Journal of Modern Applied Statistical Methods

Volume 3 | Issue 2 Article 19

11-1-2004

Pseudo-Random Number Generation In R For Commonly Used Multivariate Distributions

Hakan Demirtas
University of Illinois at Chicago, demirtas@uic.edu

DOI: 10.22237/jmasm/1099268340

Journal of Modern Applied Statistical Methods
November, 2004, Vol. 3, No. 2, 485-497

1538 – 9472/04/$95.00

Pseudo-Random Number Generation In R For Commonly Used Multivariate Distributions

Hakan Demirtas
School of Public Health
University of Illinois at Chicago

An increasing number of practitioners and applied statisticians have started using the R programming system in recent years for their computing and data analysis needs. As far as pseudo-random number generation is concerned, the built-in generator in R does not contain multivariate distributions. In this article, R routines for widely used multivariate distributions are presented.

Key words: Simulation; computation; pseudo-random numbers

Introduction

Monte Carlo simulation has become one of the key tools in all fields of science. Simulation methodology relies on a good source of numbers that appear to be random. Methods for producing pseudo-random numbers and transforming those numbers to simulate samples from various distributions are among the most important issues in computational statistics.

For doing Monte Carlo studies, it is generally better to use a software system with a compilable programming language, such as Fortran or C. In addition to more flexibility and control, the programs built in the compiler languages execute faster. Libraries that are available in both Fortran and C contain a large number of pseudo-random number generation routines. However, using these libraries efficiently may be a daunting task for practitioners largely due to the fact that operational characteristics depend on the type of compiler, computing platform, operating system, linker and debugger, which in turn may lead to implementation difficulties.

A fundamental shift has been witnessed in recent years among statistically oriented researchers towards an extensive usage of Splus. Splus is both a data analysis system and an object-oriented programming language. Unlike Fortran or C, Splus is an interpreted (not a compiled) language. A publicly available package, called R, provides essentially the same functionality as Splus. The R programming system can be downloaded and installed at www.r-project.org.

The built-in pseudo-random number generator in R does not have routines for multivariate distributions, therefore built-in codes are not available. The purpose of this paper is to provide complementary R routines for generating pseudo-random numbers from some important multivariate distributions. In the next section, eleven R functions are presented. The quality of the resulting variates has not been tested in the computer science sense (in terms of independence, d-variate uniformity, measures based on lattice structure, etc.). However, the first two moments for random vectors and the first moment for random matrices were rigorously tested. For the purposes of most applications, fulfillment of this criterion should be a reasonable approximation to reality.

Hakan Demirtas is an Assistant Professor of Biostatistics at the University of Illinois at Chicago. His research interests are the analysis of incomplete longitudinal data, multiple imputation and Bayesian computing. E-mail: demirtas@uic.edu.

Functions for Random Number Generation

The following abbreviations are used: *PDF* stands for the probability density function; *PMF* stands for the probability mass function; *CDF* stands for the cumulative distribution function; *GA* stands for the generation algorithm; *nrep* stands for the number of identically and independently distributed random variates; *d* is the dimension. The formal arguments other than *nrep* and *d* reflect the parameters in *PDF* or *PMF*. Auxiliary functions are included as needed.

Multivariate normal distribution

$$PDF: f(x \mid \mu, \Sigma) = c \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu) \right)$$

for $-\infty < x < \infty$ and $c = (2\pi)^{-d/2} |\Sigma|^{-1/2}$, Σ is symmetric and positive definite, where μ and Σ are the mean vector and the variance-covariance matrix, respectively. *GA*: Using the Cholesky decomposition and a vector of univariate normal draws (see Code 1).

Code 1. Multivariate normal distribution:

```
draw.d.variate.normal<-function(nrep,d,mean.vec,cov.mat){
if (nrep<1)|(floor(nrep)!=nrep)){ stop("Number of replicates must be
an integer whose value is at least 1!\n")}
if((d<2)|(floor(d)!=d)){
stop("Dimension must be an integer whose value is at least 2!\n")}
if(length(mean.vec)!=d){
stop("Mean vector is misspecified, dimension is wrong!\n")}
if((ncol(cov.mat)!=d)|(nrow(cov.mat)!=d)){
stop("Variance-covariance matrix is misspecified, dimension is
wrong!\n")}
if(min(eigen(cov.mat)$values)<0)
{stop("Variance-covariance matrix must be symmetric and positive
definite!\n")}
z<-matrix(rnorm(nrep*d),nrep,d)
x<- z%*%chol(cov.mat)+t(matrix (rep(mean.vec,nrep),nrow=d))
x}
```

Multivariate *t* distribution

$$PDF: f(x \mid \mu, \Sigma, \nu) = c\left(1 + \frac{1}{\nu}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)^{-(\nu+d)/2}$$

for $-\infty < x < \infty$ and

$$c = \frac{\Gamma((\nu+d)/2)}{\Gamma(\nu/2)(\nu\pi)^{d/2}} |\Sigma|^{-1/2},$$

Σ is symmetric and positive definite and $\nu > 0$, where μ, Σ and ν are the mean vector, the variance-covariance matrix and the degrees of freedom, respectively. *GA*: Using the Cholesky decomposition, a vector of univariate normal and χ^2 draws (see Code 2).

Multivariate uniform distribution

The function in Code 3 generates a d-variate $U_d(0,1)$ distribution with specified covariance matrix Σ. *GA*: An approximate method of Falk (1999) based on *CDF* of multivariate normal deviates.

Code 2. Multivariate t distribution:

```
draw.d.variate.t<-function (df,nrep,d,mean.vec,cov.mat){
if((nrep<1)|(floor(nrep)!=nrep)){
stop("Number of replicates must be an integer whose value is at least
1!\n")}
if((d<2)|(floor(d)!=d)){
stop("Dimension must be an integer whose value is at least 2!\n")}
if(length(mean.vec)!=d){
stop("Mean vector is misspecified, dimension is wrong!\n")}
if((ncol(cov.mat)!=d)|(nrow(cov.mat)!=d)){
stop("Variance-covariance matrix is misspecified, dimension is
wrong!\n")}
if(min(eigen(cov.mat)$values)<0)
{stop("Variance-covariance matrix must be symmetric and positive
definite!\n")}
if (df<=1){
stop("Degrees of freedom must be greater than 1!\n")}
z<-matrix(rnorm(nrep*d),nrep,d)
x<-z%*%chol(cov.mat)
xt<-sqrt(df/rchisq(1,df))*x+t(matrix(rep(mean.vec,nrep),nrow=d))
xt}
```

Code 3. Multivariate uniform distribution:

```
draw.d.variate.uniform<-function(nrep,d,cov.mat){
if((nrep<1)|(floor(nrep)!=nrep)){
stop("Number of subjects must be an integer whose value is at least
1!\n")}
if(d<2){
stop("Number of variables must be at least 2!\n")}
if(length(mean.vec)!=d){
stop("Mean vector is misspecified, dimension is wrong!\n")}
if((ncol(cov.mat)!=d)|(nrow(cov.mat)!=d)){
stop("Variance-covariance matrix is misspecified, dimension is
wrong!\n")}
if(sum(cov.mat!=t(cov.mat))+min(eigen(cov.mat)$values<=0)){
stop("Variance-covariance matrix must be symmetric and positive
definite!\n")}
draw<-draw.d.variate.normal(nrep,d,mean.vec,cov.mat)
x<-pnorm(draw)
x}
```

Multivariate Bernoulli distribution (correlated binary data)

The function in Code 4 generates correlated binary variates using an algorithm developed by Park, Park and Shin (1996) based on sums of Poisson random variables in which the sums have some common terms. In Code 4, *mean.vec* corresponds to the expectations for each variable and *corr.mat* is the correlation matrix .

Multivariate hypergeometric distribution

PMF for the univariate hypergeometric distribution: $f(x \mid M, L, N) = \binom{M}{x}\binom{L-M}{N-x} \Big/ \binom{L}{N}$ for $x=max(0,N-L+M),...,min(N,M)$, where N is the number of items to be sampled, independently with equal probability and without replacement, from a lot of L items of which M are special; the realization of x is the number of special items in the random sample. In the multivariate case are more than two outcomes. *GA*: Sequential generation of succeeding conditionals which are univariate hypergeometric. In Code 5, *mean.vec* stands for the number of items in each category and k is the number of items to be sampled.

Code 4. Multivariate Bernoulli distribution (correlated binary data):

```
draw.correlated.binary<-function(nrep,d,mean.vec,corr.mat){
if((nrep<1)|(floor(nrep)!=nrep)){
stop("Number of replicates must be an integer whose value is at
least 1!\n")}
if((d<2)|(floor(d)!=d)){
stop("Dimension must be an integer whose value is at least 2!\n")}
if((max(mean.vec)>=1)|(min(mean.vec)<=0)){
stop("Expectations should be greater than 0 and less than 1!\n")}
if(length(mean.vec)!=d){
stop("Mean vector is misspecified, dimension is wrong!\n")}
if((ncol(corr.mat)!=d)|(nrow(corr.mat)!=d)){
stop("Correlation matrix is misspecified, dimension is wrong!\n")}
if(sum(corr.mat!=t(corr.mat))>0){
stop("Correlation matrix is not symmetric!\n")}
if(sum(diag(corr.mat)!=rep(1,d))>0){
stop("Not all diagonal elements of correlation matrix are 1!\n")}
if((max(corr.mat)>1)|(min(corr.mat)<0)){
stop("Correlations should be greater than or equal to 0 and less
than or equal to 1!\n")}
alpha<-matrix(0,d,d) ; cor.limit<-matrix(0,d,d)
for (i in 1:d){
for (j in 1:d){
cor.limit[i,j]<-min(sqrt((mean.vec[j]*(1-mean.vec[i]))/
(mean.vec[i]*(1-mean.vec[j]))),sqrt((mean.vec[i]*(1-mean.vec[j]))/
(mean.vec[j]*(1-mean.vec[i]))))}}
```

Code 4 Continued

```
if(sum(cor.limit>=corr.mat)<d^2){
stop("Correlations are beyond their upper limits imposed by
expectations")}
for (i in 1:d){for (j in 1:d){
alpha[i,j]<-log(1+corr.mat[i,j]*sqrt((1-mean.vec[i])*
(1-mean.vec[j])/(mean.vec[i]*mean.vec[j])))}}
beta<-matrix(0,d,d*d)
summ<-1 ; counter<-0 ; while (summ>0){
counter<-counter+1 ; minloc<-min.loc.finder(alpha); w<-matrix(1,d,d)
my.min<-apply(matrix(alpha[,-minloc],d,
d-length(unique(minloc))),2,min)
if (length(my.min)==1){w[,-minloc][my.min==0]<-0
w[-minloc,][my.min==0]<-0}; if (length(my.min)>1){
w[,-minloc][,my.min==0]<-0 ; w[-minloc,][my.min==0,]<-0
w[alpha==0]<-0}
for (i in 1:d){
beta[i,counter]<-
alpha[minloc[1],minloc[2]]*1*((minloc[1]==i)|(minloc[2]==i)|
(sum(w[,i])==d))}
alpha<-alpha-alpha[minloc[1],minloc[2]]*w
summ<-sum(alpha)} ; tbeta<-t(beta) ; w<-(tbeta!=0)
x<-matrix(0,nrep,d); y<-matrix(0,nrep,d)
pois<-numeric(nrow(tbeta)); sump<-numeric(d)
for (k in 1:nrep){for (j in 1:nrow(tbeta)){
pois[j]<-rpois(1,max(tbeta[j,]))} ; for (i in 1:d){
sump[i]<-sum(pois*w[,i])}
x[k,]<-sump}; y[x==0]<-1 ; y[x!=0]<-0
y}

min.loc.finder<-function(my.mat){
w<-is.matrix(my.mat)
if (w==F){stop("This is not a matrix!\n")}
if (nrow(my.mat)!=ncol(my.mat)){
stop("This is not a square matrix!\n")}
n<-nrow(my.mat) ; my.vec<-as.vector(t(my.mat))
my.vec[my.vec==0]<-999
my.index<-min((1:length(my.vec))[my.vec==min(my.vec)])
row.index<-floor((my.index-1)/n)+1
col.index<-my.index-d*floor((my.index-1)/n)
c(row.index,col.index)}
```

Code 5. Multivariate hypergeometric distribution:

```
draw.multivariate.hypergeometric<-function(nrep,d,mean.vec,k){
if((nrep<1)|(floor(nrep)!=nrep)){
stop("Number of replicates must be an integer whose value is at least
1!\n")}
if((d<2)|(floor(d)!=d)){
stop("Dimension must be an integer whose value is at least 2!\n")}
if(length(mean.vec)!=d){
stop("Number of items are misspecified, dimension is wrong!\n")}
if(min(mean.vec)<=0){
stop("Number of items vector cannot contain non-positive numbers!\n")}
if(sum(floor(mean.vec)!=mean.vec)>0){
stop("Number of items vector cannot contain non-integer numbers!\n")}
if((k<=0)|(floor(k)!=k)){
stop("Number of items to be sampled must be a positive integer!\n")}
if(k>sum(mean.vec)){
stop("Number of items to be sampled cannot be greater than the total
items!\n")}
x<-matrix(0,nrep,d) ; tot.m<-sum(mean.vec) ; myk<-k
for (i in 1:nrep){
summ<-tot.m ; k<-myk
for (j in 1:(d-1)){
x[i,j]<-rhyper(1,mean.vec[j],summ-mean.vec[j],k)
k<-k-x[i,j] ; summ<-summ-mean.vec[j]} ; x[i,d]<-k}
x}
```

Multivariate beta (Dirichlet) distribution

$$PDF: f(x|\alpha_1,...,\alpha_d)=\frac{\Gamma(\sum\alpha_j)}{\prod\Gamma(\alpha_j)}\prod_{j=1}^{d}x_j^{\alpha_j-1}$$ for $\alpha_j>0$, $x_j\geq 0$ and $\sum_{j=1}^{d}x_j=1$, where $\alpha=(\alpha_1,...,\alpha_j)$ is the shape vector. *GA*: Using the ratios of gamma variates with common scale parameter (β), (see Code 6).

Multinomial distribution

$$PDF: f(x|\theta_1,...,\theta_d)=\frac{N!}{\prod x_j!}\prod_{j=1}^{d}\theta_j^{x_j}$$ for $0<\theta_j<1$, $x_j\geq 0$ and $\sum_{j=1}^{d}x_j=N$, where $\theta=(\theta_1,...,\theta_d)$ is the vector of cell probabilities and N is the size. *GA*: Sequential generation of marginals which are binomials (see Code 7).

Code 6. Multivariate beta (Dirichlet) distribution:

```
draw.dirichlet<-function(nrep,d,alpha,beta){
if((nrep<1)|(floor(nrep)!=nrep)){
stop("Number of replicates must be an integer whose value is at least 1!\n")}
if((d<2)|(floor(d)!=d)){
stop("Dimension must be an integer whose value is at least 2!\n")}
if(length(alpha)!=d){stop("Shape vector is misspecified, dimension is wrong!\n")}
if(min(alpha)<=0){
stop("Shape vector cannot contain non-positive numbers!\n")}
if(beta<=0){stop("Common scale parameter must be positive!\n")}
mygamma<-matrix(rgamma(nrep*d,alpha,beta),nrep,d,byrow=T)
mybeta<-matrix(0,nrep,d); for (i in 1:nrep){
mybeta[i,]<-mygamma[i,]/sum(mygamma[i,])}
mybeta}
```

Code 7. Multinomial distribution:

```
draw.multinomial<-function(nrep,d,theta,N){
if((nrep<1)|floor(nrep)!=nrep){
stop("Number of replicate samples must be integer whose value is at least 1!\n")}
if((d<1)|floor(d)!=d){
stop("Dimension must be integer whose value is at least 2!\n")}
if (length(theta)!=d){
stop("Length of the parameter vector does not match the dimension!\n")}
if (min(theta)<0){
stop("Parameter vector contains negative values!\n")}
if (sum(theta)!=1){
stop("Sum of probabilities must be 1!\n")}
if((N<2)|floor(N)!=N){
stop("Size must be an integer whose value is at least 2!\n")}
mult<-matrix(0,nrep,d) ; mytheta<-theta; for (r in 1:nrep){
theta<-mytheta ; size<-N ; mult[r,1]<-rbinom(1,size,theta[1])
for (j in 2:(d-1)){
size<-N-sum(mult[r,1:(j-1)])
theta[j]<-theta[j]/sum(theta[j:d])
mult[r,j]<-rbinom(1,size,theta[j])}
mult[r,d]<-N-sum(mult[r,1:(d-1)])}
mult}
```

Dirichlet-Multinomial distribution

This is a mixture distribution that is a multinomial with parameter θ that is a realization of a random variable having a Dirichlet distribution with shape vector α. As before, *N* is the size, β is the common scale parameter of the gamma variates that are used for generating Dirichlet variates. *GA*: An appropriate Dirichlet is generated which, in turn, is employed to generate the multinomial conditionally (see Code 8).

Multivariate Laplace (double exponential) distribution

$$PDF: f(x \mid \mu, \Sigma, \gamma) = c \exp\left(-((x-\mu)^T \Sigma^{-1} (x-\mu))^{\gamma/2}\right)$$

for $-\infty < x < \infty$ and

$$c = \frac{\gamma \Gamma(d/2)}{2\pi^{d/2}\Gamma(d/\gamma)} |\Sigma|^{-1/2}, \Sigma \text{ is}$$

symmetric and positive definite, where μ, Σ and γ are the mean vector, the variance-covariance matrix and the shape parameter, respectively. *GA*: Involves in generation of a point *s* on the d-dimensional sphere (see the auxiliary function below for *d*=2,3,4 and Marsaglia, 1972) and a generalized univariate gamma variate (Ernst, 1998) y from the density

$$f(y \mid \alpha, \gamma) = \frac{\gamma}{\Gamma(\alpha)} y^{\alpha\gamma - 1} e^{-y^{\gamma}}$$

with $\alpha = d$. Finally, $yT^T s + \mu$ delivers variates from multivariate Laplace distribution, where $T^T T = \Sigma$ (see Code 9).

Code 8. Dirichlet-Multinomial distribution:

```
draw.dirichlet.multinomial<-function(nrep,d,alpha,beta,N){
if((nrep<1)|(floor(nrep)!=nrep)){
stop("Number of replicates must be an integer whose value is at least
1!\n")}
if((d<2)|(floor(d)!=d)){
stop("Dimension must be an integer whose value is at least 2!\n")}
if(length(alpha)!=d){
stop("Shape vector is misspecified, dimension is wrong!\n")}
if(min(alpha)<=0){stop("Shape vector cannot contain non-positive
numbers!\n")}
if(beta<=0){stop("Common scale parameter must be positive!\n")}
if((N<2)|floor(N)!=N){
stop("Size must be an integer whose value is at least 2!\n")}
dirichlet<-apply(draw.dirichlet(nrep,d,alpha,beta),2,mean)
if(sum(dirichlet)!=1){dirichlet[d]<-1-sum(dirichlet[1:d-1])}
draws<-draw.multinomial(nrep,d,dirichlet,N)
draws}
```

Code 9. Multivariate Laplace (double exponential) distribution:

```
generate.point.in.sphere<-function(nrep,d){
if ((d<2)|(floor(d)!=d)){
stop("Dimension must be an integer whose value is at least 2!\n")}
my.mat<-matrix(0,nrep,d) ; if ((d==2)|(d>4)){
for (i in 1:nrep){index<-0
while (index<1){u.mat<-runif(d)*sample(c(-1,1), d, replace = TRUE)
summ<-sum(u.mat^2) ; my.mat[i,]<-u.mat/sqrt(summ)
index<-1*(summ<=1)}}}
if (d==3){for (i in 1: nrep){
index<-0 ; while (index<1){u1<-runif(1,-1,1) ; u2<-runif(1,-1,1)
s1<-u1^2+u2^2 ; w<-(s1<=1)
index<-1*w ; my.mat[i,1][w]<-2*u1[w]*sqrt(1-s1[w])
my.mat[i,2][w]<-2*u2[w]*sqrt(1-s1[w]) ; my.mat[i,3][w]<-1-2*s1[w]}}}
if (d==4){for (i in 1: nrep){index<-0 ; while (index<1){
u1<-runif(1,-1,1) ; u2<-runif(1,-1,1) ; u3<-runif(1,-1,1)
u4<-runif(1,-1,1) ; s1<-u1^2+u2^2 ; s2<-u3^2+u4^2
w1<-(s1<=1) ; w2<-(s2<=1) ; index<-1*(w1&w2)
my.mat[i,1][w1&w2]<-u1[w1&w2] ; my.mat[i,2][w1&w2]<-u2[w1&w2]
my.mat[i,3][w1&w2]<-u3[w1&w2]*sqrt((1-s1[w1&w2])/s2[w1&w2])
my.mat[i,4][w1&w2]<-u4[w1&w2]*sqrt((1-s1[w1&w2])/s2[w1&w2])}}}
my.mat}

draw.multivariate.laplace<-function(nrep,d,gamma,mu,Sigma){
if ((d<2)|(floor(d)!=d)){
stop("Dimension must be an integer whose value is at least 2!\n")}
if(gamma<=0){stop("Shape parameter must be positive!\n")}
if((nrep<2)|(floor(nrep)!=nrep)){
stop("Number of replicates must be an integer whose value is at least
2!\n")}
if(d<2){stop("Dimension must be at least 2!\n")}
if(length(mu)!=d){stop("Mean vector is misspecified, dimension is
wrong!\n")}
if((ncol(Sigma)!=d)|(nrow(Sigma)!=d)){
stop("Variance-covariance matrix is misspecified, dimension is
wrong!\n")}
if(sum(Sigma!=t(Sigma))+min(eigen(Sigma)$values<=0))
{stop("Variance-covariance matrix must be symmetric and positive
definite!\n")}
mul.laplace<-matrix(0,nrep,d)
for (i in 1: nrep){s<-generate.point.in.sphere(1,d)
mul.laplace[i,]<-(rgamma(1,d,1)^(1/gamma))*t(chol(Sigma))%*%t(s)+mu}
mul.laplace}
```

Wishart distribution

$$PDF: f(x \mid \upsilon, S) = (2^{\upsilon d/2} \pi^{d(d-1)/4} \prod_{i=1}^{d} \Gamma((\upsilon + 1 - i)/2))^{-1} \mid S \mid^{-\upsilon/2} \mid x \mid^{(\upsilon - d - 1)/2} \exp(-\tfrac{1}{2} tr(S^{-1} x)),$$

x is positive definite, $\upsilon \geq d$ and S is symmetric and positive definite, where μ and S are the degrees of freedom and the scale matrix, respectively. *GA*: Using a simple function of the variates that follow d-variate normal distribution (see Code 10).

Inverted Wishart distribution

$$PDF: f(x \mid \upsilon, S) = (2^{\upsilon d/2} \pi^{d(d-1)/4} \prod_{i=1}^{d} \Gamma((\upsilon + 1 - i)/2))^{-1} \mid S \mid^{\upsilon/2} \mid x \mid^{-(\upsilon + d + 1)/2} \exp(-\tfrac{1}{2} tr(Sx^{-1})),$$

x is positive definite, $\upsilon \geq d$ and S is symmetric and positive definite, where μ and S^{-1} are the degrees of freedom and the inverse scale matrix, respectively. *GA*: Using Wishart variates (see Code 11).

Code 10. Wishart distribution:

```
draw.wishart<-function(nrep,d,nu,sigma){
if((nrep<1)|(floor(nrep)!=nrep)){
stop("Number of replicates must be an integer whose value is at least
1!\n")}
if((d<2)|(floor(d)!=d)){
stop("Dimension must be an integer whose value is at least 2!\n")}
if(nu<d){
stop("Distribution is not proper !\n")
stop("Degrees of freedom should be greater than or equal to the
dimension!\n")}
if(floor(nu)!=nu){
stop("Degrees of freedom should be an integer!\n")}
if((ncol(sigma)!=d)|(nrow(sigma)!=d)){
stop("Scale matrix is misspecified, dimension is wrong!\n")}
if(min(eigen(sigma)$values)<0)
{stop("Scale matrix must be symmetric and positive definite!\n")}
wishart<-matrix(0,nrep,d^2)
for (i in 1:nrep){
alpha.i<-draw.d.variate.normal(nu,d,rep(0,d),sigma)
wishart[i,]<-t(alpha.i)%*%alpha.i }
# This function generates Wishart deviates in the form of rows.
# To obtain the Wishart matrix, convert each row to a matrix where
# rows are filled first.
wishart}
```

Code 11 Inverted Wishart distribution:

```
draw.inv.wishart<-function(nrep,d,nu,inv.sigma){
if((nrep<1)|(floor(nrep)!=nrep)){
stop("Number of replicates must be an integer whose value is at least 1!\n")}
if((d<2)|(floor(d)!=d)){
stop("Dimension must be an integer whose value is at least 2!\n")}
if(nu<d){
stop("Distribution is degenerate!\n")
stop("Degrees of freedom should be greater than or equal to the dimension!\n")}
if(nu==d+1){
warning("Expectation does not exist!\n")}
if(floor(nu)!=nu){
stop("Degrees of freedom should be an integer!\n")}
if((ncol(inv.sigma)!=d)|(nrow(inv.sigma)!=d)){
stop("Inverse scale matrix is misspecified, dimension is wrong!\n")}
if(min(eigen(inv.sigma)$values)<0)
{stop("Inverse scale matrix must be symmetric and positive definite!\n")}
inv.wishart<-draw.wishart(nrep,d,nu,solve(inv.sigma))
# This function generates Wishart deviates in the form of rows.
# To obtain the Inverted-Wishart matrix, convert each row to a matrix
# where rows are filled first.
inv.wishart}
```

Results

For each distribution, the parameters can take infinitely many values and first two moments virtually fluctuate on the entire real line. Although the quality of random variates was tested by a broad range of simulations to see any potential aberrances and abnormalities in some subset of the parameter domains and to avoid any selection biases, it is constructive to report the empirical and distributional moments for arbitrarily chosen parameter values.

Table 1 tabulates the theoretical and empirical means for each distribution for arbitrary values. Throughout the table, the number of replications (*nrep*) and the dimension (*d*) are chosen to be 10,000 and 3, respectively. A similar comparison is made for the variance-covariance matrices, as shown in Table 2. In both tables, the deviations from the expected moments are found to be negligible, suggesting that random number generation routines presented are accurate.

Conclusion

The reader is invited to be cautious about the following issues: 1) It is not postulated that algorithms presented are the most efficient. Furthermore, implementation of a given algorithm may not be optimal. Given sufficient time and resources, one can write more efficient routines. 2) Quality of every random number generation process depends on the uniform number generator. McCullough (1999) raised some questions about the quality of the Splus.

Table 1: Comparison of theoretical and empirical means for arbitrarily chosen parameter values.

Distribution	*Theoretical mean*	*Empirical mean*
Normal	$(0, 0, 0)$	$(0.00139, 0.00384, 0.00377)$
t	$(0, 0, 0)$	$(0.00109, 0.00300, 0.00295)$
Uniform	$(0.5, 0.5, 0.5)$	$(0.50027, 0.49985, 0.50029)$
Bernoulli	$(0.9, 0.8, 0.7)$	$(0.89925, 0.80004, 0.69890)$
Hypergeometric	$(1.66667, 3.33333, 5)$	$(1.66911, 3.33117, 4.99972)$
Dirichlet	$(0.16667, 0.33333, 0.5)$	$(0.16670, 0.33337, 0.49992)$
Multinomial	$(8, 6, 6)$	$(7.98714, 6.01304, 5.99982)$
Dirichlet-Multinomial	$(30, 40, 30)$	$(29.98379, 40.00643, 30.00978)$
Laplace	$(0, 0, 0)$	$(0.00292, -0.00213, -0.00288)$
Wishart-Inverted Wishart	$\begin{bmatrix} 5.0 & 1.0 & 1.5 \\ 1.0 & 5.0 & 1.0 \\ 1.5 & 1.0 & 5.0 \end{bmatrix}$	$\begin{bmatrix} 5.00752 & 1.00138 & 1.50141 \\ 1.00138 & 5.00305 & 1.00462 \\ 1.50141 & 1.00462 & 4.99576 \end{bmatrix}$

Table 2: Comparison of theoretical and empirical variance-covariance matrices for arbitrarily chosen parameter values.

Distribution	*Theoretical variance-covariance*	*Empirical variance-covariance*
Normal	$\begin{bmatrix} 1.0 & 0.2 & 0.3 \\ 0.2 & 1.0 & 0.2 \\ 0.3 & 0.2 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 0.99149 & 0.19815 & 0.29178 \\ 0.19815 & 0.99946 & 0.20227 \\ 0.29178 & 0.20227 & 1.00232 \end{bmatrix}$
t	$\begin{bmatrix} 1.0 & 0.2 & 0.3 \\ 0.2 & 1.0 & 0.2 \\ 0.3 & 0.2 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 1.01178 & 0.20221 & 0.29773 \\ 0.20221 & 1.01989 & 0.20640 \\ 0.29773 & 0.20640 & 1.02281 \end{bmatrix}$
Uniform	$\begin{bmatrix} 1.0 & 0.2 & 0.3 \\ 0.2 & 1.0 & 0.2 \\ 0.3 & 0.2 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 0.99958 & 0.19164 & 0.28748 \\ 0.19164 & 0.99988 & 0.19065 \\ 0.28748 & 0.19065 & 1.00039 \end{bmatrix}$
Bernoulli	$\begin{bmatrix} 0.09000 & 0.01200 & 0.06874 \\ 0.01200 & 0.16000 & 0.09165 \\ 0.06874 & 0.09165 & 0.21000 \end{bmatrix}$	$\begin{bmatrix} 0.09060 & 0.01295 & 0.06920 \\ 0.01295 & 0.15998 & 0.09228 \\ 0.06920 & 0.09228 & 0.21044 \end{bmatrix}$
Hypergeometric	$\begin{bmatrix} 1.17702 & -0.47978 & -0.70043 \\ -0.47978 & 1.88332 & -1.42106 \\ -0.70043 & -1.42106 & 2.11864 \end{bmatrix}$	$\begin{bmatrix} 1.17180 & -0.47571 & -0.69609 \\ -0.47571 & 1.89074 & -1.41503 \\ -0.69609 & -1.41503 & 2.11112 \end{bmatrix}$
Dirichlet	$\begin{bmatrix} 0.01068 & -0.00427 & -0.00641 \\ -0.00427 & 0.01709 & -0.01282 \\ -0.00641 & -0.01282 & 0.01923 \end{bmatrix}$	$\begin{bmatrix} 0.01067 & -0.00424 & -0.00643 \\ -0.00424 & 0.01699 & -0.01275 \\ -0.00643 & -0.01275 & 0.01917 \end{bmatrix}$
Multinomial	$\begin{bmatrix} 4.8 & -2.4 & -2.4 \\ -2.4 & 4.2 & -1.8 \\ -2.4 & -1.8 & 4.2 \end{bmatrix}$	$\begin{bmatrix} 4.79330 & -2.39451 & -2.39880 \\ -2.39451 & 4.19743 & -1.80293 \\ -2.39880 & -1.80293 & 4.20172 \end{bmatrix}$
Dirichlet-Multinomial	$\begin{bmatrix} 21 & -12 & -9 \\ -12 & 24 & -12 \\ -9 & -12 & 21 \end{bmatrix}$	$\begin{bmatrix} 20.96299 & -11.98345 & -8.97954 \\ -11.98345 & 24.00701 & -12.02356 \\ -8.97954 & -12.02356 & 21.00310 \end{bmatrix}$
Laplace	$\begin{bmatrix} 1.0 & 0.2 & 0.3 \\ 0.2 & 1.0 & 0.2 \\ 0.3 & 0.2 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 0.99830 & 0.19518 & 0.29604 \\ 0.19518 & 0.99736 & 0.19502 \\ 0.29604 & 0.19502 & 0.99869 \end{bmatrix}$

At the time of this writing, a source that tested the R generator is unknown to the author. In addition, the differences between empirical and distributional moments have merely been examined for each distribution. More comprehensive and computer science-minded tests are needed possibly using DIEHARD suite (Marsaglia, 1995) or other well-regarded test suites.

In a nutshell, the R routines provided may be useful for applied scientists for simulation and computation purposes. Acknowledging the fact that dependence of the random number generation libraries on specific linkers, debuggers, compilers, operating systems and computing platforms may create problems in practice, these routines could be a handy addition to a practitioner's set of tools given the growing interest in R.

References

Ernst, M. D. (1998). A multivariate generalized Laplace distribution. *Computational Statistics*, *13*, 227-232.

Falk, M. (1999). A simple approach to the generation of uniformly distributed random variables with prescribed correlations. Communications in Statistics. *Simulation and Computation*, *28*, 85-791.

Marsaglia, G. (1972). Choosing a point from the surface of a sphere. *Annals of Mathematical Statistics*, *43*, 645-646.

Marsaglia, G. (1995). *The Marsaglia Random Number CDROM*, including the DIEHARD Battery of Tests of Randomness. Department of Statistics, Florida State University, Tallahassee, FL.

McCullough, B. D. (1999). Assessing the reliability of statistical software: Part 2. *The American Statistician*, *53*, 149-159.

Park, C. G., Park, T., & Shin D. W. (1996). A simple method for generating correlated binary variates. *The American Statistician*, *50*, 306-310.