Package 'DPQ'

May 5, 2021

Title Density, Probability, Quantile ('DPQ') Computations

Version 0.4-3

Date 2021-05-05

Description Computations for approximations and alternatives for the 'DPQ' (Density (pdf), Probability (cdf) and Quantile) functions for probability distributions in R.
Primary focus is on (central and non-central) beta, gamma and related distributions such as the chi-squared, F, and t.
--
This is for the use of researchers in these numerical approximation implementations, notably for my own use in order to improve standard R pbeta(), qgamma(), ..., etc: {"``dpq"'-functions}.

Depends R (>= 3.6.0)

Imports stats, graphics, methods, utils, sfsmisc (>= 1.1-9)

Suggests Rmpfr, DPQmpfr, gmp, Matrix, MASS, mgcv, scatterplot3d, akima

SuggestsNote Matrix only for its ``test-tools-1.R"; mgcv,scatt..,akima: some tests/

License GPL (>= 2)

Encoding UTF-8

R topics documented:

DPQ-package *Density, Probability, Quantile ('DPQ') Computations*

Description

Computations for approximations and alternatives for the 'DPQ' (Density (pdf), Probability (cdf) and Quantile) functions for probability distributions in R. Primary focus is on (central and non-central) beta, gamma and related distributions such as the chi-squared, F, and t. – This is for the use of researchers in these numerical approximation implementations, notably for my own use in order to improve standard R pbeta(), qgamma(), ..., etc: '"dpq"'-functions.

Details

The DESCRIPTION file:

Package:	DPQ
Title:	Density, Probability, Quantile ('DPQ') Computations
Version:	0.4-3
Date:	2021-05-05
Authors@R:	c(person("Martin","Maechler", role=c("aut","cre"), email="maechler@stat.math.ethz.ch", comment = c(
Description:	Computations for approximations and alternatives for the 'DPQ' (Density (pdf), Probability (cdf) and Qu
Depends:	R (>= 3.6.0)
Imports:	stats, graphics, methods, utils, sfsmisc (>= 1.1-9)
Suggests:	Rmpfr, DPQmpfr, gmp, Matrix, MASS, mgcv, scatterplot3d, akima
SuggestsNote:	Matrix only for its "test-tools-1.R"; mgcv,scatt..,akima: some tests/
License:	GPL (>= 2)
Encoding:	UTF-8
Author:	Martin Maechler [aut, cre] (<https://orcid.org/0000-0002-8685-9910>), Morten Welinder [ctb] (pgamma
Maintainer:	Martin Maechler <maechler@stat.math.ethz.ch>

Index of help topics:

dntJKBf1	Non-central t-Distribution Density - Algorithms and Approximations
dtWV	Noncentral t Distribution Density by W.V.
format01prec	Format Numbers in [0,1] with "Precise" Result
hyper2binomP	Transform Hypergeometric Distribution Parameters to Binomial Probability
lbetaM	(Log) Beta Approximations
lfastchoose	R versions of Simple Formulas for Logarithmic Binomial Coefficients
lgamma1p	Accurate 'log(gamma(a+1))'
lgammaAsymp	Asymptotic Log Gamma Function
log1mexp	Compute log(1 - exp(-a)) and log(1 + exp(x)) Numerically Optimally
log1pmx	Accurate 'log(1+x) - x'
logcf	Continued Fraction Approximation of Log-Related Power Series
logspace.add	Logspace Arithmetix - Addition and Subtraction
lssum	Compute Logarithm of a Sum with Signed Large Summands
lsum	Properly Compute the Logarithm of a Sum (of Exponentials)
newton	Simple R level Newton Algorithm, Mostly for Didactical Reasons
p1l1	Numerically Stable p1l1(t) = (t+1)*log(1+t) - t
pbetaRv1	Pure R Implementation of Old pbeta()
pchisqV	Wienergerm Approximations to (Non-Central) Chi-squared Probabilities
phyper1molenaar	Molenaar's Normal Approximations to the Hypergeometric Distribution
phyperAllBin	Compute Hypergeometric Probabilities via Binomial Approximations
phyperApprAS152	Normal Approximation to cumulative Hyperbolic Distribution - AS 152
phyperBin.1	HyperGeometric Distribution via Approximate Binomial Distribution
phyperBinMolenaar	HyperGeometric Distribution via Molenaar's Binomial Approximation
phyperIbeta	Pearson's incomplete Beta Approximation to the Hyperbolic Distribution
phyperPeizer	Peizer's Normal Approximation to the Cumulative Hyperbolic
phyperR	R-only version of R's original phyper() algorithm
phyperR2	Pure R version of R's C level phyper()
phypers	The Four (4) Symmetric phyper() calls.
pl2curves	Plot 2 Noncentral Distribution Curves for Visual Comparison
pnbetaAppr2	Noncentral Beta Probabilities
pnchi1sq	(Probabilities of Non-Central Chi-squared Distribution for Special Cases
pnchisq	(Approximate) Probabilities of Non-Central Chi-squared Distribution

pnormAsymp	Asymptotic Approxmation of (Extreme Tail) 'pnorm()'
pnormL_LD10	Bounds for 1-Phi(.) - Mill's Ratio related Bounds for pnorm()
pntR	Non-central t Probability Distribution - Algorithms and Approximations
ppoisErr	Direct Computation of 'ppois()' Poisson Distribution Probabilities
qbetaAppr	Compute (Approximate) Quantiles of the Beta Distribution
qbinomR	Pure R Implementation of R's qbinom() with Tuning Parameters
qchisqAppr	Compute Approximate Quantiles of the Chi-Squared Distribution
qgammaAppr	Compute (Approximate) Quantiles of the Gamma Distribution
qnbinomR	Pure R Implementation of R's qnbinom() with Tuning Parameters
qnchisqAppr	Compute Approximate Quantiles of Noncentral Chi-Squared Distribution
qnormAppr	Approximations to 'qnorm()', i.e., z_alpha
qnormR	Pure R version of R's 'qnorm()' with Diagnostics and Tuning Parameters
qpoisR	Pure R Implementation of R's qpois() with Tuning Parameters
qtAppr	Compute Approximate Quantiles of Non-Central t Distribution
r_pois	Compute Relative Size of i-th term of Poisson Distribution Series

Further information is available in the following vignettes:

Noncentral-Chisq	Noncentral Chi-Squared Probabilities – Algorithms in R (source)
comp-beta	Computing Beta(a,b) for Large Arguments (source)
log1pmx-etc	log1pmx, bd0, stirlerr - Probability Approximations for R (source)

An important goal is to investigate diverse algorithms and approximations of R's own density (d*()), probability (p*()), and quantile (q*()) functions, notably in "border" cases where the traditional published algorithms have shown to be suboptimal, not quite accurate, or even useless.

Examples are border cases of the beta distribution, or **non-central** distributions such as the non-central chi-squared and t-distributions.

Author(s)

Principal author and maintainer: NA

See Also

The package **DPQmpfr** (not yet on CRAN), which builds on this package and on **Rmpfr**.

Examples

```
## Show problem in R's non-central t-distrib. density (and approximations):
example(dntJKBf)
```

`algdiv` *Compute log(gamma(b)/gamma(a+b)) when b >= 8*

Description

Computes

$$algdiv(a,b) := \log \frac{\Gamma(b)}{\Gamma(a+b)} = \log\Gamma(b) - \log\Gamma(a+b) = \texttt{lgamma(b) - lgamma(a+b)}$$

in a numerically stable way.

This is an auxiliary function in R's (TOMS 708) implementation of `pbeta()`, aka the incomplete beta function ratio.

Usage

```
algdiv(a, b)
```

Arguments

`a, b`	numeric vectors which will be recycled to the same length.

Details

Note that this is also useful to compute the Beta function

$$B(a,b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}.$$

Clearly,

$$\log B(a,b) = \log\Gamma(a) + algdiv(a,b) = \log\Gamma(a) - logQab(a,b)$$

```
In our ../tests/qbeta-dist.R we look into computing log(p*Beta(p,q)) accurately for p << q
       ---------------------
We are proposing a nice solution there.

How is this related to algdiv() ?
```

Value

a numeric vector of length `max(length(a),length(b))` (if neither is of length 0, in which case the result has length 0 as well).

Author(s)

Didonato, A. and Morris, A., Jr, (1992); `algdiv()`'s C version from the R sources, authored by the R core team; C and R interface: Martin Maechler

References

Didonato, A. and Morris, A., Jr, (1992) Algorithm 708: Significant digit computation of the incomplete beta function ratios, *ACM Transactions on Mathematical Software* **18**, 360–373.

See Also

`gamma`, `beta`; my own `logQab_asy()`.

Examples

```
Qab <- algdiv(2:3, 8:14)
cbind(a = 2:3, b = 8:14, Qab) # recycling with a warning

## algdiv()  and my  logQab_asy()  give *very* similar results for largish b:
all.equal( -   algdiv(3, 100),
           logQab_asy(3, 100), tol=0) # 1.283e-16 !!
(lQab <- logQab_asy(3, 1e10))
## relative error
1 + lQab/ algdiv(3, 1e10) # 0 (64b F 30 Linux; 2019-08-15)
```

Bern *Bernoulli Numbers*

Description

Return the n-th Bernoulli number B_n, (or B_n^+, see the reference), where $B_1 = +\frac{1}{2}$.

Usage

```
Bern(n, verbose = getOption("verbose", FALSE))
```

Arguments

`n`	integer, $n \geq 0$.
`verbose`	logical indicating if computation should be traced.

Value

The number B_n of type `numeric`.

A side effect is the *caching* of computed Bernoulli numbers in the hidden `environment` `.bernoulliEnv`.

Author(s)

Martin Maechler

References

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

See Also

Bernoulli in **Rmpfr** in arbitrary precision via Riemann's ζ function.

The next version of package **gmp** is to contain `BernoulliQ()`, providing exact Bernoulli numbers as big rationals (class `"bigq"`).

Examples

```
(B.0.10 <- vapply(0:10, Bern, 1/2))
## [1]  1.00000000 +0.50000000  0.16666667  0.00000000 -0.03333333  0.00000000
## [7]  0.02380952  0.00000000 -0.03333333  0.00000000  0.07575758
if(requireNamespace("MASS")) {
  print( MASS::fractions(B.0.10) )
  ## 1  +1/2   1/6    0  -1/30     0  1/42     0 -1/30     0  5/66
}
```

---

b_chi *Compute $E[\chi_\nu]/\sqrt{\nu}$ useful for t- and chi-Distributions*

Description

$$b_\chi(\nu) := E[\chi(\nu)]/\sqrt{\nu} = \frac{\sqrt{2/\nu}\Gamma((\nu+1)/2)}{\Gamma(\nu/2)},$$

where $\chi(\nu)$ denotes a chi-distributed random variable, i.e., the square of a chi-squared variable, and $\Gamma(z)$ is the Gamma function, `gamma()` in R.

This is a relatively important auxiliary function when computing with non-central t distribution functions and approximations, specifically see Johnson et al.(1994), p.520, after (31.26a), e.g., our `pntJW39()`.

Its logarithm,

$$lb_\chi(\nu) := log\Big(\frac{\sqrt{2/\nu}\Gamma((\nu+1)/2)}{\Gamma(\nu/2)}\Big),$$

is even easier to compute via `lgamma` and `log`, and I have used Maple to derive an asymptotic expansion in $\frac{1}{\nu}$ as well.

Note that $lb_\chi(\nu)$ also appears in the formula for the t-density (`dt`) and distribution (tail) functions.

Usage

```
b_chi     (nu, one.minus = FALSE, c1 = 341, c2 = 1000)
b_chiAsymp(nu, order = 2, one.minus = FALSE)
#lb_chi    (nu, ......) # not yet
lb_chiAsymp(nu, order)

c_dt(nu)     # warning("FIXME: current c_dt() is poor -- base it on lb_chi(nu) !")
c_dtAsymp(nu)  # deprecated in favour of lb_chi(nu)
c_pt(nu)     # warning("use better c_dt()") %---> FIXME deprecate even stronger ?
```

Arguments

nu	non-negative numeric vector of degrees of freedom.
one.minus	logical indicating if $1 - b()$ should be returned instead of $b()$.
c1, c2	boundaries for different approximation intervals used: for 0 < nu <= c1, internal b1() is used, for c1 < nu <= c2, internal b2() is used, and for c2 < nu, the b_chiAsymp() function is used, (and you can use that explicitly, also for smaller nu). FIXME: c1 and c2 were defined when the only asymptotic expansion known to me was the order = 2 one. A future version of b_chi will *very likely* use b_chiAsymp(*,order) for higher orders, and the c1 and c2 arguments will change, possibly be abolished.
order	the polynomial order in $\frac{1}{\nu}$ of the asymptotic expansion of $b_\chi(\nu)$ for $\nu \to \infty$. The default, order = 2 corresponds to the order you can get out of the Abramowitz and Stegun (6.1.47) formula. Higher order expansions were derived using Maple by Martin Maechler in 2002, see below, but implemented in b_chiAsymp() only in 2018.

Details

One can see that b_chi() has the properties of a CDF of a continuous positive random variable: It grows monotonely from $b_\chi(0) = 0$ to (asymptotically) one. Specifically, for large nu, b_chi(nu) = b_chiAsymp(nu) and

$$1 - b_\chi(\nu) \sim \frac{1}{4\nu}.$$

More accurately, derived from Abramowitz and Stegun, 6.1.47 (p.257) for a= 1/2, b=0,

$$\Gamma(z + 1/2)/\Gamma(z) \sim \sqrt(z) * (1 - 1/(8z) + 1/(128z^2) + O(1/z^3)),$$

and applied for $b_\chi(\nu)$ with $z = \nu/2$, we get

$$b_\chi(\nu) \sim 1 - (1/(4\nu) * (1 - 1/(8\nu)) + O(\nu^{-3})),$$

which has been implemented in b_chiAsymp(*,order=2) in 1999.

Even more accurately, Martin Maechler, used Maple to derive an asymptotic expansion up to order 15, here reported up to order 5, namely with $r := \frac{1}{4\nu}$,

$$b_\chi(\nu) = c_\chi(r) = 1 - r + \frac{1}{2}r^2 + \frac{5}{2}r^3 - \frac{21}{8}r^4 - \frac{399}{8}r^5 + O(r^6).$$

Value

a numeric vector of the same length as nu.

Author(s)

Martin Maechler

References

Johnson, Kotz, Balakrishnan (1995) *Continuous Univariate Distributions*, Vol 2, 2nd Edition; Wiley.
Formula on page 520, after (31.26a)

Abramowitz, M. and Stegun, I. A. (1972) *Handbook of Mathematical Functions*. New York: Dover. https://en.wikipedia.org/wiki/Abramowitz_and_Stegun provides links to the full text which is in public domain.

See Also

The t-distribution (base R) page pt; our pntJW39().

Examples

```
curve(b_chi, 0, 20); abline(h=0:1, v=0, lty=3)
r <- curve(b_chi, 1e-10, 1e5, log="x")
with(r, lines(x, b_chi(x, one.minus=TRUE), col = 2))

## Zoom in to c1-region
rc1 <- curve(b_chi, 340.5, 341.5, n=1001)# nothing to see
e <- 1e-3; curve(b_chi, 341-e, 341+e, n=1001) # nothing
e <- 1e-5; curve(b_chi, 341-e, 341+e, n=1001) # see noise, but no jump
e <- 1e-7; curve(b_chi, 341-e, 341+e, n=1001) # see float "granularity"+"jump"

## Zoom in to c2-region
rc2 <- curve(b_chi, 999.5, 1001.5, n=1001) # nothing visible
e <- 1e-3; curve(b_chi, 1000-e, 1000+e, n=1001) # clear small jump
c2 <- 1500
e <- 1e-3; curve(b_chi(x,c2=c2), c2-e, c2+e, n=1001)# still
## - - - -
c2 <- 3000
e <- 1e-3; curve(b_chi(x,c2=c2), c2-e, c2+e, n=1001)# ok asymp clearly better!!
curve(b_chiAsymp, add=TRUE, col=adjustcolor("red", 1/3), lwd=3)
if(requireNamespace("Rmpfr")) {
 xm <- Rmpfr::seqMpfr(c2-e, c2+e, length.out=1000)

}
## - - - -
c2 <- 4000
e <- 1e-3; curve(b_chi(x,c2=c2), c2-e, c2+e, n=1001)# ok asymp clearly better!!
curve(b_chiAsymp, add=TRUE, col=adjustcolor("red", 1/3), lwd=3)

grCol <- adjustcolor("forest green", 1/2)
curve(b_chi,                    1/2, 1e11, log="x")
curve(b_chiAsymp, add = TRUE, col = grCol, lwd = 3)
## 1-b(nu) ~= 1/(4 nu) a power function <==> linear in log-log scale:
curve(b_chi(x, one.minus=TRUE), 1/2, 1e11, log="xy")
curve(b_chiAsymp(x, one.minus=TRUE), add = TRUE, col = grCol, lwd = 3)
```

dbinom_raw *R's C (Mathlib) dbinom_raw() Binomial Probability pure R Function*

Description

A pure R implementation of R's C API ('Mathlib' specifically) `dbinom_raw()` function which computes binomial probabilities *and* is continuous in `x`, i.e., also "works" for non-integer `x`.

Usage

```
dbinom_raw (x, n, p, q = 1-p, log = FALSE, verbose = getOption("verbose"))
```

Arguments

`x`	vector with values typically in `0:n`, but here allowed to non-integer values.
`n`	called `size` in R's `dbinom()`.
`p`	called `prob` in R's `dbinom()`, the success probability, hence in $[0, 1]$.
`q`	mathemtically the same as $1 - p$, but may be (much) more accurate, notably when small.
`log`	logical indicating if the `log()` of the resulting probability should be returned; useful notably in case the probability itself would underflow to zero.
`verbose`	integer indicating the amount of verbosity of diagnostic output, `0` means no output, 1 more, etc.

Value

numeric vector of the same length as `x` (which may have to be thought of recycled along `n`, `p` and/or `q`.

Author(s)

R Core and Martin Maechler

See Also

Note that our CRAN package **Rmpfr** provides `dbinom`, an mpfr-accurate function to be used used instead of R's or this pure R version relying `bd0()` and `stirlerr()` where the latter currently only provides accurate double precision accuracy.

Examples

```
for(n in c(3, 10, 27, 100, 500, 2000, 5000, 1e4, 1e7, 1e10)) {
 x <- if(n <= 2000) 0:n else round(seq(0, n, length.out=2000))
 p <- 3/4
 stopifnot(all.equal(dbinom_raw(x, n, p, q=1-p),
                     dbinom    (x, n, p), tol = 1e-14))
}
n <- 1024 ; x <- 0:n
```

```
plot(x, dbinom_raw(x, n, p, q=1-p) - dbinom(x, n, p), type="l", main = "|db_r(x) - db(x)|")
plot(x, dbinom_raw(x, n, p, q=1-p) / dbinom(x, n, p) - 1, type="b", log="y",
     main = "rel.err.  |db_r(x / db(x) - 1)|")
```

---

dchisqApprox *Approximations of the (Noncentral) Chi-Squared Density*

Description

Compute the density function $f(x, *)$ of the (noncentral) chi-squared distribution.

Usage

```
dnchisqR     (x, df, ncp, log = FALSE,
              eps = 5e-15, termSml = 1e-10, ncpLarge = 1000)
dnchisqBessel(x, df, ncp, log = FALSE)
dchisqAsym   (x, df, ncp, log = FALSE)
dnoncentchisq(x, df, ncp, kmax = floor(ncp/2 + 5 * (ncp/2)^0.5))
```

Arguments

x	non-negative numeric vector.
df	degrees of freedom (parameter), a positive number.
ncp	non-centrality parameter δ; ....
log	logical indicating if the result is desired on the log scale.
eps	positive convergence tolerance for the series expansion: Terms are added while term * q > (1-q)*eps, where q is the term's multiplication factor.
termSml	positive tolerance: in the series expansion, terms are added to the sum as long as they are not smaller than termSml * sum even when convergence according to eps had occured. This was not part of the original C code, but was added later for safeguarding against infinite loops, from PR#14105, e.g., for dchisq(2000,2,1000).
ncpLarge	in the case where mid underflows to 0, when log is true, or ncp >= ncpLarge, use a central approximation. In theory, an optimal choice of ncpLarge would not be arbitrarily set at 1000 (hardwired in R's dchisq() here), but possibly also depend on x or df.
kmax	the number of terms in the sum for dnoncentchisq().

Details

dnchisqR() is a pure R implementation of R's own C implementation in the sources, 'R/src/nmath/dnchisq.c', additionally exposing the three "tuning parameters" eps, termSml, and ncpLarge.

dnchisqBessel() implements Fisher(1928)'s exact closed form formula based on the Bessel function I_{nu}, i.e., R's besselI() function; specifically formula (29.4) in Johnson et al. (1995).

dchisqAsym() is the simple asymptotic approximation from Abramowitz and Stegun's formula 26.4.27, p. 942.

dnoncentchisq() uses the (typically defining) infinite series expansion directly, with truncation at kmax, and terms t_k which are products of a Poisson probability and a central chi-square density, i.e., terms t.k := dpois(k,lambda = ncp/2) * dchisq(x,df = 2*k + df) for k = 0,1,...,kmax.

Value

numeric vector similar to x, containing the (logged if log=TRUE) values of the density $f(x, *)$.

Note

These functions are mostly of historical interest, notably as R's dchisq() was not always very accurate in the noncentral case, i.e., for ncp > 0.

Note

R's dchisq() is typically more uniformly accurate than the approximations nowadays, apart from dnchisqR() which should behave the same. There may occasionally exist small differences between dnchisqR(x,*) and dchisq(x,*) for the same parameters.

Author(s)

Martin Maechler, April 2008

References

Abramowitz, M. and Stegun, I. A. (1972) *Handbook of Mathematical Functions*. New York: Dover. https://en.wikipedia.org/wiki/Abramowitz_and_Stegun provides links to the full text which is in public domain.

Johnson, N.L., Kotz, S. and Balakrishnan, N. (1995) Continuous Univariate Distributions Vol~2, 2nd ed.; Wiley.
Chapter 29, Section *3 Distribution*, (29.4), p. 436.

See Also

R's own dchisq().

Examples

```
x <- sort(outer(c(1,2,5), 2^(-4:5)))
fRR <- dchisq  (x, 10, 2)
f.R <- dnchisqR(x, 10, 2)
all.equal(fRR, f.R, tol = 0) # 64bit Lnx (F 30): 1.723897e-16
stopifnot(all.equal(fRR, f.R, tol = 4e-15))
```

---

dgamma-utils *Utility Functions for* dgamma() *– Pure R Versions*

Description

Mostly, pure R transcriptions of the C code utility functions for dgamma() and similar "base" density functions by Catherine Loader.

bd0C() interfaces to C code which corresponds to R's C Mathlib bd0().

These have extra arguments with defaults that correspond to the C Mathlib code hardwired cutoffs and tolerances.

Usage

```
dpois_raw(x, lambda, log=FALSE,
          version,
          ## the defaults for version will probably change in the future
          bd0.delta = 0.1,
          ## optional arguments of log1pmx() :
        tol_logcf = 1e-14, eps2 = 0.01, minL1 = -0.79149064, trace.lcf = verbose,
          logCF = if (is.numeric(x)) logcf else logcfR,
          verbose = FALSE)
bd0(x, np,
    delta = 0.1, maxit = 1000L,
    s0 = .Machine$double.xmin,
    verbose = getOption("verbose"))
bd0C(x, np, delta = 0.1, maxit = 1000L, version = "R4.0", verbose = getOption("verbose"))
ebd0(x, M, verbose = getOption("verbose"))

stirlerr(n, scheme = c("R3", "R4.1"),
         cutoffs = switch(scheme
                        , R3   = c(15, 35, 80, 500)
                       , R4.1 = c(7.5, 8.5, 10.625, 12.125, 20, 26, 55, 200, 3300)
                          ),
         use.halves = missing(cutoffs),
         verbose = FALSE)
lgammacor(x, nalgm = 5, xbig = 2^26.5)
```

Arguments

`x, n`	`numeric` (or number-alike such as "mpfr").
`lambda, np, M`	each `numeric`; distrubution parameters.
`log`	logical indicating if the log-density should be returned, otherwise the density at `x`.
`verbose`	logical indicating if some information about the computations are to be printed.
`delta, bd0.delta`	a positive number, a cutoff for `bd0()` where the `logcf()` series expansion is used when $\lvert x - M\rvert < delta * (x + M)$.
`tol_logcf, eps2, minL1, trace.lcf, logCF`	optional tuning arguments passed to `log1pmx()`.
`maxit`	the number of `logcf()` terms to be used in `bd0()` when $\lvert x - M\rvert$ is small.
`s0`	the very small s_0 determining that `bd0() = s` already before the locf series expansion.
`version`	a `character` string specifying the version of $bd0()$ used.
`scheme`	a `character` string specifying the `cutoffs` scheme.
`cutoffs`	an increasing numeric vector, required to start with with `cutoffs[1] <= 15` specifying the cutoffs to switch from 2 to 3 to ..., up to 10 term approximations for non-small `n`, where the direct formula loses precision. When missing (as by default), `scheme` is used, where `scheme = "R3"` chooses (15, 35, 80, 500), the cutoffs in use in R versions up to (and including) 4.0.z.

use.halves — logical indicating if the full-accuracy prestored values should be use when $2n \in \{0, 1, \ldots, 30\}$, i.e., $n \le 15$ and n is integer or integer + $\frac{1}{2}$. Turn this off to judge the underlying approximation accuracy by comparison with MPFR. However, keep the default TRUE for back-compatibility.

nalgm — number of terms to use for Chebyshev polynomial approxmation in lgammacor(). The default, 5, is the value hard wired in R's C Mathlib.

xbig — a large positive number; if x >= xbig, the simple asymptotic approximation lgammacor(x) := 1/(12*x) is used. The default, $2^{26.5} = 94906265.6$, is the value hard wired in R's C Mathlib.

Details

bd0(): Loader's "Binomial Deviance" function; for $x, M > 0$ (where the limit $x \to 0$ is allowed). In the case of dbinom, x are integers (and $M = np$), but in general x is real.

$$bd_0(x, M) := M \cdot D_0\big(\frac{x}{M}\big),$$

where $D_0(u) := u\log(u) + 1 - u = u(\log(u) - 1) + 1$. Hence

$$bd_0(x, M) = M \cdot \big(\frac{x}{M}(\log(\frac{x}{M}) - 1) + 1\big) = x\log(\frac{x}{M}) - x + M.$$

A different way to rewrite this from Martyn Plummer, notably for important situation when $|x - M| \ll M$, is using $t := (x - M)/M$ (and $|t| \ll 1$ for that situation), equivalently, $\frac{x}{M} = 1 + t$. Using t,

$$bd_0(x, M) = \log(1+t) - t\cdot M = M\cdot[(t+1)(\log(1+t)-1)+1] = M\cdot[(t+1)\log(1+t) - t] = M\cdot p_1 l_1(t),$$

and

$$p_1 l_1(t) := (t+1)\log(1+t) - t = \frac{t^2}{2} - \frac{t^3}{6} \ldots$$

where the Taylor series expansion is useful for small $|t|$.

Value

a numeric vector "like" x; in some cases may also be an (high accuracy) "mpfr"-number vector, using CRAN package **Rmpfr**.

lgammacor(x) originally returned NaN for all $|x| < 10$, as its Chebyshev polynomial approximation has been constructed for $x \in [10, xbig]$, specifically for $u \in [-1, 1]$ where $t := 10/x \in [1/x_B, 1]$ and $u := 2t^2 - 1 \in [-1 + \epsilon_B, 1]$.

Author(s)

Martin Maechler

References

C. Loader (2000), see dbinom's documentation.

See Also

dgamma, dpois.

Examples

```
n <- seq(1, 50, by=1/4)
st.n <- stirlerr(n) # now vectorized
stopifnot(identical(st.n, sapply(n, stirlerr)))
plot(n, st.n, type = "b", log="xy", ylab = "stirlerr(n)")

x <- 800:1200
bd0x1k <- bd0(x, np = 1000)
plot(x, bd0x1k, type="l", ylab = "bd0(x, np=1000)")
bd0x1kC <- bd0C(x, np = 1000)
lines(x, bd0x1kC, col=2)
stopifnot(all.equal(bd0x1kC, bd0x1k, tol=1e-15)) # even tol=0 currently ..
if(FALSE) ## FIXME !
ebd0x1k <- ebd0(x, 1000) # FIXME: subscript out of bout
if(FALSE) # the bug is  only here:
    ebd0(1001, 1000)
## so this works:
ex. <- ebd0(x[x != 1001], 1000)
## but there is more wrong currently:
lines(x[x!=1001], colSums(ex.), col=3)

x <- 1:80
dp   <- dpois    (x, 48, log=TRUE)# R's 'stats' pkg function
dp.r <- dpois_raw(x, 48, log=TRUE)
all.equal(dp, dp.r, tol = 0) # on Linux 64b, see TRUE
stopifnot(all.equal(dp, dp.r, tol = 1e-14))
# matplot(x, cbind(dp, dp.r), type = "b") # looks "ok", .. but not if you look closely:
# plot(x, dp.r - dp, type = "b",
#      main = "dpois_raw(x, 48, log=T)  -  dpois(..)  -- something's wrong!")
# abline(h=0, lty=3)
```

---

dgamma.R *Gamma Density Function Alternatives*

Description

dgamma.R() is aimed to be an R level "clone" of R's C level implementation dgamma (from package **stats**).

Usage

```
dgamma.R(x, shape, scale = 1, log)
```

Arguments

x	non-negative numeric vector.
shape	non-negative shape parameter of the Gamma distribution.
scale	positive scale parameter; note we do not see the need to have a rate parameter as the standard R function.
log	logical indicating if the result is desired on the log scale.

Value

numeric vector of the same length as `x` (which may have to be thought of recycled along `shape` and/or `scale`.

Author(s)

Martin Maechler

See Also

(As R's C code) this depends crucially on the "workhorse" function `dpois_raw()`.

Examples

```
## TODO: ... regular case .. use all.equal() ...


## From R's  <R>/tests/d-p-q-r-tst-2.R -- replacing dgamma() w/ dgamma.R()
## PR#17577 - dgamma(x, shape)  for shape < 1 (=> +Inf at x=0) and very small x
stopifnot(exprs = {
    all.equal(dgamma.R(2^-1027, shape = .99 , log=TRUE), 7.1127667376, tol=1e-10)
    all.equal(dgamma.R(2^-1031, shape = 1e-2, log=TRUE), 702.8889158,  tol=1e-10)
    all.equal(dgamma.R(2^-1048, shape = 1e-7, log=TRUE), 710.30007699, tol=1e-10)
    all.equal(dgamma.R(2^-1048, shape = 1e-7, scale = 1e-315, log=TRUE),
              709.96858768, tol=1e-10)
})
## R's dgamma() gave all Inf in R <= 3.6.1 [and still there in 32-bit Windows !]
```

---

dhyperBinMolenaar *HyperGeometric (Point) Probabilities via Molenaar's Binomial Approximation*

Description

Compute hypergeometric (point) probabilities via Molenaar's binomial approximation, `hyper2binomP()`.

Usage

```
dhyperBinMolenaar(x, m, n, k, log = FALSE)
```

Arguments

`x`	..
`m`	..
`n`	..
`k`	..
`log`	`logical` indication if the logarithm `log(P)` should be returned (instead of P).

Details

...

Value

. . .

Author(s)

Martin Maechler

References

. . .

See Also

R's own `dhyper()` which uses more sophisticated computations.

Examples

```
## The function is simply defined as
function (x, m, n, k, log = FALSE)
  dbinom(x, size = k, prob = hyper2binomP(x, m, n, k), log = log)
```

`dnbinomR` *Pure R Versions of R's C (Mathlib) dnbinom() Negative Binomial Probabilities*

Description

Computer pure R implementations of R's C (Mathlib) `dnbinom()` binomial probabilities, allowing to see the effect of the cutoff `eps`.

Usage

```
dnbinomR  (x, size, prob, log = FALSE, eps = 1e-10)
dnbinom.mu(x, size, mu,   log = FALSE, eps = 1e-10)
```

Arguments

`x, size, prob, mu, log`	see R's `dnbinom()`.
`eps`	non-negative number specifying the cutoff for "small `x/size`", in which case the 2-term approximation from Abramowitz and Stegun, 6.1.47 (p.257) is preferable to the `dbinom()` based evaluation.

Value

numeric vector of the same length as `x` (which may have to be thought of recycled along `size`, `prob` and/or `mu`.

Author(s)

R Core and Martin Maechler

References

Abramowitz, M. and Stegun, I. A. (1972) *Handbook of Mathematical Functions*. New York: Dover. https://en.wikipedia.org/wiki/Abramowitz_and_Stegun provides links to the full text which is in public domain.

See Also

dbinom_raw; Note that our CRAN package **Rmpfr** provides dnbinom, dbinom and more, where mpfr-accurate functions are used instead of R's (and our pure R version of) bd0() and stirlerr().

Examples

```
stopifnot( dnbinomR(0, 1, 1) == 1 )
size <- 1000 ; x <- 0:size
dnb <- dnbinomR(x, size, prob = 5/8, log = FALSE, eps = 1e-10)
plot(x, dnb, type="b")
all.equal(dnb, dnbinom(x, size, prob = 5/8)) ## mean rel. diff: 0.00017...

dnbm <- dnbinom.mu(x, size, mu = 123, eps = 1e-10)
all.equal(dnbm, dnbinom(x, size, mu = 123)) #  Mean relative diff: 0.00069...
```

---

dnt *Non-central t-Distribution Density - Algorithms and Approximations*

Description

dntJKBf1 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 $1/j!$.

dntJKBf() is Vectorize(dntJKBf1,c("x","df","ncp")), i.e., works vectorized in all three main arguments x, df and ncp.

The functions .dntJKBch1() and .dntJKBch() are only there for didactical reasons allowing to check that indeed formula (31.15') in the reference is missing a $j!$ factor in the denominator.

The dntJKBf*() functions are written to also work with arbitrary precise numbers of class "mpfr" (from package **Rmpfr**) as arguments.

Usage

```
dntJKBf1(x, df, ncp, log = FALSE, M = 1000)
dntJKBf (x, df, ncp, log = FALSE, M = 1000)

## The "checking" versions, only for proving correctness of formula:
.dntJKBch1(x, df, ncp, log = FALSE, M = 1000, check=FALSE, tol.check = 1e-7)
.dntJKBch (x, df, ncp, log = FALSE, M = 1000, check=FALSE, tol.check = 1e-7)
```

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.
check	logical indicating if checks of the formula equalities should be done.
tol.check	tolerance to be used for all.equal() when check is true.

Details

How to choose M optimally has not been investigated yet.

Note that relatedly,

R's source code 'R/src/nmath/dnt.c' has claimed from 2003 till 2014 but **wrongly** that the non-central t density $f(x,*)$ is

```
   f(x, df, ncp) =
df^(df/2) * exp(-.5*ncp^2) /
(sqrt(pi)*gamma(df/2)*(df+x^2)^((df+1)/2)) *
 sum_{k=0}^Inf  gamma((df + k + df)/2)*ncp^k / prod(1:k)*(2*x^2/(df+x^2))^(k/2) .
```

These functions (and this help page) prove that it was wrong.

Value

a number for dntJKBf1() and .dntJKBch1().

a numeric vector of the same length as the maximum of the lengths of x,df,ncp for dntJKBf() and .dntJKBch().

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

R's dt; (an improved version of) Viechtbauer's proposal: dtWV.

Examples

```
tt <- seq(0, 10, len = 21)
ncp <- seq(0, 6, len = 31)
dt3R   <- outer(tt, ncp, dt,     df = 3)
dt3JKB <- outer(tt, ncp, dntJKBf, df = 3)
all.equal(dt3R, dt3JKB) # Lnx(64-b): 51 NA's in dt3R

x <- seq(-1,12, by=1/16)
fx <- dt(x, df=3, ncp=5)
```

```
re1 <- 1 - .dntJKBch(x, df=3, ncp=5) / fx ; summary(warnings()) # slow, with warnings
op <- options(warn = 2) # (=> warning == error, for now)
re2 <- 1 -  dntJKBf (x, df=3, ncp=5) / fx  # faster, no warnings
stopifnot(all.equal(re1[!is.na(re1)], re2[!is.na(re1)], tol=1e-6))
head( cbind(x, fx, re1, re2) , 20)
matplot(x, log10(abs(cbind(re1, re2))), type = "o", cex = 1/4)

## One of the numerical problems in "base R"'s non-central t-density:
options(warn = 0) # (factory def.)
x <- 2^seq(-12, 32, by=1/8) ; df <- 1/10
dtm <- cbind(dt(x, df=df,           log=TRUE),
             dt(x, df=df, ncp=df/2, log=TRUE),
             dt(x, df=df, ncp=df,   log=TRUE),
             dt(x, df=df, ncp=df*2, log=TRUE)) #.. quite a few warnings:
summary(warnings())
matplot(x, dtm, type="l", log = "x", xaxt="n",
        main = "dt(x, df=1/10, log=TRUE) central and noncentral")
sfsmisc::eaxis(1)
legend("right", legend=c("", paste0("ncp = df",c("/2","","*2"))),
       lty=1:4, col=1:4, bty="n")
 # ---- using MPFR high accuracy arithmetic (too slow for routine testing) ---
## no such kink here:
x. <- if(requireNamespace("Rmpfr")) Rmpfr::mpfr(x, 256) else x
system.time(dtJKB <- dntJKBf(x., df=df, ncp=df, log=TRUE)) # 21s (!) was only 7s ???
lines(x, dtJKB, col=adjustcolor(3, 1/2), lwd=3)
options(op) # reset to prev.

## Relative Difference / Approximation errors :
plot(x, 1 - dtJKB / dtm[,3], type="l", log="x")
plot(x, 1 - dtJKB / dtm[,3], type="l", log="x", xaxt="n", ylim=c(-1,1)*1e-3); sfsmisc::eaxis(1)
plot(x, 1 - dtJKB / dtm[,3], type="l", log="x", xaxt="n", ylim=c(-1,1)*1e-7); sfsmisc::eaxis(1)
plot(x, abs(1 - dtJKB / dtm[,3]), type="l", log="xy", axes=FALSE, main =
     "dt(*, 1/10, 1/10, log=TRUE) relative approx. error",
     sub= paste("Copyright © 2019  Martin Mächler  --- ", R.version.string))
for(j in 1:2) sfsmisc::eaxis(j)
```

---

dtWV *Noncentral t Distribution Density by W.V.*

Description

Compute the density function $f(x)$ of the t distribution with df degrees of freedom and non-centrality parameter ncp, according to Wolfgang Viechtbauer's proposal in 2002.

Usage

```
dtWV(x, df, ncp = 0, log = FALSE)
```

Arguments

x	numeric vector.
df	degrees of freedom (> 0, maybe non-integer). df = Inf is allowed.

ncp	non-centrality parameter δ; If omitted, use the central t distribution.
log	logical; if TRUE, $log(f(x))$ is returned instead of $f(x)$.

Details

The formula used is "asymptotic": Resnikoff and Lieberman (1957), p.1 and p.25ff, proposed to use recursive polynomials for (*integer !*) degrees of freedom $f = 1, 2, \ldots, 20$, and then, for df$=$ $f > 20$, use the asymptotic approximation which Wolfgang Viechtbauer proposed as a first version of a non-central t density for R (when `dt()` did not yet have an ncp argument).

Value

numeric vector of density values, properly recycled in `(x,df,ncp)`.

Author(s)

Wolfgang Viechtbauer (2002) post to R-help (https://stat.ethz.ch/pipermail/r-help/2002-October/026044.html), and Martin Maechler (log argument; tweaks, notably recycling).

References

Resnikoff, George J. and Lieberman, Gerald J. (1957) *Tables of the non-central t-distribution*; Technical report no. 32 (LIE ONR 32), April 1, 1957; Applied Math. and Stat. Lab., Stanford University. https://statistics.stanford.edu/research/tables-non-central-t-distribution-density-function-cum

See Also

`dt`, R's (C level) implementation of the (non-central) t density; `dntJKBf`, for Johnson et al.'s summation formula approximation.

Examples

```
tt <- seq(0, 10, len = 21)
ncp <- seq(0, 6, len = 31)
dt3R  <- outer(tt, ncp, dt  , df = 3)
dt3WV <- outer(tt, ncp, dtWV, df = 3)
all.equal(dt3R, dt3WV) # rel.err 0.00063
dt25R  <- outer(tt, ncp, dt  , df = 25)
dt25WV <- outer(tt, ncp, dtWV, df = 25)
all.equal(dt25R, dt25WV) # rel.err 1.1e-5

x <- -10:700
fx  <- dt  (x, df = 22, ncp =100)
lfx <- dt  (x, df = 22, ncp =100, log=TRUE)
lfV <- dtWV(x, df = 22, ncp =100, log=TRUE)

head(lfx, 20) # shows that R's dt(*, log=TRUE) implementation is "quite suboptimal"

## graphics
opa <- par(no.readonly=TRUE)
par(mar=.1+c(5,4,4,3), mgp = c(2, .8,0))
plot(fx ~ x, type="l")
par(new=TRUE) ; cc <- c("red", adjustcolor("orange", 0.4))
plot(lfx ~ x, type = "o", pch=".", col=cc[1], cex=2, ann=FALSE, yaxt="n")
sfsmisc::eaxis(4, col=cc[1], col.axis=cc[1], small.args = list(col=cc[1]))
```

```
lines(x, lfV, col=cc[2], lwd=3)
dtt1 <- "      dt"; dtt2 <- "(x, df=22, ncp=100"; dttL <- paste0(dtt2,", log=TRUE)")
legend("right", c(paste0(dtt1,dtt2,")"), paste0(c(dtt1,"dtWV"), dttL)),
       lty=1, lwd=c(1,1,3), col=c("black", cc), bty = "n")
par(opa) # reset
```

`format01prec` *Format Numbers in [0,1] with "Precise" Result*

Description

Format numbers in [0,1] with "precise" result, notably using `"1-.."` if needed.

Usage

```
format01prec(x, digits = getOption("digits"), width = digits + 2,
             eps = 1e-06, ...,
             FUN = function(x, ...) formatC(x, flag = "-", ...))
```

Arguments

`x`	numbers in [0,1]; (still works if not)
`digits`	number of digits to use; is used as `FUN(*,digits = digits)` or `FUN(*,digits = digits -5)` depending on `x` or `eps`.
`width`	desired width (of strings in characters), is used as `FUN(*,width = width)` or `FUN(*,width = width -2)` depending on `x` or `eps`.
`eps`	small positive number: Use `'1-'` for those `x` which are in $(1 - eps, 1]$. The author has claimed in the last millennium that (the default) 1e-6 is *optimal*.
`...`	optional further arguments passed to `FUN(x,digits,width,...)`.
`FUN`	a `function` used for `format()`ing; must accept both a `digits` and `width` argument.

Value

a `character` vector of the same length as `x`.

Author(s)

Martin Maechler, 14 May 1997

See Also

`formatC`, `format.pval`.

Examples

```
## Show that format01prec()  does reveal more precision :
cbind(format      (1 - 2^-(16:24)),
      format01prec(1 - 2^-(16:24)))

## a bit more variety
e <- c(2^seq(-24,0, by=2), 10^-(7:1))
ee <- sort(unique(c(e, 1-e)))
noquote(ff <- format01prec(ee))
data.frame(ee, format01prec = ff)
```

`hyper2binomP` *Transform Hypergeometric Distribution Parameters to Binomial Probability*

Description

Transform the three parameters of the hypergeometric distribution function to the probability parameter of the corresponding binomial distribution.

Usage

```
hyper2binomP(x, m, n, k)
```

Arguments

`x`	..
`m`	..
`n`	..
`k`	..

Details

...

Value

a number, the binomial probability.

See Also

`phyper`, `pbinom`.

`dhyperBinMolenaar()` which is based on `hyper2binomP()`.

Examples

```
hyper2binomP(3,4,5,6) # 0.38856

## The function is simply defined as
function (x, m, n, k)
{
    N <- m + n
    p <- m/N
    N.n <- N - (k - 1)/2
    (m - x/2)/N.n - k * (x - k * p - 1/2)/(6 * N.n^2)
  }
```

`lbeta` *(Log) Beta Approximations*

Description

Compute `log(beta(a,b))` in a simple (fast) or asymptotic way.

Usage

```
lbetaM   (a, b, k.max = 5, give.all = FALSE)
lbeta_asy(a, b, k.max = 5, give.all = FALSE)
lbetaMM  (a, b, cutAsy = 1e-2, verbose = FALSE)

 betaI(a, n)
lbetaI(a, n)

logQab_asy(a, b, k.max = 5, give.all = FALSE)
Qab_terms(a, k)
```

Arguments

`a, b, n`	the Beta parameters, see `beta`; `n` must be a positive integer and "small".
`k.max`	..
`give.all`	`logical` ..
`cutAsy`	cutoff value from where to switch to asymptotic formula.
`verbose`	logical (or integer) indicating if and how much monitoring information should be printed to the console.
`k`	the number of terms in the series expansion of `Qab_terms()`, currently must be in $\{0, 1, .., 5\}$.

Details

All `lbeta*()` functions compute `log(beta(a,b))`.

We use $Qab = Qab(a, b)$ for

$$Q_{a,b} := \frac{\Gamma(a+b)}{\Gamma(b)},$$

which is numerically challenging when b becomes large compared to a, or $a \ll b$.

With the beta function

$$B(a,b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)} = \frac{\Gamma(a)}{Qab},$$

and hence

$$\log B(a,b) = \log\Gamma(a) + \log\Gamma(b) - \log\Gamma(a+b) = \log\Gamma(a) - \log Qab,$$

or in R, `lBeta(a,b) := lgamma(a) -logQab(a,b)`.

Indeed, typically everything has to be computed in log scale, as both $\Gamma(b)$ and $\Gamma(a+b)$ would overflow numerically for large b. Consequently, we use `logQab*()`, and for the large b case `logQab_asy()` specifically,

$$\texttt{logQab(a,b)} := \log(Qab(a,b)).$$

Note this is related to trying to get asymptotic formula for Γ ratios, notably formula (6.1.47) in Abramowitz and Stegun.

Note how this is related to computing `qbeta()` in boundary cases, and see `algdiv()` 'Details' about this.

We also have a vignette about this, but really the problem has been adressed pragmatically by the authors of TOMS 708, see the 'References' in `pbeta`, by their routine `algdiv()` which also is available in our package **DPQ**.

Value

a fast or simple (approximate) computation of `lbeta(a,b)`.

Author(s)

Martin Maechler

References

Abramowitz, M. and Stegun, I. A. (1972) *Handbook of Mathematical Functions*. New York: Dover. https://en.wikipedia.org/wiki/Abramowitz_and_Stegun provides links to the full text which is in public domain.

Formula (6.1.47), p.257

See Also

R's `beta` function; `algdiv()`.

Examples

```
## TODO
```

`lfastchoose` *R versions of Simple Formulas for Logarithmic Binomial Coefficients*

Description

Provide R versions of simple formulas for computing the logarithm of (the absolute value of) binomial coefficients, i.e., simpler, more direct formulas than what (the C level) code of R's `lchoose()` computes.

Usage

```
lfastchoose(n, k)
 f05lchoose(n, k)
```

Arguments

n	a numeric vector.
k	a integer valued numeric vector.

Value

a numeric vector with the same attributes as n + k.

Author(s)

Martin Maechler

See Also

`lchoose`.

Examples

```
lfastchoose # function(n, k) lgamma(n + 1) - lgamma(k + 1) - lgamma(n - k + 1)
f05lchoose  # function(n, k) lfastchoose(n = floor(n + 0.5), k = floor(k + 0.5))

## interesting cases ?
```

`lgamma1p` *Accurate* `log(gamma(a+1))`

Description

Compute

$$l\Gamma_1(a) := \log\Gamma(a+1) = \log(a \cdot \Gamma(a)) = \log a + \log\Gamma(a),$$

which is "in principle" the same as `log(gamma(a+1))` or `lgamma(a+1)`, accurately also for (very) small a $(0 < a < 0.5)$.

Usage

```
lgamma1p (a, tol_logcf = 1e-14, f.tol = 1, ...)
lgamma1p.(a, cutoff.a = 1e-6, k = 3)
lgamma1p_series(x, k)
lgamma1pC(x)
```

Arguments

a, x	a numeric vector.
tol_logcf	for lgamma1p(): a non-negative number passed to logcf() (and log1pmx() which calls logcf()).
f.tol	numeric (**f**actor) used in log1pmx(*,tol_logcf = f.tol * tol_logcf).
...	further optional arguments passed on to log1pmx().
cutoff.a	for lgamma1p.(): a positive number indicating the cutoff to switch from ...
k	an integer, the number of terms in the series expansion used internally.

Details

lgamma1p() is an R translation of the function (in Fortran) in Didonato and Morris (1992) which uses a 40-degree polynomial approximation.

lgamma1p_series(x,k) is Taylor series approximation of order k, (derived via Maple), which is $-\gamma x + \pi^2 x^2/12 + O(x^3)$, where γ is Euler's constant 0.5772156649. ...

lgamma1pC() is an interface to R C API ('Rmathlib') function.

Value

a numeric vector with the same attributes as a.

Author(s)

Morten Welinder (C code of Jan 2005, see R's bug issue PR#7307) for lgamma1p().

Martin Maechler, notably for lgamma1p_series() which works with package **Rmpfr** but otherwise may be *much* less accurate than Morten's 40 term series!

References

Didonato, A. and Morris, A., Jr, (1992) Algorithm 708: Significant digit computation of the incomplete beta function ratios. *ACM Transactions on Mathematical Software*, **18**, 360–373; see also pbeta.

See Also

log1pmx, log1p, pbeta.

Examples

```
curve(-log(x*gamma(x)), 1e-30, .8, log="xy", col="gray50", lwd = 3,
      axes = FALSE, ylim = c(1e-30,1))
sfsmisc::eaxis(1); sfsmisc::eaxis(2)
at <- 10^(1-4*(0:8))
abline(h = at, v = at, col = "lightgray", lty = "dotted")
```

```
curve(-lgamma( 1+x), add=TRUE, col="red2", lwd=1/2)# underflows even earlier
curve(-lgamma1p (x), add=TRUE, col="blue") -> lgxy
curve(-lgamma1p.(x), add=TRUE, col=adjustcolor("forest green",1/4),
      lwd = 5, lty = 2)
for(k in 1:7)
  curve(-lgamma1p_series(x, k=k), add=TRUE, col=paste0("gray",30+k*8), lty = 3)
stopifnot(with(lgxy, all.equal(y, -lgamma1pC(x))))
```

`lgammaAsymp` *Asymptotic Log Gamma Function*

Description

Compute an n-th order asymptotic approximation to log Gamma function, using Bernoulli numbers `Bern(k)` for k in $1, \ldots, 2n$.

Usage

```
lgammaAsymp(x, n)
```

Arguments

x	numeric vector
n	integer specifying the approximation order.

Value

numeric vector with the same attributes (`length()` etc) as `x`, containing approximate `lgamma(x)` values.

Author(s)

Martin Maechler

See Also

`lgamma`.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (x, n)
{
    s <- (x - 1/2) * log(x) - x + log(2 * pi)/2
    if (n >= 1) {
        Ix2 <- 1/(x * x)
        k <- 1:n
        Bern(2 * n)
```

```
        Bf <- rev(.bernoulliEnv$.Bern[k]/(2 * k * (2 * k - 1)))
        bsum <- Bf[1]
        for (i in k[-1]) bsum <- Bf[i] + bsum * Ix2
        s + bsum/x
    }
    else s
}
```

`log1mexp` *Compute* log*(1 -* exp*(-a)) and* log(1 + exp(x)) *Numerically Optimally*

Description

Compute f(a) = log(1 - exp(-a)) quickly and numerically accurately.

`log1mexp()` is simple pure R code;
`log1mexpC()` is an interface to R C API ('`Rmathlib`') function.

`log1pexpC()` is an interface to R C API ('`Rmathlib`') `double` function `log1pexp()` which computes $\log(1 + \exp(x))$, accurately, notably for large x, say, $x > 720$.

Usage

```
log1mexp (x)
log1mexpC(x)
log1pexpC(x)
```

Arguments

x	numeric vector of positive values.

Author(s)

Martin Maechler

References

Martin Mächler (2012). Accurately Computing $\log(1 - \exp(-|a|))$; https://CRAN.R-project.org/package=Rmpfr/vignettes/log1mexp-note.pdf.

See Also

The `log1mexp()` function in CRAN package **copula**, and the corresponding vignette (in the 'References').

Examples

```
l1m.xy <- curve(log1mexp(x), -10, 10, n=1001)
stopifnot(with(l1m.xy, all.equal(y, log1mexpC(x))))
```

`log1pmx` *Accurate* `log(1+x) - x`

Description

Compute

$$\log(1+x) - x$$

accurately also for small x, i.e., $|x| \ll 1$.

Since April 2021, the pure R code version `log1pmx()` also works for "mpfr" numbers (from package **Rmpfr**).

Usage

```
log1pmx (x, tol_logcf = 1e-14, eps2 = 0.01, minL1 = -0.79149064,
         trace.lcf = FALSE,
         logCF = if(is.numeric(x)) logcf else logcfR)
log1pmxC(x)
```

Arguments

`x`	numeric vector with values $x > -1$.
`tol_logcf`	a non-negative number indicating the tolerance (maximal relative error) for the auxiliary `logcf()` function.
`eps2`	positive cutoff where the algorithm switches from a few terms, to using `logcf()` explicitly.
`minL1`	negative cutoff, called `minLog1Value` in Morten Welinder's C code for `log1pmx()` in 'R/src/nmath/pgamma.c'. It seems to Martin M that it is not quite optimal, at least in some cases.
`trace.lcf`	`logical` used in `logcf(..,trace=trace.lcf)`.
`logCF`	the `function` to be used as `logcf()`. The default chooses the pure R `logcfR()` when `x` is not numeric, and chooses the C-based `logcf()` when `is.numeric(x)` is true.

Details

In order to provide full accuracy, the computations happens differently in three regions for x,

$$m_l = \texttt{minL1} = -0.79149064$$

is the first cutpoint,

$x < m_l$ **or** $x > 1$**:** use `log1pmx(x) := log1p(x) -x`,
$|x| < \epsilon_2$**:** use $t((((2/9 * y + 2/7)y + 2/5)y + 2/3)y - x)$,
$x \in [ml, 1]$**, and** $|x| \geq \epsilon_2$**:** use $t(2ylogcf(y, 3, 2) - x)$,

where $t := \frac{x}{2+x}$, and $y := t^2$.

Note that the formulas based on t are based on the (fast converging) formula

$$\log(1+x) = 2\left(r + \frac{r^3}{3} + \frac{r^5}{5} + \dots\right),$$

where $r := x/(x+2)$, see the reference.

`log1pmxC()` is an interface to R C API ('Rmathlib') function.

Value

a numeric vector (with the same attributes as x).

Author(s)

A translation of Morten Welinder's C code of Jan 2005, see R's bug issue PR#7307.

References

Abramowitz, M. and Stegun, I. A. (1972) *Handbook of Mathematical Functions*. New York: Dover.
https://en.wikipedia.org/wiki/Abramowitz_and_Stegun provides links to the full text which is in public domain.
Formula (4.1.29), p.68.

See Also

logcf, the auxiliary function, lgamma1p which calls log1pmx, log1p

Examples

```
(doExtras <- DPQ:::doExtras()) # TRUE e.g. if interactive()
n1 <- if(doExtras) 1001 else 201
l1x <- curve(log1pmx, -.9999, 7, n=n1)
abline(h=0, v=-1:0, lty=3)
l1xz  <- curve(log1pmx, -.1,  .1,  n=n1); abline(h=0, v=0, lty=3)
l1xz2 <- curve(log1pmx, -.01, .01, n=n1); abline(h=0, v=0, lty=3)
l1xz3 <- curve(-log1pmx(x), -.002, .002, n=n1, log="y", yaxt="n")
sfsmisc::eaxis(2); abline(v=0, lty=3)
with(l1xz3, stopifnot(all.equal(y, -log1pmxC(x))))

e <- if(doExtras) 2^-12 else 2^-8; by.p <- 1/(if(doExtras) 256 else 64)
xd <- c(seq(-1+e, 0+100*e, by=e), seq(by.p, 5, by=by.p))
plot(xd, log1pmx(xd), type="l", col=2, main = "log1pmx(x)")
abline(h=0, v=-1:0, lty=3)

if(requireNamespace("Rmpfr") && packageVersion("sfsmisc") >= "1.1-10") withAutoprint({
  xM <- Rmpfr::mpfr(xd, 512)
  ## for MPFR numbers, really need more than tol_logcf = eps = 1e-14 (default)
  if(doExtras) print( system.time(
    lg1pM <- log1pmx(xM, tol_logcf = 1e-25, eps2 = 1e-4)
  )) # 4.5 sec if(doExtras) 0.43s otherwise, but 20s (!) on winbuilder!
  ## MM: But really, this should be more accurate anyway (?!):
  lg1pM. <- log1p(xM) - xM
  xM2k <- Rmpfr::mpfr(xd, 2048)
  lg1pM2k <- log1p(xM2k) - xM2k # even more accurate
  relErrV <- sfsmisc::relErrV
  asNumeric <- Rmpfr::asNumeric
  reE00 <- asNumeric( relErrV(lg1pM2k, lg1pM.) )
  print(signif(range(reE00)), 2) # [-1.5e-151, 4.8e-151] -- 512 bits is perfect
 if(doExtras) { ## the error of the log1pmx() "algorithm" even for "perfect" mpfr-accuracy:
    rE.log1pm <- asNumeric(relErrV(lg1pM2k, lg1pM))
    print(signif(range(  rE.log1pm ), 2)) # -1.2e-27 3.7e-49
  }
  re <- asNumeric(relErrV(lg1pM., log1pmx(xd)))
  plot(xd, re, type="b", cex=1/2)
  abline(h = (-2:2)*2^-52, lty=2, col=adjustcolor("gray20", 1/2))
```

```
 ## only negative x:
 iN <- xd < 0
 iN <- -0.84 < xd & xd < -0.4
 plot(xd[iN], re[iN], type="b", cex=1/2)
 abline(h = (-2:2)*2^-52, lty=2, col=adjustcolor("gray20", 1/2))

 plot(xd[iN], abs(re[iN]), type="b", cex=1/2, log="y",
      main = "| relErr( log1pmx(x) ) |  {via 'Rmpfr'}",
      ylim = c(4e-17, max(abs(asNumeric(re)[iN]))))
 abline(h = c(1:2,4)*2^-52, lty=2, col=adjustcolor("gray20", 1/2))
 mL1 <- eval(formals(log1pmx)$minL1)
 abline(v = mL1, lwd=3, col=adjustcolor(2, 1/2))
 axis(3, at=mL1, "minL1", col.axis=2, col=2)

 re2 <- asNumeric(sfsmisc::relErrV(lg1pM.[iN], log1pmx(xd[iN], minL1 = -0.7)))
 lines(xd[iN], abs(re2), col=adjustcolor(4, 1/2), lwd=2)
 abline(v = -0.7, lwd=3, col=adjustcolor(4, 2/3), lty=3)
axis(3, line=-1, at=-0.7, "mL1 = -0.7", col.axis=4, col=4) ## it seems that would be better

 re3 <- asNumeric(sfsmisc::relErrV(lg1pM.[iN], log1pmx(xd[iN], minL1 = -0.67)))
 lines(xd[iN], abs(re3), col=adjustcolor(6, 1/3), lwd=2)
 abline(v = -0.67, lwd=3, col=adjustcolor(6, 2/3), lty=3)
 axis(3, line=-2, at=-0.67, "mL1 = -0.67", col.axis=6, col=6)
 lines(lowess(xd[iN], abs(re[iN]), f=1/50), col=adjustcolor("gray", 1/2), lwd=6)
 lines(lowess(xd[iN], abs(re2),    f=1/50), col=adjustcolor(4, 1/2), lwd=6)
 lines(lowess(xd[iN], abs(re3),    f=1/50), col=adjustcolor(6, 1/2), lwd=6)
  # MM: I'm confused -- why does -0.7 not show problems to the right, but
  #                     but      -0.67 *does* show them .. ?


 re4 <- asNumeric(sfsmisc::relErrV(lg1pM.[iN], log1pmx(xd[iN], minL1 = -0.64)))
 lines(xd[iN], abs(re4), col=adjustcolor(7, 1/3), lwd=2)
 abline(v = -0.64, lwd=3, col=adjustcolor(7, 2/3), lty=3)
 axis(3, line=-2, at=-0.64, "mL1 = -0.64", col.axis=7, col=7)
 lines(lowess(xd[iN], abs(re[iN]), f=1/50), col=adjustcolor("gray", 1/2), lwd=6)
 lines(lowess(xd[iN], abs(re2),    f=1/50), col=adjustcolor(4, 1/2), lwd=6)
 lines(lowess(xd[iN], abs(re3),    f=1/50), col=adjustcolor(6, 1/2), lwd=6)
 lines(lowess(xd[iN], abs(re4),    f=1/50), col=adjustcolor(7, 1/2), lwd=6)
                           # ...

 re5 <- asNumeric(sfsmisc::relErrV(lg1pM.[iN], log1pmx(xd[iN], minL1 = -0.6)))
 lines(xd[iN], abs(re5), col=adjustcolor(8, 1/3), lwd=2)
 abline(v = -0.6, lwd=3, col=adjustcolor(8, 2/3), lty=3)
 axis(3, line=-1, at=-0.6, "mL1 = -0.6", col.axis=8, col=8)
 lines(lowess(xd[iN], abs(re[iN]), f=1/50), col=adjustcolor("gray", 1/2), lwd=6)
 lines(lowess(xd[iN], abs(re2),    f=1/50), col=adjustcolor(4, 1/2), lwd=6)
 lines(lowess(xd[iN], abs(re3),    f=1/50), col=adjustcolor(6, 1/2), lwd=6)
 lines(lowess(xd[iN], abs(re4),    f=1/50), col=adjustcolor(7, 1/2), lwd=6)
 lines(lowess(xd[iN], abs(re5),    f=1/50), col=adjustcolor(8, 1/2), lwd=6)

  # -0.6  now is clearly too large, -0.7 was better

})# if "Rmpfr"
```

logcf *Continued Fraction Approximation of Log-Related Power Series*

Description

Compute a continued fraction approximation to the series (infinite sum)

$$\sum_{k=0}^{\infty} \frac{x^k}{i + k \cdot d} = \frac{1}{i} + \frac{x}{i+d} + \frac{x^2}{i+2*d} + \frac{x^3}{i+3*d} + \ldots$$

Needed as auxiliary function in `log1pmx()` and `lgamma1p()`.

Usage

```
logcfR (x, i, d, eps, maxit = 10000L, trace = FALSE)
logcfR.(x, i, d, eps, maxit = 10000L, trace = FALSE)
logcf  (x, i, d, eps, trace = FALSE)
```

Arguments

`x`	numeric vector, "mpfr" now works, too.
`i`	positive numeric
`d`	non-negative numeric
`eps`	positive number, the convergence tolerance.
`maxit`	a positive integer, the maximal number of iterations or terms in the truncated series used.
`trace`	logical (or non-negative integer in the future) indicating if (and how much) diagnostic output should be printed to the console during the computations.

Details

`logcfR.()`: the first pure R version where the iterations happen vectorized in `x`, i.e., relatively fast; *however* convergence and rescaling are a "group decision" which is really suboptimal for reproducibility or careful comparisons.

`logcfR()`: a pure R version where each `x[i]` is treated separately, hence "properly" vectorized, but slowly so. Now recommended when `x` is an "mpfr"-number vector.

`logcf()`: only for `numeric` `x`, calls into (a clone of) R's own (non-API currently) `logcf()` C Mathlib function.

Value

a numeric-alike vector with the same attributes as `x`.

Note

Rescaling is done by (namespace hidden) "global" `scalefactor` which is 2^{256}, represented exactly (in `double` precision).

Author(s)

Martin Maechler, based on R's 'nmath/pgamma.c' implementation.

See Also

lgamma1p, log1pmx, and pbeta, whose prinicipal algorithm has evolved from TOMS 708.

Examples

```
l32 <- curve(logcf(x, 3,2, eps=1e-7), -3, 1)
abline(h=0,v=1, lty=3, col="gray50")
plot(y~x, l32, log="y", type = "o", main = "logcf(*, 3,2)  in log-scale")
```

---

logspace.add *Logspace Arithmetix – Addition and Subtraction*

Description

Compute the log(arithm) of a sum (or difference) from the log of terms without causing overflows and without throwing away large handfuls of accuracy.

`logspace.add(lx, ly)`:=

$$\log(\exp(lx) + \exp(ly))$$

`logspace.sub(lx, ly)`:=

$$\log(\exp(lx) - \exp(ly))$$

Usage

```
logspace.add(lx, ly)
logspace.sub(lx, ly)
```

Arguments

lx, ly	numeric vectors, typically of the same length, but will be recycled to common length as with other R arithmetic.

Value

a numeric vector of the same length as x+y.

Note

This is really from R's C source code for pgamma(), i.e., '<R>/src/nmath/pgamma.c'

The function definitions are very simple, logspace.sub() using log1mexp().

Author(s)

Morten Welinder (for R's pgamma()); Martin Maechler

See Also

lsum, lssum; then pgamma()

Examples

```
set.seed(12)
ly <- rnorm(100, sd= 50)
lx <- ly + abs(rnorm(100, sd=100))  # lx - ly must be positive for *.sub()
stopifnot(exprs = {
   all.equal(logspace.add(lx,ly),
             log(exp(lx) + exp(ly)), tol=1e-14)
   all.equal(logspace.sub(lx,ly),
             log(exp(lx) - exp(ly)), tol=1e-14)
})
```

---

lssum *Compute Logarithm of a Sum with Signed Large Summands*

Description

Properly compute $\log(x_1 + \ldots + x_n)$ for given log absolute values lxabs = $log(|x_1|), .., log(|x_n|)$ and corresponding signs signs = $sign(x_1), .., sign(x_n)$. Here, x_i is of arbitrary sign.

Notably this works in many cases where the direct sum would have summands that had overflown to +Inf or underflown to -Inf.

This is a (simpler, vector-only) version of copula:::lssum() (CRAN package **copula**).

Note that the *precision* is often not the problem for the direct summation, as R's sum() internally uses "long double" precision on most platforms.

Usage

```
lssum(lxabs, signs, l.off = max(lxabs), strict = TRUE)
```

Arguments

lxabs	n-vector of values $\log(\lvert x_1\rvert), \ldots, \log(\lvert x_n\rvert)$.
signs	corresponding signs $sign(x_1), \ldots, sign(x_n)$.
l.off	the offset to substract and re-add; ideally in the order of max(.).
strict	logical indicating if the function should stop on some negative sums.

Value

$log(x_1 + .. + x_n) == log(sum(x)) = log(sum(sign(x) * |x|)) == log(sum(sign(x) * exp(log(|x|)))) == log(exp(log$

Author(s)

Marius Hofert and Martin Maechler (for package **copula**).

See Also

lsum() which computes an exponential sum in log scale with *out* signs.

Examples

```
rSamp <- function(n, lmean, lsd = 1/4, roundN = 16) {
  lax <- sort((1+1e-14*rnorm(n))*round(roundN*rnorm(n, m = lmean, sd = lsd))/roundN)
  sx <- rep_len(c(-1,1), n)
  list(lax=lax, sx=sx, x = sx*exp(lax))
}

set.seed(101)
L1 <- rSamp(1000, lmean = 700) # here, lssum() is not needed (no under-/overflow)
summary(as.data.frame(L1))
ax <- exp(lax <- L1$lax)
hist(lax); rug(lax)
hist( ax); rug( ax)
sx <- L1$sx
table(sx)
(lsSimple <- log(sum(L1$x)))           # 700.0373
(lsS <- lssum(lxabs = lax, signs = sx))# ditto
lsS - lsSimple # even exactly zero (in 64b Fedora 30 Linux which has nice 'long double')
stopifnot(all.equal(700.037327351478, lsS, tol=1e-14), all.equal(lsS, lsSimple))

L2 <- within(L1, { lax <- lax + 10; x <- sx*exp(lax) }) ; summary(L2$x) # some -Inf, +Inf
(lsSimpl2 <- log(sum(L2$ x)))                    # NaN
(lsS2 <- lssum(lxabs = L2$ lax, signs = L2$ sx)) # 710.0373
stopifnot(all.equal(lsS2, lsS + 10, tol = 1e-14))
```

`lsum` *Properly Compute the Logarithm of a Sum (of Exponentials)*

Description

Properly compute $\log(x_1 + \ldots + x_n)$. for given $log(x_1), .., log(x_n)$. Here, $x_i > 0$ for all i.

If the inputs are denoted $l_i = log(x_i)$ for $i = 1, 2, .., n$, we compute `log(sum(exp(l[])))`, numerically stably.

Simple vector version of `copula:::lsum()` (CRAN package **copula**).

Usage

```
lsum(lx, l.off = max(lx))
```

Arguments

`lx`	n-vector of values log(x_1),..,log(x_n).
`l.off`	the offset to substract and re-add; ideally in the order of the maximum of each column.

Value

$log(x_1 + .. + x_n) = log(sum(x)) = log(sum(exp(log(x)))) == log(exp(log(x_max)) * sum(exp(log(x) - log(x_max))$

Author(s)

Originally, via paired programming: Marius Hofert and Martin Maechler.

See Also

`lssum()` which computes a sum in log scale with specified (typically alternating) signs.

Examples

```
## The "naive" version :
lsum0 <- function(lx) log(sum(exp(lx)))

lx1 <- 10*(-80:70) # is easy
lx2 <- 600:750     # lsum0() not ok [could work with rescaling]
lx3 <- -(750:900)  # lsum0() = -Inf - not good enough
m3 <- cbind(lx1,lx2,lx3)
lx6 <- lx5 <- lx4 <- lx3
lx4[149:151] <- -Inf ## = log(0)
lx5[150] <- Inf
lx6[1] <- NA_real_
m6 <- cbind(m3,lx4,lx5,lx6)
stopifnot(exprs = {
  all.equal(lsum(lx1), lsum0(lx1))
  all.equal((ls1 <- lsum(lx1)),  700.000045400960403, tol=8e-16)
  all.equal((ls2 <- lsum(lx2)),  750.458675145387133, tol=8e-16)
  all.equal((ls3 <- lsum(lx3)), -749.541324854612867, tol=8e-16)
  ## identical: matrix-version <==> vector versions
  identical(lsum(lx4), ls3)
  identical(lsum(lx4), lsum(head(lx4, -3))) # the last three were -Inf
  identical(lsum(lx5), Inf)
  identical(lsum(lx6), lx6[1])
  identical((lm3 <- apply(m3, 2, lsum)), c(lx1=ls1, lx2=ls2, lx3=ls3))
  identical(apply(m6, 2, lsum), c(lm3, lx4=ls3, lx5=Inf, lx6=lx6[1]))
})
```

---

newton *Simple R level Newton Algorithm, Mostly for Didactical Reasons*

Description

Given the function `G()` and its derivative `g()`, `newton()` uses the Newton method, starting at `x0`, to find a point xp at which G is zero. `G()` and `g()` may each depend on the same parameter (vector) `z`.

Convergence typically happens when the stepsize becomes smaller than `eps`.

`keepAll = TRUE` to also get the vectors of consecutive values of x and G(x, z);

Usage

```
newton(x0, G, g, z,
       xMin = -Inf, xMax = Inf, warnRng = TRUE,
       dxMax = 1000, eps = 0.0001, maxiter = 1000L,
       warnIter = missing(maxiter) || maxiter >= 10L,
       keepAll = NA)
```

Arguments

x0	numeric start value.
G, g	must be functions, mathematically of their first argument, but they can accept parameters; g() must be the derivative of G.
z	parameter vector for $G()$ and $g()$, to be kept fixed.
xMin, xMax	numbers defining the allowed range for x during the iterations; e.g., useful to set to 0 and 1 during quantile search.
warnRng	logical specifying if a warning should be signalled when start value x0 is outside [xMin,xMax] and hence will be changed to one of the boundary values.
dxMax	maximal step size in x-space. (The default 1000 is quite arbitrary, do set a good maximal step size yourself!)
eps	positive number, the *absolute* convergence tolerance.
maxiter	positive integer, specifying the maximal number of Newton iterations.
warnIter	logical specifying if a warning should be signalled when the algorithm has not converged in maxiter iterations.
keepAll	logical specifying if the full sequence of x- and G(x,*) values should be kept and returned: NA, the default: newton returns a small list of final "data", with 4 components x= $x*$, G= $G(x*, z)$, it, and converged. TRUE: returns an extended list, in addition containing the vectors x.vec and G.vec. FALSE: returns only the $x*$ value.

Details

Because of the quadrativc convergence at the end of the Newton algorithm, often x^* satisfies approximately $|G(x*, z)| < eps^2$.

newton() can be used to compute the quantile function of a distribution, if you have a good starting value, and provide the cumulative probability and density functions as R functions G and g respectively.

Value

The result always contains the final x-value $x*$, and typically some information about convergence, depending on the value of keepAll, see above:

x	the optimal x^* value (a number).
G	the function value $G(x*, z)$, typically very close to zero.
it	the integer number of iterations used.
convergence	logical indicating if the Newton algorithm converged within maxiter iterations.
x.vec	the full vector of x values, $\{x0, \ldots, x^*\}$.
G.vec	the vector of function values (typically tending to zero), i.e., G(x.vec,.) (even when G(x,.) would not vectorize).

Author(s)

Martin Maechler, ca. 2004

References

Newton's Method on Wikipedia, https://en.wikipedia.org/wiki/Newton%27s_method.

See Also

uniroot() is much more sophisticated, works without derivatives and is generally faster than newton().

newton(.) is currently crucially used (only) in our function qchisqN().

Examples

```
## The most simple non-trivial case :  Computing SQRT(a)
  G <- function(x, a) x^2 - a
  g <- function(x, a) 2*x

  newton(1, G, g, z = 4  ) # z = a -- converges immediately
  newton(1, G, g, z = 400) # bad start, needs longer to converge

## More interesting, and related to non-central (chisq, e.t.) computations:
## When is  x * log(x) < B,  i.e., the inverse function of G = x*log(x) :
xlx <- function(x, B) x*log(x) - B
dxlx <- function(x, B) log(x) + 1

Nxlx <- function(B) newton(B, G=xlx, g=dxlx, z=B, maxiter=Inf)$x
N1   <- function(B) newton(B, G=xlx, g=dxlx, z=B, maxiter = 1)$x
N2   <- function(B) newton(B, G=xlx, g=dxlx, z=B, maxiter = 2)$x

Bs <- c(outer(c(1,2,5), 10^(0:4)))
plot (Bs, vapply(Bs, Nxlx, pi), type = "l", log ="xy")
lines(Bs, vapply(Bs, N1  , pi), col = 2, lwd = 2, lty = 2)
lines(Bs, vapply(Bs, N2  , pi), col = 3, lwd = 3, lty = 3)

BL <- c(outer(c(1,2,5), 10^(0:6)))
plot (BL, vapply(BL, Nxlx, pi), type = "l", log ="xy")
lines(BL, BL, col="green2", lty=3)
lines(BL, vapply(BL, N1  , pi), col = 2, lwd = 2, lty = 2)
lines(BL, vapply(BL, N2  , pi), col = 3, lwd = 3, lty = 3)
## Better starting value from an approximate 1 step Newton:
iL1 <- function(B) 2*B / (log(B) + 1)
lines(BL, iL1(BL), lty=4, col="gray20") ## really better ==> use it as start

Nxlx <- function(B) newton(iL1(B), G=xlx, g=dxlx, z=B, maxiter=Inf)$x
N1   <- function(B) newton(iL1(B), G=xlx, g=dxlx, z=B, maxiter = 1)$x
N2   <- function(B) newton(iL1(B), G=xlx, g=dxlx, z=B, maxiter = 2)$x

plot (BL, vapply(BL, Nxlx, pi), type = "o", log ="xy")
lines(BL, iL1(BL),  lty=4, col="gray20")
lines(BL, vapply(BL, N1  , pi), type = "o", col = 2, lwd = 2, lty = 2)
lines(BL, vapply(BL, N2  , pi), type = "o", col = 3, lwd = 2, lty = 3)
## Manual 2-step Newton
iL2 <- function(B) { lB <- log(B) ; B*(lB+1) / (lB * (lB - log(lB) + 1)) }
lines(BL, iL2(BL), col = adjustcolor("sky blue", 0.6), lwd=6)
##==>  iL2() is very close to true curve
## relative error:
iLtrue <- vapply(BL, Nxlx, pi)
cbind(BL, iLtrue, iL2=iL2(BL), relErL2 = 1-iL2(BL)/iLtrue)
```

```
## absolute error (in log-log scale; always positive!):
plot(BL, iL2(BL) - iLtrue, type = "o", log="xy", axes=FALSE)
if(requireNamespace("sfsmisc")) {
  sfsmisc::eaxis(1)
  sfsmisc::eaxis(2, sub10=2)
} else {
  cat("no 'sfsmisc' package; maybe  install.packages(\"sfsmisc\")  ?\n")
  axis(1); axis(2)
}
## 1 step from iL2()  seems quite good:
B. <- BL[-1] # starts at 2
NL2 <- lapply(B., function(B) newton(iL2(B), G=xlx, g=dxlx, z=B, maxiter=1))
str(NL2)
iL3 <- sapply(NL2, `[[`, "x")
cbind(B., iLtrue[-1], iL2=iL2(B.), iL3, relE.3 = 1- iL3/iLtrue[-1])
x. <- iL2(B.)
all.equal(iL3, x. - xlx(x., B.) / dxlx(x.)) ## 7.471802e-8
## Algebraic simplification of one newton step :
all.equal((x.+B.)/(log(x.)+1), x. - xlx(x., B.) / dxlx(x.), tol = 4e-16)
iN1 <- function(x, B) (x+B) / (log(x) + 1)
B <- 12345
iN1(iN1(iN1(B, B),B),B)
Nxlx(B)
```

---

numer-utils *Numerical Utilities - Functions, Constants*

Description

The **DPQ** package provides some numeric constants used in some of its distribution computations.

`all_mpfr()` and `any_mpfr()` return `TRUE` iff all (or 'any', respectively) of their arguments inherit from `class` `"mpfr"` (from package **Rmpfr**).

`logr(x,a)` computes `log(x / (x + a))` in a numerically stable way.

Usage

```
## Numeric Constants : % mostly in   ../R/beta-fns.R
M_LN2        # = log(2)  = 0.693....
M_SQRT2      # = sqrt(2) = 1.4142...
M_cutoff     # := If |x| > |k| * M_cutoff, then  log[ exp(-x) * k^x ]  =~=  -x
             #  = 3196577161300663808 ~= 3.2e+18
M_minExp     # = log(2) * .Machine$double.min.exp # ~= -708.396..
G_half       # = sqrt(pi) = Gamma( 1/2 )

## Functions :
all_mpfr(...)
any_mpfr(...)
logr(x, a)    # == log(x / (x + a)) -- but numerically smart; x >= 0, a > -x
okLongDouble(lambda = 999, verbose = 0L, tol = 1e-15)
```

Arguments

`...`	numeric or `"mpfr"` numeric vectors.
`x, a`	number-like, not negative, now may be vectors of `length(.) > 1`.
`lambda`	a number, typically in the order of 500–10'000.
`verbose`	a non-negative integer, if not zero, `okLongDouble()` prints the intermediate long double computations' results.
`tol`	numerical tolerance used to determine the accuracy required for near equality in `okLongDouble()`.

Details

`all_mpfr()`,

`all_mpfr()` : test if `all` or `any` of their arguments or of class `"mpfr"` (from package **Rmpfr**). The arguments are evaluated only until the result is determined, see the example.

`logr()` computes $\log(x/(x+a))$ in a numerically stable way.

Value

The numeric constant in the first case; a numeric (or "mpfr") vector of appropriate size in the 2nd case.

`okLongDouble()` returns a `logical`, `TRUE` iff the long double arithmetic with `expl()` and `logl()` seems to work accurately

Author(s)

Martin Maechler

See Also

`.Machine`

Examples

```
(Ms <- ls("package:DPQ", pattern = "^M"))
lapply(Ms, function(nm) { cat(nm,": "); print(get(nm)) }) -> .tmp

logr(1:3, a=1e-10)

okLongDouble() # typically TRUE, but not e.g. in a valgrinded R-devel of Oct.2019
## Here is typically the "boundary":
okLongDouble(11355, verbose=TRUE) # typically TRUE  (also for lambda <= 11355)
okLongDouble(11356, verbose=TRUE) # typically FALSE (also for lambda >= 11356)
```

p1l1 *Numerically Stable p1l1(t) = (t+1)*log(1+t) - t*

Description

The binomial deviance function `bd0(x,M)` can mathematically be re-written as $bd0(x, M) = M * p1l1((x - M)/M)$ where we look into providing numerically stable formula for $p1l1(t)$ as it's mathematical formula $p1l1(t) = (t + 1)\log(1 + t) - t$ suffers from cancellation for small $|t|$, even when `log1p(t)` is used instead of `log(1+t)`.

Using a hybrid implementation, `p1l1()` uses a direct formula, now the stable one in `p1l1p()`, for $|t| > c$ and a series approximation for $|t| \le c$ for some c.

NB: The re-expression `log1pmx()` is almost perfect; it fixes the cancellation problem entirely (and exposes the fact that `log1pmx()`'s internal cutoff seems sub optimal.

Usage

```
p1l1p  (t, ...)
p1l1.  (t)
p1l1   (t,    F = t^2/2)
p1l1ser(t, k, F = t^2/2)
```

Arguments

`t`	numeric vector ("mpfr" included), larger (or equal) to -1.
`...`	optional (tuning) arguments, passed to `log1pmx()`.
`k`	small positive integer, the number of terms to use in the Taylor series approximation `p1l1ser(t,k)` of `p1l1(t)`.
`F`	numeric vector of multiplication **f**actor; *must* be `t^2/2` for the `p1l1()` function, but can be modified, e.g. in more direct `bd0()` computations.

Details

for now see in `bd0()`.

Value

numeric vector "as" `x`.

Author(s)

Martin Maechler

See Also

`bd0`; `dbinom` the latter for the C.Loader(2000) reference.

Examples

```
t <- seq(-1, 4, by=1/64)
plot(t, p1l1ser(t, 1), type="l")
lines(t, p1l1.(t), lwd=5, col=adjustcolor(1, 1/2)) # direct formula
for(k in 2:6) lines(t, p1l1ser(t, k), col=k)

## zoom in
t <- 2^seq(-59,-1, by=1/4)
t <- c(-rev(t), 0, t)
stopifnot(!is.unsorted(t))
k.s <- 1:12; names(k.s) <- paste0("k=", 1:12)

## True function values: use Rmpfr with 256 bits precision: ---
### eventually move this to ../tests/ & ../vignettes/
#### FIXME: eventually replace with  if(requireNamespace("Rmpfr")){ ......}
#### =====
if((needRmpfr <- is.na(match("Rmpfr", (srch0 <- search())))))
    require("Rmpfr")
p1l1.T <- p1l1.(mpfr(t, 256)) # "true" values
p1l1.n <- asNumeric(p1l1.T)
p1tab <-
    cbind(b1 = bd0(t+1, 1),
          b.10 = bd0(10*t+10,10)/10,
          dirct = p1l1.(t),
          p1l1p = p1l1p(t),
          p1l1  = p1l1 (t),
          sapply(k.s, function(k) p1l1ser(t,k)))
matplot(t, p1tab, type="l", ylab = "p1l1*(t)")
## (absolute) error:
##' legend for matplot()
mpLeg <- function(leg = colnames(p1tab), xy = "top", col=1:6, lty=1:5, lwd=1,
                  pch = c(1L:9L, 0L, letters, LETTERS)[seq_along(leg)], ...)
    legend(xy, legend=leg, col=col, lty=lty, lwd=lwd, pch=pch, ncol=3, ...)

titAbs <- "Absolute errors of p1l1(t) approximations"
matplot(t, asNumeric(p1tab - p1l1.T), type="o", main=titAbs); mpLeg()
i <- abs(t) <= 1/10 ## zoom in a bit
matplot(t[i], abs(asNumeric((p1tab - p1l1.T)[i,])), type="o", log="y",
        main=titAbs, ylim = c(1e-18, 0.003)); mpLeg()
## Relative Error
titR <- "|Relative error| of p1l1(t) approximations"
matplot(t[i], abs(asNumeric((p1tab/p1l1.T - 1)[i,])), type="o", log="y",
        ylim = c(1e-18, 2^-10), main=titR)
mpLeg(xy="topright", bg= adjustcolor("gray80", 4/5))
i <- abs(t) <= 2^-10 # zoom in more
matplot(t[i], abs(asNumeric((p1tab/p1l1.T - 1)[i,])), type="o", log="y",
        ylim = c(1e-18, 1e-9))
mpLeg(xy="topright", bg= adjustcolor("gray80", 4/5))


## Correct number of digits
corDig <- asNumeric(-log10(abs(p1tab/p1l1.T - 1)))
cbind(t, round(corDig, 1))# correct number of digits

matplot(t, corDig, type="o", ylim = c(1,17))
(cN <- colnames(corDig))
```

```
legend(-.5, 14, cN, col=1:6, lty=1:5, pch = c(1L:9L, 0L, letters), ncol=2)

## plot() function >>>> using global (t, corDig) <<<<<<<<<
p.relEr <- function(i, ylim = c(11,17), type = "o",
                    leg.pos = "left", inset=1/128,
                    main = sprintf(
                       "Correct #{Digits} in p1l1() approx., notably Taylor(k=1 .. %d)",
                                   max(k.s)))
{
    if((neg <- all(t[i] < 0)))
        t  <- -t
    stopifnot(all(t[i] > 0), length(ylim) == 2) # as we use log="x"
    matplot(t[i], corDig[i,], type=type, ylim=ylim, log="x", xlab = quote(t), xaxt="n",
            main=main)
    legend(leg.pos, cN, col=1:6, lty=1:5, pch = c(1L:9L, 0L, letters), ncol=2,
           bg=adjustcolor("gray90", 7/8), inset=inset)
    t.epsC <- -log10(c(1,2,4)* .Machine$double.eps)
    axis(2, at=t.epsC, labels = expression(epsilon[C], 2*epsilon[C], 4*epsilon[C]),
         las=2, col=2, line=1)
    tenRs <- function(t) floor(log10(min(t))) : ceiling(log10(max(t)))
    tenE <- tenRs(t[i])
    tE <- 10^tenE
    abline (h = t.epsC,
            v = tE, lty=3, col=adjustcolor("gray",.8), lwd=2)
    AX <- if(requireNamespace("sfsmisc")) sfsmisc::eaxis else axis
    AX(1, at= tE, labels = as.expression(
                      lapply(tenE,
                             if(neg)
                                 function(e) substitute(-10^{E}, list(E = e+0))
                             else
                                 function(e) substitute( 10^{E}, list(E = e+0)))))
}

p.relEr(t > 0, ylim = c(1,17))
p.relEr(t > 0) # full positive range
p.relEr(t < 0) # full negative range
if(FALSE) {## (actually less informative):
 p.relEr(i = 0 < t & t < .01)  ## positive small t
 p.relEr(i = -.1 < t & t < 0) ## negative small t
}

## Find approximate formulas for accuracy of k=k*  approximation
d.corrD <- cbind(t=t, as.data.frame(corDig))
names(d.corrD) <- sub("k=", "nC_",  names(d.corrD))

fmod <- function(k, data, cut.y.at = -log10(2 * .Machine$double.eps),
                 good.y = -log10(.Machine$double.eps), # ~ 15.654
                 verbose=FALSE) {
    varNm <- paste0("nC_",k)
    stopifnot(is.numeric(y <- get(varNm, data, inherits=FALSE)),
              is.numeric(t <- data$t))# '$' works for data.frame, list, environment
    i <- 3 <= y & y <= cut.y.at
    i.pos <- i & t > 0
    i.neg <- i & t < 0
    if(verbose) cat(sprintf("k=%d >> y <= %g ==> #{pos. t} = %d ;  #{neg. t} = %d\n",
                            k, cut.y.at, sum(i.pos), sum(i.neg)))
    nCoefLm <- function(x,y) `names<-`(.lm.fit(x=x, y=y)$coeff, c("int", "slp"))
```

```
   nC.t <- function(x,y) { cf <- nCoefLm(x,y); c(cf, t.0 = exp((good.y - cf[[1]])/cf[[2]])) }
    cbind(pos = nC.t(cbind(1, log( t[i.pos])), y[i.pos]),
          neg = nC.t(cbind(1, log(-t[i.neg])), y[i.neg]))
}
rr <- sapply(k.s, fmod, data=d.corrD, verbose=TRUE, simplify="array")
stopifnot(rr["slp",,] < 0) # all slopes are negative (important!)
matplot(k.s, t(rr["slp",,]), type="o", xlab = quote(k), ylab = quote(slope[k]))
## fantastcally close to linear in k
## The numbers, nicely arranged
ftable(aperm(rr, c(3,2,1)))
signif(t(rr["t.0",,]),3) # ==> Should be boundaries for the hybrid p1l1()
##           pos      neg
## k=1  6.60e-16 6.69e-16
## k=2  3.65e-08 3.65e-08
## k=3  1.30e-05 1.32e-05
## k=4  2.39e-04 2.42e-04
## k=5  1.35e-03 1.38e-03
## k=6  4.27e-03 4.34e-03
## k=7  9.60e-03 9.78e-03
## k=8  1.78e-02 1.80e-02
## k=9  2.85e-02 2.85e-02
## k=10 4.13e-02 4.14e-02
## k=11 5.62e-02 5.64e-02
## k=12 7.24e-02 7.18e-02

###------------- Well,  p1l1p()  is really basically good enough ... with a small exception:
rErr1k <- curve(asNumeric(p1l1p(x) / p1l1.(mpfr(x, 4096)) - 1), -.999, .999,
                n = 4000, col=2, lwd=2)
abline(h = c(-8,-4,-2:2,4,8)* 2^-52, lty=2, col=adjustcolor("gray20", 1/4))
## well, have a "spike" at around -0.8 -- why?

plot(abs(y) ~ x, data = rErr1k, ylim = c(4e-17, max(abs(y))),
     ylab=quote(abs(hat(p)/p - 1)),
     main = "p1l1p(x) -- Relative Error wrt mpfr(*. 4096) [log]",
     col=2, lwd=1.5, type = "b", cex=1/2, log="y", yaxt="n")
sfsmisc::eaxis(2)
eps124 <-  c(1, 2,4,8)* 2^-52
abline(h = eps124, lwd=c(3,1,1,1), lty=c(1,2,2,2), col=adjustcolor("gray20", 1/4))
axLab <- expression(epsilon[c], 2*epsilon[c], 4*epsilon[c], 8*epsilon[c])
axis(4, at = eps124, labels = axLab, col="gray20", las=1)
abline(v= -.791, lty=3, lwd=2, col="blue4") # -.789  from visual ..
##--> The "error" is in log1pmx() which has cutoff minLog1Value = -0.79149064
##--> which is clearly not optimal, at least not for computing p1l1p()

d <- 1/2048; x <- seq(-1+d, 1, by=d)
p1l1Xct <- p1l1.(mpfr(x, 4096))
rEx.5 <- asNumeric(p1l1p(x, minL1 = -0.5) / p1l1Xct - 1)
lines(x, abs(rEx.5), lwd=2.5, col=adjustcolor(4, 1/2)); abline(v=-.5, lty=2,col=4)
rEx.25 <- asNumeric(p1l1p(x, minL1 = -0.25) / p1l1Xct - 1)
lines(x, abs(rEx.25), lwd=3.5, col=adjustcolor(6, 1/2)); abline(v=-.25, lty=2,col=6)
lines(lowess(x, abs(rEx.5),  f=1/20), col=adjustcolor(4,offset=rep(1,4)/3), lwd=3)

lines(lowess(x, abs(rEx.25), f=1/20), col=adjustcolor(6,offset=rep(1,4)/3), lwd=
3)

rEx.4 <- asNumeric(p1l1p(x, tol_logcf=1e-15, minL1 = -0.4) / p1l1Xct - 1)
lines(x, abs(rEx.4), lwd=5.5, col=adjustcolor("brown", 1/2)); abline(v=-.25, lty=2,col="brown")
```

```
if(needRmpfr && isNamespaceLoaded("Rmpfr"))
    detach("package:Rmpfr")
```

---

pbetaRv1 *Pure R Implementation of Old pbeta()*

Description

pbetaRv1() is an implementation of the original ("version 1" pbeta() function in R (versions <= 2.2.x), before we started using TOMS 708 bratio() instead, see that help page also for references.

pbetaRv1() is basically a manual translation from C to R of the underlying pbeta_raw() C function, see in R's source tree at https://svn.r-project.org/R/branches/R-2-2-patches/src/nmath/pbeta.c

For consistency within R, we are using R's argument names (q,shape1,shape2) instead of C code's (x,pin,qin).

It is only for the *central* beta distribution.

Usage

```
pbetaRv1(q, shape1, shape2, lower.tail = TRUE,
         eps = 0.5 * .Machine$double.eps,
         sml = .Machine$double.xmin,
         verbose = 0)
```

Arguments

q, shape1, shape2	non-negative numbers, q in $[0, 1]$, see pbeta.
lower.tail	indicating if $F(q; *)$ should be returned or the upper tail probability $1 - F(q)$.
eps	the tolerance used to determine congerence. eps has been hard coded in C code to 0.5 * .Machine$double.eps which is equal to 2^{-53} or 1.110223e-16.
sml	the smallest positive number on the typical platform. The default .Machine$double.xmin is hard coded in the C code (as DBL_MIN), and this is equal to 2^{-1022} or 2.225074e-308 on all current platforms.
verbose	integer indicating the amount of verbosity of diagnostic output, 0 means no output, 1 more, etc.

Value

a number.

Note

The C code contains
This routine is a translation into C of a Fortran subroutine by W. Fullerton of Los Alamos Scientific Laboratory.

Author(s)

Martin Maechler

References

(From the C code:)

Nancy E. Bosten and E.L. Battiste (1974). Remark on Algorithm 179 (S14): Incomplete Beta Ratio. *Communications of the ACM*, **17**(3), 156–7.

See Also

`pbeta`.

Examples

```
all.equal(pbetaRv1(1/4, 2, 3),
          pbeta   (1/4, 2, 3))
set.seed(101)

N <- 1000
x <- sample.int(7, N, replace=TRUE) / 8
a <-   rlnorm(N)
b <- 5*rlnorm(N)
pbt <- pbeta(x, a, b)
for(i in 1:N) {
   stopifnot(all.equal(pbetaRv1(x[i], a[i], b[i]), pbt[i]))
   cat(".", if(i %% 20 == 0) paste0(i, "\n"))
}
```

---

phyperAllBin *Compute Hypergeometric Probabilities via Binomial Approximations*

Description

- `phyperAllBinM()` computes all four Molenaar binomial approximations to the hypergeometric cumulative distribution function `phyper()`.
- `phyperAllBin()` computes Molenaar's four, plus the other four `phyperBin.1()`, `*.2`, `*.3`, and `*.4`.

Usage

```
phyperAllBin (m, n, k, q = .suppHyper(m, n, k), lower.tail = TRUE, log.p = FALSE)
phyperAllBinM(m, n, k, q = .suppHyper(m, n, k), lower.tail = TRUE, log.p = FALSE)
.suppHyper(m, n, k)
```

Arguments

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 $0, 1, \ldots, m + n$.
q	vector of quantiles representing the number of white balls drawn without replacement from an urn which contains both black and white balls. The default, `.suppHyper(m,n,k)` provides the full (finite) support.
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.
log.p	logical; if TRUE, probabilities p are given as log(p).

Value

the `phyperAllBin*()` functions return a numeric `matrix`, with each column a different approximation to `phyper(m,n,k,q,lower.tail,log.p)`.

Note that the columns of `phyperAllBinM()` are a *subset* of those from `phyperAllBin()`.

Author(s)

Martin Maechler

References

See those in `phyperBinMolenaar`.

See Also

`phyperBin.1` etc, and `phyperBinMolenaar`.

`phyper`

Examples

```
.suppHyper # very simple:
stopifnot(identical(.suppHyper, ignore.environment = TRUE,
         function (m, n, k) max(0, k-n):min(k, m)))

phBall <- phyperAllBin (5,15, 7)
phBalM <- phyperAllBinM(5,15, 7)
stopifnot(identical(
            phBall[, colnames(phBalM)] ,
            phBalM)
         , .suppHyper(5, 15, 7) == 0:5
)

round(phBall, 4)
## relative Error: number of correct digits =
cbind(q = 0:5, round(-log10(abs(1 - phBall / phyper(0:5, 5,15,7))),  digits=2))
```

phyperApprAS152 *Normal Approximation to cumulative Hyperbolic Distribution – AS 152*

Description

Compute the normal approximation (via `pnorm(.)` from AS 152 to the cumulative hyperbolic distribution function `phyper()`.

Usage

```
phyperApprAS152(q, m, n, k)
```

Arguments

`q`	vector of quantiles representing 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 $0, 1, \ldots, m + n$.

Value

a `numeric` vector of the same length (etc) as `q`.

Note

I have Fortran (and C code translated from Fortran) which says

```
ALGORITHM AS R77  APPL. STATIST. (1989), VOL.38, NO.1
Replaces AS 59 and AS 152
Incorporates AS R86 from vol.40(2)
```

Author(s)

Martin Maechler, 19 Apr 1999

References

Lund, Richard E. (1980) Algorithm AS 152: Cumulative Hypergeometric Probabilities. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, **29**(2), 221–223. doi: 10.2307/2986315

Shea, B. (1989) Remark AS R77: A Remark on Algorithm AS 152: Cumulative Hypergeometric Probabilities. *JRSS C (Applied Statistics)*, **38**(1), 199–204. doi: 10.2307/2347696

Berger, R. (1991) Algorithm AS R86: A Remark on Algorithm AS 152: Cumulative Hypergeometric Probabilities. *JRSS C (Applied Statistics)*, **40**(2), 374–375. doi: 10.2307/2347606

See Also

`phyper`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (q, m, n, k)
{
    kk <- n
    nn <- m
    mm <- m + n
    ll <- q
    mean <- kk * nn/mm
    sig <- sqrt(mean * (mm - nn)/mm * (mm - kk)/(mm - 1))
    pnorm(ll + 1/2, mean = mean, sd = sig)
  }
```

---

phyperBin *HyperGeometric Distribution via Approximate Binomial Distribution*

Description

Compute hypergeometric cumulative probabilities via (good) binomial distribution approximations. The arguments of these functions are *exactly* those of R's own `phyper()`.

...

Usage

```
phyperBin.1(q, m, n, k, lower.tail = TRUE, log.p = FALSE)
phyperBin.2(q, m, n, k, lower.tail = TRUE, log.p = FALSE)
phyperBin.3(q, m, n, k, lower.tail = TRUE, log.p = FALSE)
phyperBin.4(q, m, n, k, lower.tail = TRUE, log.p = FALSE)
```

Arguments

q	vector of quantiles representing 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; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.
log.p	logical; if TRUE, probabilities p are given as log(p).

Value

...

Author(s)

Martin Maechler

See Also

phyper, pbinom

Examples

```
## The function is simply defined as
function (q, m, n, k, lower.tail = TRUE, log.p = FALSE)
  pbinom(q, size = k, prob = m/(m + n), lower.tail = lower.tail,
         log.p = log.p)
```

---

phyperBinMolenaar *HyperGeometric Distribution via Molenaar's Binomial Approximation*

Description

Compute hypergeometric cumulative probabilities via Molenaar's binomial approximations. The arguments of these functions are *exactly* those of R's own phyper().

. . .

Usage

```
phyperBinMolenaar  (q, m, n, k, lower.tail = TRUE, log.p = FALSE)
phyperBinMolenaar.1(q, m, n, k, lower.tail = TRUE, log.p = FALSE)
phyperBinMolenaar.2(q, m, n, k, lower.tail = TRUE, log.p = FALSE)
phyperBinMolenaar.3(q, m, n, k, lower.tail = TRUE, log.p = FALSE)
phyperBinMolenaar.4(q, m, n, k, lower.tail = TRUE, log.p = FALSE)
```

Arguments

q	vector of quantiles representing 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; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.
log.p	logical; if TRUE, probabilities p are given as log(p).

Value

. . .

Author(s)

Martin Maechler

References

Johnson, N.L., Kotz, S. and Kemp, A.W. (1992) Univariate Discrete Distributions, 2nd ed.; Wiley. Chapter 6, mostly Section *5 Approximations and Bounds*, p.256 ff

See Also

phyper, the hypergeometric distribution, and R's own "exact" computation. pbinom, the binomial distribution functions.

Examples

```
## The function is currently defined as
function (q, m, n, k, lower.tail = TRUE, log.p = FALSE)
pbinom(q, size = k, prob = hyper2binomP(q, m, n, k), lower.tail = lower.tail,
    log.p = log.p)
```

---

phyperIbeta — *Pearson's incomplete Beta Approximation to the Hyperbolic Distribution*

Description

Pearson's incomplete Beta function approximation to the cumulative hyperbolic distribution function phyper(.).

Note that in R, pbeta() provides a version of the incomplete Beta function.

Usage

```
phyperIbeta(q, m, n, k)
```

Arguments

q	vector of quantiles representing 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 $0, 1, \ldots, m + n$.

Value

a numeric vector "like" q with values approximately equal to phyper(q,m,n,k).

Author(s)

Martin Maechler

References

Johnson, Kotz & Kemp (1992): (6.90), p.260 – Bol'shev (1964)

See Also

phyper.

Examples

```
## The function is currently defined as
function (q, m, n, k)
{
    Np <- m
    N <- n + m
    n <- k
    x <- q
    p <- Np/N
    np <- n * p
    xi <- (n + Np - 1 - 2 * np)/(N - 2)
    d.c <- (N - n) * (1 - p) + np - 1
    cc <- n * (n - 1) * p * (Np - 1)/((N - 1) * d.c)
    lam <- (N - 2)^2 * np * (N - n) * (1 - p)/((N - 1) * d.c *
        (n + Np - 1 - 2 * np))
    pbeta(1 - xi, lam - x + cc, x - cc + 1)
  }
```

`phyperMolenaar` *Molenaar's Normal Approximations to the Hypergeometric Distribution*

Description

Compute Molenaar's two normal approximations to the (cumulative hypergeometric distribution `phyper()`.

Usage

```
phyper1molenaar(q, m, n, k)
phyper2molenaar(q, m, n, k)
```

Arguments

`q`	.
`m`	.
`n`	.
`k`	.

Details

Both approximations are from page 261 of J Johnson, Kotz & Kemp (1992). `phyper1molenaar` is formula (6.91), and `phyper2molenaar` is formula (6.92).

Value

. . .

Author(s)

Martin Maechler

References

Johnson, Kotz & Kemp (1992): p.261

See Also

phyper, pnorm.

Examples

```
## TODO
```

---

phyperPeizer *Peizer's Normal Approximation to the Cumulative Hyperbolic*

Description

Compute Peizer's extremely good normal approximation to the cumulative hyperbolic distribution.

This implementation corrects a typo in the reference

Usage

```
phyperPeizer(q, m, n, k)
```

Arguments

q	vector of quantiles representing 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 $0, 1, \ldots, m + n$.

Value

. .

Author(s)

Martin Maechler

References

Johnson, Kotz & Kemp (1992): (6.93) & (6.94), p.261 *CORRECTED* by M.M.

See Also

phyper.

Examples

```
## The function is currently defined as

phyperPeizer <- function(q, m, n, k)
{
  ## Purpose: Peizer's extremely good Normal Approx. to cumulative Hyperbolic
  ##  Johnson, Kotz & Kemp (1992):  (6.93) & (6.94), p.261 __CORRECTED__
  ## ------------------------------------------------------------------------
  Np <- m; N <- n + m; n <- k; x <- q
  ## (6.94) -- in proper order!
  nn <- Np ;  n. <- Np     + 1/6
  mm <- N - Np                  ;  m. <- N - Np + 1/6
  r <- n                        ;  r. <- n      + 1/6
  s <- N - n                    ;  s. <- N - n  + 1/6
                                   N. <- N      - 1/6
  A <- x + 1/2                  ;  A. <- x      + 2/3
  B <- Np - x - 1/2             ;  B. <- Np - x - 1/3
  C <- n  - x - 1/2             ;  C. <- n  - x - 1/3
  D <- N - Np - n + x + 1/2     ;  D. <- N - Np - n + x + 2/3

  n <- nn
  m <- mm
  ## After (6.93):
  L <-
    A * log((A*N)/(n*r)) +
    B * log((B*N)/(n*s)) +
    C * log((C*N)/(m*r)) +
    D * log((D*N)/(m*s))
  ## (6.93) :
  pnorm((A.*D. - B.*C.) / abs(A*D - B*C) *
        sqrt(2*L* (m* n* r* s* N.)/
                  (m.*n.*r.*s.*N )))
  # The book wrongly has an extra "2*" before `m* ' (after "2*L* (" ) above
}
```

---

phyperR — R*-only version of* R*'s original phyper() algorithm*

Description

An R version of the first `phyper()` algorithm in R, which was used up to svn rev `30227` on 2004-07-09.

Usage

```
phyperR(q, m, n, k)
```

Arguments

`q`	vector of quantiles representing 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 $0, 1, \ldots, m + n$.

Value

a numeric vector similar to `phyper(q,m,n,k)`.

Note

The original argument list in C was `(x,NR,NB,n)` where there were *red* and *black* balls in the urn.

Note that we have *vectorized* a translation to R of the original C code.

Author(s)

Martin Maechler

See Also

`phyper`.

Examples

```
m <- 9:12; n <- 7:10; k <- 10
x <- 0:(k+1) # length 12
for(. in 0:11) stopifnot(
  all.equal(phyper (., m=10, n=8, k=10),
            phyperR(., m=10, n=8, k=10), tol=1e-12))
## BUT the vectorization fails badly : FIXME! %% <<<<<< FIXME <<<<<<<<<<<<<<<
(A <- cbind(x, m, n, k)) # shows the recycling
cbind(A, ph = phyper (x, m, n, k),
      phR   = phyperR(x, m, n, k))
```

---

phyperR2 *Pure R version of R's C level phyper()*

Description

Use pure R functions to compute (less efficiently and usually even less accurately) hypergeometric (point) probabilities with the same "Welinder"-algorithm as R's C level code has been doing since 2004.

Apart from boundary cases, each `phyperR2()` call uses one corresponding `pdhyper()` call.

Usage

```
phyperR2(q, m, n, k, lower.tail = TRUE, log.p = FALSE, ...)
pdhyper (q, m, n, k,                    log.p = FALSE,
         epsC = .Machine$double.eps, verbose = getOption("verbose"))
```

Arguments

q	vector of quantiles representing 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; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.
log.p	logical; if TRUE, probabilities p are given as log(p).
...	further arguments, passed to pdhyper().
epsC	a non-negative number, the computer epsilon to be used; effectively a relative convergence tolerance for the while() loop in pdhyper().
verbose	logical indicating if the pdhyper() calls, typically one per phyperR2() call, should show how many terms have been computed and summed up.

Value

a number (as q).

pdhyper(q, m,n,k) computes the ratio phyper(q,m,n,k) / dhyper(q,m,n,k) but without computing numerator or denominator explicitly.

phyperR2() (in the non-boundary cases) then just computes the product dhyper(..) * pdhyper(..), of course "modulo" lower.tail and log.p transformations.

Consequently, it typically returns values very close to the corresponding R phyper(q,m,n,k,..) call.

Note

For now, all arguments of these functions must be of length **one**.

Author(s)

Martin Maechler, based on R's C code originally provided by Morton Welinder from the Gnumeric project, who thanks Ian Smith for ideas.

References

Morten Welinder (2004) phyper accuracy and efficiency; R bug report PR#6772.

See Also

phyper

Examples

```
## same example as phyper()
m <- 10; n <- 7; k <- 8
vapply(0:9, phyperR2, 0.1, m=m, n=n, k=k)  ==  phyper(0:9, m,n,k)
##  *all* TRUE (for 64b FC30)

## 'verbose=TRUE' to see the number of terms used:
vapply(0:9, phyperR2, 0.1, m=m, n=n, k=k,  verbose=TRUE)
```

```
## Larger arguments:
k <- 100 ; x <- .suppHyper(k,k,k)
ph  <- phyper(x, k,k,k)
ph2 <- vapply(x, phyperR2, 0.1, m=k, n=k, k=k)
cbind(x, ph, ph2, rE = 1-ph2/ph)
stopifnot(abs(1 -ph2/ph) < 8e-16) # 64bit FC30: see -2.22e-16 <= rE <= 3.33e-16
```

---

phypers *The Four (4) Symmetric phyper() calls.*

Description

Compute the four (4) symmetric phyper() calls which mathematically would be identical but in practice typically slightl differ numerically.

Usage

```
phypers(m, n, k, q = .suppHyper(m, n, k))
```

Arguments

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 $0, 1, \ldots, m + n$.
q	vector of quantiles representing the number of white balls drawn without replacement from an urn which contains both black and white balls. The default ..... FIXME

Value

a list with components

q	Description of 'comp1'
phyp	a numeric matrix of 4 columns with the 4 different calls to phyper() which are theoretically equivalent because of mathematical symmetry.

Author(s)

Martin Maechler

References

Johnson et al

See Also

phyper.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (m, n, k, q = .suppHyper(m, n, k))
{
    N <- m + n
    pm <- cbind(ph = phyper(q, m, n, k), p2 = phyper(q, k, N -
        k, m), Ip2 = phyper(m - 1 - q, N - k, k, m, lower.tail = FALSE),
        Ip1 = phyper(k - 1 - q, n, m, k, lower.tail = FALSE))
    stopifnot(all.equal(pm[, 1], pm[, 2]), all.equal(pm[, 2],
        pm[, 3]), all.equal(pm[, 3], pm[, 4]))
    list(q = q, phyp = pm)
  }
```

`pl2curves` *Plot 2 Noncentral Distribution Curves for Visual Comparison*

Description

Plot two noncentral (chi-squared or t or ..) distribution curves for visual comparison.

Usage

```
pl2curves(fun1, fun2, df, ncp, log = FALSE,
          from = 0, to = 2 * ncp, p.log = "", n = 2001,
          leg = TRUE, col2 = 2, lwd2 = 2, lty2 = 3, ...)
```

Arguments

`fun1, fun2`	`function()`s, both to be used via `curve()`, and called with the same 4 arguments, `(.,df,ncp,log)` (the name of the first argument is not specified).
`df, ncp, log`	parameters to be passed and used in both functions, which hence typically are non-central chi-squared or t density, probability or quantile functions.
`from, to`	numbers determining the x-range, passed to `curve()`.
`p.log`	string, passed as `curve(....,log = log.p)`.
`n`	the number of evaluation points, passed to `curve()`.
`leg`	logical specifying if a `legend()` should be drawn.
`col2, lwd2, lty2`	color, line width and line type for the second curve. (The first curve uses defaults for these graphical properties.)
`...`	further arguments passed to *first* `curve(..)` call.

Value

TODO: inivisible return both curve() results, i.e., (x,y1, y2), possibly as data frame

Author(s)

Martin Maechler

See Also

curve, ..

Examples

```
p.dnchiBessel <- function(df, ncp, log=FALSE, from=0, to = 2*ncp, p.log="", ...)
{
    pl2curves(dnchisqBessel, dchisq, df=df, ncp=ncp, log=log,
              from=from, to=to, p.log=p.log, ...)
}

  ## TODO the p.dnchiB()  examples  >>>>>> ../tests/chisq-nonc-ex.R <<<
```

---

pnbeta *Noncentral Beta Probabilities*

Description

pnbetaAppr2() and its inital version pnbetaAppr2v1() provide the "approximation 2" of Chattamvelli and Shanmugam(1997) to the noncentral Beta probability distribution.

pnbetaAS310() is an R level interface to a C translation (and "Rification") of the AS 310 Fortran implementation.

Usage

```
pnbetaAppr2(x, a, b, ncp = 0, lower.tail = TRUE, log.p = FALSE)

pnbetaAS310(x, a, b, ncp = 0, lower.tail = TRUE, log.p = FALSE,
            useAS226 = (ncp < 54.),
            errmax = 1e-6, itrmax = 100)
```

Arguments

x	numeric vector (of quantiles), typically from inside $[0, 1]$.
a, b	the shape parameters of Beta, aka as shape1 and shape2.
ncp	non-centrality parameter.
log.p	logical; if TRUE, probabilities p are given as log(p).
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.
useAS226	logical specifying if AS 226 (with R84 and R95 amendments) should be used which is said to be sufficient for small ncp. The default ncp < 54 had been hardwired in AS 310.
errmax	non-negative number determining convergence for AS 310.
itrmax	positive integer number, only if(useAS226) is passed to AS 226.

Value

a numeric vector of (log) probabilities of the same length as `x`.

Note

The authors in the reference compare AS 310 with Lam(1995), Frick(1990) and Lenth(1987) and state to be better than them. R's current (2019) noncentral beta implementation builds on these, too, with some amendments though; still, `pnbetaAS310()` may potentially be better, at least in certain corners of the 4-dimensional input space.

Author(s)

Martin Maechler; `pnbetaAppr2()` in Oct 2007.

References

Chattamvelli, R., and Shanmugam, R. (1997) Algorithm AS 310: Computing the Non-Central Beta Distribution Function. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* **46**(1), 146–156, for "approximation 2" notably p.154;

doi: 10.1111/14679876.00055.

Lenth, R. V. (1987) Algorithm AS 226, . . . , Frick, H. (1990)'s AS R84, . . . , and Lam, M.L. (1995)'s AS R95 : See 'References' in R's `pbeta` page.

See Also

R's own `pbeta`.

Examples

```
## Same arguments as for Table 1 (p.151) of the reference
a <- 5*rep(1:3, each=3)
aargs <- cbind(a = a, b = a,
               ncp = rep(c(54, 140, 170), 3),
               x = 1e-4*c(8640, 9000, 9560, 8686, 9000, 9000, 8787, 9000, 9220))
aargs
pnbA2 <- apply(aargs, 1, function(aa) do.call(pnbetaAppr2, as.list(aa)))
pnA310<- apply(aargs, 1, function(aa) do.call(pnbetaAS310, as.list(aa)))
aar2 <- aargs; dimnames(aar2)[[2]] <- c(paste0("shape", 1:2), "ncp", "q")
pnbR  <- apply(aar2,  1, function(aa) do.call(pbeta, as.list(aa)))
range(relD2   <- 1 - pnbA2 /pnbR)
range(relD310 <- 1 - pnA310/pnbR)
cbind(aargs, pnbA2, pnA310, pnbR,
      relD2 = signif(relD2, 3), relD310 = signif(relD310, 3)) # <------> Table 1
stopifnot(abs(relD2)   < 0.009) # max is 0.006286
stopifnot(abs(relD310) < 1e-5 ) # max is 6.3732e-6

## Arguments as for Table 2 (p.152) of the reference :
aarg2 <- cbind(a = c( 10, 10, 15, 20, 20, 20, 30, 30),
               b = c( 20, 10,  5, 10, 30, 50, 20, 40),
               ncp=c(150,120, 80,110, 65,130, 80,130),
               x = c(868,900,880,850,660,720,720,800)/1000)
pnbA2 <- apply(aarg2, 1, function(aa) do.call(pnbetaAppr2, as.list(aa)))
pnA310<- apply(aarg2, 1, function(aa) do.call(pnbetaAS310, as.list(aa)))
aar2 <- aarg2; dimnames(aar2)[[2]] <- c(paste0("shape", 1:2), "ncp", "q")
pnbR  <- apply(aar2,  1, function(aa) do.call(pbeta, as.list(aa)))
```

```
range(relD2   <- 1 - pnbA2 /pnbR)
range(relD310 <- 1 - pnA310/pnbR)
cbind(aarg2, pnbA2, pnA310, pnbR,
      relD2 = signif(relD2, 3), relD310 = signif(relD310, 3)) # <------> Table 2
stopifnot(abs(relD2  ) < 0.006) # max is 0.00412
stopifnot(abs(relD310) < 1e-5 ) # max is 5.5953e-6

## Arguments as for Table 3 (p.152) of the reference :
aarg3 <- cbind(a = c( 10, 10, 10, 15, 10, 12, 30, 35),
               b = c(  5, 10, 30, 20,  5, 17, 30, 30),
               ncp=c( 20, 54, 80,120, 55, 64,140, 20),
               x = c(644,700,780,760,795,560,800,670)/1000)
pnbA3 <- apply(aarg3, 1, function(aa) do.call(pnbetaAppr2, as.list(aa)))
pnA310<- apply(aarg3, 1, function(aa) do.call(pnbetaAS310, as.list(aa)))
aar3 <- aarg3; dimnames(aar3)[[2]] <- c(paste0("shape", 1:2), "ncp", "q")
pnbR  <- apply(aar3,  1, function(aa) do.call(pbeta, as.list(aa)))
range(relD2   <- 1 - pnbA3 /pnbR)
range(relD310 <- 1 - pnA310/pnbR)
cbind(aarg3, pnbA3, pnA310, pnbR,
      relD2 = signif(relD2, 3), relD310 = signif(relD310, 3)) # <------> Table 3
stopifnot(abs(relD2  ) < 0.09) # max is 0.06337
stopifnot(abs(relD310) < 1e-4) # max is 3.898e-5
```

`pnchi1sq` *(Probabilities of Non-Central Chi-squared Distribution for Special Cases*

Description

Computes probabilities for the non-central chi-squared distribution, in special cases, currently for `df = 1` and `df = 3`, using 'exact' formulas only involving the standard normal (Gaussian) cdf $\Phi()$ and its derivative $\phi()$, i.e., R's `pnorm()` and `dnorm()`.

Usage

```
pnchi1sq(q, ncp = 0, lower.tail = TRUE, log.p = FALSE, epsS = .01)
pnchi3sq(q, ncp = 0, lower.tail = TRUE, log.p = FALSE, epsS = .04)
```

Arguments

`q`	number ('quantile', i.e., abscissa value.)
`ncp`	non-centrality parameter δ; ....
`lower.tail, log.p`	logical, see, e.g., `pchisq()`.
`epsS`	small number, determining where to switch from the "small case" to the regular case, namely by defining `small <-sqrt(q/ncp) <= epsS`.

Details

In the "small case" (`epsS` above), the direct formulas suffer from cancellation, and we use Taylor series expansions in $s := \sqrt{q}$, which in turn use "probabilists"' Hermite polynomials $He_n(x)$.

The default values `epsS` have currently been determined by experiments as those in the 'Examples' below.

Value

a numeric vector "like" q+ncp, i.e., recycled to common length.

Author(s)

Martin Maechler, notably the Taylor approximations in the "small" cases.

References

Johnson et al.(1995), see 'References' in pnchisqPearson.

https://en.wikipedia.org/wiki/Hermite_polynomials

See Also

pchisq, the (simple and R-like) approximations, such as pnchisqPearson and the wienergerm approximations, pchisqW() etc.

Examples

```
qq <- seq(9500, 10500, length=1000)
m1 <- cbind(pch = pchisq  (qq, df=1, ncp = 10000),
            p1  = pnchi1sq(qq,       ncp = 10000))
matplot(qq, m1, type = "l"); abline(h=0:1, v=10000+1, lty=3)
all.equal(m1[,"p1"], m1[,"pch"], tol=0) # for now,  2.37e-12

m3 <- cbind(pch = pchisq  (qq, df=3, ncp = 10000),
             p3 = pnchi3sq(qq,       ncp = 10000))
matplot(qq, m3, type = "l"); abline(h=0:1, v=10000+3, lty=3)
all.equal(m3[,"p3"], m3[,"pch"], tol=0) # for now,  1.88e-12

stopifnot(exprs = {
  all.equal(m1[,"p1"], m1[,"pch"], tol=1e-10)
  all.equal(m3[,"p3"], m3[,"pch"], tol=1e-10)
})

### Very small 'x' i.e., 'q' would lead to cancellation: -----------

##  df = 1 --------------

qS <- c(0, 2^seq(-40,4, by=1/16))
m1s <- cbind(pch = pchisq  (qS, df=1, ncp = 1)
           , p1.0= pnchi1sq(qS,       ncp = 1, epsS = 0)
           , p1.4= pnchi1sq(qS,       ncp = 1, epsS = 1e-4)
           , p1.3= pnchi1sq(qS,       ncp = 1, epsS = 1e-3)
           , p1.2= pnchi1sq(qS,       ncp = 1, epsS = 1e-2)
        )
cols <- adjustcolor(1:5, 1/2); lws <- seq(4,2, by = -1/2)
abl.leg <- function(x.leg = "topright", epsS = 10^-(4:2), legend = NULL)
{
   abline(h = .Machine$double.eps, v = epsS^2,
          lty = c(2,3,3,3), col= adjustcolor(1, 1/2))
   if(is.null(legend))
     legend <- c(quote(epsS == 0), as.expression(lapply(epsS,
                              function(K) substitute(epsS == KK,
                                                     list(KK = formatC(K, w=1))))))
   legend(x.leg, legend, lty=1:4, col=cols, lwd=lws, bty="n")
```

```
}
matplot(qS, m1s, type = "l", log="y" , col=cols, lwd=lws)
matplot(qS, m1s, type = "l", log="xy", col=cols, lwd=lws) ; abl.leg("right")
## ====  "Errors" ===================================================
## Absolute: -------------------------
matplot(qS,     m1s[,1] - m1s[,-1] , type = "l", log="x" , col=cols, lwd=lws)
matplot(qS, abs(m1s[,1] - m1s[,-1]), type = "l", log="xy", col=cols, lwd=lws)
abl.leg("bottomright")
## Relative: -------------------------
matplot(qS,     1 - m1s[,-1]/m1s[,1] , type = "l", log="x",  col=cols, lwd=lws)
abl.leg()
matplot(qS, abs(1 - m1s[,-1]/m1s[,1]), type = "l", log="xy", col=cols, lwd=lws)
abl.leg()


##  df = 3 --------------  %% FIXME:  the 'small' case is clearly wrong <<<

qS <- c(0, 2^seq(-40,4, by=1/16))
ee <- c(1e-3, 1e-2, .04)
m3s <- cbind(pch = pchisq  (qS, df=3, ncp = 1)
           , p1.0= pnchi3sq(qS,       ncp = 1, epsS = 0)
           , p1.3= pnchi3sq(qS,       ncp = 1, epsS = ee[1])
           , p1.2= pnchi3sq(qS,       ncp = 1, epsS = ee[2])
           , p1.1= pnchi3sq(qS,       ncp = 1, epsS = ee[3])
        )
matplot(qS, m3s, type = "l", log="y" , col=cols, lwd=lws)
matplot(qS, m3s, type = "l", log="xy", col=cols, lwd=lws); abl.leg("right", ee)
## ====  "Errors" ===================================================
## Absolute: -------------------------
matplot(qS,     m3s[,1] - m3s[,-1] , type = "l", log="x" , col=cols, lwd=lws)
matplot(qS, abs(m3s[,1] - m3s[,-1]), type = "l", log="xy", col=cols, lwd=lws)
abl.leg("right", ee)
## Relative: -------------------------
matplot(qS,     1 - m3s[,-1]/m3s[,1] , type = "l", log="x",  col=cols, lwd=lws)
abl.leg(, ee)
matplot(qS, abs(1 - m3s[,-1]/m3s[,1]), type = "l", log="xy", col=cols, lwd=lws)
abl.leg(, ee)
```

---

pnchisqAppr *(Approximate) Probabilities of Non-Central Chi-squared Distribution*

Description

Compute (approximate) probabilities for the non-central chi-squared distribution.

The non-central chi-squared distribution with df$= n$ degrees of freedom and non-centrality parameter ncp $= \lambda$ has density

$$f(x) = f_{n,\lambda}(x) = e^{-\lambda/2} \sum_{r=0}^{\infty} \frac{(\lambda/2)^r}{r!} f_{n+2r}(x)$$

for $x \geq 0$; for more, see R's help page for pchisq.

- R's own historical and current versions, but with more tuning parameters;

Historical relatively simple approximations listed in Johnson, Kotz, and Balakrishnan (1995):

- Patnaik(1949)'s approximation to the non-central via central chi-squared. Is also the formula 26.4.27 in Abramowitz & Stegun, p.942. Johnson et al mention that the approxmation error is $O(1/\sqrt{(}\lambda))$ *for* $\lambda \to \infty$.
- Pearson(1959) is using 3 moments instead of 2 as Patnaik (to approximate via a central chi-squared), and therefore better than Patnaik for the right tail; further (in Johnson et al.), the approxmation error is $O(1/\lambda)$ *for* $\lambda \to \infty$.
- Abdel-Aty(1954)'s "first approximation" based on Wilson-Hilferty via Gaussian (`pnorm`) probabilities, is partly *wrongly* cited in Johnson et al., p.463, eq.$(29.61a)$.
- Bol'shev and Kuznetzov (1963) concentrate on the case of **small** `ncp` λ and provide an "approximation" via *central* chi-squared with the same degrees of freedom `df`, but a modified `q` ('x'); the approximation has error $O(\lambda^3)$ for $\lambda \to 0$ and is from Johnson et al., p.465, eq.(29.62) and (29.63).
- Sankaran(1959, 1963) proposes several further approximations base on Gaussian probabilities, according to Johnson et al., p.463. `pnchisqSankaran_d()` implements its formula $(29.61d)$.

`pnchisq()`: an R implementation of R's own C `pnchisq_raw()`, but almost only up to Feb.27, 2004, long before the `log.p=TRUE` addition there, including *logspace arithmetic* in April 2014, its finish on 2015-09-01. Currently for historical reference only.

`pnchisqV()`: a `Vectorize()`d `pnchisq`.

`pnchisqRC()`: R's C implementation as of Aug.2019; but with many more options. Currently extreme cases tend to hang on Winbuilder (?)

`pnchisqIT`: ....

`pnchisqTerms`: ....

`pnchisqT93`: pure R implementations of approximations when both `q` and `ncp` are large, by Temme(1993), from Johnson et al., p.467, formulas $(29.71a)$, and $(29.71b)$, using auxiliary functions `pnchisqT93a()` and `pnchisqT93b()` respectively, with adapted formulas for the `log.p=TRUE` cases.

`pnchisq_ss()`: ....

`ss`: ....

`ss2`: ....

`ss2.`: ....

Usage

```
pnchisq          (q, df, ncp = 0, lower.tail = TRUE,
                  cutOffncp = 80, itSimple = 110, errmax = 1e-12, reltol = 1e-11,
                   maxit = 10* 10000, verbose = 0, xLrg.sigma = 5)
pnchisqV(x, ..., verbose = 0)

pnchisqRC        (q, df, ncp = 0, lower.tail = TRUE, log.p = FALSE,
                   no2nd.call = FALSE,
                   cutOffncp = 80, small.ncp.logspace = small.ncp.logspaceR2015,
                   itSimple = 110, errmax = 1e-12,
                  reltol = 8 * .Machine$double.eps, epsS = reltol/2, maxit = 1e6,
                   verbose = FALSE)
```

```
pnchisqAbdelAty  (q, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)
pnchisqBolKuz    (q, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)
pnchisqPatnaik   (q, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)
pnchisqPearson   (q, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)
pnchisqSankaran_d(q, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)
pnchisq_ss      (x, df, ncp = 0, lower.tail = TRUE, log.p = FALSE, i.max = 10000)
pnchisqTerms     (x, df, ncp,     lower.tail = TRUE, i.max = 1000)

pnchisqT93  (q, df, ncp, lower.tail = TRUE, log.p = FALSE, use.a = q > ncp)
pnchisqT93.a(q, df, ncp, lower.tail = TRUE, log.p = FALSE)
pnchisqT93.b(q, df, ncp, lower.tail = TRUE, log.p = FALSE)

ss   (x, df, ncp, i.max = 10000, useLv = !(expMin < -lambda && 1/lambda < expMax))
ss2  (x, df, ncp, i.max = 10000, eps = .Machine$double.eps)
ss2. (q, df, ncp = 0, errmax = 1e-12, reltol = 2 * .Machine$double.eps,
      maxit = 1e+05, eps = reltol, verbose = FALSE)
```

Arguments

x	numeric vector (of 'quantiles', i.e., abscissa values).
q	number ('quantile', i.e., abscissa value.)
df	degrees of freedom > 0, maybe non-integer.
ncp	non-centrality parameter δ; ....
lower.tail, log.p	logical, see, e.g., pchisq().
i.max	number of terms in evaluation ...
use.a	logical vector for Temme pnchisqT93*() formulas, indicating to use formula 'a' over 'b'. The default is as recommended in the references, but they did not take into account log.p = TRUE situations.
cutOffncp	a positive number, the cutoff value for ncp...
itSimple	...
errmax	absolute error tolerance.
reltol	convergence tolerance for *rel*ative error.
maxit	maximal number of iterations.
xLrg.sigma	positive number ...
no2nd.call	logical indicating if a 2nd call is made to the internal function ....
small.ncp.logspace	logical vector or function, indicating if the logspace computations for "small" ncp (defined to fulfill ncp < cutOffncp !).
epsS	small positive number, the convergence tolerance of the 'simple' iterations...
verbose	logical or integer specifying if or how much the algorithm progress should be monitored.
...	further arguments passed from pnchisqV() to pnchisq().
useLv	logical indicating if logarithmic scale should be used for λ computations.
eps	convergence tolerance, a positive number.

Details

`pnchisq_ss()` uses `si <-ss(x,df,..)` to get the series terms, and returns `2*dchisq(x,df = df +2) * sum(si$s)`.

`ss()` computes the terms needed for the expansion used in `pnchisq_ss()`.

`ss2()` computes some simple "statistics" about `ss(..)`.

Value

`ss()` returns a list with 3 components

`s`	the series
`i1`	location (in `s[]`) of the first change from 0 to positive.
`max`	(first) location of the maximal value in the series (i.e., `which.max(s)`).

Author(s)

Martin Maechler, from May 1999; starting from a post to the S-news mailing list by Ranjan Maitra (@ math.umbc.edu) who showed a version of our `pchisqAppr.0()` thanking Jim Stapleton for providing it.

References

Johnson, N.L., Kotz, S. and Balakrishnan, N. (1995) Continuous Univariate Distributions Vol~2, 2nd ed.; Wiley.
Chapter 29 *Noncentral χ^2-Distributions*; notably Section *8 Approximations*, p.461 ff.

Abramowitz, M. and Stegun, I. A. (1972) *Handbook of Mathematical Functions*. New York: Dover.
https://en.wikipedia.org/wiki/Abramowitz_and_Stegun

See Also

`pchisq` and the wienergerm approximations for it: `pchisqW()` etc.

`r_pois()` and its plot function, for an aspect of the series approximations we use in `pnchisq_ss()`.

Examples

```
## set of quantiles to use :
qq <- c(.001, .005, .01, .05, (1:9)/10, 2^seq(0, 10, by= 0.5))
## Take "all interesting" pchisq-approximation from our pkg :
pkg <- "package:DPQ"
pnchNms <- c(paste0("pchisq", c("V", "W", "W.", "W.R")),
             ls(pkg, pattern = "^pnchisq"))
pnchNms <- pnchNms[!grepl("Terms$", pnchNms)]
pnchF <- sapply(pnchNms, get, envir = as.environment(pkg))
str(pnchF)
ncps <- c(0, 1/8, 1/2)
pnchR <- as.list(setNames(ncps, paste("ncp",ncps, sep="=")))
for(i.n in seq_along(ncps)) {
  ncp <- ncps[i.n]
  pnF <- if(ncp == 0) pnchF[!grepl("chisqT93", pnchNms)] else pnchF
  pnchR[[i.n]] <- sapply(pnF, function(F)
            Vectorize(F, names(formals(F))[[1]])(qq, df = 3, ncp=ncp))
}
str(pnchR, max=2)
```

```
## A case where the non-central P[] should be improved :
## First, the central P[] which is close to exact -- choosing df=2 allows
## truly exact values: chi^2 = Exp(1) !
opal <- palette()
palette(c("black", "red", "green3", "blue", "cyan", "magenta", "gold3", "gray44"))
cR  <- curve(pchisq   (x, df=2,        lower.tail=FALSE, log.p=TRUE), 0, 4000, n=2001)
cRC <- curve(pnchisqRC(x, df=2, ncp=0, lower.tail=FALSE, log.p=TRUE),
             add=TRUE, col=adjustcolor(2,1/2), lwd=3, lty=2, n=2001)
cR0 <- curve(pchisq   (x, df=2, ncp=0, lower.tail=FALSE, log.p=TRUE),
             add=TRUE, col=adjustcolor(3,1/2), lwd=4,        n=2001)
## smart "named list" constructur :
list_ <- function(...)
   `names<-`(list(...), vapply(sys.call()[-1L], as.character, ""))
JKBfn <-list_(pnchisqPatnaik,
              pnchisqPearson,
              pnchisqAbdelAty,
              pnchisqBolKuz,
              pnchisqSankaran_d)
cl. <- setNames(adjustcolor(3+seq_along(JKBfn), 1/2), names(JKBfn))
lw. <- setNames(2+seq_along(JKBfn),                   names(JKBfn))
cR.JKB <- sapply(names(JKBfn), function(nmf) {
  curve(JKBfn[[nmf]](x, df=2, ncp=0, lower.tail=FALSE, log.p=TRUE),
        add=TRUE, col=cl.[[nmf]], lwd=lw.[[nmf]], lty=lw.[[nmf]], n=2001)
})
legend("bottomleft", c("pchisq", "pchisq.ncp=0", "pnchisqRC", names(JKBfn)),
       col=c(palette()[1], adjustcolor(2:3,1/2), cl.),
       lwd=c(1,3,4, lw.), lty=c(1,2,1, lw.))
palette(opal)# revert

all.equal(cRC, cR0, tol = 1e-15) # TRUE [for now]
## the problematic "jump" :
as.data.frame(cRC)[744:750,]
if(.Platform$OS.type == "unix")
  ## verbose=TRUE  may reveal which branches of the algorithm are taken:
  pnchisqRC(1500, df=2, ncp=0, lower.tail=FALSE, log.p=TRUE, verbose=TRUE) #
  ## |-->  -Inf currently

## The *two*  principal cases (both lower.tail = {TRUE,FALSE} !), where
##  "2nd call"  happens *and* is currently beneficial :
dfs <- c(1:2, 5, 10, 20)
pL. <- pnchisqRC(.00001, df=dfs, ncp=0, log.p=TRUE, lower.tail=FALSE, verbose = TRUE)
pR. <- pnchisqRC(   100, df=dfs, ncp=0, log.p=TRUE,                   verbose = TRUE)
## R's own non-central version (specifying 'ncp'):
pL0 <- pchisq   (.00001, df=dfs, ncp=0, log.p=TRUE, lower.tail=FALSE)
pR0 <- pchisq   (   100, df=dfs, ncp=0, log.p=TRUE)
## R's *central* version, i.e., *not* specifying 'ncp' :
pL  <- pchisq   (.00001, df=dfs,        log.p=TRUE, lower.tail=FALSE)
pR  <- pchisq   (   100, df=dfs,        log.p=TRUE)
cbind(pL., pL, relEc = signif(1-pL./pL, 3), relE0 = signif(1-pL./pL0, 3))
cbind(pR., pR, relEc = signif(1-pR./pR, 3), relE0 = signif(1-pR./pR0, 3))
```

---

pnchisqWienergerm *Wienergerm Approximations to (Non-Central) Chi-squared Probabilities*

Description

Functions implementing the two Wiener germ approximations to `pchisq()`, the (non-central) chi-squared distribution, and to `qchisq()` its inverse, the quantile function.

These have been proposed by Penev and Raykov (2000) who also listed a Fortran implementation.

In order to use them in numeric boundary cases, Martin Maechler has improved the original formulas.

Auxiliary functions:

`sW()`: The $s()$ as in the Wienergerm approximation, but using Taylor expansion when needed, i.e., `(x*ncp / df^2) << 1`.

`qs()`: ...

`z0()`: ...

`z.f()`: ...

`z.s()`: ...

....

Usage

```
pchisqW. (q, df, ncp = 0, lower.tail = TRUE, log.p = FALSE,
          Fortran = TRUE, variant = c("s", "f"))
pchisqV  (q, df, ncp = 0, lower.tail = TRUE, log.p = FALSE,
          Fortran = TRUE, variant = c("s", "f"))
pchisqW  (q, df, ncp = 0, lower.tail = TRUE, log.p = FALSE, variant = c("s", "f"))
pchisqW.R(x, df, ncp = 0, lower.tail = TRUE, log.p = FALSE, variant = c("s", "f"),
          verbose = getOption("verbose"))

sW(x, df, ncp)
qs(x, df, ncp, f.s = sW(x, df, ncp), eps1 = 1/2, sMax = 1e+100)
z0(x, df, ncp)
z.f(x, df, ncp)
z.s(x, df, ncp, verbose = getOption("verbose"))
```

Arguments

`q,x`	vector of quantiles (main argument, see `pchisq`).
`df`	degrees of freedom (non-negative, but can be non-integer).
`ncp`	non-centrality parameter (non-negative).
`lower.tail,log.p`	`logical`, see `pchisq`.
`variant`	a `character` string, currently either `"f"` for the **f**irst or `"s"` for the **s**econd Wienergerm approximation in Penev \& Raykov (2000).
`Fortran`	logical specifying if the Fortran or the C version should be used.
`verbose`	logical (or integer) indicating if or how much diagnostic output should be printed to the console during the computations.
`f.s`	a number must be a "version" of $s(x, df, ncp)$.
`eps1`	for `qs()`: use direct approximation instead of `h(1 -1/s)` for `s < eps1`.
`sMax`	for `qs()`: cutoff to switch the $h(.)$ formula for `s > sMax`.

Details

....TODO... or write vignette

Value

all these functions return `numeric` vectors according to their arguments.

Note

The exact auxiliary function names etc, are still considered *provisional*; currently they are exported for easier documentation and use, but may well all disappear from the exported functions or even completely.

Author(s)

Martin Maechler, mostly end of Jan 2004

References

Penev, Spiridon and Raykov, Tenko (2000) A Wiener Germ approximation of the noncentral chi square distribution and of its quantiles. *Computational Statistics* **15**, 219–228. doi: 10.1007/s001800000029

Dinges, H. (1989) Special cases of second order Wiener germ approximations. *Probability Theory and Related Fields*, **83**, 5–57.

See Also

`pchisq`, and other approximations for it: `pnchisq()` etc.

Examples

```
## see  example(pnchisqAppr)   which looks at all of the pchisq() approximating functions
```

`pnormAsymp` *Asymptotic Approxmation of (Extreme Tail) 'pnorm()'*

Description

Provide the first few terms of the asymptotic series approximation to `pnorm()`'s (extreme) tail, from Abramawitz and Stegun's 26.2.13 (p.932).

Usage

```
pnormAsymp(x, k, lower.tail = FALSE, log.p = FALSE)
```

Arguments

x	positive (at least non-negative) numeric vector.
lower.tail, log.p	logical, see, e.g., pnorm().
k	integer ≥ 0 indicating how many terms the approximation should use; currently $k \leq 5$.

Value

a numeric vector "as" x; see the examples, on how to use it with arbitrary precise mpfr-numbers from package **Rmpfr**.

Author(s)

Martin Maechler

References

Abramowitz, M. and Stegun, I. A. (1972) *Handbook of Mathematical Functions*. New York: Dover. https://en.wikipedia.org/wiki/Abramowitz_and_Stegun provides links to the full text which is in public domain.

See Also

pnormU_S53 for (also asymptotic) upper and lower bounds.

Examples

```
x <- c((2:10)*2, 25, (3:9)*10, (1:9)*100, (1:8)*1000, (2:4)*5000)
Px <- pnorm(x, lower.tail = FALSE, log.p=TRUE)
PxA <- sapply(setNames(0:5, paste("k =",0:5)),
              pnormAsymp, x=x, lower.tail = FALSE, log.p=TRUE)
## rel.errors :
signif(head( cbind(x, 1 - PxA/Px) , 20))

## Look more closely with high precision computations
if(requireNamespace("Rmpfr")) {
  ## ensure our function uses Rmpfr's dnorm(), etc:
  environment(pnormAsymp) <- asNamespace("Rmpfr")
  environment(pnormU_S53) <- asNamespace("Rmpfr")
  x. <- Rmpfr::mpfr(x, precBits=256)
  Px. <- Rmpfr::pnorm(x., lower.tail = FALSE, log.p=TRUE)
  ## manual, better sapplyMpfr():
  PxA. <- sapply(setNames(0:5, paste("k =",0:5)),
                 pnormAsymp, x=x., lower.tail = FALSE, log.p=TRUE)
  PxA. <- new("mpfrMatrix", unlist(PxA.), Dim=dim(PxA.), Dimnames=dimnames(PxA.))
  PxA2 <- Rmpfr::cbind(pn_dbl = Px, PxA.,
                       pnormU53 = pnormU_S53(x=x., lower.tail = FALSE, log.p=TRUE))
  ## rel.errors :
  print( Rmpfr::roundMpfr(Rmpfr::cbind(x., 1 - PxA2/Px.), precBits = 13) )
  pch <- c("R", 0:5, "U")
  matplot(x, abs(1 -PxA2/Px.), type="o", log="xy", pch=pch,
          main="pnorm(<tail>) approximations' relative errors")
  legend("bottomleft", colnames(PxA2), col=1:6, pch=pch, lty=1:5, bty="n", inset=.01)
  at1 <- axTicks(1, axp=c(par("xaxp")[1:2], 3))
```

```
  axis(1, at=at1)
  abline(h = 1:2* 2^-53, v = at1, lty=3, col=adjustcolor("gray20", 1/2))
  axis(4, las=2, at= 2^-53, label = quote(epsilon[C]), col="gray20")
}
```

---

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 vector.
`lower.tail, log.p`	logical, see, e.g., `pnorm()`.

Value

a numeric vector 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`.

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)
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))
col4 <- adjustcolor(1:4, 1/2)
doLegTit <- function() {
  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=col4, 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))

## 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:
x <- - 2^seq(0,    30, by = 1/64)
##   ==
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))
```

---

pnt *Non-central t Probability Distribution - Algorithms and Approximations*

Description

Compute different approximations for the non-central t-Distribution cumulative probability distribution function.

Usage

```
pntR      (t, df, ncp, lower.tail = TRUE, log.p = FALSE,
           use.pnorm = (df > 4e5 ||
                        ncp^2 > 2*log(2)*(-.Machine$double.min.exp)),
                                  itrmax = 1000, errmax = 1e-12, verbose = TRUE)
pntR1     (t, df, ncp, lower.tail = TRUE, log.p = FALSE,
           use.pnorm = (df > 4e5 ||
                        ncp^2 > 2*log(2)*(-.Machine$double.min.exp)),
                                  itrmax = 1000, errmax = 1e-12, verbose = TRUE)

pntP94    (t, df, ncp, lower.tail = TRUE, log.p = FALSE,
                                  itrmax = 1000, errmax = 1e-12, verbose = TRUE)
pntP94.1  (t, df, ncp, lower.tail = TRUE, log.p = FALSE,
                                  itrmax = 1000, errmax = 1e-12, verbose = TRUE)

pnt3150   (t, df, ncp, lower.tail = TRUE, log.p = FALSE, M = 1000, verbose = TRUE)
pnt3150.1 (t, df, ncp, lower.tail = TRUE, log.p = FALSE, M = 1000, verbose = TRUE)

pntLrg    (t, df, ncp, lower.tail = TRUE, log.p = FALSE)

pntJW39   (t, df, ncp, lower.tail = TRUE, log.p = FALSE)
pntJW39.0 (t, df, ncp, lower.tail = TRUE, log.p = FALSE)
```

Arguments

t	vector of quantiles (called q in pt(..)).
df	degrees of freedom (> 0, maybe non-integer). df = Inf is allowed.
ncp	non-centrality parameter $\delta \geq 0$; If omitted, use the central t distribution.
log, log.p	logical; if TRUE, probabilities p are given as log(p).
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.
use.pnorm	logical indicating if the pnorm() approximation of Abramowitz and Stegun (26.7.10) should be used, which is available as pntLrg(). The default corresponds to R pt()'s own behaviour (which is most probably suboptimal).
itrmax	number of iterations / terms.
errmax	convergence bound for the iterations.
verbose	logical or integer determining the amount of diagnostic print out to the console.
M	positive integer specifying the number of terms to use in the series.

Details

pntR1(): a pure R version of the (C level) code of R's own pt(), additionally giving more flexibility (via arguments use.pnorm, itrmax, errmax whose defaults here have been hard-coded in R's C code).

This implements an improved version of the AS 243 algorithm from Lenth(1989);

R's help on non-central pt() says: *This computes the lower tail only, so the upper tail suffers from cancellation and a warning will be given when this is likely to be significant.*

and (in 'Note:') *The code for non-zero* ncp *is principally intended to be used for moderate values of* ncp*: it will not be highly accurate, especially in the tails, for large values.*

pntR(): the Vectorize()d version of pntR1().

pntP94(), pntP94.1(): New versions of pntR1(), pntR(); using the Posten (1994) algorithm. pntP94() is the Vectorize()d version of pntP94.1().

pnt3150(), pnt3150.1(): Simple inefficient but hopefully correct version of pntP94..() This is really a direct implementation of formula (31.50), p.532 of Johnson, Kotz and Balakrishnan (1995)

pntLrg(): provides the pnorm() approximation (to the non-central t) from Abramowitz and Stegun (26.7.10), p.949; which should be employed only for *large* df and/or ncp.

pntJW39.0(): use the Jennett & Welch (1939) approximation see Johnson et al. (1995), p. 520, after (31.26a). This is still *fast* for huge ncp but has *wrong* asymptotic tail for $|t| \to \infty$. Crucially needs $b =$b_chi(df).

pntJW39(): is an improved version of pntJW39.0(), using $1 - b =$b_chi(df,one.minus=TRUE) to avoid cancellation when computing $1 - b^2$.

Value

a number for `pntJKBf1()` and `.pntJKBch1()`.

a numeric vector of the same length as the maximum of the lengths of `x,df,ncp` for `pntJKBf()` and `.pntJKBch()`.

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

Lenth, R. V. (1989). *Algorithm AS 243* — Cumulative distribution function of the non-central t distribution, *Applied Statistics* **38**, 185–189.

Abramowitz, M. and Stegun, I. A. (1972) *Handbook of Mathematical Functions*. New York: Dover. Formula (26.7.10), p.949

See Also

`pt`, for R's version of non-central t probabilities.

Examples

```
tt <- seq(0, 10, len = 21)
ncp <- seq(0, 6, len = 31)
dt3R   <- outer(tt, ncp, pt, , df = 3)
dt3JKB <- outer(tt, ncp, pntR, df = 3)# currently verbose
stopifnot(all.equal(dt3R, dt3JKB, tolerance = 4e-15))# 64-bit Lnx: 2.78e-16
```

---

ppoisson *Direct Computation of 'ppois()' Poisson Distribution Probabilities*

Description

Direct computation and errors of `ppois` Poisson distribution probabilities.

Usage

```
ppoisD(q, lambda, all.from.0 = TRUE, verbose = 0L)
ppoisErr (lambda, ppFUN = ppoisD, iP = 1e-15,
          xM = qpois(iP, lambda=lambda, lower.tail=FALSE),
          verbose = FALSE)
```

Arguments

q	numeric vector of non-negative integer values, "quantiles" at which to evaluate ppois(q,la) and ppFUN(q,la).
lambda	positive parameter of the Poisson distribution, lambda$= \lambda = E[X] = Var[X]$ where $X \sim Pois(\lambda)$.
all.from.0	logical indicating if q is positive integer, and the probabilities should computed for all quantile values of 0:q.
ppFUN	alternative ppois evaluation, by default the **d**irect summation of dpois(k,lambda).
iP	small number, iP << 1, used to construct the abscissa values x at which to evaluate and compare ppois() and ppFUN(), see xM:
xM	(specified instead of iP:) the maximal x-value to be used, i.e., the values used will be x <-0:iM. The default, qpois(1-iP,lambda = lambda) is the upper tail iP-quantile of Poi(lambda).
verbose	integer (≥ 0) or logical indicating if extra information should be printed.

Value

ppoisD() contains the poisson probabilities along q, i.e., is a numeric vector of length length(q).

re <-ppoisErr() returns the relative "error" of ppois(x0,lambda) where ppFUN(x0,lambda) is assumed to be the truth and x0 the "worst case", i.e., the value (among x) with the largest such difference.

Additionally, attr(re,"x0") contains that value x0.

Author(s)

Martin Maechler, March 2004; 2019 ff

See Also

ppois

Examples

```
(lams <- outer(c(1,2,5), 10^(0:3)))# 10^4 is already slow!
system.time(e1 <- sapply(lams, ppoisErr))
e1 / .Machine$double.eps

## Try another 'ppFUN' :---------------------------------
## this relies on the fact that it's *only* used on an 'x' of the form  0:M :
ppD0 <- function(x, lambda, all.from.0=TRUE)
            cumsum(dpois(if(all.from.0) 0:x else x, lambda=lambda))
## and test it:
p0 <- ppD0 (  1000, lambda=10)
p1 <- ppois(0:1000, lambda=10)
stopifnot(all.equal(p0,p1, tol=8*.Machine$double.eps))

system.time(p0.slow <- ppoisD(0:1000, lambda=10, all.from.0=FALSE))# not very slow, here
p0.1 <- ppoisD(1000, lambda=10)
if(requireNamespace("Rmpfr")) {
 ppoisMpfr <- function(x, lambda) cumsum(Rmpfr::dpois(x, lambda=lambda))
 p0.best <- ppoisMpfr(0:1000, lambda = Rmpfr::mpfr(10, precBits = 256))
 AllEq. <- Rmpfr::all.equal
```

```
 AllEq <- function(target, current, ...)
    AllEq.(target, current, ...,
           formatFUN = function(x, ...) Rmpfr::format(x, digits = 9))
 print(AllEq(p0.best, p0,      tol = 0)) # 2.06e-18
 print(AllEq(p0.best, p0.slow, tol = 0)) # the "worst" (4.44e-17)
 print(AllEq(p0.best, p0.1,    tol = 0)) # 1.08e-18
}

## Now (with 'all.from.0 = TRUE',  it is fast too):
p15    <- ppoisErr(2^13)
p15.0. <- ppoisErr(2^13, ppFUN = ppD0)
c(p15, p15.0.) / .Machine$double.eps # on Lnx 64b, see (-10  2.5), then (-2 -2)

## lapply(), so you see "x0" values :
str(e0. <- lapply(lams, ppoisErr, ppFUN = ppD0))

## The first version [called 'err.lambd0()' for years] used simple  cumsum(dpois(..))
## NOTE: It is *stil* much faster, as it relies on special  x == 0:M  relation
## Author: Martin Maechler, Date:  1 Mar 2004, 17:40
##
e0 <- sapply(lams, function(lamb) ppoisErr(lamb, ppFUN = ppD0))
all.equal(e1, e0) # typically TRUE,  though small "random" differences:
cbind(e1, e0) * 2^53 # on Lnx 64b, seeing integer values in {-24, .., 33}
```

---

qbetaAppr *Compute (Approximate) Quantiles of the Beta Distribution*

Description

Compute quantiles (inverse distribution values) for the beta distribution, using diverse approximations.

Usage

```
qbetaAppr.1(a, p, q, y = qnormUappr(a))
qbetaAppr.2(a, p, q, lower.tail=TRUE, log.p=FALSE, logbeta = lbeta(p,q))
qbetaAppr.3(a, p, q, lower.tail=TRUE, log.p=FALSE, logbeta = lbeta(p,q))
qbetaAppr.4(a, p, q, y = qnormUappr(a),
            verbose = getOption("verbose"))

qbetaAppr  (a, p, q, y = qnormUappr(a), logbeta= lbeta(p,q),
            verbose = getOption("verbose") && length(a) == 1)

qbeta.R    (alpha, p, q,
            lower.tail = TRUE, log.p = FALSE,
    logbeta = lbeta(p,q),
    low.bnd = 3e-308, up.bnd = 1-2.22e-16,
            method = c("AS109", "Newton-log"),
            tol.outer = 1e-15,
    f.acu = function(a,p,q) max(1e-300, 10^(-13- 2.5/pp^2 - .5/a^2)),
    fpu = .Machine$ double.xmin,
    qnormU.fun = function(u, lu) qnormUappr(p=u, lp=lu)
```

```
          , R.pre.2014 = FALSE
  , verbose = getOption("verbose")
          , non.finite.report = verbose
           )
```

Arguments

a, alpha	vector of probabilities (otherwise, e.g., in qbeta(), called p).
p, q	the two shape parameters of the beta distribution; otherwise, e.g., in qbeta(), called shape1 and shape2.
y	an approximation to $\Phi^{-1}(1-\alpha)$ (aka $z_{1-\alpha}$) where $\Phi(x)$ is the standard normal cumulative probability function and $\Phi-1(x)$ its inverse, i.e., R's qnorm(x).
lower.tail, log.p	logical, see, e.g., qchisq(); must have length 1.
logbeta	must be lbeta(p,q); mainly an option to pass a value already computed.
verbose	logical or integer indicating if and how much "monitoring" information should be produced by the algorithm.
low.bnd, up.bnd	lower and upper bounds for ...TODO...
method	a string specifying the approximation method to be used.
tol.outer	the "outer loop" convergence tolerance; the default 1e-15 has been hardwired in R's qbeta().
f.acu	a function with arguments (a,p,q) ...TODO...
fpu	a very small positive number.
qnormU.fun	a function with arguments (u,lu) to compute "the same" as qnormUappr(), the upper standard normal quantile.
R.pre.2014	a logical ... TODO ...
non.finite.report	logical indicating if during the "outer loop" refining iterations, if y becomes non finite and the iterations have to stop, it should be reported (before the current best value is returned).

Value

...

Author(s)

The R Core Team for the C version in R's sources; Martin Maechler for the R port.

See Also

qbeta.

Examples

```
qbeta.R(0.6, 2, 3) # 0.4445
qbeta.R(0.6, 2, 3) - qbeta(0.6, 2,3) # almost 0

qbetaRV <- Vectorize(qbeta.R, "alpha") # now can use
curve(qbetaRV(x, 1.5, 2.5))
curve(qbeta  (x, 1.5, 2.5), add=TRUE, lwd = 3, col = adjustcolor("red", 1/2))

## an example of disagreement (and doubt, as borderline, close to underflow):
qbeta.R(0.5078, .01, 5) # ->  2.77558e-15    # but
qbeta  (0.5078, .01, 5) # -> 1.776357e-15  now gives 4.651188e-31 !!!
qbeta  (0.5078, .01, 5, ncp=0) #      also     gives 4.651188e-31
```

---

qbinomR *Pure R Implementation of R's qbinom() with Tuning Parameters*

Description

A pure R implementation, including many tuning parameter arguments, of R's own Mathlib C code algorithm, but with more flexibility.

It is using `Vectorize(qbinomR1,*)` where the hidden `qbinomR1` works for numbers (aka 'scalar', length one) arguments only, the same as the C code.

Usage

```
qbinomR(p, size, prob, lower.tail = TRUE, log.p = FALSE,
        yLarge = 4096, # was hard wired to 1e5
        incF = 1/64,   # was hard wired to .001
        iShrink = 8,   # was hard wired to 100
        relTol = 1e-15,# was hard wired to 1e-15
        pfEps.n = 8,   # was hard wired to 64: "fuzz to ensure left continuity"
        pfEps.L = 2,   # was hard wired to 64:   "   "   ..
        fpf = 4, # *MUST* be >= 1 (did not exist previously)
        trace = 0)
```

Arguments

`p, size, prob, lower.tail, log.p`
: `qbinom()` standard argument, see its help page.

`yLarge`

`incF`

`iShrink`

`relTol`

`pfEps.n`

`pfEps.L`

`fpf`

`trace`
: logical (or integer) specifying if (and how much) output should be produced from the algorithm.

Value

a numeric vector like p recycled to the common lengths of p, size, and prob.

Author(s)

Martin Maechler

See Also

qbinom, qpois.

Examples

```
set.seed(12)
pr <- (0:16)/16 # supposedly recycled
x10 <- rbinom(500, prob=pr, size =  10); p10 <- pbinom(x10, prob=pr, size= 10)
x1c <- rbinom(500, prob=pr, size = 100); p1c <- pbinom(x1c, prob=pr, size=100)
## stopifnot(exprs = {
table( x10  == (qp10  <- qbinom (p10, prob=pr, size= 10) ))
table( qp10 == (qp10R <- qbinomR(p10, prob=pr, size= 10) )); summary(warnings()) # 30 x NaN
table( x1c  == (qp1c  <- qbinom (p1c, prob=pr, size=100) ))
table( qp1c == (qp1cR <- qbinomR(p1c, prob=pr, size=100) )); summary(warnings()) # 30 x NaN
## })
```

---

qchisqAppr *Compute Approximate Quantiles of the Chi-Squared Distribution*

Description

Compute quantiles (inverse distribution values) for the chi-squared distribution. using Johnson,Kotz,.. .....TODO.....

Usage

```
qchisqKG    (p, df, lower.tail = TRUE, log.p = FALSE)
qchisqWH    (p, df, lower.tail = TRUE, log.p = FALSE)
qchisqAppr  (p, df, lower.tail = TRUE, log.p = FALSE, tol = 5e-7)
qchisqAppr.R(p, df, lower.tail = TRUE, log.p = FALSE, tol = 5e-07,
             maxit = 1000, verbose = getOption("verbose"), kind = NULL)
```

Arguments

p	vector of probabilities.
df	degrees of freedom > 0, maybe non-integer; must have length 1.
lower.tail, log.p	logical, see, e.g., qchisq(); must have length 1.
tol	non-negative number, the convergence tolerance
maxit	the maximal number of iterations
verbose	logical indicating if the algorithm should produce "monitoring" information.

kind — the *kind* of approximation; if NULL, the default, the approximation chosen depends on the arguments; notably it is chosen separately for each p. Otherwise, it must be a character string. The main approximations are Wilson-Hilferty versions, when the string contains "WH". More specifically, it must be one of the strings

"chi.small" particularly useful for small chi-squared values p;... ...

"WH" ... ...

"p1WH" ... ...

"WHchk" ... ...

"df.small" particularly useful for small degrees of freedom df... ...

Value

...

Author(s)

Martin Maechler

See Also

qchisq. Further, our approximations to the *non-central* chi-squared quantiles, qnchisqAppr

Examples

```
## TODO
```

---

qgammaAppr *Compute (Approximate) Quantiles of the Gamma Distribution*

Description

Compute approximations to the quantile (i.e., inverse cumulative) function of the Gamma distribution.

Usage

```
qgammaAppr(p, shape, lower.tail = TRUE, log.p = FALSE,
           tol = 5e-07)
qgamma.R  (p, alpha, scale = 1, lower.tail = TRUE, log.p = FALSE,
           EPS1 = 0.01, EPS2 = 5e-07, epsN = 1e-15, maxit = 1000,
           pMin = 1e-100, pMax = (1 - 1e-14),
           verbose = getOption("verbose"))

qgammaApprKG(p, shape, lower.tail = TRUE, log.p = FALSE)


qgammaApprSmallP(p, shape, lower.tail = TRUE, log.p = FALSE)
```

Arguments

p	numeric vector (possibly log tranformed) probabilities.
shape, alpha	shape parameter, non-negative.
scale	scale parameter, non-negative, see qgamma.
lower.tail, log.p	logical, see, e.g., qgamma(); must have length 1.
tol	tolerance of maximal approximation error.
EPS1	small positive number. ...
EPS2	small positive number. ...
epsN	small positive number. ...
maxit	maximal number of iterations. ...
pMin, pMax	boundaries for p. ...
verbose	logical indicating if the algorithm should produce "monitoring" information.

Details

qgammaApprSmallP(p,a) should be a good approximation in the following situation when both p and shape $= \alpha =: a$ are small :

If we look at Abramowitz&Stegun $gamma * (a, x) = x^{-}a * P(a, x)$ and its series $g * (a, x) = 1/gamma(a) * (1/a - 1/(a + 1) * x + ...)$,

then the first order approximation $P(a, x) = x^a * g * (a, x) = x^a/gamma(a + 1)$ and hence its inverse $x = qgamma(p, a) = (p * gamma(a + 1))^(1/a)$ should be good as soon as $1/a >> 1/(a + 1) * x$

<==> x « (a+1)/a = (1 + 1/a)

<==> x < eps *(a+1)/a

<==> log(x) < log(eps) + log((a+1)/a) = log(eps) + log((a+1)/a) ~ -36 - log(a) where log(x) ~= log(p * gamma(a+1)) / a = (log(p) + lgamma1p(a))/a

such that the above

<==> (log(p) + lgamma1p(a))/a < log(eps) + log((a+1)/a)

<==> log(p) + lgamma1p(a) < a*(-log(a)+ log(eps) + log1p(a))

<==> log(p) < a*(-log(a)+ log(eps) + log1p(a)) - lgamma1p(a) =: bnd(a)

Note that qgammaApprSmallP() indeed also builds on lgamma1p().

.qgammaApprBnd(a) provides this bound $bnd(a)$; it is simply a*(logEps + log1p(a) -log(a)) -lgamma1p(a), where logEps is $\log(\epsilon)$ = log(eps) where eps <-.Machine$double.eps, i.e. typically (always?) logEps= $\log \epsilon = -52 * \log(2) = -36.04365$.

Value

numeric

Author(s)

Martin Maechler

References

Abramowitz, M. and Stegun, I. A. (1972) *Handbook of Mathematical Functions*. New York: Dover. https://en.wikipedia.org/wiki/Abramowitz_and_Stegun provides links to the full text which is in public domain.

See Also

qgamma for R's Gamma distribution functions.

Examples

```
## TODO :  Move some of the curve()s from ../tests/qgamma-ex.R !!
```

---

qnbinomR *Pure R Implementation of R's qnbinom() with Tuning Parameters*

Description

A pure R implementation, including many tuning parameter arguments, of R's own Mathlib C code algorithm, but with more flexibility.

It is using `Vectorize(qnbinomR1,*)` where the hidden `qnbinomR1` works for numbers (aka 'scalar', length one) arguments only, the same as the C code.

Usage

```
qnbinomR(p, size, prob, mu, lower.tail = TRUE, log.p = FALSE,
         yLarge = 4096, # was hard wired to 1e5
         incF = 1/64,   # was hard wired to .001
         iShrink = 8,   # was hard wired to 100
         relTol = 1e-15,# was hard wired to 1e-15
        pfEps.n = 8,   # was hard wired to 64: "fuzz to ensure left continuity"
         pfEps.L = 2,   # was hard wired to 64:   "   "   ..
         fpf = 4, # *MUST* be >= 1 (did not exist previously)
         trace = 0)
```

Arguments

`p, size, prob, mu, lower.tail, log.p`	`qnbinom()` standard argument, see its help page.
`yLarge`	
`incF`	
`iShrink`	
`relTol`	
`pfEps.n`	
`pfEps.L`	
`fpf`	
`trace`	logical (or integer) specifying if (and how much) output should be produced from the algorithm.

Value

a numeric vector like p recycled to the common lengths of p, size, and either prob or mu.

Author(s)

Martin Maechler

See Also

qnbinom, qpois.

Examples

```
set.seed(12)
x10 <- rnbinom(500, mu = 4,       size = 10) ; p10 <- pnbinom(x10, mu=4,       size=10)
x1c <- rnbinom(500, prob = 31/32, size = 100); p1c <- pnbinom(x1c, prob=31/32, size=100)
stopifnot(exprs = {
    x10 == qnbinom (p10, mu=4, size=10)
    x10 == qnbinomR(p10, mu=4, size=10)
    x1c == qnbinom (p1c, prob=31/32, size=100)
    x1c == qnbinomR(p1c, prob=31/32, size=100)
})
```

---

qnchisqAppr *Compute Approximate Quantiles of Noncentral Chi-Squared Distribution*

Description

Compute quantiles (inverse distribution values) for the *non-central* chi-squared distribution.

..... using Johnson,Kotz, and other approximations .....

Usage

```
qchisqAppr.0 (p, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)
qchisqAppr.1 (p, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)
qchisqAppr.2 (p, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)
qchisqAppr.3 (p, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)
qchisqApprCF1(p, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)
qchisqApprCF2(p, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)

qchisqCappr.2 (p, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)
qchisqN       (p, df, ncp = 0, qIni = qchisqAppr.0, ...)

qnchisqAbdelAty  (p, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)
qnchisqBolKuz    (p, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)
qnchisqPatnaik   (p, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)
qnchisqPearson   (p, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)
qnchisqSankaran_d(p, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)
```

Arguments

p	vector of probabilities.
df	degrees of freedom > 0, maybe non-integer.
ncp	non-centrality parameter δ; ....
lower.tail, log.p	logical, see, e.g., qchisq().
qIni	a function that computes an approximate noncentral chi-squared quantile as starting value x0 for the Newton algorithm newton().
...	further arguments to newton(), notably eps or maxiter.

Details

Compute (approximate) quantiles, using approximations analogous to those for the probabilities, see pnchisqPearson.

qchisqAppr.0(): ...TODO...

qchisqAppr.1(): ...TODO...

qchisqAppr.2(): ...TODO...

qchisqAppr.3(): ...TODO...

qchisqApprCF1(): ...TODO...

qchisqApprCF2(): ...TODO...

qchisqCappr.2(): ...TODO...

qchisqN(): Uses Newton iterations with pchisq() and dchisq() to determine qchisq(.) values.

qnchisqAbdelAty(): ...TODO...

qnchisqBolKuz(): ...TODO...

qnchisqPatnaik(): ...TODO...

qnchisqPearson(): ...TODO...

qnchisqSankaran_d(): ...TODO...

Value

numeric vectors of (noncentral) chi-squared quantiles, corresponding to probabilities p.

Author(s)

Martin Maechler, from May 1999; starting from a post to the S-news mailing list by Ranjan Maitra (@ math.umbc.edu) who showed a version of our qchisqAppr.0() thanking Jim Stapleton for providing it.

References

Johnson, N.L., Kotz, S. and Balakrishnan, N. (1995) Continuous Univariate Distributions Vol~2, 2nd ed.; Wiley.
Chapter 29 *Noncentral χ^2-Distributions*; notably Section *8 Approximations*, p.461 ff.

See Also

qchisq.

Examples

```
 pp <- c(.001, .005, .01, .05, (1:9)/10, .95, .99, .995, .999)
 pkg <- "package:DPQ"
 qnchNms <- c(paste0("qchisqAppr.",0:3), paste0("qchisqApprCF",1:2),
              "qchisqN", "qchisqCappr.2", ls(pkg, pattern = "^qnchisq"))
 qnchF <- sapply(qnchNms, get, envir = as.environment(pkg))
 for(ncp in c(0, 1/8, 1/2)) {
   cat("\n~~~~~~~~~~~~~~\nncp: ", ncp,"\n=======\n")
   print(sapply(qnchF, function(F) Vectorize(F, "p")(pp, df = 3, ncp=ncp)))
 }

## Bug: qnchisqSankaran_d() has numeric overflow problems for large df:
qnchisqSankaran_d(pp, df=1e200, ncp = 100)

## One current (2019-08) R bug: Noncentral chi-squared quantiles on *LOG SCALE*

## a)  left/lower tail : ------------------------------------------------------
qs <- 2^seq(0,11, by=1/16)
pqL <- pchisq(qs, df=5, ncp=1, log.p=TRUE)
plot(qs, -pqL, type="l", log="xy") # + expected warning on log(0) -- all fine
qpqL <- qchisq(pqL, df=5, ncp=1, log.p=TRUE) # severe overflow :
qm <- cbind(qs, pqL, qchisq=qpqL
, qchA.0 = qchisqAppr.0 (pqL, df=5, ncp=1, log.p=TRUE)
, qchA.1 = qchisqAppr.1 (pqL, df=5, ncp=1, log.p=TRUE)
, qchA.2 = qchisqAppr.2 (pqL, df=5, ncp=1, log.p=TRUE)
, qchA.3 = qchisqAppr.3 (pqL, df=5, ncp=1, log.p=TRUE)
, qchACF1= qchisqApprCF1(pqL, df=5, ncp=1, log.p=TRUE)
, qchACF2= qchisqApprCF2(pqL, df=5, ncp=1, log.p=TRUE)
, qchCa.2= qchisqCappr.2(pqL, df=5, ncp=1, log.p=TRUE)
, qnPatnaik   = qnchisqPatnaik   (pqL, df=5, ncp=1, log.p=TRUE)
, qnAbdelAty  = qnchisqAbdelAty  (pqL, df=5, ncp=1, log.p=TRUE)
, qnBolKuz    = qnchisqBolKuz    (pqL, df=5, ncp=1, log.p=TRUE)
, qnPearson   = qnchisqPearson   (pqL, df=5, ncp=1, log.p=TRUE)
, qnSankaran_d= qnchisqSankaran_d(pqL, df=5, ncp=1, log.p=TRUE)
)

round(qm[ qs %in% 2^(0:11) , -2])
#=> Approximations don't overflow but are not good enough

## b)  right/upper tail , larger ncp ------------------------------------------
qS <- 2^seq(-3, 3, by=1/8)
pqLu <- pchisq(qS, df=5, ncp=100, log.p=TRUE, lower.tail=FALSE)
## using "the alternative" (here is currently identical):
identical(pqLu, (pqLu.<- log1p(-pchisq(qS, df=5, ncp=100)))) # here TRUE
plot (qS, -pqLu, type="l", log="xy") # fine
qpqLu <- qchisq(pqLu, df=5, ncp=100, log.p=TRUE, lower.tail=FALSE)
cbind(qS, pqLu, pqLu, qpqLu)# # severe underflow
qchMat <- cbind(qchisq = qpqLu
, qchA.0 = qchisqAppr.0 (pqLu, df=5, ncp=100, log.p=TRUE, lower.tail=FALSE)
, qchA.1 = qchisqAppr.1 (pqLu, df=5, ncp=100, log.p=TRUE, lower.tail=FALSE)
, qchA.2 = qchisqAppr.2 (pqLu, df=5, ncp=100, log.p=TRUE, lower.tail=FALSE)
, qchA.3 = qchisqAppr.3 (pqLu, df=5, ncp=100, log.p=TRUE, lower.tail=FALSE)
, qchACF1= qchisqApprCF1(pqLu, df=5, ncp=100, log.p=TRUE, lower.tail=FALSE)
, qchACF2= qchisqApprCF2(pqLu, df=5, ncp=100, log.p=TRUE, lower.tail=FALSE)
, qchCa.2= qchisqCappr.2(pqLu, df=5, ncp=100, log.p=TRUE, lower.tail=FALSE)
, qnPatnaik   = qnchisqPatnaik   (pqLu, df=5, ncp=100, log.p=TRUE, lower.tail=FALSE)
```

```
, qnAbdelAty  = qnchisqAbdelAty  (pqLu, df=5, ncp=100, log.p=TRUE, lower.tail=FALSE)
, qnBolKuz    = qnchisqBolKuz    (pqLu, df=5, ncp=100, log.p=TRUE, lower.tail=FALSE)
, qnPearson   = qnchisqPearson   (pqLu, df=5, ncp=100, log.p=TRUE, lower.tail=FALSE)
, qnSankaran_d= qnchisqSankaran_d(pqLu, df=5, ncp=100, log.p=TRUE, lower.tail=FALSE)
)
cbind(L2err <- sort(sqrt(colSums((qchMat - qS)^2))))
##--> "Sankaran_d", "CF1" and "CF2" are good here

plot (qS, qpqLu, type = "b", log="x", lwd=2)
lines(qS, qS, col="gray", lty=2, lwd=3)
top3 <- names(L2err)[1:3]
use <- c("qchisq", top3)
matlines(qS, qchMat[, use]) # 3 of the approximations are "somewhat ok"
legend("topleft", c(use,"True"), bty="n", col=c(palette()[1:4], "gray"),
                  lty = c(1:4,2), lwd = c(2, 1,1,1, 3))
```

---

qnormAppr *Approximations to 'qnorm()', i.e., z_α*

Description

Relatively simple approximations to the standard normal (aka "Gaussian") quantiles, i.e., the inverse of the normal cumulative probability function.

`qnormUappr()` is a simple approximation to (the **u**pper tail) standard normal quantiles, `qnorm()`.

Usage

```
qnormAppr(p)
qnormUappr(p, lp = .DT_Clog(p, lower.tail=lower.tail, log.p=log.p),
           lower.tail = FALSE, log.p = FALSE)
```

Arguments

`p`	numeric vector of probabilities, possibly transformed, depending on `log.p`. Does not need to be specified, if `lp` is instead.
`lp`	`log(1 -p*)`, assuming $p*$ is the `lower.tail=TRUE,log.p=FALSE` version of `p`. If passed as argument, it can be much more accurate than when computed from `p` by default.
`lower.tail`	logical; if TRUE (*not* the default here!), probabilities are $P[X \le x]$, otherwise (by default) upper tail probabilities, $P[X > x]$.
`log.p`	logical; if TRUE, probabilities p are given as $\log(p)$ in argument `p`.

Details

`qnormAppr(p)` uses the simple 4 coefficient rational approximation to `qnorm(p)`, to be used *only* for $p > 1/2$ in `qbeta()` computations, e.g., `qbeta.R`.
The relative error of this approximation is quite *asymmetric*: It is mainly < 0.

`qnormUappr(p)` uses the same rational approximation directly for the **U**pper tail where it is relatively good, and for the lower tail via "swapping the tails", so it is good there as well.

Value

numeric vector of (approximate) normal quantiles corresponding to probabilities p

Author(s)

Martin Maechler

See Also

qnorm.

Examples

```
pp <- c(.001, .005, .01, .05, (1:9)/10, .95, .99, .995, .999)
z_p <- qnorm(pp)
(R <- cbind(pp, z_p, qA = qnormAppr(pp), qUA = qnormUappr(pp, lower.tail=TRUE)))
## Errors, absolute and relative:
cbind(pp, (relE <- cbind(
               errA  = z_p - R[,"qA" ],
               errUA = z_p - R[,"qUA"],
               rE.A  = 1 - R[,"qA" ]/z_p,
               rE.UA = 1 - R[,"qUA"]/z_p)))

lp <- -c(1000, 500, 200, 100, 50, 20:10, seq(9.75, 0, by = -1/8))
qnormUappr(lp=lp) # 'p' need not be specified if 'lp' is

curve(qnorm(x, lower.tail=FALSE), n=1001)
curve(qnormUappr(x), add=TRUE,    n=1001, col = adjustcolor("red", 1/2))

curve(qnorm(x, lower.tail=FALSE) - qnormUappr(x), n=1001)
```

---

qnormR — *Pure R version of* R*'s* qnorm() *with Diagnostics and Tuning Parameters*

Description

Compute's R level implementations of R's qnorm() as implemented in C code (in R's 'mathlib').

Usage

```
qnormR1(p, mu = 0, sd = 1, lower.tail = TRUE, log.p = FALSE, trace = 0, version = )
qnormR (p, mu = 0, sd = 1, lower.tail = TRUE, log.p = FALSE, trace = 0,
        version = c("4.0.x", "2020-10-17"))
```

Arguments

p	probability p, $1-p$, or $\log(p)$, $\log(1-p)$, depending on lower.tail and log.p.
mu	mean of the normal distribution.
sd	standard deviation of the normal distribution.
lower.tail, log.p	logical, see, e.g., qnorm().

`trace`	logical or integer; if positive or `TRUE`, diagnostic output is printed to the console during the computations.
`version`	a `character` string specifying which version or variant is used. The *current* default, `"4.0.x"` is the one used in R versions up to 4.0.x; `"2020-10-17"` is the one committed to the R development sources on 2020-10-17, which prevents the worst for very large $\lvert p \rvert$ when `log.p=TRUE`.

Details

For `qnormR1(p,..)`, `p` must be of length one, whereas `qnormR(p,m,s,..)` works vectorized in `p`, `mu`, and `sd`. In the **DPQ** package source, it simply the result of `Vectorize(qnormR1,...)`.

Value

a numeric vector like the input `q`.

Author(s)

Martin Maechler

See Also

`qnorm`

Examples

```
qR <- curve(qnormR, n = 2^11)
abline(h=0, v=0:1, lty=3, col=adjustcolor(1, 1/2))
with(qR, all.equal(y, qnorm(x), tol=0)) # currently shows TRUE
with(qR, all.equal(pnorm(y), x, tol=0)) # currently: mean rel. diff.: 2e-16
stopifnot(with(qR, all.equal(pnorm(y), x, tol = 1e-14)))

## Showing why/where R's qnorm() was poor up to 2020: log.p=TRUE extreme tail
qs <- 2^seq(0, 155, by=1/8)
lp <- pnorm(qs, lower.tail=FALSE, log.p=TRUE)
## the inverse of pnorm() fails BADLY for extreme tails; this identical to qnorm(..) in R <= 4.0.x:
qp <- qnormR(lp, lower.tail=FALSE, log.p=TRUE, version="4.0.x")
## asymptotically correct approximation :
qpA <- sqrt(- 2* lp)
##^
col2 <- c("black", adjustcolor(2, 0.6))
col3 <- c(col2, adjustcolor(4, 0.6))
## instead of going toward infinity, it converges at  9.834030e+07 :
matplot(-lp, cbind(qs, qp, qpA), type="l", log="xy", lwd = c(1,1,3), col=col3,
        main = "Poorness of qnorm(lp, lower.tail=FALSE, log.p=TRUE)",
        ylab = "qnorm(lp, ..)", axes=FALSE)
sfsmisc::eaxis(1); sfsmisc::eaxis(2)
legend("top", c("truth", "qnorm(.) = qnormR(., \"4.0.x\")", "asymp. approx"),
       lwd=c(1,1,3), lty=1:3, col=col3, bty="n")

rM <- cbind(lp, qs, 1 - cbind(relE.qnorm=qp, relE.approx=qpA)/qs)
rM[ which(1:nrow(rM) %% 20 == 1) ,]
```

qpoisR *Pure R Implementation of R's qpois() with Tuning Parameters*

Description

A pure R implementation, including many tuning parameter arguments, of R's own Mathlib C code algorithm, but with more flexibility.

It is using `Vectorize(qpoisR1,*)` where the hidden `qpoisR1` works for numbers (aka 'scalar', length one) arguments only, the same as the C code.

Usage

```
qpoisR(p, lambda, lower.tail = TRUE, log.p = FALSE,
       yLarge = 4096, # was hard wired to 1e5
       incF = 1/64,   # was hard wired to .001
       iShrink = 8,   # was hard wired to 100
       relTol = 1e-15,# was hard wired to 1e-15
       pfEps.n = 8,   # was hard wired to 64: "fuzz to ensure left continuity"
       pfEps.L = 2,   # was hard wired to 64:   "   "   ..
       fpf = 4, # *MUST* be >= 1 (did not exist previously)
       trace = 0)
```

Arguments

`p, lambda, lower.tail, log.p`
: `qpois()` standard argument, see its help page.

`yLarge`
: a positive number; in R up to 2021, was internally hardwired to `yLarge = 1e5`: Uses more careful search for $y \geq y_L$, where y is the initial approximate result, derived from a Cornish-Fisher expansiion.

`incF`
: a positive "increment factor" (originally hardwired to 0.001), used only when `y >= yLarge`; defines the initial increment in the search algorithm as `incr <-floor(incF * y)`.

`iShrink`
: a positive increment shrinking factor, used only when `y >= yLarge` to define the new increment from the old one as `incr <-max(1,floor(incr/iShrink))` where the LHS was hardired original to `(incr/100)`.

`relTol`
: originally hard wired to 1e-15, defines the convergence tolerance for the search iterations when `y >= yLarge`; the iterations stop when (new) `incr <= y * relTol`.

`pfEps.n, pfEps.L`
: positive factors defining "fuzz to ensure left continuity", both originally hard-wired to 64. originally, the fuzz adjustment was

```
p <- p * (1 - 64 *.Machine$double.eps)
```

Now, `pfEps.L` is used `if(log.p)` is true and `pfEps.n` is used otherwise ("n"ormal case), and the adjustments also depend on `lower.tail`, and also on `fpf` :

`fpf`
: a number larger than 1, together with `pfEps.n` determines the fuzz-adjustment to `p` in the case `(lower=tail=FALSE,log.p=FALSE)`: with `e <-pfEps.n * .Machine$double.eps`, the adjustment `p <-p * (1 + e)` is made *iff* `1 -p > fpf*e`.

`trace`
: logical (or integer) specifying if (and how much) output should be produced from the algorithm.

Details

The defaults and exact meaning of the algorithmic tuning arguments from `yLarge` to `fpf` were experimentally determined are subject to change.

Value

a numeric vector like `p` recycled to the common lengths of `p` and `lambda`.

Author(s)

Martin Maechler

See Also

`qpois`.

Examples

```
x <- 10*(15:25)
Pp <- ppois(x, lambda = 100, lower.tail = FALSE)  # no cancellation
qPp <- qpois(Pp, lambda = 100, lower.tail=FALSE)
table(x == qPp) # all TRUE ?
## future: if(getRversion() >= "4.2") stopifnot(x == qPp) # R-devel
qpRp <- qpoisR(Pp, lambda = 100, lower.tail=FALSE)
all.equal(x, qpRp, tol = 0)
stopifnot(all.equal(x, qpRp, tol = 1e-15))
```

---

qtAppr *Compute Approximate Quantiles of Non-Central t Distribution*

Description

Compute quantiles (inverse distribution values) for the non-central t distribution. using Johnson,Kotz,.. p.521, formula (31.26 a) (31.26 b) & (31.26 c)

Note that `qt(..,ncp=*)` did not exist yet in 1999, when MM implemented `qtAppr()`.

Usage

```
qtAppr(p, df, ncp, lower.tail = TRUE, log.p = FALSE, method = c("a", "b", "c"))
```

Arguments

`p`	vector of probabilities.
`df`	degrees of freedom > 0, maybe non-integer.
`ncp`	non-centrality parameter δ; ....
`lower.tail, log.p`	logical, see, e.g., `qt()`.
`method`	a string specifying the approximation method to be used.

Value

...

Author(s)

Martin Maechler, 6 Feb 1999

See Also

qt.

Examples

```
## TODO
```

---

r_pois *Compute Relative Size of i-th term of Poisson Distribution Series*

Description

Compute

$$r_\lambda(i) := (\lambda^i/i!)/e_{i-1}(\lambda),$$

where $\lambda =$lambda, and

$$e_n(x) := 1 + x + x^2/2! + ... + x^n/n!$$

is the n-th partial sum of $\exp(x) = e^x$.

Questions: As function of i

- Can this be put in a simple formula, or at least be well approximated for large λ and/or large i?
- For which i ($:= i_m(\lambda)$) is it maximal?
- When does $r_\lambda(i)$ become smaller than (f+2i-x)/x = a + b*i ?

NB: This is relevant in computations for non-central chi-squared (and similar non-central distribution functions) defined as weighted sum with "Poisson weights".

Usage

```
r_pois(i, lambda)
r_pois_expr  # the R expression() for the asymptotic branch of r_pois()

plRpois(lambda, iset = 1:(2*lambda), do.main = TRUE,
        log = 'xy', type = "o", cex = 0.4, col = c("red","blue"),
        do.eaxis = TRUE, sub10 = "10")
```

Arguments

i	integer ..
lambda	non-negative number ...
iset	.....
do.main	logical specifying if a main title should be drawn via (main = r_pois_expr).
type	type of (line) plot, see lines.
log	string specifying if (and where) logarithmic scales should be used, see plot.default().

cex	character expansion factor.
col	colors for the two curves.
do.eaxis	logical specifying if eaxis() (package **sfsmisc**) should be used.
sub10	argument for eaxis() (with a different default than the original).

Details

r_pois() is related to our series expansions and approximations for the non-central chi-squared; in particular .....

plRpois() simply produces a "nice" plot of r_pois(ii,*) vs ii.

Value

r_pois() returns a numeric vector $r_\lambda(i)$ values.

r_pois_expr() an expression.

Author(s)

Martin Maechler, 20 Jan 2004

See Also

dpois().

Examples

```
plRpois(12)
plRpois(120)
```

Index