

Package ‘Bessel’

September 28, 2026

Title Computations and Approximations for Bessel Functions

Version 0.7-1

VersionNote Last CRAN: 0.7-0, 2026-01-10; 0.6-1, 2024-07-29; and
0.6-0, 2019-05-02

Date 2026-09-28

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 (>= 1.1-23)

SuggestsNote 'gsl' may be used in code; 'sfsmisc' is for examples &
vignettes

Depends R (>= 3.3.0)

License GPL (>= 2)

Encoding UTF-8

NeedsCompilation yes

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

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

Repository CRAN

Date/Publication 2026-09-28 09:10:19 UTC

Contents

Airy	2
Bessel	4
BesselH	5
besselI.nuAsym	7
besselIasym	10

besselIs	11
besselJs	13
bessJmlNu	15
incompleteBesselK	16

Index	20
--------------	-----------

Airy	<i>Airy Functions (and Their First Derivative)</i>
------	--

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 $\zeta := \frac{2}{3}z\sqrt{z} = \frac{2}{3}z^{\frac{3}{2}}$,

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

and

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

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 a 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 nu 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, e.g.,

See Also

From this package **Bessel**: [BesselI\(\)](#); further, [besselIasym\(\)](#) for the case when x is large and ν is small or moderate (using A.&S. (9.7.1) and (9.7.2)).

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

Examples

```
x <- c(1,2, 10, 20, 50, 100, 1000, 1e4)
nu <- c(outer(c(1,2,5), 10^(0:9)))
ks <- 0:4; ks <- setNames(ks, paste0("k=",ks))

if(requireNamespace("sfsmisc")) { # Mostly for plotting
  formatN <- if(packageVersion("sfsmisc") >= "1.1.26")
    sfsmisc::frmtNum else sfsmisc::formatN
  eaxis <- sfsmisc::eaxis
  mult.fig <- sfsmisc::mult.fig
} else { ## no {sfsmisc}
  formatN <- function(x) formatC(x, width=1)
  eaxis <- axis
  mult.fig <- function(nr.plots, main=NULL, ...) { # cheap but working substitute
    mfrow <- n2mfrow(nr.plots)
    tit.wid <- if(is.null(main)) 0 else 1 + 1.5 * par("cex.main")
    old.par <- par(mfrow = mfrow, oma = c(0, 0, tit.wid, 0),
      mar = 0.1 + c(4, 4, 2, 1), mgp = c(1.5, 0.6, 0))
    if (!is.null(main)) {
      plot.new(); mtext(main, side = 3, outer = TRUE, cex=par("cex.main")); par(new = TRUE)
    }
    invisible(list(old.par = old.par))
  }
}
```

```

    }
  }

  nus <- setNames(nu, paste0("nu=", formatN(nu)))

  mI <- sapply(simplify = "array", ks, function(k.)
    sapply(nus, function(n.)
      besselI.nuAsym(x, nu=n., k.max = k., log = TRUE)))

  mK <- sapply(simplify = "array", ks, function(k.)
    sapply(nus, function(n.)
      besselK.nuAsym(x, nu=n., k.max = k., log = TRUE)))
  str(mK) # 7 x 30 x 5 array

##----- Plotting besselI : I_nu(x = x0) vs nu -----

str(op <- mult.fig(length(x))$old.par); par("mfrow")
for(ix in seq_along(x)) {
  matplot(nu, mI[ix,,], type = "l", log = "x",
    xaxt = "n", xlab = quote(nu), ylab = quote(I[nu](x)),
    main = substitute(I[nu](x==X, nu, k.max == 0:4), list(X = (xi <- x[ix]))))
  eaxis(1)
}
par(op)

##----- Plotting besselK : K_nu(x = x0) vs nu -----

str(mult.fig(length(x), main = expression(log(K[nu](x)) ~ vs ~ nu ~ "x in log-scale"))$old.par)
par("mfrow")
for(ix in seq_along(x)) {
  y <- mK[ix,,]
  matplot(nu, y, type = "l", log = "x", ylim = c(min(y), min(max(y), 3*abs(min(y)))),
    xaxt = "n", xlab = quote(nu), ylab = quote(K[nu](x)),
    main = substitute(K[nu](x==X, nu, k.max == 0:4), list(X = (xi <- x[ix]))))
  abline(h = 0, col = "gray20", lty = 3)
  eaxis(1)
}
par(op)

str(mult.fig(length(x), main = expression(log(K[nu](x)) ~ vs ~ nu ~ "in log-log"))$old.par)
for(ix in seq_along(x)) {
  matplot(nu, mK[ix,,], type = "l", log = "xy", # --> 14 warnings: each plot h
    xaxt = "n", yaxt = "n", xlab = quote(nu), ylab = quote(K[nu](x)),
    main = substitute(K[nu](x==X, nu, k.max == 0:4), list(X = (xi <- x[ix]))))
  eaxis(1); eaxis(2)
}
par(op)

## Compare with besselK() -- which I can use expon.scaled !
dimnames(mI)[[1]] <- dimnames(mK)[[1]] <- paste0("x=", formatN(x))
str(mK)

rbind(t(mK[, "nu=200", ]), log.besselK = log(besselK(x, nu=200, expon.scaled=TRUE)) - x)

```

```
##--> apart from k=0 the asymptotic approximation seems good already for k=1 here

rbind(t(mK[, "nu=500", ]), log.besselK = log(besselK(x, nu=500, expon.scaled=TRUE)) - x)
##--> apart from k=0 the asymptotic approximation seems good already for k=1 here
##---> need other, smaller x[] range - to get non-Inf besselK()
```

besselIasym

Asymptotic Expansion of Bessel I(x,nu) and K(x,nu) For Large x

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 `besselIasym(x, a)` computes a possibly exponentially scaled and/or [logged](#) version of $I_a(x)$, `besselI.ftrms` returns the corresponding *terms* in the series expansion of $f(x, a)$ above.

Usage

```
besselIasym (x, nu, k.max = 10, expon.scaled = FALSE, log = FALSE)
besselKasym (x, nu, k.max = 10, expon.scaled = FALSE, log = FALSE)
besselI.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 $I(\cdot)$ and $|\arg z| < 3\pi/2$ for $K(\cdot)$, 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()** **BesselI()**; further, **besselI.nuAsym()** which is useful when ν is large as well, using A.&S. (9.7.7) and (9.7.8); further **base** **besselI**, 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
```

besselIs

Bessel I() function Simple Series Representation

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))

## __Deprecated__ name :
bI (x, nu, nterm = 800, expon.scaled = FALSE, 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 (package Rmpfr).
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 <code>nterm</code> was chosen too small.
expon.scaled	logical indicating if the result should be scaled by $\exp(-\text{abs}(x))$.
log	logical indicating if the logarithm $\log I.()$ is required. This allows even more precision than <code>expon.scaled=TRUE</code> in some cases.
Ceps	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

  ## Without* 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
```

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,
  use.log = log || any(nu * log(x/2) > .Machine$double.max.exp * log(2)),
  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.
use.log	logical indicating if the logarithm should work in logarithmic scale notably even when log = FALSE . Its smart default, makes <code>besselJs(4, 1e10)</code> to work, correctly giving 0, where it gave NaN previously.
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), .001, 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 = c(-2,3)); eaxis(2)
abline(h = (J01 <- besselJ(1, 0)), col=adjustcolor(2, 1/2))
axis(4, at=J01, quote(J[0](1)), las=2, col=2, col.axis=2, line = -1)
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)

## large nu (non-log case) -- gave NaN wrongly (typically when it should have given 0):
stopifnot(exprs = {
  besselJs(c(.1, 1:10), 999) == 0
  besselJs(c(.1, 1:10), 9999) == 0
  besselJs(c(.1, 1:1000), 1e10) == 0
  ## besselJs(c(.1, 1:10), 1e20) == 0 -- another *BUG*: Error ... lssum found non-positive sums
})

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'
```

bessJsm1Nu

Bessel $J_\nu(x)$ for Very Small Order ν **Description**

Use a few terms Taylor approximation for Bessel's $J_\nu(x)$ for fixed x and $\nu \rightarrow 0$.

Usage

```
bessJsm1Nu(x, nu, nTrms, m.max)
```

Arguments

x, nu	numeric Bessel J() arguments, see e.g., besselJ .
nTrms	integer, one of 0, 1, or 2, denoting the number of Taylor series terms to be used, nTrms = 0 simply returning BesselJ (x, 0).
m.max	positive integer specifying the Taylor series order used.

Details

A. & S. (9.1.10) p. 360 (Ascending Series) (9.1.64) p. 362 (derivative wrt order)
 show derivation?

Value

The function explicitly calls [Vectorize](#)(<fn>, c("x", "nu")), hence a numeric vector conforming to (e.g.,) x + nu.

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's [BesselJ](#), base R's [besselJ](#)

Examples

```
(nuSml <- 2^-c(seq(30, 53, by=1/2), 75, 100, 300, 800, 1022, 1074, Inf))
## 9.3e-10 6.6e-10 ... 1.1e-16 .. 2.22e-308 4.94e-324 0.0
## bug in R (at least up to Jan. 1 2026) :
options(digits = 14) # show more digits
Jsm1 <- sapply(nuSml, function(nu) besselJ(pi/2, nu))
tail(cbind(nuSml, Jsm1), 20) # complete "divergence" for nu <~ 10^-{16}
Jsm.nT.1 <- bessJsm1Nu(pi/2, nuSml, nTrms = 1, m.max = 30)
Jsm.nT.2 <- bessJsm1Nu(pi/2, nuSml, nTrms = 2, m.max = 30)
cbind(nuSml, Jsm1, Jsm.nT.1, Jsm.nT.2)
table(Jsm.nT.2 == Jsm.nT.1) # all TRUE (really ???)
      all.equal(Jsm.nT.2, Jsm.nT.1, tolerance = 0) # show
stopifnot(all.equal(Jsm.nT.2, Jsm.nT.1, tolerance = 4e-16))

## second example; nu *not* very small
x. <- 2
nu2 <- 2^-seq(1, 30, by = 1/8)
Jsm2.1 <- bessJsm1Nu(x., nu2, nTrms = 1, m.max = 40)
Jsm2.2 <- bessJsm1Nu(x., nu2, nTrms = 2, m.max = 40)
table(Jsm2.2 == Jsm2.1) # now they *do* differ
bessJ2 <- besselJ(x., nu2)
BessJ2 <- sapply(nu2, function(nu) BesselJ(x., nu))
## J() function values: -----
matplot(nu2, cbind(bessJ2, BessJ2, Jsm2.1, Jsm2.2), log = "x", type = "l")
sum(i <- nu2 < 1e-4)
stopifnot(all.equal(BessJ2[i], Jsm2.2[i], tolerance = 4e-14),
          all.equal(BessJ2[i], Jsm2.1[i], tolerance = 4e-9))

## |error| wrt BesselJ() -----
matplot(nu2, abs(cbind(bessJ2, Jsm2.1, Jsm2.2) - BessJ2), log = "xy",
        type = "l", ylim = c(5e-17, 1e-10),
        xlab = quote("nu" == nu), xaxt = "n", yaxt = "n",
        main = substitute(list(abs("<bess J>"(x, nu) - BesselJ(x, nu)), x == XX),
                          list(XX = formatC(x.))))
eaxis <- if(!requireNamespace("sfsmisc")) axis else sfsmisc::eaxis
eaxis(1, sub10 = c(-2,0)); eaxis(2)
legend("topleft", lty = 1:5, lwd = 2, col = 1:6, bty = "n",
      legend = paste0(c("besselJ(", paste0("bessJsm1Nu(*, nTrms = ", 1:2)), ")"))
```

incompleteBesselK

The Incomplete Bessel K Function

Description

Calculates the incomplete Bessel K function using the algorithm and code provided by Slavinsky and Safouhi (2009).

Usage

```

incompleteBesselK(x, y, nu, tol = .Machine$double.eps^0.85, nmax = 120)
incompleteBesselKR(x, y, nu, tol = .Machine$double.eps^0.85, nmax = 120, verbose = FALSE)

SSFcoef(nmax, nu)
combinatorial(nu)
GDENOM(n, x, y, nu, An, nmax, Cnp)
GNUM(n, x, y, nu, Am, An, nmax, Cnp, GM, GN)

```

Arguments

x	Numeric. Value of the first argument of the incomplete Bessel K function.
y	Numeric. Value of the second argument of the incomplete Bessel K function.
nu	Numeric. The order of the incomplete Bessel K function.
tol	Numeric. The tolerance for the difference between successive approximations of the incomplete Bessel K function.
nmax	Integer. The maximum order allowed for the approximation of the incomplete Bessel K function.
verbose	a logical indicating if a line of current state should be printed to the console during the computations.
n	Integer. Current order of the approximation. Not required to be specified by users.
An	Matrix of coefficients. Not required to be specified by users.
Am	Matrix of coefficients. Not required to be specified by users.
Cnp	Vector of elements of Pascal's triangle. Not required to be specified by users.
GN	Vector of denominators used for approximation. Not required to be specified by users.
GM	Vector of numerators used for approximation. Not required to be specified by users.

Details

The function `incompleteBesselK` implements the algorithm proposed by Slavinsky and Safouhi (2010) and uses code provided by them.

The incomplete Bessel K function is defined by

$$K_\nu(x, y) = \int_1^\infty t^{-\nu-1} \exp(-xt - y/t) dt$$

see Slavinsky and Safouhi (2010), or Harris (2008).

`incompleteBesselK` calls a Fortran routine to carry out the calculations. `incompleteBesselKR()` is a pure R version of the routine for computing the incomplete Bessel K function.

The functions `SSFcoef`, `combinatorial`, `GDENOM`, and `GNUM` are “subroutines”, i.e., auxiliary functions used in `incompleteBesselKR()`. They are not expected to be called by the user, but exported for didactical reasons.

The approximation to the incomplete Bessel K function returned by `incompleteBesselK` is highly accurate. The default value of `tol` is about 10^{-14} on a 32-bit computer. It appears that even higher accuracy is possible when $x > y$. Then the tolerance can be taken as `.Machine$double.eps` and the number of correct figures essentially coincides with the number of figures representable in the machine being used.

`incompleteBesselKR` is very slow compared to the Fortran version and is only included for those who wish to see the algorithm in R rather than Fortran.

Value

`incompleteBesselK` and `incompleteBesselKR` both return an approximation to the incomplete Bessel K function as defined above.

Note

The problem of calculation of the incomplete Bessel K function is equivalent to the problem of calculation of the cumulative distribution function of the generalized inverse Gaussian distribution. See [Generalized Inverse Gaussian](#).

Author(s)

David Scott <d.scott@auckland.ac.nz>, Thomas Tran, Richard Slevinsky, Hassan Safouhi; part of **DistributionUtils** since 2010.

References

- Harris, Frank E. (2008) Incomplete Bessel, generalized incomplete gamma, or leaky aquifer functions. *J. Comp. Appl. Math.* **215**, 260–269. doi:10.1016/j.cam.2007.04.008
- Temme, Nico M. (2009) The leaky aquifer function revisited. *Int. J. Quantum Chem.* **109**, 2826–2830. doi:10.1002/qua.22130
- Slevinsky, Richard M., and Safouhi, Hassan (2009) New formulae for higher order derivatives and applications. *J. Comp. Appl. Math.* **233**, 405–419. doi:10.1016/j.cam.2009.07.038
- Slevinsky, Richard M., and Safouhi, Hassan (2010) A recursive algorithm for the G transformation and accurate computation of incomplete Bessel functions. *Applied Numerical Mathematics* **60**(12), 1411–1417. doi:10.1016/j.apnum.2010.04.005

See Also

[besselK](#)

Examples

```
### Harris (2008) gives accurate values (16 figures) for
### x = 0.01, y = 4, and nu = 0:9
### nu = 0, Harris value is 2.22531 07612 66469
options(digits = 16)
incompleteBesselK(0.01, 4, 0)
### nu = 9, Harris value is 0.00324 67980 03149
incompleteBesselK(0.01, 4, 9)
```

```
### Other values given in Harris (2008)
### x = 4.95, y = 5.00, nu = 2
incompleteBesselK(4.95, 5, 2) ## 0.00001 22499 87981
### x = 10, y = 2, nu = 6
### Slevinsky and Safouhi (2010) suggest Harris (2008) value
### is incorrect, give value 0.00000 04150 01064 21228
incompleteBesselK(10, 2, 6)
### x = 3.1, y = 2.6, nu = 5
incompleteBesselK(3.1, 2.6, 5) ## 0.00052 85043 25244

### Check values when x > y using numeric integration
nus <- 0:9
(numIBF <- sapply(nus, incompleteBesselK, x = 4, y = 0.01))

besselFn <- function(t, x, y, nu) t^(-nu - 1) * exp(-x*t - y/t)

integIBF <- sapply(nus, integrate, f = besselFn, lower = 1, upper = Inf,
                  x = 4, y = 0.01)
intIBF <- as.numeric(integIBF["value", ])
numIBF - intIBF
max(abs(numIBF - intIBF)) ## 1.257e-11

options(digits = 7)
```

Index

* distribution

incompleteBesselK, 16

* math

Airy, 2

Bessel, 4

BesselH, 5

besselI.nuAsym, 7

besselIasym, 10

besselIs, 11

besselJs, 13

bessJsm1Nu, 15

incompleteBesselK, 16

Ai, 3

Airy, 2, 5, 7

AiryA (Airy), 2

AiryB (Airy), 2

Arg, 2, 10

Bessel, 4

BesselH, 5

BesselI, 3, 6–8, 11, 12

BesselI (Bessel), 4

besselI, 4, 5, 8, 11, 12

besselI.ftms (besselIasym), 10

besselI.nuAsym, 5, 7, 11

besselIasym, 5, 8, 10

besselIs, 11

BesselJ, 3, 6, 14–16

BesselJ (Bessel), 4

besselJ, 14–16

besselJs, 13

BesselK (Bessel), 4

besselK, 5, 18

besselK.nuAsym, 5

besselK.nuAsym (besselI.nuAsym), 7

besselKasym (besselIasym), 10

BesselY (Bessel), 4

bessJsm1Nu, 15

bI (besselIs), 11

class, 12, 13

combinatorial (incompleteBesselK), 16

complex, 7, 10, 11, 13

GDENOM (incompleteBesselK), 16

GNUM (incompleteBesselK), 16

Hankel, 3, 5

Hankel (BesselH), 5

incompleteBesselK, 16

incompleteBesselKR (incompleteBesselK),
16

log, 10

logical, 17

matrix, 4, 6

mode, 4, 6

mpfr, 11–14

nrow, 4

numeric, 11, 13

SSFcoef (incompleteBesselK), 16

Vectorize, 15