

Package ‘Bessel’

July 21, 2025

Title Computations and Approximations for Bessel Functions

Version 0.6-1

VersionNote Last CRAN: 0.6-0, packaged 2019-04-25, published
2019-05-02

Date 2024-07-29

Description Computations for Bessel function for complex, real and partly
'mpfr' (arbitrary precision) numbers; notably interfacing TOMS 644;
approximations for large arguments, experiments, etc.

Maintainer Martin Maechler <maechler@stat.math.ethz.ch>

Imports methods, Rmpfr

Suggests gsl, sfsmisc

SuggestsNote 'gsl' may be used in code; the others are for examples &
vignettes.

License GPL (>= 2)

Encoding UTF-8

URL <https://specfun.r-forge.r-project.org/>

NeedsCompilation yes

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

Repository CRAN

Date/Publication 2024-07-29 17:00:06 UTC

Contents

Airy	2
Bessel	4
BesselH	5
besselI.nuAsym	7
besselIasym	8
besselJs	10
bI	11

Description

Compute the Airy functions Ai or Bi or their first derivatives, $\frac{d}{dz} Ai(z)$ and $\frac{d}{dz} Bi(z)$.

The Airy functions are solutions of the differential equation

$$w'' = zw$$

for $w(z)$, and are related to each other and to the (modified) Bessel functions via (many identities, see <https://dlmf.nist.gov/9.6>), e.g., if $\zeta := \frac{2}{3}z\sqrt{z} = \frac{2}{3}z^{\frac{3}{2}}$,

$$Ai(z) = \pi^{-1} \sqrt{z/3} K_{1/3}(\zeta) = \frac{1}{3} \sqrt{z} (I_{-1/3}(\zeta) - I_{1/3}(\zeta)),$$

and

$$Bi(z) = \sqrt{z/3} (I_{-1/3}(\zeta) + I_{1/3}(\zeta)).$$

Usage

```
AiryA(z, deriv = 0, expon.scaled = FALSE, verbose = 0)
```

```
AiryB(z, deriv = 0, expon.scaled = FALSE, verbose = 0)
```

Arguments

<code>z</code>	complex or numeric vector.
<code>deriv</code>	order of derivative; must be 0 or 1.
<code>expon.scaled</code>	logical indicating if the result should be scaled by an exponential factor (typically to avoid under- or over-flow).
<code>verbose</code>	integer defaulting to 0, indicating the level of verbosity notably from C code.

Details

By default, when `expon.scaled` is false, `AiryA()` computes the complex Airy function $Ai(z)$ or its derivative $\frac{d}{dz} Ai(z)$ on `deriv=0` or `deriv=1` respectively.

When `expon.scaled` is true, it returns $\exp(\zeta) Ai(z)$ or $\exp(\zeta) \frac{d}{dz} Ai(z)$, effectively removing the exponential decay in $-\pi/3 < \arg(z) < \pi/3$ and the exponential growth in $\pi/3 < |\arg(z)| < \pi$, where $\zeta = \frac{2}{3}z\sqrt{z}$, and $\arg(z) = \text{Arg}(z)$.

While the Airy functions $Ai(z)$ and $d/dz Ai(z)$ are analytic in the whole z plane, the corresponding scaled functions (for `expon.scaled=TRUE`) have a cut along the negative real axis.

By default, when `expon.scaled` is false, `AiryB()` computes the complex Airy function $Bi(z)$ or its derivative $\frac{d}{dz} Bi(z)$ on `deriv=0` or `deriv=1` respectively.

When `expon.scaled` is true, it returns $\exp(-|\Re(\zeta)|) Bi(z)$ or $\exp(-|\Re(\zeta)|) \frac{d}{dz} Bi(z)$, to remove the exponential behavior in both the left and right half planes where, as above, $\zeta = \frac{2}{3} \cdot z\sqrt{z}$.

Value

a complex or numeric vector of the same length (and class) as `z`.

Author(s)

Donald E. Amos, Sandia National Laboratories, wrote the original fortran code. Martin Maechler did the R interface.

References

see [BesselJ](#); notably for many results the

Digital Library of Mathematical Functions (DLMF), Chapter 9 *Airy and Related Functions* at <https://dlmf.nist.gov/9>.

See Also

[BesselI](#) etc; the Hankel functions [Hankel](#).

The CRAN package [Rmpfr](#) has [Ai](#)(`x`) for arbitrary precise "mpfr"-numbers `x`.

Examples

```
## The AiryA() := Ai() function -----

curve(AiryA, -20, 100, n=1001)
curve(AiryA, -1, 100, n=1011, log="y") -> Aix
curve(AiryA(x, expon.scaled=TRUE), -1, 50, n=1001)
## Numerically "proving" the 1st identity above :
z <- Aix$x; i <- z > 0; head(z <- z[i <- z > 0])
Aix <- Aix$y[i]; zeta <- 2/3*z*sqrt(z)
stopifnot(all.equal(Aix, 1/pi * sqrt(z/3)* BesselK(zeta, nu = 1/3),
                    tol = 4e-15)) # 64b Lnx: 7.9e-16; 32b Win: 1.8e-15

## This gives many warnings (248 on nb-mm4, F24) about lost accuracy, but on Windows takes ~ 4 sec:
curve(AiryA(x, expon.scaled=TRUE), 1, 10000, n=1001, log="xy")

## The AiryB() := Bi() function -----

curve(AiryB, -20, 2, n=1001); abline(h=0,v=0, col="gray",lty=2)
curve(AiryB, -1, 20, n=1001, log = "y") # exponential growth (x > 0)

curve(AiryB(x,expon.scaled=TRUE), -1, 20, n=1001)
curve(AiryB(x,expon.scaled=TRUE), 1, 10000, n=1001, log="x")
```

Bessel

*Bessel Functions of Complex Arguments I(), J(), K(), and Y()***Description**

Compute the Bessel functions I(), J(), K(), and Y(), of complex arguments z and real nu ,

Usage

```
BesselI(z, nu, expon.scaled = FALSE, nSeq = 1, verbose = 0)
BesselJ(z, nu, expon.scaled = FALSE, nSeq = 1, verbose = 0)
BesselK(z, nu, expon.scaled = FALSE, nSeq = 1, verbose = 0)
BesselY(z, nu, expon.scaled = FALSE, nSeq = 1, verbose = 0)
```

Arguments

z	complex or numeric vector.
nu	numeric (scalar).
<code>expon.scaled</code>	logical indicating if the result should be scaled by an exponential factor, typically to avoid under- or over-flow. See the ‘Details’ about the specific scaling.
<code>nSeq</code>	positive integer; if > 1 , computes the result for a whole <i>sequence</i> of nu values; if $nu \geq 0$, $nu, nu+1, \dots, nu+nSeq-1$, if $nu < 0$, $nu, nu-1, \dots, nu-nSeq+1$.
<code>verbose</code>	integer defaulting to 0, indicating the level of verbosity notably from C code.

Details

The case $nu < 0$ is handled by using simple formula from Abramowitz and Stegun, see details in [besseli\(\)](#).

The scaling activated by `expon.scaled = TRUE` depends on the function and the scaled versions are

J(): $\text{BesselJ}(z, nu, \text{expo}=\text{TRUE}) := \exp(-|\Im(z)|) J_\nu(z)$

Y(): $\text{BesselY}(z, nu, \text{expo}=\text{TRUE}) := \exp(-|\Im(z)|) Y_\nu(z)$

I(): $\text{BesselI}(z, nu, \text{expo}=\text{TRUE}) := \exp(-|\Re(z)|) I_\nu(z)$

K(): $\text{BesselK}(z, nu, \text{expo}=\text{TRUE}) := \exp(z) K_\nu(z)$

Value

a complex or numeric vector (or [matrix](#) with `nSeq` columns if `nSeq > 1`) of the same length (or [nrow](#) when `nSeq > 1`) and [mode](#) as z .

Author(s)

Donald E. Amos, Sandia National Laboratories, wrote the original fortran code. Martin Maechler did the translation to C, and partial cleanup (replacing `goto`’s), in addition to the R interface.

References

- Abramowitz, M., and Stegun, I. A. (1964, etc). *Handbook of mathematical functions* (NBS AMS series 55, U.S. Dept. of Commerce), <https://personal.math.ubc.ca/~cbm/aands/>
- Wikipedia (20nn). *Bessel Function*, https://en.wikipedia.org/wiki/Bessel_function
- D. E. Amos (1986) Algorithm 644: A portable package for Bessel functions of a complex argument and nonnegative order; *ACM Trans. Math. Software* **12**, 3, 265–273.
- D. E. Amos (1983) *Computation of Bessel Functions of Complex Argument*; Sand83-0083.
- D. E. Amos (1983) *Computation of Bessel Functions of Complex Argument and Large Order*; Sand83-0643.
- D. E. Amos (1985) *A subroutine package for Bessel functions of a complex argument and nonnegative order*; Sand85-1018.
- Olver, F.W.J. (1974). *Asymptotics and Special Functions*; Academic Press, N.Y., p.420

See Also

The base R functions [besseli\(\)](#), [besselk\(\)](#), etc.

The Hankel functions (of first and second kind), $H_\nu^{(1)}(z)$ and $H_\nu^{(2)}(z)$: [Hankel](#).

The Airy functions $Ai()$ and $Bi()$ and their first derivatives, [Airy](#).

For large x and/or ν arguments, algorithm AS~644 is not good enough, and the results may overflow to Inf or underflow to zero, such that direct computation of $\log(I_\nu(x))$ and $\log(K_\nu(x))$ are desirable. For this, we provide [besseli.nuAsym\(\)](#), [besseli asym\(\)](#) and [besselk.nuAsym\(*, log=*\)](#), based on asymptotic expansions.

Examples

```
## For real small arguments, Besseli() gives the same as base::besseli() :
set.seed(47); x <- sort(round(rlnorm(20), 2))
M <- cbind(x, b = besseli(x, 3), B = Besseli(x, 3))
stopifnot(all.equal(M[, "b"], M[, "B"], tol = 2e-15)) # ~4e-16 even
M

## and this is true also for the 'exponentially scaled' version:
Mx <- cbind(x, b = besseli(x, 3, expon.scaled=TRUE),
            B = Besseli(x, 3, expon.scaled=TRUE))
stopifnot(all.equal(Mx[, "b"], Mx[, "B"], tol = 2e-15)) # ~4e-16 even
```

Description

Compute the Hankel functions $H(1,*)$ and $H(2,*)$, also called ‘H-Bessel’ function (of the third kind), of complex arguments. They are defined as

$$H(1, \nu, z) := H_\nu^{(1)}(z) = J_\nu(z) + iY_\nu(z),$$

$$H(2, \nu, z) := H_\nu^{(2)}(z) = J_\nu(z) - iY_\nu(z),$$

where $J_\nu(z)$ and $Y_\nu(z)$ are the Bessel functions of the first and second kind, see [BesselJ](#), etc.

Usage

```
BesselH(m, z, nu, expon.scaled = FALSE, nSeq = 1, verbose = 0)
```

Arguments

m	integer, either 1 or 2, indicating the kind of Hankel function.
z	complex or numeric vector of values different from 0 .
nu	numeric, must currently be non-negative.
expon.scaled	logical indicating if the result should be scaled by an exponential factor (typically to avoid under- or over-flow).
nSeq	positive integer; if > 1 , computes the result for a whole <i>sequence</i> of nu values of length nSeq, see ‘Details’ below.
verbose	integer defaulting to 0, indicating the level of verbosity notably from C code.

Details

By default (when `expon.scaled` is false), the resulting sequence (of length `nSeq`) is for $m = 1, 2$,

$$y_j = H(m, \nu + j - 1, z),$$

computed for $j = 1, \dots, nSeq$.

If `expon.scaled` is true, the sequence is for $m = 1, 2$

$$y_j = \exp(-\tilde{m}zi) \cdot H(m, \nu + j - 1, z),$$

where $\tilde{m} = 3 - 2m$ (and $i^2 = -1$), for $j = 1, \dots, nSeq$.

Value

a complex or numeric vector (or [matrix](#) if `nSeq > 1`) of the same length and [mode](#) as `z`.

Author(s)

Donald E. Amos, Sandia National Laboratories, wrote the original fortran code. Martin Maechler did the R interface.

References

see [BesselI](#).

See Also

[BesselI](#) etc; the Airy function [Airy](#).

Examples

```
##----- H(1, *) -----
nus <- c(1,2,5,10)
for(i in seq_along(nus))
  curve(BesselH(1, x, nu=nus[i]), -10, 10, add= i > 1, col=i, n=1000)
legend("topleft", paste("nu = ", format(nus)), col = seq_along(nus), lty=1)

## nu = 10 looks a bit "special" ...   hmm...
curve(BesselH(1, x, nu=10), -.3, .3, col=4,
      ylim = c(-10,10), n=1000)

##----- H(2, *) -----
for(i in seq_along(nus))
  curve(BesselH(2, x, nu=nus[i]), -10, 10, add= i > 1, col=i, n=1000)
legend("bottomright", paste("nu = ", format(nus)), col = seq_along(nus), lty=1)
## the same nu = 10 behavior ..
```

besselI.nuAsym	<i>Asymptotic Expansion of Bessel $I(x, \nu)$ and $K(x, \nu)$ for Large ν (and x)</i>
----------------	---

Description

Compute Bessel functions $I_\nu(x)$ and $K_\nu(x)$ for large ν and possibly large x , using asymptotic expansions in Debye polynomials.

Usage

```
besselI.nuAsym(x, nu, k.max, expon.scaled = FALSE, log = FALSE)
besselK.nuAsym(x, nu, k.max, expon.scaled = FALSE, log = FALSE)
```

Arguments

x	numeric or complex , with real part ≥ 0 .
nu	numeric; The <i>order</i> (maybe fractional!) of the corresponding Bessel function.
k.max	integer number of terms in the expansion. Must be in 0:5, currently.
expon.scaled	logical; if TRUE, the results are exponentially scaled, the same as in the corresponding BesselI() and BesselK() functions in order to avoid overflow (I_ν) or underflow (K_ν), respectively.
log	logical; if TRUE, $\log(f(.))$ is returned instead of f .

Details

Abramowitz & Stegun , page 378, has formula 9.7.7 and 9.7.8 for the asymptotic expansions of $I_\nu(x)$ and $K_\nu(x)$, respectively, also saying *When $\nu \rightarrow +\infty$, these expansions (of $I_\nu(\nu z)$ and $K_\nu(\nu z)$) hold uniformly with respect to z in the sector $|\arg z| \leq \frac{1}{2}\pi - \epsilon$, where ϵ is an arbitrary positive number.* and for this reason, we require $\Re(x) \geq 0$.

The Debye polynomials $u_k(x)$ are defined in 9.3.9 and 9.3.10 (page 366).

Value

a numeric vector of the same length as the long of `x` and `nu`. (usual argument recycling is applied implicitly.)

Author(s)

Martin Maechler

References

Abramowitz, M., and Stegun, I. A. (1964, etc). *Handbook of mathematical functions*, pp. 366, 378.

See Also

From this package **Bessel**: [BesselI\(\)](#); further, [bessellasym\(\)](#) for the case when x is large and ν is small or moderate.

Further, from **base**: [besselI](#), etc.

Examples

```
x <- c(1:10, 20, 50, 100, 100000)
nu <- c(1, 10, 20, 50, 10^(2:10))

sapply(0:4, function(k.)
  sapply(nu, function(n.)
    besselI.nuAsym(x, nu=n., k.max = k., log = TRUE)))

sapply(0:4, function(k.)
  sapply(nu, function(n.)
    besselK.nuAsym(x, nu=n., k.max = k., log = TRUE)))
```

Description

Compute Bessel function $I_\nu(x)$ and $K_\nu(x)$ for large x and small or moderate ν , using the asymptotic expansions (9.7.1) and (9.7.2), p.377-8 of Abramowitz & Stegun, for $x \rightarrow \infty$, even valid for complex x ,

$$I_a(x) = \exp(x)/\sqrt{2\pi x} \cdot f(x, a),$$

where

$$f(x, a) = 1 - \frac{\mu - 1}{8x} + \frac{(\mu - 1)(\mu - 9)}{2!(8x)^2} - \dots,$$

and $\mu = 4a^2$ and $|\arg(x)| < \pi/2$.

Whereas `bessellIasym(x, a)` computes a possibly exponentially scaled and/or **logged** version of $I_a(x)$, `bessellI.ftrms` returns the corresponding *terms* in the series expansion of $f(x, a)$ above.

Usage

```
bessellIasym (x, nu, k.max = 10, expon.scaled = FALSE, log = FALSE)
besselKasym (x, nu, k.max = 10, expon.scaled = FALSE, log = FALSE)
bessellI.ftrms(x, nu, K = 20)
```

Arguments

<code>x</code>	numeric or complex (with real part) ≥ 0 .
<code>nu</code>	numeric; the <i>order</i> (maybe fractional!) of the corresponding Bessel function.
<code>k.max, K</code>	integer number of terms in the expansion.
<code>expon.scaled</code>	logical; if TRUE, the results are exponentially scaled in order to avoid overflow.
<code>log</code>	logical; if TRUE, $\log(f(\cdot))$ is returned instead of f .

Details

Even though the reference (A. & S.) requires $|\arg z| < \pi/2$ for $I()$ and $|\arg z| < 3\pi/2$ for $K()$, where $\arg(z) := \text{Arg}(z)$, the zero-th order term seems correct also for negative (real) numbers.

Value

a numeric (or complex) vector of the same length as `x`.

Author(s)

Martin Maechler

References

Abramowitz, M., and Stegun, I. A. (1964, etc). *Handbook of mathematical functions* (NBS AMS series 55, U.S. Dept. of Commerce).

See Also

From this package **Bessel()** **BessellI()**; further, `bessellI.nuAsym()` which is useful when ν is large (as well); further **base** `bessellI`, etc

Examples

```
x <- c(1:10, 20, 50, 100^(2:10))
nu <- c(1, 10, 20, 50, 100)
r <- lapply(c(0:4,10,20), function(k.)
  sapply(nu, function(n.)
    besseliAsym(x, nu=n., k.max = k., log = TRUE)))
warnings()

try( # needs improvement in R [or a local workaround]
  besseliAsym(10000*(1+1i), nu=200, k.max=20, log=TRUE)
) # Error in log1p(-d) : unimplemented complex function
```

besselJs

Bessel J() function Simple Series Representation

Description

Computes the modified Bessel J function, using one of its basic definitions as an infinite series, e.g. A. & S., p.360, (9.1.10). The implementation is pure R, working for [numeric](#), [complex](#), but also e.g., for objects of class "[mpfr](#)" from package **Rmpfr**.

Usage

```
besselJs(x, nu, nterm = 800, log = FALSE,
  Ceps = if (isNum) 8e-16 else 2^(-x@.Data[[1]]@prec))
```

Arguments

x	numeric or complex vector, or of another class for which arithmetic methods are defined, notably objects of class mpfr .
nu	non-negative numeric (scalar).
nterm	integer indicating the number of terms to be used. Should be in the order of $\text{abs}(x)$, but can be smaller for large x . A warning is given, when nterm was <i>possibly</i> too small. (Currently, many of these warnings are wrong, as
log	logical indicating if the logarithm $\log J()$ is required.
Ceps	a relative error tolerance for checking if nterm has been sufficient. The default is "correct" for double precision and also for multiprecision objects.

Value

a "numeric" (or complex or "[mpfr](#)") vector of the same class and length as x .

Author(s)

Martin Maechler

References

Abramowitz, M., and Stegun, I. A. (1964–1972). *Handbook of mathematical functions* (NBS AMS series 55, U.S. Dept. of Commerce). https://personal.math.ubc.ca/~cbm/aands/page_360.htm

See Also

This package `BesselJ()`, `base` `besselJ()`, etc

Examples

```
stopifnot(all.equal(besselJs(1:10, 1), # our R code --> 4 warnings, for x = 4:7
  besselJ(1:10, 1)))# internal C code w/ different algorithm

## Large 'nu' ...
x <- (0:20)/4
if(interactive()) op <- options(nwarnings = 999)
(bx <- besselJ(x, nu=200))# base R's -- gives 19 (mostly wrong) warnings about precision lost
## Visualize:
bj <- curve(besselJ(1, x), 1, 2^10, log="xy", n=1001,
  main=quote(J[nu](1)), xlab = quote(nu), xaxt="n", yaxt="n") # 50+ warnings
eaxis <- if(!requireNamespace("sfsmisc")) axis else sfsmisc::eaxis
eaxis(1, sub10 = 3); eaxis(2)
bj6 <- curve(besselJ(6, x), add=TRUE, n=1001, col=adjustcolor(2, 1/2), lwd=2)
plot(y~x, as.data.frame(bj6), log="x", type="l", col=2, lwd=2,
  main = quote(J[nu](6)), xlab = quote(nu), xaxt="n")
eaxis(1, sub10=3); abline(h=0, lty=3)

if(require("Rmpfr")) { ## Use high precision, notably large exponent range, numbers:
  Bx <- besselJs(mpfr(x, 64), nu=200)
  all.equal(Bx, bx, tol = 1e-15)# TRUE -- warnings were mostly wrong; specifically:
  cbind(bx, Bx)
  signif(asNumeric(1 - (bx/Bx)[19:21]), 4) # only [19] had lost accuracy

  ## Without* mpfr numbers -- using log -- is accurate (here)
  lbx <- besselJs(x, nu=200, log=TRUE)
  lBx <- besselJs(mpfr(x, 64), nu=200, log=TRUE)
  cbind(x, lbx, lBx)
  stopifnot(all.equal(asNumeric(log(Bx)), lbx, tol=1e-15),
    all.equal(lBx, lbx, tol=4e-16))
} # Rmpfr
if(interactive()) options(op) # reset 'nwarnings'
```

Description

Computes the modified Bessel I function, using one of its basic definitions as an infinite series. The implementation is pure R, working for `numeric`, `complex`, but also e.g., for objects of class `"mpfr"` from package **Rmpfr**.

Usage

```
besselIs(x, nu, nterm = 800, expon.scaled = FALSE, log = FALSE,
        Ceps = if (isNum) 8e-16 else 2^(-x@.Data[[1]]@prec))
```

Arguments

<code>x</code>	numeric or complex vector, or of another <code>class</code> for which arithmetic methods are defined, notably objects of class <code>mpfr</code> (package Rmpfr).
<code>nu</code>	non-negative numeric (scalar).
<code>nterm</code>	integer indicating the number of terms to be used. Should be in the order of <code>abs(x)</code> , but can be smaller for large <code>x</code> . A warning is given, when <code>nterm</code> was chosen too small.
<code>expon.scaled</code>	logical indicating if the result should be scaled by $\exp(-\text{abs}(x))$.
<code>log</code>	logical indicating if the logarithm $\log I(\cdot)$ is required. This allows even more precision than <code>expon.scaled=TRUE</code> in some cases.
<code>Ceps</code>	a relative error tolerance for checking if <code>nterm</code> has been sufficient. The default is “correct” for double precision and also for multiprecision objects.

Value

a “numeric” (or complex or `"mpfr"`) vector of the same class and length as `x`.

Author(s)

Martin Maechler

References

Abramowitz, M., and Stegun, I. A. (1964,..., 1972). *Handbook of mathematical functions* (NBS AMS series 55, U.S. Dept. of Commerce).

See Also

This package `BesselI`, `base` `besselI`, etc

Examples

```
(nus <- c(outer((0:3)/4, 1:5, `+`)))
stopifnot(
  all.equal(besselIs(1:10, 1), # our R code
            besselI (1:10, 1)) # internal C code w/ different algorithm
  ,
```

```

sapply(nus, function(nu)
  all.equal(besselIs(1:10, nu, expon.scale=TRUE), # our R code
            BesselI (1:10, nu, expon.scale=TRUE)) # TOMS644 code
)
,
## complex argument [gives warnings 'nterm=800' may be too small]
sapply(nus, function(nu)
  all.equal(besselIs((1:10)*(1+1i), nu, expon.scale=TRUE), # our R code
            BesselI ((1:10)*(1+1i), nu, expon.scale=TRUE)) # TOMS644 code
)
)

## Large 'nu' ...
x <- (0:20)/4
(bx <- besselI(x, nu=200))# base R's -- gives (mostly wrong) warnings
if(require("Rmpfr")) { ## Use high precision (notably large exponent range) numbers:
  Bx <- besselIs(mpfr(x, 64), nu=200)
  all.equal(Bx, bx, tol = 1e-15)# TRUE -- warning were mostly wrong; specifically:
  cbind(bx, Bx)
  signif(asNumeric(1 - (bx/Bx)[19:21]), 4) # only [19] had lost accuracy

  ## With*out* mpfr numbers -- using log -- is accurate (here)
  (lbx <- besselIs(x, nu=200, log=TRUE))
  lBx <- besselIs(mpfr(x, 64), nu=200, log=TRUE)
  stopifnot(all.equal(asNumeric(log(Bx)), lbx, tol=1e-15),
            all.equal(lBx, lbx, tol=4e-16))
} # Rmpfr

```

Index

* **math**

- Airy, [2](#)
- Bessel, [4](#)
- BesselH, [5](#)
- besselI.nuAsym, [7](#)
- besselIasym, [8](#)
- besselJs, [10](#)
- bI, [11](#)

- Ai, [3](#)
- Airy, [2](#), [5](#), [7](#)
- AiryA (Airy), [2](#)
- AiryB (Airy), [2](#)
- Arg, [2](#), [9](#)

- Bessel, [4](#)
- BesselH, [5](#)
- BesselI, [3](#), [6–9](#), [12](#)
- BesselI (Bessel), [4](#)
- besselI, [4](#), [5](#), [8](#), [9](#), [12](#)
- besselI.ftrms (besselIasym), [8](#)
- besselI.nuAsym, [5](#), [7](#), [9](#)
- besselIasym, [5](#), [8](#), [8](#)
- besselIs (bI), [11](#)
- BesselJ, [3](#), [6](#), [11](#)
- BesselJ (Bessel), [4](#)
- besselJ, [11](#)
- besselJs, [10](#)
- BesselK (Bessel), [4](#)
- besselK, [5](#)
- besselK.nuAsym, [5](#)
- besselK.nuAsym (besselI.nuAsym), [7](#)
- besselKasym (besselIasym), [8](#)
- BesselY (Bessel), [4](#)
- bI, [11](#)

- class, [10](#), [12](#)
- complex, [7](#), [9](#), [10](#), [12](#)

- Hankel, [3](#), [5](#)

- Hankel (BesselH), [5](#)

- log, [9](#)

- matrix, [4](#), [6](#)
- mode, [4](#), [6](#)
- mpfr, [10](#), [12](#)

- nrow, [4](#)
- numeric, [10](#), [12](#)