Package: simEd (via r-universe)

November 4, 2024

Title Simulation Education

Version 2.0.1

Imports graphics, grDevices, methods, stats, utils, shape

Depends rstream

Suggests magick

Description Contains various functions to be used for simulation education, including simple Monte Carlo simulation functions, queueing simulation functions, variate generation functions capable of producing independent streams and antithetic variates, functions for illustrating random variate generation for various discrete and continuous distributions, and functions to compute time-persistent statistics. Also contains functions for visualizing: event-driven details of a single-server queue model; a Lehmer random number generator; variate generation via acceptance-rejection; and of generating a non-homogeneous Poisson process via thinning. Also contains two queueing data sets (one fabricated, one real-world) to facilitate input modeling. More details on the use of these functions can be found in Lawson and Leemis (2015) <doi:10.1109/WSC.2017.8248124>, in Kudlay, Lawson, and Leemis (2020) <doi:10.1109/WSC48552.2020.9384010>, and in Lawson and Leemis (2021) <doi:10.1109/WSC52266.2021.9715299>.

License MIT + file LICENSE

LazyData true

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Repository https://blawson-bates.r-universe.dev

RemoteUrl https://github.com/blawson-bates/simed

RemoteRef HEAD

RemoteSha b591e3b03cb84ab791db9774f69a48f047fcc665

Contents

---

simEd-package *Simulation Education*

Description

Contains various functions to be used for simulation education, including simple Monte Carlo simulation functions, queueing simulation functions, variate generation functions capable of producing independent streams and antithetic variates, functions for illustrating random variate generation for various discrete and continuous distributions, and functions to compute time-persistent statistics. Also contains functions for visualizing: event-driven details of a single-server queue model; a Lehmer random number generator; variate generation via acceptance-rejection; and of generating a non-homogeneous Poisson process via thinning. Also contains two queueing data sets (one fabricated, one real-world) to facilitate input modeling. More details on the use of these functions can be found in Lawson and Leemis (2015) <doi:10.1109/WSC.2017.8248124>, in Kudlay, Lawson, and Leemis (2020) <doi:10.1109/WSC48552.2020.9384010>, and in Lawson and Leemis (2021) <doi:10.1109/WSC52266.2021.9715299>.

Request From Authors: If you adopt and use this package for your simulation course, we would greatly appreciate were you to email us (addresses below) to let us know, as we would like to maintain a list of adopters. Please include your name, university/affiliation, and course name/number. Thanks!

Details

The goal of this package is to facilitate use of R for an introductory course in discrete-event simulation.

This package contains animation functions for visualizing:

- event-driven details of a single-server queue model (`ssqvis`);
- a Lehmer random number generator (`lehmer`);
- variate generation via acceptance-rejection (`accrej`);
- generation of a non-homogeneous Poisson process via thinning (`thinning`).

The package contains variate generators capable of independent streams (based on Josef Leydold's `rstream` package) and antithetic variates for four discrete and eleven continuous distributions:

- discrete: `vbinom`, `vgeom`, `vnbinom`, `vpois`
- continuous: `vbeta`, `vcauchy`, `vchisq`, `vexp`, `vgamma`, `vlnorm`, `vlogis`, `vnorm`, `vt`, `vunif`, `vweibull`

All of the variate generators use inversion, and are therefore monotone and synchronized.

The package contains functions to visualize variate generation for the same four discrete and eleven continuous distributions:

- discrete: ibinom, igeom, inbinom, ipois
- continuous: ibeta, icauchy, ichisq, iexp, igamma, ilnorm, ilogis, inorm, it, iunif, iweibull

The package also contains functions that are event-driven simulation implementations of a single-server single-queue system and of a multiple-server single-queue system:

- single-server: ssq
- multiple-server: msq

Both queueing functions are extensible in allowing the user to provide custom arrival and service process functions. As of version 2.0.0, both of these functions provide animation capability.

The package contains functions that implement Monte Carlo simulation approaches for estimating probabilities in two different dice games:

- Galileo's dice problem: galileo
- craps: craps

The package contains three functions for computing time-persistent statistics:

- time-average mean: meanTPS
- time-average standard deviation: sdTPS
- time-average quantiles: quantileTPS

The package also masks two functions from the stats package:

- set.seed, which explicitly calls the stats version in addition to setting up seeds for the independent streams in the package;
- sample, which provides capability to use independent streams and antithetic variates.

Finally, the package provides two queueing data sets to facilitate input modeling:

- queueTrace, which contains 1000 arrival times and 1000 service times (all fabricated) for a single-server queueing system;
- tylersGrill, which contains 1434 arrival times and 110 (sampled) service times corresponding to actual data collected during one business day at Tyler's Grill at the University of Richmond.

Acknowledgments

The authors would like to thank Dr. Barry L. Nelson, Walter P. Murphy Professor in the Department of Industrial Engineering & Management Sciences at Northwestern University, for meaningful feedback during the development of this package.

Author(s)

Barry Lawson [aut, cre, cph], Larry Leemis [aut], Vadim Kudlay [aut] Maintainer: Barry Lawson <blawson@bates.edu>

accrej *Acceptance-Rejection Algorithm Visualization*

Description

This function animates the process of generating variates via acceptance-rejection for a specified density function (pdf) bounded by a specified majorizing function.

Usage

```
accrej(
  n = 20,
  pdf = function(x) dbeta(x, 3, 2),
  majorizingFcn = NULL,
  majorizingFcnType = NULL,
  support = c(0, 1),
  seed = NA,
  maxTrials = Inf,
  plot = TRUE,
  showTitle = TRUE,
  plotDelay = plot * -1
)
```

Arguments

`n`	number of variates to generate.
`pdf`	desired probability density function from which random variates are to be drawn
`majorizingFcn`	majorizing function. Default value is NULL, corresponding to a constant majorizing function that is 1.01 times the maximum value of the pdf. May alternatively be provided as a user-specified function, or as a data frame requiring additional notation as either piecewise-constant or piecewise-linear. See examples.
`majorizingFcnType`	used to indicate whether a majorizing function that is provided via data frame is to be interpreted as either piecewise-constant (`"pwc"`) or piecewise-linear (`"pwl"`). If the majorizing function is either the default or a user-specified function (closure), the value of this parameter is ignored.
`support`	the lower and upper bounds of the support of the probability distribution of interest, specified as a two-element vector.
`seed`	initial seed for the uniform variates used during generation.
`maxTrials`	maximum number of accept-reject trials; infinite by default
`plot`	if TRUE, visual display will be produced. If FALSE, generated variates will be returned without visual display.
`showTitle`	if TRUE, display title in the main plot.
`plotDelay`	wait time, in seconds, between plots; -1 (default) for interactive mode, where the user is queried for input to progress.

Details

There are three modes for visualizing the acceptance-rejection algorithm for generating random variates from a particular probability distribution:

- interactive advance (`plotDelay` = `-1`), where pressing the 'ENTER' key advances to the next step (an accepted random variate) in the algorithm, typing 'j #' jumps ahead # steps, typing 'q' quits immediately, and typing 'e' proceeds to the end;
- automatic advance (`plotDelay` > 0); or
- final visualization only (`plotDelay` = `0`).

As an alternative to visualizing, variates can be generated

Value

Returns the n generated variates accepted.

Examples

```
accrej(n = 20, seed = 8675309, plotDelay = 0)
accrej(n = 10, seed = 8675309, plotDelay = 0.1)

# interactive mode
if (interactive()) {
  accrej(n = 10, seed = 8675309, plotDelay = -1)
}

# Piecewise-constant majorizing function
m <- function(x) {
  if      (x < 0.3)  1.0
  else if (x < 0.85) 2.5
  else               1.5
}
accrej(n = 10, seed = 8675309, majorizingFcn = m, plotDelay = 0)

# Piecewise-constant majorizing function as data frame
m <- data.frame(
  x = c(0.0, 0.3, 0.85, 1.0),
  y = c(1.0, 1.0, 2.5,  1.5))
accrej(n = 10, seed = 8675309, majorizingFcn = m,
       majorizingFcnType = "pwc", plotDelay = 0)

# Piecewise-linear majorizing function as data frame
m <- data.frame(
   x = c(0.0, 0.1, 0.3, 0.5, 0.7, 1.0),
   y = c(0.0, 0.5, 1.1, 2.2, 1.9, 1.0))
accrej(n = 10, seed = 8675309, majorizingFcn = m,
       majorizingFcnType = "pwl", plotDelay = 0)

# invalid majorizing function; should give warning
try(accrej(n = 20, majorizingFcn = function(x) dbeta(x, 1, 3), plotDelay = 0))
```

```
# Piecewise-linear majorizing function with power-distribution density function
m <- data.frame(x = c(0, 1, 2), y = c(0, 0.375, 1.5))
samples <- accrej(n = 10, pdf = function(x) (3 / 8) * x ^ 2, support = c(0,2),
                  majorizingFcn = m, majorizingFcnType = "pwl", plotDelay = 0)
```

`craps` *Monte Carlo Simulation of the Dice Game "Craps"*

Description

A Monte Carlo simulation of the dice game "craps". Returns a point estimate of the probability of winning craps using fair dice.

Usage

```
craps(nrep = 1000, seed = NA, showProgress = TRUE)
```

Arguments

`nrep`	Number of replications (plays of a single game of craps)
`seed`	Initial seed to the random number generator (NA uses current state of random number generator; `NULL` seeds using system clock)
`showProgress`	If `TRUE`, displays a progress bar on screen during execution

Details

Implements a Monte Carlo simulation of the dice game craps played with fair dice. A single play of the game proceeds as follows:

- Two fair dice are rolled. If the sum is 7 or 11, the player wins immediately; if the sum is 2, 3, or 12, the player loses immediately. Otherwise the sum becomes the *point*.
- The two dice continue to be rolled until either a sum of 7 is rolled (in which case the player loses) or a sum equal to the *point* is rolled (in which case the player wins).

The simulation involves `nrep` replications of the game.

Note: When the value of `nrep` is large, the function will execute noticeably faster when `showProgress` is set to `FALSE`.

Value

Point estimate of the probability of winning at craps (a real-valued scalar).

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

base::set.seed

Examples

```
# set the initial seed externally using set.seed;
# then use that current state of the generator with default nrep = 1000
set.seed(8675309)
craps()  # uses state of generator set above

# explicitly set the seed in the call to the function,
# using default nrep = 1000
craps(seed = 8675309)

# use the current state of the random number generator with nrep = 10000
prob <- craps(10000)

# explicitly set nrep = 10000 and seed = 8675309
prob <- craps(10000, 8675309)
```

---

galileo — *Monte Carlo Simulation of Galileo's Dice*

Description

A Monte Carlo simulation of the Galileo's Dice problem. Returns a vector containing point estimates of the probabilities of the sum of three fair dice for sums 3, 4, . . ., 18.

Usage

```
galileo(nrep = 1000, seed = NA, showProgress = TRUE)
```

Arguments

`nrep`	number of replications (rolls of the three dice)
`seed`	initial seed to the random number generator (NA uses current state of random number generator; NULL seeds using system clock)
`showProgress`	If TRUE, displays a progress bar on screen during execution

Details

Implements a Monte Carlo simulation of the Galileo's Dice problem. The simulation involves `nrep` replications of rolling three dice and summing the up-faces, and computing point estimates of the probabilities of each possible sum 3, 4, . . ., 18.

Note: When the value of `nrep` is large, the function will execute noticeably faster when `showProgress` is set to `FALSE`.

Value

An 18-element vector of point estimates of the probabilities. (Because a sum of 1 or 2 is not possible, the corresponding entries in the returned vector have value NA.)

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

Examples

```
# set the initial seed externally using set.seed;
# then use that current state of the generator with default nrep = 1000
set.seed(8675309)
galileo()  # uses state of generator set above

# explicitly set the seed in the call to the function,
# using default nrep = 1000
galileo(seed = 8675309)

# use the current state of the random number generator with nrep = 10000
prob <- galileo(10000)

# explicitly set nrep = 10000 and seed = 8675309
prob <- galileo(10000, 8675309)
```

---

ibeta *Visualization of Random Variate Generation for the Beta Distribution*

Description

Generates random variates from the Beta distribution by inversion. Optionally graphs the population cumulative distribution function and associated random variates, the population probability density function and a histogram of the random variates, and the empirical cumulative distribution function versus the population cumulative distribution function.

Usage

```
ibeta(
  u = runif(1),
  shape1,
  shape2,
  ncp = 0,
  minPlotQuantile = 0.01,
  maxPlotQuantile = 0.95,
  plot = TRUE,
```

```
  showCDF = TRUE,
  showPDF = TRUE,
  showECDF = TRUE,
  show = NULL,
  maxInvPlotted = 50,
  plotDelay = 0,
  sampleColor = "red3",
  populationColor = "grey",
  showTitle = TRUE,
  respectLayout = FALSE,
  restorePar = TRUE,
  ...
)
```

Arguments

`u`	vector of uniform(0,1) random numbers, or NULL to show population figures only
`shape1`	Shape parameter 1 (alpha)
`shape2`	Shape parameter 2 (beta)
`ncp`	Non-centrality parameter (default 0)
`minPlotQuantile`	minimum quantile to plot
`maxPlotQuantile`	maximum quantile to plot
`plot`	logical; if `TRUE` (default), one or more plots will appear (see parameters below); otherwise no plots appear
`showCDF`	logical; if `TRUE` (default), cdf plot appears, otherwise cdf plot is suppressed
`showPDF`	logical; if `TRUE` (default), PDF plot appears, otherwise PDF plot is suppressed
`showECDF`	logical; if `TRUE` (default), ecdf plot appears, otherwise ecdf plot is suppressed
`show`	octal number (0-7) indicating plots to display; 4: CDF, 2: PDF, 1: ECDF; sum for desired combination
`maxInvPlotted`	number of inversions to plot across CDF before switching to plotting quantiles only
`plotDelay`	delay in seconds between CDF plots
`sampleColor`	Color used to display random sample from distribution
`populationColor`	Color used to display population
`showTitle`	logical; if `TRUE` (default), displays a title in the first of any displayed plots
`respectLayout`	logical; if `TRUE` (default), respects existing settings for device layout
`restorePar`	logical; if `TRUE` (default), restores user's previous par settings on function exit
`...`	Possible additional arguments. Currently, additional arguments not considered.

Details

Generates random variates from the Beta distribution, and optionally, illustrates

- the use of the inverse-CDF technique,
- the effect of random sampling variability in relation to the PDF and CDF.

When all of the graphics are requested,

- the first graph illustrates the use of the inverse-CDF technique by graphing the population CDF and the transformation of the random numbers to random variates,
- the second graph illustrates the effect of random sampling variability by graphing the population PDF and the histogram associated with the random variates, and
- the third graph illustrates effect of random sampling variability by graphing the population CDF and the empirical CDF associated with the random variates.

All aspects of the random variate generation algorithm are output in red by default, which can be changed by specifying `sampleColor`. All aspects of the population distribution are output in gray by default, which can be changed by specifying `populationColor`.

The beta distribution has density

```
\deqn{f(x) = \frac{\Gamma(a+b)}{\Gamma(a) \ \Gamma(b)} x^{a-1}(1-x)^{b-1}}{
          f(x) = Gamma(a+b)/(Gamma(a)Gamma(b)) x^(a-1)(1-x)^(b-1)}
```

for $a > 0$, $b > 0$ and $0 \leq x \leq 1$ where the boundary values at $x = 0$ or $x = 1$ are defined as by continuity (as limits).

The mean is $\frac{a}{a+b}$ and the variance is $ab(a + b)^2(a + b + 1)$

The algorithm for generating random variates from the beta distribution is synchronized (one random variate for each random number) and monotone in u. This means that the variates generated here might be useful in some variance reduction techniques used in Monte Carlo and discrete-event simulation.

Values from the u vector are plotted in the cdf plot along the vertical axis as colored dots. A horizontal, dashed, colored line extends from the dot to the population cdf. At the intersection, a vertical, dashed colored line extends downward to the horizontal axis, where a second colored dot, denoting the associated beta random variate is plotted.

This is not a particularly fast variate generation algorithm because it uses the base R `qbeta` function to invert the values contained in `u`.

All of the elements of the `u` vector must be between 0 and 1. Alternatively, `u` can be `NULL` in which case plot(s) of the theoretical PDF and cdf are displayed according to plotting parameter values (defaulting to display of both the PDF and cdf).

The `show` parameter can be used as a shortcut way to denote plots to display. The argument to `show` can be either:

- a binary vector of length three, where the entries from left to right correspond to `showCDF`, `showPDF`, and `showECDF`, respectively. For each entry, a 1 indicates the plot should be displayed, and a 0 indicates the plot should be suppressed.

- an integer in [0,7] interpreted similar to the Unix chmod command. That is, the integer's binary representation can be transformed into a length-three vector discussed above (e.g., 6 corresponds to c(1,1,0)). See examples.

Any valid value for `show` takes precedence over existing individual values for `showCDF`, `showPDF`, and `showECDF`.

If `respectLayout` is `TRUE`, the function respects existing settings for device layout. Note, however, that if the number of plots requested (either via `show` or via `showCDF`, `showPMF`, and `showECDF`) exceeds the number of plots available in the current layout (as determined by `prod(par("mfrow"))`), the function will display all requested plots but will also display a warning message indicating that the current layout does not permit simultaneous viewing of all requested plots. The most recent plot with this attribute can be further annotated after the call.

If `respectLayout` is `FALSE`, any existing user settings for device layout are ignored. That is, the function uses `par` to explicitly set `mfrow` sufficient to show all requested plots stacked vertically to align their horizontal axes, and then resets row, column, and margin settings to their prior state on exit.

The `minPlotQuantile` and `maxPlotQuantile` arguments are present in order to compress the plots horizontally. The random variates generated are not impacted by these two arguments. Vertical, dotted, black lines are plotted at the associated quantiles on the plots.

`plotDelay` can be used to slow down or halt the variate generation for classroom explanation.

In the plot associated with the PDF, the maximum plotting height is associated with 125\ that extends above this limit will have three dots appearing above it.

Value

A vector of Beta random variates

Author(s)

Barry Lawson (<`blawson@bates.edu`>),
Larry Leemis (<`leemis@math.wm.edu`>),
Vadim Kudlay (<`vkudlay@nvidia.com`>)

See Also

`stats::rbeta`

`stats::runif`, `simEd::vunif`

Examples

```
ibeta(0.5, shape1 = 3, shape2 = 1, ncp = 2)

set.seed(8675309)
ibeta(runif(10), 3, 1, showPDF = TRUE)

set.seed(8675309)
ibeta(runif(10), 3, 1, showECDF = TRUE)

set.seed(8675309)
```

```
ibeta(runif(10), 3, 1, showPDF = TRUE, showECDF = TRUE, sampleColor = "blue3")

set.seed(8675309)
ibeta(runif(10), 3, 1, showPDF = TRUE, showCDF = FALSE)

ibeta(runif(100), 3, 1, showPDF = TRUE, minPlotQuantile = 0.02, maxPlotQuantile = 0.98)

# plot the PDF and CDF without any variates
ibeta(NULL, 3, 1, showPDF = TRUE, showCDF = TRUE)

# plot CDF with inversion and PDF using show
ibeta(runif(10), 3, 1, show = c(1,1,0))
ibeta(runif(10), 3, 1, show = 6)

# plot CDF with inversion and ECDF using show, using vunif
ibeta(vunif(10), 3, 1, show = c(1,0,1))
ibeta(vunif(10), 3, 1, show = 5)

# plot CDF with inversion, PDF, and ECDF using show
ibeta(vunif(10), 3, 1, show = c(1,1,1))
ibeta(vunif(10), 3, 1, show = 7)

# plot three different CDF+PDF+ECDF horizontal displays,
# with title only on the first display
oldpar <- par(no.readonly = TRUE)
par(mfrow = c(3,3))  # 3 rows, 3 cols, filling rows before columns
set.seed(8675309)
ibeta(runif(20), 3, 1, show = 7, respectLayout = TRUE, restorePar = FALSE)
ibeta(runif(20), 3, 1, show = 7, respectLayout = TRUE, restorePar = FALSE, showTitle = FALSE)
ibeta(runif(20), 3, 1, show = 7, respectLayout = TRUE, restorePar = TRUE,  showTitle = FALSE)
par(oldpar)

# display animation of all components
ibeta(runif(10), 3, 1, show = 7, plotDelay = 0.1)

# display animation of CDF and PDF components only
ibeta(runif(10), 3, 1, show = 5, plotDelay = 0.1)

if (interactive()) {
  # interactive -- pause at each stage of inversion
  ibeta(runif(10), 3, 1, show = 7, plotDelay = -1)
}
```

---

ibinom — *Visualization of Random Variate Generation for the Binomial Distribution*

Description

Generates random variates from the Binomial distribution by inversion. Optionally graphs the population cumulative distribution function and associated random variates, the population probability mass function and a histogram of the random variates, and the empirical cumulative distribution function versus the population cumulative distribution function.

Usage

```
ibinom(
  u = runif(1),
  size,
  prob,
  minPlotQuantile = 0,
  maxPlotQuantile = 1,
  plot = TRUE,
  showCDF = TRUE,
  showPMF = TRUE,
  showECDF = TRUE,
  show = NULL,
  maxInvPlotted = 50,
  plotDelay = 0,
  sampleColor = "red3",
  populationColor = "grey",
  showTitle = TRUE,
  respectLayout = FALSE,
  restorePar = TRUE,
  ...
)
```

Arguments

`u`	vector of uniform(0,1) random numbers, or NULL to show population figures only
`size`	number of trials (zero or more)
`prob`	probability of success on each trial ($0 <$ `prob` ≤ 1)
`minPlotQuantile`	minimum quantile to plot
`maxPlotQuantile`	maximum quantile to plot
`plot`	logical; if `TRUE` (default), one or more plots will appear (see parameters below); otherwise no plots appear
`showCDF`	logical; if `TRUE` (default), cdf plot appears, otherwise cdf plot is suppressed
`showPMF`	logical; if `TRUE` (default), PMF plot appears, otherwise PMF plot is suppressed
`showECDF`	logical; if `TRUE` (default), ecdf plot appears, otherwise ecdf plot is suppressed
`show`	octal number (0-7) indicating plots to display; 4: CDF, 2: PMF, 1: ECDF; sum for desired combination
`maxInvPlotted`	number of inversions to plot across CDF before switching to plotting quantiles only
`plotDelay`	delay in seconds between CDF plots

sampleColor	Color used to display random sample from distribution
populationColor	Color used to display population
showTitle	logical; if TRUE (default), displays a title in the first of any displayed plots
respectLayout	logical; if TRUE (default), respects existing settings for device layout
restorePar	logical; if TRUE (default), restores user's previous par settings on function exit
...	Possible additional arguments. Currently, additional arguments not considered.

Details

Generates random variates from the Binomial distribution, and optionally, illustrates

- the use of the inverse-CDF technique,
- the effect of random sampling variability in relation to the PMF and CDF.

When all of the graphics are requested,

- the first graph illustrates the use of the inverse-CDF technique by graphing the population CDF and the transformation of the random numbers to random variates,
- the second graph illustrates the effect of random sampling variability by graphing the population PMF and the histogram associated with the random variates, and
- the third graph illustrates effect of random sampling variability by graphing the population CDF and the empirical CDF associated with the random variates.

All aspects of the random variate generation algorithm are output in red by default, which can be changed by specifying `sampleColor`. All aspects of the population distribution are output in gray by default, which can be changed by specifying `populationColor`.

The binomial distribution with parameters `size` = n and `prob` = p has pmf

$$p(x) = \binom{n}{x} p^x (1-p)^{(n-x)}$$

for $x = 0, \ldots, n$.

The algorithm for generating random variates from the binomial distribution is synchronized (one random variate for each random number) and monotone in u. This means that the variates generated here might be useful in some variance reduction techniques used in Monte Carlo and discrete-event simulation.

Values from the u vector are plotted in the cdf plot along the vertical axis as colored dots. A horizontal, dashed, colored line extends from the dot to the population cdf. At the intersection, a vertical, dashed colored line extends downward to the horizontal axis, where a second colored dot, denoting the associated binomial random variate is plotted.

This is not a particularly fast variate generation algorithm because it uses the base R `qbinom` function to invert the values contained in `u`.

All of the elements of the `u` vector must be between 0 and 1. Alternatively, `u` can be `NULL` in which case plot(s) of the theoretical PMF and cdf are displayed according to plotting parameter values (defaulting to display of both the PMF and cdf).

The `show` parameter can be used as a shortcut way to denote plots to display. The argument to `show` can be either:

- a binary vector of length three, where the entries from left to right correspond to showCDF, showPMF, and showECDF, respectively. For each entry, a 1 indicates the plot should be displayed, and a 0 indicates the plot should be suppressed.
- an integer in [0,7] interpreted similar to the Unix chmod command. That is, the integer's binary representation can be transformed into a length-three vector discussed above (e.g., 6 corresponds to c(1,1,0)). See examples.

Any valid value for show takes precedence over existing individual values for showCDF, showPMF, and showECDF.

If respectLayout is TRUE, the function respects existing settings for device layout. Note, however, that if the number of plots requested (either via show or via showCDF, showPMF, and showECDF) exceeds the number of plots available in the current layout (as determined by prod(par("mfrow"))), the function will display all requested plots but will also display a warning message indicating that the current layout does not permit simultaneous viewing of all requested plots. The most recent plot with this attribute can be further annotated after the call.

If respectLayout is FALSE, any existing user settings for device layout are ignored. That is, the function uses par to explicitly set mfrow sufficient to show all requested plots stacked vertically to align their horizontal axes, and then resets row, column, and margin settings to their prior state on exit.

The minPlotQuantile and maxPlotQuantile arguments are present in order to compress the plots horizontally. The random variates generated are not impacted by these two arguments. Vertical, dotted, black lines are plotted at the associated quantiles on the plots.

plotDelay can be used to slow down or halt the variate generation for classroom explanation.

In the plot associated with the PMF, the maximum plotting height is associated with 125\ that extends above this limit will have three dots appearing above it.

Value

A vector of Binomial random variates

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

stats::rbinom

stats::runif, simEd::vunif

Examples

```
ibinom(0.5, size = 7, prob = 0.4,)

set.seed(8675309)
ibinom(runif(10), 10, 0.3, showPMF = TRUE)
```

```
set.seed(8675309)
ibinom(runif(10), 10, 0.3, showECDF = TRUE)

set.seed(8675309)
ibinom(runif(10), 10, 0.3, showPMF = TRUE, showECDF = TRUE, sampleColor = "blue3")

set.seed(8675309)
ibinom(runif(10), 10, 0.3, showPMF = TRUE, showCDF = FALSE)

ibinom(runif(100), 10, 0.3, showPMF = TRUE, minPlotQuantile = 0.02, maxPlotQuantile = 0.98)

# plot the PMF and CDF without any variates
ibinom(NULL, 10, 0.3, showPMF = TRUE, showCDF = TRUE)

# plot CDF with inversion and PMF using show
ibinom(runif(10), 10, 0.3, show = c(1,1,0))
ibinom(runif(10), 10, 0.3, show = 6)

# plot CDF with inversion and ECDF using show, using vunif
ibinom(vunif(10), 10, 0.3, show = c(1,0,1))
ibinom(vunif(10), 10, 0.3, show = 5)

# plot CDF with inversion, PMF, and ECDF using show
ibinom(vunif(10), 10, 0.3, show = c(1,1,1))
ibinom(vunif(10), 10, 0.3, show = 7)

# plot three different CDF+PMF+ECDF horizontal displays,
# with title only on the first display
oldpar <- par(no.readonly = TRUE)
par(mfrow = c(3,3))  # 3 rows, 3 cols, filling rows before columns
set.seed(8675309)
ibinom(runif(20), 10, 0.3, show = 7, respectLayout = TRUE, restorePar = FALSE)
ibinom(runif(20), 10, 0.3, show = 7, respectLayout = TRUE, restorePar = FALSE, showTitle = FALSE)
ibinom(runif(20), 10, 0.3, show = 7, respectLayout = TRUE, restorePar = TRUE,  showTitle = FALSE)
par(oldpar)

# display animation of all components
ibinom(runif(10), 10, 0.3, show = 7, plotDelay = 0.1)

# display animation of CDF and PMF components only
ibinom(runif(10), 10, 0.3, show = 5, plotDelay = 0.1)

if (interactive()) {
  # interactive -- pause at each stage of inversion
  ibinom(runif(10), 10, 0.3, show = 7, plotDelay = -1)
}
```

icauchy — *Visualization of Random Variate Generation for the Cauchy Distribution*

Description

Generates random variates from the Cauchy distribution by inversion. Optionally graphs the population cumulative distribution function and associated random variates, the population probability density function and a histogram of the random variates, and the empirical cumulative distribution function versus the population cumulative distribution function.

Usage

```
icauchy(
  u = runif(1),
  location = 0,
  scale = 1,
  minPlotQuantile = 0.05,
  maxPlotQuantile = 0.95,
  plot = TRUE,
  showCDF = TRUE,
  showPDF = TRUE,
  showECDF = TRUE,
  show = NULL,
  maxInvPlotted = 50,
  plotDelay = 0,
  sampleColor = "red3",
  populationColor = "grey",
  showTitle = TRUE,
  respectLayout = FALSE,
  restorePar = TRUE,
  ...
)
```

Arguments

`u`	vector of uniform(0,1) random numbers, or NULL to show population figures only
`location`	Location parameter (default 0)
`scale`	Scale parameter (default 1)
`minPlotQuantile`	minimum quantile to plot
`maxPlotQuantile`	maximum quantile to plot
`plot`	logical; if `TRUE` (default), one or more plots will appear (see parameters below); otherwise no plots appear
`showCDF`	logical; if `TRUE` (default), cdf plot appears, otherwise cdf plot is suppressed
`showPDF`	logical; if `TRUE` (default), PDF plot appears, otherwise PDF plot is suppressed
`showECDF`	logical; if `TRUE` (default), ecdf plot appears, otherwise ecdf plot is suppressed
`show`	octal number (0-7) indicating plots to display; 4: CDF, 2: PDF, 1: ECDF; sum for desired combination

`maxInvPlotted`	number of inversions to plot across CDF before switching to plotting quantiles only
`plotDelay`	delay in seconds between CDF plots
`sampleColor`	Color used to display random sample from distribution
`populationColor`	Color used to display population
`showTitle`	logical; if `TRUE` (default), displays a title in the first of any displayed plots
`respectLayout`	logical; if `TRUE` (default), respects existing settings for device layout
`restorePar`	logical; if `TRUE` (default), restores user's previous par settings on function exit
`...`	Possible additional arguments. Currently, additional arguments not considered.

Details

Generates random variates from the Cauchy distribution, and optionally, illustrates

- the use of the inverse-CDF technique,
- the effect of random sampling variability in relation to the PDF and CDF.

When all of the graphics are requested,

- the first graph illustrates the use of the inverse-CDF technique by graphing the population CDF and the transformation of the random numbers to random variates,
- the second graph illustrates the effect of random sampling variability by graphing the population PDF and the histogram associated with the random variates, and
- the third graph illustrates effect of random sampling variability by graphing the population CDF and the empirical CDF associated with the random variates.

All aspects of the random variate generation algorithm are output in red by default, which can be changed by specifying `sampleColor`. All aspects of the population distribution are output in gray by default, which can be changed by specifying `populationColor`.

```
The Cauchy distribution has density
\deqn{f(x) = \frac{1}{\pi s} \ \left(1 + \left( \frac{x - l}{s} \right)^2
              \right)^{-1}}{
          f(x) = 1 / (\pi s (1 + ((x-l)/s)^2))}
```

for all x.

The mean is $a/(a+b)$ and the variance is $ab/((a+b)^2(a+b+1))$.

The algorithm for generating random variates from the Cauchy distribution is synchronized (one random variate for each random number) and monotone in u. This means that the variates generated here might be useful in some variance reduction techniques used in Monte Carlo and discrete-event simulation.

Values from the u vector are plotted in the cdf plot along the vertical axis as colored dots. A horizontal, dashed, colored line extends from the dot to the population cdf. At the intersection, a vertical, dashed colored line extends downward to the horizontal axis, where a second colored dot, denoting the associated Cauchy random variate is plotted.

This is not a particularly fast variate generation algorithm because it uses the base R `qcauchy` function to invert the values contained in `u`.

All of the elements of the `u` vector must be between 0 and 1. Alternatively, `u` can be `NULL` in which case plot(s) of the theoretical PDF and cdf are displayed according to plotting parameter values (defaulting to display of both the PDF and cdf).

The `show` parameter can be used as a shortcut way to denote plots to display. The argument to `show` can be either:

- a binary vector of length three, where the entries from left to right correspond to `showCDF`, `showPDF`, and `showECDF`, respectively. For each entry, a 1 indicates the plot should be displayed, and a 0 indicates the plot should be suppressed.
- an integer in [0,7] interpreted similar to the Unix chmod command. That is, the integer's binary representation can be transformed into a length-three vector discussed above (e.g., 6 corresponds to c(1,1,0)). See examples.

Any valid value for `show` takes precedence over existing individual values for `showCDF`, `showPDF`, and `showECDF`.

If `respectLayout` is `TRUE`, the function respects existing settings for device layout. Note, however, that if the number of plots requested (either via `show` or via `showCDF`, `showPMF`, and `showECDF`) exceeds the number of plots available in the current layout (as determined by `prod(par("mfrow"))`), the function will display all requested plots but will also display a warning message indicating that the current layout does not permit simultaneous viewing of all requested plots. The most recent plot with this attribute can be further annotated after the call.

If `respectLayout` is `FALSE`, any existing user settings for device layout are ignored. That is, the function uses `par` to explicitly set `mfrow` sufficient to show all requested plots stacked vertically to align their horizontal axes, and then resets row, column, and margin settings to their prior state on exit.

The `minPlotQuantile` and `maxPlotQuantile` arguments are present in order to compress the plots horizontally. The random variates generated are not impacted by these two arguments. Vertical, dotted, black lines are plotted at the associated quantiles on the plots.

`plotDelay` can be used to slow down or halt the variate generation for classroom explanation.

In the plot associated with the PDF, the maximum plotting height is associated with 125\ that extends above this limit will have three dots appearing above it.

Value

A vector of Cauchy random variates

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

`stats::rcauchy`

`stats::runif`, `simEd::vunif`

Examples

```
icauchy(0.5, location = 3, scale = 1)

set.seed(8675309)
icauchy(runif(10), 0, 3, showPDF = TRUE)

set.seed(8675309)
icauchy(runif(10), 0, 3, showECDF = TRUE)

set.seed(8675309)
icauchy(runif(10), 0, 3, showPDF = TRUE, showECDF = TRUE, sampleColor = "blue3")

set.seed(8675309)
icauchy(runif(10), 0, 3, showPDF = TRUE, showCDF = FALSE)

icauchy(runif(100), 0, 3, showPDF = TRUE, minPlotQuantile = 0.02, maxPlotQuantile = 0.98)

# plot the PDF and CDF without any variates
icauchy(NULL, 0, 3, showPDF = TRUE, showCDF = TRUE)

# plot CDF with inversion and PDF using show
icauchy(runif(10), 0, 3, show = c(1,1,0))
icauchy(runif(10), 0, 3, show = 6)

# plot CDF with inversion and ECDF using show, using vunif
icauchy(vunif(10), 0, 3, show = c(1,0,1))
icauchy(vunif(10), 0, 3, show = 5)

# plot CDF with inversion, PDF, and ECDF using show
icauchy(vunif(10), 0, 3, show = c(1,1,1))
icauchy(vunif(10), 0, 3, show = 7)

# plot three different CDF+PDF+ECDF horizontal displays,
# with title only on the first display
oldpar <- par(no.readonly = TRUE)
par(mfrow = c(3,3))  # 3 rows, 3 cols, filling rows before columns
set.seed(8675309)
icauchy(runif(20), 0, 3, show = 7, respectLayout = TRUE, restorePar = FALSE)
icauchy(runif(20), 0, 3, show = 7, respectLayout = TRUE, restorePar = FALSE, showTitle = FALSE)
icauchy(runif(20), 0, 3, show = 7, respectLayout = TRUE, restorePar = TRUE,  showTitle = FALSE)
par(oldpar)

# display animation of all components
icauchy(runif(10), 0, 3, show = 7, plotDelay = 0.1)

# display animation of CDF and PDF components only
icauchy(runif(10), 0, 3, show = 5, plotDelay = 0.1)

if (interactive()) {
  # interactive -- pause at each stage of inversion
  icauchy(runif(10), 0, 3, show = 7, plotDelay = -1)
}
```

ichisq — *Visualization of Random Variate Generation for the Chi-Squared Distribution*

Description

Generates random variates from the Chi-Squared distribution by inversion. Optionally graphs the population cumulative distribution function and associated random variates, the population probability density function and a histogram of the random variates, and the empirical cumulative distribution function versus the population cumulative distribution function.

Usage

```
ichisq(
  u = runif(1),
  df,
  ncp = 0,
  minPlotQuantile = 0.01,
  maxPlotQuantile = 0.99,
  plot = TRUE,
  showCDF = TRUE,
  showPDF = TRUE,
  showECDF = TRUE,
  show = NULL,
  maxInvPlotted = 50,
  plotDelay = 0,
  sampleColor = "red3",
  populationColor = "grey",
  showTitle = TRUE,
  respectLayout = FALSE,
  restorePar = TRUE,
  ...
)
```

Arguments

`u`	vector of uniform(0,1) random numbers, or NULL to show population figures only
`df`	Degrees of freedom (non-negative, but can be non-integer)
`ncp`	Non-centrality parameter (non-negative)
`minPlotQuantile`	minimum quantile to plot
`maxPlotQuantile`	maximum quantile to plot

`plot`	logical; if `TRUE` (default), one or more plots will appear (see parameters below); otherwise no plots appear
`showCDF`	logical; if `TRUE` (default), cdf plot appears, otherwise cdf plot is suppressed
`showPDF`	logical; if `TRUE` (default), PDF plot appears, otherwise PDF plot is suppressed
`showECDF`	logical; if `TRUE` (default), ecdf plot appears, otherwise ecdf plot is suppressed
`show`	octal number (0-7) indicating plots to display; 4: CDF, 2: PDF, 1: ECDF; sum for desired combination
`maxInvPlotted`	number of inversions to plot across CDF before switching to plotting quantiles only
`plotDelay`	delay in seconds between CDF plots
`sampleColor`	Color used to display random sample from distribution
`populationColor`	Color used to display population
`showTitle`	logical; if `TRUE` (default), displays a title in the first of any displayed plots
`respectLayout`	logical; if `TRUE` (default), respects existing settings for device layout
`restorePar`	logical; if `TRUE` (default), restores user's previous par settings on function exit
`...`	Possible additional arguments. Currently, additional arguments not considered.

Details

Generates random variates from the Chi-Squared distribution, and optionally, illustrates

- the use of the inverse-CDF technique,
- the effect of random sampling variability in relation to the PDF and CDF.

When all of the graphics are requested,

- the first graph illustrates the use of the inverse-CDF technique by graphing the population CDF and the transformation of the random numbers to random variates,
- the second graph illustrates the effect of random sampling variability by graphing the population PDF and the histogram associated with the random variates, and
- the third graph illustrates effect of random sampling variability by graphing the population CDF and the empirical CDF associated with the random variates.

All aspects of the random variate generation algorithm are output in red by default, which can be changed by specifying `sampleColor`. All aspects of the population distribution are output in gray by default, which can be changed by specifying `populationColor`.

The chi-squared distribution with `df` = $n \geq 0$ degrees of freedom has density

```
\deqn{f_n(x) = \frac{1}{2^{n/2} \ \Gamma(n/2)} x^{n/2-1} e^{-x/2}}{
          f_n(x) = 1 / (2^(n/2) \Gamma(n/2)) x^(n/2-1) e^(-x/2)}
```

for $x > 0$. The mean and variance are n and $2n$.

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

```
\deqn{f(x) = e^{-\lambda/2} \sum_{r=0}^\infty \frac{(\lambda/2)^r}{r!} f_{n + 2r}(x)}{
      f(x) = exp(-\lambda/2) SUM_{r=0}^\infty ((\lambda/2)^r / r!) dchisq(x, df + 2r)}
```

for $x \geq 0$.

The algorithm for generating random variates from the chi-squared distribution is synchronized (one random variate for each random number) and monotone in u. This means that the variates generated here might be useful in some variance reduction techniques used in Monte Carlo and discrete-event simulation.

Values from the u vector are plotted in the cdf plot along the vertical axis as colored dots. A horizontal, dashed, colored line extends from the dot to the population cdf. At the intersection, a vertical, dashed colored line extends downward to the horizontal axis, where a second colored dot, denoting the associated chi-squared random variate is plotted.

This is not a particularly fast variate generation algorithm because it uses the base R `qchisq` function to invert the values contained in `u`.

All of the elements of the `u` vector must be between 0 and 1. Alternatively, `u` can be `NULL` in which case plot(s) of the theoretical PDF and cdf are displayed according to plotting parameter values (defaulting to display of both the PDF and cdf).

The `show` parameter can be used as a shortcut way to denote plots to display. The argument to `show` can be either:

- a binary vector of length three, where the entries from left to right correspond to `showCDF`, `showPDF`, and `showECDF`, respectively. For each entry, a 1 indicates the plot should be displayed, and a 0 indicates the plot should be suppressed.
- an integer in [0,7] interpreted similar to the Unix chmod command. That is, the integer's binary representation can be transformed into a length-three vector discussed above (e.g., 6 corresponds to c(1,1,0)). See examples.

Any valid value for `show` takes precedence over existing individual values for `showCDF`, `showPDF`, and `showECDF`.

If `respectLayout` is `TRUE`, the function respects existing settings for device layout. Note, however, that if the number of plots requested (either via `show` or via `showCDF`, `showPMF`, and `showECDF`) exceeds the number of plots available in the current layout (as determined by `prod(par("mfrow"))`), the function will display all requested plots but will also display a warning message indicating that the current layout does not permit simultaneous viewing of all requested plots. The most recent plot with this attribute can be further annotated after the call.

If `respectLayout` is `FALSE`, any existing user settings for device layout are ignored. That is, the function uses `par` to explicitly set `mfrow` sufficient to show all requested plots stacked vertically to align their horizontal axes, and then resets row, column, and margin settings to their prior state on exit.

The `minPlotQuantile` and `maxPlotQuantile` arguments are present in order to compress the plots horizontally. The random variates generated are not impacted by these two arguments. Vertical, dotted, black lines are plotted at the associated quantiles on the plots.

`plotDelay` can be used to slow down or halt the variate generation for classroom explanation.

In the plot associated with the PDF, the maximum plotting height is associated with 125\ that extends above this limit will have three dots appearing above it.

Value

A vector of Chi-Squared random variates

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

stats::rchisq

stats::runif, simEd::vunif

Examples

```
ichisq(0.5, df = 3, ncp = 2)

set.seed(8675309)
ichisq(runif(10), 3, showPDF = TRUE)

set.seed(8675309)
ichisq(runif(10), 3, showECDF = TRUE)

set.seed(8675309)
ichisq(runif(10), 3, showPDF = TRUE, showECDF = TRUE, sampleColor = "blue3")

set.seed(8675309)
ichisq(runif(10), 3, showPDF = TRUE, showCDF = FALSE)

ichisq(runif(100), 3, showPDF = TRUE, minPlotQuantile = 0.02, maxPlotQuantile = 0.98)

# plot the PDF and CDF without any variates
ichisq(NULL, 3, showPDF = TRUE, showCDF = TRUE)

# plot CDF with inversion and PDF using show
ichisq(runif(10), 3, show = c(1,1,0))
ichisq(runif(10), 3, show = 6)

# plot CDF with inversion and ECDF using show, using vunif
ichisq(vunif(10), 3, show = c(1,0,1))
ichisq(vunif(10), 3, show = 5)

# plot CDF with inversion, PDF, and ECDF using show
ichisq(vunif(10), 3, show = c(1,1,1))
ichisq(vunif(10), 3, show = 7)

# plot three different CDF+PDF+ECDF horizontal displays,
# with title only on the first display
oldpar <- par(no.readonly = TRUE)
par(mfrow = c(3,3))  # 3 rows, 3 cols, filling rows before columns
```

```
set.seed(8675309)
ichisq(runif(20), 3, show = 7, respectLayout = TRUE, restorePar = FALSE)
ichisq(runif(20), 3, show = 7, respectLayout = TRUE, restorePar = FALSE, showTitle = FALSE)
ichisq(runif(20), 3, show = 7, respectLayout = TRUE, restorePar = TRUE,  showTitle = FALSE)
par(oldpar)

# display animation of all components
ichisq(runif(10), 3, show = 7, plotDelay = 0.1)

# display animation of CDF and PDF components only
ichisq(runif(10), 3, show = 5, plotDelay = 0.1)

if (interactive()) {
  # interactive -- pause at each stage of inversion
  ichisq(runif(10), 3, show = 7, plotDelay = -1)
}
```

---

iexp   *Visualization of Random Variate Generation for the Exponential Distribution*

Description

Generates random variates from the Exponential distribution by inversion. Optionally graphs the population cumulative distribution function and associated random variates, the population probability density function and a histogram of the random variates, and the empirical cumulative distribution function versus the population cumulative distribution function.

Usage

```
iexp(
  u = runif(1),
  rate = 1,
  minPlotQuantile = 0,
  maxPlotQuantile = 0.99,
  plot = TRUE,
  showCDF = TRUE,
  showPDF = TRUE,
  showECDF = TRUE,
  show = NULL,
  maxInvPlotted = 50,
  plotDelay = 0,
  sampleColor = "red3",
  populationColor = "grey",
  showTitle = TRUE,
  respectLayout = FALSE,
  restorePar = TRUE,
  ...
)
```

Arguments

`u`	vector of uniform(0,1) random numbers, or NULL to show population figures only
`rate`	Rate of distribution (default 1)
`minPlotQuantile`	minimum quantile to plot
`maxPlotQuantile`	maximum quantile to plot
`plot`	logical; if `TRUE` (default), one or more plots will appear (see parameters below); otherwise no plots appear
`showCDF`	logical; if `TRUE` (default), cdf plot appears, otherwise cdf plot is suppressed
`showPDF`	logical; if `TRUE` (default), PDF plot appears, otherwise PDF plot is suppressed
`showECDF`	logical; if `TRUE` (default), ecdf plot appears, otherwise ecdf plot is suppressed
`show`	octal number (0-7) indicating plots to display; 4: CDF, 2: PDF, 1: ECDF; sum for desired combination
`maxInvPlotted`	number of inversions to plot across CDF before switching to plotting quantiles only
`plotDelay`	delay in seconds between CDF plots
`sampleColor`	Color used to display random sample from distribution
`populationColor`	Color used to display population
`showTitle`	logical; if `TRUE` (default), displays a title in the first of any displayed plots
`respectLayout`	logical; if `TRUE` (default), respects existing settings for device layout
`restorePar`	logical; if `TRUE` (default), restores user's previous par settings on function exit
`...`	Possible additional arguments. Currently, additional arguments not considered.

Details

Generates random variates from the Exponential distribution, and optionally, illustrates

- the use of the inverse-CDF technique,
- the effect of random sampling variability in relation to the PDF and CDF.

When all of the graphics are requested,

- the first graph illustrates the use of the inverse-CDF technique by graphing the population CDF and the transformation of the random numbers to random variates,
- the second graph illustrates the effect of random sampling variability by graphing the population PDF and the histogram associated with the random variates, and
- the third graph illustrates effect of random sampling variability by graphing the population CDF and the empirical CDF associated with the random variates.

All aspects of the random variate generation algorithm are output in red by default, which can be changed by specifying `sampleColor`. All aspects of the population distribution are output in gray by default, which can be changed by specifying `populationColor`.

```
The exponential distribution with rate \eqn{\lambda} has density

    \deqn{f(x) = \lambda e^{-\lambda x}}{
              f(x) = \lambda e^(-\lambda x)}

for \eqn{x \geq 0}.
```

The algorithm for generating random variates from the exponential distribution is synchronized (one random variate for each random number) and monotone in u. This means that the variates generated here might be useful in some variance reduction techniques used in Monte Carlo and discrete-event simulation.

Values from the u vector are plotted in the cdf plot along the vertical axis as colored dots. A horizontal, dashed, colored line extends from the dot to the population cdf. At the intersection, a vertical, dashed colored line extends downward to the horizontal axis, where a second colored dot, denoting the associated exponential random variate is plotted.

This is not a particularly fast variate generation algorithm because it uses the base R `qexp` function to invert the values contained in `u`.

All of the elements of the `u` vector must be between 0 and 1. Alternatively, `u` can be `NULL` in which case plot(s) of the theoretical PDF and cdf are displayed according to plotting parameter values (defaulting to display of both the PDF and cdf).

The `show` parameter can be used as a shortcut way to denote plots to display. The argument to `show` can be either:

- a binary vector of length three, where the entries from left to right correspond to `showCDF`, `showPDF`, and `showECDF`, respectively. For each entry, a 1 indicates the plot should be displayed, and a 0 indicates the plot should be suppressed.
- an integer in [0,7] interpreted similar to the Unix chmod command. That is, the integer's binary representation can be transformed into a length-three vector discussed above (e.g., 6 corresponds to c(1,1,0)). See examples.

Any valid value for `show` takes precedence over existing individual values for `showCDF`, `showPDF`, and `showECDF`.

If `respectLayout` is `TRUE`, the function respects existing settings for device layout. Note, however, that if the number of plots requested (either via `show` or via `showCDF`, `showPMF`, and `showECDF`) exceeds the number of plots available in the current layout (as determined by `prod(par("mfrow"))`), the function will display all requested plots but will also display a warning message indicating that the current layout does not permit simultaneous viewing of all requested plots. The most recent plot with this attribute can be further annotated after the call.

If `respectLayout` is `FALSE`, any existing user settings for device layout are ignored. That is, the function uses `par` to explicitly set `mfrow` sufficient to show all requested plots stacked vertically to align their horizontal axes, and then resets row, column, and margin settings to their prior state on exit.

The `minPlotQuantile` and `maxPlotQuantile` arguments are present in order to compress the plots horizontally. The random variates generated are not impacted by these two arguments. Vertical, dotted, black lines are plotted at the associated quantiles on the plots.

`plotDelay` can be used to slow down or halt the variate generation for classroom explanation.

In the plot associated with the PDF, the maximum plotting height is associated with 125\ that extends above this limit will have three dots appearing above it.

Value

A vector of Exponential random variates

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

stats::rexp

stats::runif, simEd::vunif

Examples

```
iexp(0.5, rate = 3)

set.seed(8675309)
iexp(runif(10), 2, showPDF = TRUE)

set.seed(8675309)
iexp(runif(10), 2, showECDF = TRUE)

set.seed(8675309)
iexp(runif(10), 2, showPDF = TRUE, showECDF = TRUE, sampleColor = "blue3")

set.seed(8675309)
iexp(runif(10), 2, showPDF = TRUE, showCDF = FALSE)

iexp(runif(100), 2, showPDF = TRUE, minPlotQuantile = 0.02, maxPlotQuantile = 0.98)

# plot the PDF and CDF without any variates
iexp(NULL, 2, showPDF = TRUE, showCDF = TRUE)

# plot CDF with inversion and PDF using show
iexp(runif(10), 2, show = c(1,1,0))
iexp(runif(10), 2, show = 6)

# plot CDF with inversion and ECDF using show, using vunif
iexp(vunif(10), 2, show = c(1,0,1))
iexp(vunif(10), 2, show = 5)

# plot CDF with inversion, PDF, and ECDF using show
iexp(vunif(10), 2, show = c(1,1,1))
iexp(vunif(10), 2, show = 7)

# plot three different CDF+PDF+ECDF horizontal displays,
```

```
# with title only on the first display
oldpar <- par(no.readonly = TRUE)
par(mfrow = c(3,3))  # 3 rows, 3 cols, filling rows before columns
set.seed(8675309)
iexp(runif(20), 2, show = 7, respectLayout = TRUE, restorePar = FALSE)
iexp(runif(20), 2, show = 7, respectLayout = TRUE, restorePar = FALSE, showTitle = FALSE)
iexp(runif(20), 2, show = 7, respectLayout = TRUE, restorePar = TRUE,  showTitle = FALSE)
par(oldpar)

# display animation of all components
iexp(runif(10), 2, show = 7, plotDelay = 0.1)

# display animation of CDF and PDF components only
iexp(runif(10), 2, show = 5, plotDelay = 0.1)

if (interactive()) {
  # interactive -- pause at each stage of inversion
  iexp(runif(10), 2, show = 7, plotDelay = -1)
}

 # overlay visual exploration of ks.test results
 oldpar <- par(no.readonly = TRUE)
 set.seed(54321)
 vals <- iexp(runif(10), 2, showECDF = TRUE, restorePar = FALSE)
 D <- as.numeric(ks.test(vals, "pexp", 2)$statistic)
 for (x in seq(0.25, 0.65, by = 0.05)) {
   y <- pexp(x, 2)
   segments(x, y, x, y + D, col = "darkgreen", lwd = 2, xpd = NA)
 }
 par(oldpar) # restore original par values, since restorePar = FALSE above
```

`ifd` *Visualization of Random Variate Generation for the FALSE Distribution*

Description

Generates random variates from the FALSE distribution by inversion. Optionally graphs the population cumulative distribution function and associated random variates, the population probability density function and a histogram of the random variates, and the empirical cumulative distribution function versus the population cumulative distribution function.

Usage

```
ifd(
  u = runif(1),
  df1,
  df2,
  ncp = 0,
```

```
    minPlotQuantile = 0.01,
    maxPlotQuantile = 0.99,
    plot = TRUE,
    showCDF = TRUE,
    showPDF = TRUE,
    showECDF = TRUE,
    show = NULL,
    maxInvPlotted = 50,
    plotDelay = 0,
    sampleColor = "red3",
    populationColor = "grey",
    showTitle = TRUE,
    respectLayout = FALSE,
    restorePar = TRUE,
    ...
)
```

Arguments

`u`	vector of uniform(0,1) random numbers, or NULL to show population figures only
`df1`	Degrees of freedom > 0
`df2`	Degrees of freedom > 0
`ncp`	Non-centrality parameter >= 0
`minPlotQuantile`	minimum quantile to plot
`maxPlotQuantile`	maximum quantile to plot
`plot`	logical; if `TRUE` (default), one or more plots will appear (see parameters below); otherwise no plots appear
`showCDF`	logical; if `TRUE` (default), cdf plot appears, otherwise cdf plot is suppressed
`showPDF`	logical; if `TRUE` (default), PDF plot appears, otherwise PDF plot is suppressed
`showECDF`	logical; if `TRUE` (default), ecdf plot appears, otherwise ecdf plot is suppressed
`show`	octal number (0-7) indicating plots to display; 4: CDF, 2: PDF, 1: ECDF; sum for desired combination
`maxInvPlotted`	number of inversions to plot across CDF before switching to plotting quantiles only
`plotDelay`	delay in seconds between CDF plots
`sampleColor`	Color used to display random sample from distribution
`populationColor`	Color used to display population
`showTitle`	logical; if `TRUE` (default), displays a title in the first of any displayed plots
`respectLayout`	logical; if `TRUE` (default), respects existing settings for device layout
`restorePar`	logical; if `TRUE` (default), restores user's previous par settings on function exit
`...`	Possible additional arguments. Currently, additional arguments not considered.

Details

Generates random variates from the FALSE distribution, and optionally, illustrates

- the use of the inverse-CDF technique,
- the effect of random sampling variability in relation to the PDF and CDF.

When all of the graphics are requested,

- the first graph illustrates the use of the inverse-CDF technique by graphing the population CDF and the transformation of the random numbers to random variates,
- the second graph illustrates the effect of random sampling variability by graphing the population PDF and the histogram associated with the random variates, and
- the third graph illustrates effect of random sampling variability by graphing the population CDF and the empirical CDF associated with the random variates.

All aspects of the random variate generation algorithm are output in red by default, which can be changed by specifying `sampleColor`. All aspects of the population distribution are output in gray by default, which can be changed by specifying `populationColor`.

The F distribution with `df1` $= n_1$ and `df2` $= n_2$ degrees of freedom has density

$$f(x) = \frac{\Gamma(n_1/2 + n_2/2)}{\Gamma(n_1/2)\,\Gamma(n_2/2)} \left(\frac{n_1}{n_2}\right)^{n_1/2} x^{n_1/2-1} \left(1 + \frac{n_1 x}{n_2}\right)^{-(n_1+n_2)/2}$$

for $x > 0$.

The algorithm for generating random variates from the FALSE distribution is synchronized (one random variate for each random number) and monotone in u. This means that the variates generated here might be useful in some variance reduction techniques used in Monte Carlo and discrete-event simulation.

Values from the u vector are plotted in the cdf plot along the vertical axis as colored dots. A horizontal, dashed, colored line extends from the dot to the population cdf. At the intersection, a vertical, dashed colored line extends downward to the horizontal axis, where a second colored dot, denoting the associated FALSE random variate is plotted.

This is not a particularly fast variate generation algorithm because it uses the base R `qf` function to invert the values contained in `u`.

All of the elements of the `u` vector must be between 0 and 1. Alternatively, `u` can be `NULL` in which case plot(s) of the theoretical PDF and cdf are displayed according to plotting parameter values (defaulting to display of both the PDF and cdf).

The `show` parameter can be used as a shortcut way to denote plots to display. The argument to `show` can be either:

- a binary vector of length three, where the entries from left to right correspond to `showCDF`, `showPDF`, and `showECDF`, respectively. For each entry, a 1 indicates the plot should be displayed, and a 0 indicates the plot should be suppressed.
- an integer in [0,7] interpreted similar to the Unix chmod command. That is, the integer's binary representation can be transformed into a length-three vector discussed above (e.g., 6 corresponds to c(1,1,0)). See examples.

Any valid value for show takes precedence over existing individual values for showCDF, showPDF, and showECDF.

If respectLayout is TRUE, the function respects existing settings for device layout. Note, however, that if the number of plots requested (either via show or via showCDF, showPMF, and showECDF) exceeds the number of plots available in the current layout (as determined by prod(par("mfrow"))), the function will display all requested plots but will also display a warning message indicating that the current layout does not permit simultaneous viewing of all requested plots. The most recent plot with this attribute can be further annotated after the call.

If respectLayout is FALSE, any existing user settings for device layout are ignored. That is, the function uses par to explicitly set mfrow sufficient to show all requested plots stacked vertically to align their horizontal axes, and then resets row, column, and margin settings to their prior state on exit.

The minPlotQuantile and maxPlotQuantile arguments are present in order to compress the plots horizontally. The random variates generated are not impacted by these two arguments. Vertical, dotted, black lines are plotted at the associated quantiles on the plots.

plotDelay can be used to slow down or halt the variate generation for classroom explanation.

In the plot associated with the PDF, the maximum plotting height is associated with 125\ that extends above this limit will have three dots appearing above it.

Value

A vector of FALSE random variates

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

stats::rf

stats::runif, simEd::vunif

Examples

```
ifd(0.5, df1 = 1, df2 = 2, ncp = 10)

set.seed(8675309)
ifd(runif(10), 5, 5, showPDF = TRUE)

set.seed(8675309)
ifd(runif(10), 5, 5, showECDF = TRUE)

set.seed(8675309)
ifd(runif(10), 5, 5, showPDF = TRUE, showECDF = TRUE, sampleColor = "blue3")

set.seed(8675309)
ifd(runif(10), 5, 5, showPDF = TRUE, showCDF = FALSE)
```

```
ifd(runif(100), 5, 5, showPDF = TRUE, minPlotQuantile = 0.02, maxPlotQuantile = 0.98)

# plot the PDF and CDF without any variates
ifd(NULL, 5, 5, showPDF = TRUE, showCDF = TRUE)

# plot CDF with inversion and PDF using show
ifd(runif(10), 5, 5, show = c(1,1,0))
ifd(runif(10), 5, 5, show = 6)

# plot CDF with inversion and ECDF using show, using vunif
ifd(vunif(10), 5, 5, show = c(1,0,1))
ifd(vunif(10), 5, 5, show = 5)

# plot CDF with inversion, PDF, and ECDF using show
ifd(vunif(10), 5, 5, show = c(1,1,1))
ifd(vunif(10), 5, 5, show = 7)

# plot three different CDF+PDF+ECDF horizontal displays,
# with title only on the first display
oldpar <- par(no.readonly = TRUE)
par(mfrow = c(3,3))  # 3 rows, 3 cols, filling rows before columns
set.seed(8675309)
ifd(runif(20), 5, 5, show = 7, respectLayout = TRUE, restorePar = FALSE)
ifd(runif(20), 5, 5, show = 7, respectLayout = TRUE, restorePar = FALSE, showTitle = FALSE)
ifd(runif(20), 5, 5, show = 7, respectLayout = TRUE, restorePar = TRUE,  showTitle = FALSE)
par(oldpar)

# display animation of all components
ifd(runif(10), 5, 5, show = 7, plotDelay = 0.1)

# display animation of CDF and PDF components only
ifd(runif(10), 5, 5, show = 5, plotDelay = 0.1)

if (interactive()) {
  # interactive -- pause at each stage of inversion
  ifd(runif(10), 5, 5, show = 7, plotDelay = -1)
}
```

---

igamma	*Visualization of Random Variate Generation for the Gamma Distribution*

Description

Generates random variates from the Gamma distribution by inversion. Optionally graphs the population cumulative distribution function and associated random variates, the population probability density function and a histogram of the random variates, and the empirical cumulative distribution function versus the population cumulative distribution function.

Usage

```
igamma(
  u = runif(1),
  shape,
  rate = 1,
  scale = 1/rate,
  minPlotQuantile = 0,
  maxPlotQuantile = 0.95,
  plot = TRUE,
  showCDF = TRUE,
  showPDF = TRUE,
  showECDF = TRUE,
  show = NULL,
  maxInvPlotted = 50,
  plotDelay = 0,
  sampleColor = "red3",
  populationColor = "grey",
  showTitle = TRUE,
  respectLayout = FALSE,
  restorePar = TRUE,
  ...
)
```

Arguments

`u`	vector of uniform(0,1) random numbers, or NULL to show population figures only
`shape`	Shape parameter
`rate`	Alternate parameterization for scale
`scale`	Scale parameter
`minPlotQuantile`	minimum quantile to plot
`maxPlotQuantile`	maximum quantile to plot
`plot`	logical; if `TRUE` (default), one or more plots will appear (see parameters below); otherwise no plots appear
`showCDF`	logical; if `TRUE` (default), cdf plot appears, otherwise cdf plot is suppressed
`showPDF`	logical; if `TRUE` (default), PDF plot appears, otherwise PDF plot is suppressed
`showECDF`	logical; if `TRUE` (default), ecdf plot appears, otherwise ecdf plot is suppressed
`show`	octal number (0-7) indicating plots to display; 4: CDF, 2: PDF, 1: ECDF; sum for desired combination
`maxInvPlotted`	number of inversions to plot across CDF before switching to plotting quantiles only
`plotDelay`	delay in seconds between CDF plots
`sampleColor`	Color used to display random sample from distribution

populationColor
: Color used to display population

showTitle
: logical; if TRUE (default), displays a title in the first of any displayed plots

respectLayout
: logical; if TRUE (default), respects existing settings for device layout

restorePar
: logical; if TRUE (default), restores user's previous par settings on function exit

...
: Possible additional arguments. Currently, additional arguments not considered.

Details

Generates random variates from the Gamma distribution, and optionally, illustrates

- the use of the inverse-CDF technique,
- the effect of random sampling variability in relation to the PDF and CDF.

When all of the graphics are requested,

- the first graph illustrates the use of the inverse-CDF technique by graphing the population CDF and the transformation of the random numbers to random variates,
- the second graph illustrates the effect of random sampling variability by graphing the population PDF and the histogram associated with the random variates, and
- the third graph illustrates effect of random sampling variability by graphing the population CDF and the empirical CDF associated with the random variates.

All aspects of the random variate generation algorithm are output in red by default, which can be changed by specifying `sampleColor`. All aspects of the population distribution are output in gray by default, which can be changed by specifying `populationColor`.

```
The gamma distribution with parameters \code{shape} = \eqn{a} and
\code{scale} = \eqn{s} has density

     \deqn{f(x) = \frac{1}{s^a\, \Gamma(a)} x^{a-1} e^{-x/s}}{
               f(x) = 1/(s^a Gamma(a)) x^(a-1) e^(-x/s)}

for \eqn{x \ge 0}, \eqn{a > 0}, and \eqn{s > 0}.
(Here \eqn{\Gamma(a)}{Gamma(a)} is the function implemented by
R's \code{\link[base:Special]{gamma}()} and defined in its help.)

The population mean and variance are \eqn{E(X) = as}
and \eqn{Var(X) = as^2}.
```

The algorithm for generating random variates from the gamma distribution is synchronized (one random variate for each random number) and monotone in u. This means that the variates generated here might be useful in some variance reduction techniques used in Monte Carlo and discrete-event simulation.

Values from the u vector are plotted in the cdf plot along the vertical axis as colored dots. A horizontal, dashed, colored line extends from the dot to the population cdf. At the intersection, a vertical, dashed colored line extends downward to the horizontal axis, where a second colored dot, denoting the associated gamma random variate is plotted.

This is not a particularly fast variate generation algorithm because it uses the base R qgamma function to invert the values contained in u.

All of the elements of the u vector must be between 0 and 1. Alternatively, u can be NULL in which case plot(s) of the theoretical PDF and cdf are displayed according to plotting parameter values (defaulting to display of both the PDF and cdf).

The show parameter can be used as a shortcut way to denote plots to display. The argument to show can be either:

- a binary vector of length three, where the entries from left to right correspond to showCDF, showPDF, and showECDF, respectively. For each entry, a 1 indicates the plot should be displayed, and a 0 indicates the plot should be suppressed.
- an integer in [0,7] interpreted similar to the Unix chmod command. That is, the integer's binary representation can be transformed into a length-three vector discussed above (e.g., 6 corresponds to c(1,1,0)). See examples.

Any valid value for show takes precedence over existing individual values for showCDF, showPDF, and showECDF.

If respectLayout is TRUE, the function respects existing settings for device layout. Note, however, that if the number of plots requested (either via show or via showCDF, showPMF, and showECDF) exceeds the number of plots available in the current layout (as determined by prod(par("mfrow"))), the function will display all requested plots but will also display a warning message indicating that the current layout does not permit simultaneous viewing of all requested plots. The most recent plot with this attribute can be further annotated after the call.

If respectLayout is FALSE, any existing user settings for device layout are ignored. That is, the function uses par to explicitly set mfrow sufficient to show all requested plots stacked vertically to align their horizontal axes, and then resets row, column, and margin settings to their prior state on exit.

The minPlotQuantile and maxPlotQuantile arguments are present in order to compress the plots horizontally. The random variates generated are not impacted by these two arguments. Vertical, dotted, black lines are plotted at the associated quantiles on the plots.

plotDelay can be used to slow down or halt the variate generation for classroom explanation.

In the plot associated with the PDF, the maximum plotting height is associated with 125\ that extends above this limit will have three dots appearing above it.

Value

A vector of Gamma random variates

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

stats::rgamma

stats::runif, simEd::vunif

Examples

```
igamma(0.5, shape = 5, scale = 3)

set.seed(8675309)
igamma(runif(10), 3, 2, showPDF = TRUE)

set.seed(8675309)
igamma(runif(10), 3, 2, showECDF = TRUE)

set.seed(8675309)
igamma(runif(10), 3, 2, showPDF = TRUE, showECDF = TRUE, sampleColor = "blue3")

set.seed(8675309)
igamma(runif(10), 3, 2, showPDF = TRUE, showCDF = FALSE)

igamma(runif(100), 3, 2, showPDF = TRUE, minPlotQuantile = 0.02, maxPlotQuantile = 0.98)

# plot the PDF and CDF without any variates
igamma(NULL, 3, 2, showPDF = TRUE, showCDF = TRUE)

# plot CDF with inversion and PDF using show
igamma(runif(10), 3, 2, show = c(1,1,0))
igamma(runif(10), 3, 2, show = 6)

# plot CDF with inversion and ECDF using show, using vunif
igamma(vunif(10), 3, 2, show = c(1,0,1))
igamma(vunif(10), 3, 2, show = 5)

# plot CDF with inversion, PDF, and ECDF using show
igamma(vunif(10), 3, 2, show = c(1,1,1))
igamma(vunif(10), 3, 2, show = 7)

# plot three different CDF+PDF+ECDF horizontal displays,
# with title only on the first display
oldpar <- par(no.readonly = TRUE)
par(mfrow = c(3,3))  # 3 rows, 3 cols, filling rows before columns
set.seed(8675309)
igamma(runif(20), 3, 2, show = 7, respectLayout = TRUE, restorePar = FALSE)
igamma(runif(20), 3, 2, show = 7, respectLayout = TRUE, restorePar = FALSE, showTitle = FALSE)
igamma(runif(20), 3, 2, show = 7, respectLayout = TRUE, restorePar = TRUE,  showTitle = FALSE)
par(oldpar)

# display animation of all components
igamma(runif(10), 3, 2, show = 7, plotDelay = 0.1)

# display animation of CDF and PDF components only
igamma(runif(10), 3, 2, show = 5, plotDelay = 0.1)

if (interactive()) {
  # interactive -- pause at each stage of inversion
  igamma(runif(10), 3, 2, show = 7, plotDelay = -1)
}
```

```
# overlay visual exploration of ks.test results
oldpar <- par(no.readonly = TRUE)
set.seed(54321)
vals <- igamma(runif(10), 3, 2, showECDF = TRUE, restorePar = FALSE)
D <- as.numeric(ks.test(vals, "pgamma", 3, 2)$statistic)
for (x in seq(1.20, 1.60, by = 0.05)) {
  y <- pgamma(x, 3, 2)
  segments(x, y, x, y + D, col = "darkgreen", lwd = 2, xpd = NA)
}
par(oldpar) # restore original par values, since restorePar = FALSE above
```

---

igeom *Visualization of Random Variate Generation for the Geometric Distribution*

Description

Generates random variates from the Geometric distribution by inversion. Optionally graphs the population cumulative distribution function and associated random variates, the population probability mass function and a histogram of the random variates, and the empirical cumulative distribution function versus the population cumulative distribution function.

Usage

```
igeom(
  u = runif(1),
  prob,
  minPlotQuantile = 0,
  maxPlotQuantile = 0.95,
  plot = TRUE,
  showCDF = TRUE,
  showPMF = TRUE,
  showECDF = TRUE,
  show = NULL,
  maxInvPlotted = 50,
  plotDelay = 0,
  sampleColor = "red3",
  populationColor = "grey",
  showTitle = TRUE,
  respectLayout = FALSE,
  restorePar = TRUE,
  ...
)
```

Arguments

`u`	vector of uniform(0,1) random numbers, or NULL to show population figures only
`prob`	Probability of success in each trial ($0 <$ `prob` ≤ 1)
`minPlotQuantile`	minimum quantile to plot
`maxPlotQuantile`	maximum quantile to plot
`plot`	logical; if `TRUE` (default), one or more plots will appear (see parameters below); otherwise no plots appear
`showCDF`	logical; if `TRUE` (default), cdf plot appears, otherwise cdf plot is suppressed
`showPMF`	logical; if `TRUE` (default), PMF plot appears, otherwise PMF plot is suppressed
`showECDF`	logical; if `TRUE` (default), ecdf plot appears, otherwise ecdf plot is suppressed
`show`	octal number (0-7) indicating plots to display; 4: CDF, 2: PMF, 1: ECDF; sum for desired combination
`maxInvPlotted`	number of inversions to plot across CDF before switching to plotting quantiles only
`plotDelay`	delay in seconds between CDF plots
`sampleColor`	Color used to display random sample from distribution
`populationColor`	Color used to display population
`showTitle`	logical; if `TRUE` (default), displays a title in the first of any displayed plots
`respectLayout`	logical; if `TRUE` (default), respects existing settings for device layout
`restorePar`	logical; if `TRUE` (default), restores user's previous par settings on function exit
`...`	Possible additional arguments. Currently, additional arguments not considered.

Details

Generates random variates from the Geometric distribution, and optionally, illustrates

- the use of the inverse-CDF technique,
- the effect of random sampling variability in relation to the PMF and CDF.

When all of the graphics are requested,

- the first graph illustrates the use of the inverse-CDF technique by graphing the population CDF and the transformation of the random numbers to random variates,
- the second graph illustrates the effect of random sampling variability by graphing the population PMF and the histogram associated with the random variates, and
- the third graph illustrates effect of random sampling variability by graphing the population CDF and the empirical CDF associated with the random variates.

All aspects of the random variate generation algorithm are output in red by default, which can be changed by specifying `sampleColor`. All aspects of the population distribution are output in gray by default, which can be changed by specifying `populationColor`.

The geometric distribution with parameter `prob` = p has density

$$p(x) = p(1-p)^x$$

for $x = 0, 1, 2, \ldots$, where $0 < p \leq 1$.

The algorithm for generating random variates from the geometric distribution is synchronized (one random variate for each random number) and monotone in u. This means that the variates generated here might be useful in some variance reduction techniques used in Monte Carlo and discrete-event simulation.

Values from the u vector are plotted in the cdf plot along the vertical axis as colored dots. A horizontal, dashed, colored line extends from the dot to the population cdf. At the intersection, a vertical, dashed colored line extends downward to the horizontal axis, where a second colored dot, denoting the associated geometric random variate is plotted.

This is not a particularly fast variate generation algorithm because it uses the base R `qgeom` function to invert the values contained in `u`.

All of the elements of the `u` vector must be between 0 and 1. Alternatively, `u` can be `NULL` in which case plot(s) of the theoretical PMF and cdf are displayed according to plotting parameter values (defaulting to display of both the PMF and cdf).

The `show` parameter can be used as a shortcut way to denote plots to display. The argument to `show` can be either:

- a binary vector of length three, where the entries from left to right correspond to `showCDF`, `showPMF`, and `showECDF`, respectively. For each entry, a 1 indicates the plot should be displayed, and a 0 indicates the plot should be suppressed.
- an integer in [0,7] interpreted similar to the Unix chmod command. That is, the integer's binary representation can be transformed into a length-three vector discussed above (e.g., 6 corresponds to c(1,1,0)). See examples.

Any valid value for `show` takes precedence over existing individual values for `showCDF`, `showPMF`, and `showECDF`.

If `respectLayout` is `TRUE`, the function respects existing settings for device layout. Note, however, that if the number of plots requested (either via `show` or via `showCDF`, `showPMF`, and `showECDF`) exceeds the number of plots available in the current layout (as determined by `prod(par("mfrow"))`), the function will display all requested plots but will also display a warning message indicating that the current layout does not permit simultaneous viewing of all requested plots. The most recent plot with this attribute can be further annotated after the call.

If `respectLayout` is `FALSE`, any existing user settings for device layout are ignored. That is, the function uses `par` to explicitly set `mfrow` sufficient to show all requested plots stacked vertically to align their horizontal axes, and then resets row, column, and margin settings to their prior state on exit.

The `minPlotQuantile` and `maxPlotQuantile` arguments are present in order to compress the plots horizontally. The random variates generated are not impacted by these two arguments. Vertical, dotted, black lines are plotted at the associated quantiles on the plots.

plotDelay can be used to slow down or halt the variate generation for classroom explanation.

In the plot associated with the PMF, the maximum plotting height is associated with 125\ that extends above this limit will have three dots appearing above it.

Value

A vector of Geometric random variates

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

stats::rgeom

stats::runif, simEd::vunif

Examples

```
igeom(0.5, prob = 0.25)

set.seed(8675309)
igeom(runif(10), 0.4, showPMF = TRUE)

set.seed(8675309)
igeom(runif(10), 0.4, showECDF = TRUE)

set.seed(8675309)
igeom(runif(10), 0.4, showPMF = TRUE, showECDF = TRUE, sampleColor = "blue3")

set.seed(8675309)
igeom(runif(10), 0.4, showPMF = TRUE, showCDF = FALSE)

igeom(runif(100), 0.4, showPMF = TRUE, minPlotQuantile = 0.02, maxPlotQuantile = 0.98)

# plot the PMF and CDF without any variates
igeom(NULL, 0.4, showPMF = TRUE, showCDF = TRUE)

# plot CDF with inversion and PMF using show
igeom(runif(10), 0.4, show = c(1,1,0))
igeom(runif(10), 0.4, show = 6)

# plot CDF with inversion and ECDF using show, using vunif
igeom(vunif(10), 0.4, show = c(1,0,1))
igeom(vunif(10), 0.4, show = 5)

# plot CDF with inversion, PMF, and ECDF using show
igeom(vunif(10), 0.4, show = c(1,1,1))
igeom(vunif(10), 0.4, show = 7)
```

```
# plot three different CDF+PMF+ECDF horizontal displays,
# with title only on the first display
oldpar <- par(no.readonly = TRUE)
par(mfrow = c(3,3))  # 3 rows, 3 cols, filling rows before columns
set.seed(8675309)
igeom(runif(20), 0.4, show = 7, respectLayout = TRUE, restorePar = FALSE)
igeom(runif(20), 0.4, show = 7, respectLayout = TRUE, restorePar = FALSE, showTitle = FALSE)
igeom(runif(20), 0.4, show = 7, respectLayout = TRUE, restorePar = TRUE,  showTitle = FALSE)
par(oldpar)

# display animation of all components
igeom(runif(10), 0.4, show = 7, plotDelay = 0.1)

# display animation of CDF and PMF components only
igeom(runif(10), 0.4, show = 5, plotDelay = 0.1)

if (interactive()) {
  # interactive -- pause at each stage of inversion
  igeom(runif(10), 0.4, show = 7, plotDelay = -1)
}
```

---

ilnorm *Visualization of Random Variate Generation for the Log-Normal Distribution*

Description

Generates random variates from the Log-Normal distribution by inversion. Optionally graphs the population cumulative distribution function and associated random variates, the population probability density function and a histogram of the random variates, and the empirical cumulative distribution function versus the population cumulative distribution function.

Usage

```
ilnorm(
  u = runif(1),
  meanlog = 0,
  sdlog = 1,
  minPlotQuantile = 0,
  maxPlotQuantile = 0.95,
  plot = TRUE,
  showCDF = TRUE,
  showPDF = TRUE,
  showECDF = TRUE,
  show = NULL,
  maxInvPlotted = 50,
  plotDelay = 0,
```

```
    sampleColor = "red3",
    populationColor = "grey",
    showTitle = TRUE,
    respectLayout = FALSE,
    restorePar = TRUE,
    ...
)
```

Arguments

u	vector of uniform(0,1) random numbers, or NULL to show population figures only
meanlog	Mean of distribution on log scale (default 0)
sdlog	Standard deviation of distribution on log scale (default 1)
minPlotQuantile	minimum quantile to plot
maxPlotQuantile	maximum quantile to plot
plot	logical; if TRUE (default), one or more plots will appear (see parameters below); otherwise no plots appear
showCDF	logical; if TRUE (default), cdf plot appears, otherwise cdf plot is suppressed
showPDF	logical; if TRUE (default), PDF plot appears, otherwise PDF plot is suppressed
showECDF	logical; if TRUE (default), ecdf plot appears, otherwise ecdf plot is suppressed
show	octal number (0-7) indicating plots to display; 4: CDF, 2: PDF, 1: ECDF; sum for desired combination
maxInvPlotted	number of inversions to plot across CDF before switching to plotting quantiles only
plotDelay	delay in seconds between CDF plots
sampleColor	Color used to display random sample from distribution
populationColor	Color used to display population
showTitle	logical; if TRUE (default), displays a title in the first of any displayed plots
respectLayout	logical; if TRUE (default), respects existing settings for device layout
restorePar	logical; if TRUE (default), restores user's previous par settings on function exit
...	Possible additional arguments. Currently, additional arguments not considered.

Details

Generates random variates from the Log-Normal distribution, and optionally, illustrates

- the use of the inverse-CDF technique,
- the effect of random sampling variability in relation to the PDF and CDF.

When all of the graphics are requested,

- the first graph illustrates the use of the inverse-CDF technique by graphing the population CDF and the transformation of the random numbers to random variates,
- the second graph illustrates the effect of random sampling variability by graphing the population PDF and the histogram associated with the random variates, and
- the third graph illustrates effect of random sampling variability by graphing the population CDF and the empirical CDF associated with the random variates.

All aspects of the random variate generation algorithm are output in red by default, which can be changed by specifying `sampleColor`. All aspects of the population distribution are output in gray by default, which can be changed by specifying `populationColor`.

The log-normal distribution has density

```
\deqn{f(x) = \frac{1}{\sqrt{2 \pi} \sigma x}
                  e^{-(\log{x} - \mu)^2 / (2 \sigma^2)} }{
          f(x) = 1/(\sqrt(2 \pi) \sigma x) e^-((log x - \mu)^2 / (2 \sigma^2))}
```

where μ and σ are the mean and standard deviation of the logarithm.

The mean is $E(X) = \exp(\mu + 1/2\sigma^2)$, the median is $med(X) = \exp(\mu)$, and the variance is $Var(X) = \exp(2\times\mu+\sigma^2)\times(\exp(\sigma^2)-1)$ and hence the coefficient of variation is $sqrt(\exp(\sigma^2)-1)$ which is approximately σ when small (e.g., $\sigma < 1/2$).

The algorithm for generating random variates from the log-normal distribution is synchronized (one random variate for each random number) and monotone in u. This means that the variates generated here might be useful in some variance reduction techniques used in Monte Carlo and discrete-event simulation.

Values from the u vector are plotted in the cdf plot along the vertical axis as colored dots. A horizontal, dashed, colored line extends from the dot to the population cdf. At the intersection, a vertical, dashed colored line extends downward to the horizontal axis, where a second colored dot, denoting the associated log-normal random variate is plotted.

This is not a particularly fast variate generation algorithm because it uses the base R `qlnorm` function to invert the values contained in `u`.

All of the elements of the `u` vector must be between 0 and 1. Alternatively, `u` can be `NULL` in which case plot(s) of the theoretical PDF and cdf are displayed according to plotting parameter values (defaulting to display of both the PDF and cdf).

The `show` parameter can be used as a shortcut way to denote plots to display. The argument to `show` can be either:

- a binary vector of length three, where the entries from left to right correspond to `showCDF`, `showPDF`, and `showECDF`, respectively. For each entry, a 1 indicates the plot should be displayed, and a 0 indicates the plot should be suppressed.
- an integer in [0,7] interpreted similar to the Unix chmod command. That is, the integer's binary representation can be transformed into a length-three vector discussed above (e.g., 6 corresponds to c(1,1,0)). See examples.

Any valid value for `show` takes precedence over existing individual values for `showCDF`, `showPDF`, and `showECDF`.

If respectLayout is TRUE, the function respects existing settings for device layout. Note, however, that if the number of plots requested (either via show or via showCDF, showPMF, and showECDF) exceeds the number of plots available in the current layout (as determined by prod(par("mfrow"))), the function will display all requested plots but will also display a warning message indicating that the current layout does not permit simultaneous viewing of all requested plots. The most recent plot with this attribute can be further annotated after the call.

If respectLayout is FALSE, any existing user settings for device layout are ignored. That is, the function uses par to explicitly set mfrow sufficient to show all requested plots stacked vertically to align their horizontal axes, and then resets row, column, and margin settings to their prior state on exit.

The minPlotQuantile and maxPlotQuantile arguments are present in order to compress the plots horizontally. The random variates generated are not impacted by these two arguments. Vertical, dotted, black lines are plotted at the associated quantiles on the plots.

plotDelay can be used to slow down or halt the variate generation for classroom explanation.

In the plot associated with the PDF, the maximum plotting height is associated with 125\ that extends above this limit will have three dots appearing above it.

Value

A vector of Log-Normal random variates

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

stats::rlnorm

stats::runif, simEd::vunif

Examples

```
ilnorm(0.5, meanlog = 5, sdlog = 0.5)

set.seed(8675309)
ilnorm(runif(10), 8, 2, showPDF = TRUE)

set.seed(8675309)
ilnorm(runif(10), 8, 2, showECDF = TRUE)

set.seed(8675309)
ilnorm(runif(10), 8, 2, showPDF = TRUE, showECDF = TRUE, sampleColor = "blue3")

set.seed(8675309)
ilnorm(runif(10), 8, 2, showPDF = TRUE, showCDF = FALSE)

ilnorm(runif(100), 8, 2, showPDF = TRUE, minPlotQuantile = 0.02, maxPlotQuantile = 0.98)
```

```
# plot the PDF and CDF without any variates
ilnorm(NULL, 8, 2, showPDF = TRUE, showCDF = TRUE)

# plot CDF with inversion and PDF using show
ilnorm(runif(10), 8, 2, show = c(1,1,0))
ilnorm(runif(10), 8, 2, show = 6)

# plot CDF with inversion and ECDF using show, using vunif
ilnorm(vunif(10), 8, 2, show = c(1,0,1))
ilnorm(vunif(10), 8, 2, show = 5)

# plot CDF with inversion, PDF, and ECDF using show
ilnorm(vunif(10), 8, 2, show = c(1,1,1))
ilnorm(vunif(10), 8, 2, show = 7)

# plot three different CDF+PDF+ECDF horizontal displays,
# with title only on the first display
oldpar <- par(no.readonly = TRUE)
par(mfrow = c(3,3))  # 3 rows, 3 cols, filling rows before columns
set.seed(8675309)
ilnorm(runif(20), 8, 2, show = 7, respectLayout = TRUE, restorePar = FALSE)
ilnorm(runif(20), 8, 2, show = 7, respectLayout = TRUE, restorePar = FALSE, showTitle = FALSE)
ilnorm(runif(20), 8, 2, show = 7, respectLayout = TRUE, restorePar = TRUE,  showTitle = FALSE)
par(oldpar)

# display animation of all components
ilnorm(runif(10), 8, 2, show = 7, plotDelay = 0.1)

# display animation of CDF and PDF components only
ilnorm(runif(10), 8, 2, show = 5, plotDelay = 0.1)

if (interactive()) {
  # interactive -- pause at each stage of inversion
  ilnorm(runif(10), 8, 2, show = 7, plotDelay = -1)
}
```

`ilogis` *Visualization of Random Variate Generation for the Logistic Distribution*

Description

Generates random variates from the Logistic distribution by inversion. Optionally graphs the population cumulative distribution function and associated random variates, the population probability density function and a histogram of the random variates, and the empirical cumulative distribution function versus the population cumulative distribution function.

Usage

```
ilogis(
  u = runif(1),
  location = 0,
  scale = 1,
  minPlotQuantile = 0.01,
  maxPlotQuantile = 0.99,
  plot = TRUE,
  showCDF = TRUE,
  showPDF = TRUE,
  showECDF = TRUE,
  show = NULL,
  maxInvPlotted = 50,
  plotDelay = 0,
  sampleColor = "red3",
  populationColor = "grey",
  showTitle = TRUE,
  respectLayout = FALSE,
  restorePar = TRUE,
  ...
)
```

Arguments

`u`	vector of uniform(0,1) random numbers, or NULL to show population figures only
`location`	Location parameter
`scale`	Scale parameter (default 1)
`minPlotQuantile`	minimum quantile to plot
`maxPlotQuantile`	maximum quantile to plot
`plot`	logical; if `TRUE` (default), one or more plots will appear (see parameters below); otherwise no plots appear
`showCDF`	logical; if `TRUE` (default), cdf plot appears, otherwise cdf plot is suppressed
`showPDF`	logical; if `TRUE` (default), PDF plot appears, otherwise PDF plot is suppressed
`showECDF`	logical; if `TRUE` (default), ecdf plot appears, otherwise ecdf plot is suppressed
`show`	octal number (0-7) indicating plots to display; 4: CDF, 2: PDF, 1: ECDF; sum for desired combination
`maxInvPlotted`	number of inversions to plot across CDF before switching to plotting quantiles only
`plotDelay`	delay in seconds between CDF plots
`sampleColor`	Color used to display random sample from distribution
`populationColor`	Color used to display population

showTitle	logical; if TRUE (default), displays a title in the first of any displayed plots
respectLayout	logical; if TRUE (default), respects existing settings for device layout
restorePar	logical; if TRUE (default), restores user's previous par settings on function exit
...	Possible additional arguments. Currently, additional arguments not considered.

Details

Generates random variates from the Logistic distribution, and optionally, illustrates

- the use of the inverse-CDF technique,
- the effect of random sampling variability in relation to the PDF and CDF.

When all of the graphics are requested,

- the first graph illustrates the use of the inverse-CDF technique by graphing the population CDF and the transformation of the random numbers to random variates,
- the second graph illustrates the effect of random sampling variability by graphing the population PDF and the histogram associated with the random variates, and
- the third graph illustrates effect of random sampling variability by graphing the population CDF and the empirical CDF associated with the random variates.

All aspects of the random variate generation algorithm are output in red by default, which can be changed by specifying `sampleColor`. All aspects of the population distribution are output in gray by default, which can be changed by specifying `populationColor`.

The logistic distribution with `location` $= \mu$ and `scale` $= \sigma$ has distribution function

$$F(x) = \frac{1}{1 + e^{-(x-\mu)/\sigma}}$$

and density

$$f(x) = \frac{1}{\sigma} \frac{e^{(x-\mu)/\sigma}}{(1 + e^{(x-\mu)/\sigma})^2}$$

It is a long-tailed distribution with mean μ and variance $\pi^2/3\sigma^2$.

The algorithm for generating random variates from the logistic distribution is synchronized (one random variate for each random number) and monotone in u. This means that the variates generated here might be useful in some variance reduction techniques used in Monte Carlo and discrete-event simulation.

Values from the u vector are plotted in the cdf plot along the vertical axis as colored dots. A horizontal, dashed, colored line extends from the dot to the population cdf. At the intersection, a vertical, dashed colored line extends downward to the horizontal axis, where a second colored dot, denoting the associated logistic random variate is plotted.

This is not a particularly fast variate generation algorithm because it uses the base R `qlogis` function to invert the values contained in u.

All of the elements of the u vector must be between 0 and 1. Alternatively, u can be NULL in which case plot(s) of the theoretical PDF and cdf are displayed according to plotting parameter values (defaulting to display of both the PDF and cdf).

The `show` parameter can be used as a shortcut way to denote plots to display. The argument to `show` can be either:

- a binary vector of length three, where the entries from left to right correspond to `showCDF`, `showPDF`, and `showECDF`, respectively. For each entry, a 1 indicates the plot should be displayed, and a 0 indicates the plot should be suppressed.
- an integer in [0,7] interpreted similar to the Unix chmod command. That is, the integer's binary representation can be transformed into a length-three vector discussed above (e.g., 6 corresponds to c(1,1,0)). See examples.

Any valid value for `show` takes precedence over existing individual values for `showCDF`, `showPDF`, and `showECDF`.

If `respectLayout` is `TRUE`, the function respects existing settings for device layout. Note, however, that if the number of plots requested (either via `show` or via `showCDF`, `showPMF`, and `showECDF`) exceeds the number of plots available in the current layout (as determined by `prod(par("mfrow"))`), the function will display all requested plots but will also display a warning message indicating that the current layout does not permit simultaneous viewing of all requested plots. The most recent plot with this attribute can be further annotated after the call.

If `respectLayout` is `FALSE`, any existing user settings for device layout are ignored. That is, the function uses `par` to explicitly set `mfrow` sufficient to show all requested plots stacked vertically to align their horizontal axes, and then resets row, column, and margin settings to their prior state on exit.

The `minPlotQuantile` and `maxPlotQuantile` arguments are present in order to compress the plots horizontally. The random variates generated are not impacted by these two arguments. Vertical, dotted, black lines are plotted at the associated quantiles on the plots.

`plotDelay` can be used to slow down or halt the variate generation for classroom explanation.

In the plot associated with the PDF, the maximum plotting height is associated with 125\ that extends above this limit will have three dots appearing above it.

Value

A vector of Logistic random variates

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

`stats::rlogis`

`stats::runif`, `simEd::vunif`

Examples

```
ilogis(0.5, location = 5, scale = 0.5)

set.seed(8675309)
ilogis(runif(10), 5, 1.5, showPDF = TRUE)

set.seed(8675309)
ilogis(runif(10), 5, 1.5, showECDF = TRUE)

set.seed(8675309)
ilogis(runif(10), 5, 1.5, showPDF = TRUE, showECDF = TRUE, sampleColor = "blue3")

set.seed(8675309)
ilogis(runif(10), 5, 1.5, showPDF = TRUE, showCDF = FALSE)

ilogis(runif(100), 5, 1.5, showPDF = TRUE, minPlotQuantile = 0.02, maxPlotQuantile = 0.98)

# plot the PDF and CDF without any variates
ilogis(NULL, 5, 1.5, showPDF = TRUE, showCDF = TRUE)

# plot CDF with inversion and PDF using show
ilogis(runif(10), 5, 1.5, show = c(1,1,0))
ilogis(runif(10), 5, 1.5, show = 6)

# plot CDF with inversion and ECDF using show, using vunif
ilogis(vunif(10), 5, 1.5, show = c(1,0,1))
ilogis(vunif(10), 5, 1.5, show = 5)

# plot CDF with inversion, PDF, and ECDF using show
ilogis(vunif(10), 5, 1.5, show = c(1,1,1))
ilogis(vunif(10), 5, 1.5, show = 7)

# plot three different CDF+PDF+ECDF horizontal displays,
# with title only on the first display
oldpar <- par(no.readonly = TRUE)
par(mfrow = c(3,3))  # 3 rows, 3 cols, filling rows before columns
set.seed(8675309)
ilogis(runif(20), 5, 1.5, show = 7, respectLayout = TRUE, restorePar = FALSE)
ilogis(runif(20), 5, 1.5, show = 7, respectLayout = TRUE, restorePar = FALSE, showTitle = FALSE)
ilogis(runif(20), 5, 1.5, show = 7, respectLayout = TRUE, restorePar = TRUE,  showTitle = FALSE)
par(oldpar)

# display animation of all components
ilogis(runif(10), 5, 1.5, show = 7, plotDelay = 0.1)

# display animation of CDF and PDF components only
ilogis(runif(10), 5, 1.5, show = 5, plotDelay = 0.1)

if (interactive()) {
  # interactive -- pause at each stage of inversion
  ilogis(runif(10), 5, 1.5, show = 7, plotDelay = -1)
}
```

`inbinom` *Visualization of Random Variate Generation for the Negative Binomial Distribution*

Description

Generates random variates from the Negative Binomial distribution by inversion. Optionally graphs the population cumulative distribution function and associated random variates, the population probability mass function and a histogram of the random variates, and the empirical cumulative distribution function versus the population cumulative distribution function.

Usage

```
inbinom(
  u = runif(1),
  size,
  prob,
  mu,
  minPlotQuantile = 0,
  maxPlotQuantile = 0.95,
  plot = TRUE,
  showCDF = TRUE,
  showPMF = TRUE,
  showECDF = TRUE,
  show = NULL,
  maxInvPlotted = 50,
  plotDelay = 0,
  sampleColor = "red3",
  populationColor = "grey",
  showTitle = TRUE,
  respectLayout = FALSE,
  restorePar = TRUE,
  ...
)
```

Arguments

`u`	vector of uniform(0,1) random numbers, or NULL to show population figures only
`size`	target for number of successful trials, or dispersion parameter (the shape parameter of the gamma mixing distribution). Must be strictly positive, need not be integer.
`prob`	Probability of success in each trial; '0 < prob <= 1'
`mu`	alternative parameterization via mean

`minPlotQuantile`	minimum quantile to plot
`maxPlotQuantile`	maximum quantile to plot
`plot`	logical; if `TRUE` (default), one or more plots will appear (see parameters below); otherwise no plots appear
`showCDF`	logical; if `TRUE` (default), cdf plot appears, otherwise cdf plot is suppressed
`showPMF`	logical; if `TRUE` (default), PMF plot appears, otherwise PMF plot is suppressed
`showECDF`	logical; if `TRUE` (default), ecdf plot appears, otherwise ecdf plot is suppressed
`show`	octal number (0-7) indicating plots to display; 4: CDF, 2: PMF, 1: ECDF; sum for desired combination
`maxInvPlotted`	number of inversions to plot across CDF before switching to plotting quantiles only
`plotDelay`	delay in seconds between CDF plots
`sampleColor`	Color used to display random sample from distribution
`populationColor`	Color used to display population
`showTitle`	logical; if `TRUE` (default), displays a title in the first of any displayed plots
`respectLayout`	logical; if `TRUE` (default), respects existing settings for device layout
`restorePar`	logical; if `TRUE` (default), restores user's previous par settings on function exit
`...`	Possible additional arguments. Currently, additional arguments not considered.

Details

Generates random variates from the Negative Binomial distribution, and optionally, illustrates

- the use of the inverse-CDF technique,
- the effect of random sampling variability in relation to the PMF and CDF.

When all of the graphics are requested,

- the first graph illustrates the use of the inverse-CDF technique by graphing the population CDF and the transformation of the random numbers to random variates,
- the second graph illustrates the effect of random sampling variability by graphing the population PMF and the histogram associated with the random variates, and
- the third graph illustrates effect of random sampling variability by graphing the population CDF and the empirical CDF associated with the random variates.

All aspects of the random variate generation algorithm are output in red by default, which can be changed by specifying `sampleColor`. All aspects of the population distribution are output in gray by default, which can be changed by specifying `populationColor`.

The negative binomial distribution with `size` $= n$ and `prob` $= p$ has density

```
\deqn{p(x) = \frac{\Gamma(x+n)}{\Gamma(n) \ x!} p^n (1-p)^x}{
          p(x) = Gamma(x+n)/(Gamma(n) x!) p^n (1-p)^x}
```

for $x = 0, 1, 2, \ldots, n > 0$ and $0 < p \leq 1$. This represents the number of failures which occur in a sequence of Bernoulli trials before a target number of successes is reached.

The mean is $\mu = n(1-p)/p$ and variance $n(1-p)/p^2$

The algorithm for generating random variates from the negative binomial distribution is synchronized (one random variate for each random number) and monotone in u. This means that the variates generated here might be useful in some variance reduction techniques used in Monte Carlo and discrete-event simulation.

Values from the u vector are plotted in the cdf plot along the vertical axis as colored dots. A horizontal, dashed, colored line extends from the dot to the population cdf. At the intersection, a vertical, dashed colored line extends downward to the horizontal axis, where a second colored dot, denoting the associated negative binomial random variate is plotted.

This is not a particularly fast variate generation algorithm because it uses the base R `qnbinom` function to invert the values contained in `u`.

All of the elements of the `u` vector must be between 0 and 1. Alternatively, `u` can be `NULL` in which case plot(s) of the theoretical PMF and cdf are displayed according to plotting parameter values (defaulting to display of both the PMF and cdf).

The `show` parameter can be used as a shortcut way to denote plots to display. The argument to `show` can be either:

- a binary vector of length three, where the entries from left to right correspond to `showCDF`, `showPMF`, and `showECDF`, respectively. For each entry, a 1 indicates the plot should be displayed, and a 0 indicates the plot should be suppressed.
- an integer in [0,7] interpreted similar to the Unix chmod command. That is, the integer's binary representation can be transformed into a length-three vector discussed above (e.g., 6 corresponds to c(1,1,0)). See examples.

Any valid value for `show` takes precedence over existing individual values for `showCDF`, `showPMF`, and `showECDF`.

If `respectLayout` is `TRUE`, the function respects existing settings for device layout. Note, however, that if the number of plots requested (either via `show` or via `showCDF`, `showPMF`, and `showECDF`) exceeds the number of plots available in the current layout (as determined by `prod(par("mfrow"))`), the function will display all requested plots but will also display a warning message indicating that the current layout does not permit simultaneous viewing of all requested plots. The most recent plot with this attribute can be further annotated after the call.

If `respectLayout` is `FALSE`, any existing user settings for device layout are ignored. That is, the function uses `par` to explicitly set `mfrow` sufficient to show all requested plots stacked vertically to align their horizontal axes, and then resets row, column, and margin settings to their prior state on exit.

The `minPlotQuantile` and `maxPlotQuantile` arguments are present in order to compress the plots horizontally. The random variates generated are not impacted by these two arguments. Vertical, dotted, black lines are plotted at the associated quantiles on the plots.

`plotDelay` can be used to slow down or halt the variate generation for classroom explanation.

In the plot associated with the PMF, the maximum plotting height is associated with 125\ that extends above this limit will have three dots appearing above it.

Value

A vector of Negative Binomial random variates

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

stats::rnbinom

stats::runif, simEd::vunif

Examples

```
inbinom(0.5, size = 10, mu = 10)

set.seed(8675309)
inbinom(runif(10), 10, 0.25, showPMF = TRUE)

set.seed(8675309)
inbinom(runif(10), 10, 0.25, showECDF = TRUE)

set.seed(8675309)
inbinom(runif(10), 10, 0.25, showPMF = TRUE, showECDF = TRUE, sampleColor = "blue3")

set.seed(8675309)
inbinom(runif(10), 10, 0.25, showPMF = TRUE, showCDF = FALSE)

inbinom(runif(100), 10, 0.25, showPMF = TRUE, minPlotQuantile = 0.02, maxPlotQuantile = 0.98)

# plot the PMF and CDF without any variates
inbinom(NULL, 10, 0.25, showPMF = TRUE, showCDF = TRUE)

# plot CDF with inversion and PMF using show
inbinom(runif(10), 10, 0.25, show = c(1,1,0))
inbinom(runif(10), 10, 0.25, show = 6)

# plot CDF with inversion and ECDF using show, using vunif
inbinom(vunif(10), 10, 0.25, show = c(1,0,1))
inbinom(vunif(10), 10, 0.25, show = 5)

# plot CDF with inversion, PMF, and ECDF using show
inbinom(vunif(10), 10, 0.25, show = c(1,1,1))
inbinom(vunif(10), 10, 0.25, show = 7)

# plot three different CDF+PMF+ECDF horizontal displays,
# with title only on the first display
oldpar <- par(no.readonly = TRUE)
par(mfrow = c(3,3))  # 3 rows, 3 cols, filling rows before columns
```

```
set.seed(8675309)
inbinom(runif(20), 10, 0.25, show = 7, respectLayout = TRUE, restorePar = FALSE)
inbinom(runif(20), 10, 0.25, show = 7, respectLayout = TRUE, restorePar = FALSE, showTitle = FALSE)
inbinom(runif(20), 10, 0.25, show = 7, respectLayout = TRUE, restorePar = TRUE,  showTitle = FALSE)
par(oldpar)

# display animation of all components
inbinom(runif(10), 10, 0.25, show = 7, plotDelay = 0.1)

# display animation of CDF and PMF components only
inbinom(runif(10), 10, 0.25, show = 5, plotDelay = 0.1)

if (interactive()) {
  # interactive -- pause at each stage of inversion
  inbinom(runif(10), 10, 0.25, show = 7, plotDelay = -1)
}
```

`inorm` *Visualization of Random Variate Generation for the Normal Distribution*

Description

Generates random variates from the Normal distribution by inversion. Optionally graphs the population cumulative distribution function and associated random variates, the population probability density function and a histogram of the random variates, and the empirical cumulative distribution function versus the population cumulative distribution function.

Usage

```
inorm(
  u = runif(1),
  mean = 0,
  sd = 1,
  minPlotQuantile = 0.01,
  maxPlotQuantile = 0.99,
  plot = TRUE,
  showCDF = TRUE,
  showPDF = TRUE,
  showECDF = TRUE,
  show = NULL,
  maxInvPlotted = 50,
  plotDelay = 0,
  sampleColor = "red3",
  populationColor = "grey",
  showTitle = TRUE,
  respectLayout = FALSE,
```

```
  restorePar = TRUE,
  ...
)
```

Arguments

u	vector of uniform(0,1) random numbers, or NULL to show population figures only
mean	Mean of distribution (default 0)
sd	Standard deviation of distribution (default 1)
minPlotQuantile	minimum quantile to plot
maxPlotQuantile	maximum quantile to plot
plot	logical; if TRUE (default), one or more plots will appear (see parameters below); otherwise no plots appear
showCDF	logical; if TRUE (default), cdf plot appears, otherwise cdf plot is suppressed
showPDF	logical; if TRUE (default), PDF plot appears, otherwise PDF plot is suppressed
showECDF	logical; if TRUE (default), ecdf plot appears, otherwise ecdf plot is suppressed
show	octal number (0-7) indicating plots to display; 4: CDF, 2: PDF, 1: ECDF; sum for desired combination
maxInvPlotted	number of inversions to plot across CDF before switching to plotting quantiles only
plotDelay	delay in seconds between CDF plots
sampleColor	Color used to display random sample from distribution
populationColor	Color used to display population
showTitle	logical; if TRUE (default), displays a title in the first of any displayed plots
respectLayout	logical; if TRUE (default), respects existing settings for device layout
restorePar	logical; if TRUE (default), restores user's previous par settings on function exit
...	Possible additional arguments. Currently, additional arguments not considered.

Details

Generates random variates from the Normal distribution, and optionally, illustrates

- the use of the inverse-CDF technique,
- the effect of random sampling variability in relation to the PDF and CDF.

When all of the graphics are requested,

- the first graph illustrates the use of the inverse-CDF technique by graphing the population CDF and the transformation of the random numbers to random variates,
- the second graph illustrates the effect of random sampling variability by graphing the population PDF and the histogram associated with the random variates, and

- the third graph illustrates effect of random sampling variability by graphing the population CDF and the empirical CDF associated with the random variates.

All aspects of the random variate generation algorithm are output in red by default, which can be changed by specifying `sampleColor`. All aspects of the population distribution are output in gray by default, which can be changed by specifying `populationColor`.

The normal distribution has density

```
\deqn{f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x - \mu)^2/(2 \sigma^2)}}{
         f(x) = 1/(\sqrt(2\pi)\sigma) e^(-(x - \mu)^2/(2 \sigma^2))}
```

for $-\infty < x < \infty$ and $\sigma > 0$, where μ is the mean of the distribution and σ the standard deviation.

The algorithm for generating random variates from the normal distribution is synchronized (one random variate for each random number) and monotone in u. This means that the variates generated here might be useful in some variance reduction techniques used in Monte Carlo and discrete-event simulation.

Values from the u vector are plotted in the cdf plot along the vertical axis as colored dots. A horizontal, dashed, colored line extends from the dot to the population cdf. At the intersection, a vertical, dashed colored line extends downward to the horizontal axis, where a second colored dot, denoting the associated normal random variate is plotted.

This is not a particularly fast variate generation algorithm because it uses the base R `qnorm` function to invert the values contained in `u`.

All of the elements of the `u` vector must be between 0 and 1. Alternatively, `u` can be `NULL` in which case plot(s) of the theoretical PDF and cdf are displayed according to plotting parameter values (defaulting to display of both the PDF and cdf).

The `show` parameter can be used as a shortcut way to denote plots to display. The argument to `show` can be either:

- a binary vector of length three, where the entries from left to right correspond to `showCDF`, `showPDF`, and `showECDF`, respectively. For each entry, a 1 indicates the plot should be displayed, and a 0 indicates the plot should be suppressed.
- an integer in [0,7] interpreted similar to the Unix chmod command. That is, the integer's binary representation can be transformed into a length-three vector discussed above (e.g., 6 corresponds to c(1,1,0)). See examples.

Any valid value for `show` takes precedence over existing individual values for `showCDF`, `showPDF`, and `showECDF`.

If `respectLayout` is `TRUE`, the function respects existing settings for device layout. Note, however, that if the number of plots requested (either via `show` or via `showCDF`, `showPMF`, and `showECDF`) exceeds the number of plots available in the current layout (as determined by `prod(par("mfrow"))`), the function will display all requested plots but will also display a warning message indicating that the current layout does not permit simultaneous viewing of all requested plots. The most recent plot with this attribute can be further annotated after the call.

If `respectLayout` is `FALSE`, any existing user settings for device layout are ignored. That is, the function uses `par` to explicitly set `mfrow` sufficient to show all requested plots stacked vertically to align their horizontal axes, and then resets row, column, and margin settings to their prior state on exit.

The `minPlotQuantile` and `maxPlotQuantile` arguments are present in order to compress the plots horizontally. The random variates generated are not impacted by these two arguments. Vertical, dotted, black lines are plotted at the associated quantiles on the plots.

`plotDelay` can be used to slow down or halt the variate generation for classroom explanation.

In the plot associated with the PDF, the maximum plotting height is associated with 125\ that extends above this limit will have three dots appearing above it.

Value

A vector of Normal random variates

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

stats::rnorm

stats::runif, simEd::vunif

Examples

```
inorm(0.5, mean = 3, sd = 1)

set.seed(8675309)
inorm(runif(10), 10, 2, showPDF = TRUE)

set.seed(8675309)
inorm(runif(10), 10, 2, showECDF = TRUE)

set.seed(8675309)
inorm(runif(10), 10, 2, showPDF = TRUE, showECDF = TRUE, sampleColor = "blue3")

set.seed(8675309)
inorm(runif(10), 10, 2, showPDF = TRUE, showCDF = FALSE)

inorm(runif(100), 10, 2, showPDF = TRUE, minPlotQuantile = 0.02, maxPlotQuantile = 0.98)

# plot the PDF and CDF without any variates
inorm(NULL, 10, 2, showPDF = TRUE, showCDF = TRUE)

# plot CDF with inversion and PDF using show
inorm(runif(10), 10, 2, show = c(1,1,0))
inorm(runif(10), 10, 2, show = 6)

# plot CDF with inversion and ECDF using show, using vunif
inorm(vunif(10), 10, 2, show = c(1,0,1))
inorm(vunif(10), 10, 2, show = 5)
```

```
# plot CDF with inversion, PDF, and ECDF using show
inorm(vunif(10), 10, 2, show = c(1,1,1))
inorm(vunif(10), 10, 2, show = 7)

# plot three different CDF+PDF+ECDF horizontal displays,
# with title only on the first display
oldpar <- par(no.readonly = TRUE)
par(mfrow = c(3,3))  # 3 rows, 3 cols, filling rows before columns
set.seed(8675309)
inorm(runif(20), 10, 2, show = 7, respectLayout = TRUE, restorePar = FALSE)
inorm(runif(20), 10, 2, show = 7, respectLayout = TRUE, restorePar = FALSE, showTitle = FALSE)
inorm(runif(20), 10, 2, show = 7, respectLayout = TRUE, restorePar = TRUE,  showTitle = FALSE)
par(oldpar)

# display animation of all components
inorm(runif(10), 10, 2, show = 7, plotDelay = 0.1)

# display animation of CDF and PDF components only
inorm(runif(10), 10, 2, show = 5, plotDelay = 0.1)

if (interactive()) {
  # interactive -- pause at each stage of inversion
  inorm(runif(10), 10, 2, show = 7, plotDelay = -1)
}

 # overlay visual exploration of ks.test results
 oldpar <- par(no.readonly = TRUE)
 set.seed(54321)
 vals <- inorm(runif(10), 10, 2, showECDF = TRUE, restorePar = FALSE)
 D <- as.numeric(ks.test(vals, "pnorm", 10, 2)$statistic)
 for (x in seq(9.5, 10.5, by = 0.1)) {
   y <- pnorm(x, 10, 2)
   segments(x, y, x, y + D, col = "darkgreen", lwd = 2, xpd = NA)
 }
 par(oldpar) # restore original par values, since restorePar = FALSE above
```

`ipois` *Visualization of Random Variate Generation for the Poisson Distribution*

Description

Generates random variates from the Poisson distribution by inversion. Optionally graphs the population cumulative distribution function and associated random variates, the population probability mass function and a histogram of the random variates, and the empirical cumulative distribution function versus the population cumulative distribution function.

Usage

```
ipois(
  u = runif(1),
  lambda,
  minPlotQuantile = 0,
  maxPlotQuantile = 0.95,
  plot = TRUE,
  showCDF = TRUE,
  showPMF = TRUE,
  showECDF = TRUE,
  show = NULL,
  maxInvPlotted = 50,
  plotDelay = 0,
  sampleColor = "red3",
  populationColor = "grey",
  showTitle = TRUE,
  respectLayout = FALSE,
  restorePar = TRUE,
  ...
)
```

Arguments

`u`	vector of uniform(0,1) random numbers, or NULL to show population figures only
`lambda`	Rate of distribution
`minPlotQuantile`	minimum quantile to plot
`maxPlotQuantile`	maximum quantile to plot
`plot`	logical; if `TRUE` (default), one or more plots will appear (see parameters below); otherwise no plots appear
`showCDF`	logical; if `TRUE` (default), cdf plot appears, otherwise cdf plot is suppressed
`showPMF`	logical; if `TRUE` (default), PMF plot appears, otherwise PMF plot is suppressed
`showECDF`	logical; if `TRUE` (default), ecdf plot appears, otherwise ecdf plot is suppressed
`show`	octal number (0-7) indicating plots to display; 4: CDF, 2: PMF, 1: ECDF; sum for desired combination
`maxInvPlotted`	number of inversions to plot across CDF before switching to plotting quantiles only
`plotDelay`	delay in seconds between CDF plots
`sampleColor`	Color used to display random sample from distribution
`populationColor`	Color used to display population
`showTitle`	logical; if `TRUE` (default), displays a title in the first of any displayed plots

`respectLayout`	logical; if `TRUE` (default), respects existing settings for device layout
`restorePar`	logical; if `TRUE` (default), restores user's previous par settings on function exit
`...`	Possible additional arguments. Currently, additional arguments not considered.

Details

Generates random variates from the Poisson distribution, and optionally, illustrates

- the use of the inverse-CDF technique,
- the effect of random sampling variability in relation to the PMF and CDF.

When all of the graphics are requested,

- the first graph illustrates the use of the inverse-CDF technique by graphing the population CDF and the transformation of the random numbers to random variates,
- the second graph illustrates the effect of random sampling variability by graphing the population PMF and the histogram associated with the random variates, and
- the third graph illustrates effect of random sampling variability by graphing the population CDF and the empirical CDF associated with the random variates.

All aspects of the random variate generation algorithm are output in red by default, which can be changed by specifying `sampleColor`. All aspects of the population distribution are output in gray by default, which can be changed by specifying `populationColor`.

The Poisson distribution has density

$$p(x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

for $x = 0, 1, 2, \ldots$. The mean and variance are $E(X) = Var(X) = \lambda$

The algorithm for generating random variates from the Poisson distribution is synchronized (one random variate for each random number) and monotone in u. This means that the variates generated here might be useful in some variance reduction techniques used in Monte Carlo and discrete-event simulation.

Values from the u vector are plotted in the cdf plot along the vertical axis as colored dots. A horizontal, dashed, colored line extends from the dot to the population cdf. At the intersection, a vertical, dashed colored line extends downward to the horizontal axis, where a second colored dot, denoting the associated Poisson random variate is plotted.

This is not a particularly fast variate generation algorithm because it uses the base R `qpois` function to invert the values contained in `u`.

All of the elements of the `u` vector must be between 0 and 1. Alternatively, `u` can be `NULL` in which case plot(s) of the theoretical PMF and cdf are displayed according to plotting parameter values (defaulting to display of both the PMF and cdf).

The `show` parameter can be used as a shortcut way to denote plots to display. The argument to `show` can be either:

- a binary vector of length three, where the entries from left to right correspond to `showCDF`, `showPMF`, and `showECDF`, respectively. For each entry, a 1 indicates the plot should be displayed, and a 0 indicates the plot should be suppressed.

- an integer in [0,7] interpreted similar to the Unix chmod command. That is, the integer's binary representation can be transformed into a length-three vector discussed above (e.g., 6 corresponds to c(1,1,0)). See examples.

Any valid value for `show` takes precedence over existing individual values for `showCDF`, `showPMF`, and `showECDF`.

If `respectLayout` is `TRUE`, the function respects existing settings for device layout. Note, however, that if the number of plots requested (either via `show` or via `showCDF`, `showPMF`, and `showECDF`) exceeds the number of plots available in the current layout (as determined by `prod(par("mfrow"))`), the function will display all requested plots but will also display a warning message indicating that the current layout does not permit simultaneous viewing of all requested plots. The most recent plot with this attribute can be further annotated after the call.

If `respectLayout` is `FALSE`, any existing user settings for device layout are ignored. That is, the function uses `par` to explicitly set `mfrow` sufficient to show all requested plots stacked vertically to align their horizontal axes, and then resets row, column, and margin settings to their prior state on exit.

The `minPlotQuantile` and `maxPlotQuantile` arguments are present in order to compress the plots horizontally. The random variates generated are not impacted by these two arguments. Vertical, dotted, black lines are plotted at the associated quantiles on the plots.

`plotDelay` can be used to slow down or halt the variate generation for classroom explanation.

In the plot associated with the PMF, the maximum plotting height is associated with 125\ that extends above this limit will have three dots appearing above it.

Value

A vector of Poisson random variates

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

`stats::rpois`

`stats::runif`, `simEd::vunif`

Examples

```
ipois(0.5, lambda = 5)

set.seed(8675309)
ipois(runif(10), 3, showPMF = TRUE)

set.seed(8675309)
ipois(runif(10), 3, showECDF = TRUE)

set.seed(8675309)
```

```
ipois(runif(10), 3, showPMF = TRUE, showECDF = TRUE, sampleColor = "blue3")

set.seed(8675309)
ipois(runif(10), 3, showPMF = TRUE, showCDF = FALSE)

ipois(runif(100), 3, showPMF = TRUE, minPlotQuantile = 0.02, maxPlotQuantile = 0.98)

# plot the PMF and CDF without any variates
ipois(NULL, 3, showPMF = TRUE, showCDF = TRUE)

# plot CDF with inversion and PMF using show
ipois(runif(10), 3, show = c(1,1,0))
ipois(runif(10), 3, show = 6)

# plot CDF with inversion and ECDF using show, using vunif
ipois(vunif(10), 3, show = c(1,0,1))
ipois(vunif(10), 3, show = 5)

# plot CDF with inversion, PMF, and ECDF using show
ipois(vunif(10), 3, show = c(1,1,1))
ipois(vunif(10), 3, show = 7)

# plot three different CDF+PMF+ECDF horizontal displays,
# with title only on the first display
oldpar <- par(no.readonly = TRUE)
par(mfrow = c(3,3))  # 3 rows, 3 cols, filling rows before columns
set.seed(8675309)
ipois(runif(20), 3, show = 7, respectLayout = TRUE, restorePar = FALSE)
ipois(runif(20), 3, show = 7, respectLayout = TRUE, restorePar = FALSE, showTitle = FALSE)
ipois(runif(20), 3, show = 7, respectLayout = TRUE, restorePar = TRUE,  showTitle = FALSE)
par(oldpar)

# display animation of all components
ipois(runif(10), 3, show = 7, plotDelay = 0.1)

# display animation of CDF and PMF components only
ipois(runif(10), 3, show = 5, plotDelay = 0.1)

if (interactive()) {
  # interactive -- pause at each stage of inversion
  ipois(runif(10), 3, show = 7, plotDelay = -1)
}
```

`it` *Visualization of Random Variate Generation for the Student T Distribution*

Description

Generates random variates from the Student T distribution by inversion. Optionally graphs the population cumulative distribution function and associated random variates, the population probability density function and a histogram of the random variates, and the empirical cumulative distribution function versus the population cumulative distribution function.

Usage

```
it(
  u = runif(1),
  df,
  ncp,
  minPlotQuantile = 0.01,
  maxPlotQuantile = 0.99,
  plot = TRUE,
  showCDF = TRUE,
  showPDF = TRUE,
  showECDF = TRUE,
  show = NULL,
  maxInvPlotted = 50,
  plotDelay = 0,
  sampleColor = "red3",
  populationColor = "grey",
  showTitle = TRUE,
  respectLayout = FALSE,
  restorePar = TRUE,
  ...
)
```

Arguments

`u`	vector of uniform(0,1) random numbers, or NULL to show population figures only
`df`	Degrees of freedom > 0
`ncp`	Non-centrality parameter delta (default NULL)
`minPlotQuantile`	minimum quantile to plot
`maxPlotQuantile`	maximum quantile to plot
`plot`	logical; if `TRUE` (default), one or more plots will appear (see parameters below); otherwise no plots appear
`showCDF`	logical; if `TRUE` (default), cdf plot appears, otherwise cdf plot is suppressed
`showPDF`	logical; if `TRUE` (default), PDF plot appears, otherwise PDF plot is suppressed
`showECDF`	logical; if `TRUE` (default), ecdf plot appears, otherwise ecdf plot is suppressed
`show`	octal number (0-7) indicating plots to display; 4: CDF, 2: PDF, 1: ECDF; sum for desired combination
`maxInvPlotted`	number of inversions to plot across CDF before switching to plotting quantiles only
`plotDelay`	delay in seconds between CDF plots

`sampleColor`	Color used to display random sample from distribution
`populationColor`	Color used to display population
`showTitle`	logical; if `TRUE` (default), displays a title in the first of any displayed plots
`respectLayout`	logical; if `TRUE` (default), respects existing settings for device layout
`restorePar`	logical; if `TRUE` (default), restores user's previous par settings on function exit
`...`	Possible additional arguments. Currently, additional arguments not considered.

Details

Generates random variates from the Student T distribution, and optionally, illustrates

- the use of the inverse-CDF technique,
- the effect of random sampling variability in relation to the PDF and CDF.

When all of the graphics are requested,

- the first graph illustrates the use of the inverse-CDF technique by graphing the population CDF and the transformation of the random numbers to random variates,
- the second graph illustrates the effect of random sampling variability by graphing the population PDF and the histogram associated with the random variates, and
- the third graph illustrates effect of random sampling variability by graphing the population CDF and the empirical CDF associated with the random variates.

All aspects of the random variate generation algorithm are output in red by default, which can be changed by specifying `sampleColor`. All aspects of the population distribution are output in gray by default, which can be changed by specifying `populationColor`.

The t-distribution with `df` $= v$ degrees of freedom has density

$$f(x) = \frac{\Gamma((v+1)/2)}{\sqrt{v\pi}\,\Gamma(v/2)}\,(1 + x^2/v)^{-(v+1)/2}$$

for all real x. It has mean 0 (for $v > 1$) and variance $v/(v-2)$ (for $v > 2$).

The general non-central t with parameters (ν, δ) = `(df, ncp)` is defined as the distribution of $T_\nu(\delta) := (U + \delta) \;/\; \sqrt{(V/\nu)}$ where U and V are independent random variables, $U \sim \mathcal{N}(0, 1)$ and $V \sim \chi^2(\nu)$.

The algorithm for generating random variates from the Student t distribution is synchronized (one random variate for each random number) and monotone in u. This means that the variates generated here might be useful in some variance reduction techniques used in Monte Carlo and discrete-event simulation.

Values from the `u` vector are plotted in the cdf plot along the vertical axis as colored dots. A horizontal, dashed, colored line extends from the dot to the population cdf. At the intersection, a vertical, dashed colored line extends downward to the horizontal axis, where a second colored dot, denoting the associated Student t random variate is plotted.

This is not a particularly fast variate generation algorithm because it uses the base R `qt` function to invert the values contained in `u`.

All of the elements of the u vector must be between 0 and 1. Alternatively, u can be NULL in which case plot(s) of the theoretical PDF and cdf are displayed according to plotting parameter values (defaulting to display of both the PDF and cdf).

The show parameter can be used as a shortcut way to denote plots to display. The argument to show can be either:

- a binary vector of length three, where the entries from left to right correspond to showCDF, showPDF, and showECDF, respectively. For each entry, a 1 indicates the plot should be displayed, and a 0 indicates the plot should be suppressed.
- an integer in [0,7] interpreted similar to the Unix chmod command. That is, the integer's binary representation can be transformed into a length-three vector discussed above (e.g., 6 corresponds to c(1,1,0)). See examples.

Any valid value for show takes precedence over existing individual values for showCDF, showPDF, and showECDF.

If respectLayout is TRUE, the function respects existing settings for device layout. Note, however, that if the number of plots requested (either via show or via showCDF, showPMF, and showECDF) exceeds the number of plots available in the current layout (as determined by prod(par("mfrow"))), the function will display all requested plots but will also display a warning message indicating that the current layout does not permit simultaneous viewing of all requested plots. The most recent plot with this attribute can be further annotated after the call.

If respectLayout is FALSE, any existing user settings for device layout are ignored. That is, the function uses par to explicitly set mfrow sufficient to show all requested plots stacked vertically to align their horizontal axes, and then resets row, column, and margin settings to their prior state on exit.

The minPlotQuantile and maxPlotQuantile arguments are present in order to compress the plots horizontally. The random variates generated are not impacted by these two arguments. Vertical, dotted, black lines are plotted at the associated quantiles on the plots.

plotDelay can be used to slow down or halt the variate generation for classroom explanation.

In the plot associated with the PDF, the maximum plotting height is associated with 125\ that extends above this limit will have three dots appearing above it.

Value

A vector of Student T random variates

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

stats::rt

stats::runif, simEd::vunif

Examples

```
it(0.5, df = 5, ncp = 10)

set.seed(8675309)
it(runif(10), 4, showPDF = TRUE)

set.seed(8675309)
it(runif(10), 4, showECDF = TRUE)

set.seed(8675309)
it(runif(10), 4, showPDF = TRUE, showECDF = TRUE, sampleColor = "blue3")

set.seed(8675309)
it(runif(10), 4, showPDF = TRUE, showCDF = FALSE)

it(runif(100), 4, showPDF = TRUE, minPlotQuantile = 0.02, maxPlotQuantile = 0.98)

# plot the PDF and CDF without any variates
it(NULL, 4, showPDF = TRUE, showCDF = TRUE)

# plot CDF with inversion and PDF using show
it(runif(10), 4, show = c(1,1,0))
it(runif(10), 4, show = 6)

# plot CDF with inversion and ECDF using show, using vunif
it(vunif(10), 4, show = c(1,0,1))
it(vunif(10), 4, show = 5)

# plot CDF with inversion, PDF, and ECDF using show
it(vunif(10), 4, show = c(1,1,1))
it(vunif(10), 4, show = 7)

# plot three different CDF+PDF+ECDF horizontal displays,
# with title only on the first display
oldpar <- par(no.readonly = TRUE)
par(mfrow = c(3,3))  # 3 rows, 3 cols, filling rows before columns
set.seed(8675309)
it(runif(20), 4, show = 7, respectLayout = TRUE, restorePar = FALSE)
it(runif(20), 4, show = 7, respectLayout = TRUE, restorePar = FALSE, showTitle = FALSE)
it(runif(20), 4, show = 7, respectLayout = TRUE, restorePar = TRUE,  showTitle = FALSE)
par(oldpar)

# display animation of all components
it(runif(10), 4, show = 7, plotDelay = 0.1)

# display animation of CDF and PDF components only
it(runif(10), 4, show = 5, plotDelay = 0.1)

if (interactive()) {
  # interactive -- pause at each stage of inversion
  it(runif(10), 4, show = 7, plotDelay = -1)
}
```

---

iunif — *Visualization of Random Variate Generation for the Uniform Distribution*

Description

Generates random variates from the Uniform distribution by inversion. Optionally graphs the population cumulative distribution function and associated random variates, the population probability density function and a histogram of the random variates, and the empirical cumulative distribution function versus the population cumulative distribution function.

Usage

```
iunif(
  u = runif(1),
  min = 0,
  max = 1,
  minPlotQuantile = 0,
  maxPlotQuantile = 1,
  plot = TRUE,
  showCDF = TRUE,
  showPDF = TRUE,
  showECDF = TRUE,
  show = NULL,
  maxInvPlotted = 50,
  plotDelay = 0,
  sampleColor = "red3",
  populationColor = "grey",
  showTitle = TRUE,
  respectLayout = FALSE,
  restorePar = TRUE,
  ...
)
```

Arguments

`u`	vector of uniform(0,1) random numbers, or NULL to show population figures only
`min`	lower limit of distribution (default 0)
`max`	upper limit of distribution (default 1)
`minPlotQuantile`	minimum quantile to plot
`maxPlotQuantile`	maximum quantile to plot

`plot`	logical; if `TRUE` (default), one or more plots will appear (see parameters below); otherwise no plots appear
`showCDF`	logical; if `TRUE` (default), cdf plot appears, otherwise cdf plot is suppressed
`showPDF`	logical; if `TRUE` (default), PDF plot appears, otherwise PDF plot is suppressed
`showECDF`	logical; if `TRUE` (default), ecdf plot appears, otherwise ecdf plot is suppressed
`show`	octal number (0-7) indicating plots to display; 4: CDF, 2: PDF, 1: ECDF; sum for desired combination
`maxInvPlotted`	number of inversions to plot across CDF before switching to plotting quantiles only
`plotDelay`	delay in seconds between CDF plots
`sampleColor`	Color used to display random sample from distribution
`populationColor`	Color used to display population
`showTitle`	logical; if `TRUE` (default), displays a title in the first of any displayed plots
`respectLayout`	logical; if `TRUE` (default), respects existing settings for device layout
`restorePar`	logical; if `TRUE` (default), restores user's previous par settings on function exit
`...`	Possible additional arguments. Currently, additional arguments not considered.

Details

Generates random variates from the Uniform distribution, and optionally, illustrates

- the use of the inverse-CDF technique,
- the effect of random sampling variability in relation to the PDF and CDF.

When all of the graphics are requested,

- the first graph illustrates the use of the inverse-CDF technique by graphing the population CDF and the transformation of the random numbers to random variates,
- the second graph illustrates the effect of random sampling variability by graphing the population PDF and the histogram associated with the random variates, and
- the third graph illustrates effect of random sampling variability by graphing the population CDF and the empirical CDF associated with the random variates.

All aspects of the random variate generation algorithm are output in red by default, which can be changed by specifying `sampleColor`. All aspects of the population distribution are output in gray by default, which can be changed by specifying `populationColor`.

The uniform distribution has density

```
\deqn{f(x) = \frac{1}{max-min}}{
          f(x) = 1/(max-min)}
```

for $min \leq x \leq max$.

The algorithm for generating random variates from the uniform distribution is synchronized (one random variate for each random number) and monotone in u. This means that the variates generated here might be useful in some variance reduction techniques used in Monte Carlo and discrete-event simulation.

Values from the u vector are plotted in the cdf plot along the vertical axis as colored dots. A horizontal, dashed, colored line extends from the dot to the population cdf. At the intersection, a vertical, dashed colored line extends downward to the horizontal axis, where a second colored dot, denoting the associated uniform random variate is plotted.

This is not a particularly fast variate generation algorithm because it uses the base R `qunif` function to invert the values contained in `u`.

All of the elements of the `u` vector must be between 0 and 1. Alternatively, `u` can be `NULL` in which case plot(s) of the theoretical PDF and cdf are displayed according to plotting parameter values (defaulting to display of both the PDF and cdf).

The `show` parameter can be used as a shortcut way to denote plots to display. The argument to `show` can be either:

- a binary vector of length three, where the entries from left to right correspond to `showCDF`, `showPDF`, and `showECDF`, respectively. For each entry, a 1 indicates the plot should be displayed, and a 0 indicates the plot should be suppressed.
- an integer in [0,7] interpreted similar to the Unix chmod command. That is, the integer's binary representation can be transformed into a length-three vector discussed above (e.g., 6 corresponds to c(1,1,0)). See examples.

Any valid value for `show` takes precedence over existing individual values for `showCDF`, `showPDF`, and `showECDF`.

If `respectLayout` is `TRUE`, the function respects existing settings for device layout. Note, however, that if the number of plots requested (either via `show` or via `showCDF`, `showPMF`, and `showECDF`) exceeds the number of plots available in the current layout (as determined by `prod(par("mfrow"))`), the function will display all requested plots but will also display a warning message indicating that the current layout does not permit simultaneous viewing of all requested plots. The most recent plot with this attribute can be further annotated after the call.

If `respectLayout` is `FALSE`, any existing user settings for device layout are ignored. That is, the function uses `par` to explicitly set `mfrow` sufficient to show all requested plots stacked vertically to align their horizontal axes, and then resets row, column, and margin settings to their prior state on exit.

The `minPlotQuantile` and `maxPlotQuantile` arguments are present in order to compress the plots horizontally. The random variates generated are not impacted by these two arguments. Vertical, dotted, black lines are plotted at the associated quantiles on the plots.

`plotDelay` can be used to slow down or halt the variate generation for classroom explanation.

In the plot associated with the PDF, the maximum plotting height is associated with 125\ that extends above this limit will have three dots appearing above it.

Value

A vector of Uniform random variates

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

stats::runif

stats::runif, simEd::vunif

Examples

```
iunif(0.5, min = -10, max = 10)

set.seed(8675309)
iunif(runif(10), 0, 10, showPDF = TRUE)

set.seed(8675309)
iunif(runif(10), 0, 10, showECDF = TRUE)

set.seed(8675309)
iunif(runif(10), 0, 10, showPDF = TRUE, showECDF = TRUE, sampleColor = "blue3")

set.seed(8675309)
iunif(runif(10), 0, 10, showPDF = TRUE, showCDF = FALSE)

iunif(runif(100), 0, 10, showPDF = TRUE, minPlotQuantile = 0.02, maxPlotQuantile = 0.98)

# plot the PDF and CDF without any variates
iunif(NULL, 0, 10, showPDF = TRUE, showCDF = TRUE)

# plot CDF with inversion and PDF using show
iunif(runif(10), 0, 10, show = c(1,1,0))
iunif(runif(10), 0, 10, show = 6)

# plot CDF with inversion and ECDF using show, using vunif
iunif(vunif(10), 0, 10, show = c(1,0,1))
iunif(vunif(10), 0, 10, show = 5)

# plot CDF with inversion, PDF, and ECDF using show
iunif(vunif(10), 0, 10, show = c(1,1,1))
iunif(vunif(10), 0, 10, show = 7)

# plot three different CDF+PDF+ECDF horizontal displays,
# with title only on the first display
oldpar <- par(no.readonly = TRUE)
par(mfrow = c(3,3))  # 3 rows, 3 cols, filling rows before columns
set.seed(8675309)
iunif(runif(20), 0, 10, show = 7, respectLayout = TRUE, restorePar = FALSE)
iunif(runif(20), 0, 10, show = 7, respectLayout = TRUE, restorePar = FALSE, showTitle = FALSE)
iunif(runif(20), 0, 10, show = 7, respectLayout = TRUE, restorePar = TRUE,  showTitle = FALSE)
```

```
par(oldpar)

# display animation of all components
iunif(runif(10), 0, 10, show = 7, plotDelay = 0.1)

# display animation of CDF and PDF components only
iunif(runif(10), 0, 10, show = 5, plotDelay = 0.1)

if (interactive()) {
  # interactive -- pause at each stage of inversion
  iunif(runif(10), 0, 10, show = 7, plotDelay = -1)
}

 # overlay visual exploration of ks.test results
 oldpar <- par(no.readonly = TRUE)
 set.seed(54321)
 vals <- iunif(runif(10), 0, 10, showECDF = TRUE, restorePar = FALSE)
 D <- as.numeric(ks.test(vals, "punif", 0, 10)$statistic)
 for (x in seq(4.0, 6.0, by = 0.1)) {
   y <- punif(x, 0, 10)
   segments(x, y, x, y + D, col = "darkgreen", lwd = 2, xpd = NA)
 }
 par(oldpar) # restore original par values, since restorePar = FALSE above
```

`iweibull` *Visualization of Random Variate Generation for the Weibull Distribution*

Description

Generates random variates from the Weibull distribution by inversion. Optionally graphs the population cumulative distribution function and associated random variates, the population probability density function and a histogram of the random variates, and the empirical cumulative distribution function versus the population cumulative distribution function.

Usage

```
iweibull(
  u = runif(1),
  shape,
  scale = 1,
  minPlotQuantile = 0.01,
  maxPlotQuantile = 0.99,
  plot = TRUE,
  showCDF = TRUE,
  showPDF = TRUE,
  showECDF = TRUE,
  show = NULL,
```

```
  maxInvPlotted = 50,
  plotDelay = 0,
  sampleColor = "red3",
  populationColor = "grey",
  showTitle = TRUE,
  respectLayout = FALSE,
  restorePar = TRUE,
  ...
)
```

Arguments

u	vector of uniform(0,1) random numbers, or NULL to show population figures only
shape	Shape parameter
scale	Scale parameter (default 1)
minPlotQuantile	minimum quantile to plot
maxPlotQuantile	maximum quantile to plot
plot	logical; if TRUE (default), one or more plots will appear (see parameters below); otherwise no plots appear
showCDF	logical; if TRUE (default), cdf plot appears, otherwise cdf plot is suppressed
showPDF	logical; if TRUE (default), PDF plot appears, otherwise PDF plot is suppressed
showECDF	logical; if TRUE (default), ecdf plot appears, otherwise ecdf plot is suppressed
show	octal number (0-7) indicating plots to display; 4: CDF, 2: PDF, 1: ECDF; sum for desired combination
maxInvPlotted	number of inversions to plot across CDF before switching to plotting quantiles only
plotDelay	delay in seconds between CDF plots
sampleColor	Color used to display random sample from distribution
populationColor	Color used to display population
showTitle	logical; if TRUE (default), displays a title in the first of any displayed plots
respectLayout	logical; if TRUE (default), respects existing settings for device layout
restorePar	logical; if TRUE (default), restores user's previous par settings on function exit
...	Possible additional arguments. Currently, additional arguments not considered.

Details

Generates random variates from the Weibull distribution, and optionally, illustrates

- the use of the inverse-CDF technique,
- the effect of random sampling variability in relation to the PDF and CDF.

When all of the graphics are requested,

- the first graph illustrates the use of the inverse-CDF technique by graphing the population CDF and the transformation of the random numbers to random variates,
- the second graph illustrates the effect of random sampling variability by graphing the population PDF and the histogram associated with the random variates, and
- the third graph illustrates effect of random sampling variability by graphing the population CDF and the empirical CDF associated with the random variates.

All aspects of the random variate generation algorithm are output in red by default, which can be changed by specifying `sampleColor`. All aspects of the population distribution are output in gray by default, which can be changed by specifying `populationColor`.

The Weibull distribution with parameters `shape` = a and `scale` = b has density

```
\deqn{f(x) = \frac{a}{b} \left(\frac{x}{b}\right)^{a-1} e^{-(x/b)^a}}{
         f(x) = (a/b) (x/b)^(a-1) exp(-(x/b)^a)}
```

for $x \geq 0$, $a > 0$, and $b > 0$.

The algorithm for generating random variates from the Weibull distribution is synchronized (one random variate for each random number) and monotone in u. This means that the variates generated here might be useful in some variance reduction techniques used in Monte Carlo and discrete-event simulation.

Values from the u vector are plotted in the cdf plot along the vertical axis as colored dots. A horizontal, dashed, colored line extends from the dot to the population cdf. At the intersection, a vertical, dashed colored line extends downward to the horizontal axis, where a second colored dot, denoting the associated Weibull random variate is plotted.

This is not a particularly fast variate generation algorithm because it uses the base R `qweibull` function to invert the values contained in `u`.

All of the elements of the `u` vector must be between 0 and 1. Alternatively, `u` can be `NULL` in which case plot(s) of the theoretical PDF and cdf are displayed according to plotting parameter values (defaulting to display of both the PDF and cdf).

The `show` parameter can be used as a shortcut way to denote plots to display. The argument to `show` can be either:

- a binary vector of length three, where the entries from left to right correspond to `showCDF`, `showPDF`, and `showECDF`, respectively. For each entry, a 1 indicates the plot should be displayed, and a 0 indicates the plot should be suppressed.
- an integer in [0,7] interpreted similar to the Unix chmod command. That is, the integer's binary representation can be transformed into a length-three vector discussed above (e.g., 6 corresponds to c(1,1,0)). See examples.

Any valid value for `show` takes precedence over existing individual values for `showCDF`, `showPDF`, and `showECDF`.

If `respectLayout` is `TRUE`, the function respects existing settings for device layout. Note, however, that if the number of plots requested (either via `show` or via `showCDF`, `showPMF`, and `showECDF`) exceeds the number of plots available in the current layout (as determined by `prod(par("mfrow"))`), the function will display all requested plots but will also display a warning message indicating that the current layout does not permit simultaneous viewing of all requested plots. The most recent plot with this attribute can be further annotated after the call.

If respectLayout is FALSE, any existing user settings for device layout are ignored. That is, the function uses par to explicitly set mfrow sufficient to show all requested plots stacked vertically to align their horizontal axes, and then resets row, column, and margin settings to their prior state on exit.

The minPlotQuantile and maxPlotQuantile arguments are present in order to compress the plots horizontally. The random variates generated are not impacted by these two arguments. Vertical, dotted, black lines are plotted at the associated quantiles on the plots.

plotDelay can be used to slow down or halt the variate generation for classroom explanation.

In the plot associated with the PDF, the maximum plotting height is associated with 125\ that extends above this limit will have three dots appearing above it.

Value

A vector of Weibull random variates

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

stats::rweibull

stats::runif, simEd::vunif

Examples

```
iweibull(0.5, shape = 2, scale = 0.5)

set.seed(8675309)
iweibull(runif(10), 1, 2, showPDF = TRUE)

set.seed(8675309)
iweibull(runif(10), 1, 2, showECDF = TRUE)

set.seed(8675309)
iweibull(runif(10), 1, 2, showPDF = TRUE, showECDF = TRUE, sampleColor = "blue3")

set.seed(8675309)
iweibull(runif(10), 1, 2, showPDF = TRUE, showCDF = FALSE)

iweibull(runif(100), 1, 2, showPDF = TRUE, minPlotQuantile = 0.02, maxPlotQuantile = 0.98)

# plot the PDF and CDF without any variates
iweibull(NULL, 1, 2, showPDF = TRUE, showCDF = TRUE)
```

```
# plot CDF with inversion and PDF using show
iweibull(runif(10), 1, 2, show = c(1,1,0))
iweibull(runif(10), 1, 2, show = 6)

# plot CDF with inversion and ECDF using show, using vunif
iweibull(vunif(10), 1, 2, show = c(1,0,1))
iweibull(vunif(10), 1, 2, show = 5)

# plot CDF with inversion, PDF, and ECDF using show
iweibull(vunif(10), 1, 2, show = c(1,1,1))
iweibull(vunif(10), 1, 2, show = 7)

# plot three different CDF+PDF+ECDF horizontal displays,
# with title only on the first display
oldpar <- par(no.readonly = TRUE)
par(mfrow = c(3,3))  # 3 rows, 3 cols, filling rows before columns
set.seed(8675309)
iweibull(runif(20), 1, 2, show = 7, respectLayout = TRUE, restorePar = FALSE)
iweibull(runif(20), 1, 2, show = 7, respectLayout = TRUE, restorePar = FALSE, showTitle = FALSE)
iweibull(runif(20), 1, 2, show = 7, respectLayout = TRUE, restorePar = TRUE,  showTitle = FALSE)
par(oldpar)

# display animation of all components
iweibull(runif(10), 1, 2, show = 7, plotDelay = 0.1)

# display animation of CDF and PDF components only
iweibull(runif(10), 1, 2, show = 5, plotDelay = 0.1)

if (interactive()) {
  # interactive -- pause at each stage of inversion
  iweibull(runif(10), 1, 2, show = 7, plotDelay = -1)
}
```

`lehmer` *Lehmer Generator Visualization*

Description

This function animates the processes of a basic Lehmer pseudo-random number generator (PRNG). Also known in the literature as a multiplicative linear congruential generator (MLCG), the generator is based on the formula:

$$X_{k+1} \equiv a \cdot X_k \pmod{m}$$

where 'm' is the prime modulus, 'a' is the multiplier chosen from {1, m-1}, and 'X_0' is the initial seed chosen from {1, m-1}. The random numbers generated in (0,1) are X_{k+1}/m.

Usage

```
lehmer(
  a = 13,
  m = 31,
  seed = 1,
  animate = TRUE,
  numSteps = NA,
  title = NA,
  showTitle = TRUE,
  plotDelay = -1
)
```

Arguments

a	multiplier in MLCG equation.
m	prime modulus in MLCG equation.
seed	initial seed for the generator, i.e., the initial value X_0
animate	should the visual output be displayed.
numSteps	number of steps to animate; default value is Inf if plotDelay is -1, or the size of the period otherwise. Ignored if animate is false.
title	optional title to display in plot (NA uses default title)
showTitle	if TRUE, display title in the main plot.
plotDelay	wait time between transitioning; -1 (default) for interactive mode, where the user is queried for input to progress.

Value

the entire period from the PRNG cycle, as a vector of integers in {1, m-1}.

References

Lehmer, D.H. (1951). Mathematical Models in Large-Scale Computing Units. *Ann. Comput. Lab.* Harvard University, **26**, 141-146.

Examples

```
# Default case (m, a = 31, 13); small full period
lehmer(plotDelay = 0, numSteps = 16)
lehmer(numSteps = 10, plotDelay = 0.1)   # auto-advance mode

if (interactive()) {
  lehmer(plotDelay = -1)  # plotDelay -1 uses interactive mode
}

# multiplier producing period of length 5, with different seeds
lehmer(a = 8, m = 31, seed = 1, numSteps = 5, plotDelay = 0.1)
lehmer(a = 8, m = 31, seed = 24, numSteps = 5, plotDelay = 0.1)
```

```
# degenerate cases where seed does not appear in the final period
lehmer(a = 12, m = 20, seed = 7, numSteps = 4, plotDelay = 0.1)  # length  4
lehmer(a = 4, m = 6, seed = 1, numSteps = 1, plotDelay = 0.1)  # length 1
```

---

meanTPS *Mean of Time-Persistent Statistics (TPS)*

Description

Computes the sample mean of a time-persistent function.

Usage

```
meanTPS(times = NULL, numbers = NULL)
```

Arguments

`times`	A numeric vector of non-decreasing time observations
`numbers`	A numeric vector containing the values of the time-persistent statistic between the time observation

Details

The lengths of `times` and `numbers` either must be the same, or `times` may have one more entry than `numbers` (interval endpoints vs. interval counts). The sample mean is the area under the step-function created by the values in `numbers` between the first and last element in `times` divided by the length of the observation period.

Value

the sample mean of the time-persistent function provided

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

Examples

```
times  <- c(1,2,3,4,5)
counts <- c(1,2,1,1,2)
meanTPS(times, counts)

output <- ssq(seed = 54321, maxTime = 100, saveServerStatus = TRUE)
utilization <- meanTPS(output$serverStatusT, output$serverStatusN)

# compute and graphically display mean of number in system vs time
```

```
output <- ssq(maxArrivals = 60, seed = 54321, saveAllStats = TRUE)
plot(output$numInSystemT, output$numInSystemN, type = "s", bty = "l",
    las = 1, xlab = "time", ylab = "number in system")
timeAvgNumInSysMean <- meanTPS(output$numInSystemT, output$numInSystemN)
abline(h = timeAvgNumInSysMean, lty = "solid", col = "red", lwd = 2)
```

---

msq *Multi-Server Queue Simulation*

Description

A next-event simulation of a single-queue multiple-server service node, with extensible arrival and service processes.

Usage

```
msq(
  maxArrivals = Inf,
  seed = NA,
  numServers = 2,
  serverSelection = c("LRU", "LFU", "CYC", "RAN", "ORD"),
  interarrivalFcn = NULL,
  serviceFcn = NULL,
  interarrivalType = "M",
  serviceType = "M",
  maxTime = Inf,
  maxDepartures = Inf,
  maxInSystem = Inf,
  maxEventsPerSkyline = 15,
  saveAllStats = FALSE,
  saveInterarrivalTimes = FALSE,
  saveServiceTimes = FALSE,
  saveWaitTimes = FALSE,
  saveSojournTimes = FALSE,
  saveNumInQueue = FALSE,
  saveNumInSystem = FALSE,
  saveServerStatus = FALSE,
  showOutput = TRUE,
  animate = FALSE,
  showQueue = NULL,
  showSkyline = NULL,
  showSkylineSystem = FALSE,
  showSkylineQueue = FALSE,
  showSkylineServer = FALSE,
  showTitle = TRUE,
  showProgress = TRUE,
```

```
    plotQueueFcn = defaultPlotMSQ,
    plotSkylineFcn = defaultPlotSkyline,
    jobImage = NA,
    plotDelay = NA,
    respectLayout = FALSE
  )
```

Arguments

`maxArrivals` maximum number of customer arrivals allowed to enter the system

`seed` initial seed to the random number generator (NA uses current state of random number generator; NULL seeds using system clock)

`numServers` Number of servers to simulation (an integer between 1 and 24)

`serverSelection` Algorithm to use for selecting among idle servers (default is "LRU")

`interarrivalFcn` Function for generating interarrival times for queue simulation. Default value (`NA`) will result in use of default interarrival function based on `interarrivalType`. See examples.

`serviceFcn` Function for generating service times for queue simulation. Default value (`NA`) will result in use of default service function based on `serviceType`. See examples.

`interarrivalType` string representation of desired interarrival process. Options are "M" – exponential with rate 1; "G" – uniform(0,2), having mean 1; and "D" – deterministic with constant value 1. Default is "M".

`serviceType` string representation of desired service process . Options are "M" – exponential with rate 10/9; "G" – uniform(0, 1.8), having mean 9/10; and "D" – deterministic with constant value 9/10. Default is "M".

`maxTime` maximum time to simulate

`maxDepartures` maximum number of customer departures to process

`maxInSystem` maximum number of customers that the system can hold (server(s) plus queue). Infinite by default.

`maxEventsPerSkyline` maximum number of events viewable at a time in the skyline plot. A large value for this parameter may result in plotting delays. This parameter does not impact the final plotting, which will show all end-of-simulation results.

`saveAllStats` if `TRUE`, returns all vectors of statistics (see below) collected by the simulation

`saveInterarrivalTimes` if `TRUE`, returns a vector of all interarrival times generated

`saveServiceTimes` if `TRUE`, returns a vector of all service times generated

`saveWaitTimes` if `TRUE`, returns a vector of all wait times (in the queue) generated

`saveSojournTimes` if `TRUE`, returns a vector of all sojourn times (time spent in the system) generated

saveNumInQueue
: if `TRUE`, returns a vector of times and a vector of counts for whenever the number in the queue changes

saveNumInSystem
: if `TRUE`, returns a vector of times and a vector of counts for whenever the number in the system changes

saveServerStatus
: if `TRUE`, returns a vector of times and a vector of server status (0:idle, 1:busy) for whenever the status changes

showOutput
: if `TRUE`, displays summary statistics upon completion

animate
: If FALSE, no animation will be shown.

showQueue
: if TRUE, displays a visualization of the queue

showSkyline
: If `NULL` (default), defers to each individual showSkyline... parameter below; otherwise, supersedes individual showSkyline... parameter values. If `TRUE`, displays full skyline plot; `FALSE` suppresses skyline plot. Can alternatively be specified using chmod-like octal component specification: use 1, 2, 4 for system, queue, and server respectively, summing to indicate desired combination (e.g., 7 for all). Can also be specified as a binary vector (e.g., c(1,1,1) for all).

showSkylineSystem
: logical; if `TRUE`, includes number in system as part of skyline plot. Value for `showSkyline` supersedes this parameter's value.

showSkylineQueue
: logical; if `TRUE`, includes number in queue as part of skyline plot. Value for `showSkyline` supersedes this parameter's value.

showSkylineServer
: logical; if `TRUE`, includes number in server as part of skyline plot. Value for `showSkyline` supersedes this parameter's value.

showTitle
: if `TRUE`, display title at the top of the main plot

showProgress
: if TRUE, displays a progress bar on screen during no-animation execution

plotQueueFcn
: Plotting function to display Queue visualization. By default, this is provided by `defaultPlotSSQ`. Please refer to the corresponding help for more details about required arguments.

plotSkylineFcn
: Plotting function to display Skyline visualization. By default, this is provided by `defaultPlotSkyline`. Please refer to the corresponding help for more details about required arguments.

jobImage
: a vector of URLs/local addresses of images to use as jobs. Requires package `'magick'`.

plotDelay
: a positive numeric value indicating seconds between plots. A value of -1 enters 'interactive' mode, where the state will pause for user input at each step. A value of 0 will display only the final end-of-simulation plot.

respectLayout
: If TRUE, plot layout (i.e., par, device, etc.) settings will be respected. Not recommended except for specialized use.

Details

Implements a next-event implementation of a single-queue multiple-server queue simulation.

The `seed` parameter can take one of three valid argument types:

- `NA` (default), which will use the current state of the random number generator without explicitly setting a new seed (see examples);
- a positive integer, which will be used as the initial seed passed in an explicit call to `set.seed`; or
- `NULL`, which will be passed in an explicit call to to `set.seed`, thereby setting the initial seed using the system clock.

The server selection mechanism can be chosen from among five options, with `"LRU"` being the default:

- `"LRU"` (least recently used): from among the currently available (idle) servers, selects the server who has been idle longest.
- `"LFU"` (least frequently used): from among the currently available servers, selects the server having the lowest computed utilization.
- `"CYC"` (cyclic): selects a server in a cyclic manner; i.e, indexing the servers 1, 2, . . ., `numServers` and incrementing cyclically, starts from one greater than the index of the most recently engaged server and selects the first idle server encountered.
- `"RAN"` (random): selects a server at random from among the currently available servers.
- `"ORD"` (in order): indexing the servers 1, 2, . . ., `numServers`, selects the idle server having the lowest index.

Value

The function returns a list containing:

- the number of arrivals to the system (`customerArrivals`),
- the number of customers processed (`customerDepartures`),
- the ending time of the simulation (`simulationEndTime`),
- average wait time in the queue (`avgWait`),
- average time in the system (`avgSojourn`),
- average number in the system (`avgNumInSystem`),
- average number in the queue (`avgNumInQueue`), and
- server utilization (`utilization`).

of the queue as computed by the simulation. When requested via the "save..." parameters, the list may also contain:

- a vector of interarrival times (`interarrivalTimes`),
- a vector of wait times (`waitTimes`),
- a vector of service times (`serviceTimes`),
- a vector of sojourn times (`sojournTimes`),

- two vectors (time and count) noting changes to number in the system (`numInSystemT`, `numInSystemN`),
- two vectors (time and count) noting changes to number in the queue (`numInQueueT`, `numInQueueN`), and
- two vectors (time and status) noting changes to server status (`serverStatusT`, `serverStatusN`).

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

rstream, set.seed, stats::runif

Examples

```
# process 100 arrivals, R-provided seed (via NULL seed), default 2 servers
msq(100, NULL)
# process 100 arrivals, seed 8675309, 3 servers, LFU server selection
msq(100, 8675309, 3, 'LFU')

msq(maxArrivals = 100, seed = 8675309)
msq(maxTime = 100, seed = 8675309)

############################################################################
# example to show use of seed = NA (default) to rely on current state of generator
output1 <- msq(200, 8675309, showOutput = FALSE, saveAllStats = TRUE)
output2 <- msq(300,          showOutput = FALSE, saveAllStats = TRUE)
set.seed(8675309)
output3 <- msq(200,          showOutput = FALSE, saveAllStats = TRUE)
output4 <- msq(300,          showOutput = FALSE, saveAllStats = TRUE)
sum(output1$sojournTimes != output3$sojournTimes) # should be zero
sum(output2$sojournTimes != output4$sojournTimes) # should be zero

############################################################################
# use same service function for (default) two servers
myArrFcn <- function() { vexp(1, rate = 1/4, stream = 1) }               # mean is 4
mySvcFcn <- function() { vgamma(1, shape = 1, rate = 0.3, stream = 2) }  # mean is 3.3
output <- msq(maxArrivals = 100, interarrivalFcn = myArrFcn,
    serviceFcn = mySvcFcn, saveAllStats = TRUE)
mean(output$interarrivalTimes)
mean(output$serviceTimes)

############################################################################
# use different service function for (default) two servers
myArrFcn  <- function() { vexp(1, rate = 1/4, stream = 1) }                # mean is 4
mySvcFcn1 <- function() { vgamma(1, shape = 3, scale = 1.1, stream = 2) }  # mean is 3.3
mySvcFcn2 <- function() { vgamma(1, shape = 3, scale = 1.2, stream = 3) }  # mean is 3.6
output <- msq(maxArrivals = 100, interarrivalFcn = myArrFcn,
    serviceFcn = list(mySvcFcn1, mySvcFcn2), saveAllStats = TRUE)
```

```
mean(output$interarrivalTimes)
meanTPS(output$numInQueueT, output$numInQueueN)  # compute time-averaged num in queue
mean(output$serviceTimesPerServer[[1]])  # compute avg service time for server 1
mean(output$serviceTimesPerServer[[2]])  # compute avg service time for server 2
meanTPS(output$serverStatusT[[1]], output$serverStatusN[[1]])  # compute server 1 utilization
meanTPS(output$serverStatusT[[2]], output$serverStatusN[[2]])  # compute server 2 utilization

############################################################################
# example to show use of (simple) trace data for arrivals and service times,
# allowing for reuse of trace data times
smallQueueTrace <- list()
smallQueueTrace$arrivalTimes <- c(15, 47, 71, 111, 123, 152, 166, 226, 310, 320)
smallQueueTrace$serviceTimes <- c(43, 36, 34,  30,  38,  40,  31,  29,  36,  30)

interarrivalTimes <- NULL
serviceTimes      <- NULL

getInterarr <- function()
{
    if (length(interarrivalTimes) == 0) {
          interarrivalTimes <<- c(smallQueueTrace$arrivalTimes[1],
                                 diff(smallQueueTrace$arrivalTimes))
    }
    nextInterarr <- interarrivalTimes[1]
    interarrivalTimes <<- interarrivalTimes[-1]  # remove 1st element globally
    return(nextInterarr)
}

getService <- function()
{
    if (length(serviceTimes) == 0) {
        serviceTimes <<- smallQueueTrace$serviceTimes
    }
    nextService <- serviceTimes[1]
    serviceTimes <<- serviceTimes[-1]  # remove 1st element globally
    return(nextService)
}

output <- msq(maxArrivals = 100, numServers = 2, interarrivalFcn = getInterarr,
              serviceFcn = getService, saveAllStats = TRUE)
mean(output$interarrivalTimes)
mean(output$serviceTimes)
mean(output$serviceTimesPerServer[[1]])  # compute avg service time for server 1
mean(output$serviceTimesPerServer[[2]])  # compute avg service time for server 2

############################################################################
# Testing with visualization

# Visualizing msq with a set seed, infinite queue capacity, 10 arrivals,
# and showing queue (default) and skyline for all 3 attributes
msq(seed = 1234, numServers = 5, maxArrivals = 10, showSkyline = 7,
    plotDelay = 0.1)
```

```
# Same simulation as above but using default interactive mode
if (interactive()) {
  msq(seed = 1234, numServers = 5, maxArrivals = 10, showSkyline = 7)
}

# Visualizing msq with a set seed, finite queue capacity, 20 arrivals,
# and showing queue (default) and skyline for all 3 attributes
msq(seed = 1234, numServers = 5, maxArrivals = 25, showSkyline = 7,
    maxInSystem = 5, plotDelay = 0)

# Using default distributions to simulate an M/G/2 queue
msq(seed = 1234, maxDepartures = 10,
    interarrivalType = "M", serviceType = "G", plotDelay = 0)
```

`quantileTPS` *Sample Quantiles of Time-Persistent Statistics (TPS)*

Description

Computes the sample quantiles of a time-persistent function, corresponding to the given probabilities.

Usage

```
quantileTPS(times = NULL, numbers = NULL, probs = c(0, 0.25, 0.5, 0.75, 1))
```

Arguments

`times`	A numeric vector of non-decreasing time observations
`numbers`	A numeric vector containing the values of the time-persistent statistic between the time observation
`probs`	A numeric vector of probabilities with values in [0,1]

Details

```
   The lengths of \code{times} and \code{numbers} either must be
the same, or \code{times} may have one more entry than \code{numbers}
(interval endpoints vs. interval counts).  The sample quantiles are calculated
by determining the length of time spent in each state, sorting these times,
then calculating the quantiles associated with the values in the \code{prob}
vector in the same fashion as one would calculate quantiles associated with
a univariate discrete probability distribution.
```

Value

a vector of the sample quantiles of the time-persistent function provided

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

Examples

```
times  <- c(1,2,3,4,5)
counts <- c(1,2,1,1,2)
meanTPS(times, counts)
sdTPS(times, counts)
quantileTPS(times, counts)

output <- ssq(seed = 54321, maxTime = 100, saveNumInSystem = TRUE)
utilization <- meanTPS(output$numInSystemT, output$numInSystemN)
sdServerStatus <- sdTPS(output$numInSystemT, output$numInSystemN)
quantileServerStatus <- quantileTPS(output$numInSystemT, output$numInSystemN)

# compute and graphically display quantiles of number in system vs time
output <- ssq(maxArrivals = 60, seed = 54321, saveAllStats = TRUE)
quantileSys <- quantileTPS(output$numInSystemT, output$numInSystemN)
plot(output$numInSystemT, output$numInSystemN, type = "s", bty = "l",
    las = 1, xlab = "time", ylab = "number in system")
labels <- c("0%", "25%", "50%", "75%", "100%")
mtext(text = labels, side = 4, at = quantileSys, las = 1, col = "red")
abline(h = quantileSys, lty = "dashed", col = "red", lwd = 2)
```

---

queueTrace *Trace Data for Single-Server Queue Simulation*

Description

This data set contains the arrival and service times for 1000 jobs arriving to a generic single-server queue.

Usage

```
queueTrace
```

Format

A list of two vectors, `arrivalTimes` and `serviceTimes`.

Details

This trace data could be used as input for the ssq function, but not directly. That is, ssq expects interarrival and service functions as input, not vectors of arrival times and service times. Accordingly, the user will need to write functions to extract the interarrival and service times from this trace, which can then be passed to ssq. See examples below.

Source

Discrete-Event Simulation: A First Course (2006). L.M. Leemis and S.K. Park. Pearson/Prentice Hall, Upper Saddle River, NJ. ISBN-13: 978-0131429178

Examples

```
interarrivalTimes   <- c(queueTrace$arrivalTimes[1], diff(queueTrace$arrivalTimes))
serviceTimes        <- queueTrace$serviceTimes

avgInterarrivalTime <- mean(interarrivalTimes)
avgServiceTime      <- mean(serviceTimes)

# functions to use this trace data for the ssq() function;
# note that the functions below destroy the global values of the copied
# interarrivalTimes and serviceTimes vectors along the way...
#
interarrivalTimes <- NULL
serviceTimes      <- NULL
getInterarr <- function(...)
{
   if (length(interarrivalTimes) == 0) {
          interarrivalTimes <- c(queueTrace$arrivalTimes[1],
                                  diff(queueTrace$arrivalTimes))
    }
    nextInterarr <- interarrivalTimes[1]
    interarrivalTimes <- interarrivalTimes[-1]
    return(nextInterarr)
}
getService <- function(...)
{
    if (length(serviceTimes) == 0) {
        serviceTimes <- queueTrace$serviceTimes
    }
    nextService <- serviceTimes[1]
    serviceTimes <- serviceTimes[-1]
    return(nextService)
}
ssq(maxArrivals = 1000, interarrivalFcn = getInterarr, serviceFcn = getService)
```

`sample` *Random Samples*

Description

`sample` takes a sample of the specified size from the elements of `x`, either with or without replacement, and with capability to use independent streams and antithetic variates in the draws.

Usage

```
sample(
  x,
  size,
  replace = FALSE,
  prob = NULL,
  stream = NULL,
  antithetic = FALSE
)
```

Arguments

`x`	Either a vector of one or more elements from which to choose, or a positive integer
`size`	A non-negative integer giving the number of items to choose
`replace`	If `FALSE` (default), sampling is without replacement; otherwise, sample is with replacement
`prob`	A vector of probability weights for obtaining the elements of the vector being sampled
`stream`	If `NULL` (default), directly calls `base::sample` and returns its result; otherwise, an integer in 1:100 indicates the `rstream` stream used to generate the sample
`antithetic`	If `FALSE` (default), uses u = uniform(0,1) variate(s)generated via `rstream::rstream.sample` to generate the sample; otherwise, uses $1 - u$. (NB: ignored if `stream` is `NULL`.)

Details

If `stream` is `NULL`, sampling is done by direct call to `base::sample` (refer to its documentation for details). In this case, a value of `TRUE` for `antithetic` is ignored.

The remainder of details below presume that `stream` has a positive integer value, corresponding to use of the `vunif` variate generator for generating the random sample.

If `x` has length 1 and is numeric, sampling takes place from `1:x` only if `x` is a positive integer; otherwise, sampling takes place using the single value of `x` provided (either a floating-point value or a non-positive integer). Otherwise `x` can be a valid R vector, list, or data frame from which to sample.

The default for `size` is the number of items inferred from `x`, so that `sample(x, stream =` m`)` generates a random permutation of the elements of `x` (or `1:x`) using random number stream m.

It is allowed to ask for `size = 0` samples (and only then is a zero-length `x` permitted), in which case `base::sample` is invoked to return the correct (empty) data type.

The optional `prob` argument can be used to give a vector of probabilities for obtaining the elements of the vector being sampled. Unlike `base::sample`, the weights here must sum to one. If `replace` is false, these probabilities are applied successively; that is the probability of choosing the next item is proportional to the weights among the remaining items. The number of nonzero probabilities must be at least `size` in this case.

Value

If x is a single positive integer, sample returns a vector drawn from the integers 1:x. Otherwise, sample returns a vector, list, or data frame consistent with typeof(x).

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

base::sample, vunif

Examples

```
set.seed(8675309)

# use base::sample (since stream is NULL) to generate a permutation of 1:5
sample(5)

# use vunif(1, stream = 1) to generate a permutation of 1:5
sample(5, stream = 1)

# generate a (boring) sample of identical values drawn using the single value 867.5309
sample(867.5309, size = 10, replace = TRUE, stream = 1)

# use vunif(1, stream = 1) to generate a size-10 sample drawn from 7:9
sample(7:9, size = 10, replace = TRUE, stream = 1)

# use vunif(1, stream = 1) to generate a size-10 sample drawn from c('x','y','z')
sample(c('x','y','z'), size = 10, replace = TRUE, stream = 1)

# use vunif(1, stream = 1) to generate a size-5 sample drawn from a list
mylist <- list()
mylist$a <- 1:5
mylist$b <- 2:6
mylist$c <- 3:7
sample(mylist, size = 5, replace = TRUE, stream = 1)

# use vunif(1, stream = 1) to generate a size-5 sample drawn from a data frame
mydf <- data.frame(a = 1:6, b = c(1:3, 1:3))
sample(mydf, size = 5, replace = TRUE, stream = 1)
```

sdTPS *Standard Deviation of Time-Persistent Statistics (TPS)*

Description

Computes the sample standard deviation of a time-persistent function.

Usage

```
sdTPS(times = NULL, numbers = NULL)
```

Arguments

`times`	A numeric vector of non-decreasing time observations
`numbers`	A numeric vector containing the values of the time-persistent statistic between the time observation

Details

```
The lengths of \code{times} and \code{numbers} either must be
the same, or \code{times} may have one more entry than \code{numbers}
(interval endpoints vs. interval counts). The sample variance is the
area under the square of the step-function created by the values in
\code{numbers} between the first and last element in \code{times} divided
by the length of the observation period, less the square of the sample mean.
The sample standard deviation is the square root of the sample variance.
```

Value

the sample standard deviation of the time-persistent function provided

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

Examples

```
times  <- c(1,2,3,4,5)
counts <- c(1,2,1,1,2)
meanTPS(times, counts)
sdTPS(times, counts)

output <- ssq(seed = 54321, maxTime = 100, saveServerStatus = TRUE)
utilization <- meanTPS(output$serverStatusT, output$serverStatusN)
sdServerStatus <- sdTPS(output$serverStatusT, output$serverStatusN)

# compute and graphically display mean and sd of number in system vs time
output <- ssq(maxArrivals = 60, seed = 54321, saveAllStats = TRUE)
plot(output$numInSystemT, output$numInSystemN, type = "s", bty = "l",
   las = 1, xlab = "time", ylab = "number in system")
meanSys <- meanTPS(output$numInSystemT, output$numInSystemN)
sdSys   <- sdTPS(output$numInSystemT, output$numInSystemN)
```

```
abline(h = meanSys, lty = "solid", col = "red", lwd = 2)
abline(h = c(meanSys - sdSys, meanSys + sdSys),
   lty = "dashed", col = "red", lwd = 2)
```

set.seed — *Seeding Random Variate Generators*

Description

`set.seed` in the `simEd` package allows the user to simultaneously set the initial seed for both the `stats` and `simEd` variate generators.

Usage

```
set.seed(seed, kind = NULL, normal.kind = NULL)
```

Arguments

`seed`	A single value, interpreted as an integer, or `NULL` (see 'Details')
`kind`	Character or `NULL`. This is passed verbatim to `base::set.seed`.
`normal.kind`	Character or `NULL`. This is passed verbatim to `base::set.seed`.

Details

This function intentionally masks the `base::set.seed` function, allowing the user to simultaneously set the initial seed for the `stats` variate generators (by explicitly calling `base::set.seed`) and for the `simEd` variate generators (by explicitly setting up 10 streams using the `rstream.mrg32k3a` generator from the `rstream` package).

Any call to `set.seed` re-initializes the seed for the `stats` and `simEd` generators as if no seed had been set. If called with `seed = NULL`, both the `stats` and `simEd` variate generators are re-initialized using a random seed based on the system clock.

If the user wishes to set the seed for the `stats` generators without affecting the seeds of the `simEd` generators, an explicit call to `base::set.seed` can be made.

Note that once `set.seed` is called, advancing the `simEd` generator state using any of the stream-based `simEd` variate generators will not affect the state of the non-stream-based `stats` generators, and vice-versa.

As soon as the `simEd` package is attached (i.e., when `simEd` is the parent of the global environment), `simEd::set.seed` becomes the default for a call to `set.seed`. When the `simEd` package is detached, `base::set.seed` will revert to the default.

Value

`set.seed` returns `NULL`, invisibly, consistent with `base::set.seed`.

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

base::set.seed

Examples

```
set.seed(8675309)
rexp(3, rate = 2)  # explicit call of stats::rexp

set.seed(8675309)
vexp(3, rate = 2)  # also uses stats::rexp

set.seed(8675309)
vexp(3, rate = 2, stream = 1) # uses rstream and stats::qexp
vexp(3, rate = 2, stream = 2)
rexp(3, rate = 2) # explicit call of stats::rexp, starting with seed 8675309

set.seed(8675309)
vexp(1, rate = 2, stream = 1) # uses rstream and stats::qexp
vexp(1, rate = 2, stream = 2)
vexp(1, rate = 2, stream = 1)
vexp(1, rate = 2, stream = 2)
vexp(1, rate = 2, stream = 1)
vexp(1, rate = 2, stream = 2)
vexp(3, rate = 2)             # calls stats::rexp, starting with seed 8675309
```

---

ssq — *Single-Server Queue Simulation*

Description

A next-event simulation of a single-server queue, with extensible arrival and service processes.

Usage

```
ssq(
  maxArrivals = Inf,
  seed = NA,
  interarrivalFcn = NULL,
  serviceFcn = NULL,
  interarrivalType = "M",
  serviceType = "M",
```

```
  maxTime = Inf,
  maxDepartures = Inf,
  maxInSystem = Inf,
  maxEventsPerSkyline = 15,
  saveAllStats = FALSE,
  saveInterarrivalTimes = FALSE,
  saveServiceTimes = FALSE,
  saveWaitTimes = FALSE,
  saveSojournTimes = FALSE,
  saveNumInQueue = FALSE,
  saveNumInSystem = FALSE,
  saveServerStatus = FALSE,
  showOutput = TRUE,
  animate = FALSE,
  showQueue = NULL,
  showSkyline = NULL,
  showSkylineSystem = FALSE,
  showSkylineQueue = FALSE,
  showSkylineServer = FALSE,
  showTitle = TRUE,
  showProgress = TRUE,
  plotQueueFcn = defaultPlotSSQ,
  plotSkylineFcn = defaultPlotSkyline,
  jobImage = NA,
  plotDelay = NA,
  respectLayout = FALSE
)
```

Arguments

`maxArrivals` maximum number of customer arrivals allowed to enter the system

`seed` initial seed to the random number generator (NA uses current state of random number generator; NULL seeds using system clock)

`interarrivalFcn`
function for generating interarrival times for queue simulation. Default value (`NA`) will result in use of default interarrival function based on `interarrivalType`. See examples.

`serviceFcn` function for generating service times for queue simulation. Default value (`NA`) will result in use of default service function based on `serviceType`. See examples.

`interarrivalType`
string representation of desired interarrival process. Options are "M" – exponential with rate 1; "G" – uniform(0,2), having mean 1; and "D" – deterministic with constant value 1. Default is "M".

`serviceType` string representation of desired service process . Options are "M" – exponential with rate 10/9; "G" – uniform(0, 1.8), having mean 9/10; and "D" – deterministic with constant value 9/10. Default is "M".

`maxTime` maximum time to simulate

`maxDepartures` maximum number of customer departures to process

`maxInSystem` maximum number of customers that the system can hold (server(s) plus queue). Infinite by default.

`maxEventsPerSkyline` maximum number of events viewable at a time in the skyline plot. A large value for this parameter may result in plotting delays. This parameter does not impact the final plotting, which will show all end-of-simulation results.

`saveAllStats` if `TRUE`, returns all vectors of statistics (see below) collected by the simulation

`saveInterarrivalTimes` if `TRUE`, returns a vector of all interarrival times generated

`saveServiceTimes` if `TRUE`, returns a vector of all service times generated

`saveWaitTimes` if `TRUE`, returns a vector of all wait times (in the queue) generated

`saveSojournTimes` if `TRUE`, returns a vector of all sojourn times (time spent in the system) generated

`saveNumInQueue` if `TRUE`, returns a vector of times and a vector of counts for whenever the number in the queue changes

`saveNumInSystem` if `TRUE`, returns a vector of times and a vector of counts for whenever the number in the system changes

`saveServerStatus` if `TRUE`, returns a vector of times and a vector of server status (0:idle, 1:busy) for whenever the status changes

`showOutput` if `TRUE`, displays summary statistics upon completion

`animate` logical; if `FALSE`, no animation plots will be shown.

`showQueue` logical; if `TRUE`, displays a visualization of the queue

`showSkyline` If `NULL` (default), defers to each individual showSkyline... parameter below; otherwise, supersedes individual showSkyline... parameter values. If `TRUE`, displays full skyline plot; `FALSE` suppresses skyline plot. Can alternatively be specified using chmod-like octal component specification: use 1, 2, 4 for system, queue, and server respectively, summing to indicate desired combination (e.g., 7 for all). Can also be specified as a binary vector (e.g., c(1,1,1) for all).

`showSkylineSystem` logical; if `TRUE`, includes number in system as part of skyline plot. Value for `showSkyline` supersedes this parameter's value.

`showSkylineQueue` logical; if `TRUE`, includes number in queue as part of skyline plot. Value for `showSkyline` supersedes this parameter's value.

`showSkylineServer` logical; if `TRUE`, includes number in server as part of skyline plot. Value for `showSkyline` supersedes this parameter's value.

`showTitle` if `TRUE`, display title at the top of the main plot

showProgress	if TRUE, displays a progress bar on screen during no-animation execution
plotQueueFcn	plotting function to display queue visualization. By default, this is provided by defaultPlotSSQ. Please refer to that associated help for more details about required arguments.
plotSkylineFcn	plotting function to display Skyline visualization. By default, this is provided by defaultPlotSkyline. Please refer to that associated help for more details about required arguments.
jobImage	a vector of URLs/local addresses of images to use as jobs. Requires package 'magick'.
plotDelay	a positive numeric value indicating seconds between plots. A value of -1 enters 'interactive' mode, where the state will pause for user input at each step. A value of 0 will display only the final end-of-simulation plot.
respectLayout	logical; if TRUE, plot layout (i.e. par, device, etc.) settings will be respected.

Details

Implements a next-event implementation of a single-server queue simulation.

The seed parameter can take one of three valid argument types:

- NA (default), which will use the current state of the random number generator without explicitly setting a new seed (see examples);
- a positive integer, which will be used as the initial seed passed in an explicit call to set.seed; or
- NULL, which will be passed in an explicit call to to set.seed, thereby setting the initial seed using the system clock.

Value

The function returns a list containing:

- the number of arrivals to the system (customerArrivals),
- the number of customers processed (customerDepartures),
- the ending time of the simulation (simulationEndTime),
- average wait time in the queue (avgWait),
- average time in the system (avgSojourn),
- average number in the system (avgNumInSystem),
- average number in the queue (avgNumInQueue), and
- server utilization (utilization).

of the queue as computed by the simulation. When requested via the "save..." parameters, the list may also contain:

- a vector of interarrival times (interarrivalTimes),
- a vector of wait times (waitTimes),
- a vector of service times (serviceTimes),

- a vector of sojourn times (sojournTimes),
- two vectors (time and count) noting changes to number in the system (numInSystemT, numInSystemN),
- two vectors (time and count) noting changes to number in the queue (numInQueueT, numInQueueN), and
- two vectors (time and status) noting changes to server status (serverStatusT, serverStatusN).

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

rstream, set.seed, stats::runif

Examples

```
# process 100 arrivals, R-provided seed (via NULL seed)
ssq(100, NULL)

ssq(maxArrivals = 100, seed = 54321)
ssq(maxDepartures = 100, seed = 54321)
ssq(maxTime = 100, seed = 54321)

############################################################################
# example to show use of seed = NA (default) to rely on current state of generator
output1 <- ssq(200, 8675309, showOutput = FALSE, saveAllStats = TRUE)
output2 <- ssq(300,          showOutput = FALSE, saveAllStats = TRUE)
set.seed(8675309)
output3 <- ssq(200,          showOutput = FALSE, saveAllStats = TRUE)
output4 <- ssq(300,          showOutput = FALSE, saveAllStats = TRUE)
sum(output1$sojournTimes != output3$sojournTimes) # should be zero
sum(output2$sojournTimes != output4$sojournTimes) # should be zero

myArrFcn <- function() { vexp(1, rate = 1/4, stream = 1)  }  # mean is 4
mySvcFcn <- function() { vgamma(1, shape = 1, rate = 0.3) }  # mean is 3.3

output <- ssq(maxArrivals = 100, interarrivalFcn = myArrFcn, serviceFcn = mySvcFcn,
             saveAllStats = TRUE)
mean(output$interarrivalTimes)
mean(output$serviceTimes)
meanTPS(output$numInQueueT, output$numInQueueN) # compute time-averaged num in queue
meanTPS(output$serverStatusT, output$serverStatusN) # compute server utilization

############################################################################
# example to show use of (simple) trace data for arrivals and service times;
# ssq() will need one more interarrival (arrival) time than jobs processed
#
arrivalTimes      <- NULL
interarrivalTimes <- NULL
```

```
serviceTimes      <- NULL

initTimes <- function() {
    arrivalTimes      <<- c(15, 47, 71, 111, 123, 152, 232, 245, 99999)
    interarrivalTimes <<- c(arrivalTimes[1], diff(arrivalTimes))
    serviceTimes      <<- c(43, 36, 34, 30, 38, 30, 31, 29)
}

getInterarr <- function() {
    nextInterarr <- interarrivalTimes[1]
    interarrivalTimes <<- interarrivalTimes[-1]  # remove 1st element globally
    return(nextInterarr)
}

getService <- function() {
    nextService <- serviceTimes[1]
    serviceTimes <<- serviceTimes[-1]  # remove 1st element globally
    return(nextService)
}

initTimes()
numJobs <- length(serviceTimes)
output <- ssq(maxArrivals = numJobs, interarrivalFcn = getInterarr,
              serviceFcn = getService, saveAllStats = TRUE)
mean(output$interarrivalTimes)
mean(output$serviceTimes)


############################################################################
# example to show use of (simple) trace data for arrivals and service times,
# allowing for reuse (recycling) of trace data times
arrivalTimes      <- NULL
interarrivalTimes <- NULL
serviceTimes      <- NULL

initArrivalTimes <- function() {
  arrivalTimes      <<- c(15, 47, 71, 111, 123, 152, 232, 245)
  interarrivalTimes <<- c(arrivalTimes[1], diff(arrivalTimes))
}

initServiceTimes <- function() {
    serviceTimes    <<- c(43, 36, 34, 30, 38, 30, 31, 29)
}

getInterarr <- function() {
    if (length(interarrivalTimes) == 0)  initArrivalTimes()

    nextInterarr <- interarrivalTimes[1]
    interarrivalTimes <<- interarrivalTimes[-1] # remove 1st element globally
    return(nextInterarr)
}

getService <- function() {
```

```
    if (length(serviceTimes) == 0)  initServiceTimes()

    nextService <- serviceTimes[1]
    serviceTimes <<- serviceTimes[-1]  # remove 1st element globally
    return(nextService)
}

initArrivalTimes()
initServiceTimes()
output <- ssq(maxArrivals = 100, interarrivalFcn = getInterarr,
              serviceFcn = getService, saveAllStats = TRUE)
mean(output$interarrivalTimes)
mean(output$serviceTimes)

############################################################################
# Testing with visualization

# Visualizing ssq with a set seed, infinite queue capacity, 20 arrivals,
# interactive mode (default), showing skyline for all 3 attributes (default)
if (interactive()) {
  ssq(seed = 1234, maxArrivals = 20, animate = TRUE)
}

# Same as above, but jump to final queue visualization using plotDelay 0
ssq(seed = 1234, maxArrivals = 20, animate = TRUE, plotDelay = 0)

# Perform simulation again with finite queue of low capacity. Note same
# variate generation but different outcomes due to rejection pathway
ssq(seed = 1234, maxArrivals = 25, animate = TRUE, maxInSystem = 5, plotDelay = 0)

# Using default distributions to simulate a default M/G/1 Queue
ssq(seed = 1234, maxDepartures = 10, interarrivalType = "M", serviceType = "G",
    animate = TRUE, plotDelay = 0)
```

---

ssqvis *Single-Server Queue Simulation Visualization*

Description

A modified ssq implementation that illustrates event-driven details, including the event calendar, inversion for interarrival and service time variate generation, the simulation clock, the status of the queueing system, and statistics collection. The function plots step-by-step in either an interactive mode or time-delayed automatic mode.

Usage

```
ssqvis(
  maxArrivals = Inf,
  seed = NA,
```

```
  interarrivalType = "M",
  serviceType = "M",
  maxTime = Inf,
  maxDepartures = Inf,
  maxEventsPerSkyline = 15,
  saveAllStats = FALSE,
  saveInterarrivalTimes = FALSE,
  saveServiceTimes = FALSE,
  saveWaitTimes = FALSE,
  saveSojournTimes = FALSE,
  saveNumInQueue = FALSE,
  saveNumInSystem = FALSE,
  saveServerStatus = FALSE,
  showOutput = TRUE,
  showSkyline = NULL,
  showSkylineQueue = TRUE,
  showSkylineSystem = TRUE,
  showSkylineServer = TRUE,
  showTitle = TRUE,
  jobImage = NA,
  plotDelay = -1
)
```

Arguments

`maxArrivals`	maximum number of customer arrivals allowed to enter the system
`seed`	initial seed to the random number generator (NA uses current state of random number generator; NULL seeds using system clock)
`interarrivalType`	string representation of desired interarrival process. Options are "M" – exponential with rate 1; "G" – uniform(0,2), having mean 1; and "D" – deterministic with constant value 1. Default is "M".
`serviceType`	string representation of desired service process . Options are "M" – exponential with rate 10/9; "G" – uniform(0, 1.8), having mean 9/10; and "D" – deterministic with constant value 9/10. Default is "M".
`maxTime`	maximum time to simulate
`maxDepartures`	maximum number of customer departures to process
`maxEventsPerSkyline`	maximum number of events viewable at a time in the skyline plot. A large value for this parameter may result in plotting delays. This parameter does not impact the final plotting, which will show all end-of-simulation results.
`saveAllStats`	if `TRUE`, returns all vectors of statistics (see below) collected by the simulation
`saveInterarrivalTimes`	if `TRUE`, returns a vector of all interarrival times generated
`saveServiceTimes`	if `TRUE`, returns a vector of all service times generated

- `saveWaitTimes` — if `TRUE`, returns a vector of all wait times (in the queue) generated
- `saveSojournTimes` — if `TRUE`, returns a vector of all sojourn times (time spent in the system) generated
- `saveNumInQueue` — if `TRUE`, returns a vector of times and a vector of counts for whenever the number in the queue changes
- `saveNumInSystem` — if `TRUE`, returns a vector of times and a vector of counts for whenever the number in the system changes
- `saveServerStatus` — if `TRUE`, returns a vector of times and a vector of server status (0:idle, 1:busy) for whenever the status changes
- `showOutput` — if `TRUE`, displays summary statistics upon completion
- `showSkyline` — If `NULL` (default), defers to each individual showSkyline... parameter below; otherwise, supersedes individual showSkyline... parameter values. If `TRUE`, displays full skyline plot; `FALSE` suppresses skyline plot. Can alternatively be specified using chmod-like octal component specification: use 1, 2, 4 for system, queue, and server respectively, summing to indicate desired combination (e.g., 7 for all). Can also be specified as a binary vector (e.g., c(1,1,1) for all).
- `showSkylineQueue` — logical; if `TRUE`, includes number in queue as part of skyline plot. Value for `showSkyline` supersedes this parameter's value.
- `showSkylineSystem` — logical; if `TRUE`, includes number in system as part of skyline plot. Value for `showSkyline` supersedes this parameter's value.
- `showSkylineServer` — logical; if `TRUE`, includes number in server as part of skyline plot. Value for `showSkyline` supersedes this parameter's value.
- `showTitle` — if `TRUE`, display title at the top of the main plot
- `jobImage` — a vector of URLs/local addresses of images to use as jobs. Requires package `'magick'`.
- `plotDelay` — a positive numeric value indicating seconds between plots. A value of -1 enters 'interactive' mode, where the state will pause for user input at each step. A value of 0 will display only the final end-of-simulation plot.

Details

Animates the details of an event-driven implementation of a single-server queue simulation.

The event calendar, inversion for interarrival and service time variates, and an abbreviated (current) timeline are animated in the top pane of the window. In this pane, blue corresponds to the arrival process, orange corresponds to the service process, and purple corresponds to uniform variates used in inversion. Yellow is used to highlight recent updates.

The state of the queueing system is animated in the middle pane of the window. In this pane, red indicates an idle server, orange indicates that a new customer has just arrived to the server and a corresponding service time is being generated, and green indicates a busy server. By default, customers are depicted as black rectangles and identified by increasing arrival number, but this depiction can be overridden by the `jobImage` parameter.

Statistics are displayed in the bottom pane of the window. Time-persistent statistics are shown as "skyline functions" in the left portion of this pane. Both time-persistent and based-on-observation statistics are shown in respective tables in the right portion of this pane. In the tables, yellow is used to highlight recent updates.

The `seed` parameter can take one of three valid argument types:

- `NA` (default), which will use the current state of the random number generator without explicitly setting a new seed (see examples);
- a positive integer, which will be used as the initial seed passed in an explicit call to `set.seed`; or
- `NULL`, which will be passed in an explicit call to to `set.seed`, thereby setting the initial seed using the system clock.

Value

The function returns a list containing:

- the number of arrivals to the system (`customerArrivals`),
- the number of customers processed (`customerDepartures`),
- the ending time of the simulation (`simulationEndTime`),
- average wait time in the queue (`avgWait`),
- average time in the system (`avgSojourn`),
- average number in the system (`avgNumInSystem`),
- average number in the queue (`avgNumInQueue`), and
- server utilization (`utilization`).

of the queue as computed by the simulation. When requested via the “save...” parameters, the list may also contain:

- a vector of interarrival times (`interarrivalTimes`),
- a vector of wait times (`waitTimes`),
- a vector of service times (`serviceTimes`),
- a vector of sojourn times (`sojournTimes`),
- two vectors (time and count) noting changes to number in the system (`numInSystemT`, `numInSystemN`),
- two vectors (time and count) noting changes to number in the queue (`numInQueueT`, `numInQueueN`), and
- two vectors (time and status) noting changes to server status (`serverStatusT`, `serverStatusN`).

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

rstream, set.seed, stats::runif

Examples

```
# Visualizing ssq with a set seed, infinite queue capacity, 4 arrivals,
# and showing skyline with number in system, queue, and server.
ssqvis(seed = 1234, maxArrivals = 4, showSkyline = 7, plotDelay = 0.01)
```

---

thinning *Thinning Algorithm Visualization*

Description

This function animates the "thinning" approach the generation of the random event times for a non-homogeneous Poisson process with a specified intensity function, given a majorizing function that dominates the intensity function.

Usage

```
thinning(
  maxTime = 24,
  intensityFcn = function(x) (5 - sin(x/0.955) - (4 * cos(x/3.82)))/0.5,
  majorizingFcn = NULL,
  majorizingFcnType = NULL,
  seed = NA,
  maxTrials = Inf,
  plot = TRUE,
  showTitle = TRUE,
  plotDelay = plot * -1
)
```

Arguments

maxTime	maximum time of the non-homogeneous Poisson process. (The minimum time is assumed to be zero.)
intensityFcn	intensity function corresponding to rate of arrivals across time.
majorizingFcn	majorizing function. Default value is NULL, corresponding to a constant majorizing function that is 1.01 times the maximum value of the intensity function. May alternatively be provided as a user-specified function, or as a data frame requiring additional notation as either piecewise-constant or piecewise-linear. See examples.
majorizingFcnType	used to indicate whether a majorizing function that is provided via data frame is to be interpreted as either piecewise-constant ("pwc") or piecewise-linear ("pwl"). If the majorizing function is either the default or a user-specified function (closure), the value of this parameter is ignored.

`seed`	initial seed for the uniform variates used during generation.
`maxTrials`	maximum number of accept-reject trials; infinite by default.
`plot`	if TRUE, visual display will be produced. If FALSE, generated event times will be returned without visual display.
`showTitle`	if TRUE, display title in the main plot.
`plotDelay`	wait time, in seconds, between plots; -1 (default) for interactive mode, where the user is queried for input to progress.

Details

There are three modes for visualizing Lewis and Shedler's thinning algorithm for generating random event times for a non-homogeneous Poisson process with a particular intensity function:

- interactive advance (`plotDelay = -1`), where pressing the 'ENTER' key advances to the next step (an accepted random variate) in the algorithm, typing 'j #' jumps ahead # steps, typing 'q' quits immediately, and typing 'e' proceeds to the end;
- automatic advance (`plotDelay` > 0); or
- final visualization only (`plotDelay = 0`).

As an alternative to visualizing, event times can be generated

Value

returns a vector of the generated random event times

References

Lewis, P.A.W. and Shedler, G.S. (1979). Simulation of non-homogeneous Poisson processes by thinning. *Naval Research Logistics*, **26**, 403–413.

Examples

```
nhpp <- thinning(maxTime = 12, seed = 8675309, plotDelay = 0)
nhpp <- thinning(maxTime = 24, seed = 8675309, plotDelay = 0)


nhpp <- thinning(maxTime = 48, seed = 8675309, plotDelay = 0)

# thinning with custom intensity function and default majorizing function
intensity <- function(x) {
    day <- 24 * floor(x/24)
    return(80 * (dnorm(x, day + 6,    2.5) +
                 dnorm(x, day + 12.5, 1.5) +
                 dnorm(x, day + 19,   2.0)))
}
nhpp <- thinning(maxTime = 24, plotDelay = 0, intensityFcn = intensity)

# thinning with custom intensity and constant majorizing functions
major <- function(x) { 25 }
nhpp <- thinning(maxTime = 24, plotDelay = 0, intensityFcn = intensity,
```

```
                 majorizingFcn = major)

# piecewise-constant data.frame for bounding default intensity function
fpwc <- data.frame(
    x = c(0, 2, 20, 30, 44, 48),
    y = c(5, 5, 20, 12, 20,  5)
)
nhpp <- thinning(maxTime = 24, plotDelay = 0, majorizingFcn = fpwc, majorizingFcnType = "pwc")

# piecewise-linear data.frame for bounding default intensity function
fpwl <- data.frame(
    x = c(0, 12, 24, 36, 48),
    y = c(5, 25, 10, 25, 5)
)
nhpp <- thinning(maxTime = 24, plotDelay = 0, majorizingFcn = fpwl, majorizingFcnType = "pwl")

# piecewise-linear closure/function bounding default intensity function
fclo <- function(x) {
    if (x <= 12) (5/3)*x + 5
    else if (x <= 24) 40 - (5/4)*x
    else if (x <= 36) (5/4)*x - 20
    else 85 - (5/3) * x
}
nhpp <- thinning(maxTime = 48, plotDelay = 0, majorizingFcn = fclo)

# thinning with fancy custom intensity function and default majorizing
intensity <- function(x) {
    day <- 24 * floor(x/24)
    return(80 * (dnorm(x, day + 6,    2.5) +
                 dnorm(x, day + 12.5, 1.5) +
                 dnorm(x, day + 19,   2.0)))
}
nhpp <- thinning(maxTime = 24, plotDelay = 0, intensityFcn = intensity)

# piecewise-linear data.frame for bounding custom intensity function
fpwl <- data.frame(
    x = c(0,  6,  9, 12, 16, 19, 24, 30, 33, 36, 40, 43, 48),
    y = c(5, 17, 12, 28, 14, 18,  7, 17, 12, 28, 14, 18,  7)
)
nhpp <- thinning(maxTime = 48, plotDelay = 0, intensityFcn = intensity,
          majorizingFcn = fpwl, majorizingFcnType = "pwl")

# thinning with simple custom intensity function and custom majorizing
intensity <- function(t) {
    if      (t < 12) t
    else if (t < 24) 24 - t
    else if (t < 36) t - 24
    else             48 - t
}
majorizing <- data.frame(
    x = c(0, 12, 24, 36, 48),
    y = c(1, 13,  1, 13,  1))
times <- thinning(plotDelay = 0, intensityFcn = intensity,
```

```
    majorizingFcn = majorizing , majorizingFcnType = "pwl", maxTime = 48)
```

---

tylersGrill *Arrival and Service Data for Tyler's Grill (University of Richmond)*

Description

This data set contains a list of two vectors of data.

The first vector in the list contains the arrival times for 1434 customers arriving to Tyler's Grill at the University of Richmond during a single day in 2005. The arrival times were collected during operating hours, from 07:30 until 21:00. Arrival times are provided in seconds from opening (07:30).

The second vector contains service times sample for 110 customers at Tyler's Grill in 2005. Service times are provided in seconds.

Usage

```
tylersGrill
```

Format

`tylersGrill$arrivalTimes` returns the vector of 1434 arrival times.
`tylersGrill$serviceTimes` returns the vector of 110 service times.

Source

CMSC 326 Simulation course at the University of Richmond, 2005.

Examples

```
interarr <- c(0, diff(tylersGrill$arrivalTimes))
svc      <- tylersGrill$serviceTimes

avgInterarrivalTime <- mean(interarr)
avgServiceTime      <- mean(svc)

# use method of moments to fit gamma to Tyler's Grill service times
aHat <- mean(svc)^2 / var(svc)
bHat <- var(svc) / mean(svc)
hist(svc, freq = FALSE, las = 1, xlab = "service time", ylab = "density")
x <- 1:max(svc)
curve(dgamma(x, shape = aHat, scale = bHat), add = TRUE, col = "red", lwd = 2)
```

vbeta *Variate Generation for Beta Distribution*

Description

Variate Generation for Beta Distribution

Usage

```
vbeta(
  n,
  shape1,
  shape2,
  ncp = 0,
  stream = NULL,
  antithetic = FALSE,
  asList = FALSE
)
```

Arguments

n	number of observations
shape1	Shape parameter 1 (alpha)
shape2	Shape parameter 2 (beta)
ncp	Non-centrality parameter (default 0)
stream	if NULL (default), uses stats::runif to generate uniform variates to invert via stats::qbeta; otherwise, an integer in 1:25 indicates the rstream stream from which to generate uniform variates to invert via stats::qbeta;
antithetic	if FALSE (default), inverts u = uniform(0,1) variate(s) generated via either stats::runif or rstream::rstream.sample; otherwise, uses $1 - u$
asList	if FALSE (default), output only the generated random variates; otherwise, return a list with components suitable for visualizing inversion. See return for details

Details

Generates random variates from the beta distribution.

Beta variates are generated by inverting uniform(0,1) variates produced either by stats::runif (if stream is NULL) or by rstream::rstream.sample (if stream is not NULL). In either case, stats::qbeta is used to invert the uniform(0,1) variate(s). In this way, using vbeta provides a monotone and synchronized binomial variate generator, although not particularly fast.

The stream indicated must be an integer between 1 and 25 inclusive.

The beta distribution has density

```
\deqn{f(x) = \frac{\Gamma(a+b)}{\Gamma(a) \ \Gamma(b)} x^{a-1}(1-x)^{b-1}}{
          f(x) = Gamma(a+b)/(Gamma(a)Gamma(b)) x^(a-1)(1-x)^(b-1)}
```

for $a > 0$, $b > 0$ and $0 \leq x \leq 1$ where the boundary values at $x = 0$ or $x = 1$ are defined as by continuity (as limits).

The mean is $\frac{a}{a+b}$ and the variance is $ab(a + b)^2(a + b + 1)$

Value

If asList is FALSE (default), return a vector of random variates.

Otherwise, return a list with components suitable for visualizing inversion, specifically:

u	A vector of generated U(0,1) variates
x	A vector of beta random variates
quantile	Parameterized quantile function
text	Parameterized title of distribution

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

rstream, set.seed, stats::runif

stats::rbeta

Examples

```
set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qbeta
vbeta(3, shape1 = 3, shape2 = 1, ncp = 2)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qbeta
vbeta(3, 3, 1, stream = 1)
vbeta(3, 3, 1, stream = 2)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qbeta
vbeta(1, 3, 1, stream = 1)
vbeta(1, 3, 1, stream = 2)
vbeta(1, 3, 1, stream = 1)
vbeta(1, 3, 1, stream = 2)
vbeta(1, 3, 1, stream = 1)
vbeta(1, 3, 1, stream = 2)

set.seed(8675309)
variates <- vbeta(100, 3, 1, stream = 1)
set.seed(8675309)
variates <- vbeta(100, 3, 1, stream = 1, antithetic = TRUE)
```

vbinom *Variate Generation for Binomial Distribution*

Description

Variate Generation for Binomial Distribution

Usage

```
vbinom(n, size, prob, stream = NULL, antithetic = FALSE, asList = FALSE)
```

Arguments

n	number of observations
size	number of trials (zero or more)
prob	probability of success on each trial ($0 <$ prob ≤ 1)
stream	if NULL (default), uses stats::runif to generate uniform variates to invert via stats::qbinom; otherwise, an integer in 1:25 indicates the rstream stream from which to generate uniform variates to invert via stats::qbinom;
antithetic	if FALSE (default), inverts u = uniform(0,1) variate(s) generated via either stats::runif or rstream::rstream.sample; otherwise, uses $1 - u$
asList	if FALSE (default), output only the generated random variates; otherwise, return a list with components suitable for visualizing inversion. See return for details

Details

Generates random variates from the binomial distribution.

Binomial variates are generated by inverting uniform(0,1) variates produced either by stats::runif (if stream is NULL) or by rstream::rstream.sample (if stream is not NULL). In either case, stats::qbinom is used to invert the uniform(0,1) variate(s). In this way, using vbinom provides a monotone and synchronized binomial variate generator, although not particularly fast.

The stream indicated must be an integer between 1 and 25 inclusive.

The binomial distribution with parameters size = n and prob = p has pmf

$$p(x) = \binom{n}{x} p^x (1-p)^{(n-x)}$$

for $x = 0, \ldots, n$.

Value

If asList is FALSE (default), return a vector of random variates.

Otherwise, return a list with components suitable for visualizing inversion, specifically:

u	A vector of generated U(0,1) variates
x	A vector of binomial random variates
quantile	Parameterized quantile function
text	Parameterized title of distribution

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

rstream, set.seed, stats::runif

stats::rbinom

Examples

```
set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qbinom
vbinom(3, size = 10, prob = 0.25)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qbinom
vbinom(3, 10, 0.25, stream = 1)
vbinom(3, 10, 0.25, stream = 2)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qbinom
vbinom(1, 10, 0.25, stream = 1)
vbinom(1, 10, 0.25, stream = 2)
vbinom(1, 10, 0.25, stream = 1)
vbinom(1, 10, 0.25, stream = 2)
vbinom(1, 10, 0.25, stream = 1)
vbinom(1, 10, 0.25, stream = 2)

set.seed(8675309)
variates <- vbinom(100, 10, 0.25, stream = 1)
set.seed(8675309)
variates <- vbinom(100, 10, 0.25, stream = 1, antithetic = TRUE)
```

---

vcauchy *Variate Generation for Cauchy Distribution*

Description

Variate Generation for Cauchy Distribution

Usage

```
vcauchy(
  n,
  location = 0,
```

```
    scale = 1,
    stream = NULL,
    antithetic = FALSE,
    asList = FALSE
)
```

Arguments

n	number of observations
location	Location parameter (default 0)
scale	Scale parameter (default 1)
stream	if NULL (default), uses stats::runif to generate uniform variates to invert via stats::qcauchy; otherwise, an integer in 1:25 indicates the rstream stream from which to generate uniform variates to invert via stats::qcauchy;
antithetic	if FALSE (default), inverts u = uniform(0,1) variate(s) generated via either stats::runif or rstream::rstream.sample; otherwise, uses $1 - u$
asList	if FALSE (default), output only the generated random variates; otherwise, return a list with components suitable for visualizing inversion. See return for details

Details

Generates random variates from the Cauchy distribution.

Cauchy variates are generated by inverting uniform(0,1) variates produced either by stats::runif (if stream is NULL) or by rstream::rstream.sample (if stream is not NULL). In either case, stats::qcauchy is used to invert the uniform(0,1) variate(s). In this way, using vcauchy provides a monotone and synchronized binomial variate generator, although not particularly fast.

The stream indicated must be an integer between 1 and 25 inclusive.

```
The Cauchy distribution has density
\deqn{f(x) = \frac{1}{\pi s} \ \left(1 + \left( \frac{x - l}{s} \right)^2
              \right)^{-1}}{
          f(x) = 1 / (\pi s (1 + ((x-l)/s)^2))}
```

for all x.

The mean is $a/(a + b)$ and the variance is $ab/((a + b)^2(a + b + 1))$.

Value

If asList is FALSE (default), return a vector of random variates.

Otherwise, return a list with components suitable for visualizing inversion, specifically:

u	A vector of generated U(0,1) variates
x	A vector of Cauchy random variates
quantile	Parameterized quantile function
text	Parameterized title of distribution

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

rstream, set.seed, stats::runif

stats::rcauchy

Examples

```
set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qcauchy
vcauchy(3, location = 3, scale = 1)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qcauchy
vcauchy(3, 0, 3, stream = 1)
vcauchy(3, 0, 3, stream = 2)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qcauchy
vcauchy(1, 0, 3, stream = 1)
vcauchy(1, 0, 3, stream = 2)
vcauchy(1, 0, 3, stream = 1)
vcauchy(1, 0, 3, stream = 2)
vcauchy(1, 0, 3, stream = 1)
vcauchy(1, 0, 3, stream = 2)

set.seed(8675309)
variates <- vcauchy(100, 0, 3, stream = 1)
set.seed(8675309)
variates <- vcauchy(100, 0, 3, stream = 1, antithetic = TRUE)
```

---

vchisq *Variate Generation for Chi-Squared Distribution*

Description

Variate Generation for Chi-Squared Distribution

Usage

```
vchisq(n, df, ncp = 0, stream = NULL, antithetic = FALSE, asList = FALSE)
```

Arguments

n	number of observations
df	Degrees of freedom (non-negative, but can be non-integer)
ncp	Non-centrality parameter (non-negative)
stream	if NULL (default), uses stats::runif to generate uniform variates to invert via stats::qchisq; otherwise, an integer in 1:25 indicates the rstream stream from which to generate uniform variates to invert via stats::qchisq;
antithetic	if FALSE (default), inverts u = uniform(0,1) variate(s) generated via either stats::runif or rstream::rstream.sample; otherwise, uses $1 - u$
asList	if FALSE (default), output only the generated random variates; otherwise, return a list with components suitable for visualizing inversion. See return for details

Details

Generates random variates from the chi-squared distribution.

Chi-Squared variates are generated by inverting uniform(0,1) variates produced either by stats::runif (if stream is NULL) or by rstream::rstream.sample (if stream is not NULL). In either case, stats::qchisq is used to invert the uniform(0,1) variate(s). In this way, using vchisq provides a monotone and synchronized binomial variate generator, although not particularly fast.

The stream indicated must be an integer between 1 and 25 inclusive.

The chi-squared distribution with df = $n \geq 0$ degrees of freedom has density

```
\deqn{f_n(x) = \frac{1}{2^{n/2} \ \Gamma(n/2)} x^{n/2-1} e^{-x/2}}{
          f_n(x) = 1 / (2^(n/2) \Gamma(n/2)) x^(n/2-1) e^(-x/2)}
```

for $x > 0$. The mean and variance are n and $2n$.

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

```
\deqn{f(x) = e^{-\lambda/2} \sum_{r=0}^\infty \frac{(\lambda/2)^r}{r!} f_{n + 2r}(x)}{
       f(x) = exp(-\lambda/2) SUM_{r=0}^\infty ((\lambda/2)^r / r!) dchisq(x, df + 2r)}
```

for $x \geq 0$.

Value

If asList is FALSE (default), return a vector of random variates.

Otherwise, return a list with components suitable for visualizing inversion, specifically:

u	A vector of generated U(0,1) variates
x	A vector of chi-squared random variates
quantile	Parameterized quantile function
text	Parameterized title of distribution

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

rstream, set.seed, stats::runif

stats::rchisq

Examples

```
set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qchisq
vchisq(3, df = 3, ncp = 2)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qchisq
vchisq(3, 3, stream = 1)
vchisq(3, 3, stream = 2)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qchisq
vchisq(1, 3, stream = 1)
vchisq(1, 3, stream = 2)
vchisq(1, 3, stream = 1)
vchisq(1, 3, stream = 2)
vchisq(1, 3, stream = 1)
vchisq(1, 3, stream = 2)

set.seed(8675309)
variates <- vchisq(100, 3, stream = 1)
set.seed(8675309)
variates <- vchisq(100, 3, stream = 1, antithetic = TRUE)
```

---

vexp *Variate Generation for Exponential Distribution*

Description

Variate Generation for Exponential Distribution

Usage

```
vexp(n, rate = 1, stream = NULL, antithetic = FALSE, asList = FALSE)
```

Arguments

n	number of observations
rate	Rate of distribution (default 1)
stream	if NULL (default), uses stats::runif to generate uniform variates to invert via stats::qexp; otherwise, an integer in 1:25 indicates the rstream stream from which to generate uniform variates to invert via stats::qexp;
antithetic	if FALSE (default), inverts u = uniform(0,1) variate(s) generated via either stats::runif or rstream::rstream.sample; otherwise, uses $1 - u$
asList	if FALSE (default), output only the generated random variates; otherwise, return a list with components suitable for visualizing inversion. See return for details

Details

Generates random variates from the exponential distribution.

Exponential variates are generated by inverting uniform(0,1) variates produced either by stats::runif (if stream is NULL) or by rstream::rstream.sample (if stream is not NULL). In either case, stats::qexp is used to invert the uniform(0,1) variate(s). In this way, using vexp provides a monotone and synchronized binomial variate generator, although not particularly fast.

The stream indicated must be an integer between 1 and 25 inclusive.

```
The exponential distribution with rate \eqn{\lambda} has density

    \deqn{f(x) = \lambda e^{-\lambda x}}{
              f(x) = \lambda e^(-\lambda x)}

for \eqn{x \geq 0}.
```

Value

If asList is FALSE (default), return a vector of random variates.

Otherwise, return a list with components suitable for visualizing inversion, specifically:

u	A vector of generated U(0,1) variates
x	A vector of exponential random variates
quantile	Parameterized quantile function
text	Parameterized title of distribution

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

rstream, set.seed, stats::runif

stats::rexp

Examples

```
set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qexp
vexp(3, rate = 2)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qexp
vexp(3, 2, stream = 1)
vexp(3, 2, stream = 2)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qexp
vexp(1, 2, stream = 1)
vexp(1, 2, stream = 2)
vexp(1, 2, stream = 1)
vexp(1, 2, stream = 2)
vexp(1, 2, stream = 1)
vexp(1, 2, stream = 2)

set.seed(8675309)
variates <- vexp(100, 2, stream = 1)
set.seed(8675309)
variates <- vexp(100, 2, stream = 1, antithetic = TRUE)

set.seed(8675309)
# NOTE: Default functions for M/M/1 ssq(), ignoring fixed n
interarrivals <- vexp(1000, rate = 1,    stream = 1)
services      <- vexp(1000, rate = 10/9, stream = 2)
```

---

vfd *Variate Generation for FALSE Distribution*

Description

Variate Generation for FALSE Distribution

Usage

```
vfd(n, df1, df2, ncp = 0, stream = NULL, antithetic = FALSE, asList = FALSE)
```

Arguments

n	number of observations
df1	Degrees of freedom > 0
df2	Degrees of freedom > 0
ncp	Non-centrality parameter >= 0

`stream`	if `NULL` (default), uses `stats::runif` to generate uniform variates to invert via `stats::qf`; otherwise, an integer in 1:25 indicates the `rstream` stream from which to generate uniform variates to invert via `stats::qf`;
`antithetic`	if `FALSE` (default), inverts u = uniform(0,1) variate(s) generated via either `stats::runif` or `rstream::rstream.sample`; otherwise, uses $1 - u$
`asList`	if `FALSE` (default), output only the generated random variates; otherwise, return a list with components suitable for visualizing inversion. See return for details

Details

Generates random variates from the FALSE distribution.

FALSE variates are generated by inverting uniform(0,1) variates produced either by `stats::runif` (if `stream` is `NULL`) or by `rstream::rstream.sample` (if `stream` is not `NULL`). In either case, `stats::qf` is used to invert the uniform(0,1) variate(s). In this way, using `vfd` provides a monotone and synchronized binomial variate generator, although not particularly fast.

The stream indicated must be an integer between 1 and 25 inclusive.

The F distribution with $\texttt{df1} = n_1$ and $\texttt{df2} = n_2$ degrees of freedom has density

$$f(x) = \frac{\Gamma(n_1/2 + n_2/2)}{\Gamma(n_1/2)\,\Gamma(n_2/2)} \left(\frac{n_1}{n_2}\right)^{n_1/2} x^{n_1/2-1} \left(1 + \frac{n_1 x}{n_2}\right)^{-(n_1+n_2)/2}$$

for $x > 0$.

Value

If `asList` is FALSE (default), return a vector of random variates.

Otherwise, return a list with components suitable for visualizing inversion, specifically:

`u`	A vector of generated U(0,1) variates
`x`	A vector of FALSE random variates
`quantile`	Parameterized quantile function
`text`	Parameterized title of distribution

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

`rstream`, `set.seed`, `stats::runif`

`stats::rf`

Examples

```
set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qf
vfd(3, df1 = 1, df2 = 2, ncp = 10)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qf
vfd(3, 5, 5, stream = 1)
vfd(3, 5, 5, stream = 2)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qf
vfd(1, 5, 5, stream = 1)
vfd(1, 5, 5, stream = 2)
vfd(1, 5, 5, stream = 1)
vfd(1, 5, 5, stream = 2)
vfd(1, 5, 5, stream = 1)
vfd(1, 5, 5, stream = 2)

set.seed(8675309)
variates <- vfd(100, 5, 5, stream = 1)
set.seed(8675309)
variates <- vfd(100, 5, 5, stream = 1, antithetic = TRUE)
```

---

vgamma *Variate Generation for Gamma Distribution*

Description

Variate Generation for Gamma Distribution

Usage

```
vgamma(
  n,
  shape,
  rate = 1,
  scale = 1/rate,
  stream = NULL,
  antithetic = FALSE,
  asList = FALSE
)
```

Arguments

n	number of observations
shape	Shape parameter

rate	Alternate parameterization for scale
scale	Scale parameter
stream	if NULL (default), uses stats::runif to generate uniform variates to invert via stats::qgamma; otherwise, an integer in 1:25 indicates the rstream stream from which to generate uniform variates to invert via stats::qgamma;
antithetic	if FALSE (default), inverts u = uniform(0,1) variate(s) generated via either stats::runif or rstream::rstream.sample; otherwise, uses $1 - u$
asList	if FALSE (default), output only the generated random variates; otherwise, return a list with components suitable for visualizing inversion. See return for details

Details

Generates random variates from the gamma distribution.

Gamma variates are generated by inverting uniform(0,1) variates produced either by stats::runif (if stream is NULL) or by rstream::rstream.sample (if stream is not NULL). In either case, stats::qgamma is used to invert the uniform(0,1) variate(s). In this way, using vgamma provides a monotone and synchronized binomial variate generator, although not particularly fast.

The stream indicated must be an integer between 1 and 25 inclusive.

```
The gamma distribution with parameters \code{shape} = \eqn{a} and
\code{scale} = \eqn{s} has density

     \deqn{f(x) = \frac{1}{s^a\, \Gamma(a)} x^{a-1} e^{-x/s}}{
               f(x) = 1/(s^a Gamma(a)) x^(a-1) e^(-x/s)}

for \eqn{x \ge 0}, \eqn{a > 0}, and \eqn{s > 0}.
(Here \eqn{\Gamma(a)}{Gamma(a)} is the function implemented by
R's \code{\link[base:Special]{gamma}()} and defined in its help.)

The population mean and variance are \eqn{E(X) = as}
and \eqn{Var(X) = as^2}.
```

Value

If asList is FALSE (default), return a vector of random variates.

Otherwise, return a list with components suitable for visualizing inversion, specifically:

u	A vector of generated U(0,1) variates
x	A vector of gamma random variates
quantile	Parameterized quantile function
text	Parameterized title of distribution

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

rstream, set.seed, stats::runif

stats::rgamma

Examples

```
set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qgamma
vgamma(3, shape = 2, rate = 1)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qgamma
vgamma(3, 2, scale = 1, stream = 1)
vgamma(3, 2, scale = 1, stream = 2)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qgamma
vgamma(1, 2, scale = 1, stream = 1)
vgamma(1, 2, scale = 1, stream = 2)
vgamma(1, 2, scale = 1, stream = 1)
vgamma(1, 2, scale = 1, stream = 2)
vgamma(1, 2, scale = 1, stream = 1)
vgamma(1, 2, scale = 1, stream = 2)

set.seed(8675309)
variates <- vgamma(100, 2, scale = 1, stream = 1)
set.seed(8675309)
variates <- vgamma(100, 2, scale = 1, stream = 1, antithetic = TRUE)
```

---

vgeom *Variate Generation for Geometric Distribution*

Description

Variate Generation for Geometric Distribution

Usage

```
vgeom(n, prob, stream = NULL, antithetic = FALSE, asList = FALSE)
```

Arguments

n	number of observations
prob	Probability of success in each trial ($0 <$ prob ≤ 1)
stream	if NULL (default), uses stats::runif to generate uniform variates to invert via stats::qgeom; otherwise, an integer in 1:25 indicates the rstream stream from which to generate uniform variates to invert via stats::qgeom;

`antithetic` if `FALSE` (default), inverts u = uniform(0,1) variate(s) generated via either `stats::runif` or `rstream::rstream.sample`; otherwise, uses $1 - u$

`asList` if `FALSE` (default), output only the generated random variates; otherwise, return a list with components suitable for visualizing inversion. See return for details

Details

Generates random variates from the geometric distribution.

Geometric variates are generated by inverting uniform(0,1) variates produced either by `stats::runif` (if `stream` is `NULL`) or by `rstream::rstream.sample` (if `stream` is not `NULL`). In either case, `stats::qgeom` is used to invert the uniform(0,1) variate(s). In this way, using `vgeom` provides a monotone and synchronized binomial variate generator, although not particularly fast.

The stream indicated must be an integer between 1 and 25 inclusive.

The geometric distribution with parameter `prob` = p has density

$$p(x) = p(1 - p)^x$$

for $x = 0, 1, 2, \ldots$, where $0 < p \le 1$.

Value

If `asList` is FALSE (default), return a vector of random variates.

Otherwise, return a list with components suitable for visualizing inversion, specifically:

`u`	A vector of generated U(0,1) variates
`x`	A vector of geometric random variates
`quantile`	Parameterized quantile function
`text`	Parameterized title of distribution

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

`rstream`, `set.seed`, `stats::runif`

`stats::rgeom`

Examples

```
set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qgeom
vgeom(3, prob = 0.3)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qgeom
```

```
vgeom(3, 0.3, stream = 1)
vgeom(3, 0.3, stream = 2)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qgeom
vgeom(1, 0.3, stream = 1)
vgeom(1, 0.3, stream = 2)
vgeom(1, 0.3, stream = 1)
vgeom(1, 0.3, stream = 2)
vgeom(1, 0.3, stream = 1)
vgeom(1, 0.3, stream = 2)

set.seed(8675309)
variates <- vgeom(100, 0.3, stream = 1)
set.seed(8675309)
variates <- vgeom(100, 0.3, stream = 1, antithetic = TRUE)
```

---

vlnorm *Variate Generation for Log-Normal Distribution*

Description

Variate Generation for Log-Normal Distribution

Usage

```
vlnorm(
  n,
  meanlog = 0,
  sdlog = 1,
  stream = NULL,
  antithetic = FALSE,
  asList = FALSE
)
```

Arguments

n	number of observations
meanlog	Mean of distribution on log scale (default 0)
sdlog	Standard deviation of distribution on log scale (default 1)
stream	if NULL (default), uses stats::runif to generate uniform variates to invert via stats::qlnorm; otherwise, an integer in 1:25 indicates the rstream stream from which to generate uniform variates to invert via stats::qlnorm;
antithetic	if FALSE (default), inverts u = uniform(0,1) variate(s) generated via either stats::runif or rstream::rstream.sample; otherwise, uses $1 - u$
asList	if FALSE (default), output only the generated random variates; otherwise, return a list with components suitable for visualizing inversion. See return for details

Details

Generates random variates from the log-normal distribution.

Log-Normal variates are generated by inverting uniform(0,1) variates produced either by `stats::runif` (if `stream` is `NULL`) or by `rstream::rstream.sample` (if `stream` is not `NULL`). In either case, `stats::qlnorm` is used to invert the uniform(0,1) variate(s). In this way, using `vlnorm` provides a monotone and synchronized binomial variate generator, although not particularly fast.

The stream indicated must be an integer between 1 and 25 inclusive.

The log-normal distribution has density

```
\deqn{f(x) = \frac{1}{\sqrt{2 \pi} \sigma x}
                   e^{-(\log{x} - \mu)^2 / (2 \sigma^2)} }{
           f(x) = 1/(\sqrt(2 \pi) \sigma x) e^-((log x - \mu)^2 / (2 \sigma^2))}
```

where μ and σ are the mean and standard deviation of the logarithm.

The mean is $E(X) = \exp(\mu + 1/2\sigma^2)$, the median is $med(X) = \exp(\mu)$, and the variance is $Var(X) = \exp(2 \times \mu + \sigma^2) \times (\exp(\sigma^2) - 1)$ and hence the coefficient of variation is $sqrt(\exp(\sigma^2) - 1)$ which is approximately σ when small (e.g., $\sigma < 1/2$).

Value

If `asList` is FALSE (default), return a vector of random variates.

Otherwise, return a list with components suitable for visualizing inversion, specifically:

`u`	A vector of generated U(0,1) variates
`x`	A vector of log-normal random variates
`quantile`	Parameterized quantile function
`text`	Parameterized title of distribution

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

`rstream`, `set.seed`, `stats::runif`

`stats::rlnorm`

Examples

```
set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qlnorm
vlnorm(3, meanlog = 5, sdlog = 0.5)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qlnorm
```

```
vlnorm(3, 8, 2, stream = 1)
vlnorm(3, 8, 2, stream = 2)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qlnorm
vlnorm(1, 8, 2, stream = 1)
vlnorm(1, 8, 2, stream = 2)
vlnorm(1, 8, 2, stream = 1)
vlnorm(1, 8, 2, stream = 2)
vlnorm(1, 8, 2, stream = 1)
vlnorm(1, 8, 2, stream = 2)

set.seed(8675309)
variates <- vlnorm(100, 8, 2, stream = 1)
set.seed(8675309)
variates <- vlnorm(100, 8, 2, stream = 1, antithetic = TRUE)
```

---

vlogis *Variate Generation for Logistic Distribution*

Description

Variate Generation for Logistic Distribution

Usage

```
vlogis(
  n,
  location = 0,
  scale = 1,
  stream = NULL,
  antithetic = FALSE,
  asList = FALSE
)
```

Arguments

`n`	number of observations
`location`	Location parameter
`scale`	Scale parameter (default 1)
`stream`	if `NULL` (default), uses `stats::runif` to generate uniform variates to invert via `stats::qlogis`; otherwise, an integer in 1:25 indicates the `rstream` stream from which to generate uniform variates to invert via `stats::qlogis`;
`antithetic`	if `FALSE` (default), inverts u = uniform(0,1) variate(s) generated via either `stats::runif` or `rstream::rstream.sample`; otherwise, uses $1 - u$
`asList`	if `FALSE` (default), output only the generated random variates; otherwise, return a list with components suitable for visualizing inversion. See return for details

Details

Generates random variates from the logistic distribution.

Logistic variates are generated by inverting uniform(0,1) variates produced either by `stats::runif` (if `stream` is `NULL`) or by `rstream::rstream.sample` (if `stream` is not `NULL`). In either case, `stats::qlogis` is used to invert the uniform(0,1) variate(s). In this way, using `vlogis` provides a monotone and synchronized binomial variate generator, although not particularly fast.

The stream indicated must be an integer between 1 and 25 inclusive.

The logistic distribution with `location` $= \mu$ and `scale` $= \sigma$ has distribution function

$$F(x) = \frac{1}{1 + e^{-(x-\mu)/\sigma}}$$

and density

$$f(x) = \frac{1}{\sigma} \frac{e^{(x-\mu)/\sigma}}{(1 + e^{(x-\mu)/\sigma})^2}$$

It is a long-tailed distribution with mean μ and variance $\pi^2/3\sigma^2$.

Value

If `asList` is FALSE (default), return a vector of random variates.

Otherwise, return a list with components suitable for visualizing inversion, specifically:

`u`	A vector of generated U(0,1) variates
`x`	A vector of logistic random variates
`quantile`	Parameterized quantile function
`text`	Parameterized title of distribution

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

`rstream`, `set.seed`, `stats::runif`

`stats::rlogis`

Examples

```
set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qlogis
vlogis(3, location = 5, scale = 0.5)

set.seed(8675309)
```

```
# NOTE: following inverts rstream::rstream.sample using stats::qlogis
vlogis(3, 5, 1.5, stream = 1)
vlogis(3, 5, 1.5, stream = 2)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qlogis
vlogis(1, 5, 1.5, stream = 1)
vlogis(1, 5, 1.5, stream = 2)
vlogis(1, 5, 1.5, stream = 1)
vlogis(1, 5, 1.5, stream = 2)
vlogis(1, 5, 1.5, stream = 1)
vlogis(1, 5, 1.5, stream = 2)

set.seed(8675309)
variates <- vlogis(100, 5, 1.5, stream = 1)
set.seed(8675309)
variates <- vlogis(100, 5, 1.5, stream = 1, antithetic = TRUE)
```

---

vnbinom *Variate Generation for Negative Binomial Distribution*

Description

Variate Generation for Negative Binomial Distribution

Usage

```
vnbinom(n, size, prob, mu, stream = NULL, antithetic = FALSE, asList = FALSE)
```

Arguments

n	number of observations
size	target for number of successful trials, or dispersion parameter (the shape parameter of the gamma mixing distribution). Must be strictly positive, need not be integer.
prob	Probability of success in each trial; '0 < prob <= 1'
mu	alternative parameterization via mean
stream	if NULL (default), uses stats::runif to generate uniform variates to invert via stats::qnbinom; otherwise, an integer in 1:25 indicates the rstream stream from which to generate uniform variates to invert via stats::qnbinom;
antithetic	if FALSE (default), inverts u = uniform(0,1) variate(s) generated via either stats::runif or rstream::rstream.sample; otherwise, uses $1 - u$
asList	if FALSE (default), output only the generated random variates; otherwise, return a list with components suitable for visualizing inversion. See return for details

Details

Generates random variates from the negative binomial distribution.

Negative Binomial variates are generated by inverting uniform(0,1) variates produced either by `stats::runif` (if `stream` is `NULL`) or by `rstream::rstream.sample` (if `stream` is not `NULL`). In either case, `stats::qnbinom` is used to invert the uniform(0,1) variate(s). In this way, using `vnbinom` provides a monotone and synchronized binomial variate generator, although not particularly fast.

The stream indicated must be an integer between 1 and 25 inclusive.

The negative binomial distribution with `size` $= n$ and `prob` $= p$ has density

```
\deqn{p(x) = \frac{\Gamma(x+n)}{\Gamma(n) \ x!} p^n (1-p)^x}{
           p(x) = Gamma(x+n)/(Gamma(n) x!) p^n (1-p)^x}
```

for $x = 0, 1, 2, \ldots, n > 0$ and $0 < p \leq 1$. This represents the number of failures which occur in a sequence of Bernoulli trials before a target number of successes is reached.

The mean is $\mu = n(1-p)/p$ and variance $n(1-p)/p^2$

Value

If `asList` is FALSE (default), return a vector of random variates.

Otherwise, return a list with components suitable for visualizing inversion, specifically:

`u`	A vector of generated U(0,1) variates
`x`	A vector of negative binomial random variates
`quantile`	Parameterized quantile function
`text`	Parameterized title of distribution

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

`rstream`, `set.seed`, `stats::runif`

`stats::rnbinom`

Examples

```
set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qnbinom
vnbinom(3, size = 10, mu = 10)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qnbinom
vnbinom(3, 10, 0.25, stream = 1)
vnbinom(3, 10, 0.25, stream = 2)
```

```
set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qnbinom
vnbinom(1, 10, 0.25, stream = 1)
vnbinom(1, 10, 0.25, stream = 2)
vnbinom(1, 10, 0.25, stream = 1)
vnbinom(1, 10, 0.25, stream = 2)
vnbinom(1, 10, 0.25, stream = 1)
vnbinom(1, 10, 0.25, stream = 2)

set.seed(8675309)
variates <- vnbinom(100, 10, 0.25, stream = 1)
set.seed(8675309)
variates <- vnbinom(100, 10, 0.25, stream = 1, antithetic = TRUE)
```

---

vnorm *Variate Generation for Normal Distribution*

Description

Variate Generation for Normal Distribution

Usage

```
vnorm(n, mean = 0, sd = 1, stream = NULL, antithetic = FALSE, asList = FALSE)
```

Arguments

n	number of observations
mean	Mean of distribution (default 0)
sd	Standard deviation of distribution (default 1)
stream	if `NULL` (default), uses `stats::runif` to generate uniform variates to invert via `stats::qnorm`; otherwise, an integer in 1:25 indicates the `rstream` stream from which to generate uniform variates to invert via `stats::qnorm`;
antithetic	if `FALSE` (default), inverts u = uniform(0,1) variate(s) generated via either `stats::runif` or `rstream::rstream.sample`; otherwise, uses $1 - u$
asList	if `FALSE` (default), output only the generated random variates; otherwise, return a list with components suitable for visualizing inversion. See return for details

Details

Generates random variates from the normal distribution.

Normal variates are generated by inverting uniform(0,1) variates produced either by `stats::runif` (if `stream` is `NULL`) or by `rstream::rstream.sample` (if `stream` is not `NULL`). In either case, `stats::qnorm` is used to invert the uniform(0,1) variate(s). In this way, using `vnorm` provides a monotone and synchronized binomial variate generator, although not particularly fast.

The stream indicated must be an integer between 1 and 25 inclusive.

The normal distribution has density

```
\deqn{f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x - \mu)^2/(2 \sigma^2)}}{
          f(x) = 1/(\sqrt(2\pi)\sigma) e^(-(x - \mu)^2/(2 \sigma^2))}
```

for $-\infty < x < \infty$ and $\sigma > 0$, where μ is the mean of the distribution and σ the standard deviation.

Value

If `asList` is FALSE (default), return a vector of random variates.

Otherwise, return a list with components suitable for visualizing inversion, specifically:

`u`	A vector of generated U(0,1) variates
`x`	A vector of normal random variates
`quantile`	Parameterized quantile function
`text`	Parameterized title of distribution

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

`rstream`, `set.seed`, `stats::runif`

`stats::rnorm`

Examples

```
set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qnorm
vnorm(3, mean = 2, sd = 1)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qnorm
vnorm(3, 10, 2, stream = 1)
vnorm(3, 10, 2, stream = 2)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qnorm
vnorm(1, 10, 2, stream = 1)
vnorm(1, 10, 2, stream = 2)
vnorm(1, 10, 2, stream = 1)
vnorm(1, 10, 2, stream = 2)
vnorm(1, 10, 2, stream = 1)
vnorm(1, 10, 2, stream = 2)
```

```
set.seed(8675309)
variates <- vnorm(100, 10, 2, stream = 1)
set.seed(8675309)
variates <- vnorm(100, 10, 2, stream = 1, antithetic = TRUE)
```

---

vpois *Variate Generation for Poisson Distribution*

Description

Variate Generation for Poisson Distribution

Usage

```
vpois(n, lambda, stream = NULL, antithetic = FALSE, asList = FALSE)
```

Arguments

n	number of observations
lambda	Rate of distribution
stream	if NULL (default), uses stats::runif to generate uniform variates to invert via stats::qpois; otherwise, an integer in 1:25 indicates the rstream stream from which to generate uniform variates to invert via stats::qpois;
antithetic	if FALSE (default), inverts u = uniform(0,1) variate(s) generated via either stats::runif or rstream::rstream.sample; otherwise, uses $1 - u$
asList	if FALSE (default), output only the generated random variates; otherwise, return a list with components suitable for visualizing inversion. See return for details

Details

Generates random variates from the Poisson distribution.

Poisson variates are generated by inverting uniform(0,1) variates produced either by stats::runif (if stream is NULL) or by rstream::rstream.sample (if stream is not NULL). In either case, stats::qpois is used to invert the uniform(0,1) variate(s). In this way, using vpois provides a monotone and synchronized binomial variate generator, although not particularly fast.

The stream indicated must be an integer between 1 and 25 inclusive.

The Poisson distribution has density

$$p(x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

for $x = 0, 1, 2, \ldots$. The mean and variance are $E(X) = Var(X) = \lambda$

Value

If `asList` is FALSE (default), return a vector of random variates.

Otherwise, return a list with components suitable for visualizing inversion, specifically:

`u`	A vector of generated U(0,1) variates
`x`	A vector of Poisson random variates
`quantile`	Parameterized quantile function
`text`	Parameterized title of distribution

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

`rstream`, `set.seed`, `stats::runif`

`stats::rpois`

Examples

```
set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qpois
vpois(3, lambda = 5)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qpois
vpois(3, 3, stream = 1)
vpois(3, 3, stream = 2)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qpois
vpois(1, 3, stream = 1)
vpois(1, 3, stream = 2)
vpois(1, 3, stream = 1)
vpois(1, 3, stream = 2)
vpois(1, 3, stream = 1)
vpois(1, 3, stream = 2)

set.seed(8675309)
variates <- vpois(100, 3, stream = 1)
set.seed(8675309)
variates <- vpois(100, 3, stream = 1, antithetic = TRUE)
```

vt — *Variate Generation for Student T Distribution*

Description

Variate Generation for Student T Distribution

Usage

```
vt(n, df, ncp = 0, stream = NULL, antithetic = FALSE, asList = FALSE)
```

Arguments

`n`	number of observations
`df`	Degrees of freedom > 0
`ncp`	Non-centrality parameter delta (default NULL)
`stream`	if `NULL` (default), uses `stats::runif` to generate uniform variates to invert via `stats::qt`; otherwise, an integer in 1:25 indicates the `rstream` stream from which to generate uniform variates to invert via `stats::qt`;
`antithetic`	if `FALSE` (default), inverts u = uniform(0,1) variate(s) generated via either `stats::runif` or `rstream::rstream.sample`; otherwise, uses $1 - u$
`asList`	if `FALSE` (default), output only the generated random variates; otherwise, return a list with components suitable for visualizing inversion. See return for details

Details

Generates random variates from the Student t distribution.

Student T variates are generated by inverting uniform(0,1) variates produced either by `stats::runif` (if `stream` is `NULL`) or by `rstream::rstream.sample` (if `stream` is not `NULL`). In either case, `stats::qt` is used to invert the uniform(0,1) variate(s). In this way, using `vt` provides a monotone and synchronized binomial variate generator, although not particularly fast.

The stream indicated must be an integer between 1 and 25 inclusive.

The t-distribution with `df` $= v$ degrees of freedom has density

$$f(x) = \frac{\Gamma((v+1)/2)}{\sqrt{v\pi}\,\Gamma(v/2)}\,(1 + x^2/v)^{-(v+1)/2}$$

for all real x. It has mean 0 (for $v > 1$) and variance $v/(v-2)$ (for $v > 2$).

The general non-central t with parameters (ν, δ) = (`df`, `ncp`) is defined as the distribution of $T_\nu(\delta) := (U + \delta)\ /\ \sqrt{(V/\nu)}$ where U and V are independent random variables, $U \sim \mathcal{N}(0, 1)$ and $V \sim \chi^2(\nu)$.

Value

If asList is FALSE (default), return a vector of random variates.

Otherwise, return a list with components suitable for visualizing inversion, specifically:

u	A vector of generated U(0,1) variates
x	A vector of Student t random variates
quantile	Parameterized quantile function
text	Parameterized title of distribution

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

rstream, set.seed, stats::runif

stats::rt

Examples

```
set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qt
vt(3, df = 3, ncp = 2)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qt
vt(3, 2, stream = 1)
vt(3, 2, stream = 2)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qt
vt(1, 2, stream = 1)
vt(1, 2, stream = 2)
vt(1, 2, stream = 1)
vt(1, 2, stream = 2)
vt(1, 2, stream = 1)
vt(1, 2, stream = 2)

set.seed(8675309)
variates <- vt(100, 2, stream = 1)
set.seed(8675309)
variates <- vt(100, 2, stream = 1, antithetic = TRUE)
```

vunif *Variate Generation for Uniform Distribution*

Description

Variate Generation for Uniform Distribution

Usage

```
vunif(n, min = 0, max = 1, stream = NULL, antithetic = FALSE, asList = FALSE)
```

Arguments

n	number of observations
min	lower limit of distribution (default 0)
max	upper limit of distribution (default 1)
stream	if NULL (default), uses stats::runif to generate uniform variates; otherwise, an integer in 1:25 indicates the rstream stream from which to generate uniform variates;
antithetic	if FALSE (default), inverts u = uniform(0,1) variate(s) generated via either stats::runif or rstream::rstream.sample; otherwise, uses $1 - u$
asList	if FALSE (default), output only the generated random variates; otherwise, return a list with components suitable for visualizing inversion. See return for details

Details

Generates random variates from the uniform distribution.

Uniform variates are generated by inverting uniform(0,1) variates produced either by stats::runif (if stream is NULL) or by rstream::rstream.sample (if stream is not NULL). In either case, stats::qunif is used to invert the uniform(0,1) variate(s). In this way, using vunif provides a monotone and synchronized binomial variate generator, although not particularly fast.

The stream indicated must be an integer between 1 and 25 inclusive.

The uniform distribution has density

```
\deqn{f(x) = \frac{1}{max-min}}{
          f(x) = 1/(max-min)}
```

for $min \leq x \leq max$.

Value

If `asList` is FALSE (default), return a vector of random variates.

Otherwise, return a list with components suitable for visualizing inversion, specifically:

`u`	A vector of generated U(0,1) variates
`x`	A vector of uniform random variates
`quantile`	Parameterized quantile function
`text`	Parameterized title of distribution

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

`rstream`, `set.seed`, `stats::runif`

`stats::runif`

Examples

```
set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qunif
vunif(3, min = -2, max = 2)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qunif
vunif(3, 0, 10, stream = 1)
vunif(3, 0, 10, stream = 2)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qunif
vunif(1, 0, 10, stream = 1)
vunif(1, 0, 10, stream = 2)
vunif(1, 0, 10, stream = 1)
vunif(1, 0, 10, stream = 2)
vunif(1, 0, 10, stream = 1)
vunif(1, 0, 10, stream = 2)

set.seed(8675309)
variates <- vunif(100, 0, 10, stream = 1)
set.seed(8675309)
variates <- vunif(100, 0, 10, stream = 1, antithetic = TRUE)
```

vweibull *Variate Generation for Weibull Distribution*

Description

Variate Generation for Weibull Distribution

Usage

```
vweibull(
  n,
  shape,
  scale = 1,
  stream = NULL,
  antithetic = FALSE,
  asList = FALSE
)
```

Arguments

n	number of observations
shape	Shape parameter
scale	Scale parameter (default 1)
stream	if `NULL` (default), uses `stats::runif` to generate uniform variates to invert via `stats::qweibull`; otherwise, an integer in 1:25 indicates the `rstream` stream from which to generate uniform variates to invert via `stats::qweibull`;
antithetic	if `FALSE` (default), inverts u = uniform(0,1) variate(s) generated via either `stats::runif` or `rstream::rstream.sample`; otherwise, uses $1 - u$
asList	if `FALSE` (default), output only the generated random variates; otherwise, return a list with components suitable for visualizing inversion. See return for details

Details

Generates random variates from the Weibull distribution.

Weibull variates are generated by inverting uniform(0,1) variates produced either by `stats::runif` (if `stream` is `NULL`) or by `rstream::rstream.sample` (if `stream` is not `NULL`). In either case, `stats::qweibull` is used to invert the uniform(0,1) variate(s). In this way, using `vweibull` provides a monotone and synchronized binomial variate generator, although not particularly fast.

The stream indicated must be an integer between 1 and 25 inclusive.

The Weibull distribution with parameters `shape` = a and `scale` = b has density

```
\deqn{f(x) = \frac{a}{b} \left(\frac{x}{b}\right)^{a-1} e^{-(x/b)^a}}{
          f(x) = (a/b) (x/b)^(a-1) exp(-(x/b)^a)}
```

for $x \geq 0$, $a > 0$, and $b > 0$.

Value

If asList is FALSE (default), return a vector of random variates.

Otherwise, return a list with components suitable for visualizing inversion, specifically:

u	A vector of generated U(0,1) variates
x	A vector of Weibull random variates
quantile	Parameterized quantile function
text	Parameterized title of distribution

Author(s)

Barry Lawson (<blawson@bates.edu>),
Larry Leemis (<leemis@math.wm.edu>),
Vadim Kudlay (<vkudlay@nvidia.com>)

See Also

rstream, set.seed, stats::runif

stats::rweibull

Examples

```
set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qweibull
vweibull(3, shape = 2, scale = 1)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qweibull
vweibull(3, 2, 1, stream = 1)
vweibull(3, 2, 1, stream = 2)

set.seed(8675309)
# NOTE: following inverts rstream::rstream.sample using stats::qweibull
vweibull(1, 2, 1, stream = 1)
vweibull(1, 2, 1, stream = 2)
vweibull(1, 2, 1, stream = 1)
vweibull(1, 2, 1, stream = 2)
vweibull(1, 2, 1, stream = 1)
vweibull(1, 2, 1, stream = 2)

set.seed(8675309)
variates <- vweibull(100, 2, 1, stream = 1)
set.seed(8675309)
variates <- vweibull(100, 2, 1, stream = 1, antithetic = TRUE)
```

Index