

1.10 Probability distributions and random number generation

SAS and R can calculate quantiles and cumulative distribution values as well as generate random numbers for a large number of distributions. Random variables are commonly needed for simulation and analysis. SAS includes comprehensive random number generation through the `rand` function, while R provides a series of `r`-commands.

Both packages allow specification of a seed for the random number generator. This is important to allow replication of results (e.g., while testing and debugging). Information about random number seeds can be found in section 1.10.9.

Table 1.1 summarizes support for quantiles, cumulative distribution functions, and random numbers. More information on probability distributions within R can be found in the CRAN Probability Distributions Task View.

1.10.1 Probability density function

Both R and SAS use similar syntax for a variety of distributions. Here we use the Normal distribution as an example; others are shown in Table 1.1 (page 43).

SAS

```
data ...;
  y = cdf('NORMAL', 1.96, 0, 1);
run;
```

R

```
y <- pnorm(1.96, 0, 1)
```

1.10.2 Quantiles of a probability density function

Both R and SAS use similar syntax for a variety of distributions. Here we use the Normal distribution as an example; others are shown in Table 1.1 (p. 43).

SAS

```
data ...;
  y = quantile('NORMAL', .975, 0, 1);
run;
```

R

```
y <- qnorm(.975, 0, 1)
```

1.10.3 Uniform random variables**SAS**

```
data ...;
  x1 = uniform(seed);
  x2 = rand('UNIFORM');
run;
```

Note: The variables x_1 and x_2 are uniform on the interval (0,1). The `ranuni()` function is a synonym for `uniform()`.

R

```
x <- runif(n, 0, 1)
```

Note: The arguments specify the number of variables to be created and the range over which they are distributed.

1.10.4 Multinomial random variables**SAS**

```
data ...;
  x1 = rantbl(seed, p1, p2, ..., pk);
  x2 = rand('TABLE', p1, p2, ..., pk);
run;
```

Note: The variables x_1 and x_2 take the value i with probability p_i and value $k + 1$ with value $1 - \sum_{i=1}^k p_i$.

Table 1.1: Quantiles, probabilities, and pseudo-random number generation: distributions available in SAS and R

Distribution	R DISTNAME	SAS DISTNAME
Beta	<code>beta</code>	BETA
Beta-binomial	<code>betabin*</code>	
binomial	<code>binom</code>	BINOMIAL
Cauchy	<code>cauchy</code>	CAUCHY
chi-square	<code>chisq</code>	CHISQUARE
exponential	<code>exp</code>	EXPONENTIAL
F	<code>f</code>	F
gamma	<code>gamma</code>	GAMMA
geometric	<code>geom</code>	GEOMETRIC
hypergeometric	<code>hyper</code>	HYPERGEOMETRIC
inverse Normal	<code>inv.gaussian*</code>	IGAUSS ⁺
Laplace	<code>laplace*</code>	LAPLACE
logistic	<code>logis</code>	LOGISTIC
lognormal	<code>lnorm</code>	LOGNORMAL
negative binomial	<code>nbinom</code>	NEGBINOMIAL
normal	<code>norm</code>	NORMAL
Poisson	<code>pois</code>	POISSON
Student's t	<code>t</code>	T
Uniform	<code>unif</code>	UNIFORM
Weibull	<code>weibull</code>	WEIBULL

Note: For R, prepend `d` to the command to compute quantiles of a distribution `dDISTNAME(xvalue, parm1, ..., parmn)`, `p` for the cumulative distribution function, `pDISTNAME(xvalue, parm1, ..., parmn)`, `q` for the quantile function `qDISTNAME(prob, parm1, ..., parmn)`, and `r` to generate random variables `rDISTNAME(nrand, parm1, ..., parmn)` where in the last case a vector of `nrand` values is the result. For SAS, random variates can be generated from the `rand` function: `rand('DISTNAME', parm1, ..., parmn)`, the areas to the left of a value via the `cdf` function: `cdf('DISTNAME', quantile, parm1, ..., parmn)`, and the quantile associated with a probability (the inverse CDF) via the `quantile` function: `quantile('DISTNAME', probability, parm1, ..., parmn)`, where the number of `parms` varies by distribution. Details are available through the on-line help: Contents; SAS Products; Base SAS; SAS 9.2 Language Reference: Dictionary; Dictionary of Language Elements; Functions and CALL Routines; RAND Function. Note that in this instance SAS is case-sensitive.

* The `betabin()`, `inv.gaussian()`, and `laplace()` families of distributions are available using `library(VGAM)`.

⁺ The inverse normal is not available in the `rand` function; inverse Normal variates can be generated by taking the inverse of Normal random variates.

R

```
library(Hmisc)
x <- rMultinom(matrix(c(p1, p2, ..., pr), 1, r), n)
```

Note: The function `rMultinom()` from the `Hmisc` library allows the specification of the desired multinomial probabilities ($\sum_r p_r = 1$) as a $1 \times r$ matrix. The final parameter is the number of variates to be generated. See also `rmultinom()` in the `stats` package).

1.10.5 Normal random variables**SAS**

HELP example: see 1.13.5

```
data ...;
  x1 = normal(seed);
  x2 = rand('NORMAL', mu, sigma);
run;
```

Note: The variable X_1 is a standard Normal ($\mu = 0$ and $\sigma = 1$), while X_2 is Normal with specified mean and standard deviation. The function `rannor()` is a synonym for `normal()`.

R

```
x1 <- rnorm(n)
x2 <- rnorm(n, mu, sigma)
```

Note: The arguments specify the number of variables to be created and (optionally) the mean and standard deviation (default $\mu = 0$ and $\sigma = 1$).

1.10.6 Multivariate normal random variables

For the following, we first create a 3×3 covariance matrix. Then we generate 1000 realizations of a multivariate Normal vector with the appropriate correlation or covariance.

SAS

```
data Sigma (type=cov);
infile cards;
input _type_ $ _Name_ $ x1 x2 x3;
cards;
cov  x1      3 1 2
cov  x2      1 4 0
cov  x3      2 0 5
;
run;

proc simnormal data=sigma out=outtest2 numreal=1000;
  var x1 x2 x3;
run;
```

Note: The `type=cov` option to the `data` step defines `Sigma` as a special type of SAS dataset which contains a covariance matrix in the format shown. A similar `type=corr` dataset can be used to generate using a correlation matrix instead of a covariance matrix.

R

```
library(MASS)
mu <- rep(0, 3)
Sigma <- matrix(c(3, 1, 2,
                 1, 4, 0,
                 2, 0, 5), nrow=3)
xvals <- mvrnorm(1000, mu, Sigma)
apply(xvals, 2, mean)
```

or

```
rmultnorm <- function(n, mu, vmat, tol=1e-07)
# a function to generate random multivariate Gaussians
{
  p <- ncol(vmat)
  if (length(mu)!=p)
    stop("mu vector is the wrong length")
  if (max(abs(vmat - t(vmat))) > tol)
    stop("vmat not symmetric")
  vs <- svd(vmat)
  vsqrt <- t(vs$v %*% (t(vs$u) * sqrt(vs$d)))
  ans <- matrix(rnorm(n * p), nrow=n) %*% vsqrt
  ans <- sweep(ans, 2, mu, "+")
  dimnames(ans) <- list(NULL, dimnames(vmat)[[2]])
  return(ans)
}
xvals <- rmultnorm(1000, mu, Sigma)
apply(xvals, 2, mean)
```

Note: The returned object `xvals`, of dimension 1000×3 , is generated from the variance covariance matrix denoted by `Sigma`, which has first row and column (3,1,2). An arbitrary mean vector can be specified using the `c()` function.

Several techniques are illustrated in the definition of the `rmultnorm` function. The first lines test for the appropriate arguments, and return an error if the conditions are not satisfied. The singular value decomposition (see 1.9.10) is carried out on the variance covariance matrix, and the `sweep` function is used to transform the univariate normal random variables generated by `rnorm` to the desired mean and covariance. The `dimnames()` function applies the existing names (if any) for the variables in `vmat`, and the result is returned.

1.10.7 Exponential random variables**SAS**

```
data ...;
  x1 = ranexp(seed);
  x2 = rand('EXPONENTIAL');
run;
```

Note: The expected value of both X_1 and X_2 is 1: for exponentials with expected value k , multiply the generated value by k .

R

```
x <- rexp(n, lambda)
```

Note: The arguments specify the number of variables to be created and (optionally) the inverse of the mean (default $\lambda = 1$).

1.10.8 Other random variables

HELP example: see 1.13.5

The list of probability distributions supported within SAS and R can be found in Table 1.1, page 43. In addition to these distributions, the inverse probability integral transform can be used to generate arbitrary random variables with invertible cumulative density function F (exploiting the fact that $F^{-1} \sim U(0,1)$). As an example, consider the generation of random variates from an exponential distribution with rate parameter λ , where $F(X) = 1 - \exp(-\lambda X) = U$. Solving for X yields $X = -\log(1-U)/\lambda$. If we generate a Uniform(0,1) variable, we can use this relationship to generate an exponential with the desired rate parameter.

SAS

```
data ds;
  lambda = 2;
  uvar = uniform(42);
  expvar = -1 * log(1-uvar)/lambda;
run;
```

R

```
lambda <- 2
expvar <- -log(1-runif(1))/lambda
```

1.10.9 Setting the random number seed

SAS includes comprehensive random number generation through the `rand` function. For variables created this way, an initial seed is selected automatically by SAS based on the system clock. Sequential calls use a seed derived from this initial seed. To generate a replicable series of random variables, use the call `streaminit` function before the first call to `rand`.

SAS

```
call streaminit(42);
```

Note: A set of separate SAS functions for random number generation includes `normal`, `ranbin`, `rancau`, `ranexp`, `rangam`, `rannor`, `ranpoi`, `rantbl`, `rantri`, `ranuni`, and `uniform`. For these functions, calling with an argument of (0) is equivalent to calling the `rand` function without first running `call streaminit`; an initial seed is generated from the system clock. Calling the same functions with an integer greater than 0 as argument is equivalent to running `call streaminit` before an initial use of `rand`. In other words, this will result in a series of variates based on the first specified integer. Note that `call streaminit` or specifying an integer to one of the specific functions need only be performed once per `data` step; all seeds within that `data` step will be based on that seed.

In R, the default behavior is a seed based on the system clock. To generate a replicable series of variates, first run `set.seed(seedval)` where `seedval` is a single integer for the default “Mersenne-Twister” random number generator. For example:

R

```
set.seed(42)  
set.seed(Sys.time())
```

Note: More information can be found using `help(.Random.seed)`.