Package ‘nimbleNoBounds’

February 9, 2023

Type Package
Title Transformed Distributions for Improved MCMC Efficiency
Version 1.0.2
Encoding UTF-8
Date 2023-02-08
Maintainer David Pleydell <david.pleydell@inrae.fr>
Author David Pleydell [aut, cre, cph] (Package developer, <https://orcid.org/0000-0002-6450-1475>)
URL https://github.com/DRJP/nimbleNoBounds
Description A collection of common univariate bounded probability distributions transformed to the unbounded real line, for the purpose of increased MCMC efficiency.
License BSD_3_clause + file LICENSE
Copyright See COPYRIGHTS file.
RoxygenNote 7.2.0
Depends R (>= 3.1.2), nimble
Imports methods
Suggests knitr, rmarkdown, coda
VignetteBuilder knitr
NeedsCompilation no
Repository CRAN
Date/Publication 2023-02-09 10:20:02 UTC

R topics documented:

dLogChisq ............................................................... 2
dLogExp ............................................................... 3
dLogGamma ............................................................ 5
dLogHalfflat ........................................................... 7
dLogInvgamma ......................................................... 8
dLogChisq

Log transformed chi-squared distribution.

dLogChisq and rLogChisq provide a log-transformed chi-squared distribution.

Usage

dLogChisq(x, df = 1, log = 0)
rLogChisq(n = 1, df = 1)

Arguments

- `x`: A continuous random variable on the real line, where \( y = \exp(x) \) and \( y \sim \text{dchisq}(df) \).
- `df`: \( df \) parameter of \( y \sim \text{dchisq}(df) \).
- `log`: Logical flag to toggle returning the log density.
- `n`: Number of random variables. Currently limited to 1, as is common in nimble. See `?replicate` for an alternative.

Value

The density or log density of \( x \), such that \( x = \log(y) \) and \( y \sim \text{dchisq}(df) \).

Author(s)

David R.J. Pleydell

Examples

```r
## Create a Chi-squared random variable, and transform it to the log scale
n = 100000
df = 3
y = rchisq(n=n, df=df)
x = log(y)

## Plot histograms of the two random variables
oldpar <- par()
```
Log transformed exponential distribution.

dLogExp

**Description**

`dLogExp` and `rLogExp` provide a log-transformed exponential distribution.
Usage

dLogExp(x, rate = 1, log = 0)

rLogExp(n = 1, rate = 1)

Arguments

x A continuous random variable on the real line. Let y = exp(x). Then y ~ dexp(rate).
rate Rate parameter of y ~ dexp(rate).
log Logical flag to toggle returning the log density.
n Number of random variables. Currently limited to 1, as is common in nimble.
See ?replicate for an alternative.

Value

The density or log density of x, such that x = log(y) and y ~ dexp(rate).

Author(s)

David R.J. Pleydell

Examples

## Create a exponential random variable, and transform it to the log scale
n = 100000
lambda = 3
y = rexp(n=n, rate=lambda)
x = log(y)

## Plot histograms of the two random variables
oldpar <- par()
par(mfrow=n2mfrow(2))
## Plot 1
hist(x, n=100, freq=FALSE)
curve(dLogExp(x, rate=0.5), -15, 4, n=1001, col="red", add=TRUE, lwd=3)
## Plot 2: back-transformed
xNew = replicate(n=n, rLogExp(n=1, rate=lambda))
yNew = exp(xNew)
hist(yNew, n=100, freq=FALSE, xlab="exp(x)"
curve(dexp(x, rate=lambda), 0, 4, n=1001, col="red", lwd=3, add=TRUE)
par(oldpar)

## Create a NIMBLE model that uses this distribution
code = nimbleCode(
  x ~ dLogExp(rate = 0.5)
  y <- exp(x)
)

\textit{dLogExp}
## Build & compile the model
modelR = nimbleModel(code=code)
modelC = compileNimble(modelR)

## Configure, build and compile an MCMC
conf = configureMCMC(modelC, monitors=c("x","y"))
mcmc = buildMCMC(conf=conf)
cMcmc = compileNimble(mcmc)

## Run the MCMC
samps = runMCMC(mcmc=cMcmc, niter=50000)
x = samps[,"x"]
y = samps[,"y"]

## Plot MCMC output
oldpar <- par()
par(mfrow=n2mfrow(3))
## Plot 1: MCMC trajectory
plot(x, typ="l")
## Plot 2: target density on unbounded sampling scale
hist(x, n=100, freq=FALSE)
curve(dLogExp(x, rate=0.5), -15, 5, n=1001, col="red", lwd=3, add=TRUE)
## Plot 3: target density on bounded scale
hist(y, n=100, freq=FALSE)
curve(dexp(x, rate=0.5), 0, 25, n=1001, col="red", lwd=3, add=TRUE)
par(oldpar)

---

dLogGamma

Log transformed gamma distribution.

**Description**

dLogGamma and rLogGamma provide a log-transformed gamma distribution.

**Usage**

dLogGamma(x, shape = 1, scale = 1, log = 0)
rLogGamma(n = 1, shape = 1, scale = 1)

**Arguments**

- **x**: A continuous random variable on the real line. Let y=exp(x). Then y ~ dgamma(shape,scale).
- **shape**: Shape parameter of y ~ dgamma(shape,scale).
- **scale**: Scale parameter of y ~ dgamma(shape,scale).
- **log**: Logical flag to toggle returning the log density.
- **n**: Number of random variables. Currently limited to 1, as is common in nimble. See ?replicate for an alternative.
The density or log density of x, such that x = log(y) and y ~ dgamma(shape, scale).

Author(s)
David R.J. Pleydell

Examples

```r
## Create a gamma random variable, and transform it to the log scale
n = 100000
shape = 2
scale = 2
y = rgamma(n=n, shape=shape, scale=scale)
x = log(y)

## Plot histograms of the two random variables
oldpar <- par()
par(mfrow=n2mfrow(2))
## Plot 1
hist(x, n=100, freq=FALSE)
curve(dLogGamma(x, shape=shape, scale=scale), -4, 5, n=1001, col="red", add=TRUE, lwd=3)
## Plot 2: back-transformed
xNew = replicate(n=n, rLogGamma(n=1, shape=shape, scale=scale))
yNew = exp(xNew)
hist(yNew, n=100, freq=FALSE, xlab="exp(x)")
curve(dgamma(x, shape=shape, scale=scale), 0, 100, n=1001, col="red", lwd=3, add=TRUE)
par(oldpar)

## Create a NIMBLE model that uses this distribution
code = nimbleCode(
  log(y) ~ dLogGamma(shape=shape, scale=scale)
  log(y2) ~ dLogGamma(shape=shape, rate=1/scale)
  log(y3) ~ dLogGamma(mean=shape*scale, sd=scale * sqrt(shape))
)

## Build & compile the model
const = list (shape=shape, scale=scale)
modelR = nimbleModel(code=code, const=const)
simulate(modelR)
modelC = compileNimble(modelR)

## Configure, build and compile an MCMC
conf = configureMCMC(modelC, monitors2=c("y", "y2", "y3"))
mcmc = buildMCMC(conf=conf)
cMcmc = compileNimble(mcmc)

## Run the MCMC & extract samples
samps = runMCMC(mcmc=cMcmc, niter=50000)
x = as.vector(samps[[1]][,"log_y"])```
\texttt{dLogHalfflat

\begin{verbatim}
x2 = as.vector(samps[[1]][,"log_y2"])
x3 = as.vector(samps[[1]][,"log_y3"])
y = as.vector(samps[[2]][,"y"])
y2 = as.vector(samps[[2]][,"y2"])
y3 = as.vector(samps[[2]][,"y3"])

## Plot MCMC output
oldpar <- par()
par(mfrow=n2mfrow(4))
## Plot 1: MCMC trajectory
plot(x, typ="l")
## Plot 2: taget density on unbounded sampling scale
hist(x, n=100, freq=FALSE)
curve(dLogGamma(x, shape=shape, scale=scale), -4, 3, n=1001, col="red", lwd=3, add=TRUE)
## Plot 3: taget density on bounded scale
hist(y, n=100, freq=FALSE)
curve(dgamma(x, shape=shape, scale=scale), 0, 25, n=1001, col="red", lwd=3, add=TRUE)
## Plot 4: different parameterisations
nBreaks=51
xLims = range(pretty(range(samps[[1]]))))
hist(x, breaks=seq(xLims[1], xLims[2], l=nBreaks), col=rgb(1, 0, 0, 0.1))
hist(x2, breaks=seq(xLims[1], xLims[2], l=nBreaks), col=rgb(0, 1, 0, 0.1), add=TRUE)
hist(x3, breaks=seq(xLims[1], xLims[2], l=nBreaks), col=rgb(0, 0, 1, 0.1), add=TRUE)
par(oldpar)
\end{verbatim

\textbf{dLogHalfflat

Log transformed half-flat distribution.}

\textbf{Description}

dLogHalfflat and \texttt{rLogHalfflat} provide a log-transformed half-flat distribution. Note, both dhalfflat and dLogHalfflat are improper. Thus, rLogHalfflat returns NAN, just as rhalfflat does.

\textbf{Usage}

dLogHalfflat(x, log = 0)

rLogHalfflat(n = 1)

\textbf{Arguments}

\begin{itemize}
  \item \texttt{x} \hspace{1cm} A continuous random variable on the real line, where \(y=\exp(x)\) and \(y \sim \text{dhalfflat}()\).
  \item \texttt{log} \hspace{1cm} Logical flag to toggle returning the log density.
  \item \texttt{n} \hspace{1cm} Number of random variables. Currently limited to 1, as is common in nimble. See \texttt{?replicate} for an alternative. Note, NAN will be returned because distribution is improper.
\end{itemize}
**dLogInvgamma**

**Value**

A value proportional to the density, or the log of that value, of \( x \), such that \( x = \log(y) \) and \( y \sim dhalfflat() \).

**Author(s)**

David R.J. Pleydell

**Examples**

```r
oldpar <- par()
par(mfrow=n2mfrow(2))
## Plot 1
curve(dhalfflat(x), -11, 11, n=1001, col="red", lwd=3, xlab="y = exp(x)", ylab="dhalfflat(y)"
## Plot 2: back-transformed
curve(dLogInvgamma(x), -5, 1.5, n=1001, col="red", lwd=3, xlab="x = log(y)
abline(v=0:1, h=c(0,1,exp(1)), col="grey"
par(oldpar)
```

---

**dLogInvgamma**

Log transformed inverse-gamma distribution.

**Description**

\( dLogInvgamma \) and \( rLogInvgamma \) provide a log-transformed inverse gamma distribution.

**Usage**

```r
dLogInvgamma(x, shape, scale = 1, log = 0)
rLogInvgamma(n = 1, shape, scale = 1)
```

**Arguments**

- **x**
  - A continuous random variable on the real line. Let \( y = \exp(x) \). Then \( y \sim dinvgamma(shape, scale) \).
- **shape**
  - Shape parameter of \( y \sim dinvgamma(shape, scale) \).
- **scale**
  - Scale parameter of \( y \sim dinvgamma(shape, scale) \).
- **log**
  - Logical flag to toggle returning the log density.
- **n**
  - Number of random variables. Currently limited to 1, as is common in nimble. See `?replicate` for an alternative.

**Value**

The density or log density of \( x \), such that \( x = \log(y) \) and \( y \sim dinvgamma(shape, scale) \).
Examples

```r
## Create an inverse gamma random variable, and transform it to the log scale
n   = 100000
shape = 2.5
scale = 0.01
y    = rinvgamma(n=n, shape=shape, scale=scale)
x    = log(y)

## Plot histograms of the two random variables
oldpar <- par()
par(mfrow=n2mfrow(4))
## Plot 1
hist(y, n=100, freq=FALSE, xlab="y")
curve(dinvgamma(x, shape=shape, scale=scale), 0, 1.0, n=5001, col="red", add=TRUE, lwd=2)
## Plot 2
hist(x, n=100, freq=FALSE)
curve(dLogInvgamma(x, shape=shape, scale=scale), -8, 1, n=1001, col="red", add=TRUE, lwd=3)
## Plot 3: back-transformed
z    = rgamma(n=n, shape=shape, scale=1/scale)
hist(1/z, n=100, freq=FALSE, xlab="y")
curve(dinvgamma(x, shape=shape, scale=scale), 0, 1, n=5001, col="red", lwd=3, add=TRUE)
## Plot 4: back-transformed
xNew  = replicate(n=n, rLogInvgamma(n=1, shape=shape, scale=scale))
yNew  = exp(xNew)
hist(yNew, n=100, freq=FALSE, xlab="exp(x)")
curve(dinvgamma(x, shape=shape, scale=scale), 0, 1, n=5001, col="red", lwd=3, add=TRUE)
par(oldpar)

## Create a NIMBLE model that uses this transformed distribution
code = nimbleCode({
  log(y) ~ dLogInvgamma(shape=shape, scale=scale)
})

## Build & compile the model
const = list(shape=shape, scale=scale)
modelR = nimbleModel(code=code, const=const)
simulate(modelR)
modelC = compileNimble(modelR)

## Configure, build and compile an MCMC
conf  = configureMCMC(modelC, monitors=c("log_y", "y"))
mcmc  = buildMCMC(conf=conf)
cMcmc = compileNimble(mcmc)

## Run the MCMC
samps = runMCMC(mcmc=cMcmc, niter=50000)
```
## Plot MCMC output
```r
oldpar <- par()
par(mfrow=n2mfrow(3))
## Plot 1: MCMC trajectory
plot(x, typ="l")
## Plot 2: target density on unbounded sampling scale
hist(x, n=100, freq=FALSE)
curve(dLogInvgamma(x, shape=shape, scale=scale), -10, 1, n=1001, col="red", lwd=3, add=TRUE)
## Plot 3: target density on bounded scale
curve(dinvgamma(x, shape=shape, scale=scale), xlab="y = exp(x)", 0, 0.5, n=1001, col="red", lwd=3)
hist(y, n=100, freq=FALSE, add=TRUE)
curve(dinvgamma(x, shape=shape, scale=scale), 0, 0.5, n=1001, col="red", add=TRUE, lwd=3)
par(oldpar)
```
Examples

```r
## Create a beta random variable, and transform it to the logit scale
n = 100000
sh1 = 1
sh2 = 11
y = rbeta(n=n, sh1, sh2)
x = logit(y)

## Plot histograms of the two random variables
oldpar <- par()
par(mfrow=n2mfrow(2))
## Plot 1
hist(x, n=100, freq=FALSE)
curve(dLogitBeta(x, sh1, sh2), -15, 4, n=1001, col="red", add=TRUE, lwd=3)
## Plot 2: back-transformed
xNew = replicate(n=n, rLogitBeta(n=1, sh1, sh2))
yNew = ilogit(xNew)
hist(yNew, n=100, freq=FALSE, xlab="exp(x)"
curve(dbeta(x, sh1, sh2), 0, 1, n=1001, col="red", lwd=3, add=TRUE)
par(oldpar)

## Create a NIMBLE model that uses this transformed distribution
code = nimbleCode(
  x ~ dLogitBeta(sh1, sh2)
)
## Build & compile the model
const = list(sh1=sh1, sh2=sh2)
modelR = nimbleModel(code=code, const=const)
modelC = compileNimble(modelR)

## Configure, build and compile an MCMC
conf = configureMCMC(modelC)
mcmc = buildMCMC(conf=conf)
cMcmc = compileNimble(mcmc)

## Run the MCMC
x = as.vector(runMCMC(mcmc=cMcmc, niter=50000))

## Plot MCMC output
oldpar <- par()
par(mfrow=n2mfrow(3))
## Plot 1: MCMC trajectory
plot(x, typ="l")
## Plot 2: target density on unbounded sampling scale
hist(x, n=100, freq=FALSE, xlab="x = logit(y)"
curve(dLogitBeta(x, sh1, sh2), -15, 5, n=1001, col="red", lwd=3, add=TRUE)
## Plot 3: target density on bounded scale
hist(ilogit(x), n=100, freq=FALSE, xlab="y"
curve(dbeta(x, sh1, sh2), 0, 1, n=1001, col="red", lwd=3, add=TRUE)
```
Description
Transformation of uniform distribution, via scaled-logit transform, to the real line.

Usage
dLogitUnif(x, min = 0, max = 1, log = 0)
rLogitUnif(n = 1, min = 0, max = 1)

Arguments
x A continuous random variable on the real line, where y=ilogit(x)*(max-min)+min and y ~ dunif(min, max).
min lower limit of the distribution. Must be finite.
max upper limit of the distribution. Must be finite.
log logical flag. Returns log-density if TRUE.
n Number of random variables. Currently limited to 1, as is common in nimble. See ?replicate for an alternative.

Value
density, or log-density, of uniform distribution transformed to real line via scaling and logit function.

Author(s)
David R.J. Pleydell

Examples

## Create a uniform random variable, and transform it to the log scale
n = 100000
lower = -5
upper = 11
y = runif(n=n, lower, upper)
x = logit((y-lower)/(upper-lower))

## Plot histograms of the two random variables
oldpar <- par()
par(mfrow=n2mfrow(2))
## Plot 1
dLogLnorm

hist(x, n=100, freq=FALSE)
curve(dLogitUnif(x, lower, upper), -15, 15, n=1001, col="red", add=TRUE, lwd=3)
## Plot 2: back-transformed
xNew = replicate(n=n, rLogitUnif(n=1, lower, upper))
yNew = ilogit(xNew) * (upper-lower) + lower
hist(yNew, n=100, freq=FALSE, xlab="exp(x)")
curve(dunif(x, lower, upper), -15, 15, n=1001, col="red", lwd=3, add=TRUE)
par(oldpar)

## Create a NIMBLE model that uses this transformed distribution
code = nimbleCode(
  x ~ dLogitUnif(lower, upper)
)

## Build & compile the model
const = list(lower=lower, upper=upper)
modelR = nimbleModel(code=code, const=const)
modelC = compileNimble(modelR)

## Configure, build and compile an MCMC
conf = configureMCMC(modelC)
mcmc = buildMCMC(conf=conf)
cMcmc = compileNimble(mcmc)

## Run the MCMC
x = as.vector(runMCMC(mcmc=cMcmc, niter=50000))

## Plot MCMC output
oldpar <- par()
par(mfrow=n2mfrow(3))
## Plot 1: MCMC trajectory
plot(x, typ="l")
## Plot 2: taget density on unbounded sampling scale
hist(x, n=100, freq=FALSE, xlab="x = logit(y)")
curve(dLogitUnif(x, lower, upper), -15, 15, n=1001, col="red", lwd=3, add=TRUE)
## Plot 3: taget density on bounded scale
hist(ilogit(x)*(upper-lower)+lower, n=100, freq=FALSE, xlab="y")
curve(dunif(x, lower, upper), -15, 15, n=1001, col="red", lwd=3, add=TRUE)
par(oldpar)

---

dLogLnorm  

Log transformed log-normal distribution.

Description

dLogLnorm and rLogLnorm provide a log-transformed log-normal distribution.
Usage

dLogLnorm(x, meanlog = 0, sdlog = 1, log = 0)
rLogLnorm(n = 1, meanlog = 0, sdlog = 1)

Arguments

x A continuous random variable on the real line. Let \( y = \exp(x) \). Then \( y \sim \text{dlnorm}(\text{meanlog}, \text{sdlog}) \).

meanlog mean of the distribution on the log scale with default values of ‘0’.

sdlog standard deviation of the distribution on the log scale with default values of ‘1’.

log Logical flag to toggle returning the log density.

n Number of random variables. Currently limited to 1, as is common in nimble. See ?replicate for an alternative.

Value

The density or log density of \( x \), such that \( x = \log(y) \) and \( y \sim \text{dlnorm}(\text{meanlog}, \text{sdlog}) \).

Author(s)

David R.J. Pleydell

Examples

```r
## Create a log-normal random variable, and transform it to the log scale
n = 100000
meanlog = -3
sdlog = 0.1
y = rlnorm(n=n, meanlog=meanlog, sdlog=sdlog)
x = log(y)

## Plot histograms of the two random variables
oldpar <- par()
par(mfrow=n2mfrow(2))
## Plot 1
hist(x, n=100, freq=FALSE)
curve(dLogLnorm(x, meanlog=meanlog, sdlog=sdlog), -4, -2, n=1001, col="red", add=TRUE, lwd=3)
## Plot 2: back-transformed
xNew = replicate(n=n, rLogLnorm(n=1, meanlog=meanlog, sdlog=sdlog))
yNew = exp(xNew)
hist(yNew, n=100, freq=FALSE, xlab="exp(x)")
curve(dlnorm(x, meanlog=meanlog, sdlog=sdlog), 0, 0.1, n=1001, col="red", lwd=3, add=TRUE)
par(oldpar)

## Create a NIMBLE model that uses this transformed distribution
code = nimbleCode(
    log(y) ~ dLogLnorm(meanlog=meanlog, sdlog=sdlog)
)
```
## Build & compile the model

```r
const = list(meanlog=meanlog, sdlog=sdlog)
modelR = nimbleModel(code=code, const=const)
simulate(modelR)
modelC = compileNimble(modelR)
```

## Configure, build and compile an MCMC

```r
cnf = configureMCMC(modelC)
mcmc = buildMCMC(conf=conf)
cMcmc = compileNimble(mcmc)
```

## Run the MCMC

```r
x = as.vector(runMCMC(mcmc=cMcmc, niter=50000))
y = exp(x)
```

## Plot MCMC output

```r
oldpar <- par()
par(mfrow=n2mfrow(3))
## Plot 1: MCMC trajectory
plot(x, typ="l")
## Plot 2: target density on unbounded sampling scale
hist(x, n=100, freq=FALSE)
curve(dLogLnorm(x, meanlog=meanlog, sdlog=sdlog), -4, -2, n=1001, col="red", lwd=3, add=TRUE)
## Plot 3: target density on bounded scale
hist(y, n=100, freq=FALSE)
curve(dlnorm(x, meanlog=meanlog, sdlog=sdlog), 0, 0.1, n=1001, col="red", lwd=3, add=TRUE)
par(oldpar)
```

---

**dLogWeib**

*Log transformed Weibull distribution.*

---

**Description**

*dLogWeib* and *rLogWeib* provide a log-transformed Weibull distribution.

**Usage**

```r
dLogWeib(x, shape = 1, scale = 1, log = 0)
rLogWeib(n = 1, shape = 1, scale = 1)
```

**Arguments**

- **x**
  
  A continuous random variable on the real line, where \(y = \exp(x)\) and \(y \sim \text{dweib}(\text{shape}, \text{scale})\).

- **shape**
  
  Shape parameter of \(y \sim \text{dweib}(\text{shape}, \text{scale})\).

- **scale**
  
  Scale parameter of \(y \sim \text{dweib}(\text{shape}, \text{scale})\).
log

Logical flag to toggle returning the log density.

n

Number of random variables. Currently limited to 1, as is common in nimble. See ?replicate for an alternative.

Value

The density or log density of x, such that x = log(y) and y ~ dweib(shape, scale).

Author(s)

David R.J. Pleydell

Examples

```r
## Create a Weibull random variable, and transform it to the log scale
n = 100000
shape = 2
scale = 2
y = rweibull(n=n, shape=shape, scale=scale)
x = log(y)

## Plot histograms of the two random variables
oldpar <- par()
par(mfrow=n2mfrow(2))
## Plot 1
hist(x, n=100, freq=FALSE, xlab="x = log(y)",
    main="Histogram of log-transformed random numbers from rweibull.")
curve(dLogWeib(x, shape=shape, scale=scale), -6, 3, n=1001, col="red", add=TRUE, lwd=3)
## Plot 2: back-transformed
xNew = replicate(n=n, rLogWeib(n=1, shape=shape, scale=scale))
yNew = exp(xNew)
hist(yNew, n=100, freq=FALSE, xlab="y = exp(x)", main="Histogram of random numbers from rLogWeib.")
curve(dweibull(x, shape=shape, scale=scale), 0, 100, n=1001, col="red", lwd=3, add=TRUE)
par(oldpar)

## Create a NIMBLE model that uses this transformed distribution
code = nimbleCode(
  log(y) ~ dLogWeib(shape=shape, scale=scale)
)

## Build & compile the model
const = list (shape=shape, scale=scale)
modelR = nimbleModel(code=code, const=const)
simulate(modelR)
modelC = compileNimble(modelR)

## Configure, build and compile an MCMC
conf = configureMCMC(modelC)
mcmc = buildMCMC(conf=conf)
cMcmc = compileNimble(mcmc)
```
Efficiency gain estimates from vignette example

Description

The toy example in the vignette provides a simple Monte Carlo experiment that tests the gain in sampling efficiency when switching to sampling on unbounded scales (i.e. the real line). Since that Monte Carlo experiment takes 10 minutes to run, 'efficiencyGain' is provided as an example set of output data - providing an alternative to re-running the Monte Carlo code.

Format

A data frame with 111 rows (number of Monte Carlo replicates) and 8 variables

- beta  Efficiency gain when sampling a beta distribution on the real line.
- chisq  Efficiency gain when sampling a chi-squared distribution on the real line.
- exp    Efficiency gain when sampling an exponential distribution on the real line.
- gamma  Efficiency gain when sampling a gamma distribution on the real line.
- invgamma Efficiency gain when sampling an inverse-gamma distribution on the real line.
- lnorm  Efficiency gain when sampling a log-normal distribution on the real line.
- unif   Efficiency gain when sampling a uniform distribution on the real line.
- weib  Efficiency gain when sampling a Weibull distribution on the real line.

Author(s)

David Pleydell
Source

See `vignette("nimbleNoBounds")`.

Description

A collection of NIMBLE functions for sampling common probability distributions on unbounded scales.

Transformed probability distributions for unbounded sampling in NIMBLE.

NA

See Also

https://r-nimble.org/
Index

* data
  
  - efficiencyGain, 17

  dLogChisq, 2
  dLogExp, 3
  dLogGamma, 5
  dLogHalfflat, 7
  dLogInvgamma, 8
  dLogitBeta, 10
  dLogitUnif, 12
  dLogLnorm, 13
  dLogWeib, 15

  efficiencyGain, 17

  nimbleNoBounds, 18

  rLogChisq (dLogChisq), 2
  rLogExp (dLogExp), 3
  rLogGamma (dLogGamma), 5
  rLogHalfflat (dLogHalfflat), 7
  rLogInvgamma (dLogInvgamma), 8
  rLogitBeta (dLogitBeta), 10
  rLogitUnif (dLogitUnif), 12
  rLogLnorm (dLogLnorm), 13
  rLogWeib (dLogWeib), 15