Package ‘Rmpfr’

November 15, 2016

Title R MPFR - Multiple Precision Floating-Point Reliable
Version 0.6-1
Date 2016-11-15
DateNote Previous CRAN version 0.6-0 on 2015-12-04
Type Package
Author Martin Maechler
Maintainer Martin Maechler <maechler@stat.math.ethz.ch>
SystemRequirements gmp (>= 4.2.3), mpfr (>= 3.0.0)
SystemRequirementsNote MPFR (MP Floating-Point Reliable Library, http://mpfr.org/) and GMP (GNU Multiple Precision library, http://gmplib.org/), see >> README.md
Depends gmp (>= 0.5-8), R (>= 3.0.1)
Imports stats, utils, methods
Suggests MASS, polynom, sfsmisc (>= 1.0-20), Matrix
SuggestsNote MASS, polynom, sfsmisc: only for vignette; Matrix: test-tools
URL http://rmpfr.r-forge.r-project.org/
Description Arithmetic (via S4 classes and methods) for arbitrary precision floating point numbers, including transcendental ("special") functions. To this end, Rmpfr interfaces to the LGPL'ed MPFR (Multiple Precision Floating-Point Reliable) Library which itself is based on the GMP (GNU Multiple Precision) Library.
License GPL (>= 2)
NeedsCompilation yes
Repository CRAN
Date/Publication 2016-11-15 19:38:16
R topics documented:

Rmpfr-package .................................................. 2
array_or_vector-class ........................................ 4
asNumeric-methods ........................................... 5
atomicVector-class ........................................... 6
Bernoulli ............................................................ 7
Bessel_mpfr ......................................................... 8
bind-methods ..................................................... 9
chooseMpfr ......................................................... 10
factorialMpfr .................................................... 11
formatHex .......................................................... 13
formatMpfr ........................................................ 15
gmp-conversions .................................................. 17
hjkMpfr ............................................................. 18
integrateR .......................................................... 20
is.whole ............................................................. 23
Mnumber-class .................................................... 24
mpfr ................................................................. 25
mpfr-class ........................................................ 27
mpfr-distr-etc .................................................... 32
mpfr-special-functions ......................................... 33
mpfr-utils ........................................................ 34
mpfr.utils ........................................................ 37
mpfrArray ........................................................ 38
mpfrMatrix ......................................................... 40
mpfrMatrix-utils ................................................ 43
optimizeR .......................................................... 44
pbetaI .............................................................. 46
pmax ................................................................. 48
roundMpfr ........................................................ 49
seqMpfr ............................................................. 50
str.mpfr ........................................................... 51
sumBinomMpfr ..................................................... 52
unirootR ........................................................... 54

Index ............................................................... 57

Rmpfr-package

R MPFR - Multiple Precision Floating-Point Reliable

Description

Rmpfr provides S4 classes and methods for arithmetic including transcendental ("special") functions for arbitrary precision floating point numbers, here often called “mpfr - numbers”. To this end, it interfaces to the LGPL’ed MPFR (Multiple Precision Floating-Point Reliable) Library which itself is based on the GMP (GNU Multiple Precision) Library.
Details

Package: Rmpfr
SystemRequirements: gmp (>= 4.2.3), mpfr (>= 3.0.0)
                      (C (not R!) libraries; must be installed)
Depends: methods, gmp (>= 0.5-8), R (>= 2.12.0)
Imports: gmp, stats, utils
Suggests: MASS, polynom, sfsmisc (>= 1.0-20), Matrix
SuggestNotes: MASS, polynom, sfsmisc are only needed for vignette; Matrix only because of its test-tools
URL: http://rmpfr.r-forge.r-project.org/
License: GPL (>= 2)

The following (help pages) index does not really mention that we provide many methods for mathematical functions, including gamma, digamma, etc, namely, all of R's (S4) Math group (with the only exception of trigamma), see the list in the examples. Additionally also pnorm, the "error function", and more, see the list in zeta, and further note the first vignette (below).

Partial index:

- mpfr: Create "mpfr" Numbers (Objects)
- mpfrArray: Construct "mpfrArray" almost as by array()
- mpfr-class: Class "mpfr" of Multiple Precision Floating Point Numbers
- mpfrMatrix-class: Classes "mpfrMatrix" and "mpfrArray"
- Bernoulli: Bernoulli Numbers in Arbitrary Precision
- Bessel_mpf: Bessel functions of Integer Order in multiple precisions
- c.mpf: MPFR Number Utilities
- cbind: "mpfr" . . . - Methods for Functions cbind(), rbind()
- chooseMpf: Binomial Coefficients and Pochhammer Symbol aka Rising Factorial
- factorialMpf: Factorial 'n!' in Arbitrary Precision
- formatMpf: Formatting MPFR (multiprecision) Numbers
- getPrec: Rmpfr - Utilities for Precision Setting, Printing, etc
- roundMpf: Rounding to Binary bits, "mpfr-internally"
- seqMpf: "mpfr" Sequence Generation
- sumBinomMpf: (Alternating) Binomial Sums via Rmpfr
- zeta: Special Mathematical Functions (MPFR)
- integrateR: One-Dimensional Numerical Integration - in pure R
- unirootR: One Dimensional Root (Zero) Finding - in pure R
- optimizeR: High Precisione One-Dimensional Optimization
- hjkMpf: Hooke-Jeeves Derivative-Free Minimization R (working for MPFR)

Further information is available in the following vignettes:

- Rmpfr-pkg: Rmpfr (source, pdf)
- log1exp-note: Accurately Computing log(1 - exp(.)) – Assessed by Rmpfr (source, pdf)
Author(s)

Martin Maechler

References

MPFR (MP Floating-Point Reliable Library), http://mpfr.org/
GMP (GNU Multiple Precision library), http://gmplib.org/
and see the vignettes mentioned above.

See Also

The R package gmp for big integer and rational numbers (bigrational) on which Rmpfr now depends.

Examples

```r
## Using "mpfr" numbers instead of regular numbers...
n1.25 <- mpfr(5, precBits = 256)/4
n1.25

## and then "everything" just works with the desired chosen precision: high
n1.25 ^ c(1:7, 20, 30) ## fully precise; compare with
print(1.25 ^ 30, digits=19)

exp(n1.25)

## Show all math functions which work with "MPFR" numbers (1 exception: trigamma)
getGroupMembers("Math")

## We provide *many* arithmetic, special function, and other methods:
showMethods(classes = "mpfr")
showMethods(classes = "mpfrArray")
```

array_or_vector-class

Auxiliary Class "array\_or\_vector"

Description

"array_or_vector" is the class union of c("array", "matrix", "vector") and exists for its use in signatures of method definitions.

Details

Using "array_or_vector" instead of just "vector" in a signature makes an important difference: E.g., if we had setMethod(crossprod, c(x="mpfr", y="vector"), function(x,y) CPR(x,y)), a call crossprod(x, matrix(1:6, 2,3)) would extend into a call of CPR(x, as(y, "vector")) such that CPR()'s second argument would simply be a vector instead of the desired $2 \times 3$ matrix.
Objects from the Class

A virtual Class: No objects may be created from it.

Examples

showClass("array_or_vector")

Description

Methods for function `asNumeric` (in package `gmp`).

Usage

## S4 method for signature 'mpfrArray'
asNumeric(x)

Arguments

- `x` a “number-like” object, here, a `mpfr` or typically `mpfrArray`one.

Value

an R object of type (typeof) "numeric", a `matrix` or `array` if `x` had non-NULL dimension `dim()`.

Methods

signature(x = "mpfrArray") this method also dispatches for `mpfrMatrix` and returns a numeric array.

signature(x = "mpfr") for non-array/matrix, `asNumeric(x)` is basically the same as `as.numeric(x)`.

Author(s)

Martin Maechler

See Also

our lower level (non-generic) `toNum()`. Further, `asNumeric` (package `gmp`), standard R’s `as.numeric()`.
Examples

```r
x <- (0:7)/8 # (exact)
X <- mpfr(x, 99)
stopifnot(identical(asNumeric(x), x),
           identical(asNumeric(X), x))

m <- matrix(1:6, 3, 2)
(M <- mpfr(m, 99) / 5) #-- "mpfrMatrix"
asNumeric(M) # numeric matrix
stopifnot(all.equal(asNumeric(M), m/5),
           identical(asNumeric(m), m)) # remains matrix
```

atomicVector-class

Virtual Class "atomicVector" of Atomic Vectors

Description

The class "atomicVector" is a virtual class containing all atomic vector classes of base \texttt{R}, as also implicitly defined via \texttt{is.atomic}.

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

In the \texttt{Matrix} package, the "atomicVector" is used in signatures where typically “old-style” "matrix" objects can be used and can be substituted by simple vectors.

Extends

The atomic classes "logical", "integer", "double", "numeric", "complex", "raw" and "character" are extended directly. Note that "numeric" already contains "integer" and "double", but we want all of them to be direct subclasses of "atomicVector".

Author(s)

Martin Maechler

See Also

\texttt{is.atomic, integer, numeric, complex}, etc.

Examples

```r
showClass("atomicVector")
```
Bernoulli Numbers in Arbitrary Precision

Description

Computes the Bernoulli numbers in the desired (binary) precision. The computation happens via the \texttt{zeta} function and the formula

\[ B_k = -k \zeta(1 - k), \]

and hence the only non-zero odd Bernoulli number is \( B_1 = +1/2 \). (Another tradition defines it, equally sensibly, as \(-1/2\).)

Usage

\texttt{Bernoulli(k, precBits = 128)}

Arguments

- \( k \): non-negative integer vector
- \( \text{precBits} \): the precision in \textit{bits} desired.

Value

an \texttt{mpfr} class vector of the same length as \( k \), with \( i \)-th component the \( k[i] \)-th Bernoulli number.

Author(s)

Martin Maechler

References

http://en.wikipedia.org/wiki/Bernoulli_number

See Also

\texttt{zeta} is used to compute them.

Examples

\begin{verbatim}
Bernoulli(0:10)
plot(as.numeric(Bernoulli(0:15)), type = "h")

curve(-x*zeta(1-x), -2, 15.03, n=300,
     main = expression(-x \times \zeta(1-x)))
legend("top", paste(c("even","odd "), "Bernoulli numbers"),
    pch=c(1,3), col=2, pt.cex=2, inset=1/64)
abline(h=0, v=0, lty=3, col="gray")
k <- 0:15; k[1] <- 1e-4
\end{verbatim}
Bessel functions of Integer Order in multiple precisions

Description

Bessel functions of integer orders, provided via arbitrary precision algorithms from the MPFR library.

Note that the computation can be very slow when \( n \) and \( x \) are large (and of similar magnitude).

Usage

\[
\begin{align*}
\text{Ai}(x) \\
\text{j0}(x) \\
\text{j1}(x) \\
\text{jn}(n, x, \text{rnd.mode} = \text{c}("N","D","U","Z","A")) \\
\text{y0}(x) \\
\text{y1}(x) \\
\text{yn}(n, x, \text{rnd.mode} = \text{c}("N","D","U","Z","A"))
\end{align*}
\]

Arguments

- \( x \) : a numeric or mpfr vector.
- \( n \) : non-negative integer (vector).
- \( \text{rnd.mode} \) : a 1-letter string specifying how rounding should happen at C-level conversion to MPFR, see mpfr.

Value

Computes multiple precision versions of the Bessel functions of integer order, \( J_n(x) \) and \( Y_n(x) \), and—when using MPFR library 3.0.0 or newer—also of the Airy function \( Ai(x) \).

See Also

besselJ, and bessely compute the same bessel functions but for arbitrary real order and only precision of a bit more than ten digits.
Examples

```r
x <- (0:100)/8 # (have exact binary representation)
stopifnot(
  all.equal(besselY(x, 0), bY0 <- y0(x)),
  all.equal(besselJ(x, 1), bJ1 <- j1(x)),
  all.equal(yn(0,x), bY0),
  all.equal(jn(1,x), bJ1)
)
```

bind-methods  

"mpfr" '...' - Methods for Functions cbind(), rbind()

Description

`cbind` and `rbind` methods for signature `...` (see `dotsMethods` are provided for class `Mnumber`, i.e., for binding numeric vectors and class "mpfr" vectors and matrices ("mpfrMatrix") together.

Usage

```r
cbind(..., deparse.level = 1)
rbind(..., deparse.level = 1)
```

Arguments

- `...` matrix-/vector-like `R` objects to be bound together, see the `base` documentation, `cbind`.
- `deparse.level` integer determining under which circumstances column and row names are built from the actual arguments’ `expression`, see `cbind`.

Value

typically a ‘matrix-like’ object, here typically of class "mpfrMatrix".

Methods

- `... = "Mnumber"` is used to (c|r)bind multiprecision “numbers” (inheriting from class "mpfr") together, maybe combined with simple numeric vectors.
- `... = "ANY"` reverts to `cbind` and `rbind` from package `base`.

Author(s)

Martin Maechler

See Also

`cbind2`, `cbind`, Documentation in base `R`’s `methods` package

Examples

```r
cbind(1, mpfr(6:3, 70)/7, 3:0)
```
chooseMpfr

_binomial Coefficients and Pochhammer Symbol aka Rising Factorial_

**Description**

Compute binomial coefficients, `chooseMpfr(a, n)` being mathematically the same as `choose(a, n)`, but using high precision (MPFR) arithmetic.

`chooseMpfr.all(n)` means the vector `choose(n, 1:n)`, using enough bits for exact computation via MPFR. However, `chooseMpfr.all()` is now deprecated in favor of `chooseZ` from package _gmp_, as that is now vectorized.

`pochmpfr(a, n)` computes the Pochhammer symbol or “rising factorial”, also called the “Pochhammer function”, “Pochhammer polynomial”, “ascending factorial”, “rising sequential product” or “upper factorial”;

\[ x^{(n)} = x(x + 1)(x + 2) \cdots (x + n - 1) = \frac{(x + n - 1)!}{(x - 1)!} = \frac{\Gamma(x + n)}{\Gamma(x)}. \]

**Usage**

```
chooseMpfr (a, n, rnd.mode = c("N", "D", "U", "Z", "A"))
chooseMpfr.all(n, precBits=NULL, k0=1, alternating=FALSE)
pochmpfr(a, n, rnd.mode = c("N", "D", "U", "Z", "A"))
```

**Arguments**

- `a`: a numeric or `mpfr` vector.
- `n`: an integer vector; if not of length one, `n` and `a` are recycled to the same length.
- `rnd.mode`: a 1-letter string specifying how rounding should happen at C-level conversion to MPFR, see _mpfr_.
- `precBits`: integer or NULL for increasing the default precision of the result.
- `k0`: integer scalar
- `alternating`: logical, for `chooseMpfr.all()`, indicating if alternating sign coefficients should be returned, see below.

**Value**

For

- `chooseMpfr()`, `pochmpfr()`: an `mpfr` vector of length `max(length(a), length(n))`;
- `chooseMpfr.all(n, k0)`: a `mpfr` vector of length `n-k0+1`, of binomial coefficients \( C_{n,m} \) or, if alternating is true, \((-1)^m \cdot C_{n,m}\) for \( m \in k0:n \).

**Note**

If you need high precision `choose(a, n)` (or `Pochhammer(a,n)`) for large `n`, maybe better work with the corresponding `factorial(mpfr(...))`, or `gamma(mpfr(...))` terms.
factorialMpfr

See Also

`choose(n, m)` (*base R*) computes the binomial coefficient $C_{n,m}$ which can also be expressed via Pochhammer symbol as $C_{n,m} = (n - m + 1)! / m!$.

`chooseZ` from package *gmp*; for now, *factorialMpfr*.

For (alternating) binomial sums, directly use `sumBinomMpfr`, as that is potentially more efficient.

Examples

```r
pochMpfr(100, 4) == 100*101*102*103 # TRUE
a <- 100:110
pochMpfr(a, 10) # exact (but too high precision)
x <- mpfr(a, 70)# should be enough
(px <- pochMpfr(x, 10)) # the same as above (needing only 70 bits)
stopifnot(pochMpfr(a, 10) == px,
         px[1] == prod(mpfr(100:109, 100)))# used to fail

(c1 <- chooseMpfr(1000:997, 60)) # -> automatic "correct" precision
stopifnot(all.equal(c1, choose(1000:997, 60), tolerance=1e-12))

## --- Experimenting & Checking
if(!Rmpfr:::doExtras()) { ## speed up: smaller set
  n. <- n.set[-(1:10)]
  n.set <- c(1:10, n.[ c(TRUE, diff(n.) > 1)])
}
C1 <- C2 <- numeric(length(n.set))
for(i.n in seq_along(n.set)) {
  cat(n. <- n.set[i.n],":")
  C1[i.n] <- system.time(c.c <- chooseMpfr.all(n.))[1]
  C2[i.n] <- system.time(c.2 <- chooseMpfr(n. , 1:n.))[1]
  stopifnot(is.whole(c.c), c.c == c.2,
            if(n > 60) TRUE else all.equal(c.c, choose(n., 1:n.), tolerance = 1e-15))
  cat(" [OK]\n")
}
matplot(n.set, cbind(C1, C2), type="b", log="xy",
         xlab = "n", ylab = "system.time(.) [s]"
legend("topleft", c("chooseMpfr.all(n)", "chooseMpfr(n, 1:n)"),
        pch=as.character(1:2), col=1:2, lty=1:2, bty="n")

## Currently, chooseMpfr.all() is faster only for large n (>= 300)
## That would change if we used c-code for the *.all() version
```

*factorialMpfr*  
*Factorial ’n!’ in Arbitrary Precision*
Description

Efficiently compute \( n! \) in arbitrary precision, using the MPFR-internal implementation. This is mathematically (but not numerically) the same as \( \Gamma(n + 1) \).

\texttt{factorialZ} (package \texttt{gmp}) should typically be used \textit{instead} of \texttt{factorialMpf}r() nowadays. Hence, \texttt{factorialMpf}r now is somewhat \textit{deprecated}.

Usage

\begin{verbatim}
factorialMpf\texttt{r}(n, \text{precBits} = \text{max(2, ceiling(lgamma(n+1)/log(2)))),
    \text{rnd.mode} = \text{c("N","D","U","Z","A")})
\end{verbatim}

Arguments

- \( n \) non-negative integer (vector).
- \( \text{precBits} \) desired precision in bits (“binary digits”); the default sets the precision high enough for the result to be \textit{exact}.
- \( \text{rnd.mode} \) a 1-letter string specifying how \textit{rounding} should happen at C-level conversion to MPFR, see \texttt{mpfr}.

Value

a number of (S4) class \texttt{mpfr}.

See Also

\texttt{factorial} and \texttt{gamma} in base \texttt{R}.

\texttt{factorialZ} (package \texttt{gmp}), to \textit{replace} \texttt{factorialMpf}r, see above.

\texttt{chooseMpf}r() and \texttt{pochMpf}r() (on the same page).

Examples

\begin{verbatim}
factorialMpf\texttt{r}(200)
n <- 1'000:1'010
f1000 <- factorialMpf\texttt{r}(n)
stopifnot(1e-15 > abs(as.numeric(1 - \text{lfactorial}(n)/log(f1000)))))

## Note that---astonishingly--- measurements show only
## *small* efficiency gain of \( \approx 10\% \) over using the previous "technique"
system.time(repalate(8, f1e4 <- factorialMpf\texttt{r}(100000)))
system.time(repalate(8, f.1e4 <- factorial(mpfr(10000,
    prec=1+lfactorial(10000)/log(2))))))
\end{verbatim}
**formatHex**

*Flexibly Format Numbers in Binary, Hex and Decimal Format*

**Description**

Show numbers in binary, hex and decimal format. The resulting character-like objects can be back-transformed to "mpfr" numbers via `mpfr()`.

**Usage**

```r
formatHex(x, precBits = min(getPrec(x)), style = "+")

formatBin(x, precBits = min(getPrec(x)), scientific = TRUE,
          left.pad = " ", right.pad = left.pad, style = "+")

formatDec(x, precBits = min(getPrec(x)), digits = decdigits,
          nsmall = NULL, scientific = FALSE, style = "+", ...)
```

**Arguments**

- **x**: A numeric or `mpfr` R object.
- **precBits**: Integer, the number of bits of precision, typically derived from `x`, see `getPrec`. Numeric, i.e., double precision numbers have 53 bits. For more detail, see `mpfr`.
- **style**: A single character, to be used in `sprintf`'s format (fmt), immediately after the " sets a sign in the output, i.e., "+" or "-", where as style = " " may seem more standard.
- **scientific**: Logical indicating that `formatBin` should display the binary representation in scientific notation (`mpfr(3, 5)` is displayed as `+0b1.1000p+1`). When FALSE, `formatBin` will display the binary representation in regular format shifted to align binary points (`mpfr(3, 5)` is displayed `+0b11.000`).
- **...**: Additional optional arguments.
  - `formatHex`, `formatBin`: `precBits` is the only ... argument acted on. Other ... arguments are ignored.
  - `formatDec`: `precBits` is acted on. Any argument accepted by `format` (except `nsmall`) is acted on. Other ... arguments are ignored.
- **left.pad, right.pad**: Characters (one-character strings) that will be used for left- and right-padding of the formatted string when `scientific=FALSE`. *Do not change these unless for display-only purpose !!*
- **nsmall**: Only used when `scientific` is false, then passed to `format()`. If NULL, the default is computed from the range of the non-zero values of `x`. 
digits

integer; the number of decimal digits displayed is the larger of this argument and the internally generated value that is a function of precBits. This is related to but different than digits in format.

Details

For the hexadecimal representation, when the precision is not larger than double precision, sprintf() is used directly, otherwise formatMpfr() is used and post processed. For the binary representation, the hexadecimal value is calculated and then edited by substitution of the binary representation of the hex characters coded in the HextoBin vector. For binary with scientific=FALSE, the result of the scientific=TRUE version is edited to align binary points. For the decimal representation, the hexadecimal value is calculated with the specified precision and then sent to the format function for scientific=FALSE or to the sprintf function for scientific=TRUE.

Value

A character vector (or matrix) like x, say r, containing the formatted representation of x, with a class (unless left.pad or right.pad were not "."). In that case, formatHex() returns class "B\text{character}\n" and formatBin() gives class "B\text{character}"; for these, mpfr(.) has methods and will basically return x, i.e., work as inverse function.

Author(s)

Richard M. Heiberger <rmh@temple.edu>, with minor tweaking by Martin M.

References

R FAQ 7.31: Why doesn’t R think these numbers are equal? system.file("..\../doc/FAQ")

See Also

mpfr, sprintf

Examples

```r
FourBits <- mpfr(matrix(0:31, 8, 4), dimnames = list(0:7, c(0,8,16,24)),
                 precBits=4) ## 4 significant bits
FourBits

formatHex(FourBits)
formatBin(FourBits, style = " ")
formatBin(FourBits, scientific=FALSE)
formatDec(FourBits)

FBB <- formatBin(FourBits)
(nFBB <- mpfr(FBB))
stopifnot(all.equal(nFBB, FourBits, tol=0))

FBH <- formatHex(FourBits)
(nFBH <- mpfr(FBH))
stopifnot(all.equal(nFBH, FourBits, tol=0))
```
formatMpfr

TenFrac <- matrix((1:10)/10, dimnames=list(1:10, expression(1/x)))
TenFrac9 <- mpfr(TenFrac, precBits=9) ## 9 significant bits
TenFrac9
formatHex(TenFrac9)
formatBin(TenFrac9)
formatBin(TenFrac9, scientific=FALSE)
formatDec(TenFrac9)
formatDec(TenFrac9, precBits=10)

---

**Description**

Flexible formatting of “multiprecision numbers”, i.e., objects of class `mpfr`. `formatMpfr()` is also the `mpfr` method of the generic `format` function.

The `formatN()` methods for `mpfr` numbers renders them differently than their double precision equivalents, by appending "_M".

**Usage**

```r
formatMpfr(x, digits = NULL, trim = FALSE, scientific = NA,
            base = 10, showNeg0 = TRUE,
            big.mark = "", big.interval = 3L,
            small.mark = ".", small.interval = 5L,
            decimal.mark = ".",
            exponent.char = if(base <= 14) "e" else if(base <= 36) "E" else "|e",
            zero.print = NULL, drop0trailing = FALSE, ...)
```

## S3 method for class 'mpfr'

```r
formatN(x, drop0trailing = TRUE, ...)
```

**Arguments**

- `x` an MPFR number (vector or array).
- `digits` how many significant digits (in the base chosen!) are to be used in the result. The default, `NULL`, uses enough digits to represent the full precision, often one or two digits more than “you” would expect.
- `trim` logical; if `FALSE`, numbers are right-justified to a common width; if `TRUE` the leading blanks for justification are suppressed.
- `scientific` either a logical specifying whether MPFR numbers should be encoded in scientific format, or an integer penalty (see `options("scipen")`). Missing values correspond to the current default penalty.
- `base` an integer in 2, 3, ..., 62; the base (“basis”) in which the numbers should be represented. Apart from the default base 10, binary (`base = 2`) or hexadecimal (`base = 16`) are particularly interesting.
showNeg0 logical indicating if "negative" zeros should be shown with a "-". The default, TRUE is intentionally different from `format(<numeric>)`.

exponent.char the "exponent" character to be used in scientific notation. The default takes into account that for base \( B \geq 15 \), "e" is part of the (mantissa) digits and the same is true for "E" when \( B \geq 37 \).

big.mark, big.interval, small.mark, small.interval, decimal.mark, zero.print, drop.trailing used for prettying decimal sequences, these are passed to `prettyNum` and that help page explains the details.

... further arguments passed to or from other methods.

Value

a character vector or array, say `cx`, of the same length as `x`. Since Rmpfr version 0.5-3 (2013-09), if `x` is an `mpfrArray`, then `cx` is a character array with the same `dim` and `dimnames` as `x`.

Note that in scientific notation, the integer exponent is always in decimal, i.e., base 10 (even when base is not 10), but of course meaning base powers, e.g., in base 32, "u.giE3" is the same as "ug10" which is 32 \(^3\) times "u.g1". This is in contrast, e.g., with `sprintf("%a", x)` where the powers after "p" are powers of 2.

Author(s)

Martin Maechler

References

The MPFR manual’s description of ‘`mpfr_get_str()`’ which is the C-internal workhorse for the (internal) R function .mpfr2str() on which `formatMpfrr` builds.

See Also

`mpfr` for creation and the `mpfr` class description with its many methods. The `format` generic, and the `prettyNum` utility on which `formatMpfrr` is based as well. The S3 generic function `formatN` from package `gmp`.

Examples

```r
## Printing of MPFR numbers uses formatMpfrr() internally.
## Note how each components uses the "necessary" number of digits:
(x3 <- c(Const("pi", 168), mpfr(pi, 140), 3.14))
format(x3[3], 15)
format(x3[3], 15, drop0 = TRUE)# "3.14" .. dropping the trailing zeros
x3[4] <- 2^30
x3[4] # automatically drops trailing zeros
format(x3[1]), dig = 41, small.mark = "\"") # (41 - 1 = ) 40 digits after "."

rbind(formatN(x3, digits = 15),
      formatN(as.numeric(x3), digits = 15))
(Zero <- mpfr(c(0,1/-Inf), 20)) # 0 and "-0"
xx <- c(Zero, 1:2, Const("pi", 120), -100*pi, -00987)
```
gmp-conversions

Conversion Utilities gmp <-> Rmpfr

Description

Coerce from and to big integers (bigz) and mpfr numbers.

Further, coerce from big rationals (bigq) to mpfr numbers.

Usage

```r
.bigz2mpfr(x, precB = NULL, rnd.mode = c('N','D','U','Z','A'))
.bigq2mpfr(x, precB = NULL, rnd.mode = c('N','D','U','Z','A'))
.mpfr2bigz(x, mod = NA)
```

Arguments

- **x**: an R object of class bigz, bigq or mpfr respectively.
- **precB**: precision in bits for the result. The default, NULL, means to use the minimal precision necessary for correct representation.
- **rnd.mode**: a 1-letter string specifying how rounding should happen at C-level conversion to MPFR, see details of mpfr.
- **mod**: a possible modulus, see as.bigz in package gmp.

Details

Note that we also provide the natural (S4) coercions, as(x, "mpfr") for x inheriting from class "bigz" or "bigq".

```r
gmp-conversions
;
format(xx, digits = 2)
format(xx, digits = 1, showNeg0 = FALSE)# "-0" no longer shown

## Output in other bases :
formatMprfr(mpfr(10^6, 40), base=32, drop0trailing=TRUE)
## "ugi0"
mpfr("ugi0", base=32) #-> 1'0000'000

i32 <- mpfr(1:32, precBits = 64)
format(i32, base=2, drop0trailing=TRUE)
format(i32, base=16, drop0trailing=TRUE)
format((1/i32, base=2, drop0trailing=TRUE)# using scientific notation for [17..32]
format((1/i32, base=32)
format((1/i32, base=62, drop0trailing=TRUE)
format(mprfr(2, 64)^-(1:16), base=16, drop0trailing=TRUE)
```
Value

a numeric vector of the same length as \( x \), of the desired class.

See Also

`mpfr()`, `as.bigz` and `as.bigq` in package `gmp`.

Examples

```r
S <- gmp::Stirling2(50,10)
show(S)
SS <- S * as.bigz(1:3)^128
stopifnot(all.equal(log2(SS[2]) - log2(S), 128, tolerance=1e-15),
           identical(SS, mpfr2bigz(.bigz2mpfr(SS))))

.bigz2mpfr(S)    # 148 bit precision
.bigz2mpfr(S, precB=256) # 256 bit

## rational --> mpfr:
sq <- SS / as.bigz(2)^100
MP <- as(sq, "mpfr")
stopifnot(identical(MP, .bigz2mpfr(sq)),
           SS == MP * as(2, "mpfr")^100)
```

Description

An implementation of the Hooke-Jeeves algorithm for derivative-free optimization.

This is a slight adaption `hjk()` from package `dfoptim`

Usage

`hjkMpfr(par, fn, control = list(), ...)`

Arguments

- `par` Starting vector of parameter values. The initial vector may lie on the boundary. If `lower[i]=upper[i]` for some \( i \), the \( i \)-th component of the solution vector will simply be kept fixed.
- `fn` Nonlinear objective function that is to be optimized. A scalar function that takes a real vector as argument and returns a scalar that is the value of the function at that point.
- `control` `list` of control parameters. See `Details` for more information.
- `...` Additional arguments passed to `fn`.
Details

Argument control is a list specifying changes to default values of algorithm control parameters. Note that parameter names may be abbreviated as long as they are unique.

The list items are as follows:

tol Convergence tolerance. Iteration is terminated when the step length of the main loop becomes smaller than tol. This does not imply that the optimum is found with the same accuracy. Default is 1.e-06.

maxfeval Maximum number of objective function evaluations allowed. Default is Inf, that is no restriction at all.

maximize A logical indicating whether the objective function is to be maximized (TRUE) or minimized (FALSE). Default is FALSE.

target A real number restricting the absolute function value. The procedure stops if this value is exceeded. Default is Inf, that is no restriction.

info A logical variable indicating whether the step number, number of function calls, best function value, and the first component of the solution vector will be printed to the console. Default is FALSE.

If the minimization process threatens to go into an infinite loop, set either maxfeval or target.

Value

A list with the following components:

par Best estimate of the parameter vector found by the algorithm.

value value of the objective function at termination.

convergence indicates convergence (TRUE) or not (FALSE).

feval number of times the objective fn was evaluated.

niter number of iterations (“steps”) in the main loop.

Note

This algorithm is based on the Matlab code of Prof. C. T. Kelley, given in his book “Iterative methods for optimization”. It has been implemented for package dfoptim with the permission of Prof. Kelley.

This version does not (yet) implement a cache for storing function values that have already been computed as searching the cache makes it slower.

Author(s)

Hans W Borchers <hwborchers@googlemail.com>; for Rmpfr: John Nash, June 2012. Modifications by Martin Maechler.

References

C.T. Kelley (1999), Iterative Methods for Optimization, SIAM.
See Also

Standard R’s optim, optimizeR provides one-dimensional minimization methods that work with mpfr-class numbers.

Examples

```r
## simple smooth example:
ff <- function(x) sum((x - c(2:4))^2)
str(rr <- hjkMpf(r(mpfr(0, 128), 3), ff, control=list(info=TRUE)))

## Hooke-Jeeves solves high-dim. Rosenbrock function (but slowly!)
rosenbrook <- function(x) {
  n <- length(x)
  sum (100*(x[1] - x[1:(n-1)])^2 + (x[n]-1)^2)
}
par0 <- rep(0, 10)
str(rb.db <- hjkMpf(rep(0, 10), rosenbrook, control=list(info=TRUE)))

## rosenbrook() is quite slow with mpfr-numbers:
str(rb.M <- hjkMpf(mpfr(numeric(10), prec=128), rosenbrook, 
  control = list(tol = 1e-8, info=TRUE)))

## Hooke-Jeeves does not work well on non-smooth functions
nsf <- function(x) {
  max(f1, f2, f3)
}
par0 <- c(1, 1) # true min 7.2 at (1.2, 2.4)
h.d <- hjkMpf(par0, 
  nsf) # fmin=8 at xmin=(2,2)

## and this is not at all better (but slower!)
h.M <- hjkMpf(mpfr(c(1,1), 128), nsf, control = list(tol = 1e-15))

## --> demo(hjkMpf) # -> Fletcher's chebyquad function m = n -- residuals
```
**Description**

Numerical integration of one-dimensional functions in pure R, with care so it also works for "mpfr"-numbers.

Currently, only classical Romberg integration of order `ord` is available.

**Usage**

```r
integrateR(f, lower, upper, ..., ord = NULL,
    rel.tol = .Machine$double.eps^0.25, abs.tol = rel.tol,
    max.ord = 19, verbose = FALSE)
```

**Arguments**

- `f`: an R function taking a numeric or "mpfr" first argument and returning a numeric (or "mpfr") vector of the same length. Returning a non-finite element will generate an error.
- `lower`, `upper`: the limits of integration. Currently must be finite. Do use "mpfr"-numbers to get higher than double precision, see the examples.
- `...`: additional arguments to be passed to `f`.
- `ord`: integer, the order of Romberg integration to be used. If this is NULL, as per default, and either `rel.tol` or `abs.tol` are specified, the order is increased until convergence.
- `rel.tol`: relative accuracy requested. The default is 1.2e-4, about 4 digits only, see the Note.
- `abs.tol`: absolute accuracy requested.
- `max.ord`: only used, when neither `ord` or one of `rel.tol`, `abs.tol` are specified: Stop Romberg iterations after the order reaches `max.ord`; may prevent infinite loops or memory explosion.
- `verbose`: logical or integer, indicating if and how much information should be printed during computation.

**Details**

Note that arguments after `...` must be matched exactly.

For convergence, both relative and absolute changes must be smaller than `rel.tol` and `abs.tol`, respectively.

`rel.tol` cannot be less than `max(50*.Machine$double.eps, 0.5e-28)` if `abs.tol <= 0`.

**Value**

A list of class "integrateR" (as from standard R's `integrate()`) with a print method and components

- `value`: the final estimate of the integral.
- `abs.error`: estimate of the modulus of the absolute error.
- `subdivisions`: for Romberg, the number of function evaluations.
message "OK" or a character string giving the error message.
call the matched call.

Note
f must accept a vector of inputs and produce a vector of function evaluations at those points. The Vectorize function may be helpful to convert f to this form.

If you want to use higher accuracy, you must set lower or upper to "mpfr" numbers (and typically lower the relative tolerance, rel.tol), see also the examples.

Note that the default tolerances (rel.tol, abs.tol) are not very accurate, but the same as for integrate, which however often returns considerably more accurate results than requested. This is typically not the case for integrateR().

Note
We use practically the same print S3 method as print.integrate, provided by R, with a difference when the message component is not "Ok".

Author(s)
Martin Maechler

References

See Also
R’s standard, integrate, is much more adaptive, also allowing infinite integration boundaries, and typically considerably faster for a given accuracy.

Examples

```r
## See more from ?integrate
## this is in the region where integrate() can get problems:
integrateR(dnorm, 0, 2000)
integrateR(dnorm, 0, 2000, rel.tol = 1e-15)
(Id <- integrateR(dnorm, 0, 2000, rel.tol = 1e-15, verbose = TRUE))
Id$value == 0.5 # exactly

## Demonstrating that 'subdivisions' is correct:
Exp <- function(x) { .N <<- .N + length(x); exp(x) }
.N <- 0; str(integrateR(Exp, 0, 1, rel.tol = 1e-10), digits = 15); .N

## Using high-precision functions ------

## Polynomials are very nice:
integrateR(function(x) (x-2)^4 - 3*(x-3)^2, 0, 5, verbose = TRUE)
# n=1, 2^n=
2 | I =
46.04, abs.err = 98.9583
```
is.whole

Whole ("Integer") Numbers

Description
Check which elements of \( x[] \) are integer valued aka “whole” numbers, including MPFR numbers (class `mpfr`).

Usage

```r
## S3 method for class 'mpfr'
is.whole(x)
```

Arguments

- **x** any \( \mathbb{R} \) vector, here of class `mpfr`.

Value

logical vector of the same length as \( x[] \), indicating where \( x[] \) is integer valued.

Author(s)

Martin Maechler
See Also

`is.integer(x)` (base package) checks for the `internal` mode or class, not if `x[i]` are integer valued.

The `is.whole()` methods in package `gmp`.

Examples

```r
is.integer(3)  # FALSE, it's internally a double
is.whole(3)    # TRUE
x <- c(as(2,"mpfr") ^ 100, 3, 3.2, 1000000, 2^40)
is.whole(x)    # one FALSE, only
```

---

### Mnumber-class

**Class "Mnumber" and "mNumber" of "mpfr" and regular numbers and arrays from them**

---

**Description**

Classes "Mnumber" "mNumber" are class unions of "mpfr" and regular numbers and arrays from them.

Its purpose is for method dispatch, notably defining a `cbind(...)` method where `...` contains objects of one of the member classes of "Mnumber".

Classes "mNumber" is considerably smaller is it does not contain "matrix" and "array" since these also extend "character" which is not really desirable for generalized numbers. It extends the simple "numericVector" class by mpfr* classes.

**Methods**

```r
%*% signature(x = "mpfrMatrix", y = "Mnumber"): ...
crossprod signature(x = "mpfr", y = "Mnumber"): ...
tcrossprod signature(x = "Mnumber", y = "mpfr"): ...
```

etc. These are documented with the classes `mpfr` and or `mpfrMatrix`.

**See Also**

the `array_or_vector` sub class; `cbind-methods`.

**Examples**

```r
## "Mnumber" encompasses (i.e., "extends") quite a few
## "vector / array - like" classes:
showClass("Mnumber")
stopifnot(extends("mpfrMatrix", "Mnumber"),
          extends("array", "Mnumber"))

Mnsub <- names(getClass("Mnumber")@subclasses)
```
mpfr

Description

Create multiple (i.e. typically high) precision numbers, to be used in arithmetic and mathematical computations with R.

Usage

```r
mpfr(x, precBits, ...)  # Default S3 method:
mpfr(x, precBits, base = 10,
     rnd.mode = c("N","D","U","Z","A"), scientific = NA, ...)

Const(name = c("pi", "gamma", "catalan", "log2"), prec = 120L,
      rnd.mode = c("N","D","U","Z","A"))
```

Arguments

- `x` a numeric, mpfr, bigz, bigq, or character vector or array.
- `precBits`, `prec` a number, the maximal precision to be used, in bits; i.e. 53 corresponds to double precision. Must be at least 2. If missing, getPrec(x) determines a default precision.
- `base` (only when `x` is character) the base with respect to which `x[i]` represent numbers; base `b` must fulfill `2 ≤ b ≤ 62`.
- `rnd.mode` a 1-letter string specifying how rounding should happen at C-level conversion to MPFR, see details.
- `scientific` (used only when `x` is the result of formatBin(), i.e., of class "Bcharacter":) logical indicating that the binary representation of `x` is in scientific notation. When TRUE, mpfr() will substitute 0 for _; when NA, mpfr() will guess, and use TRUE when finding a "p" in `x`; see also formatBin.
- `name` a string specifying the mpfrlib - internal constant computation. "gamma" is Euler's gamma (γ), and "catalan" Catalan's constant.
- `...` potentially further arguments passed to and from methods.
Details

The "mpfr" method of mpfr() is a simple wrapper around roundMpfr().

MPFR supports the following rounding modes,

- **GMP_RNDN**: round to nearest (roundTiesToEven in IEEE 754-2008).
- **GMP_RNDZ**: round toward zero (roundTowardZero in IEEE 754-2008).
- **GMP_RNDU**: round toward plus infinity ("Up", roundTowardPositive in IEEE 754-2008).
- **GMP_RNDD**: round toward minus infinity ("Down", roundTowardNegative in IEEE 754-2008).
- **GMP_RNDA**: round away from zero (new since MPFR 3.0.0).

The ‘round to nearest’ ("N") mode, the default here, works as in the IEEE 754 standard: in case the number to be rounded lies exactly in the middle of two representable numbers, it is rounded to the one with the least significant bit set to zero. For example, the number 5/2, which is represented by (10.1) in binary, is rounded to (10.0)=2 with a precision of two bits, and not to (11.0)=3. This rule avoids the "drift" phenomenon mentioned by Knuth in volume 2 of The Art of Computer Programming (Section 4.2.2).

When x is character, mpfr() will detect the precision of the input object.

Value

an object of (S4) class mpfr, or for mpfr(x) when x is an array, mpfrMatrix, or mpfrArray which the user should just as a normal numeric vector or array.

Author(s)

Martin Maechler

References


See Also

The class documentation mpfr contains more details. Use asNumeric to transform back to double precision ("numeric").

Examples

mpfr(pi, 120) # the double-precision pi "translated" to 120-bit precision

pi. <- Const("pi", prec = 260) # pi "computed" to correct 260-bit precision
pi. # nicely prints 80 digits [260 * log10(2) ~= 78.3 - 80]

Const("gamma", 128L) # 0.5772697345162373
Const("catalan", 128L) # 0.9159655941772149
x <- mpfr(0:7, 100)/7 # a more precise version of k/7, k=0,...,7
x
1 / x

## character input :
mpfr("2.718281828459045235360287471352662497757") - exp(mpfr(1, 150))
## -4 * 10^-40
## Also works for NA, NaN, ... :
Cx <- c("1234567890123456", 345, "NA", "NaN", "Inf", "-Inf")
mpfr(Cx)

## with some 'base' choices :
print(mpfr("111.1111", base=2)) * 2^4

mpfr("af21.01020300a0b0c", base=16)
## 68 bit prec.  44833.00393694653820642

mpfr("ugi0", base = 32) == 10^6  ## TRUE

## --- Large integers from package 'gmp':
Z <- as.bigz(7)^((1:200)
head(Z, 40)
## mpfr(Z) by default chooses the correct *maximal* default precision:
Mz. <- mpfr(Z)
## more efficiently chooses precision individually
Mz <- mpfr(Z, precBits = frexp(Z)$exp)
## the precBits chosen are large enough to keep full precision:
stopifnot(identical(cz <- as.character(Z),
   as(Mz, "character"),
   identical(cz, as(Mz, "character")))

## compare mpfr-arithmetic with exact rational one:
stopifnot(all.equal(mpfr(as.bigq(355,113), 99),
   mpfr(355, 99) / 113, tol = 2^-98))

## look at different "rounding modes":
sapply(c("N", "D","U","Z","A"), function(RND)
   mpfr(c(-1,1)/5, 20, rnd.mode = RND), simplify=FALSE)
symnum(sapply(c("N", "D","U","Z","A"),
   function(RND) mpfr(0.2, prec = 5:15, rnd.mode = RND) < 0.2 ))

---

mpfr-class

Class "mpfr" of Multiple Precision Floating Point Numbers

Description

"mpfr" is the class of Multiple Precision Floatingpoint numbers with Reliable arithmetic. For the high-level user, "mpfr" objects should behave as standard R’s numeric vectors. They would just print differently and use the prespecified (typically high) precision instead of the double precision of ‘traditional’ R numbers (with class(.) == "numeric" and typeof(.) == "double").
hypot($x, y$) computes the hypothenuse length $z$ in a rectangular triangle with “leg” side lengths $x$ and $y$, i.e.,

$$z = hypot(x, y) = \sqrt{x^2 + y^2},$$

in a numerically stable way.

Usage

hypot($x, y$, rnd.mode = c("N", "D", "U", "Z", "A"))

Arguments

$x, y$ an object of class mpfr.

rnd.mode a 1-letter string specifying how rounding should happen at C-level conversion to MPFR, see mpfr.

Objects from the Class

Objects are typically created by mpfr(<number>, precBits).

Slots

Internally, "mpfr" objects just contain standard R lists where each list element is of class "mpfr1", representing one MPFR number, in a structure with four slots, very much parallelizing the C struct in the mpfr C library to which the Rmpfr package interfaces.

An object of class "mpfr1" has slots

prec: "integer" specifying the maximal precision in bits.

exp: "integer" specifying the base-$2$ exponent of the number.

sign: "integer", typically -1 or 1, specifying the sign (i.e. sign(.) of the number.

d: an "integer" vector (of 32-bit ‘limbs’) which corresponds to the full mantissa of the number.

Methods

abs signature(x = "mpfr"): ...

atan2 signature(y = "mpfr", x = "ANY"), and

atan2 signature(x = "ANY", y = "mpfr"): compute the arc-tangent of two arguments:

atan2($y, x$) returns the angle between the x-axis and the vector from the origin to ($x, y$),

i.e., for positive arguments atan2($y, x$) == atan($y/x$).

lbeta signature(a = "ANY", b = "mpfrArray"), is log(|B(a, b)|) where $B(a, b)$ is the Beta function, beta(a, b).

beta signature(a = "mpfr", b = "ANY"),
beta signature(a = "mpfr", b = "mpfr"),.... etc: Compute the beta function $B(a, b)$, using high precision, building on internal gamma or lgamma. See the help for R’s base function beta for more. Currently, there, $a, b \geq 0$ is required. Here, we provide (non-NaN) for all numeric a, b.
When either \(a\), \(b\), or \(a + b\) is a negative integer, \(\Gamma()\) has a pole there and is undefined (NaN). However the Beta function can be defined there as “limit”, in some cases. Following other software such as SAGE, Maple or Mathematica, we provide finite values in these cases. However, note that these are not proper limits (two-dimensional in \((a, b)\)), but useful for some applications. E.g., \(B(a, b)\) is defined as zero when \(a + b\) is a negative integer, but neither \(a\) nor \(b\) is.

Further, if \(a > b > 0\) are integers, \(B(-a, b) = B(b, -a)\) can be seen as \((-1)^b B(a-b+1, b)\).

dim<: \(\text{signature}(x = \text{"mpfr"}):\) Setting a dimension \(\text{dim}\) on an \(\text{"mpfr"}\) object makes it into an object of class \"mpfrArray\" or (more specifically) \"mpfrMatrix\" for a length-2 dimension, see their help page; note that \(t(x)\) (below) is a special case of this.

\begin{verbatim}
Ops signature(e1 = "mpfr", e2 = "ANY"): ...
Ops signature(e1 = "ANY", e2 = "mpfr"): ...
Arith signature(e1 = "mpfr", e2 = "missing"): ...
Arith signature(e1 = "mpfr", e2 = "mpfr"): ...
Arith signature(e1 = "mpfr", e2 = "integer"): ...
Arith signature(e1 = "mpfr", e2 = "numeric"): ...
Arith signature(e1 = "integer", e2 = "mpfr"): ...
Arith signature(e1 = "numeric", e2 = "mpfr"): ...
Compare signature(e1 = "mpfr", e2 = "mpfr"): ...
Compare signature(e1 = "mpfr", e2 = "integer"): ...
Compare signature(e1 = "mpfr", e2 = "numeric"): ...
Compare signature(e1 = "integer", e2 = "mpfr"): ...
Compare signature(e1 = "numeric", e2 = "mpfr"): ...
Logic signature(e1 = "mpfr", e2 = "mpfr"): ...
Summary signature(x = "mpfr"): The S4 \texttt{Summary} group functions, \texttt{max}, \texttt{min}, \texttt{range}, \texttt{prod}, \texttt{sum}, \texttt{any}, and \texttt{all} are all defined for MPFR numbers. \texttt{mean(x, trim)} for non-0 trim works analogously to \texttt{mean.default}.
median signature(x = "mpfr"): works via \texttt{quantile} signature(x = "mpfr"): a simple wrapper of the \texttt{quantile.default} method from \texttt{stats}.
Math signature(x = "mpfr"): All the S4 \texttt{Math} group functions are defined, using multiple precision (MPFR) arithmetic, from \texttt{getGroupMembers("Math")}, these are (in alphabetical order):
\begin{verbatim}
abs, sign, sqrt, ceiling, floor, trunc, cummax, cummin, cumprod, cumsum, exp, expm1, log, log10, log2, log1p, cos, cosh, sin, sinh, tan, tanh, acos, acosh, asin, asinh, atan, atanh, gamma, lgamma, digamma, and trigamma.
\end{verbatim}
Currently, trigamma is not provided by the MPFR library and hence not yet implemented. Further, the \texttt{cum*} methods are \texttt{not yet} implemented.

factorial signature(x = "mpfr"): this will \texttt{round} the result when \(x\) is integer valued. Note however that \texttt{factorialMpfr(n)} for integer \(n\) is slightly more efficient, using the MPFR function \texttt{mpfr_fac_ui}'.

Math2 signature(x = "mpfr"): \texttt{round(x,digits)} and \texttt{signif(x, digits)} methods. Note that these do not change the formal precision (\texttt{prec} slot), and you may often want to apply \texttt{roundMpfr()} in addition or preference.
\end{verbatim}
as.numeric signature(x = "mpfr"): ...

as.vector signature(x = "mpfrArray"): as for standard arrays, this “drops” the dim (and
dimnames), i.e., transforms x into an ‘MPFR’ number vector, i.e., class mpfr.

[[ signature(x = "mpfr", i = "ANY"), and
[ signature(x = "mpfr", i = "ANY", j = "missing", drop = "missing"): subsetting aka
“indexing” happens as for numeric vectors.

format signature(x = "mpfr"). further arguments digits = NULL, scientific = NA, etc:
returns character vector of same length as x; when digits is NULL, with enough digits to
recreate x accurately. For details, see formatMpfr.

is.finite signature(x = "mpfr"): ...

is.infinite signature(x = "mpfr"): ...

is.na signature(x = "mpfr"): ...

is.nan signature(x = "mpfr"): ...

log signature(x = "mpfr"): ...

show signature(object = "mpfr"): ...

sign signature(x = "mpfr"): ...

Re, Im signature(z = "mpfr"): simply return z or 0 (as "mpfr" numbers of correct precision),
as mpfr numbers are 'real' numbers.

Arg, Mod, Conj signature(z = "mpfr"): these are trivial for our 'real' mpfr numbers, but
defined to work correctly when used in R code that also allows complex number input.

all.equal signature(target = "mpfr", current = "mpfr").

all.equal signature(target = "mpfr", current = "ANY"), and

all.equal signature(target = "ANY", current = "mpfr"): methods for numerical (approx-
imate) equality, all.equal of multiple precision numbers. Note that the default tolerance
(argument) is taken to correspond to the (smaller of the two) precisions when both main argu-
ments are of class "mpfr", and hence can be considerably less than double precision machine
epsilon .Machine$double.eps.

coerce signature(from = "numeric", to = "mpfr"): as(. , "mpfr") coercion methods are
available for character strings, numeric, integer, logical, and even raw. Note however, that
mpfr(. , precBits, base) is more flexible.

coerce signature(from = "mpfr", to = "bigz"): coerces to biginteger, see bigz in package
gmp.

coerce signature(from = "mpfr", to = "numeric"): ...

coerce signature(from = "mpfr", to = "character"): ...

unique signature(x = "mpfr"): and
duplicated signature(x = "mpfr"): just work as with numbers.

t signature(x = "mpfr"): makes x into an n × 1 mpfrMatrix.

which.min signature(x = "mpfr"): gives the index of the first minimum, see which.min.

which.max signature(x = "mpfr"): gives the index of the first maximum, see which.max.
Note

Many more methods ("functions") automagically work for "mpfr" number vectors (and matrices, see the mpfrMatrix class doc), notably sort, order, quantile, rank.

Author(s)

Martin Maechler

See Also

The "mpfrMatrix" class, which extends the "mpfr" one.

roundMmpfr to change precision of an "mpfr" object which is typically desirable instead of or in addition to signif() or round(); is.whole() etc.

Special mathematical functions such as some Bessel ones, e.g., jn; further, zeta(.) (= ζ(.)), Ei() etc. Bernoulli numbers and the Pochhammer function pochMmpfr.

Examples

```r
## 30 digit precision
str(x <- mpfr(c(2:3, pi), prec = 30 * log2(10)))
x^2
x[1] / x[2] # 0.66666... ~ 30 digits

## indexing - as with numeric vectors
stopifnot(identical(x[2], x[[2]]),
  ## indexing "outside" gives NA (well: "mpfr=NaN" for now):
  is.na(x[5]),
  ## whereas "[]" cannot index outside:
  is(try(x[[5]]), "try-error"),
  ## and only select *one* element:
  is(try(x[[2:3]]), "try-error"))

## factorial() & ifactorial would work automagically via [lgamma()],
## but factorial() additionally has an "mpfr" method which rounds
f200 <- factorial(mpfr(200, prec = 1500)) # need high prec!।
f200
as.numeric(log2(f200))# 1245.38 -- need precBits >= 1246 for full precision

## see factorialMmpfr() for more such computations.

## "Underflow" **much** later -- exponents have 30(+1) bits themselves:

mpfr.min.exp2 <- - (2^30 + 1)
two <- mpfr(2, 55)
stopifnot(two ^ mpfr.min.exp2 == 0)
## whereas
two ^ (mpfr.min.exp2 * (1 - 1e-15))
## 2.38256490488795107e-3232284977  ["typically"]

## Assert that {sort}, {order}, {quantile}, {rank}, all work :
```
For some R standard (probability) density, distribution or quantile functions, we provide MPFR versions.

Usage

dpois(x, lambda, log = FALSE)
dbinom(x, size, prob, log = FALSE)
dnorm(x, mean = 0, sd = 1, log = FALSE)

pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)

Arguments

x, q, lambda, size, prob, mean, sd

numeric or mpfr vectors. All of these are “recycled” to the length of the longest one.

log, log.p, lower.tail

logical, see pnorm, dpois, etc.

Details

pnorm() is based on erf() and erfc() which have direct MPFR counter parts and are both reparametrizations of pnorm, erf(x) = 2*pnorm(sqrt(2)*x) and erfc(x) = 2* pnorm(sqrt(2)*x, lower=FALSE).
mpfr-special-functions

Value
A vector of the same length as the longest of \( x, q, \ldots \), of class mpfr with the high accuracy results of the corresponding standard R function.

See Also
pnorm, dbinom, dpois in standard package stats.

pbetaI(x, a, b) is a mpfr version of pbeta only for integer a and b.

Examples
```r
x <- 14000:0:10
print(dpois(x, 1000), digits = 18)  # standard R's double precision
dpois(mpfr(x, 120), 1000)  # more accuracy for the same
```

Description
Special Mathematical Functions, supported by the MPFR Library.

Usage
```r
zeta(x)
Ei(x)
Li2(x)
erf(x)
erfc(x)
```

Arguments
```r
x            a numeric or mpfr vector.
```

Details

zeta(x) computes Riemann’s Zeta function \( \zeta(x) \) important in analytical number theory and related fields. The traditional definition is

\[
\zeta(x) = \sum_{n=1}^{\infty} \frac{1}{n^x}.
\]

Ei(x) computes the exponential integral,

\[
\int_{-\infty}^{x} \frac{e^t}{t} dt.
\]
Li2(x) computes the dilogarithm,
\[ \int_0^x \frac{-\log(1-t)}{t} \, dt. \]

erf(x) and erfc(x) are the error, respectively complementary error function which are both reparametrizations of \( \text{pnorm} \), \( \text{erf}(x) = 2 \times \text{pnorm}(\sqrt{2} \times x) \) and \( \text{erfc}(x) = 2 \times \text{pnorm}(\sqrt{2} \times x, \text{lower}=\text{FALSE}) \), and hence \texttt{Rmpfr} provides its own version of \texttt{pnorm}.

Value
A vector of the same length as \( x \), of class \texttt{mpfr}.

See Also
\texttt{pnorm} in standard package \texttt{stats}; the class description \texttt{mpfr} mentioning the generic arithmetic and mathematical functions (\( \sin \), \( \log \), ..., etc) for which “\texttt{mpfr}” methods are available.

Examples
```r
curve(El, 0, 5, n=2001)
if(mpfrVersion() >= "2.4.0") {  ## Li2() is not available in older MPFR versions
  curve(Li2, 0, 5, n=2001)
  curve(Li2, -2, 13, n=2000); abline(h=0,v=0, lty=3)
  curve(Li2, -200, 400, n=2000); abline(h=0,v=0, lty=3)
}
curve(ef, -3,3, col = "red", ylim = c(-1,2))
curve(erc, add = TRUE, col = "blue")
abline(h=0, v=0, lty=3)
legend(-3,1, c("erf(x)", "erfc(x)"), col = c("red","blue"), lty=1)
```

Description
This page documents utilities from package \texttt{Rmpfr} which are typically not called by the user, but may come handy in some situations.

Usage
```r
getPrec(x, base = 10, doNumeric = TRUE, is.mpfr = NA, bigq. = 128L)
getD(x)
mpfr_default_prec(prec)
## S3 method for class 'mpfrArray'
print(x, digits = NULL, drop0trailing = FALSE)
```
right = TRUE, ...)
# S3 method for class 'mpfr'
print(x, digits = NULL, drop0trailing = TRUE, 
   right = TRUE, ...)
toNum(from, rnd.mode = c('N', 'D', 'U', 'Z', 'A'))
mpfr2array(x, dim, dimnames = NULL, check = FALSE)

mpfrXport(x)
mpfrImport(mxp)

Arguments

x, from typically, an R object of class "mpfr", or "mpfrArray", respectively. For getPrec(),
any number-like R object, or NULL.
base (only when x is character) the base with respect to which x[i] represent num-
bers; base b must fulfill 2 ≤ b ≤ 62.
doNumeric logical indicating integer or double typed x should be accepted and a default
precision be returned. Should typically be kept at default TRUE.
is.mpfr logical indicating if class(x) is already known to be "mpfr"; typically should
be kept at default, NA.
bigq. for getPrec(), the precision to use for a big rational (class "bigq"); if not
specified gives warning when used.
prec a positive integer, or missing.
drop0trailing logical indicating if trailing "0"s should be omitted.
right logical indicating print()ing should right justify the strings; see print.default()
to which it is passed.
digits, ... further arguments to print methods.
rnd.mode a 1-letter string specifying how rounding should happen at C-level conversion
to MPFR, see details of mpfr.
dim, dimnames for "mpfrArray" construction.
check logical indicating if the mpfrArray construction should happen with internal
safety check. Previously, the implicit default used to be true.
mxp an "mpfrXport" object, as resulting from mpfrXport().

Details

The print method is currently built on the format method for class mpfr. This, currently does not
format columns jointly which leads to suboptimally looking output. There are plans to change this.

Value

getPrec(x) returns a integer vector of the same length as x when that is positive, whereas
getPrec(NULL) returns mpfr_default_prec(), see below.
If you need to change the precision of x, i.e., need something like “setPrec”, use roundMpfr().
getD(x) is intended to be a fast version of x@.Data, and should not be used outside of lower level functions.

mpfr_default_prec() returns the current MPFR default precision, an integer. This is currently not made use of, in all of package Rmpfr, where functions have their own default precision where needed.

mpfr_default_prec(prec) sets the current MPFR default precision and returns the previous one; see above.

toNum(m) returns a numeric array or matrix, when m is of class "mpfrArray" or "mpfrMatrix", respectively. It should be equivalent to as(m, "array") or ... "matrix". Note that the slightly more general asNumeric() is preferred now.

mpfr2array() a slightly more flexible alternative to dim(.) <- dd.

Note

mpfrXport() and mpfrImport() are experimental and used to explore reported platform incompatibilities of save()d and load()ed "mpfr" objects between Windows and non-Windows platforms.

In other words, the format of the result of mpfrXport() and hence the mxp argument to mpfrImport() are considered internal, not part of the API and subject to change.

See Also

Start using mpfr(.), and compute with these numbers.

mpfrArray(x) is for numeric ("non-mpfr") x, whereas mpfr2array(x) is for "mpfr" classed x, only.

Examples

getPrec(as(c(1,pi), "mpfr")) # 128 for both

(opr <- mpfr_default_prec()) ## typically 53, the MPFR system default
stopifnot(opr == (opr <- mpfr_default_prec(70)),
70 == mpfr_default_prec())
## and reset it:
mpfr_default_prec(opr)

## Explore behavior of rounding modes 'rnd.mode':
x <- mpfr(10.99)^512 # too large for regular (double prec. / numeric):
  sapply(c("N", "D", "U", "Z", "A"), function(RM)
        sapply(list(-x,x), function(.) toNum(.), RM)))
##       N     D     U     Z     A
##  -Inf -Inf -1.797693e+308 -Inf -1.797693e+308
##  Inf 1.797693e+308   Inf   Inf

## Printing of "MPFR" matrices is less nice than R's usual matrix printing:
m <- outer(c(1, 3.14, -1024.5678), c(1, 1e-3, 10,100))
m[3,3] <- round(m[3,3])
m mpfr(m, 50)
mpfr.utils

Description

mpfrVersion() returns the version of the MPFR library which Rmpfr is currently linked to.

c(x, y, ...) can be used to combine MPFR numbers in the same way as regular numbers IFF the first argument x is of class mpfr.

mpfrIs0(.) uses the MPFR library in the documented way to check if (a vector of) MPFR numbers are zero. It was called mpfr.is.0 which is strongly deprecated now.

.mpfr.is.whole(x) uses the MPFR library in the documented way to check if (a vector of) MPFR numbers is integer valued. This is equivalent to x == round(x), but not at all to is.integer(as(x, "numeric")). You should typically rather use (the "mpfr" method of the generic function) is.whole(x) instead. The former name mpfr.is.integer is deprecated now.

Usage

mpfrVersion()
mpfrIs0(x)

## S3 method for class 'mpfr'
c(...)

## S3 method for class 'mpfr'
diff(x, lag = 1L, differences = 1L, ...)

```r
B6 <- mpfr2array(Bernoulli(1:6, 60), c(2,3),
                  dimnames = list(LETTERS[1:2], letters[1:3]))
B6

## Looking at internal representation [for power users only!]:
i8 <- mpfr(-2:5, 32)
x4 <- mpfr(c(NA, NaN, -Inf, Inf), 32)
## The output of the following depends on the GMP "numb" size
## (32 bit vs. 64 bit), and may be even more platform specifics:
str( .mpfr2list(i8) )
str( .mpfr2list(x4) )

str(xp4 <- mpfrXport(x4))
stopifnot(identical(x4, mpfrImport(mpfrXport(x4))),
          identical(i8, mpfrImport(mpfrXport(i8))))
if(FALSE) ##FIXME: not yet working:
  stopifnot(identical(B6, mpfrImport(mpfrXport(B6))))
```
Arguments

- `x`: an object of class `mpfr`.
- `lag`, differences:
  - for `diff()`: exact same meaning as in `diff()`'s default method, `diff.default`.

Value

- `mpfrIs0` returns a logical vector of length `length(x)` with values `TRUE` iff the corresponding `x[i]` is an MPFR representation of zero (0).
- `mpfrVersion` returns an object of S3 class "numeric_version", so it can be used in comparisons.
- The other functions return MPFR number (vectors), i.e., extending class `mpfr`.

See Also

- `str.mpfr` for the `str` method, `erf` for special mathematical functions on MPFR.

Examples

```r
mpfrVersion()
(x <- c(Const("pi", 64), mpfr(-2:2, 64)))
mpfrIs0(x)  # one of them is
x[mpfrIs0(x)]  # but it may not have been obvious..
str(x)

x <- rep(-2:2, 5)
stopifnot(is.whole(mpfr(2, 500) ^ (1:200)),
  all.equal(diff(x), diff(as.numeric(x))))
```

Description

Utility to construct an R object of class `mpfrArray`, very analogously to the numeric `array` function.

Usage

```r
mpfrArray(x, precBits, dim = length(x), dimnames = NULL,
  rnd.mode = c("N","D","U","Z","A"))
```
Arguments

x   numeric(vector), typically of length prod(dim) or shorter in which case it is recycled.

precBits  a number, the maximal precision to be used, in bits; i.e., 53 corresponds to double precision. Must be at least 2.

dim  the dimension of the array to be created, that is a vector of length one or more giving the maximal indices in each dimension.

dimnames  either NULL or the names for the dimensions. This is a list with one component for each dimension, either NULL or a character vector of the length given by dim for that dimension.

rnd.mode  a 1-letter string specifying how rounding should happen at C-level conversion to MPFR, see details of mpfr.

Value

an object of class "mpfrArray", specifically "mpfrMatrix" when length(dim) == 2.

See Also

mpfr, array; asNumeric() as “inverse” of mpfrArray(), to get back a numeric array.

mpfr2array(x) is for "mpfr" classed x, only, whereas mpfrArray(x) is for numeric ("non-mpfr") x.

Examples

## preallocating is possible here too
ma <- mpfrArray(NA, prec = 80, dim = 2:4)
validObject(A2 <- mpfrArray(1:24, prec = 64, dim = 2:4))

## recycles, gives an "mpfrMatrix" and dimnames :
mat <- mpfrArray(1:5, 64, dim = c(5,3), dimnames=list(NULL, letters[1:3]))
mat
asNumeric(mat)
stopifnot(identical(asNumeric(mat),
                     matrix(1:5 +0, 5,3, dimnames=dimnames(mat))))

## Testing the apply() method :
apply(mat, 2, range)
apply(A2, 1:2, range)
apply(A2, 2:3, max)
apply(A2, 2, fivenum)
stopifnot(as(apply(A2, 2, range), "matrix") ==
          apply(as(A2,"array"), 2, range))
**mpfrMatrix**  
*Classes "mpfrMatrix" and "mpfrArray"*

**Description**

The classes "mpfrMatrix" and "mpfrArray" are, analogously to the base matrix and array functions and classes simply “numbers” of class mpfr with an additional Dim and Dimnames slot.

**Objects from the Class**

Objects should typically be created by mpfrarray(), but can also be created by new("mpfrMatrix", ...) or new("mpfrArray", ...), or also by t(x), dim(x) <- dd, or mpfr2array(x, dim=dd) where x is a an mpfr “number vector”.

A (slightly more flexible) alternative to dim(x) <- dd is mpfr2array(x, dd, dimnames).

**Slots**

`.Data`: as for the mpfr class, a "list" of mpfr numbers.

`Dim`: of class “integer”, specifying the array dimension.

`Dimnames`: of class "list" and the same length as Dim, each list component either NULL or a character vector of length Dim[j].

**Extends**

Class "mpfrMatrix" extends "mpfrArray", directly.

Class "mpfrArray" extends class "mpfr", by class "mpfrArray", distance 2; class "list", by class "mpfrArray", distance 3; class "vector", by class "mpfrArray", distance 4.

**Methods**

Arith signature(e1 = "mpfr", e2 = "mpfrArray"): ...

Arith signature(e1 = "numeric", e2 = "mpfrArray"): ...

Arith signature(e1 = "mpfrArray", e2 = "mpfrArray"): ...

Arith signature(e1 = "mpfrArray", e2 = "mpfr"): ...

Arith signature(e1 = "mpfrArray", e2 = "numeric"): ...

as.vector signature(x = "mpfrArray", mode = "missing"): drops the dimension ‘attribute’, i.e., transforms x into a simple mpfr vector. This is an inverse of t(.) or dim(.) <- * on such a vector.

atan2 signature(y = "ANY", x = "mpfrArray"): ...

atan2 signature(y = "mpfrArray", x = "mpfrArray"): ...

atan2 signature(y = "mpfrArray", x = "ANY"): ...

[<- signature(x = "mpfrArray", i = "ANY", j = "ANY", value = "ANY"): ...

[ signature(x = "mpfrArray", i = "ANY", j = "ANY", drop = "ANY"): ...
mpfrMatrix

[ signature(x = "mpfrArray", i = "ANY", j = "missing", drop = "missing"): "mpfrArray"s can be subset ("indexed") as regular R arrays.

%*% signature(x = "mpfr", y = "mpfrMatrix"): Compute the matrix/vector product \( xy \) when the dimensions (\( \text{dim} \)) of \( x \) and \( y \) match. If \( x \) is not a matrix, it is treated as a 1-row or 1-column matrix (aka "row vector" or "column vector") depending on which one makes sense, see the documentation of the base function %*%.

%*% signature(x = "mpfr", y = "Mnumber"): method definition for cases with one mpfr and any "number-like" argument are to use MPFR arithmetic as well.

%*% signature(x = "mpfrMatrix", y = "mpfrMatrix").

%*% signature(x = "mpfrMatrix", y = "mpfr"). etc. Further method definitions with identical semantic.

crossprod signature(x = "mpfr", y = "missing"): Computes \( x'x \), i.e., \( t(x) \) %*% x, typically more efficiently.

crossprod signature(x = "mpfr", y = "mpfrMatrix"): Computes \( x'y \), i.e., t(x) %*% y, typically more efficiently.

crossprod signature(x = "mpfrMatrix", y = "mpfrMatrix"): ...

crossprod signature(x = "mpfrMatrix", y = "mpfr"): ...
Author(s)
Martin Maechler

See Also

`mpfrArray`, also for more examples.

Examples

```r
showClass("mpfrMatrix")
validObject(mm <- new("mpfrMatrix"))
validObject(aa <- new("mpfrArray"))

v6 <- mpfr(1:6, 128)
m6 <- new("mpfrMatrix", v6, Dim = c(2L, 3L))
validObject(m6)
m6
which(m6 == 3, arr.ind = TRUE) # |--> (1, 2)
## Coercion back to "vector": Both of these work:
stopifnot(identical(as(m6, "mpfr"), v6),
           identical(as.vector(m6), v6)) # < but this is a "coincidence"

S2 <- m6[,-3] # 2 x 2
S3 <- rbind(m6, c(1:2,10)); s3 <- asNumeric(S3)
det(S2)
str(determinant(S2))
det(S3)
stopifnot(all.equal(det(S2), det(asNumeric(S2)), tol=1e-15),
           all.equal(det(S3), det(s3), tol=1e-15))

## 2-column matrix indexing and replacement:
(sS <- S3[i2 <- cbind(1:2, 2:3)])
stopifnot(identical(asNumeric(sS), s3[i2]))
C3 <- S3; c3 <- s3
C3[i2] <- 10:11
c3[i2] <- 10:11
stopifnot(identical(asNumeric(C3), c3))

AA <- new("mpfrArray", as.vector(cbind(S3, -S3)), Dim=c(3L,3:2))
stopifnot(identical(AA[,1], S3), identical(AA[,2], -S3))
aa <- asNumeric(AA)

i3 <- cbind(3:1, 1:3, c(2L, 1:2))
i13 <- Rmpfr:::mat2ind(i3, dim(AA), dimnames(AA))
stopifnot(aa[i3] == new("mpfr", getD(AA[i13])))
stopifnot(identical(aa[i3], asNumeric(AA[i13])))
CA <- AA; ca <- aa
ca[i3] <- ca[i3] * 3
CA[i3] <- CA[i3] * 3

## scale():
```
S2. <- scale(S2)
stopifnot(all.equal(abs(as.vector(S2.)), rep(sqrt(1/mpfr(2, 128)), 4),
        tol = 1e-30))

## norm():
norm(S2)
stopifnot(identical(norm(S2), norm(S2, "1")),
        norm(S2, "1") == 6,
        norm(S2, "\text{\texttt{I}}") == 4,
        abs(norm(S2, "\text{\texttt{F}}") - 5.477225575051661) < 1e-15)

### mpfrMatrix-utils

#### Functions for mpfrMatrix Objects

#### Description

determinant(x, ..) computes the determinant of the mpfr square matrix x. May work via coercion to "numeric", i.e., compute determinant(asNumeric(x), logarithm), if asNumeric is true, by default, if the dimension is larger than three. Otherwise, use precision precBits for the "accumulator" of the result, and use the recursive mathematical definition of the determinant (with computational complexity \(n!\), where \(n\) is the matrix dimension, i.e., very inefficient for all but small matrices!)

#### Usage

```r
## S3 method for class 'mpfrMatrix'
determinant(x, logarithm = TRUE,
        asNumeric = (d[1] > 3), precBits = max(.getPrec(x)), ...)
```

#### Arguments

- **x**: an mpfrMatrix object of square dimension.
- **logarithm**: logical indicating if the log of the absolute determinant should be returned.
- **asNumeric**: logical .. if rather determinant(asNumeric(x), ...) should be computed.
- **precBits**: the number of binary digits for the result (and the intermediate accumulations).
- ... unused (potentially further arguments passed to methods).

#### Value

as determinant(), an object of S3 class "det", a list with components

- **modulus**: the (logarithm of) the absolute value (abs) of the determinant of x.
- **sign**: the sign of the determinant.

#### Author(s)

Martin Maechler
See Also
determinant in base R, which relies on a fast LU decomposition. mpfrMatrix

Examples

```r
m6 <- mpfrArray(1:6, prec=128, dim = c(2L, 3L))
m6
S2 <- m6[,-3] # 2 x 2
S3 <- rbind(m6, c(1:2,10))
det(S2)
str(determinant(S2))
det(S3)
stopifnot(all.equal(det(S2), det(asNumeric(S2)), tolerance=1e-15),
  all.equal(det(S3), det(asNumeric(S3)), tolerance=1e-15))
```

---

optimizeR

**High Precision One-Dimensional Optimization**

**Description**

optimizeR searches the interval from lower to upper for a minimum of the function f with respect to its first argument.

**Usage**

```r
optimizeR(f, lower, upper, ..., tol = 1e-20,
  method = c("Brent", "GoldenRatio"),
  maximum = FALSE,
  precFactor = 2.0, precBits = -log2(tol) * precFactor,
  maxiter = 1000, trace = FALSE)
```

**Arguments**

- **f** the function to be optimized. f(x) must work “in Rmpfr arithmetic” for optimizer() to make sense. The function is either minimized or maximized over its first argument depending on the value of maximum.
- **...** additional named or unnamed arguments to be passed to f.
- **lower** the lower end point of the interval to be searched.
- **upper** the upper end point of the interval to be searched.
- **tol** the desired accuracy, typically higher than double precision, i.e., tol < 2e-16.
- **method** character string specifying the optimization method.
- **maximum** logical indicating if f() should be maximized or minimized (the default).
- **precFactor** only for default precBits construction: a factor to multiply with the number of bits directly needed for tol.
- **precBits** number of bits to be used for mpfr numbers used internally.
maxiter maximal number of iterations to be used.
trace integer or logical indicating if and how iterations should be monitored; if an integer $k$, print every $k$-th iteration.

Details

"Brent": Brent(1973)'s simple and robust algorithm is a hybrid, using a combination of the golden ratio and local quadratic ("parabolic") interpolation. This is the same algorithm as standard R’s optimize(), adapted to high precision numbers.
In smooth cases, the convergence is considerably faster than the golden section or Fibonacci ratio algorithms.

"GoldenRatio": The golden ratio method works as follows: from a given interval containing the solution, it constructs the next point in the golden ratio between the interval boundaries.

Value

A list with components minimum (or maximum) and objective which give the location of the minimum (or maximum) and the value of the function at that point; iter specifying the number of iterations, the logical convergence indicating if the iterations converged and estimPrec which is an estimate or an upper bound of the final precision (in $x$). method the string of the method used.

Author(s)

"GoldenRatio" is based on Hans W Borchert’s golden_ratio; modifications and "Brent" by Martin Maechler.

See Also

R’s standard optimize; Rmpfr’s unrootR.

Examples

iG5 <- function(x) -exp(-(x-5)^2/2)
curve(iG5, 0, 10, 200)
o.dp <- optimize(iG5, c(0, 10)) #-> 5 of course
o.M.gs <- optimizeR(iG5, 0, 10, method="Golden")
o.M.Br <- optimizeR(iG5, 0, 10, method="Brent", trace=TRUE)
oM.gs$min ; oM.gs$iter
oM.Br$min ; oM.Br$iter
(doExtras <- Rmpfr::doExtras())
if(doExtras) {## more accuracy (takes a few seconds)
oM.gs <- optimizeR(iG5, 0, 10, method="Golden", tol = 1e-70)
oM.Br <- optimizeR(iG5, 0, 10, tol = 1e-70)
}
rbind(Golden = c(err = as.numeric(oM.gs$min -5), iter = oM.gs$iter),
      Brent = c(err = as.numeric(oM.Br$min -5), iter = oM.Br$iter))

## == Brent is orders of magnitude more efficient!

## Testing on the sine curve with 40 correct digits:
pbetaI

Accurate Incomplete Beta / Beta Probabilities For Integer Shapes

Description

For integers a, b, I_x(a, b) aka pbeta(x, a, b) is a polynomial in x with rational coefficients, and hence arbitrarily accurately computable.
Usage

\[ pbetaI(q, \text{shape1}, \text{shape2}, \text{ncp} = 0, \text{lower.tail} = \text{TRUE}, \text{log.p} = \text{FALSE}, \text{precBits} = \text{NULL}, \text{rnd.mode} = \text{c("N","D","U","Z","A")}) \]

Arguments

- **q**: called \( x \), above; vector of quantiles, in \([0, 1]\).
- **shape1**, **shape2**: the positive Beta “shape” parameters, called \( a, b \), above. **Must** be integer valued for this function.
- **ncp**: unused, only for compatibility with \( \text{pbeta} \), must be kept at its default, 0.
- **lower.tail**: logical; if \( \text{TRUE} \) (default), probabilities are \( P[X \leq x] \), otherwise, \( P[X > x] \).
- **log.p**: logical; if \( \text{TRUE} \), probabilities \( p \) are given as \( \log(p) \).
- **precBits**: the precision (in number of bits) to be used in \( \text{sumBinomMpfr}() \).
- **rnd.mode**: a 1-letter string specifying how rounding should happen at C-level conversion to MPFR, see \( \text{mpfr} \).

Value

an "mpfr" vector of the same length as \( q \).

Note

For upper tail probabilities, i.e., when \( \text{lower.tail} = \text{FALSE} \), we may need large \( \text{precBits} \), because the implicit or explicit \( 1 - P \) computation suffers from severe cancellation.

Author(s)

Martin Maechler

See Also

\( \text{pbeta}, \text{sumBinomMpfr} \), \text{chooseZ}.

Examples

```
x <- (0:12)/16 # not all the way up ..
a <- 7; b <- 788

p. <- pbetaI(x, a, b) ## still slow: % TOO slow -- FIXME
pp <- pbetaI(x, a, b, precBits = 2048)
## Currently, the lower.tail=FALSE are computed "badly":
lp <- log(pp)  ## = pbetaI(x, a, b, log.p=TRUE)
lp <- log1p(-pp)  ## = pbetaI(x, a, b, lower.tail=FALSE, log.p=TRUE)
lp <- 1 - pp  ## = pbetaI(x, a, b, lower.tail=FALSE)

if(Rmpfr:::doExtras()) {  ## somewhat slow
  stopifnot(
    all.equal(lp, pbetaI(x, a, b, precBits = 2048, log.p=TRUE)),
    all.equal(lp, pbetaI(x, a, b, precBits = 2048, lower.tail=FALSE, log.p=TRUE)),
  )
}
```
pmax

Parallel Maxima and Minima

Description

Returns the parallel maxima and minima of the input values.

The functions \texttt{pmin} and \texttt{pmax} have been made S4 generics, and this page documents the “... method for class "mNumber””, i.e., for arguments that are numeric or from \texttt{class "mpfr"}.

Usage

\begin{verbatim}
pmax(..., na.rm = FALSE)
pmin(..., na.rm = FALSE)
\end{verbatim}

Arguments

\begin{verbatim}
... numeric or arbitrary precision numbers (class \texttt{mpfr}).
na.rm a logical indicating whether missing values should be removed.
\end{verbatim}

Details

See \texttt{pmax}, the documentation of the base functions, i.e., default methods.
Value

vector-like, of length the longest of the input vectors; typically of class mpfr, for the methods here.

Methods

... = "ANY" the default method, really just base::min or base::max, respectively.
... = "mNumber" the method for mpfr arguments, mixed with numbers; designed to follow the same semantic as the default method.

See Also

The documentation of the base functions, pmin and pmax; also min and max; further, range (both min and max).

Examples

```r
p <- pmin(1.35, mpfr(0:10, 77))
stopifnot(p == pmin(1.35, 0:10))
```

---

roundMpfr Rounding to Binary bits, "mpfr-internally"

Description

Rounding to binary bits, not decimal digits. Closer to the number representation, this also allows to increase or decrease a number's precBits. In other words, it acts as setPrec(), see getPrec().

Usage

```r
roundMpfr(x, precBits, rnd.mode = c("N","D","U","Z","A"))
```

Arguments

- `x` an mpfr number (vector)
- `precBits` integer specifying the desired precision in bits.
- `rnd.mode` a 1-letter string specifying how rounding should happen at C-level conversion to MPFR, see mpfr.

Value

an mpfr number as x but with the new 'precBits' precision

See Also

The mpfr class group method Math2 implements a method for round(x, digits) which rounds to decimal digits.
Examples

```r
(p1 <- Const("pi", 100)) # 100 bit prec
roundMpfr(p1, 120) # 20 bits more, but "random noise"
Const("pi", 120) # same "precision", but really precise
```

Description

Generate `regular`, i.e., arithmetic sequences. This is in lieu of methods for `seq` (dispatching on all three of `from`, `to`, and `by`).

Usage

```r
seqmpfr(from = 1, to = 1, by = ((to - from)/(length.out - 1)),
length.out = NULL, along.with = NULL, ...)
```

Arguments

- `from, to`: the starting and (maximal) end value (numeric or "mpfr") of the sequence.
- `by`: number (numeric or "mpfr"): increment of the sequence.
- `length.out`: desired length of the sequence. A non-negative number, which will be rounded up if fractional.
- `along.with`: take the length from the length of this argument.
- `...`: arguments passed to or from methods.

Details

see `seq` (default method in package `base`), whose semantic we want to replicate (almost).

Value

a `vector` of class "mpfr", when one of the first three arguments was.

Author(s)

Martin Maechler

See Also

The documentation of the `base` function `seq; mpfr`
Examples

```r
seqMpfr(0, 1, by = mpfr(0.25, prec=88))
seqMpfr(7, 3) # -> default prec.
```

Description

The `str` method for objects of class `mpfr` produces a bit more useful output than the default method `str.default`.

Usage

```r
## S3 method for class 'mpfr'
str(object, nest.lev, give.head=TRUE, digits.d = 12, vec.len = NULL, drop0trailing=TRUE, 
width = getOption("width"), ...)
```

Arguments

- `object`: an object of class `mpfr`.
- `nest.lev`: for `str()`, typically only used when called by a higher level `str()`.
- `give.head`: logical indicating if the “header” should be printed.
- `digits.d`: the number of digits to be used, will be passed `formatMpfr()` and hence NULL will use “as many as needed”, i.e. often too many. If this is a number, as per default, less digits will be used in case the precision (`getPrec(object)`) is smaller.
- `vec.len`: the number of `elements` that will be shown. The default depends on the precision of `object` and `width` (since `Rmpfr` 0.6-0, it was 3 previously).
- `drop0trailing`: logical, passed to `formatMpfr()` (with a different default here).
- `width`: the (approximately) desired width of output, see `options(width = .)`.
- `...`: further arguments, passed to `formatMpfr()`.

See Also

`mpfrIs0` for many more utilities.
Examples

```r
(x <- c(Const("pi", 64), mpfr(-2:2, 64)))
str(x)
str(list(pi = pi, x.mpfr = x))
str(x ^ 1000)
str(x ^ -1e4, digits=NULL) # full precision

uu <- Const("pi", 16)# unaccurate
str(uu) # not using default of 12 digits
```

**sumBinomMpfr**

(Alternating) Binomial Sums via Rmpfr

Description

Compute (alternating) binomial sums via high-precision arithmetic. If $sBn(f, n) := \text{sumBinomMpfr}(n, f)$, (default alternating is true, and $n0 = 0$),

$$sBn(f, n) = \sum_{k=n0}^{n} (-1)^{(n-k)} \binom{n}{k} \cdot f(k) = \Delta^n f,$$

see Details for the $n$-th forward difference operator $\Delta^n f$. If alternating is false, the $(-1)^{(n-k)}$ factor is dropped (or replaced by 1) above.

Such sums appear in different contexts and are typically challenging, i.e., currently impossible, to evaluate reliably as soon as $n$ is larger than around 50 $-$ 70.

Usage

```r
sumBinomMpfr(n, f, n0 = 0, alternating = TRUE, precBits = 256, 
               f.k = f(mpfr(k, precBits=precBits)))
```

Arguments

- **n**
  - upper summation index (integer).
- **f**
  - function to be evaluated at $k$ for $k$ in $n0:n$ (and which must return one value per $k$).
- **n0**
  - lower summation index, typically 0 (= default) or 1.
- **alternating**
  - logical indicating if the sum is alternating, see below.
- **precBits**
  - the number of bits for MPFR precision, see mpfr.
- **f.k**
  - can be specified instead of f and precBits, and must contain the equivalent of its default, $f(mpfr(k, precBits=precBits))$. 

Details

The alternating binomial sum \( sB(f, n) := \text{sumBinom}(n, f, n0 = 0) \) is equal to the \( n \)-th forward difference operator \( \Delta^n f \),

\[
sB(f, n) = \Delta^n f,
\]

where

\[
\Delta^n f = \sum_{k=0}^{n} (-1)^{n-k} \binom{n}{k} \cdot f(k),
\]

is the \( n \)-fold iterated forward difference \( \Delta f(x) = f(x + 1) - f(x) \) (for \( x = 0 \)).

The current implementation might be improved in the future, notably for the case where \( sB(f, n) = \text{sumBinomMpfr}(n, f, \ast) \) is to be computed for a whole sequence \( n = 1, \ldots, N \).

Value

an \texttt{mpfr} number of precision \texttt{precbits}. \texttt{s}. If alternating is true (as per default),

\[
s = \sum_{k=n0}^{n} (-1)^k \binom{n}{k} \cdot f(k),
\]

if alternating is false, the \( (-1)^k \) factor is dropped (or replaced by 1) above.

Author(s)

Martin Maechler, after conversations with Christophe Dutang.

References


See Also

\texttt{chooseMpfr}, \texttt{chooseZ} from package \texttt{gmp}.

Examples

```r
## "naive" R implementation:
sumBinom <- function(n, f, n0=0, ...) {
  k <- n0:n
  sum( choose(n, k) * (-1)^*(n-k) * f(k, ...))
}

## compute \texttt{sumBinomMpfr(.)} for a whole set of \'n\' values:
sumBin.all <- function(n, f, n0=0, precBits = 256, ...) {
  N <- length(n)
  precBits <- rep(precBits, length = N)
  ll <- lapply(seq_len(N), function(i)
```
unirootR

One Dimensional Root (Zero) Finding – in pure R

Description

The function unirootR searches the interval from lower to upper for a root (i.e., zero) of the function f with respect to its first argument.

unirootR() is “clone” of uniroot(), written entirely in R, in a way that it works with mpfr-numbers as well.

Usage

unirootR(f, interval, ..., lower = min(interval), upper = max(interval), f.lower = f(lower, ...), f.upper = f(upper, ...), verbose = FALSE, tol = .Machine$double.eps^0.25, maxiter = 1000, epsC = NULL)

Arguments

f the function for which the root is sought.
interval a vector containing the end-points of the interval to be searched for the root.
... additional named or unnamed arguments to be passed to f
lower, upper the lower and upper end points of the interval to be searched.
f.lower, f.upper the same as f(upper) and f(lower), respectively. Passing these values from the caller where they are often known is more economical as soon as f() contains non-trivial computations.
verbose logical (or integer) indicating if (and how much) verbose output should be produced during the iterations.
tol  the desired accuracy (convergence tolerance).
maxiter  the maximum number of iterations.
epsC  positive number or NULL in which case a smart default is sought. This should specify the “achievable machine precision” for the given numbers and their arithmetic.

The default will set this to \( \text{.Machine} \text{double} \cdot \text{eps} \) for double precision numbers, and will basically use \( 2^{-\min(\text{getPrec}(f.\text{lower}), \text{getPrec}(f.\text{upper}))} \) when that works (as, e.g., for \text{mpfr}-numbers) otherwise. This is factually a lower bound for the achievable lower bound, and hence, setting tol smaller than epsC is typically non-sensical sense and produces a warning.

Details

Note that arguments after \( \ldots \) must be matched exactly.

Either interval or both lower and upper must be specified: the upper endpoint must be strictly larger than the lower endpoint. The function values at the endpoints must be of opposite signs (or zero).

The function only uses \( R \) code with basic arithmetic, such that it should also work with “generalized” numbers (such as \text{mpfr}-numbers) as long the necessary \text{Ops} methods are defined for those.

The underlying algorithm assumes a continuous function (which then is known to have at least one root in the interval).

Convergence is declared either if \( f(x) = 0 \) or the change in \( x \) for one step of the algorithm is less than tol (plus an allowance for representation error in \( x \)).

If the algorithm does not converge in maxiter steps, a warning is printed and the current approximation is returned.

\( f \) will be called as \( f(x, \ldots) \) for a (generalized) numeric value of \( x \).

Value

A list with four components: root and \( f.\text{root} \) give the location of the root and the value of the function evaluated at that point. iter and estim.prec give the number of iterations used and an approximate estimated precision for root. (If the root occurs at one of the endpoints, the estimated precision is NA.)

Source

Based on \text{zeroin()} (in package \text{rootoned}) by John Nash who manually translated the C code in \( R \)’s \text{zeroin.c} and on \text{uniroot()} in \( R \)’s sources.

References

Brent, R. (1973), see \text{uniroot}.

See Also

\text{polyroot} for all complex roots of a polynomial; \text{optimize, nlm}.
Examples

```r
require(utils) # for str

## some platforms hit zero exactly on the first step:
## if so the estimated precision is 2/3.

f <- function(x, a) x - a
str(xmin <- unirootR(f, c(0, 1), tol = 0.0001, a = 1/3))

## handheld calculator example: fixpoint of cos(.):
rc <- unirootR(function(x) cos(x) - x, lower=-pi, upper=pi, tol = 1e-9)
rc$root

## the same with much higher precision:
rcM <- unirootR(function(x) cos(x) - x,
                interval= mpfr(c(-3,3), 300), tol = 1e-40)
rcM
x0 <- rcM$root
stopifnot(all.equal(cos(x0), x0,
                   tol = 1e-40))## 40 digits accurate!

str(unirootR(function(x) x*(x^2-1) + .5, lower = -2, upper = 2,
             tol = 0.0001), digits.d = 10)
str(unirootR(function(x) x*(x^2-1) + .5, lower = -2, upper = 2,
             tol = 1e-10 ), digits.d = 10)

## A sign change of f(.), but not a zero but rather a "pole":
tan <- function(x) tan(x * (Const("pi",200)/180))
## tan( <angle> )
(rtan <- unirootR(tan., interval = mpfr(c(80,100), 200), tol = 1e-40))
## finds 90 ("ok"), and now gives a warning

## Find the smallest value x for which exp(x) > 0 (numerically):
r <- unirootR(function(x) 1e80*exp(x)-1e-300, c(-1000,0), tol = 1e-15)
str(r, digits.d = 15) ##> around -745, depending on the platform.

exp(r$root)  # = 0, but not for r$root * 0.999...
minexp <- r$root * (1 - 10*.Machine$double.eps)
exp(minexp)  # typically denormalized

## --- using mpfr-numbers :

## Find the smallest value x for which exp(x) > 0 ("numerically");
## Note that mpfr-numbers underflow *MUCH* later than doubles:
## one of the smallest mpfr-numbers (see also ?mpfr-class ):
(ep.M <- mpfr(2, 55) ^ - ((2^30 + 1) * (1 - 1e-15)))
r <- unirootR(function(x) 1e99* exp(x) - ep.M, mpfr(c(-1e20, 0), 200))
r # 97 iterations; f.root is very similar to ep.M
```
Index

*Topic arith
Bernoulli, 7
chooseM mpfr, 10
factorialM mpfr, 11
formatHex, 13
gmp-conversions, 17
mpfr-utils, 37
pbetaI, 46
pmax, 48
roundM mpfr, 49
sumBinomM mpfr, 52

*Topic array
mpfrArray, 38
mpfrMatrix-utils, 43

*Topic character
formatM mpfr, 15

*Topic classes
array_or_vector-class, 4
atomicVector-class, 6
M number-class, 24
mpfr, 25
mpfr-class, 27
mpfrMatrix, 40

*Topic distribution
mpfr-distr-etc, 32
pbetaI, 46

*Topic manip
seqM mpfr, 50

*Topic math
Bessel_mpfr, 8
integrateR, 20
is.whole, 23
mpfr-special-functions, 33

*Topic methods
asNumeric-methods, 5
bind-methods, 9
pmax, 48

*Topic optimize
hjkM mpfr, 18

optimizeR, 44
unirootR, 54

*Topic package
Rmpfr-package, 2

*Topic print
formatM mpfr, 15

*Topic univar
pmax, 48

*Topic utilities
integrateR, 20
mpfr-utils, 34
str.M mpfr, 51

Machine, 30, 55
.bigq2mpfr (gmp-conversions), 17
.bigz2mpfr (gmp-conversions), 17
.mpfr.is.whole (mpfr-utils), 37
.mpfr2bigz (gmp-conversions), 17
.mpfr2list (mpfr-utils), 34

[ , mpfr, ANY, missing, missing-method
(mpfr-class), 27

[ , mpfrArray, ANY, ANY, ANY-method
 (mpfrMatrix), 40

[ , mpfrArray, ANY, missing, missing-method
(mpfrMatrix), 40

[ , mpfrArray, matrix, missing, missing-method
(mpfrMatrix), 40

[ , mpfr, ANY, missing, ANY-method
(mpfr-class), 27

[ , mpfr, ANY, missing, mpfr-method
(mpfr-class), 27

[ , mpfr, missing, missing, ANY-method
(mpfr-class), 27

[ , mpfrArray, ANY, ANY, ANY-method
(mpfrMatrix), 40

[ , mpfrArray, ANY, ANY, mpfr-method
(mpfrMatrix), 40

[ , mpfrArray, ANY, missing, ANY-method
(mpfrMatrix), 40

[ , mpfrArray, ANY, missing, mpfr-method
(mpfrMatrix), 40

57
(mpfrMatrix), 40
[<-, mpfrArray, matrix, missing, ANY-method
(mpfrMatrix), 40
[<-, mpfrArray, matrix, missing, mpfr-method
(mpfrMatrix), 40
[<-, mpfrArray, missing, ANY, ANY-method
(mpfrMatrix), 40
[<-, mpfrArray, missing, ANY, mpfr-method
(mpfrMatrix), 40
[<-, mpfrArray, missing, ANY, mpfr-method
(mpfrMatrix), 40
[<-, mpfrArray, missing, missing, ANY-method
(mpfrMatrix), 40
[<-, mpfrArray, missing, missing, mpfr-method
(mpfrMatrix), 40
[[, mpfr-method (mpfr-class), 27
%%, Mnumber, mpfr-method (mpfrMatrix), 40
%%, array_or_vector, mpfr-method
(mpfr-class), 27
%%, mpfr, Mnumber-method (mpfrMatrix), 40
%%, mpfr, array_or_vector-method
(mpfr-class), 27
%%, mpfr, mpfr-method (mpfrMatrix), 40
%%, mpfr, mpfrMatrix-method
(mpfrMatrix), 40
%%, mpfrMatrix, mpfr-method
(mpfrMatrix), 40
%%, mpfrMatrix, mpfrMatrix-method
(mpfrMatrix), 40
%%., 41
abs, 29, 43
abs, mpfr-method (mpfr-class), 27
acos, 29
acosh, 29
Ai (Bessel_mpf), 8
all, 29
all.equal, 30
all.equal, ANY, mpfr-method (mpfr-class), 27
all.equal, mpfr, ANY-method (mpfr-class), 27
all.equal, mpfr, mpfr-method
(mpfr-class), 27
any, 29
aperm, 41
aperm, mpfrArray-method (mpfrMatrix), 40
apply, mpfrArray-method (mpfrMatrix), 40
Arg, mpfr-method (mpfr-class), 27
Arith, array, mpfr-method (mpfr-class), 27
Arith, integer, mpfr-method (mpfr-class), 27
Arith, mpfr, array-method (mpfr-class), 27
Arith, mpfr, integer-method (mpfr-class), 27
Arith, mpfr, missing-method (mpfr-class), 27
Arith, mpfr, mpfr-method (mpfr-class), 27
Arith, mpfr, mpfrArray-method (mpfrMatrix), 40
Arith, mpfr, numeric-method (mpfr-class), 27
Arith, mpfrArray, mpfr-method
(mpfrMatrix), 40
Arith, mpfrArray, mpfrArray-method
(mpfrMatrix), 40
Arith, numeric, mpfr-method (mpfr-class), 27
Arith, numeric, mpfrArray-method
(mpfrMatrix), 40
array, 3, 5, 16, 25, 30, 36, 38–41
array_or_vector, 24
array_or_vector-class, 4
as, 30
as.bigq, 18
as.bigz, 17, 18
as.integer, mpfr-method (mpfr-class), 27
as.numeric, 5
as.numeric, mpfr-method (mpfr-class), 27
as.vector, mpfrArray-method
(mpfrMatrix), 40
as.vector, mpfrArray-method
(mpfr-class), 27
asin, 29
asinh, 29
asNumeric, 5, 26, 36, 39, 43
asNumeric, mpfr-method
(asNumeric-methods), 5
asNumeric, mpfrArray-method
(asNumeric-methods), 5
asNumeric-methods, 5
atan, 28, 29
atan2, ANY, mpfr-method (mpfr-class), 27
atan2, ANY, mpfrArray-method
(mpfr-class), 27
atan2, mpfr, ANY-method (mpfr-class), 27
atan2, mpfr, mpfr-method (mpfr-class), 27
atan2, numeric-method (mpfr-class), 27
atan2, mpfrArray, ANY-method (mpfr-class), 27
atan2, mpfrArray, mpfrArray-method (mpfr-class), 27
atan2, numeric, mpfr-method (mpfr-class), 27
atanh, 29
atomicVector-class, 6
base::pmin, 49
Bernoulli, 3, 7, 31
Bessel_mpfr, 3, 8
besselJ, 8
bessely, 8
beta, 28
beta, ANY, mpfr-method (mpfr-class), 27
beta, ANY, mpfrArray-method (mpfr-class), 27
beta, mpfr, ANY-method (mpfr-class), 27
beta, mpfr, mpfr-method (mpfr-class), 27
beta, mpfr, numeric-method (mpfr-class), 27
beta, mpfrArray, ANY-method (mpfr-class), 27
beta, mpfrArray, mpfrArray-method (mpfr-class), 27
beta, numeric, mpfr-method (mpfr-class), 27
bigq, 17, 25
bigrational, 4
bigz, 17, 25, 30
bind-methods, 9
c, 37
c.mpfr, 3
c.mpfr (mpfr.utils), 37
cbind, 3, 9
cbind (bind-methods), 9
cbind, ANY-method (bind-methods), 9
cbind, Mnumber-method (bind-methods), 9
cbind-methods (bind-methods), 9
cbind2, 9
ceiling, 29
character, 25, 26, 30, 35, 40, 44
choose, 10, 11
chooseMpfrr, 3, 10, 53
chooseZ, 10, 11, 47, 53
class, 6, 9, 14, 23, 27, 35, 48
coerce, array, mpfr-method (mpfr-class), 27
coerce, bigq, mpfr-method (gmp-conversions), 17
coerce, bigz, mpfr-method (gmp-conversions), 17
coerce, character, mpfr-method (mpfr-class), 27
coerce, integer, mpfr-method (mpfr-class), 27
coerce, logical, mpfr-method (mpfr-class), 27
coerce, mpfr, bigz-method (mpfr-class), 27
coerce, mpfr, character-method (mpfr-class), 27
coerce, mpfr, integer-method (mpfr-class), 27
coerce, mpfr, mpfr1-method (mpfr-class), 27
coerce, mpfr, numeric-method (mpfr-class), 27
coerce, mpfr1, mpfr-method (mpfr-class), 27
coerce, mpfr1, numeric-method (mpfr-class), 27
coerce, mpfrArray, array-method (mpfrMatrix), 40
coerce, mpfrArray, mpfr-method (mpfrMatrix), 40
coerce, mpfrArray, vector-method (mpfrMatrix), 40
coerce, mpfrMatrix, mpfr-method (mpfrMatrix), 40
coerce, numeric, mpfr-method (mpfr-class), 27
coerce, numeric, mpfr1-method (mpfr-class), 27
coerce, raw, mpfr-method (mpfr-class), 27
coerce<-, mpfr-method (mpfrMatrix), 40
colMeans, mpfrArray-method (mpfrMatrix), 40
colSums, mpfrArray-method (mpfrMatrix), 40
compareTo, array, mpfr-method (mpfr-class), 27
Compare, integer, mpfr-method
(mpfr-class), 27
Compare, mpfr, array-method (mpfr-class), 27
Compare, mpfr, integer-method
(mpfr-class), 27
Compare, mpfr, mpfr-method (mpfr-class), 27
Compare, mpfr, mpfrArray-method
(mpfrMatrix), 40
Compare, mpfr, numeric-method
(mpfr-class), 27
Compare, mpfrArray, mpfr-method
(mpfrMatrix), 40
Compare, mpfrArray, numeric-method
(mpfrMatrix), 40
Compare, numeric, mpfr-method
(mpfr-class), 27
Compare, numeric, mpfrArray-method
(mpfrMatrix), 40
complex, 6
Conj, mpfr-method (mpfr-class), 27
Const (mpfr), 25
cos, 29
cosh, 29
crossprod, array_or_vector, mpfr-method
(mpfr-class), 27
crossprod, Mnumber, mpfr-method
(mpfrMatrix), 40
crossprod, mpfr, array_or_vector-method
(mpfr-class), 27
crossprod, mpfr, missing-method
(mpfrMatrix), 40
crossprod, mpfr, Mnumber-method
(mpfrMatrix), 40
crossprod, mpfr, mpfr-method
(mpfrMatrix), 40
crossprod, mpfr, mpfrMatrix-method
(mpfrMatrix), 40
crossprod, mpfrMatrix, mpfr-method
(mpfrMatrix), 40
crossprod, mpfrMatrix, mpfrMatrix-method
(mpfrMatrix), 40
cummax, 29
cummin, 29
cumprod, 29
cumsum, 29
dbinom, 33
determinant, 43, 44
determinant.mpfrMatrix
(mpfrMatrix-utils), 43
diag.mpfrMatrix-method (mpfrMatrix), 40
diag<-, mpfrMatrix-method (mpfrMatrix), 40
diff, 38
diff.default, 38
diff.mpfr (mpfr.utils), 37
digamma, 3, 29
dim, 5, 16, 29, 41
dim, mpfrArray-method (mpfrMatrix), 40
dim<-, mpfr-method (mpfr-class), 27
dimnames, 16
dimnames, mpfrArray-method (mpfrMatrix), 40
dimnames<-, mpfrArray-method
(mpfrMatrix), 40
dnorm (mpfr-distr-etc), 32
dotsMethods, 9
double, 35
dpois, 32, 33
dpois (mpfr-distr-etc), 32
dsum (mpfr-special-functions), 33
dsum (mpfr-special-functions), 33
derfc (mpfr-special-functions), 33
exp, 29
expm1, 29
factorial, 10, 12
factorial, mpfr-method (mpfr-class), 27
factorialMPfr, 3, 11, 11, 29
factorialZ, 12
floor, 29
format, 13–16, 35
format, mpfr-method (mpfr-class), 27
formatBin, 25
formatBin (formatHex), 13
formatDec (formatHex), 13
formatHex, 13
formatMpfFr, 3, 14, 15, 30, 51
formatN, 16
formatN.mpfr(formatMpfFr), 15
function, 52
gamma, 3, 10, 12, 28, 29
INDEX

getD (mpfr-utils), 34
getGroupMembers, 29
getPrec, 3, 13, 25, 49, 51
getPrec (mpfr-utils), 34
gmp, 4
gmp-conversions, 17
golden_ratio, 45

hjk, 18
hjkMpfr, 3, 18
hypot (mpfr-class), 27

Im, mpfr-method (mpfr-class), 27
integer, 6, 30, 35, 36
integrate, 21, 22
integrateR, 3, 20
is.atomic, 6
is.finite, mpfr-method (mpfr-class), 27
is.finite, mpfrArray-method
  (mpfr-class), 27
is.infinite, mpfr-method (mpfr-class), 27
is.infinite, mpfrArray-method
  (mpfr-class), 27
is.integer, 24
is.na, mpfr-method (mpfr-class), 27
is.na, mpfrArray-method (mpfr-class), 27
is.nan, mpfr-method (mpfr-class), 27
is.nan, mpfrArray-method (mpfr-class), 27
is.whole, 23, 24, 31, 37

j0 (Bessel_mpfr), 8
j1 (Bessel_mpfr), 8
jn, 31
jn (Bessel_mpfr), 8

lbeta, ANY, mpfr-method (mpfr-class), 27
lbeta, ANY, mpfrArray-method
  (mpfr-class), 27
lbeta, mpfr, ANY-method (mpfr-class), 27
lbeta, mpfr, mpfr-method (mpfr-class), 27
lbeta, mpfr, numeric-method (mpfr-class), 27
lbeta, mpfrArray, ANY-method
  (mpfr-class), 27
lbeta, mpfrArray, mpfrArray-method
  (mpfr-class), 27
lbeta, numeric, mpfr-method (mpfr-class), 27
lgamma, 28, 29

Li2 (mpfr-special-functions), 33
list, 18, 19, 28, 40, 43, 45
load, 36
log, 29, 43
log, mpfr-method (mpfr-class), 27
log10, 29
log1p, 29
log2, 29
Logic, mpfr, mpfr-method (mpfr-class), 27
Logic, mpfr, numeric-method (mpfr-class), 27

logical, 13, 30

Math, 29
Math, mpfr-method (mpfr-class), 27
Math2, mpfr-method (mpfr-class), 27
matrix, 5, 26, 40
max, 29
mean, 29
mean, mpfr-method (mpfr-class), 27
mean.default, 29
median, mpfr-method (mpfr-class), 27
min, 29, 49
missing, 25
Mnumber, 9
Mnumber-class, 24
mNumber-class (Mnumber-class), 24
Mod, mpfr-method (mpfr-class), 27
mpfr, 3, 5, 7–10, 12–18, 20, 22–25, 25, 26, 28,
  30, 32–41, 44, 47–55
mpfr-class, 3, 27
mpfr-distr (mpfr-distr-etc), 32
mpfr-distr-etc, 32
mpfr-special-functions, 33
mpfr-utils, 34
mpfr.is.0 (mpfr.utils), 37
mpfr.is.integer (mpfr.utils), 37
mpfr.utils, 37
mpfr1, 40
mpfr1-class (mpfr-class), 27
mpfr2array, 39, 40
mpfr2array (mpfr-utils), 34
mpfr_default_prec (mpfr-utils), 34
mpfrArray, 3, 5, 16, 26, 29, 35, 36, 38, 39, 40, 42
mpfrArray-class (mpfrMatrix), 40
mpfrImport (mpfr-utils), 34
mpfrIs0, 51
mpfrIs0 (mpfr.utils), 37
mpfrMatrix, 5, 9, 24, 26, 30, 31, 36, 39, 40, 43, 44
mpfrMatrix-class, 3
mpfrMatrix-class (mpfrMatrix), 40
mpfrMatrix-utils, 43
mpfrVersion (mpfr.utils), 37
mpfrXport (mpfr-utils), 34

NaN, 28
nlm, 55
norm, 41
norm, ANY, missing-method (mpfrMatrix), 40
norm, mpfrMatrix, character-method
(mpfrMatrix), 40
NULL, 35, 40
numeric, 6, 8, 25–27, 30, 32, 33, 38
numeric_version, 38
numericVector-class (Mnumber-class), 24

Ops, 55
Ops, ANY, mpfr-method (mpfr-class), 27
Ops, array, mpfr-method (mpfr-class), 27
Ops, bigq, mpfr-method (mpfr-class), 27
Ops, bigz, mpfr-method (mpfr-class), 27
Ops, mpfr, ANY-method (mpfr-class), 27
Ops, mpfr, array-method (mpfr-class), 27
Ops, mpfr, bigq-method (mpfr-class), 27
Ops, mpfr, bigz-method (mpfr-class), 27
Ops, mpfr, vector-method (mpfr-class), 27
Ops, vector, mpfr-method (mpfr-class), 27
optim, 20
optimize, 45, 55
optimizeR, 3, 20, 44
options, 15, 51
order, 31

pbeta, 33, 47
pbetaI, 33, 46
pmax, 48, 48
pmax, ANY-method (pmax), 48
pmax, Mnumber-method (pmax), 48
pmax-methods (pmax), 48
pmin, 49
pmin (pmax), 48
pmin, ANY-method (pmax), 48
pmin, Mnumber-method (pmax), 48
pmin-methods (pmax), 48

pnorm, 3, 32–34
pnorm (mpfr-distr-etc), 32
pochMpfr, 12, 31
pochMpfr (chooseMpfr), 10
polyroot, 55
prettyNum, 16
print, 21
print.Bcharacter (formatHex), 13
print.default, 35
print.integrate, 22
print.integrateR (integrateR), 20
print.mpfr (mpfr-utils), 34
print.mpfr1 (mpfr-class), 27
print.mpfrArray (mpfr-utils), 34
prod, 29

quantile, 31
quantile, mpfr-method (mpfr-class), 27

range, 29, 49
rank, 31
raw, 30
rbind, 9
rbind (bind-methods), 9
rbind, ANY-method (bind-methods), 9
rbind, Mnumber-method (bind-methods), 9
rbind-methods (bind-methods), 9
Re, mpfr-method (mpfr-class), 27
Rmpfr (Rmpfr-package), 2
Rmpfr-package, 2
round, 29, 49
roundMpfr, 3, 26, 29, 31, 35, 49
rowMeans, mpfrArray-method (mpfrMatrix), 40
rowSums, mpfrArray-method (mpfrMatrix), 40

save, 36
seq, 50
seqMpfrr, 3, 50
setPrec (roundMpfr), 49
show, integrateR-method (integrateR), 20
show, mpfr-method (mpfr-class), 27
show, mpfr1-method (mpfr-class), 27
show, mpfrArray-method (mpfrMatrix), 40
sign, 28, 29
sign, mpfr-method (mpfr-class), 27
sign, mpfrArray-method (mpfrMatrix), 40
signif, 29
INDEX

sin, 29  
sinh, 29  
sort, 31  
sprintf, 13, 14, 16  
sqrt, 29  
str, 38, 51  
str.default, 51  
str.mpfr, 38, 51  
sum, 29  
sumBinomMpf, 3, 11, 47, 52  
Summary, 29  
Summary, mpfr-method (mpfr-class), 27  
t, mpfr-method (mpfr-class), 27  
t, mpfrMatrix-method (mpfrMatrix), 40  
tan, 29  
tanh, 29  
tcrossprod, array_or_vector, mpfr-method (mpfr-class), 27  
tcrossprod, Mnumber, mpfr-method (mpfrMatrix), 40  
tcrossprod, mpfr, array_or_vector-method (mpfr-class), 27  
tcrossprod, mpfr, missing-method (mpfrMatrix), 40  
tcrossprod, mpfr, Mnumber-method (mpfrMatrix), 40  
tcrossprod, mpfr, mpfr-method (mpfrMatrix), 40  
tcrossprod, mpfr, mpfrMatrix-method (mpfrMatrix), 40  
tcrossprod, mpfrMatrix, mpfr-method (mpfrMatrix), 40  
tcrossprod, mpfrMatrix, mpfrMatrix-method (mpfrMatrix), 40  
toNum, 5  
toNum (mpfr-utils), 34  
trigamma, 3, 29  
trunc, 29  
typeof, 5, 27  
unique, mpfr, missing-method (mpfr-class), 27  
uniroot, 54, 55  
unirootR, 3, 43, 54  
vector, 40  
Vectorize, 22  
which.max, 30