or `flint` vectors containing data for conversion to `arf` and `mag`, respectively. Use these for initialization "by parts" (midpoint and radius).
`prec`	the precision used for conversion of midpoints. `NULL` means to convert exactly if possible and to round to the global default precision otherwise; see `flintPrec`.

Details

The class generator function has six distinct usages:

```
arb()
arb(length=)
arb(x)
arb(x, length=)
arb(mid=, mid=)
arb(mid=, mid=, length=)
```

The first usage generates an empty vector. The second usage generates a zero vector of the indicated length. The third usage converts `x`, preserving dimensions, dimension names, and names. The fourth usage converts `x`, recycling its elements to the indicated length and discarding its dimensions, dimension names, and names. The fifth and sixth usages, in which either of `mid` and `rad` can be missing, use `arf(mid)` and `mag(rad)` to separately initialize the midpoints and radii of the `arb` return value.

Attempts to recycle `mid`, `rad`, or `x` of length zero to nonzero length are an error.

Usage of `arb.array` is modelled after `array`.

Value

An arb vector, possibly an array; see 'Details'.

Conversion

Real numbers and real parts of complex numbers are rounded according to the precision set by prec, always in the direction of zero. Ball midpoints are the numbers obtained by rounding. Ball radii are upper bounds on the absolute errors incurred by rounding. Imaginary parts of complex numbers are discarded.

Character strings are scanned for format "*s*(*km*+/-*r*)", where *k* and *m* define the sign and absolute value of the signed ball midpoint, and *r* defines the unsigned ball radius. *k* can be empty if the ball midpoint is NaN or non-negative. *s* is an optional unary plus or minus to be reconciled with *k*.

The sequences *km* and *r* are converted using function mpfr_strtofr from the GNU MPFR library with argument base set to 0; see https://www.mpfr.org/mpfr-current/mpfr.html#Assignment-Functions.

Slots

.xData, dim, dimnames, 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.

%*%, crossprod, tcrossprod signature(x = "arb", y = "arb"):
signature(x = "arb", y = "ANY"):
signature(x = "ANY", y = "arb"):
matrix products. The "other" operand must be atomic or inherit from virtual class flint. crossprod and tcrossprod behave as if y = x when y is missing or NULL. Operands are promoted as necessary and must be conformable (have compatible dimensions). Non-array operands of length k are handled as 1-by-k or k-by-1 matrices depending on the call.

+ 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 = "arb", e2 = "arb"):
signature(e1 = "arb", e2 = "ANY"):
signature(e1 = "ANY", e2 = "arb"):
binary arithmetic, comparison, and logical operators; see S4groupGeneric. The "other" operand must be atomic or inherit from virtual class flint. Operands are promoted as necessary. Array operands must be conformable (have identical dimensions). Non-array operands are recycled.

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. See also asVector.

backsolve signature(r = "arb", x = "arb"):
signature(r = "arb", x = "ANY"):
signature(r = "ANY", x = "arb"):
solution of the triangular system op2(op1(r)) %*% y = x, where op1=ifelse(upper.tri, triu, tril) and op2=ifelse(transpose, t, identity) and upper.tri and transpose are optional logical arguments with default values TRUE and FALSE, respectively. The "other" operand must be atomic or inherit from virtual class flint. If x is missing, then the return value is the inverse of op2(op1(r)), as if x were the identity matrix. Operands are promoted as necessary and must be conformable (have compatible dimensions). Non-array x are handled as length(x)-by-1 matrices.

chol signature(x = "arb"):
returns the upper triangular Cholesky factor of the positive definite matrix whose upper triangular part is the upper triangular part of x.

chol2inv signature(x = "arb"):
returns the inverse of the positive definite matrix whose upper triangular Cholesky factor is the upper triangular part of x.

coerce signature(from = "ANY", to = "arb"):
returns the value of arb(from).

colSums, colMeans signature(x = "arb"):
returns an arb vector or array containing the column sums or means of x, defined as sums or means over dimensions 1:dims.

det signature(x = "arb"):
returns the determinant of x as an arb vector of length 1.

determinant signature(x = "arb"):
returns a list with components modulus and sign specifying the determinant of x, following

the documented behaviour of the **base** function. The sign is NA if the interval computed by det(x) contains both negative numbers and positive numbers.

diff signature(x = "arb"):
: returns a vector storing lagged differences of the elements of x or (if x is a matrix) a matrix storing lagged differences of the rows of x, following the documented behaviour of the S3 default method.

diffinv signature(x = "arb"):
: returns the vector or matrix y such that x = diff(y, ...), following the documented behaviour of the S3 default method.

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.

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.

rowSums, rowMeans signature(x = "arb"):
: returns an arb vector or array containing the row sums or means of x, defined as sums or means over dimensions (dims+1):length(dim(x)).

solve signature(a = "arb", b = "arb"):
: signature(a = "arb", b = "ANY"):
signature(a = "ANY", b = "arb"):
solution of the general system a %*% x = b. The “other” operand must be atomic or inherit from virtual class flint. If b is missing, then the return value is the inverse of a, as if b were the identity matrix. Operands are promoted as necessary and must be conformable (have compatible dimensions). Non-array b are handled as length(b)-by-1 matrices.

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

Arguments

`x, 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_bessel_j` *Bessel and Related Functions*

Description

Compute the principal branches of the (modified) Bessel functions of the first and second kind. The Bessel functions of the first and second kind solve Bessel's equation

$$z^2 \frac{\mathrm{d}^2 w}{\mathrm{d}z^2} + z \frac{\mathrm{d}w}{\mathrm{d}z} + (z^2 - \nu^2) w = 0$$

and are given by

$$J_\nu(z) = (\tfrac{1}{2} z)^\nu \sum_{k=0}^{\infty} (-1)^k \frac{(\frac{1}{4} z^2)^k}{k! \Gamma(\nu + k + 1)}$$
$$Y_\nu(z) = \frac{Y_\nu(z) \cos(\nu\pi) - J_{-\nu}(z)}{\sin(\nu\pi)}$$

The modified Bessel functions of the first and second kind solve the modified Bessel's equation

$$z^2 \frac{\mathrm{d}^2 w}{\mathrm{d}z^2} + z \frac{\mathrm{d}w}{\mathrm{d}z} - (z^2 + \nu^2) w = 0$$

and are given by

$$I_\nu(z) = (\tfrac{1}{2} z)^\nu \sum_{k=0}^{\infty} \frac{(\frac{1}{4} z^2)^k}{k! \Gamma(\nu + k + 1)}$$
$$K_\nu(z) = \frac{\pi}{2} \frac{I_{-\nu}(z) - I_\nu(z)}{\sin(\nu\pi)}$$

Usage

```
arb_hypgeom_bessel_j(nu, x, prec = flintPrec())
acb_hypgeom_bessel_j(nu, z, prec = flintPrec())

arb_hypgeom_bessel_y(nu, x, prec = flintPrec())
acb_hypgeom_bessel_y(nu, z, prec = flintPrec())

arb_hypgeom_bessel_i(nu, x, prec = flintPrec())
acb_hypgeom_bessel_i(nu, z, prec = flintPrec())

arb_hypgeom_bessel_k(nu, x, prec = flintPrec())
acb_hypgeom_bessel_k(nu, z, prec = flintPrec())
```

Arguments

`nu, x, z`	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/arb_hypgeom.html, https://flintlib.org/doc/acb_hypgeom.html

NIST Digital Library of Mathematical Functions: https://dlmf.nist.gov/10

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` *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, x, 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_integrate *Integration of Functions of One Variable*

Description

Compute an enclosure of the definite integral

$$\int_a^b f(z)\mathrm{d}z$$

taking as the path of integration the line segment from a to b.

Usage

```
arb_integrate(func, a, b, param = NULL, rtol = NULL, atol = NULL,
              control = NULL, prec = flintPrec())
acb_integrate(func, a, b, param = NULL, rtol = NULL, atol = NULL,
              control = NULL, prec = flintPrec())
```

Arguments

func	a function of the form `function (z, param, order, prec)` specifying the integrand. Unused trailing arguments can omitted.
a, b	real or complex numbers or enclosures indicating finite limits of integration.
param	an R object typically specifying parameters of the integrand, passed to `func`.
rtol	a positive real number less than 1 that the relative error in any subinterval should not exceed. `2^-prec` by default.
atol	a positive real number that the absolute error in any subinterval should not exceed. The value 0 indicates that convergence should account only for relative error. `2^-prec` by default.
control	a named list of options for integration.
prec	a positive integer indicating the working precision as a number of bits, passed to `func`.

Details

`func(z, param, order, prec)` computes an enclosure for the integrand on `z`, where `z` is (and the return value of `func` must be) an `arb` or `acb` vector of length 1. If the integer `order` is nonzero, then `func` must give a nonfinite result if the integrand is not holomorphic on `z`, in particular if the integrand composes functions that are bounded on `z` with branch cuts whose intersection with `z` is nonempty.

The list `control` admits components `deg.limit`, `eval.limit`, `depth.limit`, `use.heap`, and `verbose`. These correspond to so-named members of the C struct `acb_calc_integrate_opt_struct`; see the FLINT documentation for details.

Value

An `arb` or `acb` vector of length 1 giving an enclosure of the definite integral.

References

The FLINT documentation of the underlying C function: https://flintlib.org/doc/arb_calc.html

See Also

Classes arb and acb; function integrate in **base**.

Examples

```
## TODO
```

---

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

```
## Class generator functions

arf(x = 0, length = 0L, names = NULL,
    prec = NULL, rnd = NULL)

arf.array(x = 0, dim = length(x), dimnames = NULL,
          prec = NULL, rnd = NULL)
```

Arguments

`x`	an atomic or `flint` vector containing data for conversion to `arf`.
`length`	a numeric vector of length one giving the length of the return value. If that exceeds the length of `x`, then `x` is recycled. Non-integer values are rounded in the direction of zero.
`names`	the `names` slot of the return value, either `NULL` or a character vector of equal length. Non-character `names` are coerced to character.
`dim`	the `dim` slot of the return value, an integer vector of nonzero length. If the product exceeds the length of `x`, then `x` is recycled. Non-integer numeric `dim` are coerced to integer.
`dimnames`	the `dimnames` slot of the return value, either `NULL` or a list of length equal to the length of `dim`. The components are either `NULL` or character vectors of length given by `dim`. Non-character vector components of `dimnames` are coerced to character.
`prec`	the precision used for conversion. `NULL` means to convert exactly if possible and to round to the global default precision otherwise; see `flintPrec`.
`rnd`	the rounding mode used for inexact conversion. `NULL` means to round according to the global default rounding mode; see `flintRnd`.

Details

The class generator function has four distinct usages:

```
arf()
arf(length=)
arf(x)
arf(x, length=)
```

The first usage generates an empty vector. The second usage generates a zero vector of the indicated length. The third usage converts `x`, preserving dimensions, dimension names, and names. The fourth usage converts `x`, recycling its elements to the indicated length and discarding its dimensions, dimension names, and names. Attempts to recycle `x` of length zero to nonzero length are an error.

Usage of `arf.array` is modelled after `array`.

Value

A `arf` vector, possibly an array; see 'Details'.

Conversion

Real numbers and real parts of complex numbers are rounded according to the precision and rounding mode set by `prec` and `rnd`. Imaginary parts of complex numbers are discarded.

Character strings are converted using function `mpfr_strtofr` from the GNU MPFR library with argument `base` set to 0; see https://www.mpfr.org/mpfr-current/mpfr.html#Assignment-Functions.

Slots

`.xData`, `dim`, `dimnames`, `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`.

`%*%`, `crossprod`, `tcrossprod` `signature(x = "arf", y = "arf")`:
`signature(x = "arf", y = "ANY")`:
`signature(x = "ANY", y = "arf")`:
matrix products. The "other" operand must be atomic or inherit from virtual class `flint`. `crossprod` and `tcrossprod` behave as if `y = x` when `y` is missing or `NULL`. Operands are promoted as necessary and must be conformable (have compatible dimensions). Non-array operands of length `k` are handled as 1-by-k or k-by-1 matrices depending on the call. The return value is approximate insofar that it may not be correctly rounded.

`+` `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`.

`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 = "arf", e2 = "arf")`:
: `signature(e1 = "arf", e2 = "ANY")`:
: `signature(e1 = "ANY", e2 = "arf")`:
: binary arithmetic, comparison, and logical operators; see `S4groupGeneric`. The “other” operand must be atomic or inherit from virtual class `flint`. Operands are promoted as necessary. Array operands must be conformable (have identical dimensions). Non-array operands are recycled.

`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. See also `asVector`.

`backsolve` `signature(r = "arf", x = "arf")`:
: `signature(r = "arf", x = "ANY")`:
: `signature(r = "ANY", x = "arf")`:
: solution of the triangular system `op2(op1(r)) %*% y = x`, where `op1=ifelse(upper.tri, triu, tril)` and `op2=ifelse(transpose, t, identity)` and `upper.tri` and `transpose` are optional logical arguments with default values `TRUE` and `FALSE`, respectively. The “other” operand must be atomic or inherit from virtual class `flint`. If `x` is missing, then the return value is the inverse of `op2(op1(r))`, as if `x` were the identity matrix. Operands are promoted as necessary and must be conformable (have compatible dimensions). Non-array `x` are handled as `length(x)`-by-1 matrices.

`chol` `signature(x = "arf")`:
: returns the upper triangular Cholesky factor of the positive definite matrix whose upper triangular part is the upper triangular part of `x`.

`chol2inv` `signature(x = "arf")`:
: returns the inverse of the positive definite matrix whose upper triangular Cholesky factor is the upper triangular part of `x`.

coerce `signature(from = "ANY", to = "arf")`:
: returns the value of `arf(from)`.

colSums, colMeans `signature(x = "arf")`:
: returns an `arf` vector or array containing the column sums or means of `x`, defined as sums or means over dimensions `1:dims`.

det `signature(x = "arf")`:
: returns the determinant of `x` as an `arf` vector of length 1.

determinant `signature(x = "arf")`:
: returns a list with components `modulus` and `sign` specifying the determinant of `x`, following the documented behaviour of the **base** function.

diff `signature(x = "arf")`:
: returns a vector storing lagged differences of the elements of `x` or (if `x` is a matrix) a matrix storing lagged differences of the rows of `x`, following the documented behaviour of the S3 default method.

diffinv `signature(x = "arf")`:
: returns the vector or matrix `y` such that `x = diff(y, ...)`, following the documented behaviour of the S3 default method.

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

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`).

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

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

rowSums, rowMeans `signature(x = "arf")`:
: returns an `arf` vector or array containing the row sums or means of `x`, defined as sums or means over dimensions `(dims+1):length(dim(x))`.

solve `signature(a = "arf", b = "arf")`:
: `signature(a = "arf", b = "ANY")`:
`signature(a = "ANY", b = "arf")`:
solution of the general system `a %*% x = b`. The "other" operand must be atomic or inherit from virtual class flint. If `b` is missing, then the return value is the inverse of `a`, as if `b` were the identity matrix. Operands are promoted as necessary and must be conformable (have compatible dimensions). Non-array `b` are handled as `length(b)`-by-1 matrices.

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")
```

---

arf_rk — *Numerical Solution of Systems of Ordinary Differential Equations*

Description

Solves numerically the initial value problem

$$y'(t) = f(t, y(t)), \quad y(0) = y_0,$$

using an explicit, adaptive or non-adaptive Runge-Kutta method, by default the Dormand-Prince method.

The main difference between this function and function `rk` in package **deSolve** is the use of `arf` instead of `double`, enabling computation of times, solution values, and solution derivatives in arbitrary precision. A corollary is that users can choose arbitrarily small `rtol` and `atol` provided that the working precision is sufficiently high.

Interpolation is not yet implemented.

Usage

```
arf_rk(func, t, y0, param = NULL, rtol = NULL, atol = NULL,
       hmin = 0, hmax = Inf, hini = NULL, smax = NULL,
       method = .rk.method.dormand.prince(), progress = 0L,
       prec = flintPrec())
```

Arguments

`func`	a function of the form `function (t, y, param, prec)` specifying the system. Unused trailing arguments can omitted.
`t`	an increasing numeric or `arf` vector storing time points.
`y0`	a numeric or `arf` vector storing the initial value.
`param`	an R object typically specifying parameters of the system, passed to `func`.

rtol	a positive number less than 1 controlling external step adaptation in adaptive methods. `2^(-prec/2)` by default, unused by non-adaptive methods.
atol	a non-negative number less than 1 controlling external step adaptation in in adaptive methods. `2^(-prec/2)` by default, unused by non-adaptive methods.
hmin	a non-negative number indicating a minimum external step size in adaptive methods. Early termination results if it seems that a smaller step size is needed to achieve sufficiently small error. The default value is `0`, indicating that the step size can become arbitrarily small. Unused by non-adaptive methods.
hmax	a positive number indicating a maximum external step size in adaptive methods. The default value is `Inf`, indicating that the step size is bounded above by `diff(t)` and nothing else. Unused by non-adaptive methods.
hini	a positive number indicating the initial external step size in adaptive methods and the fixed external step size in non-adaptive methods. `min(diff(t))` by default.
smax	a non-negative integer indicating a maximum number of internal steps per external step. Early termination results after `smax * (length(t) - 1)` internal steps. `256 * prec` by default.
method	a list specifying a Runge-Kutta method.
progress	an integer flag determining how progress is indicated. '`.`' is printed after each external step if `progress >= 1`; '`o`' and '`x`' are printed after each accepted and rejected internal step if `progress >= 2`.
prec	a positive integer indicating the working precision as a number of bits, passed to `func`.

Details

`func(t, y, param, prec)` computes the derivative of the solution at time `t` given the value `y` of the solution at time `t` and optional parameters `param`, where `t` is an `arf` vector of length 1 and `y` is (and the return value of `func` must be) an `arf` vector of length equal to `length(y0)`.

The list `method` must have exactly the following components for a `d`-stage method:

- `a` a numeric or `fmpq` or `arf` vector of length `d*(d-1)/2` storing coefficients.
- `b, bb` numeric or `fmpq` or `arf` vectors of length `d` storing lower and higher order weights, each summing to 1. Set `bb` to `NULL` to specify a non-adaptive method.
- `c` a numeric or `fmpq` or `arf` vector of length `d` storing nodes for internal steps. The first element must be 0.
- `p` a positive integer giving the order of the method, such that the global error is $O(h^p)$.

Value

A list with components:

t	an increasing `arf` vector storing time points.
y	an `arf` matrix with `length(t)` rows and `length(y0)` columns storing the numerical solution. In the event of early termination, trailing rows are filled with `NaN`.

See Also

Class arf; function rk in package **deSolve**.

Examples

```
F.linexp <- function (t, y) c(arf(1), -y[2L])
tt <- 0:10
y0 <- c(u = 1, v = 1)
L <- arf_rk(F.linexp, tt, y0)
L. <- list(t = tt, y = cbind(u = y0[1] + tt, v = y0[2] * exp(-tt)))
stopifnot(all.equal(L, L., check.class = FALSE))
```

---

asVector *Coerce an Object to a Vector Class*

Description

asVector is a generalization of as.vector enabling coercion from and to flint vector classes (in addition to basic vector classes) and providing more uniform handling of attributes.

asMatrix and asArray are analogues generalizing as.matrix and as.array.

Usage

```
asVector(x, mode = "any", strict = TRUE)
asMatrix(x, mode = "any", strict = TRUE)
asArray (x, mode = "any", strict = TRUE)
```

Arguments

x	an R object coercible to the target class.
mode	a character string indicating the target class.
strict	a logical indicating if attributes of x should be discarded and if the class of the return value must match the target class exactly (and hence not be a subclass of the target class).

Details

Argument mode can be one of the basic vector classes "raw", "logical", "integer", "numeric" (synonym "double"), "complex", "character", "list", and "expression"; one of the flint vector classes "ulong", "slong", "fmpz", "fmpq", "mag", "arf", "acf", "arb", and "acb"; or one of "any", "vector", and "flint", indicating the vector class, basic vector class, and flint vector class "nearest" the class of x. Note that as.vector supports mode equal to "name" (synonym "symbol") or "pairlist". asVector does not: names and pairlists are not vectors!

Value

The result of coercing x to the target class indicated by mode.

See Also

Virtual class `vector` and related functions `as.vector` and `as`.

Examples

```
str(J  <- diag(ulong(1L), 2L))

as.integer(J)
as.vector(J, "integer")
as(J, "integer")
asVector(J, "integer")
asVector(J, "integer", FALSE)

setClass("ulongExtension", contains = "ulong")
str(J. <- new("ulongExtension", J))

str(asVector(J , "ulong"))
str(asVector(J., "ulong"))
str(asVector(J , "ulong", FALSE))
str(asVector(J., "ulong", FALSE))
```

---

c.flint *Concatenate Vectors*

Description

Function `c` is primitive and internally generic but it 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 a component of `...`.

Functions `cbind` and `rbind` are internally generic and dispatch on all components of `...`, creating the possibility of dispatch ambiguities; see `cbind2` and `rbind2`.

S3 methods `c.flint`, `cbind.flint` and `rbind.flint` are registered *and exported* to enable users to bypass internal dispatch.

Usage

```
## S3 method for class 'flint'
c(..., recursive = FALSE, use.names = TRUE)
## S3 method for class 'flint'
cbind(..., deparse.level = 1)
## S3 method for class 'flint'
rbind(..., deparse.level = 1)
```

Arguments

`...`	objects inheriting from virtual class `flint` or whose type is one of the vector types or one of the non-vector types `NULL`, `pairlist`, `symbol`, and `language`.

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.
deparse.level	an integer (0, 1, or 2) indicating how names are chosen for rows or columns derived from untagged, non-matrix arguments. 0 is to use empty names, 2 is to deparse unevaluated arguments, and 1 (the default value) is to deparse unevaluated arguments only if they are symbols and otherwise use empty names.

Value

If none of the arguments is a flint vector, then the internal default methods are 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 pairlist, name, call, list, or expression, in which case the return value is a list or expression.

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.

Examples

```
x <- slong(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.

Usage

```
## Class generator functions

flint(class, ...)

flint.array(class, ...)
```

Arguments

`class`	a character string giving the name of a nonvirtual subclass of `flint`, one of `"ulong"`, `"slong"`, `"fmpz"`, `"fmpq"`, `"mag"`, `"arf"`, `"acf"`, `"arb"`, and `"acb"`.
`...`	arguments passed to the class generator function corresponding to `class`.

Value

An object of class `class` generated by the corresponding class generator function. For example, `flint("ulong", ...)` returns `ulong(...)` and `flint.array("slong", ...)` returns `slong.array(...)`.

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.

dim either NULL, indicating that the object is not an array, or an integer vector of length d greater than 0 and with product equal to the object length, indicating that the object is a d-dimensional array with dimensions dim. Array entries are stored in colexicographic order, meaning that the first subscript moves fastest.

dimnames either NULL, indicating that the object is not an array or is an array whose dimensions are not named, or a list of length d equal to length(dim) such that dimnames[[i]] is either NULL or a character vector of length dim[[i]], for all i in 1L:d.

names either NULL, indicating that the object is not named, or a character vector 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 = "flint"):
signals an error as x is "atomic-like" and in any case not recursive or NULL.

[signature(x = "flint", i = "ANY", j = "ANY"):
signature(x = "ANY", i = "flint", j = "ANY"):
signature(x = "ANY", i = "ANY", j = "flint"):
returns a traditional vector or flint vector containing the elements of x indexed by (i, j, ...) (the "subscript"). The components of 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 one option is to call [as `[`(i = i, x = x) rather than as x[i]. However, it is not guaranteed that such usage of [, which is mostly undocumented, continues to work in future versions of R.

[<- signature(x = "flint", i = "ANY", j = "ANY", value = "ANY"):
signature(x = "ANY", i = "flint", j = "ANY", value = "ANY"):
signature(x = "ANY", i = "ANY", j = "flint", value = "ANY"):
signature(x = "ANY", i = "ANY", j = "ANY", value = "flint"):
returns the traditional vector or flint vector obtained by replacing the elements of x indexed by (i, j, ...) (the "subscript") with elements of value, which are recycled as necessary. The components of 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 one option is to call [<- as `[`(i = i, x = x) <- value rather than as x[i] <- value. However, it is not guaranteed that such usage of [<-, which is mostly undocumented, continues to work in future versions of R.

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

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

all.equal signature(x = "flint", y = "flint"):
signature(x = "flint", y = "ANY"):
signature(x = "ANY", 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 the method for identical for much stricter comparison of flint objects.

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

aperm signature(a = "flint"):
returns the array obtained by permuting the dimensions of a according to a second argument perm, following the documented behaviour of the S3 default method.

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 are 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 as.numeric, section 'S4 methods'.

as.matrix, as.array, as.Date, as.POSIXct, as.POSIXlt signature(x = "flint"):
coerces the argument with as.vector, restores dimensions, dimension names, and names, and dispatches. as.matrix and as.array obtain the same result more efficiently.

as.data.frame signature(x = "flint"):
behaves as as.data.frame.vector, as.data.frame.matrix, or as.data.frame.array, depending on the length of the dim slot. 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.

asplit signature(x = "flint"):
returns a list array containing the marginal splits of x indicated by a second argument MARGIN, following the documented behaviour of the **base** function.

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.

cbind2 signature(x = "flint", y = "flint"):
signature(x = "flint", y = "ANY"):
signature(x = "ANY", y = "flint"):
returns cbind.flint(x, y, ...), the horizontal concatenation of x and y. These methods are dispatched by cbind in case of S3 dispatch ambiguities.

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 S3 default method with argument labels set to FALSE, provided that breaks is sorted and no element of x is out of bounds.

diag signature(x = "flint"):
if x is a matrix, then returns a flint vector containing the diagonal entries of x; otherwise, returns a diagonal matrix with diagonal entries taken from x. Optional arguments nrow, ncol, and names are handled as by the **base** function.

diag<- signature(x = "flint", value = "ANY"):
returns x, which must be a matrix, after setting its main diagonal to value, whose length must be equal to 1 or the length of x. Arguments x and value are coerced to a common class following the rules used for general subassignment; see the methods for [<- and [[<-.

dim signature(x = "flint"):
returns the dim slot of x.

dim<- signature(x = "flint", value = "NULL"):
returns x with dim and dimnames slots set to NULL.

dim<- signature(x = "flint", value = "numeric"):
returns x with dim slot set to value and dimnames slot set to NULL. value of double type is coerced to integer.

dimnames signature(x = "flint"):
returns the dimnames slot of x.

dimnames<- signature(x = "flint", value = "NULL"):
returns x with dimnames slot set to NULL.

dimnames<- signature(x = "flint", value = "list"):
returns x with dimnames slot set to value. Elements of value of a vector type are coerced to character using as.character.default. Exceptionally, factors are coerced to character using as.character.factor.

drop signature(x = "flint"):
returns x with dim, dimnames, and names slots modified, following the documented behaviour of the **base** function.

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. A caveat is that an error is signaled if x contains NaN, because ulong has no representation for R's missing value NA_integer_.

identical signature(x = "flint", y = "flint"):
returns a logical indicating if x and y are "exactly equal". Compared to the default method (which is the **base** function), this method handles the .xData slots of x and y specially: by default (if extptr.as.ref is FALSE), it does not test for equality of the stored pointers but rather for entrywise equality of the *pointed to* arrays. Hence by default the .xData slots are compared as if they were traditional numeric or complex vectors.

is.array signature(x = "flint"):
returns a logical indicating if x has a non-NULL dim slot.

is.matrix signature(x = "flint"):
returns a logical indicating if x has a dim slot of length 2.

`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`.

`isSymmetric` `signature(x = "flint")`:
: returns a logical indicating if `x` is a Hermitian matrix or if `x` is a symmetric matrix, depending on optional argument `trans`, following the documented behaviour of the S3 method for traditional matrices.

`kronecker` `signature(X = "flint", Y = "flint")`:
: `signature(X = "flint", Y = "ANY")`:
`signature(X = "ANY", Y = "flint")`:
these methods are copies of the **base** function `.kronecker` with calls to `as.array` substituted for calls to `asArray`, as only `asArray` preserves `flint` subclass inheritance.

`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 = "flint", table = "flint")`:
: `signature(x = "flint", table = "ANY")`:
`signature(x = "ANY", 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 **base** function.

`mtfrm` `signature(x = "flint")`:
: returns `format(x, base = 62L, digits = 0L, digits.mag = 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.

`names<-` `signature(x = "flint", value = "NULL")`:
: returns `x` with `names` slot set to `NULL`.

`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.

`norm` `signature(x = "flint")`:
: returns the matrix norm of `x` as a `flint` vector of length 1. The class of the return value can depend on the norm type indicated by argument `type`; see `norm`.

`outer` `signature(X = "flint", Y = "flint")`:
: `signature(X = "flint", Y = "ANY")`:
`signature(X = "ANY", Y = "flint")`:
these methods are copies of the **base** function `outer` with calls to `as.vector` substituted for calls to `asVector`, as only `asVector` preserves `flint` subclass inheritance.

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

rbind2 signature(x = "flint", y = "flint"):
signature(x = "flint", y = "ANY"):
signature(x = "ANY", y = "flint"):
returns rbind.flint(x, y, ...), the vertical concatenation of x and y. These methods are dispatched by rbind in case of S3 dispatch ambiguities.

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. 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.

scale signature(x = "flint"):
returns the result of optionally centering and optionally scaling the columns of x, following the documented behaviour of the S3 default method.

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.

t signature(x = "flint"):
returns the transpose of x if x is a matrix, handling non-array x as length(x)-by-1 matrices.

unique signature(x = "flint"):
if x is not an array or MARGIN is empty, returns a vector containing the unique elements of x; otherwise, returns an array containing the unique splits of x by margin MARGIN. Elements (splits) of x are considered distinct if the corresponding elements (splits) of mtfrm(x) are not identical.

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, 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

```
## Class generator functions

fmpq(x = 0L, length = 0L, names = NULL, num = 0L, den = 1L)

fmpq.array(x = 0L, dim = length(x), dimnames = NULL, num = 0L, den = 1L)
```

Arguments

x	an atomic or flint vector containing data for conversion to fmpq.
length	a numeric vector of length one giving the length of the return value. If that exceeds the length of x, then x is recycled. Non-integer values are rounded in the direction of zero.
names	the names slot of the return value, either NULL or a character vector of equal length. Non-character names are coerced to character.
dim	the dim slot of the return value, an integer vector of nonzero length. If the product exceeds the length of x, then x is recycled. Non-integer numeric dim are coerced to integer.
dimnames	the dimnames slot of the return value, either NULL or a list of length equal to the length of dim. The components are either NULL or character vectors of length given by dim. Non-character vector components of dimnames are coerced to character.
num, den	atomic or flint vectors containing data for conversion to fmpz. Use these instead of x for initialization "by parts" (numerator and denominator).

Details

The class generator function has six distinct usages:

```
fmpq()
fmpq(length=)
fmpq(x)
fmpq(x, length=)
fmpq(num=, den=)
fmpq(num=, den=, length=)
```

The first usage generates an empty vector. The second usage generates a zero vector of the indicated length. The third usage converts x, preserving dimensions, dimension names, and names. The fourth usage converts x, recycling its elements to the indicated length and discarding its dimensions, dimension names, and names. The fifth and sixth usages, in which either of num and den can be missing, use fmpz(num) and fmpz(den) to separately initialize the numerators and denominators of the fmpq return value.

Attempts to recycle num, den, or x of length zero to nonzero length are an error.

Usage of fmpq.array is modelled after array.

Value

An fmpq vector, possibly an array; see 'Details'.

Conversion

Real numbers and real parts of complex numbers are converted exactly, as floating-point numbers are rational by definition. Imaginary parts of complex numbers are discarded.

Character strings are converted using function mpq_set_str from the GNU MP library with argument base set to 0; see https://gmplib.org/manual/Initializing-Rationals.

An error is signaled if elements of num, den, or x are NaN, -Inf, or Inf or if elements of den are 0.

Slots

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

Methods

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

`%*%`, `crossprod`, `tcrossprod` `signature(x = "fmpq", y = "fmpq")`:
: `signature(x = "fmpq", y = "ANY")`:
`signature(x = "ANY", y = "fmpq")`:
matrix products. The "other" operand must be atomic or inherit from virtual class `flint`. `crossprod` and `tcrossprod` behave as if `y = x` when `y` is missing or `NULL`. Operands are promoted as necessary and must be conformable (have compatible dimensions). Non-array operands of length `k` are handled as 1-by-`k` or `k`-by-1 matrices depending on the call.

`+` `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 = "fmpq", e2 = "fmpq")`:
: `signature(e1 = "fmpq", e2 = "ANY")`:
`signature(e1 = "ANY", e2 = "fmpq")`:
binary arithmetic, comparison, and logical operators; see `S4groupGeneric`. The "other" operand must be atomic or inherit from virtual class `flint`. Operands are promoted as necessary. Array operands must be conformable (have identical dimensions). Non-array operands are recycled.

`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. See also asVector.

backsolve signature(r = "fmpq", x = "fmpq"):
: signature(r = "fmpq", x = "ANY"):
signature(r = "ANY", x = "fmpq"):
solution of the triangular system op2(op1(r)) %*% y = x, where op1=ifelse(upper.tri, triu, tril) and op2=ifelse(transpose, t, identity) and upper.tri and transpose are optional logical arguments with default values TRUE and FALSE, respectively. The “other” operand must be atomic or inherit from virtual class flint. If x is missing, then the return value is the inverse of op2(op1(r)), as if x were the identity matrix. Operands are promoted as necessary and must be conformable (have compatible dimensions). Non-array x are handled as length(x)-by-1 matrices. If r and (if not missing) x are both formally rational, then the solution is exact and the return value is an fmpq matrix.

chol signature(x = "fmpq"):
: coerces x to class arf and dispatches.

chol2inv signature(x = "fmpq"):
: returns the inverse of the positive definite matrix whose upper triangular Cholesky factor is the upper triangular part of x. The return value is the exact inverse, being computed as tcrossprod(backsolve(x)).

coerce signature(from = "ANY", to = "fmpq"):
: returns the value of fmpq(from).

colSums, colMeans signature(x = "fmpq"):
: returns an fmpq vector or array containing the column sums or means of x, defined as sums or means over dimensions 1:dims.

colSums signature(x = "fmpq"):
: returns an fmpq vector or array containing the column sums of x, defined as sums over dimensions 1:dims.

colMeans signature(x = "fmpq"):
: returns an fmpq vector or array containing the column means of x, defined as means over dimensions 1:dims.

det signature(x = "fmpq"):
: returns the determinant of x as an fmpq vector of length 1.

determinant signature(x = "fmpq"):
: returns a list with components modulus and sign specifying the determinant of x, following the documented behaviour of the **base** function. Note that det(x) and determinant(x, logarithm = FALSE) are exact, but determinant(x) is not in general due to rounding.

diff signature(x = "fmpq"):
: returns a vector storing lagged differences of the elements of x or (if x is a matrix) a matrix storing lagged differences of the rows of x, following the documented behaviour of the S3 default method.

diffinv signature(x = "fmpq"):
: returns the vector or matrix y such that x = diff(y, ...), following the documented behaviour of the S3 default method.

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

`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.

`rowSums`, `rowMeans` `signature(x = "fmpq")`:
: returns an `fmpq` vector or array containing the row sums or means of `x`, defined as sums or means over dimensions `(dims+1):length(dim(x))`.

`solve` `signature(a = "fmpq", b = "fmpq")`:
: `signature(a = "fmpq", b = "ANY")`:
: `signature(a = "ANY", b = "fmpq")`:
: solution of the general system `a %*% x = b`. The “other” operand must be atomic or inherit from virtual class `flint`. If `b` is missing, then the return value is the inverse of `a`, as if `b` were the identity matrix. Operands are promoted as necessary and must be conformable (have compatible dimensions). Non-array `b` are handled as `length(b)`-by-1 matrices. If `a` and (if not missing) `b` are both formally rational, then the solution is exact and the return value is an `fmpq` matrix.

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")

## Only the canonical representation is stored
(h <- fmpq(num = c(0L, 2L), den = -4L))
stopifnot(identical(Num(h), fmpz(c(0L, -1L))),
          identical(Den(h), fmpz(c(1L,  2L))))
try(fmpq(num = -1L:1L, den = 0L)) # canonical => nonzero denominator

## All floating-point numbers are rational
(xm <- fmpq(xm. <- exp(-100)))
(xp <- fmpq(xp. <- exp( 100)))
stopifnot(identical(xp. + xm. - xp.,  0), # floating-point arithmetic
```

```
         identical(xp  + xm  - xp , xm)) # exact rational arithmetic

## Exactness isn't free: higher precision => bigger allocations
x3 <- c(xm, xp, xm + xp)
flintBits(Num(x3))
flintBits(Den(x3))

## Conversion of "p/q" format strings
(pq <- fmpq(c("0/1", " -1/2", "2 /3", "-3/ 4", "4/5 ", "  -5  /  6  ")))
stopifnot(identical(fmpq(format(pq)), pq)) # always

## Conversion to "double" rounds towards zero
(z <- 1L - fmpz(2L)^-128L)
stopifnot(as.double(z) < 1)
as.double(z) == 1 - .Machine$double.neg.eps # typically

## Conversion to "arf" depends on precision and rounding mode
c4 <- c(fmpq(arf(z, prec = 128L, rnd = "Z")),
        fmpq(arf(z, prec = 127L, rnd = "Z")),
        fmpq(arf(z, prec = 128L, rnd = "A")),
        fmpq(arf(z, prec = 127L, rnd = "A")))

fmpq.array(z - c4, c(2L, 2L), list(c("128", "127"), c("Z", "A")))
```

`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

```
## Class generator functions

fmpz(x = 0L, length = 0L, names = NULL)

fmpz.array(x = 0L, dim = length(x), dimnames = NULL)
```

Arguments

`x`	an atomic or `flint` vector containing data for conversion to `fmpz`.
`length`	a numeric vector of length one giving the length of the return value. If that exceeds the length of `x`, then `x` is recycled. Non-integer values are rounded in the direction of zero.
`names`	the `names` slot of the return value, either `NULL` or a character vector of equal length. Non-character `names` are coerced to character.

`dim`	the `dim` slot of the return value, an integer vector of nonzero length. If the product exceeds the length of `x`, then `x` is recycled. Non-integer numeric `dim` are coerced to integer.
`dimnames`	the `dimnames` slot of the return value, either `NULL` or a list of length equal to the length of `dim`. The components are either `NULL` or character vectors of length given by `dim`. Non-character vector components of `dimnames` are coerced to character.

Details

The class generator function has four distinct usages:

```
fmpz()
fmpz(length=)
fmpz(x)
fmpz(x, length=)
```

The first usage generates an empty vector. The second usage generates a zero vector of the indicated length. The third usage converts `x`, preserving dimensions, dimension names, and names. The fourth usage converts `x`, recycling its elements to the indicated length and discarding its dimensions, dimension names, and names. Attempts to recycle `x` of length zero to nonzero length are an error.

Usage of `fmpz.array` is modelled after `array`.

Value

An `fmpz` vector, possibly an array; see 'Details'.

Conversion

Real numbers and real parts of complex numbers are rounded in the direction of 0. Imaginary parts of complex numbers are discarded.

Character strings are converted using function `mpz_set_str` from the GNU MP library with argument `base` set to 0; see `https://gmplib.org/manual/Assigning-Integers`.

An error is signaled if elements of `x` are `NaN`, `-Inf`, or `Inf`.

Slots

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

Methods

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

`%*%`, `crossprod`, `tcrossprod` `signature(x = "fmpz", y = "fmpz")`:
: `signature(x = "fmpz", y = "ANY")`:
`signature(x = "ANY", y = "fmpz")`:
matrix products. The "other" operand must be atomic or inherit from virtual class `flint`. `crossprod` and `tcrossprod` behave as if `y = x` when `y` is missing or `NULL`. Operands are

promoted as necessary and must be conformable (have compatible dimensions). Non-array operands of length k are handled as 1-by-k or k-by-1 matrices depending on the call.

`+` `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 = "fmpz", e2 = "fmpz")`:
`signature(e1 = "fmpz", e2 = "ANY")`:
`signature(e1 = "ANY", e2 = "fmpz")`:
binary arithmetic, comparison, and logical operators; see `S4groupGeneric`. The "other" operand must be atomic or inherit from virtual class `flint`. Operands are promoted as necessary. Array operands must be conformable (have identical dimensions). Non-array operands are recycled.

`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. See also `asVector`.

`backsolve` `signature(r = "fmpz", x = "fmpz")`:
`signature(r = "fmpz", x = "ANY")`:
`signature(r = "ANY", x = "fmpz")`:
solution of the triangular system `op2(op1(r)) %*% y = x`, where `op1=ifelse(upper.tri, triu, tril)` and `op2=ifelse(transpose, t, identity)` and `upper.tri` and `transpose` are optional logical arguments with default values `TRUE` and `FALSE`, respectively. The "other" operand must be atomic or inherit from virtual class `flint`. If `x` is missing, then the return value is the inverse of `op2(op1(r))`, as if `x` were the identity matrix. Operands are promoted as necessary and must be conformable (have compatible dimensions). Non-array `x` are handled as `length(x)`-by-1 matrices. If `r` and (if not missing) `x` are both formally rational, then the solution is exact and the return value is an `fmpq` matrix.

`chol` `signature(x = "fmpz")`:
: coerces `x` to class `arf` and dispatches.

`chol2inv` `signature(x = "fmpz")`:
: returns the inverse of the positive definite matrix whose upper triangular Cholesky factor is the upper triangular part of `x`. The return value is the exact inverse, being computed as `tcrossprod(backsolve(x))`.

`coerce` `signature(from = "ANY", to = "fmpz")`:
: returns the value of `fmpz(from)`.

`colSums` `signature(x = "fmpz")`:
: returns an `fmpz` vector or array containing the column sums of `x`, defined as sums over dimensions `1:dims`.

`colMeans` `signature(x = "fmpz")`:
: returns an `fmpq` vector or array containing the column means of `x`, defined as means over dimensions `1:dims`.

`det` `signature(x = "fmpz")`:
: returns the determinant of `x` as an `fmpz` vector of length 1.

`determinant` `signature(x = "fmpz")`:
: returns a list with components `modulus` and `sign` specifying the determinant of `x`, following the documented behaviour of the **base** function. Note that `det(x)` and `determinant(x, logarithm = FALSE)` are exact, but `determinant(x)` is not in general due to rounding.

`diff` `signature(x = "fmpz")`:
: returns a vector storing lagged differences of the elements of `x` or (if `x` is a matrix) a matrix storing lagged differences of the rows of `x`, following the documented behaviour of the S3 default method.

`diffinv` `signature(x = "fmpz")`:
: returns the vector or matrix `y` such that `x = diff(y, ...)`, following the documented behaviour of the S3 default method.

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

`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.

`rowSums` `signature(x = "fmpz")`:
: returns an `fmpz` vector or array containing the row sums of `x`, defined as sums over dimensions `(dims+1):length(dim(x))`.

rowMeans signature(x = "fmpz"):
: returns an fmpq vector or array containing the row means of x, defined as means over dimensions (dims+1):length(dim(x)).

solve signature(a = "fmpz", b = "fmpz"):
: signature(a = "fmpz", b = "ANY"):
signature(a = "ANY", b = "fmpz"):
solution of the general system a %*% x = b. The "other" operand must be atomic or inherit from virtual class flint. If b is missing, then the return value is the inverse of a, as if b were the identity matrix. Operands are promoted as necessary and must be conformable (have compatible dimensions). Non-array b are handled as length(b)-by-1 matrices. If a and (if not missing) b are both formally rational, then the solution is exact and the return value is an fmpq matrix.

References

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

See Also

Virtual class flint and its nonvirtual subclass fmpq, the latter representing rational numbers as constrained pairs of fmpz.

Examples

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

    fmpz(0x1p+1023)  # with(.Machine, 2L^(double.max.exp-1L))
try(fmpz(0x1p+1024)) # no representation for Inf after
                     # floating-point overflow

    fmpz(2L)^1024L  # powers are rational in general (signed exponent)
Num(fmpz(2L)^1024L) # get the numerator

## Allocation in bytes as a function of the most significant bit (B)
##     B <= M - 2: a limb
##     B >  M - 2: a limb, an 'mpz' struct, and N more limbs
## where M := flintABI(), N := floor((B - 1)/M) + 1
B <- seq_len(1024L)
rle(vapply(as.list(Num(fmpz(2L)^(B - 1L))), flintSize, 0))

## Conversion of decimal format strings
fmpz(       "1234567890"        )
fmpz(strrep("1234567890", 1L:6L))

## Conversion of hexadecimal format strings
hs <- paste0("-0x1", strrep("0", 256L))
hz <- fmpz(hs)
stopifnot(identical(hz, Num(-fmpz(2L)^1024L)),
          identical(format(hz, base = 16L), sub("0x", "", hs)))
```

```
## Exact 'sqrt' preserves class, hence requires perfect squares
    sqrt(fmpz(81L))  # ok
try(sqrt(fmpz(80L))) # error
sqrt(arf(fmpz(80L))) # ok, thanks to coercion to floating-point type

## Quotients are formally rational
(J <- fmpz.array(0L:11L, c(6L, 2L), list(NULL, col = c("aa", "bb"))))
J/2L
J/1L
try(J/0L) # NaN, -Inf, and Inf have no representation
rowMeans(J)
colMeans(J)
summary(J)
summary(J, quantile.type = 6L) # types 1 through 9 are all implemented

## Floored integer division
p <- fmpz(-127L)
q <- c(-(3L:1L), 1L:3L)
stopifnot(identical(p %/% q * q + p %% q, rep(p, 6L)))

## Exact rational solution of linear systems with integer coefficients
(A3 <- diag(c(0x1p+53, 1, 1)))
try(solve(A3)) # system is computationally singular
(A3 <- diag(Num(fmpz(2L)^c(53L, 0L, 0L))))
(A3inv <- solve(A3))
(I3 <- diag(1L, 3L, 3L))
(b3 <- -1L:1L)
stopifnot(identical(solve(A3, I3), A3inv),
          identical(solve(A3, b3), A3inv %*% b3),
          identical(solve(A3inv), fmpq(A3)),
          identical(A3 %*% A3inv, fmpq(I3)))

## Conversion to "double" rounds towards zero
(z <- Num(fmpz(2L)^54L))
(off <- fmpz(0L:8L))
(offZ <- 4L * (off %/% 4L))
stopifnot(identical(fmpz(as.double(z + off)) - z, offZ))

## Conversion to "arf" is exact *by default*
stopifnot(identical(fmpz(arf(z + off)) - z, off),
          identical(fmpz(arf(z + off, prec = 53L, rnd = "Z")) - z, offZ))
```

`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, sep = NULL,
       digits.mag = NULL, rnd.mag = NULL, ...)
## S4 method for signature 'arf'
format(x, base = 10L, sep = NULL,
       digits = NULL, rnd = NULL, ...)
## S4 method for signature 'acf'
format(x, base = 10L, sep = NULL,
       digits = NULL, rnd = NULL, ...)
## S4 method for signature 'arb'
format(x, base = 10L, sep = NULL,
       digits = NULL, digits.mag = NULL,
       rnd = NULL, rnd.mag = "A", ...)
## S4 method for signature 'arb'
format(x, base = 10L, sep = NULL,
       digits = NULL, digits.mag = NULL,
       rnd = NULL, rnd.mag = "A", ...)
```

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.

`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.

`digits`, `digits.mag` integers indicating how many digits of the significand are reported when formatting floating-point numbers. When more than one digit is printed, a radix point is 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 values `NULL` are equivalent to `getOption("digits")` and `getOption("digits.mag", 4L)`.

`rnd`, `rnd.mag` character strings indicating the rounding modes used when formatting floating-point numbers. The default values `NULL` are equivalent to `flintRnd()` and `flintRndMag()`; see there for a description of valid character strings and the corresponding rounding modes.

`...` further optional arguments, though these are currently unused.

Details

Formatting of `arf` and `arf` midpoints of `acf`, `arb`, and `acb` uses arguments `digits` and `rnd`. Formatting of `mag` and `mag` radii of `arb` and `acb` uses arguments `digits.mag` and `rnd.mag`.

Note that radii are *not* incremented to account for error introduced by rounding of midpoints. Hence it is possible that the enclosure obtained by formatting does not contain the enclosure represented in memory.

Value

A character vector containing ASCII strings of equal length, preserving the length, dimensions, dimension names, and names of `x`.

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

```
## Class generator functions

mag(x = 0, length = 0L, names = NULL, rnd.mag = NULL)

mag.array(x = 0, dim = length(x), dimnames = NULL, rnd.mag = NULL)
```

Arguments

`x`	an atomic or `flint` vector containing data for conversion to `mag`.
`length`	a numeric vector of length one giving the length of the return value. If that exceeds the length of `x`, then `x` is recycled. Non-integer values are rounded in the direction of zero.
`names`	the `names` slot of the return value, either `NULL` or a character vector of equal length. Non-character `names` are coerced to character.
`dim`	the `dim` slot of the return value, an integer vector of nonzero length. If the product exceeds the length of `x`, then `x` is recycled. Non-integer numeric `dim` are coerced to integer.
`dimnames`	the `dimnames` slot of the return value, either `NULL` or a list of length equal to the length of `dim`. The components are either `NULL` or character vectors of length given by `dim`. Non-character vector components of `dimnames` are coerced to character.
`rnd.mag`	the rounding mode used for inexact conversion. `NULL` means to round according to the global default rounding mode; see `flintRndMag`.

Details

The class generator function has four distinct usages:

```
mag()
mag(length=)
mag(x)
mag(x, length=)
```

The first usage generates an empty vector. The second usage generates a zero vector of the indicated length. The third usage converts `x`, preserving dimensions, dimension names, and names. The fourth usage converts `x`, recycling its elements to the indicated length and discarding its dimensions, dimension names, and names. Attempts to recycle `x` of length zero to nonzero length are an error.

Usage of `mag.array` is modelled after `array`.

Value

A `mag` vector, possibly an array; see 'Details'.

Conversion

Magnitudes of real numbers and real parts of complex numbers are rounded towards or away from zero according to the rounding mode set by `rnd.mag`. Imaginary parts of complex numbers are discarded.

It is guaranteed that the result of conversion is a lower or upper bound on the converted value. It is not guaranteed that the bound is optimal; in particular, the result of conversion can be inexact even if the converted value is exactly representable. Indeed, the computed bound and the optimal bound can differ by several ulps. If that seems unusual, then note that `mag` exists primarily to represent the radii of `arb` and `acb`, and arithmetic involving `arb` or `acb` benefits from fast and "precise enough" operations on the radii.

Character strings are converted using function `mpfr_strtofr` from the GNU MPFR library with argument base set to 0; see https://www.mpfr.org/mpfr-current/mpfr.html#Assignment-Functions.

An error is signaled if elements of `x` are `NaN`.

Slots

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

Methods

Methods that return a `mag` vector are below marked by an asterisk '`*`' after `signature(...)`. Where the generic function is defined as evaluating a real-valued mathematical function F, these methods compute lower or upper bounds on $|F|$. In this sense, the marked methods can be seen as violating the generic function's "contract". For contract-adhering behaviour, dispatch methods for `arf`, e.g., do `log(arf(x))` instead of `log(x)` for `x` of class `mag`. The bounds computed by the marked methods are not optimal in general; see 'Conversion'. Whether lower (as opposed to upper) bounds are computed depends on the global default rounding mode; see `flintRndMag`.

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

`%*%`, `crossprod`, `tcrossprod` `signature(x = "mag", y = "mag")`:
: `signature(x = "mag", y = "ANY")`:
`signature(x = "ANY", y = "mag")`:
coerces the `mag` operand to class `arf`, `acf`, `arb`, or `acb` (depending on the class of the other operand) and dispatches.

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

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

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

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

`Ops` `signature(e1 = "mag", e2 = "mag")*`:
: `signature(e1 = "mag", e2 = "ANY")`:
`signature(e1 = "ANY", e2 = "mag")`:
binary arithmetic, comparison, and logical operators; see `S4groupGeneric`. The "other" operand must be atomic or inherit from virtual class `flint`. Operands are promoted as necessary. Array operands must be conformable (have identical dimensions). Non-array operands are recycled. For arithmetic, the return value is a `mag` vector if and only if both operands are `mag` vectors.

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

`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 away from zero though not always to the nearest greater number. Coercion to types "character", "symbol" (synonym "name"), "pairlist", "list", and "expression", which are not "number-like", is handled specially. See also asVector.

backsolve signature(r = "mag", x = "mag"):
signature(r = "mag", x = "ANY"):
signature(r = "ANY", x = "mag"):
coerces the mag operand to class arf, acf, arb, or acb (depending on the class of the other operand) and dispatches.

chol, chol2inv signature(x = "mag"):
coerces x to class arf and dispatches.

coerce signature(from = "ANY", to = "mag")*:
returns the value of mag(from).

colSums, colMeans signature(x = "mag")*:
returns a mag vector or array containing the column sums or means of x, defined as sums or means over dimensions 1:dims.

det, determinant, diff, diffinv signature(x = "mag"):
coerces x to class arf and dispatches.

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

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 the logarithm of the argument. The natural logarithm is computed by default (when optional argument base is unset).

mean signature(x = "mag")*:
returns the arithmetic mean.

rowSums, rowMeans signature(x = "mag")*:
returns a mag vector or array containing the row sums or means of x, defined as sums or means over dimensions (dims+1):length(dim(x)).

solve signature(a = "mag", b = "mag"):
signature(a = "mag", b = "ANY"):
signature(a = "ANY", b = "mag"):
coerces the mag operand to class arf, acf, arb, or acb (depending on the class of the other operand) and dispatches.

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")

(ornd <- flintRndMag()) # getting the original rounding mode

## Output gives 4 significant digits by default
(magpi <- mag(pi))

## Number of reliable digits is 8 == floor((30 - 1) * log10(2))
as.character(magpi)

## Integers in range of 30-bit unsigned type are converted exactly
(x0 <- -1L:1L * (0x1p+30L - 1L))
mag(x0) == abs(x0) #     all TRUE
(x1 <- -1L:1L * (0x1p+30L - 0L))
mag(x1) == abs(x1) # not all TRUE

## Conversion of most other input is influenced by the rounding mode
flintRndMag("A")
mag( 1) >  1
mag(pi) > pi
flintRndMag("Z")
mag( 1) <  1
mag(pi) < pi

## Computing bounds on composite functions needs care;
## e.g., for an upper bound on abs(tan(2)) = sin(2)/abs(cos(2)):
tt <- mag(2L) # exact
flintRndMag("Z") # some quantities must be rounded towards 0
cos2 <- cos(tt)
flintRndMag("A") # others must be rounded away from 0
sin2 <- sin(tt)
tan2 <- sin2/cos2
tan2    - abs(tan(2))
tan(tt) - abs(tan(2)) # direct hence sharper

flintRndMag(ornd) # resetting the original rounding mode
```

OptionalCharacter-class — *Unions of 'NULL' and Vector Classes*

Description

Class unions in the style of `OptionalFunction` from package **methods**, whose purpose is to allow slots `dim`, `dimnames`, and `names` of virtual class `flint` to be `NULL` or a vector of suitable type.

`OptionalInteger`, `OptionalList`, and `OptionalCharacter` are the unions of `NULL` and `integer`, `list`, and `character`, respectively.

Examples

```
showClass("OptionalInteger")
showClass("OptionalList")
(oc <- getClass("OptionalCharacter"))

stopifnot(isVirtualClass(oc),
          isClassUnion(oc),
          all(c("NULL", "character") %in% names(oc@subclasses)),
          any(extends("NULL") == "OptionalCharacter"),
          any(extends("character") == "OptionalCharacter"))

getClass("flint")@slots
```

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, of length equal to 1 or the length of the argument.

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).

For convenience, Mid and its replacement form also work for acb x, getting and setting the complex midpoint defined by the midpoints of the real and imaginary parts of x.

Value

Num, Den, Mid, Rad, Real, and Imag and their replacement forms return a vector-like R object preserving the length, dimensions, dimension names, and names of the argument. 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)
```

TypeClass *Test What is Representable by a Type or Class*

Description

`isSigned` tests if the type or class of the argument can represent nonzero numbers with sign (value divided by modulus) not equal to 1.

`isComplex` tests if the type or class of the argument can represent complex numbers with nonzero imaginary part.

`isFloating` tests if the type or class of the argument represents floating-point real or complex numbers.

Usage

```
isSigned(x)
isComplex(x)
isFloating(x)
```

Arguments

x	a vector-like R object representing numbers.

Details

`isSigned(x)` equal to `FALSE` implies that `sign(x)` is all 0 or 1. The converse is not true in general.

`isComplex(x)` equal to `FALSE` implies that `Imag(x)` (equivalently `Im(x)`) is all 0. The converse is not true in general.

For `x` of class `arb` or `acb`, methods inherit behaviour from the class of the midpoint, returning the value of `is*(Mid(x))`.

Value

A logical, either `TRUE` or `FALSE`.

See Also

Virtual class `flint`.

Examples

```
try(isSigned(NULL)) # an error if 'x' does not represent numbers

L <- sapply(c(if (getRversion() >= "4.5") "raw", # for as(NULL, "raw")
              "logical", "integer", "double", "complex",
              "ulong", "slong", "fmpz", "fmpq", "mag", "arf", "acf",
              "arb", "acb"),
            as, object = NULL, simplify = FALSE)
F <- function (x) c(isSigned   = isSigned  (x),
                    isComplex  = isComplex (x),
                    isFloating = isFloating(x))
t(vapply(L, F, logical(3L)))
```

---

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

```
## Class generator functions

ulong(x = 0L, length = 0L, names = NULL)
slong(x = 0L, length = 0L, names = NULL)

ulong.array(x = 0L, dim = length(x), dimnames = NULL)
slong.array(x = 0L, dim = length(x), dimnames = NULL)

## Limits

ULONG_MAX
SLONG_MIN
SLONG_MAX
```

Arguments

`x`	an atomic or `flint` vector containing data for conversion to `ulong` or `slong`.
`length`	a numeric vector of length one giving the length of the return value. If that exceeds the length of `x`, then `x` is recycled. Non-integer values are rounded in the direction of zero.
`names`	the `names` slot of the return value, either `NULL` or a character vector of equal length. Non-character `names` are coerced to character.

`dim`	the `dim` slot of the return value, an integer vector of nonzero length. If the product exceeds the length of `x`, then `x` is recycled. Non-integer numeric `dim` are coerced to integer.
`dimnames`	the `dimnames` slot of the return value, either `NULL` or a list of length equal to the length of `dim`. The components are either `NULL` or character vectors of length given by `dim`. Non-character vector components of `dimnames` are coerced to character.

Details

The class generator functions have four distinct usages:

```
ulong()
ulong(length=)
ulong(x)
ulong(x, length=)

slong()
slong(length=)
slong(x)
slong(x, length=)
```

The first usage generates an empty vector. The second usage generates a zero vector of the indicated length. The third usage converts `x`, preserving dimensions, dimension names, and names. The fourth usage converts `x`, recycling its elements to the indicated length and discarding its dimensions, dimension names, and names. Attempts to recycle `x` of length zero to nonzero length are an error.

Usage of `ulong.array` and `slong.array` is modelled after `array`.

`ULONG_MAX` is a `ulong` vector of length 1 storing the greatest integer representable by `ulong`, namely $2^n - 1$, where n is the value of `flintABI()`. `SLONG_MIN` and `SLONG_MAX` are `slong` vectors of length 1 storing the least and greatest integers representable by `slong`, namely -2^{n-1} and $2^{n-1} - 1$.

Value

A `ulong` or `slong` vector, possibly an array; see 'Details'.

Conversion

Real numbers and real parts of complex numbers are rounded in the direction of 0. Imaginary parts of complex numbers are discarded.

Character strings are converted using function `mpz_set_str` from the GNU MP library with argument `base` set to 0; see https://gmplib.org/manual/Assigning-Integers.

An error is signaled if elements of `x` are not in the range of the C type, in particular if elements of `x` are `NaN`, `-Inf`, or `Inf`. 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()`.

Slots

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

Methods

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

`%*%`, `crossprod`, `tcrossprod` `signature(x = "ulong", y = "ulong")`:
  `signature(x = "slong", y = "slong")`:
  `signature(x = "ulong", y = "ANY")`:
  `signature(x = "slong", y = "ANY")`:
  `signature(x = "ANY", y = "ulong")`:
  `signature(x = "ANY", y = "slong")`:
  matrix products. The "other" operand must be atomic or inherit from virtual class `flint`. `crossprod` and `tcrossprod` behave as if `y = x` when `y` is missing or `NULL`. Operands are promoted as necessary and must be conformable (have compatible dimensions). Non-array operands of length `k` are handled as 1-by-`k` or `k`-by-1 matrices depending on the call.

`+` `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 = "ulong", e2 = "ulong")`:
  `signature(e1 = "slong", e2 = "slong")`:
  `signature(e1 = "ulong", e2 = "ANY")`:
  `signature(e1 = "slong", e2 = "ANY")`:
  `signature(e1 = "ANY", e2 = "ulong")`:
  `signature(e1 = "ANY", e2 = "slong")`:
  binary arithmetic, comparison, and logical operators; see `S4groupGeneric`. The "other" operand must be atomic or inherit from virtual class `flint`. Operands are promoted as necessary. Array operands must be conformable (have identical dimensions). Non-array operands are recycled.

`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. See also asVector.

backsolve signature(r = "ulong", x = "ulong"):
signature(r = "slong", x = "slong"):
signature(r = "ulong", x = "ANY"):
signature(r = "slong", x = "ANY"):
signature(r = "ANY", x = "ulong"):
signature(r = "ANY", x = "slong"):
solution of the triangular system op2(op1(r)) %*% y = x, where op1=ifelse(upper.tri, triu, tril) and op2=ifelse(transpose, t, identity) and upper.tri and transpose are optional logical arguments with default values TRUE and FALSE, respectively. The "other" operand must be atomic or inherit from virtual class flint. If x is missing, then the return value is the inverse of op2(op1(r)), as if x were the identity matrix. Operands are promoted as necessary and must be conformable (have compatible dimensions). Non-array x are handled as length(x)-by-1 matrices. If r and (if not missing) x are both formally rational, then the solution is exact and the return value is an fmpq matrix.

chol signature(x = "ulong"):
signature(x = "slong"):
coerces x to class arf and dispatches.

chol2inv signature(x = "ulong"):
signature(x = "slong"):
returns the inverse of the positive definite matrix whose upper triangular Cholesky factor is the upper triangular part of x. The return value is the exact inverse, being computed as tcrossprod(backsolve(x)).

coerce signature(from = "ANY", to = "ulong"):
signature(from = "ANY", to = "slong"):
returns the value of ulong(from) or slong(from).

colSums signature(x = "ulong"):
signature(x = "slong"):
returns a ulong or (in case of overflow) fmpz vector or array containing the column sums of x, defined as sums over dimensions 1:dims.

colMeans signature(x = "ulong"):
signature(x = "slong"):
returns an fmpq vector or array containing the column means of x, defined as means over dimensions 1:dims.

det, determinant, diff, diffinv `signature(x = "ulong")`:
`signature(x = "slong")`:
coerces x to class fmpz and dispatches.

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

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.

rowSums `signature(x = "ulong")`:
`signature(x = "slong")`:
returns a ulong or (in case of overflow) fmpz vector or array containing the row sums of x, defined as sums over dimensions `(dims+1):length(dim(x))`.

rowMeans `signature(x = "ulong")`:
`signature(x = "slong")`:
returns an fmpq vector or array containing the row means of x, defined as means over dimensions `(dims+1):length(dim(x))`.

solve `signature(a = "ulong", b = "ulong")`:
`signature(a = "slong", b = "slong")`:
`signature(a = "ulong", b = "ANY")`:
`signature(a = "slong", b = "ANY")`:
`signature(a = "ANY", b = "ulong")`:
`signature(a = "ANY", b = "slong")`:
solution of the general system a %*% x = b. The "other" operand must be atomic or inherit from virtual class flint. If b is missing, then the return value is the inverse of a, as if b were the identity matrix. Operands are promoted as necessary and must be conformable (have compatible dimensions). Non-array b are handled as `length(b)`-by-1 matrices. If a and (if not missing) b are both formally rational, then the solution is exact and the return value is an fmpq matrix.

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"))

(zu <- ulong(length = 12L))
stopifnot(flintSize(zu) * 8L == length(zu) * flintABI())

## Conversion of data not representable in target type/class depends on
## existence of NA value
as.integer(SLONG_MIN)
try(ulong(-1L))
try(slong(NA_integer_))

## Overflow in +, -, *, %/% promotes to arbitrary precision
ulong(0L) - ulong(1L)
ULONG_MAX + ulong(1L)

## Mixture of unsigned and signed promotes to signed arbitrary precision
## ("raw" is unsigned, "logical" and "integer" are signed)
c(ulong(1L), as.raw(0L))
c(ulong(1L), FALSE)
c(ulong(1L), 0L)
c(ulong(1L), slong(0L))

## Quotients are formally rational
SLONG_MAX/seq_len(8L)
mean(slong(c(1L, 1L, 2L)))
summary(slong(c(1L, 1L, 2L, 3L, 5L, 8L, 13L, 21L)))

## Floored integer division
p <- slong(-127L)
q <- c(-(3L:1L), 1L:3L)
stopifnot(identical(p %/% q * q + p %% q, rep(p, 6L)))

## A sequence
d <- 3L
(s <- seq.int(from = SLONG_MIN + 1L, by = SLONG_MAX, length.out = d))
stopifnot(identical(s, c(-SLONG_MAX, 0L, SLONG_MAX)),
          identical(s %/% SLONG_MAX, slong(-1L:1L)),
          identical(s  /  SLONG_MAX, fmpq(-1L:1L)))

## An array
(Xs <- diag(s, d))
(Iu <- cbind(ulong(1L:d), ulong(1L:d)))
stopifnot(identical(Xs, s * slong.array(rep(c(1L, 0L), c(1L, d)), c(d, d))),
          identical(Xs[Iu], s), identical(diag(Xs), s))
```

Index