Package 'simEd'

November 27, 2017

Version 1.0.3

Title Simulation Education

Author Barry Lawson, Larry Leemis

Maintainer Barry Lawson <blawson@richmond.edu>

Imports graphics, grDevices, methods, stats, utils

Depends rstream

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 two queueing data sets (one fabricated, one real-world) to facilitate input modeling.

License GPL (>= 2)

LazyData true

RoxygenNote 6.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2017-11-27 04:05:15 UTC

R topics documented:

`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 two queueing data sets (one fabricated, one real-world) to facilitate input modeling.

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

- discrete: `vbinom`, `vgeom`
- continuous: `vexp`, `vgamma`, `vnorm`, `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 two discrete and five continuous distributions:

- discrete: ibinom, igeom
- continuous: iexp, igamma, inorm, iunif, iweibull

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 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.

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, Larry Leemis

Maintainer: Barry Lawson <blawson@richmond.edu>

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@richmond.edu>), Larry Leemis (<leemis@math.wm.edu>)

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
probs <- 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@richmond.edu>), Larry Leemis (<leemis@math.wm.edu>)

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
probs <- galileo(10000)

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

---

ibinom *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 the empirical probability mass function 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         = FALSE,
       showECDF        = FALSE,
       show            = NULL,
       plotDelay       = 0,
       maxPlotTime     = 30,
       showTitle       = TRUE,
       respectLayout   = TRUE)
```

Arguments

u	vector of uniform(0,1) random numbers
size	integer number of trials (zero or more)
prob	probability of success on each trial
minPlotQuantile	controls the minimum quantile to appear in the plots
maxPlotQuantile	controls the maximum quantile to appear in the plots
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 FALSE (default), pmf plot is suppressed, otherwise pmf plot appears
showECDF	logical; if FALSE (default), ecdf plot is suppressed, otherwise ecdf plot appears
show	shortcut way of denoting plots to display; either a binary vector of length three or an integer in [0,7] (see "Details" below)
plotDelay	delay, in seconds, between generation of the random variates
maxPlotTime	maximum time, in seconds, to plot all of the random variates
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 (see "Details" below)

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 empirical pmf 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. All aspects of the population distribution are output in black.

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

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

for $x = 0, \ldots, n$. Note that binomial coefficients can be computed by choose in R.

The population mean and variance are $E(X) = np$ and $Var(X) = np(1-p)$.

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 red dots. A horizontal, dashed, red line extends from the red dot to the population cdf. At the intersection, a vertical, dashed red line extends downward to the horizontal axis, where a second red 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 Unix's 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.

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 R default values 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.

The `plotDelay` and `maxPlotTime` arguments can be used to slow down the variate generation for classroom explanation.

In the plot associated with the pmf, the maximum plotting height is associated with 125% of the maximum height of pmf. Any histogram cell that extends above this limit will have three black dots appearing above it.

Value

A vector of binomial random variates.

Author(s)

Barry Lawson (<blawson@richmond.edu>), Larry Leemis (<leemis@math.wm.edu>)

Examples

```
ibinom(0.7, 7, 0.4)

par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and PMF plots
set.seed(8675309)
ibinom(runif(10), 7, 0.4, showPMF = TRUE)

par(mfrow = c(1,1)) # 1 row, 1 col for PMF only plot
set.seed(8675309)
ibinom(runif(10), 7, 0.4, showPMF = TRUE, showCDF = FALSE)

par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and PMF plots
ibinom(runif(120), 200, 0.5, showPMF = TRUE, minPlotQuantile = 0.02, maxPlotQuantile = 0.98)

# overlay visual exploration of ks.test results
par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and ECDF plots
set.seed(54321)
vals <- ibinom(runif(10), 10, 0.3, showECDF = TRUE)
D <- as.numeric(ks.test(vals, "pbinom", 10, 0.3)$statistic)
x <- 2.5
y <- pbinom(x, 10, 0.3)
segments(x, y, x, y + D, col = "darkgreen", lwd = 3)

# plot the PMF and CDF without any variates
par(mfrow = c(2,1)) # 2 rows, 1 col for PMF and CDF plots
ibinom(NULL, 10, 0.3)

# plot CDF with inversion and PMF using show
par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and PMF plots
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
par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and ECDF plots
ibinom(runif(10), 10, 0.3, show = c(1,0,1))
ibinom(runif(10), 10, 0.3, show = 5)

# plot CDF with inversion, PMF, and ECDF using show
par(mfrow = c(3,1)) # 3 rows, 1 col for CDF, PMF, and ECDF plots
ibinom(runif(10), 10, 0.3, show = c(1,1,1))
ibinom(runif(10), 10, 0.3, show = 7)

# plot three different CDF+PMF+ECDF vertical displays
par(mfcol = c(3,3))  # 3 rows, 3 cols, filling columns before rows
set.seed(8675309)
ibinom(runif(20), size = 10, prob = 0.3, show = 7)
ibinom(runif(20), size = 10, prob = 0.3, show = 7)
ibinom(runif(20), size = 10, prob = 0.3, show = 7)
```

```
# plot three different CDF+PMF+ECDF horizontal displays, with title only
# on the first display
par(mfrow = c(3,3))  # 3 rows, 3 cols, filling rows before columns
set.seed(8675309)
ibinom(runif(20), size = 10, prob = 0.3, show = 7)
ibinom(runif(20), size = 10, prob = 0.3, show = 7, showTitle = FALSE)
ibinom(runif(20), size = 10, prob = 0.3, show = 7, showTitle = FALSE)

# exhibit use of the respectLayout = FALSE option
par(mfrow = c(3,3))  # this will be ignored below since respectLayout = FALSE
set.seed(8675309)
ibinom(runif(20), size = 10, prob = 0.3, show = 7, respectLayout = FALSE)
par("mfrow")  # will have been reset c(1,1)
par("mar")    # will have been reset to c(5.1, 4.1, 4.1, 2.1)
```

`iexp` *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 = 0.95,
       plot            = TRUE,
       showCDF         = TRUE,
       showPDF         = FALSE,
       showECDF        = FALSE,
       show            = NULL,
       plotDelay       = 0,
       maxPlotTime     = 30,
       showTitle       = TRUE,
       respectLayout   = TRUE)
```

Arguments

`u`	vector of uniform(0,1) random numbers, or NULL
`rate`	rate parameter; must be positive
`minPlotQuantile`	controls the minimum quantile to appear in the plots

maxPlotQuantile	controls the maximum quantile to appear in the plots
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 FALSE (default), pdf plot is suppressed, otherwise pdf plot appears
showECDF	logical; if FALSE (default), ecdf plot is suppressed, otherwise ecdf plot appears
show	shortcut way of denoting plots to display; either a binary vector of length three or an integer in [0,7] (see "Details" below)
plotDelay	delay, in seconds, between generation of the random variates
maxPlotTime	maximum time, in seconds, to plot all of the random variates
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 (see "Details" below)

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. All aspects of the population distribution are output in black.

The exponential distribution with rate λ has density

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

for $x \geq 0$.

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 red dots. A horizontal, dashed, red line extends from the red dot to the population cdf. At the intersection, a vertical, dashed red line extends downward to the horizontal axis, where a second red 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 Unix's 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.

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 R default values 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.

The plotDelay and maxPlotTime arguments can be used to slow down the variate generation for classroom explanation.

In the plot associated with the pdf, the maximum plotting height is associated with 125% of the maximum height of pdf. Any histogram cell that extends above this limit will have three black dots appearing above it.

Value

A vector of exponential random variates

Author(s)

Barry Lawson (<blawson@richmond.edu>), Larry Leemis (<leemis@math.wm.edu>)

Examples

```
iexp(0.7, rate = 2)

par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and PDF plots
```

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

par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and ECDF plots
set.seed(8675309)
iexp(runif(10), rate = 2, showECDF = TRUE)

par(mfrow = c(3,1)) # 3 rows, 1 col for CDF, PDF, and ECDF plots
set.seed(8675309)
iexp(runif(10), rate = 2, showPDF = TRUE, showECDF = TRUE)

par(mfrow = c(1,1)) # 1 row, 1 col for PDF only plot
set.seed(8675309)
iexp(runif(10), rate = 2, showPDF = TRUE, showCDF = FALSE)

par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and PDF plots
iexp(runif(120), rate = 2, showPDF = TRUE, minPlotQuantile = 0.02, maxPlotQuantile = 0.98)

# overlay visual exploration of ks.test results
par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and ECDF plots
set.seed(54321)
vals <- iexp(runif(10), rate = 2, showECDF = TRUE)
D <- as.numeric(ks.test(vals, "pexp", 2)$statistic)
for (x in seq(0.05, 0.75, by = 0.05)) {
   y <- pexp(x, rate = 2)
   segments(x, y, x, y + D, col = "darkgreen", lwd = 2, xpd = NA)
}

# plot the PDF and CDF without any variates
par(mfrow = c(2,1)) # 2 rows, 1 col for PDF and CDF plots
iexp(NULL, rate = 2)

# plot CDF with inversion and PDF using show
par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and PDF plots
iexp(runif(10), rate = 2, show = c(1,1,0))
iexp(runif(10), rate = 2, show = 6)

# plot CDF with inversion and ECDF using show
par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and ECDF plots
iexp(runif(10), rate = 2, show = c(1,0,1))
iexp(runif(10), rate = 2, show = 5)

# plot CDF with inversion, PDF, and ECDF using show
par(mfrow = c(3,1)) # 3 rows, 1 col for CDF, PDF, and ECDF plots
iexp(runif(10), rate = 2, show = c(1,1,1))
iexp(runif(10), rate = 2, show = 7)

# plot three different CDF+PDF+ECDF vertical displays
par(mfcol = c(3,3))  # 3 rows, 3 cols, filling columns before rows
set.seed(8675309)
iexp(runif(20), rate = 2, show = 7)
iexp(runif(20), rate = 2, show = 7)
iexp(runif(20), rate = 2, show = 7)
```

```
# plot three different CDF+PDF+ECDF horizontal displays, with title only
# on the first display
par(mfrow = c(3,3))  # 3 rows, 3 cols, filling rows before columns
set.seed(8675309)
iexp(runif(20), rate = 2, show = 7)
iexp(runif(20), rate = 2, show = 7, showTitle = FALSE)
iexp(runif(20), rate = 2, show = 7, showTitle = FALSE)

# exhibit use of the respectLayout = FALSE option
par(mfrow = c(3,3))  # this will be ignored below since respectLayout = FALSE
set.seed(8675309)
iexp(runif(20), rate = 2, show = 7, respectLayout = FALSE)
par("mfrow")  # will have been reset c(1,1)
par("mar")    # will have been reset to c(5.1, 4.1, 4.1, 2.1)
```

---

igamma *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          = FALSE,
       showECDF         = FALSE,
       show             = NULL,
       plotDelay        = 0,
       maxPlotTime     = 30,
       showTitle       = TRUE,
       respectLayout   = TRUE)
```

Arguments

u	vector of uniform(0,1) random numbers, or NULL
shape, scale	shape and scale parameters (must be positive)
rate	an alternative way to specify the scale

`minPlotQuantile`
: controls the minimum quantile to appear in the plots

`maxPlotQuantile`
: controls the maximum quantile to appear in the plots

`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 `FALSE` (default), pdf plot is suppressed, otherwise pdf plot appears

`showECDF`
: logical; if `FALSE` (default), ecdf plot is suppressed, otherwise ecdf plot appears

`show`
: shortcut way of denoting plots to display; either a binary vector of length three or an integer in [0,7] (see "Details" below)

`plotDelay`
: delay, in seconds, between generation of the random variates

`maxPlotTime`
: maximum time, in seconds, to plot all of the random variates

`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 (see "Details" below)

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. All aspects of the population distribution are output in black.

The gamma distribution with parameters `shape` = a and `scale` = s has density

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

for $x \geq 0$, $a > 0$, and $s > 0$. (Here $\Gamma(a)$ is the function implemented by R's `gamma()` and defined in its help.)

The population mean and variance are $E(X) = as$ and $Var(X) = as^2$.

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 red dots. A horizontal, dashed, red line extends from the red dot to the population cdf. At the intersection, a vertical, dashed red line extends downward to the horizontal axis, where a second red 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 user may specify a scale parameter or a rate parameter, but not both.

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 Unix's 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.

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 R default values 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.

The plotDelay and maxPlotTime arguments can be used to slow down the variate generation for classroom explanation.

In the plot associated with the pdf, the maximum plotting height is associated with 125% of the maximum height of pdf. Any histogram cell that extends above this limit will have three black dots appearing above it.

Value

A vector of gamma random variates

Author(s)

Barry Lawson (<blawson@richmond.edu>), Larry Leemis (<leemis@math.wm.edu>)

Examples

```
igamma(0.7, 3, 2)

par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and PDF plots
set.seed(8675309)
igamma(runif(10), 3, 2, showPDF = TRUE)

par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and ECDF plots
set.seed(8675309)
igamma(runif(10), 3, 2, showECDF = TRUE)

par(mfrow = c(3,1)) # 3 rows, 1 col for CDF, PDF, and ECDF plots
set.seed(8675309)
igamma(runif(10), 3, 2, showPDF = TRUE, showECDF = TRUE)

par(mfrow = c(1,1)) # 1 row, 1 col for PDF only plot
set.seed(8675309)
igamma(runif(10), 3, 2, showPDF = TRUE, showCDF = FALSE)

par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and PDF plots
igamma(runif(120), 3, 2, showPDF = TRUE, minPlotQuantile = 0.02, maxPlotQuantile = 0.98)

# overlay visual exploration of ks.test results
par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and ECDF plots
set.seed(54321)
vals <- igamma(runif(10), 3, 2, showECDF = TRUE)
D <- as.numeric(ks.test(vals, "pgamma", 3, 2)$statistic)
for (x in seq(1.0, 2.0, by = 0.05)) {
   y <- pgamma(x, 3, 2)
   segments(x, y, x, y + D, col = "darkgreen", lwd = 2, xpd = NA)
}

# plot the PDF and CDF without any variates
par(mfrow = c(2,1)) # 2 rows, 1 col for PDF and CDF plots
igamma(NULL, 3, 2)

# plot CDF with inversion and PDF using show
par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and PDF plots
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
par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and ECDF plots
igamma(runif(10), 3, 2, show = c(1,0,1))
igamma(runif(10), 3, 2, show = 5)

# plot CDF with inversion, PDF, and ECDF using show
par(mfrow = c(3,1)) # 3 rows, 1 col for CDF, PDF, and ECDF plots
```

```
igamma(runif(10), 3, 2, show = c(1,1,1))
igamma(runif(10), 3, 2, show = 7)

# plot three different CDF+PDF+ECDF vertical displays
par(mfcol = c(3,3))  # 3 rows, 3 cols, filling columns before rows
set.seed(8675309)
igamma(runif(20), 3, 2, show = 7)
igamma(runif(20), 3, 2, show = 7)
igamma(runif(20), 3, 2, show = 7)

# plot three different CDF+PDF+ECDF horizontal displays, with title only
# on the first display
par(mfrow = c(3,3))  # 3 rows, 3 cols, filling rows before columns
set.seed(8675309)
igamma(runif(20), 3, 2, show = 7)
igamma(runif(20), 3, 2, show = 7, showTitle = FALSE)
igamma(runif(20), 3, 2, show = 7, showTitle = FALSE)

# exhibit use of the respectLayout = FALSE option
par(mfrow = c(3,3))  # this will be ignored below since respectLayout = FALSE
set.seed(8675309)
igamma(runif(20), 3, 2, show = 7, respectLayout = FALSE)
par("mfrow")  # will have been reset c(1,1)
par("mar")    # will have been reset to c(5.1, 4.1, 4.1, 2.1)
```

---

igeom *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 the empirical probability mass function 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         = FALSE,
       showECDF        = FALSE,
       show            = NULL,
       plotDelay       = 0,
       maxPlotTime     = 30,
       showTitle       = TRUE,
       respectLayout   = TRUE)
```

Arguments

u	vector of uniform(0,1) random numbers
prob	probability of success in each trial ($0 <$ prob ≤ 1)
minPlotQuantile	controls the minimum quantile to appear in the plots
maxPlotQuantile	controls the maximum quantile to appear in the plots
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 FALSE (default), pmf plot is suppressed, otherwise pmf plot appears
showECDF	logical; if FALSE (default), ecdf plot is suppressed, otherwise ecdf plot appears
show	shortcut way of denoting plots to display; either a binary vector of length three or an integer in [0,7] (see "Details" below)
plotDelay	delay, in seconds, between generation of the random variates
maxPlotTime	maximum time, in seconds, to plot all of the random variates
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 (see "Details" below)

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 empirical pmf 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. All aspects of the population distribution are output in black.

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 red dots. A horizontal, dashed, red line extends from the red dot to the population cdf. At the intersection, a vertical, dashed red line extends downward to the horizontal axis, where a second red 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 Unix's 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.

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 R default values 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.

The plotDelay and maxPlotTime arguments can be used to slow down the variate generation for classroom explanation.

In the plot associated with the pmf, the maximum plotting height is associated with 125% of the maximum height of pmf. Any histogram cell that extends above this limit will have three black dots appearing above it.

Value

A vector of geometric random variates.

Author(s)

Barry Lawson (<blawson@richmond.edu>), Larry Leemis (<leemis@math.wm.edu>)

Examples

```
igeom(0.7, 0.4)

par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and PMF plots
set.seed(8675309)
igeom(runif(10), 0.4, showPMF = TRUE)

par(mfrow = c(1,1)) # 1 row, 1 col for PMF only plot
set.seed(8675309)
igeom(runif(10), 0.4, showPMF = TRUE, showCDF = FALSE)

par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and PMF plots
igeom(runif(120), 0.4, showPMF = TRUE, minPlotQuantile = 0.02, maxPlotQuantile = 0.98)

# plot the PMF and CDF without any variates
par(mfrow = c(2,1)) # 2 rows, 1 col for PMF and CDF plots
igeom(NULL, 0.4)

# plot CDF with inversion and PMF using show
par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and PMF plots
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
par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and ECDF plots
igeom(runif(10), 0.4, show = c(1,0,1))
igeom(runif(10), 0.4, show = 5)

# plot CDF with inversion, PMF, and ECDF using show
par(mfrow = c(3,1)) # 3 rows, 1 col for CDF, PMF, and ECDF plots
igeom(runif(10), 0.4, show = c(1,1,1))
igeom(runif(10), 0.4, show = 7)

# plot three different CDF+PMF+ECDF vertical displays
par(mfcol = c(3,3))  # 3 rows, 3 cols, filling columns before rows
set.seed(8675309)
igeom(runif(20), prob = 0.4, show = 7)
igeom(runif(20), prob = 0.4, show = 7)
igeom(runif(20), prob = 0.4, show = 7)

# plot three different CDF+PMF+ECDF horizontal displays, with title only
# on the first display
par(mfrow = c(3,3))  # 3 rows, 3 cols, filling rows before columns
set.seed(8675309)
igeom(runif(20), prob = 0.4, show = 7)
igeom(runif(20), prob = 0.4, show = 7, showTitle = FALSE)
igeom(runif(20), prob = 0.4, show = 7, showTitle = FALSE)
```

```
# exhibit use of the respectLayout = FALSE option
par(mfrow = c(3,3))  # this will be ignored below since respectLayout = FALSE
set.seed(8675309)
igeom(runif(20), prob = 0.4, show = 7, respectLayout = FALSE)
par("mfrow")  # will have been reset c(1,1)
par("mar")    # will have been reset to c(5.1, 4.1, 4.1, 2.1)
```

---

inorm *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         = FALSE,
      showECDF        = FALSE,
      show            = NULL,
      plotDelay       = 0,
      maxPlotTime     = 30,
      showTitle       = TRUE,
      respectLayout   = TRUE)
```

Arguments

u	vector of uniform(0,1) random numbers, or NULL
mean	mean of the distribution
sd	standard deviation of the distribution
minPlotQuantile	controls the minimum quantile to appear in the plots
maxPlotQuantile	controls the maximum quantile to appear in the plots
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 FALSE (default), pdf plot is suppressed, otherwise pdf plot appears

showECDF	logical; if FALSE (default), ecdf plot is suppressed, otherwise ecdf plot appears
show	shortcut way of denoting plots to display; either a binary vector of length three or an integer in [0,7] (see "Details" below)
plotDelay	delay, in seconds, between generation of the random variates
maxPlotTime	maximum time, in seconds, to plot all of the random variates
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 (see "Details" below)

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. All aspects of the population distribution are output in black.

The normal distribution has density

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

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 red dots. A horizontal, dashed, red line extends from the red dot to the population cdf. At the intersection, a vertical, dashed red line extends downward to the horizontal axis, where a second red dot, denoting the associated gamma 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 Unix's 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.

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 R default values 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.

The plotDelay and maxPlotTime arguments can be used to slow down the variate generation for classroom explanation.

In the plot associated with the pdf, the maximum plotting height is associated with 125% of the maximum height of pdf. Any histogram cell that extends above this limit will have three black dots appearing above it.

Value

A vector of normal random variates

Author(s)

Barry Lawson (<blawson@richmond.edu>), Larry Leemis (<leemis@math.wm.edu>)

Examples

```
inorm(0.7, mean = 3, sd = 1)

par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and PDF plots
set.seed(8675309)
inorm(runif(10), 3, 1, showPDF = TRUE)

par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and ECDF plots
set.seed(8675309)
inorm(runif(10), 3, 1, showECDF = TRUE)

par(mfrow = c(3,1)) # 3 rows, 1 col for CDF, PDF, and ECDF plots
set.seed(8675309)
```

```
inorm(runif(10), 3, 1, showPDF = TRUE, showECDF = TRUE)

par(mfrow = c(1,1)) # 1 row, 1 col for PDF only plot
set.seed(8675309)
inorm(runif(10), 3, 1, showPDF = TRUE, showCDF = FALSE)

par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and PDF plots
inorm(runif(120), 3, 1, showPDF = TRUE, minPlotQuantile = 0.02, maxPlotQuantile = 0.98)

# overlay visual exploration of ks.test results
par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and ECDF plots
set.seed(54321)
vals <- inorm(runif(10), 3, 1, showECDF = TRUE)
D <- as.numeric(ks.test(vals, "pnorm", 3, 1)$statistic)
for (x in seq(2.5, 3.5, by = 0.05)) {
   y <- pnorm(x, 3, 1)
   segments(x, y, x, y + D, col = "darkgreen", lwd = 2, xpd = NA)
}

# plot the PDF and CDF without any variates
par(mfrow = c(2,1)) # 2 rows, 1 col for PDF and CDF plots
inorm(NULL, 3, 1)

# plot CDF with inversion and PDF using show
par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and PDF plots
inorm(runif(10), 3, 1, show = c(1,1,0))
inorm(runif(10), 3, 1, show = 6)

# plot CDF with inversion and ECDF using show
par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and ECDF plots
inorm(runif(10), 3, 1, show = c(1,0,1))
inorm(runif(10), 3, 1, show = 5)

# plot CDF with inversion, PDF, and ECDF using show
par(mfrow = c(3,1)) # 3 rows, 1 col for CDF, PDF, and ECDF plots
inorm(runif(10), 3, 1, show = c(1,1,1))
inorm(runif(10), 3, 1, show = 7)

# plot three different CDF+PDF+ECDF vertical displays
par(mfcol = c(3,3))  # 3 rows, 3 cols, filling columns before rows
set.seed(8675309)
inorm(runif(20), 3, 1, show = 7)
inorm(runif(20), 3, 1, show = 7)
inorm(runif(20), 3, 1, show = 7)

# plot three different CDF+PDF+ECDF horizontal displays, with title only
# on the first display
par(mfrow = c(3,3))  # 3 rows, 3 cols, filling rows before columns
set.seed(8675309)
inorm(runif(20), 3, 1, show = 7)
inorm(runif(20), 3, 1, show = 7, showTitle = FALSE)
inorm(runif(20), 3, 1, show = 7, showTitle = FALSE)
```

```
# exhibit use of the respectLayout = FALSE option
par(mfrow = c(3,3))  # this will be ignored below since respectLayout = FALSE
set.seed(8675309)
inorm(runif(20), 3, 1, show = 7, respectLayout = FALSE)
par("mfrow")  # will have been reset c(1,1)
par("mar")    # will have been reset to c(5.1, 4.1, 4.1, 2.1)
```

---

iunif *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         = FALSE,
      showECDF        = FALSE,
      show            = NULL,
      plotDelay       = 0,
      maxPlotTime     = 30,
      showTitle       = TRUE,
      respectLayout   = TRUE)
```

Arguments

u	vector of uniform(0,1) random numbers, or NULL
min, max	lower and upper limits of the distribution (must be finite)
minPlotQuantile	controls the minimum quantile to appear in the plots
maxPlotQuantile	controls the maximum quantile to appear in the plots
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 FALSE (default), pdf plot is suppressed, otherwise pdf plot appears
showECDF	logical; if FALSE (default), ecdf plot is suppressed, otherwise ecdf plot appears

show	shortcut way of denoting plots to display; either a binary vector of length three or an integer in [0,7] (see "Details" below)
plotDelay	delay, in seconds, between generation of the random variates
maxPlotTime	maximum time, in seconds, to plot all of the random variates
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 (see "Details" below)

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. All aspects of the population distribution are output in black.

The uniform distribution has density

$$f(x) = \frac{1}{max - min}$$

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 red dots. A horizontal, dashed, red line extends from the red dot to the population cdf. At the intersection, a vertical, dashed red line extends downward to the horizontal axis, where a second red dot, denoting the associated gamma 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 Unix's 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.

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 R default values 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.

The plotDelay and maxPlotTime arguments can be used to slow down the variate generation for classroom explanation.

In the plot associated with the pdf, the maximum plotting height is associated with 125% of the maximum height of pdf. Any histogram cell that extends above this limit will have three black dots appearing above it.

Value

A vector of uniform random variates

Author(s)

Barry Lawson (<blawson@richmond.edu>), Larry Leemis (<leemis@math.wm.edu>)

Examples

```
iunif(0.7, min = 2, max = 4)

par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and PDF plots
set.seed(8675309)
iunif(runif(10), 2, 4, showPDF = TRUE)

par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and ECDF plots
set.seed(8675309)
iunif(runif(10), 2, 4, showECDF = TRUE)

par(mfrow = c(3,1)) # 3 rows, 1 col for CDF, PDF, and ECDF plots
set.seed(8675309)
```

```
iunif(runif(10), 2, 4, showPDF = TRUE, showECDF = TRUE)

par(mfrow = c(1,1)) # 1 row, 1 col for PDF only plot
set.seed(8675309)
iunif(runif(10), 2, 4, showPDF = TRUE, showCDF = FALSE)

par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and PDF plots
iunif(runif(120), 2, 4, showPDF = TRUE, minPlotQuantile = 0.02, maxPlotQuantile = 0.98)

# overlay visual exploration of ks.test results
par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and ECDF plots
set.seed(54321)
vals <- iunif(runif(10), 2, 4, showECDF = TRUE)
D <- as.numeric(ks.test(vals, "punif", 2, 4)$statistic)
for (x in seq(2.5, 3.5, by = 0.05)) {
   y <- punif(x, 2, 4)
   segments(x, y, x, y + D, col = "darkgreen", lwd = 2, xpd = NA)
}

# plot the PDF and CDF without any variates
par(mfrow = c(2,1)) # 2 rows, 1 col for PDF and CDF plots
iunif(NULL, 2, 4)

# plot CDF with inversion and PDF using show
par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and PDF plots
iunif(runif(10), 2, 4, show = c(1,1,0))
iunif(runif(10), 2, 4, show = 6)

# plot CDF with inversion and ECDF using show
par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and ECDF plots
iunif(runif(10), 2, 4, show = c(1,0,1))
iunif(runif(10), 2, 4, show = 5)

# plot CDF with inversion, PDF, and ECDF using show
par(mfrow = c(3,1)) # 3 rows, 1 col for CDF, PDF, and ECDF plots
iunif(runif(10), 2, 4, show = c(1,1,1))
iunif(runif(10), 2, 4, show = 7)

# plot three different CDF+PDF+ECDF vertical displays
par(mfcol = c(3,3))  # 3 rows, 3 cols, filling columns before rows
set.seed(8675309)
iunif(runif(20), 2, 4, show = 7)
iunif(runif(20), 2, 4, show = 7)
iunif(runif(20), 2, 4, show = 7)

# plot three different CDF+PDF+ECDF horizontal displays, with title only
# on the first display
par(mfrow = c(3,3))  # 3 rows, 3 cols, filling rows before columns
set.seed(8675309)
iunif(runif(20), 2, 4, show = 7)
iunif(runif(20), 2, 4, show = 7, showTitle = FALSE)
iunif(runif(20), 2, 4, show = 7, showTitle = FALSE)
```

```
# exhibit use of the respectLayout = FALSE option
par(mfrow = c(3,3))  # this will be ignored below since respectLayout = FALSE
set.seed(8675309)
iunif(runif(20), 2, 4, show = 7, respectLayout = FALSE)
par("mfrow")  # will have been reset c(1,1)
par("mar")    # will have been reset to c(5.1, 4.1, 4.1, 2.1)
```

`iweibull` *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,
       maxPlotQuantile = 0.99,
       plot            = TRUE,
       showCDF         = TRUE,
       showPDF         = FALSE,
       showECDF        = FALSE,
       show            = NULL,
       plotDelay       = 0,
       maxPlotTime     = 30,
       showTitle       = TRUE,
       respectLayout   = TRUE)
```

Arguments

`u`	vector of uniform(0,1) random numbers, or NULL
`shape, scale`	shape and scale parameters (must be positive)
`minPlotQuantile`	controls the minimum quantile to appear in the plots
`maxPlotQuantile`	controls the maximum quantile to appear in the plots
`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 `FALSE` (default), pdf plot is suppressed, otherwise pdf plot appears
`showECDF`	logical; if `FALSE` (default), ecdf plot is suppressed, otherwise ecdf plot appears

show	shortcut way of denoting plots to display; either a binary vector of length three or an integer in [0,7] (see "Details" below)
plotDelay	delay, in seconds, between generation of the random variates
maxPlotTime	maximum time, in seconds, to plot all of the random variates
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 (see "Details" below)

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. All aspects of the population distribution are output in black.

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

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

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 red dots. A horizontal, dashed, red line extends from the red dot to the population cdf. At the intersection, a vertical, dashed red line extends downward to the horizontal axis, where a second red 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 Unix's 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.

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 R default values 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.

The plotDelay and maxPlotTime arguments can be used to slow down the variate generation for classroom explanation.

In the plot associated with the pdf, the maximum plotting height is associated with 125% of the maximum height of pdf. Any histogram cell that extends above this limit will have three black dots appearing above it.

Value

A vector of Weibull random variates

Author(s)

Barry Lawson (<blawson@richmond.edu>), Larry Leemis (<leemis@math.wm.edu>)

Examples

```
iweibull(0.7, 3, 2)

par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and PDF plots
set.seed(8675309)
iweibull(runif(10), 3, 2, showPDF = TRUE)

par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and ECDF plots
set.seed(8675309)
iweibull(runif(10), 3, 2, showECDF = TRUE)

par(mfrow = c(3,1)) # 3 rows, 1 col for CDF, PDF, and ECDF plots
set.seed(8675309)
```

```
iweibull(runif(10), 3, 2, showPDF = TRUE, showECDF = TRUE)

par(mfrow = c(1,1)) # 1 row, 1 col for PDF only plot
set.seed(8675309)
iweibull(runif(10), 3, 2, showPDF = TRUE, showCDF = FALSE)

par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and PDF plots
iweibull(runif(120), 3, 2, showPDF = TRUE, minPlotQuantile = 0.02, maxPlotQuantile = 0.98)

# overlay visual exploration of ks.test results
par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and ECDF plots
set.seed(54321)
vals <- iweibull(runif(10), 3, 2, showECDF = TRUE)
D <- as.numeric(ks.test(vals, "pweibull", 3, 2)$statistic)
for (x in seq(1.5, 2.0, by = 0.025)) {
   y <- pweibull(x, 3, 2)
   segments(x, y, x, y + D, col = "darkgreen", lwd = 2, xpd = NA)
}

# plot the PDF and CDF without any variates
par(mfrow = c(2,1)) # 2 rows, 1 col for PDF and CDF plots
iweibull(NULL, 3, 2)

# plot CDF with inversion and PDF using show
par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and PDF plots
iweibull(runif(10), 3, 2, show = c(1,1,0))
iweibull(runif(10), 3, 2, show = 6)

# plot CDF with inversion and ECDF using show
par(mfrow = c(2,1)) # 2 rows, 1 col for CDF and ECDF plots
iweibull(runif(10), 3, 2, show = c(1,0,1))
iweibull(runif(10), 3, 2, show = 5)

# plot CDF with inversion, PDF, and ECDF using show
par(mfrow = c(3,1)) # 3 rows, 1 col for CDF, PDF, and ECDF plots
iweibull(runif(10), 3, 2, show = c(1,1,1))
iweibull(runif(10), 3, 2, show = 7)

# plot three different CDF+PDF+ECDF vertical displays
par(mfcol = c(3,3))  # 3 rows, 3 cols, filling columns before rows
set.seed(8675309)
iweibull(runif(20), 3, 2, show = 7)
iweibull(runif(20), 3, 2, show = 7)
iweibull(runif(20), 3, 2, show = 7)

# plot three different CDF+PDF+ECDF horizontal displays, with title only
# on the first display
par(mfrow = c(3,3))  # 3 rows, 3 cols, filling rows before columns
set.seed(8675309)
iweibull(runif(20), 3, 2, show = 7)
iweibull(runif(20), 3, 2, show = 7, showTitle = FALSE)
iweibull(runif(20), 3, 2, show = 7, showTitle = FALSE)
```

```
# exhibit use of the respectLayout = FALSE option
par(mfrow = c(3,3))  # this will be ignored below since respectLayout = FALSE
set.seed(8675309)
iweibull(runif(20), 3, 2, show = 7, respectLayout = FALSE)
par("mfrow")  # will have been reset c(1,1)
par("mar")    # will have been reset to c(5.1, 4.1, 4.1, 2.1)
```

---

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

Description

Computes the sample mean of a time-persistent statistic.

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 observations

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

Computes the sample mean of the time-persistent statistic provided.

Author(s)

Larry Leemis (<leemis@math.wm.edu>), Barry Lawson (<blawson@richmond.edu>)

Examples

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

output <- ssq(seed = 54321, maxTime = 1000, 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` *Multiple-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      = defaultInterarrival,
  serviceFcn           = defaultService,
  maxTime              = Inf,
  maxDepartures        = Inf,
  saveAllStats         = FALSE,
  saveInterarrivalTimes = FALSE,
  saveServiceTimes     = FALSE,
  saveWaitTimes        = FALSE,
  saveSojournTimes     = FALSE,
  saveNumInQueue       = FALSE,
  saveNumInSystem      = FALSE,
  saveServerStatus     = FALSE,
  showOutput           = TRUE,
  showProgress         = TRUE
)
```

Arguments

`maxArrivals`	maximum number of 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 from among idle servers (default is "LRU")

`interarrivalFcn`
: function that generates next interarrival time (default uses vexp(1, 1.0, stream = 1))

`serviceFcn`
: one function, or a list of functions, that generate(s) next service time (default uses `vexp(1, 10/(9*k), stream = 2)`, where k is the number of servers)

`maxTime`
: maximum time to simulate

`maxDepartures`
: maximum number of customer departures to process

`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 (time in the system) times 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

`showProgress`
: if TRUE, displays a progress bar on screen during execution

Details

Implements a next-event implementation of a single-queue multiple-server service node simulation. 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`),
- a vector of server utilizations (`utilization`), and
- a vector of server shares of overall workload (`serverShare`).

of the multiple-server 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`) and a list of k vectors of service times per server (`serviceTimesPerServer`),
- 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 lists (time and status) each containing k vectors (one per server) noting changes to server status (`serverStatusT`, `serverStatusN`).

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

A list.

Author(s)

Barry Lawson (<blawson@richmond.edu>), Larry Leemis (<leemis@math.wm.edu>)

Examples

```
# process 2000 arrivals, R-provided seed (via NULL seed), default 2 servers
msq(2000, NULL)
# process 2000 arrivals, seed 8675309, 3 servers, LFU server selection
msq(2000, 8675309, 3, 'LFU')

msq(maxArrivals = 2000, seed = 8675309)
msq(maxTime = 1000, seed = 8675309)

##############################################################################
# example to show use of seed = NA (default) to rely on current state of generator
output1 <- msq(2000, 8675309, showOutput = FALSE, saveAllStats = TRUE)
output2 <- msq(3000,          showOutput = FALSE, saveAllStats = TRUE)
set.seed(8675309)
output3 <- msq(2000,          showOutput = FALSE, saveAllStats = TRUE)
output4 <- msq(3000,          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 = 2000, 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 = 2000, 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
```

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

Description

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

Usage

```
quantileTPS(times = NULL, numbers = NULL, probs=c(0,0.25,0.5,0.75,1.0))
```

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 observations
`probs`	a numeric vector of probabilities with values in [0,1]

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 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 `prob` vector in the same fashion as one would calculate quantiles associated with a univariate discrete probability distribution.

Value

Computes the sample quantiles of the time-persistent statistic provided.

Author(s)

Larry Leemis (<leemis@math.wm.edu>), Barry Lawson (<blawson@richmond.edu>)

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 = 1000, 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

```
data(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.

Examples

```
data(queueTrace)
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] # remove 1st element globally
    return(nextInterarr)
}
getService <- function()
{
    if (length(serviceTimes) == 0) {
        serviceTimes <<- queueTrace$serviceTimes
    }
    nextService <- serviceTimes[1]
    serviceTimes <<- serviceTimes[-1]  # remove 1st element globally
    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 amongst 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@richmond.edu>), Larry Leemis (<leemis@math.wm.edu>)

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 statistic.

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 observations

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 variance is the area under the square of 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, less the square of the sample mean. The sample standard deviation is the square root of the sample variance.

Value

Computes the sample standard deviation of the time-persistent statistic provided.

Author(s)

Larry Leemis (<leemis@math.wm.edu>), Barry Lawson (<blawson@richmond.edu>)

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 = 1000, 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@richmond.edu>), Larry Leemis (<leemis@math.wm.edu>)

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      = defaultInterarrival,
  serviceFcn           = defaultService,
  maxTime              = Inf,
  maxDepartures        = Inf,
  saveAllStats         = FALSE,
  saveInterarrivalTimes = FALSE,
  saveServiceTimes     = FALSE,
  saveWaitTimes        = FALSE,
  saveSojournTimes     = FALSE,
  saveNumInQueue       = FALSE,
  saveNumInSystem      = FALSE,
  saveServerStatus     = FALSE,
  showOutput           = TRUE,
  showProgress         = TRUE
)
```

Arguments

maxArrivals	maximum number of 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 that generates next interarrival time (default uses `vexp(1, 1.0, stream = 1)`)

`serviceFcn`	function that generates next service time (default uses `vexp(1, 10/9, stream = 2)`)
`maxTime`	maximum time to simulate
`maxDepartures`	maximum number of customer departures to process
`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 (time in the system) times 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
`showProgress`	if TRUE, displays a progress bar on screen during execution

Details

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

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

A list.

Author(s)

Barry Lawson (<blawson@richmond.edu>), Larry Leemis (<leemis@math.wm.edu>)

Examples

```
# process 2000 arrivals, R-provided seed (via NULL seed)
ssq(2000, NULL)

ssq(maxArrivals = 2000, seed = 54321)
ssq(maxDepartures = 2000, seed = 54321)
ssq(maxTime = 1000, seed = 54321)

################################################################################
# example to show use of seed = NA (default) to rely on current state of generator
output1 <- ssq(2000, 8675309, showOutput = FALSE, saveAllStats = TRUE)
output2 <- ssq(3000,          showOutput = FALSE, saveAllStats = TRUE)
set.seed(8675309)
output3 <- ssq(2000,          showOutput = FALSE, saveAllStats = TRUE)
output4 <- ssq(3000,          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 = 1000, 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
#
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

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)
```

`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 persons 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 sampled for 110 persons at Tyler's Grill in 2005. Service times are provided in seconds.

Usage

```
data(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, University of Richmond.

Examples

```
data(tylersGrill)
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)
```

---

vbinom *Variate Generator for the Binomial Distribution*

Description

Generates random variates from the binomial distribution, with options for independent streams and antithetic variates.

Usage

```
vbinom(n, size, prob, stream = NULL, antithetic = FALSE)
```

Arguments

n	number of observations
size	integer number of trials (zero or more)
prob	probability of success on each trial
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 $1 - u$

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

A vector of binomial random variates.

Author(s)

Barry Lawson (<blawson@richmond.edu>), Larry Leemis (<leemis@math.wm.edu>)

See Also

stats::rbinom, stats::runif, rstream, set.seed

Examples

```
set.seed(8675309)
vbinom(3, size = 10, prob = 0.3)  # inverts stats::runif using stats::qbinom

set.seed(8675309)
vbinom(3, 10, 0.3, stream = 1) # inverts rstream::rstream.sample using stats::qbinom
vbinom(3, 10, 0.3, stream = 2)

set.seed(8675309)
vbinom(1, 10, 0.3, stream = 1) # inverts rstream::rstream.sample using stats::qbinom
vbinom(1, 10, 0.3, stream = 2)
vbinom(1, 10, 0.3, stream = 1)
vbinom(1, 10, 0.3, stream = 2)
vbinom(1, 10, 0.3, stream = 1)
vbinom(1, 10, 0.3, stream = 2)

set.seed(8675309)
variates <- vbinom(1000, size = 10, prob = 0.3, stream = 1)
set.seed(8675309)
variates <- vbinom(1000, size = 10, prob = 0.3, stream = 1, antithetic = TRUE)
```

vexp *Variate Generator for the Exponential Distribution*

Description

Generates random variates from the exponential distribution, with options for independent streams and antithetic variates.

Usage

```
vexp(n, rate = 1, stream = NULL, antithetic = FALSE)
```

Arguments

n	number of observations
rate	rate parameter; must be positive
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 $1 - u$

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 exponential variate generator, although not particularly fast.

The stream indicated must be an integer between 1 and 25 inclusive.

If rate is not specified, it assumes the default value of 1.

The exponential distribution with rate λ has density

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

for $x \geq 0$.

Value

A vector of exponential random variates.

Author(s)

Barry Lawson (<blawson@richmond.edu>), Larry Leemis (<leemis@math.wm.edu>)

See Also

stats::rexp, stats::runif, rstream, set.seed

Examples

```
set.seed(8675309)
vexp(3)  # inverts stats::runif using stats::qexp

set.seed(8675309)
vexp(3, rate = 2)  # inverts stats::runif using stats::qexp

set.seed(8675309)
vexp(3, rate = 2, stream = 1) # inverts rstream::rstream.sample using stats::qexp
vexp(3, rate = 2, stream = 2)

set.seed(8675309)
vexp(1, rate = 2, stream = 1) # inverts rstream::rstream.sample using 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)

set.seed(8675309)
interarrivals <- vexp(1000, rate = 1,    stream = 1)
services      <- vexp(1000, rate = 10/9, stream = 2)

set.seed(8675309)
interarrivals <- vexp(1000, rate = 1,    stream = 1, antithetic = TRUE)
services      <- vexp(1000, rate = 10/9, stream = 2, antithetic = TRUE)
```

---

vgamma *Variate Generator for the Gamma Distribution*

Description

Generates random variates from the gamma distribution, with options for independent streams and antithetic variates.

Usage

```
vgamma(n, shape, rate = 1, scale = 1/rate, stream = NULL, antithetic = FALSE)
```

Arguments

n	number of observations
shape, scale	shape and scale parameters (must be positive)
rate	an alternative way to specify the scale

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 $1 - u$

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 gamma variate generator, although not particularly fast.

The stream indicated must be an integer between 1 and 25 inclusive.

If scale is omitted, it assumes the default value of 1.

The gamma distribution with parameters shape = a and scale = s has density

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

for $x \geq 0$, $a > 0$, and $s > 0$. (Here $\Gamma(a)$ is the function implemented by R's gamma() and defined in its help.)

The mean and variance are $E(X) = as$ and $Var(X) = as^2$.

Value

A vector of gamma random variates.

Author(s)

Barry Lawson (<blawson@richmond.edu>), Larry Leemis (<leemis@math.wm.edu>)

See Also

stats::rgamma, stats::runif, rstream, set.seed

Examples

```
set.seed(8675309)
vgamma(3, shape = 2, scale = 1)  # inverts stats::runif using stats::qgamma

set.seed(8675309)
vgamma(3, shape = 2, scale = 1, stream = 1)  # inverts rstream::rstream.sample using stats::qgamma
vgamma(3, shape = 2, scale = 1, stream = 2)

set.seed(8675309)
vgamma(1, shape = 2, scale = 1, stream = 1)  # inverts rstream::rstream.sample using stats::qgamma
vgamma(1, shape = 2, scale = 1, stream = 2)
vgamma(1, shape = 2, scale = 1, stream = 1)
```

```
vgamma(1, shape = 2, scale = 1, stream = 2)
vgamma(1, shape = 2, scale = 1, stream = 1)
vgamma(1, shape = 2, scale = 1, stream = 2)

set.seed(8675309)
variates <- vgamma(1000, shape = 2, scale = 1, stream = 1)
set.seed(8675309)
variates <- vgamma(1000, shape = 2, scale = 1, stream = 1, antithetic = TRUE)
```

---

vgeom *Variate Generator for the Geometric Distribution*

Description

Generates random variates from the geometric distribution, with options for independent streams and antithetic variates.

Usage

```
vgeom(n, prob, stream = NULL, antithetic = 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 $1 - u$

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 geometric 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 $0 < p \leq 1$.

Value

A vector of geometric random variates.

Author(s)

Barry Lawson (<blawson@richmond.edu>), Larry Leemis (<leemis@math.wm.edu>)

See Also

stats::rgeom, stats::runif, rstream, set.seed

Examples

```
set.seed(8675309)
vgeom(3, prob = 0.3)  # inverts stats::runif using stats::qgeom

set.seed(8675309)
vgeom(3, 0.3, stream = 1) # inverts rstream::rstream.sample using stats::qgeom
vgeom(3, 0.3, stream = 2)

set.seed(8675309)
vgeom(1, 0.3, stream = 1) # inverts rstream::rstream.sample using stats::qgeom
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(1000, prob = 0.3, stream = 1)
set.seed(8675309)
variates <- vgeom(1000, prob = 0.3, stream = 1, antithetic = TRUE)
```

---

vnorm *Variate Generator for the Normal Distribution*

Description

Generates random variates from the normal distribution, with options for independent streams and antithetic variates.

Usage

```
vnorm(n, mean = 0, sd = 1, stream = NULL, antithetic = FALSE)
```

Arguments

n	number of observations
mean	mean of the distribution
sd	standard deviation of the distribution
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 $1 - u$

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 normal variate generator, although not particularly fast.

The stream indicated must be an integer between 1 and 25 inclusive.

If mean or sd are not specified, they assume the default values of 0 and 1, respectively.

The normal distribution has density

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

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

Value

A vector of normal random variates

Author(s)

Barry Lawson (<blawson@richmond.edu>), Larry Leemis (<leemis@math.wm.edu>)

See Also

stats::rnorm, stats::runif, rstream, set.seed

Examples

```
set.seed(8675309)
vnorm(3)  # generate standard normals: inverts stats::runif using stats::qnorm

set.seed(8675309)
vnorm(3, mean = 2, sd = 1)  # inverts stats::runif using stats::qnorm

set.seed(8675309)
```

```
vnorm(3, mean = 2, sd = 1, stream = 1) # inverts rstream::rstream.sample using stats::qnorm
vnorm(3, mean = 2, sd = 1, stream = 2)

set.seed(8675309)
vnorm(1, mean = 2, sd = 1, stream = 1) # inverts rstream::rstream.sample using stats::qnorm
vnorm(1, mean = 2, sd = 1, stream = 2)
vnorm(1, mean = 2, sd = 1, stream = 1)
vnorm(1, mean = 2, sd = 1, stream = 2)
vnorm(1, mean = 2, sd = 1, stream = 1)
vnorm(1, mean = 2, sd = 1, stream = 2)

set.seed(8675309)
variates <- vnorm(1000, mean = 10, sd = 2, stream = 1)
set.seed(8675309)
variates <- vnorm(1000, mean = 10, sd = 2, stream = 1, antithetic = TRUE)
```

---

vunif *Variate Generator for the Uniform Distribution*

Description

Generates random variates from the uniform distribution, with options for independent streams and antithetic variates.

Usage

```
vunif(n, min = 0, max = 1, stream = NULL, antithetic = FALSE)
```

Arguments

n	number of observations
min, max	lower and upper limits of the distribution (must be finite)
stream	if NULL (default), uses stats::runif to generate uniform variates to invert via stats::qunif; otherwise, an integer in 1:25 indicates the rstream stream from which to generate uniform variates to invert via stats::qunif
antithetic	if FALSE (default), inverts u = uniform(0,1) variate(s) generated via either stats::runif or rstream::rstream.sample; otherwise, uses $1 - u$

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 uniform variate generator, although not particularly fast.

The stream indicated must be an integer between 1 and 25 inclusive.

The uniform distribution has density

$$f(x) = \frac{1}{max - min}$$

for $min \leq x \leq max$.

Value

A vector of uniform random variates.

Author(s)

Barry Lawson (<blawson@richmond.edu>), Larry Leemis (<leemis@math.wm.edu>)

See Also

stats::runif, rstream, set.seed

Examples

```
set.seed(8675309)
vunif(3)  # inverts stats::runif using stats::qunif

set.seed(8675309)
vunif(3, min = -2, max = 2)  # inverts stats::runif using stats::qunif

set.seed(8675309)
vunif(3, -2, 2, stream = 1) # inverts rstream::rstream.sample using stats::qunif
vunif(3, -2, 2, stream = 2)

set.seed(8675309)
vunif(1, -2, 2, stream = 1) # inverts rstream::rstream.sample using stats::qunif
vunif(1, -2, 2, stream = 2)
vunif(1, -2, 2, stream = 1)
vunif(1, -2, 2, stream = 2)
vunif(1, -2, 2, stream = 1)
vunif(1, -2, 2, stream = 2)

set.seed(8675309)
variates <- vunif(1000, min = 0, max = 10, stream = 1)
set.seed(8675309)
variates <- vunif(1000, min = 0, max = 10, stream = 1, antithetic = TRUE)
```

vweibull *Variate Generator for the Weibull Distribution*

Description

Generates random variates from the Weibull distribution, with options for independent streams and antithetic variates.

Usage

```
vweibull(n, shape, scale = 1, stream = NULL, antithetic = FALSE)
```

Arguments

n	number of observations
shape, scale	shape and scale parameters (must be positive)
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 $1 - u$

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 Weibull variate generator, although not particularly fast.

The stream indicated must be an integer between 1 and 25 inclusive.

If scale is omitted, it assumes the default value of 1.

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

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

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

Value

A vector of Weibull random variates.

Author(s)

Barry Lawson (<blawson@richmond.edu>), Larry Leemis (<leemis@math.wm.edu>)

See Also

stats::rweibull, stats::runif, rstream, set.seed

Examples

```
set.seed(8675309)
vweibull(3, shape = 2, scale = 1)  # inverts stats::runif using stats::qweibull

set.seed(8675309)
vweibull(3, shape = 2, scale = 1, stream = 1)  # inverts rstream.sample using stats::qweibull
vweibull(3, shape = 2, scale = 1, stream = 2)

set.seed(8675309)
vweibull(1, shape = 2, scale = 1, stream = 1)  # inverts rstream.sample using stats::qweibull
vweibull(1, shape = 2, scale = 1, stream = 2)
vweibull(1, shape = 2, scale = 1, stream = 1)
vweibull(1, shape = 2, scale = 1, stream = 2)
vweibull(1, shape = 2, scale = 1, stream = 1)
vweibull(1, shape = 2, scale = 1, stream = 2)

set.seed(8675309)
variates <- vweibull(1000, shape = 2, scale = 1, stream = 1)
set.seed(8675309)
variates <- vweibull(1000, shape = 2, scale = 1, stream = 1, antithetic = TRUE)
```

Index