Package ‘DPQmpfr’

December 5, 2023

Title  DPQ (Density, Probability, Quantile) Distribution Computations using MPFR

Version  0.3-2

Date  2023-12-04

Description  An extension to the 'DPQ' package with computations for 'DPQ' (Density (pdf), Probability (cdf) and Quantile) functions, where the functions here partly use the 'Rmpfr' package and hence the underlying 'MPFR' and 'GMP' C libraries.

Depends  R (>= 3.6.0)

Imports  DPQ (>= 0.5-3), Rmpfr (>= 0.9-0), gmp, sfsmisc, stats, graphics, methods, utils

Suggests  Matrix

SuggestsNote  Matrix for its test-tools-1.R

License  GPL (>= 2)

Encoding  UTF-8

URL  https://specfun.r-forge.r-project.org/,
     https://r-forge.r-project.org/R/?group_id=611,
     https://r-forge.r-project.org/scm/viewvc.php/pkg/DPQmpfr/?root=specfun,
     svn://svn.r-forge.r-project.org/svnroot/specfun/pkg/DPQmpfr

BugReports  https://r-forge.r-project.org/tracker/?atid=2462&group_id=611

NeedsCompilation  no

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

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

Repository  CRAN

Date/Publication  2023-12-05 22:30:02 UTC
Description

An extension to the 'DPQ' package with computations for 'DPQ' (Density (pdf), Probability (cdf) and Quantile) functions, where the functions here partly use the 'Rmpfr' package and hence the underlying 'MPFR' and 'GMP' C libraries.

Details

The DESCRIPTION file:

Package: DPQmpfr
Title: DPQ (Density, Probability, Quantile) Distribution Computations using MPFR
Version: 0.3-2
Date: 2023-12-04
Authors@R: person("Martin","Maechler",role=c("aut","cre"), email="maechler@stat.math.ethz.ch", comment = c(ORCID = "0000-0002-8685-9910"))
Description: An extension to the 'DPQ' package with computations for 'DPQ' (Density (pdf), Probability (cdf) and Quantile) functions, where the functions here partly use the 'Rmpfr' package and hence the underlying 'MPFR' and 'GMP' C libraries.
Depends: R (>= 3.6.0)
Imports: DPQ (>= 0.5-3), Rmpfr (>= 0.9-0), gmp, sfsmisc, stats, graphics, methods, utils
Suggests: Matrix
SuggestsNote: Matrix for its test-tools-1.R
License: GPL (>= 2)
Encoding: UTF-8
URL: https://specfun.r-forge.r-project.org/, https://r-forge.r-project.org/R/?group_id=611, https://r-forge.r-project.org/R/?group_id=611
BugReports: https://r-forge.r-project.org/tracker/?atid=2462&group_id=611
Author: Martin Maechler [aut, cre] (<https://orcid.org/0000-0002-8685-9910>)
Maintainer: Martin Maechler <maechler@stat.math.ethz.ch>

Index of help topics:
betaD94

Ding(1994) (non-central) Beta Distribution Functions

Description

The three functions "p" (cumulative distribution, CDF), "d" (density (PDF)), and "q" (quantile) use Ding(1994)'s algorithm A, B, and C, respectively, each of which implements a recursion formula using only simple arithmetic and log and exp.

These are particularly useful also for using with high precision "mpfr" numbers from the Rmpfr CRAN package.
Usage

```r
dbetaD94(x, shape1, shape2, ncp = 0, log = FALSE,
        eps = 1e-10, itrmx = 100000L, verbose = FALSE)
pbetaD94(q, shape1, shape2, ncp = 0, lower.tail = TRUE, log.p = FALSE,
        log_scale = (a * b > 0) && (a + b > 100 || c >= 500),
        eps = 1e-10, itrmx = 100000L, verbose = FALSE)
qbetaD94(p, shape1, shape2, ncp = 0, lower.tail = TRUE, log.p = FALSE,
        log_scale = (a * b > 0) && (a + b > 100 || c >= 500),
        delta = 1e-6,
        eps = delta^2,
        itrmax = 100000L,
        iterN = 1000L,
        verbose = FALSE)
```

Arguments

- `x`, `q`: numeric vector of values in [0, 1] as beta variates.
- `shape1`, `shape2`: the two shape parameters of the beta distribution, must be positive.
- `ncp`: the noncentrality parameter; by default zero for the (central) beta distribution; if positive, we have a noncentral beta distribution.
- `p`: numeric vector of probabilities, log()ged in case log.p is true.
- `log`, `log.p`: logical indicating if the density or probability values should be log()ged.
- `lower.tail`: logical indicating if the lower or upper tail probability should be computed, or for qbeta*() are provided.
- `eps`: a non-negative number specifying the desired accuracy for computing F() and f().
- `itrmax`: the maximal number of steps for computing F() and f().
- `delta`: [For qbeta*():] non-negative number indicating the desired accuracy for computing x_p (the root of pbeta * () == p), i.e., the convergence tolerance for the Newton iterations. This sets default eps = delta^2 which is sensible but may be too small, such that eps should be specified in addition to delta.
- `iterN`: [For qbeta*():] The maximal number of Newton iterations.
- `log_scale`: logical indicating if most of the computations should happen in log scale, which protects from “early” overflow and underflow but takes more computations. The current default is somewhat arbitrary, still derived from the facts that gamma(172) overflows to Inf already and exp(-750) underflows to 0 already.
- `verbose`: logical (or integer) indicating the amount of diagnostic output during computation; by default none.

Value

In all three cases, a numeric vector with the same attributes as `x` (or `q` respectively), containing (an approximation) to the corresponding beta distribution function.
Author(s)

Martin Maechler, notably log_scale was not part of Ding’s proposals.

References


See Also

pbeta. Package Rmpfr's pbetaI() needs both shape1 and shape2 to be integer but is typically more efficient than the current pbetaD94() implementation.

Examples

```r
## Low precision (eps, delta) values as "e.g." in Ding(94): --------------

## Compare with Table 3 of Baharev_et_al 2017 %% => ./qbBaha2017.Rd <<<<<<<<<
aa <- c(0.5, 1, 1.5, 2, 2.5, 3, 5, 10, 25)
bb <- c(1:15, 10*c(2:5, 10, 25, 50))

utime <-
  qbet <- matrix(NA_real_, length(aa), length(bb),
  dimnames = list(a = formatC(aa), b = formatC(bb)))
(doExtras <- DPQmpfr:::doExtras())
if(doExtras) qbetL <- utimeL <- utime

p <- 0.95
delta <- 1e-4
eps <- 1e-6

system.time <- function(expr)
  system.time(gcFirst = FALSE, expr)["user.self"]

system.time(
  for(ia in seq_along(aa)) {
    a <- aa[ia]; cat("\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\n-\н

## Low precision (eps, delta) values as "e.g." in Ding(94): --------------

## Compare with Table 3 of Baharev_et_al 2017 %% => ./qbBaha2017.Rd <<<<<<<<<

# system.time(.): ~ 1 sec (lynne i7-7700T, Fedora 32, 2020)
sum(print(table(round(1000*utime)))) # lynne .. :
```
### Description

Computes exact probabilities for the hypergeometric distribution (see, e.g., `dhyper()` in R), using package `gmp`'s big integer and rational numbers, notably `chooseZ()`.

### Usage

```r

```
dhyperQ(x, m, n, k)
phyperQ(x, m, n, k, lower.tail=TRUE)
phyperQall(m, n, k, lower.tail=TRUE)
```

### Arguments

- **x**
  - the number of white balls drawn without replacement from an urn which contains both black and white balls.
- **m**
  - the number of white balls in the urn.
- **n**
  - the number of black balls in the urn.
- **k**
  - the number of balls drawn from the urn, hence must be in \(0, 1, \ldots, m + n\).
- **lower.tail**
  - logical indicating if the lower or upper tail probability should be computed.

### Value

a bigrational (class “bigq” from package `gmp`) vector “as” \(x\); currently of length one (as all the function arguments must be “scalar”, currently).

### Author(s)

Martin Maechler

### See Also

- `chooseZ (pkg gmp)`, and R’s own `Hypergeometric`
Examples

```r
## dhyperQ() is simply
function (x, m, n, k)
{
  stopifnot(k - x == as.integer(k - x))
  chooseZ(m, x) * chooseZ(n, k - x) / chooseZ(m + n, k)
}

# a case where phyper(11, 15, 0, 12, log=TRUE) gave 'NaN'
(phyp5.0.12 <- cumsum(dhyperQ(0:12, m=15, n=0, k=12)))
stopifnot(phyp5.0.12 == c(rep(0, 12), 1))

for(x in 0:9)
  stopifnot(phyperQ(x, 10,7,8) +
            phyperQ(x, 10,7,8, lower.tail=FALSE) == 1)

(ph. <- phyperQall(m=10, n=7, k=8))
## Big Rational (bigq) object of length 8:
## [1] 1/2431 5/374 569/4862 2039/4862 3803/4862 4685/4862 4853/4862 1
stopifnot(identical(gmp::c_bigq(list(0, ph.)),
              1- c(phyperQall(10,7,8, lower.tail=FALSE), 0)))

(doExtras <- DPQmpfr:::doExtras())
if(doExtras) { # too slow for standard testing
  k <- 5000
  system.time(ph <- phyper(k, 2*k, 2*k, 2*k)) # 0 (< 0.001 sec)
  system.time(phQ <- phyperQ(k, 2*k, 2*k, 2*k)) # 5.6 (was 6.3) sec
  ## Relative error of R's phyper()
  stopifnot(print(gmp::asNumeric(1 - ph/phQ)) < 1e-14) # seen 1.063e-15
}
```

---

dnt

Non-central t-Distribution Density

dntJKBm is a fully Rmpfr-ified vectorized version of
dntJKBf() from DPQ which implements the summation formulas of Johnson, Kotz and Balakrishnan (1995), (31.15) on page 516 and (31.15') on p.519, the latter being typo-corrected for a missing factor $1/j!$.

Usage

dntJKBm(x, df, ncp, log = FALSE, M = 1000)
Arguments

- `x`, `df`, `ncp`  see R’s `dt()`; note that each can be of class "mpfr".
- `log` as in `dt()`, a logical indicating if \( \log(f(x, *)) \) should be returned instead of \( f(x, *) \).
- `M` the number of terms to be used, a positive integer.

Details

How to choose `M` optimally has not been investigated yet and is probably also a function of the precision of the first three arguments (see `getPrec` from `Rmpfr`).

Value

an `mpfr` vector of the same length as the maximum of the lengths of `x`, `df`, `ncp`.

Author(s)

Martin Maechler

References

Chapter 31, Section 5 Distribution Function, p.514 ff

See Also

dt.

Examples

```r
require(Rmpfr)

## [not too large, as dntJKBm() is currently somewhat slow]
(mt <- mpfr(tt <- seq(0, 9, by = 1), 128))
(mcp <- mpfr(ncp <- seq(0, 5, by = 1/2), 128))
dt3R <- outer(tt, ncp, dt, df = 3)
dt3M <- outer(mt, mcp, dntJKBm, df = 3, M = 128)# for speed

all.equal(dt3R, dt3M) # TRUE, and show difference
all.equal(dt3R, dt3M, tol=0) # 1.2e-12
```
Description

Utilities for package DPQmpfr

Usage

ldexp(f, E)

Arguments

f  
‘fraction’, as such with absolute value in [0.5, 1), but can be any numbers.

E  
integer-valued exponent(s).

Details

ldexp() is a simple wrapper, either calling DPQ::ldexp from DPQ or ldexpMpfr from the Rmpfr package.

\[ ldexp(f, E) := f \times 2^E, \]

computed accurately and fast on typical platforms with internally binary arithmetic.

Value

either a numeric or a "mpfr", depending on the type of \( f \), vector as (the recycled) combination of \( f \) and \( E \).

See Also

ldexp from package DPQ and ldexpMpfr from package Rmpfr.

Examples

ldexp(1:10, 2)
ldexp(Rmpfr::Const("pi", 96), -2:2) # = pi * (1/4 1/2 1 2 4)
Description

Bounds for $1 - \Phi(x)$, i.e., pnorm(x, *, lower.tail=FALSE), typically related to Mill’s Ratio.

Usage

    pnormL_LD10(x, lower.tail = FALSE, log.p = FALSE)
    pnormU_S53 (x, lower.tail = FALSE, log.p = FALSE)

Arguments

    x                positive (at least non-negative) numeric "mpfr" vector (or array).
    lower.tail, log.p   logical, see, e.g., pnorm().

Value

    vector/array/mpfr like x.

Author(s)

    Martin Maechler

References

    Lutz Duembgen (2010) Bounding Standard Gaussian Tail Probabilities; arXiv preprint 1012.2063,
    https://arxiv.org/abs/1012.2063

See Also

    pnorm. The same functions “numeric-only” are in my DPQ package.

Examples

    x <- seq(1/64, 10, by=1/64)
    px <- cbind(
        lQ = pnorm (x, lower.tail=FALSE, log.p=TRUE), 
        Lo = pnormL_LD10(x, lower.tail=FALSE, log.p=TRUE),
        Up = pnormU_S53 (x, lower.tail=FALSE, log.p=TRUE))
    matplot(x, px, type="l") # all on top of each other
    matplot(x, (D <- px[,2:3] - px[,1]), type="l") # the differences
    abline(h=0, lty=3, col=adjustcolor(1, 1/2))

    ## check they are lower and upper bounds indeed:
\begin{verbatim}
stopifnot(D[,"Lo"] < 0, D[,"Up"] > 0)
matplot(x[x>4], D[x>4,], type="l") # the differences
abline(h=0, lty=3, col=adjustcolor(1, 1/2))

### zoom out to larger x : [1, 1000]
x <- seq(1, 1000, by=1/4)
px <- cbind(
    1Q = pnorm (x, lower.tail=FALSE, log.p=TRUE),
    Lo = pnormL_LD10(x, lower.tail=FALSE, log.p=TRUE),
    Up = pnormU_S53 (x, lower.tail=FALSE, log.p=TRUE)
)  # all on top of each other
matplot(x, px, type="l") # all on top of each other
matplot(x, (D <- px[,2:3] - px[,1]), type="l") # the differences
abline(h=0, lty=3, col=adjustcolor(1, 1/2))

## check they are lower and upper bounds indeed:
table(D[,"Lo"] < 0) # no longer always true
table(D[,"Up"] > 0)
## not even when equality (where it's much better though):
table(D[,"Lo"] <= 0)
table(D[,"Up"] >= 0)

## *relative* differences:
matplot(x, (rD <- 1 - px[,2:3] / px[,1]), type="l", log = "x")
abline(h=0, lty=3, col=adjustcolor(1, 1/2))

## abs()
matplot(x, abs(rD), type="l", log = "xy", axes=FALSE, # NB: curves *cross*
    main = "relative differences 1 - pnormUL(x, *)/pnorm(x, *)")
legend("top", c("Low.Bnd(D10)", "Upp.Bnd(S53)"), bty="n", col=1:2, lty=1:2)
sfsmisc::eaxis(1, sub10 = 2)
sfsmisc::eaxis(2)
abline(h=(1:4)*2^-53, col=adjustcolor(1, 1/4))

### zoom out to LARGE x : ---------------------------

x <- 2^seq(0, 30, by = 1/64)
col4 <- adjustcolor(1:4, 1/2)
options(width = 111) -> oop # (nicely printing "tables")
if(FALSE)## or even HUGE:
x <- 2^seq(4, 513, by = 1/16)
px <- cbind(
    1Q = pnorm (x, lower.tail=FALSE, log.p=TRUE),
    a0 = dnorm(x, log=TRUE),
    a1 = dnorm(x, log=TRUE) - log(x),
    Lo = pnormL_LD10(x, lower.tail=FALSE, log.p=TRUE),
    Up = pnormU_S53 (x, lower.tail=FALSE, log.p=TRUE))
doLegTit <- function(col=1:4) {
    title(main = "relative differences 1 - pnormUL(x, *)/pnorm(x, *)")
    legend("top", c("phi(x)", "phi(x)/x", "Low.Bnd(D10)", "Upp.Bnd(S53)"),
            bty="n", col=col, lty=1:4)
}
## *relative* differences are relevant:
matplot(x, (rD <- 1 - px[-1] / px[1]), type="l", log = "x",

\end{verbatim}
...}

```r

# if(x[length(x)] > 1e150) # the "HUGE" case (not default)
# print(tail(cbind(x, px), 20))
##--> For very large x ~= 1e154, the approximations overflow *later* than pnorm() itself !!

## abs(rel.Diff) ---> can use log-log:
matplot(x, abs(rD), type="l", log = "xy", xaxt="n", yaxt="n"); doLegTit()
sfsmisc::eaxis(1, sub10=2)
sfsmisc::eaxis(2)
abline(h=(1:4)*2^-53, col=adjustcolor(1, 1/4))

## lower.tail=TRUE (w/ log.p=TRUE) works "the same" for x < 0:
require(Rmpfr)
x <- - 2^seq(0, 30, by = 1/64)
## ==
log1mexp <- Rmpfr::log1mexp # Rmpfr version >= 0.8-2 (2020-11-11 on CRAN)
px <- cbind(
  lQ = pnorm(x, lower.tail=TRUE, log.p=TRUE)
, a0 = log1mexp(- dnorm(-x, log=TRUE))
, a1 = log1mexp(- (dnorm(-x, log=TRUE) - log(-x)))
, Lo = log1mexp(- pnormL_LD10(-x, lower.tail=TRUE, log.p=TRUE))
, Up = log1mexp(- pnormU_S53 (-x, lower.tail=TRUE, log.p=TRUE))
) matplot(-x, (rD <- 1 - px[, -1] / px[, 1]), type="l", log = "x",
ylim = c(-1,1)/2^8, col=col4) ; doLegTit()
abline(h=0, lty=3, col=adjustcolor(1, 1/2))

## Comparison with Rmpfr::erf() / erfc() based pnorm():
## Set the exponential ranges to maximal -- to evade underflow as long as possible
.mpfr_erange_set(value = (1-2^-52) * .mpfr_erange(c("min.emin","max.emax")))
l2t <- seq(0, 32, by=1/4)
twos <- mpfr(2, 1024)^l2t
Qt <- pnorm(twos, lower.tail=FALSE)
pnU <- pnormU_S53 (twos, log.p=TRUE)
pnL <- pnormL_LD10(twos, log.p=TRUE)
logQt <- log(Qt)
M <- cbind(twos, Qt, logQt = logQt, pnU)
roundMpfr(M, 48)
dM <- asNumeric(cbind(dU = pnU - logQt, dL = logQt - pnL, # NB: the numbers are *negative*
  rdU= 1 - pnU/logQt, rdL = pnL/logQt - 1))
data.frame(l2t, dM)

## The bounds are ok (where Qt does not underflow): L < p < U :
stopifnot(pnU > pnL, pnU > logQt, (logQt > pnL)[Qt > 0])
roundMpfr(cbind(twos, pnL, pnU, D=pnU-pnL, relD=(pnU-pnL)/((pnU+pnL)/2)), 40)

## ----- R's pnorm() -- is it always inside [L, U] ?? ------------------------
nQt <- stats::pnorm(asNumeric(twos), lower.tail=FALSE, log.p=TRUE)
data.frame(l2t, check.names=FALSE, nQt
```
### Description

These functions provide the first terms of asymptotic series approximations to \( \text{pnorm}() \)’s (extreme) tail, from Abramowitz and Stegun’s 26.2.13 (p.932), or \( \text{qnorm}() \) where the approximations have been derived via iterative plugin using Abramowitz and Stegun’s formula.

### Usage

```r
pnormAsymp(x, k, lower.tail = FALSE, log.p = FALSE)
qnormAsymp(p, lp = .DT_Clog(p, lower.tail = lower.tail, log.p = log.p),
  order, M_2PI =,
  lower.tail = TRUE, log.p = missing(p))
```

### Arguments

- **x**: positive (at least non-negative) numeric vector.
- **k**: integer \( \geq 0 \) indicating how many terms the approximation should use; currently \( k \leq 5 \).
- **p**: numeric vector of probabilities, possibly transformed, depending on \( \log.p \). Does not need to be specified, if \( lp \) is instead.
- **lp**: numeric (vector) of \( \log(1-p) \) values; if not specified, computed from \( p \), depending on \( \text{lower.tail} \) and \( \log.p \).
- **order**: an integer in \( \{0, 1, \ldots, 5\} \), specifying the approximation order.
- **M_2PI**: the number \( 2\pi \) in the same precision as \( p \) or \( lp \), i.e., `numeric` or of class "mpfr".
- **lower.tail**: logical; if true, probabilities are \( P[X \leq x] \), otherwise upper tail probabilities, \( P[X > x] \).
- **log.p**: logical; if TRUE (default for \( \text{qnormAsymp}() \)), probabilities \( p \) are given as \( \log(p) \) in argument \( p \) or \( \log(1 - p) \) in \( lp \).
Details

see both help pages \texttt{pnormAsymp} and \texttt{qnormAsymp} from our package DPQ.

Value

vector/array/mpfr like first argument \texttt{x} or \texttt{p} or \texttt{lp}, respectively.

Author(s)

Martin Maechler

See Also

\texttt{pnorm}. The same functions “numeric-only” are in my DPQ package with more extensive documentation.

Examples

```r
require("Rmpfr") # (in strong dependencies of this pkg {DPQmpfr})
x <- seq(1/64, 10, by=1/64)
xm <- mpfr(x, 96)
"TODO"

## More extreme tails: ----------------------------------------------
##
## 1. pnormAsymp() ---------------------
x <- c((2:10)*2, 25, (3:9)*10, (1:9)*100, (1:8)*1000, (2:7)*5000)
xm <- mpfr(x, 256)
Px <- pnorm(xm, lower.tail = FALSE, log.p=TRUE)
PxA <- sapplyMpfr(setNames(0:5, paste("k =",0:5)), pnormAsymp, x=xm, lower.tail = FALSE, log.p=TRUE)
if(interactive())
  roundMpfr(PxA, 40)
  # rel.errors :
  relE <- asNumeric(1 - PxA/Px)
options(width = 99) -> oop # (nicely printing the matrices)
cbind(lx, relE)
matplot(lx, abs(relE), type="b", cex = 1/2, log="xy", pch=as.character(0:5),
  axes=FALSE,
  main = "|relE( <pnormAsymp(lx, k=*, lower.tail=FALSE, log.p=TRUE) )|")
sfsmisc::eaxis(1, subl0=2); sfsmisc::eaxis(2)
legend("bottom", paste("k =", 0:5), col=1:6, lty=1:5,
  pch = as.character(0:5), pt.cex=1/2, bty="n")
## NB: rel.Errors go down to 7e-59 ==> need precision of ~-log2(7e-59) ~ 193.2 bits

## 2. qnormAsymp() ---------------------
QPx <- sapplyMpfr(setNames(0:5, paste("k =",0:5)),
  function(k) qnormAsymp(Px, order=k, lower.tail = FALSE, log.p=TRUE))
(relE.q <- asNumeric(QPx/lx - 1))
  # note how consistent the signs are (!) <=> have upper/lower bounds
```
Computable "accurate" \texttt{qbeta()} values from Baharev et al (2017)'s Program.

Usage

\begin{verbatim}
data("qbBaha2017")
\end{verbatim}

Details

MM constructed this data as follows (TODO: say more..):

\begin{verbatim}
ff <- "~/R/MM/NUMERICS/dpq-functions/beta-gamma-etc/Baharev_et_al-2017_table3.txt"
qbB2017 <- t( data.matrix(read.table(ff)) )
dimnames(qbB2017) <- dimnames(qbet)
saveRDS(qbB2017, "./.qbBaha2017.rds")
\end{verbatim}
Source


The paper mentions the first author’s ‘github’ repos where source code and executables are available from: https://github.com/baharev/mindiffver/

Examples

data(qbBaha2017)
str(qbBaha2017)
str(ab <- lapply(dimnames(qbBaha2017), as.numeric))
stopifnot(ab$a == c((1:6)/2, 5, 10, 25),
ab$b == c(1:15, 10*c(2:5, 10, 25, 50)))
matplot(ab$b, t(qbBaha2017)
[,9:1], type="l", log = "x", xlab = "b",
ylab = "qbeta(.95, a,b)",
main = "Guaranteed accuracy 95% percentiles of Beta distribution")
legend("right", paste("a = ", format(ab$a)),
lyt=1:5, col=1:6, bty="n")

## Relative error of R's qbeta() -- given that the table only shows 6
## digits, there is *no* relevant error: R's qbeta() is accurate enough:
x.ab <- do.call(expand.grid, ab)
matplot(ab$b, 1 - t(qbeta(0.95, x.ab$a, x.ab$b) / qbBaha2017),
main = "rel.error of R's qbeta() -- w/ 6 digits, it is negligible",
ylab = "1 - qbeta() / true",
type = "l", log="x", xlab="b")
abline(h=0, col=adjustcolor("gray", 1/2))

---

stirlerrM

Stirling Formula Approximation Error

Description

Compute the log() of the error of Stirling’s formula for n!. Used in certain accurate approximations of (negative) binomial and Poisson probabilities.

stirlerrM() currently simply uses the direct mathematical formula, based on lgamma(), adapted for use with mpfr-numbers.

Usage

stirlerrM(n, minPrec = 128L)
stirlerrSer(n, k)
stirlerrM

Arguments

- **n**: numeric or “numeric-alike” vector, typically “large” positive integer or half integer valued, here typically an "mpfr"-number vector.
- **k**: integer *scalar*, now in 1:11.
- **minPrec**: minimal precision (in bits) to be used when coercing number-alikes, say, biginteger (*bigz*) to "mpfr".

Details

Stirling’s approximation to $n!$ has been

$$n! \approx \left(\frac{n}{e}\right)^n \sqrt{2\pi n},$$

where by definition the error is the difference of the left and right hand side of this formula, in log-scale,

$$\delta(n) = \log \Gamma(n+1) - n \log(n) + n - \log(2\pi n)/2.$$

See the vignette *log1pmx, bd0, stirlerr, ...* from package *DPQ*, where the series expansion of $\delta(n)$ is used with 11 terms, starting with

$$\delta(n) = \frac{1}{12n} - \frac{1}{360n^3} + \frac{1}{1260n^5} \pm O(n^{-7}).$$

Value

a numeric or other “numeric-alike” class vector, e.g., *mpfr*, of the same length as $n$.

Note

In principle, the direct formula should be replaced by a few terms of the series in powers of $1/n$ for large $n$, but we assume using high enough precision for $n$ should be sufficient and “easier”.

Author(s)

Martin Maechler

References

Catherine Loader, see *dbinom*;

Martin Maechler (2021) *log1pmx, bd0, stirlerr* – Computing Poisson, Binomial, Gamma Probabilities in R. [https://CRAN.R-project.org/package=DPQ/vignettes/log1pmx-etc.pdf](https://CRAN.R-project.org/package=DPQ/vignettes/log1pmx-etc.pdf)

See Also

dbinom, stirlerr() in package *DPQ* which is a pure R version R’s mathlib-internal C function.
Examples

### ---------------- Regular R double precision -------------------------------

def n <- c(1:10, 15, 20, 30, 50*(1:6), 100*(4:9), 10^(3:12))
(stE <- stirllerrM(n))  # direct formula is *not* good when n is large:
require(graphics)
plot(stirlerrM(n) ~ n, log = "x", type = "b", xaxt="n")
sfsmisc::eaxis(1, sub10=3)
for(k in 1:8) lines(n, stirlerrSer(n, k), col = k+1)
legend("top", c("stirlerrM(n)", paste0("stirlerrSer(n, k=" , 1:8, ")")),
       pch=c(1,rep(NA,8)), col=1:(8+1), lty=1, bty="n")

## for larger n, current values are even *negative* ==> dbl prec *not* sufficient
## y in log-scale [same conclusion]
plot (stirlerrM(n) ~ n, log = "xy", type = "b", ylim = c(1e-13, 0.08))
for(k in 1:8) lines(n, stirlerrSer(n, k), col = k+1)
legend("topright", c("stirlerrM(n)", paste0("stirlerrSer(n, k=" , 1:8, ")")),
       pch=c(1,rep(NA,8)), col=1:(8+1), lty=1, ncol=2, bty="n")

## the numbers:
options(digits=4, width=111)

stEmat. <- cbind(sM = stirllerrM(n),
                 sapply(setNames(1:8, paste0("k=" , 1:8)),
                        function(k) stirlerrSer(n=n, k=k)))

stEmat.

for printing n=<nice>:
N <- Rmpfr::asNumeric
dfm <- function(n, mm) data.frame(n=formatC(N(n)), N(mm), check.names=FALSE)

## relative differences:
dfm(n, stEmat.[,-1]/stEmat.[,1] - 1)
  # => stirllerrM() (with dbl prec) deteriorates after ~ n = 200--500

dfm(n, stEmat.[,-(1+8)]/stEmat.[,1+8] - 1)

### ---------------- MPFR High Accuracy -------------------------------

stopifnot(require(gmp),
           require(Rmpfr))
n <- as.bigz(n.)

## now repeat everything .. from above ... FIXME shows bugs !
## fully accurate using big rational arithmetic
class(stEserQ <- sapply(setNames(1:8, paste0("k=" , 1:8)),
                        function(k) stirlerrSer(n=n, k=k)))  # list ..
stopifnot(sapply(stEserQ, class) == "bigq")  # of exact big rationals

str(stEsQM <- lapply(stEserQ, as, Class="mpfr"))# list of 8; each prec. 128..702

stEsQM. <- lapply(stEserQ, .bigq2mpfr, precB = 512)  # constant higher precision
stEsQMm <- sapply(stEserQ, asNumeric)  # a matrix

stEM <- stirllerrM(mpfr(n, 128))  # now ok (loss of precision, but still ~ 10 digits correct)
stirlerrM(mfr(n, 4096)) # assume "perfect"
## >>> what's the accuracy of the 128-bit 'stEM'?
N <- asNumeric # short
dfm(n, stEM/stEM4k - 1)
## 29 1e+06 4.470e-25
## 30 1e+07 -7.405e-23
## 31 1e+08 -4.661e-21
## 32 1e+09 -7.693e-20
## 33 1e+10 3.452e-17 (still ok)
## 34 1e+11 -3.472e-15 << now start losing
## 35 1e+12 -3.138e-13 <<<
## same conclusion via number of correct (decimal) digits:
dfm(n, log10(abs(stEM/stEM4k - 1)))
plot(N(-log10(abs(EM4k - 1))) ~ N(n), type="o", log="x",
  xlab = quote(n), main = "(# correct digits) of 128-bit stirlerrR(n)"
ubits <- c(128, 52) # above 128-bit and double precision
abline(h = ubits* log10(2), lty=2)
text(1, ubits* log10(2), paste0(ubits,"-bit"), adj=c(0,0))
stopifnot(identical(stirlerrM(n), stEM)) # for bigz & bigq, we default to precBits = 128
all.equal(roundMpfr(stEM4k, 64),
  stirlerrSer (n, 8)) # 0.00212 .. because of 1st few n. ==> drop these
all.equal(roundMpfr(stEM4k, 64)[n. >= 3], stirlerrSer (n.[n. >= 3], 8)) # 6.238e-8
plot(asNumeric(abs(stirlerrSer(n, 8) - stEM4k)) ~ n., log="xy", type="b", main="absolute error of stirlerrSer(n, 8) & (n, 5)"
  legend("top", c("stirlerrM(n)", paste0("stirlerrSer(n, k="), 1:8, ","))),
    pch=c(1,rep(NA,8)), col=1:(8+1), lty=1, bty="n")
plot(asNumeric(stirlerrM(n)) ~ n., log = "x", type = "b")
for(k in 1:8) lines(n, stirlerrSer(n, k), col = k+1)
legend("top", c("stirlerrM(n)", paste0("stirlerrSer(n, k="), 1:8, ","))),
  pch=c(1,rep(NA,8)), col=1:(8+1), lty=1, bty="n")
## all "looks" perfect (so we could skip this)
## the numbers ...
## >>> FIXME a list instead of mpfrMatrix ... FIXME
## FIXME ... asNumeric() needed or as(n, "mpfr") or ....
ks <- 1:8 # k <= 5 === FIXME --- use DPQ's version !!
stir1s.l <- lapply(setNames(ks, paste0("k="), 1:8, ",")),
  function(k) stirrlerSer(n=n, k=k))
## ==> an mpfrMatrix of dim 35 x 5 :
mss <- do.call(cbind, lapply(stir1s.l, mpfr, precBits=256))
stEmat <- cbind(sM = stEM4k, mss)
signif(asNumeric(stEmat), 6) # so it prints nicely
## print *relative errors* nicely :

## simple double precision version of direct formula (cancellation for n >> 1 !):
stE <- stirlerM(n.)
dfm(n, cbind(stEmat[-1], dbl=stE)/stEM4k - 1)
## relative differences:
dfm(n, stEmat[-1] / stEmat[1] - 1)
dfm(n., stEmat[-(1+8)]/ stEmat[1+8] - 1)
Index

* arithmetic
  DPQmpfr-utils, 9
* arith
  stirlerrM, 16
* datasets
  qbBaha2017, 15
* distribution
  betaD94, 3
dhyperQ, 6
dnt, 7
DPQmpfr-package, 2
pnormLU, 10
pqnormAsymp, 13
* math
  betaD94, 3
dnt, 7
DPQmpfr-package, 2
stirlerrM, 16
* package
  DPQmpfr-package, 2

array, 10
betaD94, 3
bigz, 17
chooseZ, 6
dbetaD94(betaD94), 3
dbinom, 17
dhyper, 6
dhyperQ, 6
dnt, 7
dntJKbf, 7
dntJKBm (dnt), 7
DPQ, 3
DPQmpfr (DPQmpfr-package), 2
DPQmpfr-package, 2
DPQmpfr-utils, 9
dt, 8
exp, 3, 4
gamma, 4
getPrec, 8
Hypergeometric, 6
ldexp, 9
ldexp (DPQmpfr-utils), 9
ldexp Mpfr, 9
lgamma, 16
log, 3, 4, 16
mpfr, 8–10, 13, 16, 17
numeric, 13
pbeta, 5
pbetaD94 (betaD94), 3
pbetaI, 5
phyperQ (dhyperQ), 6
phyperQall (dhyperQ), 6
pnorm, 10, 13, 14
pnormAsymp, 14
pnormAsymp (pqnormAsymp), 13
pnormLU_LD10 (pnormLU), 10
pnormLU, 10
pnormLU_S53 (pnormLU), 10
pqnormAsymp, 13
qbBaha2017, 15
qbeta, 15
qbetaD94 (betaD94), 3
qnorm, 13
qnormAsymp, 14
qnormAsymp (pqnormAsymp), 13
Rmpfr, 3
stirlerr, 17
stirlerrM, 16
stirlerrSer (stirlerrM), 16
str, 15