Package ‘DPQ’

December 10, 2021

Title Density, Probability, Quantile (‘DPQ’) Computations

Version 0.5-1

Date 2021-12-10

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 (>= 1.1-10)

Suggests Rmpfr, DPQmpfr (>= 0.3-1), 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

URL https://specfun.r-forge.r-project.org/, https://r-forge.r-project.org/R/?group_id=611

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

R topics documented:

DPQ-package ............................................................... 3
algdiv ................................................................. 6
Bern ................................................................. 7
b_ch ................................................................. 8
dbinom_raw .......................................................... 11
dchisqApprox ......................................................... 12
dgamma-utils .......................................................... 13
dgamma.R ............................................................. 17
dhyperBinMolenaar ...................................................... 18
dbinomR ............................................................. 19
dnt ............................................................. 20
<table>
<thead>
<tr>
<th>R topics documented:</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtWV</td>
<td>22</td>
</tr>
<tr>
<td>format01prec</td>
<td>24</td>
</tr>
<tr>
<td>fr_ld_exp</td>
<td>25</td>
</tr>
<tr>
<td>hyper2binomP</td>
<td>26</td>
</tr>
<tr>
<td>lbeta</td>
<td>27</td>
</tr>
<tr>
<td>lfastchoose</td>
<td>29</td>
</tr>
<tr>
<td>lgamma1p</td>
<td>29</td>
</tr>
<tr>
<td>lgammaAsymp</td>
<td>31</td>
</tr>
<tr>
<td>log1mexp</td>
<td>32</td>
</tr>
<tr>
<td>log1pmx</td>
<td>33</td>
</tr>
<tr>
<td>logcf</td>
<td>35</td>
</tr>
<tr>
<td>logspace.add</td>
<td>36</td>
</tr>
<tr>
<td>lssum</td>
<td>37</td>
</tr>
<tr>
<td>lsum</td>
<td>38</td>
</tr>
<tr>
<td>newton</td>
<td>39</td>
</tr>
<tr>
<td>numer-utils</td>
<td>42</td>
</tr>
<tr>
<td>pII1</td>
<td>44</td>
</tr>
<tr>
<td>pbetaRv1</td>
<td>48</td>
</tr>
<tr>
<td>phyperAllBin</td>
<td>50</td>
</tr>
<tr>
<td>phyperApprAS152</td>
<td>51</td>
</tr>
<tr>
<td>phyperBin</td>
<td>52</td>
</tr>
<tr>
<td>phyperBinMolenaar</td>
<td>53</td>
</tr>
<tr>
<td>phyperIbeta</td>
<td>54</td>
</tr>
<tr>
<td>phyperMolenaar</td>
<td>55</td>
</tr>
<tr>
<td>phyperPeizer</td>
<td>56</td>
</tr>
<tr>
<td>phyperR</td>
<td>58</td>
</tr>
<tr>
<td>phyperR2</td>
<td>59</td>
</tr>
<tr>
<td>phyers</td>
<td>60</td>
</tr>
<tr>
<td>pl2curves</td>
<td>61</td>
</tr>
<tr>
<td>pnbeta</td>
<td>62</td>
</tr>
<tr>
<td>pnchi1sq</td>
<td>64</td>
</tr>
<tr>
<td>pnchisqAppr</td>
<td>67</td>
</tr>
<tr>
<td>pnchisqWienergerm</td>
<td>71</td>
</tr>
<tr>
<td>pnormAsymp</td>
<td>73</td>
</tr>
<tr>
<td>pnormLU</td>
<td>74</td>
</tr>
<tr>
<td>put</td>
<td>76</td>
</tr>
<tr>
<td>ppoisson</td>
<td>79</td>
</tr>
<tr>
<td>qbetaAppr</td>
<td>80</td>
</tr>
<tr>
<td>qbinomR</td>
<td>82</td>
</tr>
<tr>
<td>qchisqAppr</td>
<td>83</td>
</tr>
<tr>
<td>qgammaAppr</td>
<td>85</td>
</tr>
<tr>
<td>qnbinomR</td>
<td>86</td>
</tr>
<tr>
<td>qnchisqAppr</td>
<td>88</td>
</tr>
<tr>
<td>qnormAppr</td>
<td>90</td>
</tr>
<tr>
<td>qnormR</td>
<td>92</td>
</tr>
<tr>
<td>qpoisR</td>
<td>93</td>
</tr>
<tr>
<td>qtAppr</td>
<td>95</td>
</tr>
<tr>
<td>r_pois</td>
<td>96</td>
</tr>
</tbody>
</table>

**Index** 98
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: 0.5-1
Date: 2021-12-10
Authors@R: c(person("Martin", "Maechler", role=c("aut", "cre"), email="maechler@stat.math.ethz.ch", comment = c(ORCID = "0000-0002-8685-9910")), person("R Foundation", role = "cph", comment = "src/qchisq-appr.c") )
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 (>= 1.1-10)
Suggests: Rmpfr, DPQmpfr (>= 0.3-1), 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
URL: https://specfun.r-forge.r-project.org/, https://r-forge.r-project.org/R/?group_id=611
BugReports: https://r-forge.r-project.org/tracker/?atid=2462&group_id=611
Author: Martin Maechler [aut, cre] (<https://orcid.org/0000-0002-8685-9910>), Morten Welinder [ctb] (pgamma C), R-core [ctb] (src/dpq.h, algdiv.c, pnchisq.c, bd0.c), R Foundation [cph] (src/qchisq-appr.c)
Maintainer: Martin Maechler <maechler@stat.math.ethz.ch>

Index of help topics:

Bern
DPQ-package Density, Probability, Quantile ('DPQ') Computations
M_LN2 Numerical Utilities - Functions, Constants
algdiv Compute log(gamma(b)/gamma(a+b)) when b >= 8
b_chi Compute E[chi_nu]/sqrt(nu) useful for t- and chi-Distributions
bd0 Utility Functions for 'dgamma()' - Pure R Versions
dbinom_raw R's C Mathlib (Rmath) dbinom_raw() Binomial Probability pure R Function
dgamma.R Gamma Density Function Alternatives
dhyperBinMolenaar HyperGeometric (Point) Probabilities via Molenaar's Binomial Approximation
dnbincR Pure R Versions of R's C (Mathlib) dnbinom() Negative Binomial Probabilities
Approximations of the (Noncentral) Chi-Squared Density
Non-central t-Distribution Density - Algorithms and Approximations
Noncentral t Distribution Density by W.V.
Format Numbers in [0,1] with "Precise" Result
Base-2 Representation and Multiplication of Numbers
Transform Hypergeometric Distribution Parameters to Binomial Probability
R versions of Simple Formulas for Logarithmic Binomial Coefficients
Accurate 'log(gamma(a+1))' Asymptotic Log Gamma Function
Compute log(1 - exp(-a)) and log(1 + exp(x)) Numerically Optimally
Accurate 'log(1+x) - x' Computation
Continued Fraction Approximation of Log-Related Power Series
Logspace Arithmetix - Addition and Subtraction
Compute Logarithm of a Sum with Signed Large Summands
Properly Compute the Logarithm of a Sum (of Exponentials)
Simple R level Newton Algorithm, Mostly for Didactical Reasons
Numerically Stable p1l1(t) = (t+1)*log(1+t) - t Pure R Implementation of Old pbeta()
Wienergerm Approximations to (Non-Central) Chi-squared Probabilities
Molenaar's Normal Approximations to the Hypergeometric Distribution
Compute Hypergeometric Probabilities via Binomial Approximations
Normal Approximation to cumulative Hyperbolic Distribution - AS 152
HyperGeometric Distribution via Approximate Binomial Distribution
HyperGeometric Distribution via Molenaar's Binomial Approximation
Pearson's incomplete Beta Approximation to the Hyperbolic Distribution
Peizer's Normal Approximation to the Cumulative Hyperbolic
R-only version of R's original phyper() algorithm
Pure R version of R's C level phyper() calls.
The Four (4) Symmetric phyper() calls.
Plot 2 Noncentral Distribution Curves for Visual Comparison
Noncentral Beta Probabilities
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 Computations in 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

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

### Description

Computes\( \text{algdiv}(a, b) := \log \Gamma(b) - \log \Gamma(a+b) = \text{lgamma}(b) - \text{lgamma}(a+b) \) in a numerically stable way.

This is an auxiliary function in R’s (TOMS 708) implementation of \texttt{pbeta()}, aka the incomplete beta function ratio.

### Usage

```r
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,

\[
\log B(a, b) = \log \Gamma(a) + \text{algdiv}(a, b) = \log \Gamma(a) - \log Qab(a, b)
\]

In our ../tests/qbeta-dist.R we look into computing \( \log(p*Beta(p,q)) \) accurately for \( p \ll q \)

---------------------

We are proposing a nice solution there.

How is this related to \texttt{algdiv()}?

### Value

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

```r
algdiv

\[\text{Compute } \log(\Gamma(b)/\Gamma(a+b)) \text{ when } b \geq 8\]
```
Author(s)

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

References


See Also

\texttt{gamma}, \texttt{beta}; my own \texttt{logQab\_asy()}.

Examples

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

```r
##  cbind(a = 2:3, b = 8:14, Qab) # recycling with a warning
##   a b Qab
## 2 3  0.2905265
## 3 8  0.1392002
```n

## algdiv() and my logQab\_asy() give *very* similar results for largish \texttt{b}:

```r
all.equal(-algdiv(3, 100),
          logQab\_asy(3, 100), tol=0) # 1.283e-16 !
```

## relative error

```r
1 + logQab\_asy(3, 1e10)
```

Bernoulli Numbers

Description

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

Usage

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

Arguments

\(n\) integer, \(n \geq 0\).

\texttt{verbose} logical indicating if computation should be traced.

Value

The number \(B_n\), of type \texttt{numeric}.

A side effect is the \textit{caching} of computed Bernoulli numbers in the hidden \texttt{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 \( \zeta \) function.

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

Examples

```r
(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, \texttt{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 \texttt{pntJW39()}.

Its logarithm,

\[
lb_\chi(\nu) := \log\left(\frac{\sqrt{2/\nu}\Gamma((\nu + 1)/2)}{\Gamma(\nu/2)}\right),
\]

is even easier to compute via \texttt{lgamma} and \texttt{log}, and I have used Maple to derive an asymptotic expansion in \( 1/\nu \) as well.

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

### Usage

```r
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 \(b\_chi\_Asymp()\) 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 \(b\_chi\_Asymp(*, order)\) for higher orders, and the \(c1\) and \(c2\) arguments will change, possibly be abolished.

- **order**: the polynomial order in \(\nu\) of the asymptotic expansion of \(b(\nu)\) 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 \(b\_chi\_Asymp()\) 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 \(b(0) = 0\) to (asymptotically) one. Specifically, for large \(nu\), \(b\_chi(nu) = b\_chi\_Asymp(nu)\) and

\[
1 - b(\nu) \sim \frac{1}{4\nu}.
\]

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

\[
\Gamma(z + 1/2)/\Gamma(z) \sim \sqrt{\Gamma(z)} \cdot \left(1 - 1/(8z) + 1/(128z^2) + O(1/z^3)\right),
\]

and applied for \(b(\nu)\) with \(z = \nu/2\), we get

\[
b(\nu) \sim 1 - (1/(4\nu) \cdot (1 - 1/(8\nu)) + O(\nu^{-3})),
\]

which has been implemented in \(b\_chi\_Asymp(*, order=2)\) 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 \(r := \frac{1}{4\nu}\),

\[
b(\nu) = c(\nu) = 1 - r + \frac{1}{2} r^2 + \frac{5}{2} r^3 - \frac{21}{8} r^4 - \frac{399}{8} r^5 + O(r^6).
\]

**Value**

a numeric vector of the same length as \(nu\).

**Author(s)**

Martin Maechler
References

Formula on page 520, after (31.26a)

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, l=10, l=5, add=TRUE)
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"+

## 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", 0), lwd=3)
if(requireNamespace("Rmpfr")) {
xm <- Rmpfr::sequence(2-e, 2+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), lwd=3)
grCol <- adjustcolor("forest green", 1/2)
curve(b_chi, 1/2, l=11, 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, l=11, log="xy")
curve(b_chiasymp(x, one.minus=TRUE), add = TRUE, col = grCol, lwd = 3)
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

```r
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** mathematically 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 instead of R’s or this pure R version relying `bd0()` and `stirlerr()` where the latter currently only provides accurate double precision accuracy.

### Examples

```r
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 $f(x, \cdot)$ of the (noncentral) chi-squared distribution.

**Usage**

```r
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 $\delta$.
- `log` logical indicating if the result is desired on the log scale.
- `eps` positive convergence tolerance for the series expansion: Terms are added while $\text{term} \times q > (1-q) \times \text{eps}$, 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 $\text{termSml} \times \text{sum}$ even when convergence according to `eps` had occurred. 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 $\mid \text{mid} \mid$ 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 $t \cdot k := \text{dpois}(k, \lambda = \text{ncp}/2) \times \text{dchisq}(x, \text{df} = 2 \times k + \text{df})$ for $k = 0, 1, \ldots, \text{kmax}$.
**Value**

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

**Note**

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

**Note**

\( \text{R}'s \) `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, \cdot)` and `dchisq(x, \cdot)` for the same parameters.

**Author(s)**

Martin Maechler, April 2008

**References**

[https://en.wikipedia.org/wiki/Abramowitz_and_Stegun](https://en.wikipedia.org/wiki/Abramowitz_and_Stegun) provides links to the full text which is in public domain.

Chapter 29, Section 3 *Distribution*, (29.4), p. 436.

**See Also**

\( \text{R}'s \) own `dchisq()`.

**Examples**

```r
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))
```

**Description**

Mostly, pure \( \text{R} \) 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 (Rmath) `bd0()`.

These have extra arguments with defaults that correspond to `R`’s Mathlib C 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 = as.integer(-1100 / log2(delta)),
    s0 = .Machine$double.xmin,
    verbose = getOption("verbose"))
bd0C(x, np, delta = 0.1, maxit = 1000L, version = "R4.0", verbose = getOption("verbose"))

# "simple" log1pmx() based versions :
bd0_p1ld1(x, M, tol_logcf = 1e-14, ...)
bd0_p1ld (x, M, tol_logcf = 1e-14, ...)
bd0_l1pm (x, M, tol_logcf = 1e-14, ...)

ebd0 (x, M, verbose = getOption("verbose"), ...) # experimental, may disappear !!
ebd0C(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 (or number-alike ..); distribution 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 non-negative number < 1 (practically required to be ≤ .99), a cutoff for bd0() where a continued fraction series expansion is used when \(|x - M| < \delta \ast (x + M)\).
tol_logcf, eps2, minL1, trace.lcf, logCF, ...
    optional tuning arguments passed to log1pmx(), and to its options passed to logcf().
maxit the number of series expansion terms to be used in bd0() when \(|x - M| \) is small.
The default is \(k\) such that \(2k \leq 2^{-1022-52}\), i.e., will underflow to zero.
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 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 ∈ {0, 1, ..., 30}, i.e., n <= 15 and n is integer or integer + 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 approximation 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, $2^{26.5} = 949062656$, 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.

$$bd0(x, M) := M \cdot D_0\left(\frac{x}{M}\right),$$

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

$$bd0(x, M) = M \cdot \left(\frac{x}{M}\log\left(\frac{x}{M}\right) - 1\right) = x \log\left(\frac{x}{M}\right) - x + M.$$ 

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 $t$,

$$bd0(x, M) = \log(1+t) - t \cdot M = M \cdot ((t+1)(\log(1+t) - 1) + 1) = M \cdot ((t+1)\log(1+t) - t) = M \cdot p_1(t),$$

and

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

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

Note that bd0(x, M) now also works when $x$ and/or $M$ are arbitrary-accurate mpfr-numbers (package Rmpfr).

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 $u := 2t^2 - 1 \in [-1 + e_B, 1]$.

Author(s)

Martin Maechler
References

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

Our package vignette `log1pmx`, bd0, stirlerr - Probability Computations in R.

See Also

dgamma, dpois. High precision versions `stirlerrM(n)` and `stirlerrSer(n,k)` in package DPQmpfr (via the Rmpfr and gmp packages).

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)"
lines(x, bd0x1kC, col=2)
b0.1d1 <- bd0_pl1ld(x, 1000)
b0.1d <- bd0_pl1ld (x, 1000)
b0.1pm <- bd0_pl1pm (x, 1000)
stopifnot(exprs = {
  all.equal(bd0x1kC, bd0x1k, tol=1e-15) # even tol=0 currently ..
  all.equal(bd0x1kC, bd0.1d1, tol=1e-15)
  all.equal(bd0x1kC, bd0.1d , tol=1e-15)
  all.equal(bd0x1kC, bd0.1pm, tol=1e-15)
})

str(log1pmx) ##--> play with { tol_logcf, eps2, minL1, trace.lcf, logCF }

ebd0x1k <- ebd0 (x, 1000)
exC <- ebd0C(x, 1000)
stopifnot(all.equal(exC, ebd0x1k, tol=4e-16))
lines(x, colSums(ebd0x1k), col=adjustcolor(4, 1/2), lwd=4)

x <- 0:250
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))
## dpois_raw() versions:
(vers <- eval(formals(dpois_raw)$version))
mv <- sapply(vers, function(v) dpois_raw(x, 48, version=v))
matplot(x, mv, type="h", log="y", main="dpois_raw(x, 48, version=*)") # "fine"

if(all(mv[,"ebd0_C1"] == mv[,"ebd0_v1")){
cat("versions 'ebd0_C1' and 'ebd0_v1' are identical for lambda=48\n")
}
## now look at *relative* errors -- need "Rmpfr" for "truth"
if(requireNamespace("Rmpfr")) {
  dM <- Rmpfr::dpois(Rmpfr::mpfr(x, 256), 48)
asN <- Rmpfr::asNumeric
relE <- asN(mv / dM - 1)
cols <- adjustcolor(1:ncol(mv), 1/2)

mtit <- "relative Errors of dpois_raw(x, 48, version = *)"
matplot(x, relE, type="l", col=cols, lwd=3, lty=1, main=mtit)
legend("topleft", colnames(mv), col=cols, lwd=3, bty="n")

matplot(x, abs(relE), ylim=pmax(1e-18, range(abs(relE))), type="l", log="y",
main=mtit, col=cols, lwd=2, lty=1, yaxt="n")
sfsmisc::eaxis(2)
legend("bottomright", colnames(mv), col=cols, lwd=2, bty="n", ncol=3)
ee <- c(.5, 1, 2)* 2^-52; eC <- quote(epsilon[C])
abline(h=c(ee[2:3], eC), lty=2, col="gray", lwd=c(1,2,1))
axis(4, at=ee[2:3], expression(epsilon[C], 2 * epsilon[C]), col="gray", las=1)
par(new=TRUE)
plot(x, asN(dM), type="h", col=adjustcolor("darkgreen", 1/3), axes=FALSE, ann=FALSE)
stopifnot(abs(relE) < 8e-13) # seen 2.57e-13

---

**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().
## 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

```r
   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

...
 dnbinomR

See Also

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

Examples

```r
## 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)
```

### Description

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

### Usage

```r
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 `dbinom()`.
- `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` and `prob` or `mu`.

### Author(s)

R Core and Martin Maechler

### References


### 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

```r
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**

```r
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

\[
\begin{align*}
f(x, df, ncp) &= \frac{df^{(df/2)} \exp(-.5*ncp^2)}{(\sqrt{\pi} \times \gamma(df/2) \times (df+x^2)^{(df+1)/2})} \times \sum_{k=0}^{\infty} \frac{\gamma((df + k + df)/2) \times ncp^k}{\prod_{1:k} (2\times x^2/(df+x^2))^{k/2}}.
\end{align*}
\]

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

Value

a number for \( \text{dntJKBf1()} \) and \( \text{.dntJKBch1()} \).

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

Author(s)

Martin Maechler

References


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

See Also

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

Examples

```r
## 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),
             log=TRUE),
```
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). \( df = \infty \) is allowed.
- **ncp**: non-centrality parameter \( \delta \); 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 \( \text{integer} \ f \) 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 \( \mathbb{R} \) (when \( dt() \) did not yet have an \( ncp \) argument).
Value

numeric vector of density values, properly recycled in \((x, \text{df}, \text{ncp})\).

Author(s)


References


See Also
dt, R’s (C level) implementation of the (non-central) t density; dntJKBF, for Johnson et al.’s summation formula approximation.

Examples

```r
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)
1fx <- dt(x, df = 22, ncp =100, log=TRUE)
1fV <- dtWV(x, df = 22, ncp =100, log=TRUE)

head(1fV, 20) # shows that R's dt(*, log=TRUE) implementation is "quite suboptimal"
```

## graphics

```r
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(1fx ~ 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, 1fV, 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
```
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 formatting; 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)))
nounquote(ff <- format01prec(ee))
data.frame(ee, format01prec = ff)
Description

Both are \texttt{R} versions of C99 (and POSIX) standard C (and C++) mathlib functions of the same name. \texttt{frexp}(x) computes base-2 exponent \( e \) and "mantissa", or fraction \( r \), such that \( x = r \times 2^e \), where \( r \in (0.5, 1) \) (unless when \( x \) is in \( c(0, -\text{Inf}, \text{Inf}, \text{NaN}) \) where \( r = x \) and \( e = 0 \), and \( e \) is integer valued.

\texttt{ldexp}(f, E) is the inverse of \texttt{frexp}(): Given fraction or mantissa \( f \) and integer exponent \( E \), it returns \( x = f \times 2^E \). Viewed differently, it's the fastest way to multiply or divide (double precision) numbers with \( 2^E \).

Usage

\begin{verbatim}
frexp(x)
ldevp(f, E)
\end{verbatim}

Arguments

\begin{itemize}
  \item \texttt{x} numeric (coerced to double) vector.
  \item \texttt{f} numeric fraction (vector), in \([0.5, 1)\).
  \item \texttt{E} integer valued, exponent of 2, i.e., typically in \((-1024\sim50):1024\), otherwise the result will underflow to 0 or overflow to \(+/- \text{Inf}\).
\end{itemize}

Value

\texttt{frexp} returns a list with named components \( r \) (of type double) and \( e \) (of type integer).

Author(s)

Martin Maechler

References

On unix-alikes, typically \texttt{man frexp} and \texttt{man ldexp}

See Also

Vaguely relatedly, \texttt{log1mexp()}, \texttt{lsum}, \texttt{logspace.add}.

Examples

\begin{verbatim}
set.seed(47)
x <- c(0, 2^(-3:3), (-1:1)/0,
      rlnorm(2^12, 10, 20) * sample(c(-1,1), 512, replace=TRUE))
head(x, 12)
which(!(iF <- is.finite(x))) # 9 10 11
rF <- frexp(x)
sapply(rF, summary) # (nice only when x had no NA's ..)
data.frame(x=x[!iF], lapply(rF, '[', !iF))
\end{verbatim}
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)
}

logBeta
(Log) Beta Approximations

Description

Compute \( \log(\text{beta}(a, b)) \) 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 \texttt{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 \texttt{Qab_terms()}, currently must be in \{0, 1, .., 5\}.

Details

All \texttt{lbeta*()} functions compute \( \log(\text{beta}(a, b)) \).

We use \( Q_{a,b} = Q_{a,b}(a, b) \) 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, $\text{1Beta}(a, b) := \text{lgamma}(a) - \log\text{Qab}(a, b)$.

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 $\log\text{Qab}()$, and for the large $b$ case $\log\text{Qab\_asy}()$ specifically,

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

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

Note how this is related to computing $\text{qbeta}()$ in boundary cases, and see $\text{algdiv}()$ ‘Details’ about this.

We also have a vignette about this, but really the problem has been addressed pragmatically by the authors of TOMS 708, see the ‘References’ in $\text{pbeta}$, by their routine $\text{algdiv}()$ which also is available in our package $\text{DPQ}$.

**Value**

a fast or simple (approximate) computation of $\text{1Beta}(a, b)$.

**Author(s)**

Martin Maechler

**References**


Formula (6.1.47), p.257

**See Also**

R’s $\text{beta}$ function; $\text{algdiv}()$.

**Examples**

```r
## TODO
```
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 \texttt{lchoose()} computes.

Usage

\begin{verbatim}
lfastchoose(n, k)
f05lchoose(n, k)
\end{verbatim}

Arguments

\begin{itemize}
  \item \texttt{n} \hspace{1cm} a numeric vector.
  \item \texttt{k} \hspace{1cm} a integer valued numeric vector.
\end{itemize}

Value

a numeric vector with the same attributes as \texttt{n + k}.

Author(s)

Martin Maechler

See Also

\texttt{lchoose}.

Examples

\begin{verbatim}
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 ?
\end{verbatim}

\texttt{lgamma1p} \hspace{1cm} \textit{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 \texttt{log(gamma(a+1))} or \texttt{lgamma(a+1)}, accurately also for (very) small \( a \) (\( 0 < a < 0.5 \)).
Usage

\texttt{lgamma1p (a, tol\_logcf = 1e-14, f.tol = 1, ...)}
\texttt{lgamma1p.(a, cutoff.a = 1e-6, k = 3)}
\texttt{lgamma1p\_series(x, k)}
\texttt{lgamma1pC(x)}

Arguments

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

Details

\texttt{lgamma1p()} is an \texttt{R} translation of the function (in Fortran) in Didonato and Morris (1992) which uses a 40-degree polynomial approximation.
\texttt{lgamma1p\_series(x, k)} is Taylor series approximation of order \texttt{k}, (derived via Maple), which is 
\[-\gamma x + \pi^2 x^2 / 12 + O(x^3), \text{ where } \gamma \text{ is Euler's constant 0.5772156649.} \ldots\]
\texttt{lgamma1pC()} is an interface to \texttt{R} C API ('Mathlib' / 'Rmath.h') function.

Value

a numeric vector with the same attributes as \texttt{a}.

Author(s)

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

Martin Maechler, notably for \texttt{lgamma1p\_series()} which works with package \texttt{Rmpfr} but otherwise may be much less accurate than Morten's 40 term series!

References


See Also

\texttt{log1pmx, log1p, pbeta}.

Examples

\begin{verbatim}
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^(-4*1:8)
abline(h = at, v = at, col = "lightgray", lty = "dotted")
\end{verbatim}
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 \( \text{Bern}(k) \) for \( k \in 1, \ldots, 2n \).

**Usage**

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

**Arguments**

- `x` numeric vector
- `n` integer specifying the approximation order.

**Value**

numeric vector with the same attributes (\texttt{length()}) as \( x \), containing approximate \texttt{lgamma(x)} values.

**Author(s)**

Martin Maechler

**See Also**

\texttt{lgamma}.

**Examples**

```r
##---- 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 - 0.5) * log(x) - x + log(2 * pi)/2
  if (n >= 1) {
    Ix2 <- 1/(x * x)
    k <- 1:n
    Bern(2 * n)
  }

```
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 (‘Mathlib’ / ‘Rmath.h’) function.

\( \log1pexpC() \) is an interface to \( R \)‘s ‘Mathlib’ double function \( \log1pexp() \) which computes \( \log(1 + \exp(x)) \), accurately, notably for large \( x \), say, \( x > 720 \).

Usage

\begin{verbatim}
log1mexp (x)
log1mexpC(x)
log1pexpC(x)
\end{verbatim}

Arguments

\( x \) numeric vector of positive values.

Author(s)

Martin Maechler

References


See Also

The \( \log1mexp() \) function in CRAN package \texttt{copula}, and the corresponding vignette (in the ‘References’).

Examples

\begin{verbatim}
l1m.xy <- curve(log1mexp(x), -10, 10, n=1001)
stopifnot(with(l1m.xy, all.equal(y, log1mexpC(x))))
x <- seq(0, 710, length=1+710*2^4); stopifnot(diff(x) == 1/2^4)
l1pm <- cbind(log1p(exp(x)),
             log1pexpC(x))
matplot(x, l1pm, type="l", log="xy") # both look the same
ifF <- is.finite(l1pm[,1])
stopifnot(all.equal(l1pm[ifF,2], l1pm[ifF,1], tol=1e-15))
\end{verbatim}
**log1pmx**

**Accurate log(1+x) - x Computation**

**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 \( \text{Rmpfr} \)).

**Usage**

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

```r
log1pmxC(x) # TODO in future: arguments (minL1, eps2, tol_logcf),
# possibly with *different* defaults (!)
```

**Arguments**

- **x** numeric (or "mpfr" number) vector with values \( x > -1 \).
- **tol_logcf** a non-negative number indicating the tolerance (maximal relative error) for the auxiliary `logcf()` function.
- **eps2** non-negative cutoff where the algorithm switches from a few terms, to using `logcf()` explicitly. Note that for more accurate mpfr-numbers the default \( \text{eps} = 0.01 \) is too large, even more though when the tolerance is lowered (from 1e-14).
- **minL1** negative cutoff, called `minLog1Value` in Morten Welinder’s C code for `log1pmx()` in ‘R/src/nmath/pgamma.c’, hard coded there to -0.79149064 which seems not optimal for computation of `log1pmx()`, at least in some cases, and hence the default may be changed in the future.
- **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 \( x \),

\[ m_l = \text{minL1} = -0.79149064 \]

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`,
- \( x \in [m_l, 1], \text{and } |x| \geq \epsilon_2 \): use `t(2ylogcf(y, 3, 2) - x)`,
where \( t := \frac{r}{2r + 2} \), and \( y := t^2 \).

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

\[
\log(1 + x) = 2 \left( r + \frac{r^3}{3} + \frac{r^5}{5} + \ldots \right),
\]

where \( r := \frac{x}{x + 2} \), 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, parametrized and tuned by Martin Maechler.

**References**


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, \( 1gamma1p \) which calls \( \log1pmx \), \( \log1p \)

**Examples**

```r
(doExtras <- DPQ:::doExtras()) # TRUE e.g. if interactive()
n1 <- if(doExtras) 1001 else 201
curve(log1pmx, -0.9999, 7, n=n1); abline(h=0, v=-1:0, lty=3)
curve(log1pmx, -1, 1, n=n1); abline(h=0, v=0, lty=3)
curve(log1pmx, -0.01, .01, n=n1) -> l1xz2; abline(h=0, v=0, lty=3)
## C and R versions correspond closely:
with(l1xz2, stopifnot(all.equal(y, log1pmxC(x), tol = 1e-15)))
e <- if(doExtras) 2^-12 else 2^-8; by.p <- 1/(if(doExtras) 256 else 64)
xd <- c(seq(-e, e+100*e, by=e), seq(by.p, 5, by=by.p)) # length 676 or 5476 if do.X.
plot(xd, log1pmx(xd), type="l", col=2, main = "log1pmx(x)")
abline(h=0, v=-1:0, lty=3)
## much more graphics etc in ../tests/dnbinom-tst.R (and the vignette, see above)
```
Continued Fraction Approximation of Log-Related Power Series

Description

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

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

Needed as auxiliary function in `log1p()` and `lgamma1p()`.

Usage

\[
\text{logcfR}(x, i, d, \text{eps}, \text{maxit} = 10000L, \text{trace} = \text{FALSE})
\]

Arguments

- \(x\) numeric vector of values typically less than 1. "mpfr" (of potentially high precision, package \texttt{Rmpfr}) work in \texttt{logcfR*(x,*)}.
- \(i\) positive numeric
- \(d\) non-negative numeric
- \(\text{eps}\) positive number, the convergence tolerance.
- \(\text{maxit}\) a positive integer, the maximal number of iterations or terms in the truncated series used.
- \(\text{trace}\) 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

- \texttt{logcfR().} a pure \texttt{R} version where the iterations happen vectorized in \(x\), only for those components \(x[i]\) they have not yet converged. This is particularly beneficial for not-very-short "mpfr" vectors \(x\), and still conceptually equivalent to the \texttt{logcfR()} version.
- \texttt{logcfR():} a pure \texttt{R} version where each \(x[i]\) is treated separately, hence “properly” vectorized, but slowly so.
- \texttt{logcf():} only for \texttt{numeric} \(x\), calls into (a clone of) \texttt{R}'s own (non-API currently) \texttt{logcf()} \texttt{C Rmathlib} function.

Value

a numeric-alike vector with the same attributes as \(x\). For the \texttt{logcfR*()} versions, an "mpfr" vector if \(x\) is one.

Note

Rescaling is done by (namespace hidden) “global” scalefactor which is \(2^{256}\), represented exactly (in \texttt{double} precision).
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.

\[
\text{logspace.add}(lx, ly) := \log(\exp(lx) + \exp(ly))
\]

\[
\text{logspace.sub}(lx, ly) := \log(\exp(lx) - \exp(ly))
\]

Usage

\[
\text{logspace.add}(lx, ly)
\]
\[
\text{logspace.sub}(lx, ly)
\]

Arguments

- \(lx, ly\): numeric vectors, typically of the same length, but will be recycled to common length as with other \(R\) arithmetic.
### lssum

#### Description

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

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

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

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

#### Arguments

- `lxabs`: n-vector of values \( \log(|x_1|), \ldots, \log(|x_n|) \).
- `signs`: corresponding signs \( \text{sign}(x_1), \ldots, \text{sign}(x_n) \).
- `l.off`: the offset to substract and re-add; ideally in the order of \( \text{max}(\cdot) \).
- `strict`: logical indicating if the function should stop on some negative sums.
Value

\[ \log(x_1 + \ldots + x_n) = \log(\text{sum}(x)) = \log(\text{sum}(|x|)) = \log(\text{sum}(\text{sign}(x) \cdot |x|)) = \log(\exp(\log(x_0) + \text{sum}(\text{sign} \cdot \exp(\log(|x|))))) = \log(\exp(\log(x_0) + \text{sum}(\text{sign} \cdot \exp(\log(|x|))))) \]

Author(s)

Marius Hofert and Martin Maechler (for package \texttt{copula}).

See Also

\texttt{lsum()} which computes an exponential sum in log scale with \textit{out} signs.

Examples

```r
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, lsum() 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 <- lsum(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 <- lsum(lxabs = L2$lax, signs = L2$sx)) # 710.0373
stopifnot(all.equal(lsS2, lsS + 10, tol = 1e-14))
```

\texttt{lsum} \hspace{1cm} \textit{Properly Compute the Logarithm of a Sum (of Exponentials)}

Description

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

If the inputs are denoted \( l_i = \log(x_i) \) for \( i = 1, 2, \ldots, n \), we compute \( \log(\text{sum}(\exp(l[])))) \), numerically stably.

Simple vector version of \texttt{copula::lsum()} (CRAN package \texttt{copula}).

Usage

\texttt{lsum(1x, 1.off = max(1x))}
Arguments

1x   n-vector of values log(x_1)...log(x_n).
1.off   the offset to subtract and re-add; ideally in the order of the maximum of each column.

Value

\[ \log(x_1 + \ldots + x_n) = \log(\text{sum}(x)) = \log(\exp(\log(x))) = \log(\exp(\log(x_{\text{max}})) \cdot \text{sum}(\exp(\log(x) - \log(x_{\text{max}})))) \]

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

```r
## The "naive" version :
1sum0 <- 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(lsum(lx1), lssum(lx1)), 700.000045400960403, tol=8e-16)
  all.equal(lsum(lx2), 750.458675145387133, tol=8e-16)
  all.equal(lsum(lx3), -749.541324854612867, tol=8e-16)
  ## identical: matrix-version <==> vector versions
  identical(lsum(lx4), l3)
  identical(lsum(lx4), lsum(\text{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=lx1, lx2=lx2, lx3=lx3))
  identical(apply(m6, 2, lsum), c(lm3, lx4=lx3, lx5=Inf, lx6=lx6[[1]]))
})
```
Description

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

Convergence typically happens when the steps size becomes smaller than \(\text{eps}\).

\texttt{keepAll = TRUE} to also get the vectors of consecutive values of \(x\) and \(G(x, z)\);

Usage

\[
\text{newton}(x0, \text{G, g, z,}}
\]
\[
x\text{Min} = \text{-Inf, xMax = Inf, warnRng = TRUE,}
\]
\[
dx\text{Max} = 1000, \text{eps = 0.0001, maxiter = 1000L,}
\]
\[
\text{warnIter = missing(maxiter) || maxiter >= 10L,}
\]
\[
\text{keepAll = NA)}
\]

Arguments

- \(x0\): numeric start value.
- \(\text{G, g}\): must be \texttt{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.
- \(x\text{Min, xMax}\): numbers defining the allowed range for \(x\) during the iterations; e.g., useful to set to \(0\) and 1 during quantile search.
- \(\text{warnRng}\): \texttt{logical} specifying if a \texttt{warning} should be signalled when start value \(x0\) is outside \([x\text{Min, xMax}]\) and hence will be changed to one of the boundary values.
- \(dx\text{Max}\): maximal step size in \(x\)-space. (The default 1000 is quite arbitrary, do set a good maximal step size yourself!)
- \(\text{eps}\): positive number, the \texttt{absolute} convergence tolerance.
- \(\text{maxiter}\): positive integer, specifying the maximal number of Newton iterations.
- \(\text{warnIter}\): \texttt{logical} specifying if a \texttt{warning} should be signalled when the algorithm has not converged in \(\text{maxiter}\) iterations.
- \(\text{keepAll}\): \texttt{logical} specifying if the full sequence of \(x\)- and \(G(x,*)\) values should be kept and returned:

\texttt{NA}, the default: \texttt{newton} returns a small list of final “data”, with 4 components \(x = x*, G = G(x*, z), \text{it}, \text{and converged}.
\texttt{TRUE}: returns an extended \texttt{list}, in addition containing the vectors \(x.\text{vec}\) and \(G.\text{vec}.
\texttt{FALSE}: returns only the \(x*\) value.

Details

Because of the quadratic convergence at the end of the Newton algorithm, often \(x^*\) satisfies approximately \(|G(x^*, z)| < \text{eps}^2|\).

\texttt{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 \(\text{R}\) 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, $\{x_0, \ldots, x^*\}$.
- $G.vec$ 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


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

```r
## 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
Nx.lx <- 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, col="green2", lty=3)
lines(BL, vapply(BL, N1 , pi), col = 2, lwd = 2, lty = 2)
```
### Description

The DPQ package provides some numeric constants used in some of its distribution computations. The `all_mpfr()` and `any_mpfr()` return `TRUE` if 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.
modf(x) splits each x into integer part (as trunc(x)) and fractional (remainder) part in \((-1, 1)\) and corresponds to the R version of the C99 (and POSIX) standard C (and C++) mathlib functions of the same name.

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 ] \(\approx\) \(-x\)
# = 3196577161300663808 \(\approx\) 3.2e+18
M_minExp # = log(2) * .Machine$double.min.exp # \(\approx\) -708.396..
G_half # = sqrt(pi) = Gamma( 1/2 )

## Functions :
all_mpfr(...)
any_mpfr(...)
logr(x, a) # == log(x / (x + a)) -- but numerically smart; \(x \geq 0\), \(a > -x\)
modf(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 and consistently for \(\exp(-\lambda)\) and \(\log(\lambda)\).

Author(s)

Martin Maechler

See Also

.Machine
Examples

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

logr(1:3, a=1e-10)
okLongDouble(verbose=TRUE) # verbose: show (C-level) computations
## typically TRUE, but not e.g. in a valgrinded R-devel of Oct.2019
## Here is typically the "boundary":
rr <- try(unquote(function(x) okLongDouble(x) - 1/2,
c(11350, 11400), tol=1e-7, extendInt = "yes"))
str(rr, digits=9) ## seems somewhat platform dependent: now see
## $ root : num 11376.563
## $ estim.prec: num 9.313e-08
## $ iter : int 29

set.seed(2021); x <- runif(100, -7,7)
mx <- modf(x)
with(mx, head( cbind(x, i=mx$i, fr=mx$fr) )) # showing the first cases
with(mx, stopifnot( x == fr + i,
i == trunc(x),
 sign(fr) == sign(x)))
```

---

**p1l1**

Numerically Stable \( p_{1l1}(t) = (t+1)\log(1+t) - t \)

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 its 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 \( |t| \leq c \) 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

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

Arguments

- \( t \): numeric a-like vector ("mpfr" included), larger (or equal) to -1.
- \( ... \): optional (tuning) arguments, passed to \( \log1pmx() \).
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 factor; 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” t.

Author(s)
Martin Maechler

See Also
bd0: our package vignette log1pmx, bd0, stirlerr - Probability Computations in R. dbinom the latter for the C.Loaders(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
\(t \leftarrow 2 \times \text{seq}(-59, -1, \text{by}=1/4)\)
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/log1pmx-etc.Rnw
#### 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)
all.equal(sapply(k.s, function(k) p1l1ser(t,k)) -> m.p1l1,
          sapply(k.s, function(k) .p1l1ser(t,k)) -> m.p1l., tolerance = 0)
p1tab <-
cbind(b1 = bd0(t+1, 1),
      b.10 = bd0(10*t+10,10)/10,
      direct = p1l1.(t),
      p1l1p = p1l1p(t),
      p1l1 = p1l1(t),
      sapply(k.s, function(k) p1l1ser(t,k)))
matplot(t, p1tab, type="l", col=1:6, lty=1:5, lwd=1,
        ## (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 \( p_{1l1}(t) \) 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 \( p_{1l1}(t) \) 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 \( p_{1l1}() \) 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^n*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,
    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
", k, cut.y.at, sum(i.pos), sum(i.neg)))
  nCoefLm <- function(x,y) coef(lm.fit(x=x, y=y))
  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]))

## fantasticaly 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()

###------------- 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)))),
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 \( F(q; \ast) \) should be returned or the upper tail probability \( 1 - F(q) \).
- **eps**: the tolerance used to determine convergence. \( \text{eps} \) has been hard coded in C code to \( 0.5 \times .\text{Machine}\$\text{double}.\text{eps} \), which is equal to \( 2^{-53} \) or \( 1.110223e-16 \).
- **sml**: the smallest positive number on the typical platform. The default \( .\text{Machine}\$\text{double}.\text{xmin} \) is hard coded in the C code (as \( \text{DBL_MIN} \)), 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:)


See Also

- `pbeta`

Examples

```r
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"))
}
```
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**

```r
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, ..., 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


See Also

`phyper`

Examples

```r
##---- 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)

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


See Also

- `phyper`, the hypergeometric distribution, and R’s own “exact” computation. `pbinom`, the binomial distribution functions.

Examples

```r
## 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)
```

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 \( \text{phyper}(q, m, n, k) \).

Author(s)

Martin Maechler

References


See Also

\( \text{phyper} \).

Examples

```r
## 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)
}
```

\textbf{Description}

Compute Molenaar’s two normal approximations to the (cumulative hypergeometric distribution \( \text{phyper} \)).
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


See Also

aphyper, pnorm.

Examples

## TODO

phyperPeizer

---

**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


See Also

phyper.

Examples

```r
## The function is currently defined as

phyperPeizer <- function(q, m, n, k)
{
  ## Purpose: Peizer's extremely good Normal Approx. to cumulative Hyperbolic
  ## ----------------------------------------------------------------------
  ## (6.94) -- in proper order!
  ## 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))
  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

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! %%%%%%%%%%%%%%%%%%%
(A <- cbind(x, m, n, k)) # shows the recycling
cbind(A, ph = phyper (x, m, n, k),
  phR = phyperR(x, m, n, k))
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, ..., m + n.
lower.tail logical; if TRUE (default), probabilities are P[X ≤ 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


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 slight 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, ..., 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

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.
pl2curves

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, 1og) (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(. . . , 1og = log,p).
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: invisible return both curve() results, i.e., (x,y1, y2), possibly as data frame

**Author(s)**

Martin Maechler

**See Also**

`curve`, ..

**Examples**

```r
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 initial 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**

```r
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**


**See Also**

R’s own pbeta.

**Examples**

```r
# 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")
```
pnchi1sq <- 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(868,900,850,800,720,720,800,1000))
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")
pnR <- 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")
pnR <- 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 3.0980e-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 $\text{df} = 1$ and $\text{df} = 3$, using ‘exact’ formulas only involving the standard normal (Gaussian) cdf $\Phi()$ and its derivative $\phi()$, i.e., R’s `pnorm()` and `dnorm()`.

**Usage**

```r
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 \( H_n(x) \).

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`


See Also

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

Examples

```r
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: """

```r
```
## df = 1 ---------------------------------------------------------

```r
qS <- c(0, 2^seq(-40, 4, by=1/16))
mls <- cbind(pch = pchisq (qS, df=1, ncp = 1)
  , p1.0= pchisq(qS, ncp = 1, epsS = 0)
  , p1.4= pchisq(qS, ncp = 1, epsS = 1e-4)
  , p1.3= pchisq(qS, ncp = 1, epsS = 1e-3)
  , p1.2= pchisq(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, mls, type = "l", log="y" , col=cols, lwd=lws)
matplot(qS, mls, type = "l", log="xy", col=cols, lwd=lws) ; abl.leg("right")
## ==== "Errors" ===================================================
## Absolute: -------------------------
matplot(qS, mls[,1] - mls[,-1] , type = "l", log="x" , col=cols, lwd=lws)
matplot(qS, abs(mls[,1] - mls[,-1]), type = "l", log="xy", col=cols, lwd=lws)
abl.leg("bottomright")
rbind(all = range(aE1e2 <- abs(mls[,"pch"] - mls[,"p1.2"])),
      less.75 = range(aE1e2[ qS <= 3/4]))
## Lnx(F34;i7) M1mac(BDR)
## all 0 7.772e-16 1.110e-15
## less.75 0 1.665e-16 2.220e-16
stopifnot(aE1e2[ qS <= 3/4] <= 4e-16, aE1e2 <= 2e-15) # check
## Relative: -------------------------
matplot(qS, 1 - mls[,-1]/mls[,1] , type = "l", log="x", col=cols, lwd=lws)
abl.leg()
matplot(qS, abs(1 - mls[,-1]/mls[,1]), type = "l", log="xy", col=cols, lwd=lws)
abl.leg()
```

```r
## number of correct digits (Inf' --> 17) :
corrDigs <- pmin(round(-log10(abs(1 - mls[,-1]/mls[,1])[-1,]), 1), 17)
table(corrDigs > 9.8) # all
range(corrDigs[ qS[-1] > 1e-8, 1 ] , corrDigs[, 2:4]) # [11.8 , 17]
(min (corrDigs[ qS[-1] > 1e-6, 1:2], corrDigs[, 3:4]) -> mi6) # 13
(min (corrDigs[ qS[-1] > 1e-4, 1:3], corrDigs[, 4]) -> mi4) # 13.9
stopifnot(exprs = {
  corrdigs >= 9.8
  c(corrdigs[ qS[-1] > 1e-8, 1 ] , corrdigs[, 2]) >= 11.5
  mi6 >= 12.7
  mi4 >= 13.6
})
```

## df = 3 -------------- NOTE: epsS=0 for small qS is "non-sense" -------

```r
qS <- c(0, 2^seq(-40, 4, by=1/16))
ee <- c(1e-3, 1e-2, .04)
mls <- cbind(pch = pchisq (qS, df=3, ncp = 1)
  , p1.0= pchisq(qS, ncp = 1, epsS = 0)
  , p1.4= pchisq(qS, ncp = 1, epsS = 1e-4)
  , p1.3= pchisq(qS, ncp = 1, epsS = 1e-3)
  , p1.2= pchisq(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, mls, type = "l", log="y" , col=cols, lwd=lws)
matplot(qS, mls, type = "l", log="xy", col=cols, lwd=lws) ; abl.leg("right")
## ==== "Errors" ===================================================
## Absolute: -------------------------
matplot(qS, mls[,1] - mls[,-1] , type = "l", log="x" , col=cols, lwd=lws)
matplot(qS, abs(mls[,1] - mls[,-1]), type = "l", log="xy", col=cols, lwd=lws)
abl.leg("bottomright")
rbind(all = range(aE1e2 <- abs(mls[,"pch"] - mls[,"p1.2"])),
      less.75 = range(aE1e2[ qS <= 3/4]))
```

```r
## Lnx(F34;i7) M1mac(BDR)
## all 0 7.772e-16 1.110e-15
## less.75 0 1.665e-16 2.220e-16
stopifnot(aE1e2[ qS <= 3/4] <= 4e-16, aE1e2 <= 2e-15) # check
## Relative: -------------------------
matplot(qS, 1 - mls[,-1]/mls[,1] , type = "l", log="x", col=cols, lwd=lws)
abl.leg()
matplot(qS, abs(1 - mls[,-1]/mls[,1]), type = "l", log="xy", col=cols, lwd=lws)
abl.leg()
```

```r
## number of correct digits (Inf' --> 17) :
corrDigs <- pmin(round(-log10(abs(1 - mls[,-1]/mls[,1])[-1,]), 1), 17)
table(corrDigs > 9.8) # all
range(corrDigs[ qS[-1] > 1e-8, 1 ] , corrDigs[, 2:4]) # [11.8 , 17]
(min (corrDigs[ qS[-1] > 1e-6, 1:2], corrDigs[, 3:4]) -> mi6) # 13
(min (corrDigs[ qS[-1] > 1e-4, 1:3], corrDigs[, 4]) -> mi4) # 13.9
stopifnot(exprs = {
  corrdigs >= 9.8
  c(corrdigs[ qS[-1] > 1e-8, 1 ] , corrdigs[, 2]) >= 11.5
  mi6 >= 12.7
  mi4 >= 13.6
})
```
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

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

for \( x \geq 0 \); for more, see \( \texttt{R}'s \) help page for \texttt{pchisq}.

- \( \texttt{R}'s \) 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 approximation 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 approximation error is \( O(1/\lambda) \) for \( \lambda \to \infty \).
- Abdel-Aty (1954)'s “first approximation” based on Wilson-Hilferty via Gaussian (\texttt{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 \( ncp \) \( \lambda \) 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. \texttt{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

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

pnchisqV(x, ..., verbose = 0)

pnchisqRC

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

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_sss

\( x, df, ncp, lower.tail = TRUE, log.p = FALSE, i.max = 10000) \)

pnchisqTerms

\( x, df, ncp, lower.tail = TRUE, i.max = 10000) \)

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.

\( q, df, ncp = 0, errmax = 1e-12, rettol = 2 * .Machine$double.eps, \\
maxit = 1e+05, eps = rettol, verbose = FALSE) \)

Arguments

\( x \)

numeric vector (of ‘quantiles’, i.e., abscissa values).
pnchisqAppr

- **q**: number (‘quantile’, i.e., abscissa value.)
- **df**: degrees of freedom > 0, maybe non-integer.
- **ncp**: non-centrality parameter \( \delta \); ....
- **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 \( \log.p = \text{TRUE} \) situations.
- **cutOffncp**: a positive number, the cutoff value for \( \text{ncp} \)...
- **itSimple**: ...
- **errmax**: absolute error tolerance.
- **reltol**: convergence tolerance for relative 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” \( \text{ncp} \) (defined to fulfill \( \text{ncp} < \text{cutOffncp} \) !).
- **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 \( \lambda \) computations.
- **eps**: convergence tolerance, a positive number.

**Details**

`pnchisq_ss()` uses \( \text{si} \leftarrow \text{ss}(x, \text{df}, \ldots) \) to get the series terms, and returns \( \text{2*dcisq}(x, \text{df} = \text{df} + 2) * \sum(\text{si}$\$) \).

`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

Chapter 29 Noncentral \( \chi^2 \)-Distributions; notably Section 8 Approximations, p.461 ff.  
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 pchisq_ss().

Examples

## set of quantiles to use:  
qq <- c(0.001, 0.005, 0.01, 0.05, (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 = "pchisq"))  
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_len(length(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[I] should be improved:  
## First, the central P[I] which is close to exact -- choosing df=2 allows  
## truly exact values: \( \chi^2 = \text{Exp}(1) \) !  
opal <- palette()  
palette(c("black", "red", "green3", "blue", "magenta", "gold3", "gray44"))  
cR <- curve(pchisq (x, df=2, lower.tail=FALSE, log.p=TRUE), 0, 4000, n=2001)  
cRC <- curve(pchisqRC(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_(pchisqPatnaik,  
pchisqPearson,  
pchisqAbdelAty,  
pchisqBolKuz,  
pchisqSankaran_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))

---

## 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.,
  \[ (x*ncp / df^2) \ll 1. \]

- `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 first or "s" for the second 
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


See Also

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

Examples

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

---

### `pnormAsymp`

**Asymptotic Approximation of (Extreme Tail) `pnorm()`**

**Description**

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

**Usage**

```r
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 \(\geq 0\) 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**

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

```r
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 sapplyMpf():
  PXA. <- sapply(setNames(0:5, paste("k =",0:5)),
                 pnormAsymp, x=x., lower.tail = FALSE, log.p=TRUE)
  ## rel.errors :
  print(Rmpfr::roundMpf(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=1:5, lty=1:5, bty="n", inset=.01)
  at1 <- axTicks(1, col=pch[1:2], 3))
  axis(1, at=at1)
  abline(h = 1:2* 2^-53, v = at1, lty=3)
  axis(4, las=2, at= 2^-53, label = quote(epsilon[C]), col="gray20")
}
```

---

**pnormLU**

*Bounds for 1-\(\Phi(\cdot)\) – Mill’s Ratio related Bounds for pnorm()*

**Description**

Bounds for \(1 - \Phi(x)\), i.e., \(\text{pnorm}(x, *, \text{lower.tail}=\text{FALSE})\), 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., \(\text{pnorm()}\).
Value

a numeric vector like x

Author(s)

Martin Maechler

References


See Also

pnorm.

Examples

```r
x <- seq(1/64, 10, by=1/64)
px <- cbind(
  IQ = 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, pnx[,1:3], 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)
pnx <- cbind(
  IQ = 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, pnx[,1:3], 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*
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*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*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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>vector of quantiles (called q in pt(.)).</td>
</tr>
<tr>
<td>df</td>
<td>degrees of freedom (&gt; 0, maybe non-integer). df = Inf is allowed.</td>
</tr>
<tr>
<td>ncp</td>
<td>non-centrality parameter δ ≥ 0; If omitted, use the central t distribution.</td>
</tr>
<tr>
<td>log, log.p</td>
<td>logical; if TRUE, probabilities p are given as log(p).</td>
</tr>
<tr>
<td>lower.tail</td>
<td>logical; if TRUE (default), probabilities are P[X ≤ x], otherwise, P[X &gt; x].</td>
</tr>
<tr>
<td>use.pnorm</td>
<td>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).</td>
</tr>
<tr>
<td>itrmax</td>
<td>number of iterations / terms.</td>
</tr>
<tr>
<td>errmax</td>
<td>convergence bound for the iterations.</td>
</tr>
<tr>
<td>verbose</td>
<td>logical or integer determining the amount of diagnostic print out to the console.</td>
</tr>
<tr>
<td>M</td>
<td>positive integer specifying the number of terms to use in the series.</td>
</tr>
</tbody>
</table>
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 $b = b_{\text{chi}}(df)$.

pntJW39(): is an improved version of pntJW39.0(), using $1 - b = b_{\text{chi}}(df, \text{one.minus}=\text{TRUE})$ 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


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**

```r
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, 1a)` and `ppFUN(q, 1a).
- `lambda` positive parameter of the Poisson distribution, `lambda = \lambda = E[X] = Var[X]` where `X \sim \text{Poi}(\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 direct 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, lower.tail=FALSE)` is the upper tail `iP-quantile` of `Poi(lambda)`.
- `verbose` integer `(\geq 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

```r
(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 :
pp00 <- function(x, lambda, all.from.0=TRUE)
    cumsum(dpois(if(all.from.0) 0:x else x, lambda=lambda))
## and test it:
p0 <- pp00( 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 = pp00)
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 = pp00))

## The first version [called 'err.lambd0()' for years] used simple cumsum(dpois(..))
## NOTE: It is *still* 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 = pp00))
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...
fpw            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

\[
\begin{align*}
\text{qbeta.R}(0.6, 2, 3) & \approx 0.4445 \\
\text{qbeta.R}(0.6, 2, 3) - \text{qbeta}(0.6, 2,3) & \approx 0
\end{align*}
\]

\[
\begin{align*}
\text{qbetaRV} & \leftarrow \text{Vectorize(\text{qbeta.R}, "alpha") } # \text{now can use}
\text{curve(\text{qbetaRV}(x, 1.5, 2.5))}
\text{curve(\text{qbeta}(x, 1.5, 2.5), add=TRUE, lwd = 3, col = \text{adjustcolor("red", 1/2)})}
\end{align*}
\]

## an example of disagreement (and doubt, as borderline, close to underflow):
\[
\begin{align*}
\text{qbeta.R}(0.5078, .01, 5) & \rightarrow 2.77558e-15 \quad \text{# but}
\text{qbeta}(0.5078, .01, 5) & \rightarrow 1.776357e-15 \quad \text{now gives 4.651188e-31} \quad !!
\text{qbeta}(0.5078, .01, 5, \text{ncp}=0) & \quad \text{also gives 4.651188e-31}
\end{align*}
\]

---

### 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 Rmathlib 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**

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

Compute Approximate Quantiles of the Chi-Squared Distribution

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 de-
pends 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
Compute (Approximate) Quantiles of the Gamma Distribution

Description

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

Usage

\texttt{qgammaAppr(p, shape, lower.tail = TRUE, log.p = FALSE, tol = 5e-07)}

\texttt{qgamma.R(p, alpha, scale = 1, lower.tail = TRUE, log.p = FALSE,}
\quad \texttt{EPS1 = 0.01, EPS2 = 5e-07, epsN = 1e-15, maxit = 1000,}
\quad \texttt{pMin = 1e-100, pMax = (1 - 1e-14),}
\quad \texttt{verbose = getOption("verbose"))}

\texttt{qgammaApprKG(p, shape, lower.tail = TRUE, log.p = FALSE)}

\texttt{qgammaApprSmallP(p, shape, lower.tail = TRUE, log.p = FALSE)}

Arguments

\begin{itemize}
  \item \texttt{p} \hspace{1cm} numeric vector (possibly log transformed) probabilities.
  \item \texttt{shape, alpha} \hspace{1cm} shape parameter, non-negative.
  \item \texttt{scale} \hspace{1cm} scale parameter, non-negative, see \texttt{qgamma}.
  \item \texttt{lower.tail, log.p} \hspace{1cm} logical, see, e.g., \texttt{qgamma()}; must have length 1.
  \item \texttt{tol} \hspace{1cm} tolerance of maximal approximation error.
  \item \texttt{EPS1} \hspace{1cm} small positive number. ...
  \item \texttt{EPS2} \hspace{1cm} small positive number. ...
  \item \texttt{epsN} \hspace{1cm} small positive number. ...
  \item \texttt{maxit} \hspace{1cm} maximal number of iterations. ...
  \item \texttt{pMin, pMax} \hspace{1cm} boundaries for \texttt{p}. ...
  \item \texttt{verbose} \hspace{1cm} logical indicating if the algorithm should produce “monitoring” information.
\end{itemize}

Details

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

If we look at Abramowitz&Stegun \textit{gamma} \( * (a, x) = x^{-a} \ * F(a, x) \) and its series \( g * (a, x) = \frac{1}{gamma(a)} * (1/a - 1/(a+1) * x + ...) \),
then the first order approximation \( P(a, x) = x^{-a} \cdot g \cdot (a, x) = x^{-a} / \text{gamma}(a + 1) \) and hence its inverse \( x = qgamma(p, a) = (p \cdot \text{gamma}(a + 1)) / a \) should be good as soon as \( 1/a >> 1/(a + 1) \cdot x \)

\[ \Rightarrow x \approx (a+1)/a = (1 + 1/a) \]

\[ \Rightarrow x < \varepsilon \cdot (a+1)/a \]

\[ \Rightarrow \log(x) < \log(\varepsilon) + \log((a+1)/a) \approx -36 - \log(a) \text{ where } \log(x) \approx \log(p \cdot \text{gamma}(a+1)) / a = (\log(p) + \text{lgamma1p}(a)) / a \]

such that the above

\[ \Rightarrow (\log(p) + \text{lgamma1p}(a)) / a < \log(\varepsilon) + \log((a+1)/a) \]

\[ \Rightarrow \log(p) + \text{lgamma1p}(a) < a \cdot (\log(a) + \log(\varepsilon) + \log1p(a)) \]

\[ \Rightarrow \log(p) < a \cdot (\log(a) + \log(\varepsilon) + \log1p(a)) \cdot \text{lgamma1p}(a) =: \text{bnd}(a) \]

Note that \( \text{qgammaApprSmallP()} \) indeed also builds on \( \text{lgamma1p}() \).

\( .\text{qgammaApprBnd}(a) \) provides this bound \( \text{bnd}(a) \); it is simply \( a \cdot (\log(\varepsilon) + \log1p(a) - \log(a)) \) - \( \text{lgamma1p}(a) \), where \( \log(\varepsilon) = \log(\varepsilon) \approx -52 \cdot \log(2) = -36.04365 \).
qnbinomR

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 <- qnbinom(x10, mu=4, size=10)
x1c <- rnbinom(500, prob = 31/32, size = 100); p1c <- qnbinom(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)`
- `qchisqAppr.2 (p, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)`
- `qchisqN (p, df, ncp = 0, qIni = qchisqAppr.0, ...)`

qnchisqAppr.2 (p, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)

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 \( \delta \); ....
- `lower.tail`, `log.p` logical, see, e.g., `qchisq()`.
- `qIni` a function that computes an approximate noncentral chi-squared quantile as starting value \( x_0 \) 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 `qnchisqPearson`.

- `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.
qchisqAbdelAty(): ...TODO...
qchisqBolKuz(): ...TODO...
qchisqPatnaik(): ...TODO...
qchisqPearson(): ...TODO...
qchisqSankaran_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

Chapter 29 Noncentral $\chi^2$-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~~~~~~~~~~~~~
ncp: ", 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 :
qpql <- cbind(qs, pqL, qchisq=pqql, qchA.0 = qchisqAppr.0(pqL, df=5, ncp=1, log.p=TRUE)
Approximations to 'qnorm()', i.e., \( z_{\alpha} \)

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 upper 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 \( P[X \leq x] \), 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 Upper 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 'Rmathlib').

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 |p| 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

```r
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, ..)"
)
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")
```

```r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 Rmathlib 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**

```r
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 ≥ yL`, where `y` is the initial approximate result, derived from a Cornish-Fisher expansion.

- `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`:
    - Defines the new increment from the old one as `incr <- max(1, floor(incr/iShrink))`.
    - Where the LHS was hardwired 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 hardwired 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

```r
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 ?
```
all.equal(x, qR, tol = 0)
stopifnot(all.equal(x, qR, 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 \( \lambda = \text{lambda} \), and

\[ e_n(x) := 1 + x + x^2/2! + \ldots + x^n/n! \]

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 \( \lambda \) 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 $r_X(i)$ values.
r_pois_expr() an expression.

Author(s)

Martin Maechler, 20 Jan 2004

See Also

dpois().

Examples

plRpois(12)
plRpois(120)
Index

* arith
  fr ld exp, 25
  logspace.add, 36
* character
  format01prec, 24
* distribution
  b ch1, 8
  dbinom raw, 11
dchisqApprox, 12
dgamma utils, 13
dgamma R, 17
dhyperBinMolenaar, 18
dnbincor, 19
dnt, 20
dPQ package, 3
dtWV, 22
hyper2binomP, 26
lgamma1p, 29
lgammaAsymp, 31
lssum, 37
pbetarv1, 48
phyperAllBin, 50
phyperApprAS152, 51
phyperBin, 52
phyperBinMolenaar, 53
phyperIbeta, 54
phyperMolenaar, 55
phyperPeizer, 56
phyper R, 58
phyperR2, 59
pnbeta, 62
pnchi1sq, 64
pnchi2Apppr, 67
pnchisqWienergerm, 71
pnormAsymp, 73
pnormLU, 74
pnt, 76
ppoisson, 79
qbetaApppr, 80
qbinomR, 82
qchisqApppr, 83
qgammaApppr, 85
qnbinomR, 86
qnchisqApppr, 88
qnormApppr, 90
qnormR, 92
qpoisR, 93
qtApppr, 95
r pois, 96
* hplot
  p12curves, 61
* math
  algdiv, 6
  b ch1, 8
  Bern, 7
dchisqApprox, 12
dgamma R, 17
dnt, 20
dPQ package, 3
dtWV, 22
lbeta, 27
lfastchoose, 29
lgamma1p, 29
lgammaAsymp, 31
log1mexp, 32
log1pmx, 33
logcf, 35
newton, 39
numer utils, 42
pbetarv1, 48
pnbeta, 62
pnchi1sq, 64
pnchisqWienergerm, 71
pnt, 76
* package
  DPQ package, 3
* print
  format01prec, 24
* utilities
  hyper2binomP, 26
.Machine, 43
.dntJKBch (dnt), 20
.dntJKBch1 (dnt), 20
.p11ser (p11), 44
.qgammaAppprBnd (qgammaApppr), 85
.suppHyper (phyperAllBin), 50
INDEX

algdiv, 6, 28
all, 43
all.equal, 20
all_mpfr (numer-utils), 42
any, 43
any_mpfr (numer-utils), 42
b_chi, 8, 78
b_chiAsymp (b_chi), 8
bd0, 11, 19, 44, 45
bd0 (dgamma-utils), 13
bd0_l1pm (dgamma-utils), 13
bd0_p1ld (dgamma-utils), 13
bd0_p1ldt (dgamma-utils), 13
bd0C (dgamma-utils), 13
Bern, 7, 31
Bernoulli, 8
besselI, 12
beta, 7, 27, 28
betaI, 12
betaM, 13
betaMM (beta), 27
c_dt (b_chi), 8
c_dtAsymp (b_chi), 8
c_pt (b_chi), 8
character, 15, 24, 72, 84, 92
class, 20, 42
curve, 61, 62
dbinom, 11, 15, 16, 19, 45
dbinom.raw, 11, 19
dchisq, 12, 13, 69, 89
dchisqApprox, 12
dchisqAsym (dchisqApprox), 12
dgamma, 13, 16, 17
dgamma-utils, 13
dgamma.R, 17
dhyper, 19
dhyperBinMolenaar, 18, 26
dnbinom, 19
dnbinom.mui (dnbinomR), 19
dnbinomR, 19
dnchisqBessel (dchisqApprox), 12
dnchisqR (dchisqApprox), 12
dnoncentchisq (dchisqApprox), 12
dnorm, 64
dt, 20
dtJKBF, 23
dtJKBF (dt), 20
dtJKBF1 (dt), 20
double, 35
dpois, 12, 16, 79, 97
dpois.raw, 17
dpois.raw (dgamma-utils), 13
DPQ (DPQ-package), 3
DPQ-package, 3
dt, 8, 20–23
dtWV, 21, 22
eaxis, 96
ebd0 (dgamma-utils), 13
ebd0C (dgamma-utils), 13
environment, 7
expression, 97
f051choose (lfastchoose), 29
format, 24
format.pval, 24
format1prec, 24
formatC, 24
fr_1d_exp, 25
fr_exp (fr_1d_exp), 25
function, 24, 33, 40, 61, 69, 81, 88
g2 (pnchisqWienergerm), 71
G_half (numer-utils), 42
gamma, 7, 8, 29
gnt (pnchisqWienergerm), 71
h (pnchisqWienergerm), 71
h0 (pnchisqWienergerm), 71
h1 (pnchisqWienergerm), 71
h2 (pnchisqWienergerm), 71
hnt (pnchisqWienergerm), 71
hyper2binomP, 18, 26
integer, 79
lb_chi (b_chi), 8
lb_chi0 (b_chi), 8
lb_chiAsymp (b_chi), 8
lbeta, 27, 28, 81
lbeta_asy (lbeta), 27
lbetaI (lbeta), 27
lbetaM (lbeta), 27
lbetaMM (lbeta), 27
lchoose, 29
ldexp (fr_1d_exp), 25
legend, 62
length, 31, 36
lfastchoose, 29
lgamma, 8, 29, 31
lgamma1p, 29, 34–36, 86
lgamma1p_series (lgamma1p), 29
lgamma1pC (lgamma1p), 29
lgammaAsymp, 31
lgammacorr (dgamma-utils), 13
lines, 96
list, 25, 40
log, 8, 11, 18, 29, 42
log1mexp, 25, 32, 32, 37
log1mexpC (log1mexp), 32
log1p, 30, 33, 34, 44
log1pexpC (log1mexp), 32
log1pmx, 14, 30, 33, 35, 36, 44
log1pmxC (log1pmx), 33
logcf, 14, 30, 33, 34, 35
logcfR (logcf), 35
logical, 15, 18, 27, 33, 37, 40, 43, 63, 69, 72, 77, 79, 81, 82, 96
logQab_asy, 7
logQab_asy (lbeta), 27
logr (numerator-utils), 42
logspace.add, 25, 36
logspace.sub (logspace.add), 36
lssum, 37, 37
lsun, 25, 37, 38, 38
M_cutoff (numerator-utils), 42
M_LN2 (numerator-utils), 42
M_minExp (numerator-utils), 42
M_SQRT2 (numerator-utils), 42
matrix, 50, 60
max, 37
modf (numerator-utils), 42
mpfr, 20, 33, 73
newton, 39, 88
numerator-utils, 42
numeric, 7, 14, 35, 37, 51, 72, 89
okLongDouble (numerator-utils), 42
p11, 44
p11p (p11), 44
p11ser (p11), 44
pbeta, 6, 28, 30, 36, 48, 49, 54, 63
pbetaRv1, 48
pbinnom, 26, 53, 54
pchi3sq, 65, 67, 69–73, 89
pchi3sq (pchi3sqWienergerm), 71
pchi3sqW, 65, 70
pchi3sqW (pchi3sqWienergerm), 71
pdi hyper (phyperR2), 59
pgamma, 37
phyper, 26, 50–58, 60, 61
phyper1molenaar (phyperMolenaar), 55
phyper2molenaar (phyperMolenaar), 55
phyperAllBin, 50
phyperAllBinM (phyperAllBin), 50
phyperApprAS152, 51
phyperBin, 52
phyperBin.1, 50
phyperBinMolenaar, 50, 53
phyperIbeta, 54
phyperMolenaar, 55
phyperPeizer, 56
phyperR, 58
phyperR2, 59
phypers, 60
p12curves, 61
plot.default, 96
plRpois (r_pois), 96
pnbeta, 62
pnbetaAppr2 (pnbeta), 62
pnbetaAppr2v1 (pnbeta), 62
pnbetaAS310 (pnbeta), 62
pnbetaC, 64
pnbetaAS (pnbetaAppr), 64
pnbetaAppr, 67
pnbetaAdelAty (pnbetaAppr), 67
pnbetaAppr, 67
pnbetaHeadKuz (pnbetaAppr), 67
pnbetaHit (pnbetaAppr), 67
pnbetaHitnaik (pnbetaAppr), 67
pnbetaPearson, 65, 88
pnbetaPearson (pnbetaAppr), 67
pnbetaRC (pnbetaAppr), 67
pnbetaSankaran_d (pnbetaAppr), 67
pnbetaSankaran (pnbetaAppr), 67
pnbetaTerms (pnbetaAppr), 67
pnbetaV (pnbetaAppr), 67
pnbetaWienergerm, 71
pnorm, 51, 56, 64, 67, 73–75, 77, 78
pnormAsymp, 73
pnormL_LD10 (pnormLU), 74
pnormLU, 74
pnormU_S53, 74
pnormU_S53 (pnormLU), 74
pnt, 76
pnt3150, 76
pnt3150 (pnt), 76
pntChShP94 (pnt), 76
pntJW39, 8
pntJW39 (pnt), 76
pntLrg (pnt), 76
pntP94 (pnt), 76
pntR (pnt), 76
pntR1 (pnt), 76
ppois, 79, 80
ppoisD (poisson), 79
ppoisErr (poisson), 79
Rmpfr, 6
scalefactor (pnchisqWienergerm), 71