Package 'DPQmpfr'

May 17, 2021

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

Version 0.3-1

Date 2021-05-17

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 (>= 0.4-3), Rmpfr, gmp, stats, graphics, methods, utils

Suggests sfsmisc, Matrix

SuggestsNote Matrix for its test-tools-1.R

License GPL (>= 2)

Encoding UTF-8

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 2021-05-17 21:10:02 UTC

R topics documented:

DPQmpfr-package *DPQ (Density, Probability, Quantile) Distribution Computations using MPFR*

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:	0.3-1
Date:	2021-05-17
Authors@R:	person("Martin","Maechler", role=c("aut","cre"), email="maechler@stat.math.ethz.ch", comment = c(ORCI
Description:	An extension to the 'DPQ' package with computations for 'DPQ' (Density (pdf), Probability (cdf) and Quant
Depends:	R (>= 3.6.0)
Imports:	DPQ (>= 0.4-3), Rmpfr, gmp, stats, graphics, methods, utils
Suggests:	sfsmisc, Matrix
SuggestsNote:	Matrix for its test-tools-1.R
License:	GPL (>= 2)
Encoding:	UTF-8
Author:	Martin Maechler [aut, cre] (<https://orcid.org/0000-0002-8685-9910>)
Maintainer:	Martin Maechler <maechler@stat.math.ethz.ch>

Index of help topics:

Author(s)

NA

Maintainer: NA

See Also

Packages DPQ, Rmpfr are both used by this package.

Examples

```
## An example how mpfr-numbers  "just work" with reasonable R functions:
.srch <- search() ; doAtt <- is.na(match("Rmpfr:package", .srch))
if(doAtt) require(Rmpfr)
nu.s <- 2^seqMpfr(mpfr(-30, 64), mpfr(100, 64), by = 1/mpfr(4, 64))
b0 <- DPQ::b_chi(nu.s)
b1 <- DPQ::b_chi(nu.s, one.minus=TRUE)
stopifnot(inherits(b0,"mpfr"), inherits(b1, "mpfr"),
          b0+b1 == 1,  diff(log(b1)) < 0)
plot(nu.s,          log(b1),  type="l", log="x")
plot(nu.s[-1], diff(log(b1)), type="l", log="x")
if(doAtt) # detach the package(s) we've attached above
  for(pkg in setdiff(search(), .srch)) detach(pkg, character.only=TRUE)
```

---

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

```
dbetaD94(x, shape1, shape2, ncp = 0, log = FALSE,
         eps = 1e-10, itrmax = 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, itrmax = 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 **N**ewton 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, containing (an approximation) to the correponding beta distribution function.

Author(s)

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

References

Cherng G. Ding (1994) On the computation of the noncentral beta distribution. *Computational Statistics & Data Analysis* **18**, 449–455.

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

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

system.time(
for(ia in seq_along(aa)) {
    a <- aa[ia]; cat("\n--==--\na=",a,":\n")
    for(ib in seq_along(bb)) {
        b <- bb[ib]; cat("\n>> b=",b,"\n")
        utime [ia, ib] <- system.t.usr(
         qbet[ia, ib] <-   qbetaD94(p, a, b, ncp = 0, delta=delta, eps=eps, verbose = 2))
        if(doExtras)
          utimeL[ia, ib] <- system.t.usr(
           qbetL[ia, ib] <-   qbetaD94(p, a, b, ncp = 0, delta=delta, eps=eps,
                                       verbose = 2, log_scale=TRUE))
    }
    cat("\n")
}
)# system.time(.): ~ 1 sec (lynne i7-7700T, Fedora 32, 2020)
sum(print(table(round(1000*utime)))) # lynne .. :
##  0  1  2  3  4  5  6  7  8  9 10 11 14 15 16 29
## 53 94 15  3  3 12  2  2  2  2  1  2  3  1  2  1
## [1] 198
if(doExtras) print(sum(print(table(round(1000*utimeL))))) # lynne .. :
```

---

dhyperQ *Exact Hypergeometric Distribution Probabilites*

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

```
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

```
## 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(0, ph.),
                    1- c(phyperQall(10,7,8, lower.tail=FALSE), 0)))
```

```
 ## 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)) # 6.3 sec
## Relative error of R's phyper()
gmp::asNumeric(1 - ph/phQ) # 1.063e-15 -- very small
```

---

dnt *Non-central t-Distribution Density*

Description

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

Johnson, N.L., Kotz, S. and Balakrishnan, N. (1995) Continuous Univariate Distributions Vol~2, 2nd ed.; Wiley.
Chapter 31, Section *5 Distribution Function*, p.514 ff

See Also

dt.

Examples

```
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
```

---

pnormLU *Bounds for 1-Phi(.) – Mill's Ratio related Bounds for pnorm()*

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 :
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(
    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 :
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)
if(FALSE)## or even HUGE:
   x <- 2^seq(4, 513, by = 1/16)
px <- cbind(
    lQ = 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",
            ylim = c(-1,1)/2^8, col=col4) ; doLegTit()
abline(h=0, lty=3, col=adjustcolor(1, 1/2))

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, 40)
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
         , "L <= p" = c(" ", "W")[2 -(pnL <= nQt)]
         , "p <= U" = c(" ", "W")[2- (nQt <= pnU)])
## ==> pnorm() is *outside* sometimes for l2t >= 7.25; always as soon as l2t >= 9.25

## *but* the relative errors are around c_epsilon  in all these cases :
plot (2^l2t, asNumeric(abs(nQt-pnL)/abs(pnU)), type="o", cex=1/4, log="xy", axes=FALSE)
sfsmisc::eaxis(1, sub10 = 2)
sfsmisc::eaxis(2)
lines(2^l2t, asNumeric(abs(nQt-pnU)/abs(pnU)), type="o", cex=1/4, col=2)
abline(h=c(1:4)*2^-53, lty=2, col=adjustcolor(1, 1/4))
```

---

qbBaha2017 *Accurate qbeta() values from Baharev et al (2017)'s Program*

Description

Compuate "accurate" `qbeta()` values from Baharev et al (2017)'s Program.

Usage

```
data("qbBaha2017")
```

Format

FIXME: Their published table only shows 6 digits, but running their (32-bit statically linked) Linux executable '`mindiffver`' (from their github repos, see "source") with their own '`input.txt`' gives *12* digits accuracy, which we should be able to increase even more,

see https://github.com/baharev/mindiffver/blob/master/README.md

A numeric matrix, 9×22 with guaranteed accuracy `qbeta(0.95,a,b)` values, for $a = 0.5, 1, 1.5, 2, 2.5, 3, 5, 10, 25$ and $b =$ with `str()`

```
num [1:9, 1:22] 0.902 0.95 0.966 0.975 0.98 ...
- attr(*, "dimnames")=List of 2
 ..$ a: chr [1:9] "0.5" "1" "1.5" "2" ...
 ..$ b: chr [1:22] "1" "2" "3" "4" ...
```

Details

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

```
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")
```

Source

This matrix comprises all entries of Table 3, p. 776 of
Baharev, A., Schichl, H. and Rév, E. (2017) Computing the noncentral-F distribution and the power of the F-test with guaranteed accuracy; *Comput. Stat.* **32**(2), 763–779. doi: 10.1007/s00180016-07013

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)),
       lty=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)
```

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

See Also

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

Examples

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

n <- n. <- c(1:10, 15, 20, 30, 50*(1:6), 100*(4:9), 10^(3:12))
(stE <- stirlerrM(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 = stirlerrM(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)
    # => stirlerrM() {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  <- stirlerrM(mpfr(n, 128)) # now ok (loss of precision, but still ~ 10 digits correct)
stEM4k <- stirlerrM(mpfr(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(stEM/stEM4k - 1))) ~ N(n), type="o", log="x",
     xlab = quote(n), main = "#{correct digits} of 128-bit stirlerrM(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)")
abline(h = 2^-52, lty=2); text(1, 2^-52, "52-bits", adj=c(1,-1)/8)
lines(asNumeric(abs(stirlerrSer(n., 5) - stEM4k)) ~ n., col=2)

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")

## y in log-scale
plot(asNumeric(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("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(*, "mpfr") or ...
ks <- 1:8 ## k <= 5 === FIXME --- use DPQ's  version !!
stirlS.l <- lapply(setNames(ks, paste0("k=",ks)),
                   function(k) stirlerrSer(n=n, k=k))
## ==> an mpfrMatrix of dim  35 x 5 :
mss <- do.call(cbind, lapply(stirlS.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 <- stirlerrM(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