Simulating random variables and vectors

Ignacio Cascos

2019

Outline

Discrete distributions in R

Distributions	Random numbers generation
Binomial, $\mathrm{B}(n, p)$	`rbinom(size,prob)`
Geometric, $\mathcal{G}(p)$	`rgeom(prob)`
Negative Binomial, $\mathrm{NB}(k, p)$	`rnbinom(size,prob)`
Hypergeometric, $\mathrm{H}(N_1, N_2, k)$	`rhyper(m,n,k)`
Poisson, $\mathcal{P}(\lambda)$	`rpois(lambda)`

Continuous distributions in R

Distributions	R command
Uniform, $\mathrm{U}(a, b)$	`runif(min=0,max=1)`
Exponential, $\mathrm{Exp}(\lambda)$	`rexp(rate=1)`
Normal, $\mathrm{N}(\mu, \sigma)$	`rnorm(mean=0,sd=1)`
Gamma, $\mathrm{Gamma}(k, \lambda)$	`rgamma(shape,rate=1)`
Beta, $\mathrm{Beta}(\alpha, \beta)$	`rbeta(shape1,shape2)`
Chi-square, χ^2_n	`rchisq(df)`
Student's t, t_n	`rt(df)`
Fisher's F, F_{n_1,n_2}	`rf(df1,df2)`

2.1 Inverse transform

Consider rv X with cdf $F(x) = P(X \leq x)$ and having as quantile function the *generalized inverse function of* F,

$$F^{-1}(u) = \inf\{x : F(x) \geq u\}.$$

If $U \sim \mathrm{U}(0,1)$ we have $P(U \leq u) = u$ for any $u \in [0,1]$ and given rv $X = F^{-1}(U)$, its cdf is

$$\begin{aligned} P(X \leq x) = P(F^{-1}(U) \leq x) = P(F(F^{-1}(U)) \leq F(x)) \\ = P(U \leq F(x)) = F(x), \end{aligned}$$

that is, $X \sim F$.

Inverse transform for discrete random variables

Consider X discrete rv that can assume the k values $x_1, x_2, \ldots, x_k$ with probabilities $p_1 \geq p_2 \geq \ldots \geq p_k$. The cdf of X is $F(x) = \sum_{i:x_i \leq x} p_i$. The inverse transform algorithm to obtain values from X at random is

1. Generate U random number in $(0, 1)$.
2. Set $X = x_i$ if $F(x_{i-1}) < U \leq F(x_i)$ and stop.

The values in the support of X have been ordered in terms of their probabilities in order to gain efficiency.

Inverse transform for discrete random variables

We are simulated from rv X with probability mass function

$$P(X = 0) = 0.05 \quad P(X = 1) = 0.1 \quad P(X = 2) = 0.45 \quad P(X = 3) = 0.4.$$

Which algorithm is more efficient?

1. Generate U random number in $(0, 1)$.
2. If $U < 0.05$ set $X = 0$ and stop.
3. If $U < 0.15$ set $X = 1$ and stop.
4. If $U < 0.6$ set $X = 2$ and stop.
5. Set $X = 3$ and stop.

1. Generate U random number in $(0, 1)$.
2. If $U < 0.45$ set $X = 2$ and stop.
3. If $U < 0.85$ set $X = 3$ and stop.
4. If $U < 0.95$ set $X = 1$ and stop.
5. Set $X = 0$ and stop.

Inverse transform for discrete random variables

```
sim.disc1 <- function(MC) {
  sim.disc <- rep(3,MC)
  sim.u <- runif(MC)
  for(i in 1:MC){
    if(sim.u[i]<0.05) sim.disc[i] <- 0
    else if(sim.u[i]<0.15) sim.disc[i] <- 1
    else if(sim.u[i]<0.6) sim.disc[i] <- 2
  }
  return(sim.disc)
}
```

Inverse transform for discrete random variables

```
sim.disc2 <- function(MC) {
  sim.disc <- rep(0,MC)
  sim.u <- runif(MC)
  for(i in 1:MC){
    if(sim.u[i]<0.45) sim.disc[i] <- 2
    else if(sim.u[i]<0.85) sim.disc[i] <- 3
    else if(sim.u[i]<0.95) sim.disc[i] <- 1
  }
  return(sim.disc)
}
```

Inverse transform for discrete random variables

```
sim.disc3 <- function(MC) {
  sim.disc <- rep(3,MC)
  sim.u <- runif(MC)
  zero <- which(sim.u<0.05) ; sim.disc[zero] <- 0
  one <- which(sim.u>0.05 & sim.u<0.15) ; sim.disc[one] <- 1
  two <- which(sim.u>0.15 & sim.u<0.6) ; sim.disc[two] <- 2
  return(sim.disc)
}
require(microbenchmark)
```

```
## Loading required package: microbenchmark
```

Inverse transform for discrete random variables

```
microbenchmark(sim.disc1(10),sim.disc2(10),sim.disc3(10))
```

```
## Unit: microseconds
##              expr   min    lq      mean median      uq      max
##  sim.disc1(10) 3.137 3.422 80.67657  3.707  4.1355 7588.292
##  sim.disc2(10) 2.852 3.137 75.37945  3.423  6.7005 7046.597
##  sim.disc3(10) 4.562 5.133 79.08568  8.269 10.2650 7061.138
```

```
microbenchmark(sim.disc1(1000),sim.disc2(1000),sim.disc3(1000)
```

```
## Unit: microseconds
##               expr     min       lq      mean   median       u
##  sim.disc1(1000) 141.412 149.9650 167.87232 152.2455 157.66
##  sim.disc2(1000) 116.608 125.0195 141.22962 128.5820 133.00
##  sim.disc3(1000)  61.298  88.3825  99.38794  92.0895  99.35
```

Inverse transform for discrete random variables

Whe can use the chi-square GoF test to check that any of the algorithms that we have written actually samples from the desired distribution.

```
MC <- 1000 ; set.seed(1)
sim.disc <- sim.disc3(MC)
chisq.test(x=table(sim.disc),p=c(0.05,0.1,0.45,0.4))
```

```
##
##  Chi-squared test for given probabilities
##
## data:  table(sim.disc)
## X-squared = 1.5389, df = 3, p-value = 0.6733
```

Inverse transform for discrete uniform random variable

Consider X discrete rv with $P(X = r) = 1/k$ for $x \in \{1, 2, \ldots, k\}$.

$$F_X(x) = \begin{cases} 0 & \text{if } x < 1 \\ r/k & \text{if } r \leq x < r+1,\ i \in \{1, 2, \ldots, k-1\} \\ 1 & \text{if } x \geq k \end{cases} .$$

$$F_X^{-1}(u) = r \text{ if } \frac{r-1}{k} < u \leq \frac{r}{k}, \quad \text{equiv.} \quad F_X^{-1}(u) = \lfloor uk \rfloor + 1$$

1. Generate U random number in $(0, 1)$.
2. Return $X = \lfloor U * k \rfloor + 1$. (or equiv. $X = \lceil U * k \rceil$)

Inverse transform for discrete uniform random variable

```
MC <- 1000 ; set.seed(2)
k <- 5
sim.dunif <- ceiling(runif(MC)*k)
chisq.test(x=table(sim.dunif),p=rep(1,k)/k)
```

```
##
##  Chi-squared test for given probabilities
##
## data:  table(sim.dunif)
## X-squared = 5.71, df = 4, p-value = 0.2219
```

Inverse transform for geometric random variable

Consider $X \sim \mathcal{G}(p)$ with $P(X = r) = p(1-p)^r$ for $r \in \{0, 1, 2, \ldots\}$.

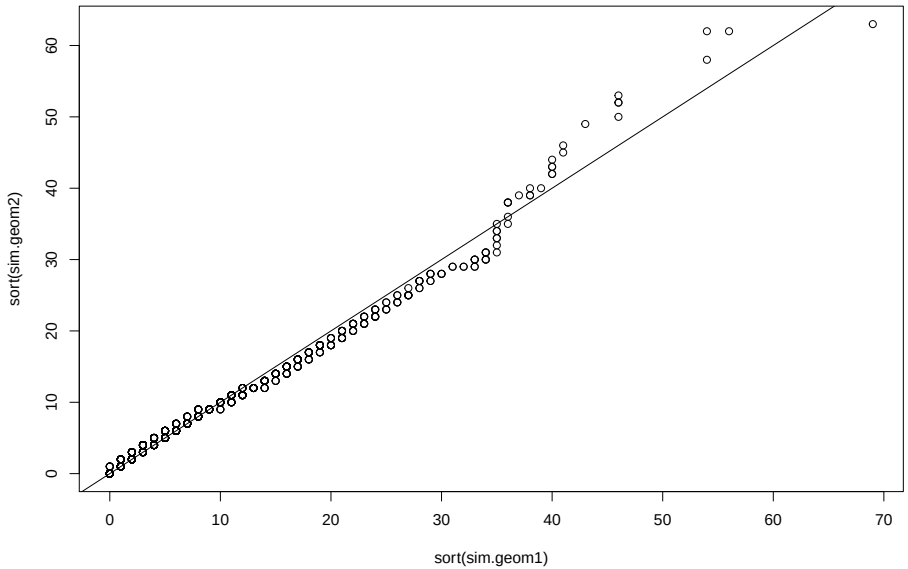
$$F_X(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 - (1-p)^{r+1} & \text{if } r \leq x < r+1,\ r \in \{0, 1, \ldots, k-1\} \end{cases}.$$

$$F_X^{-1}(u) = \lfloor \ln(1-u)/\ln(1-p) \rfloor$$

1. Generate U random number in $(0, 1)$.
2. Set $X = \lfloor \ln(U)/\ln(1-p) \rfloor$ and stop.

Inverse transform for discrete uniform random variable

```
MC <- 1000 ; set.seed(2)
p <- 0.1
sim.geom1 <- floor(log(runif(MC))/log(1-p))
sim.geom2 <- rgeom(MC,prob=p)
plot(sort(sim.geom1),sort(sim.geom2))
abline(0,1)
```



Inverse transform for continuous random variables

Consider X continuous rv with cdf F and quantile function F^{-1}. The inverse transform algorithm to obtain values from X at random is

1. Generate U random number in $(0, 1)$.
2. Set $X = F^{-1}(U)$ and stop.

Example, exponential

Let $X \sim \mathrm{Exp}(\lambda)$, its cdf is $F(x) = 1 - e^{-\lambda x}$ for $x > 0$, while its quantile function is

$$u \in (0,1),\ F^{-1}(u) = x \Leftrightarrow u = F(x) \Leftrightarrow u = 1 - e^{-\lambda x} \Leftrightarrow 1 - u = e^{-\lambda x}$$
$$\Leftrightarrow -\ln(1-u) = \lambda x \Leftrightarrow x = -\ln(1-u)/\lambda\,.$$

Inverse transform for exponential random variable

1. Generate U random number in $(0, 1)$.
2. Set $X = -\ln(1 - U)/\lambda$ (or $X = -\ln(U)/\lambda$) and stop.

```
MC <- 1000 ; set.seed(1)
sim.exp <- -log(runif(MC))
ks.test(sim.exp,"pexp",rate=1)
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  sim.exp
## D = 0.024366, p-value = 0.5928
## alternative hypothesis: two-sided
```

2.2 Aceptance-rejection

Our goal is to simulate rvs with dmf (pmf) f. We need an *instrumental* dmf (pmf) g from which we can simulate. Some values simulated from g will be rejected. Conditions on g:

- If $f(x) > 0$, then $g(x) > 0$.
- There exists $M > 0$ such that $f(x)/g(x) \leq M$ for every x.

1. Generate Y with density (mass) g.
2. Generate U random number in $(0, 1)$.
3. If $U < \frac{f(Y)}{Mg(Y)}$ set $X = Y$ and stop. Otherwise, return to Step 1.

$$P\left(Y \leq x | U < \frac{f(Y)}{Mg(Y)}\right) = \frac{P\left(Y \leq x, U < \frac{f(Y)}{Mg(Y)}\right)}{P\left(U < \frac{f(Y)}{Mg(Y)}\right)}$$

$$= \frac{\int_{-\infty}^{x} F_U\left(\frac{f(y)}{Mg(y)}\right) g(y)dy}{\int_{-\infty}^{\infty} F_U\left(\frac{f(y)}{Mg(y)}\right) g(y)dy} = \frac{\int_{-\infty}^{x} \frac{f(y)}{Mg(y)} g(y)dy}{\int_{-\infty}^{\infty} \frac{f(y)}{Mg(y)} g(y)dy} = P(X \leq x).$$

Rejection for $|Z|$ with $Z \sim \mathrm{N}(0,1)$

The dmf of $|Z|$ with $Z \sim \mathrm{N}(0,1)$ is $f(x) = \frac{2}{\sqrt{2\pi}} e^{-x^2/2}$ for $x > 0$, while the dmf of an Exponential random variable with $\lambda = 1$ is $g(x) = e^{-x}$ for $x > 0$.

$$\frac{f(x)}{g(x)} = \frac{\frac{2}{\sqrt{2\pi}} e^{-x^2/2}}{e^{-x}} = \sqrt{2/\pi} e^{x - x^2/2} \leq \sqrt{2e/\pi}.$$

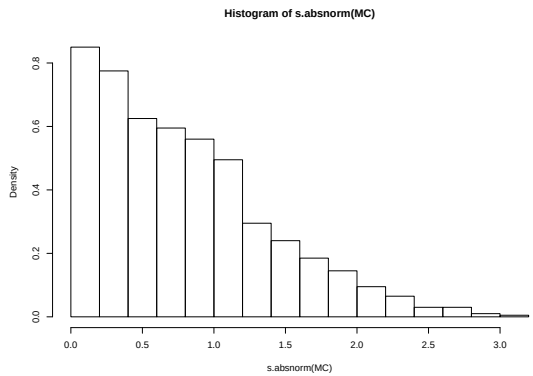
Set $M = \sqrt{2e/\pi}$ and

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

1. Generate Y with density (mass) g.
2. Generate U random number in $(0,1)$.
3. If $U < \exp\{-(Y-1)^2/2\}$ set $X = Y$ and stop. Otherwise, return to Step 1.

Rejection for $|Z|$ with $Z \sim \mathrm{N}(0,1)$

```
s.absnorm <- function(n) {
  y <- vector(length=n)
  for(i in 1:n){
    y[i] <- -log(runif(1))
    while(runif(1)>exp(-(y[i]-1)^2/2)) y[i] <- -log(runif(1))
  }
  return(y)
}
MC <- 1000 ; set.seed(1)
hist(s.absnorm(MC),probability=T)
```



Rejection for truncated models

A Zero Trucated Poisson rv X with parameter λ has pmf

$$p(r) = P(X = r) = \frac{e^{-\lambda}}{1 - e^{-\lambda}} \frac{\lambda^r}{r!}, \quad r \in \{1, 2, \ldots\}$$

Set q equal to the pmf of a $\mathcal{P}(\lambda)$ rv and $M = (1 - e^{-\lambda})^{-1}$, then $p(r)/(Mq(r)) = 1$ (and accepted) if $r \geq 1$, and $p(0)/(Mq(0)) = 0$ (and rejected).

1. Generate Y a Poisson rv.
2. If $Y \geq 1$ set $X = Y$ and stop. Otherwise, return to Step 1.

2.3 Composition approach

If we must simulate a random variable X with a **mixture distribution** either finite (or countable) or continuous

$$F(x) = \sum_{i \in I} p_i F_i \quad \text{or} \quad F(x) = \int_A \omega(a) F_a(x) \, \mathrm{d}a$$

and we are able to simulate from F_i (F_a) and a rv with pmf $(p_i)_{i \in I}$ or dmf ω, then:

1. Generate Y with pmf $(p_i)_{i \in I}$ or dmf ω.
2. Generate X with cdf F_Y and stop.

Generating a Laplace random variable

The **Laplace** (**double exponential**) distribution model with location parameter $m \in \mathbb{R}$ and rate parameter $\lambda > 0$ has density mass function

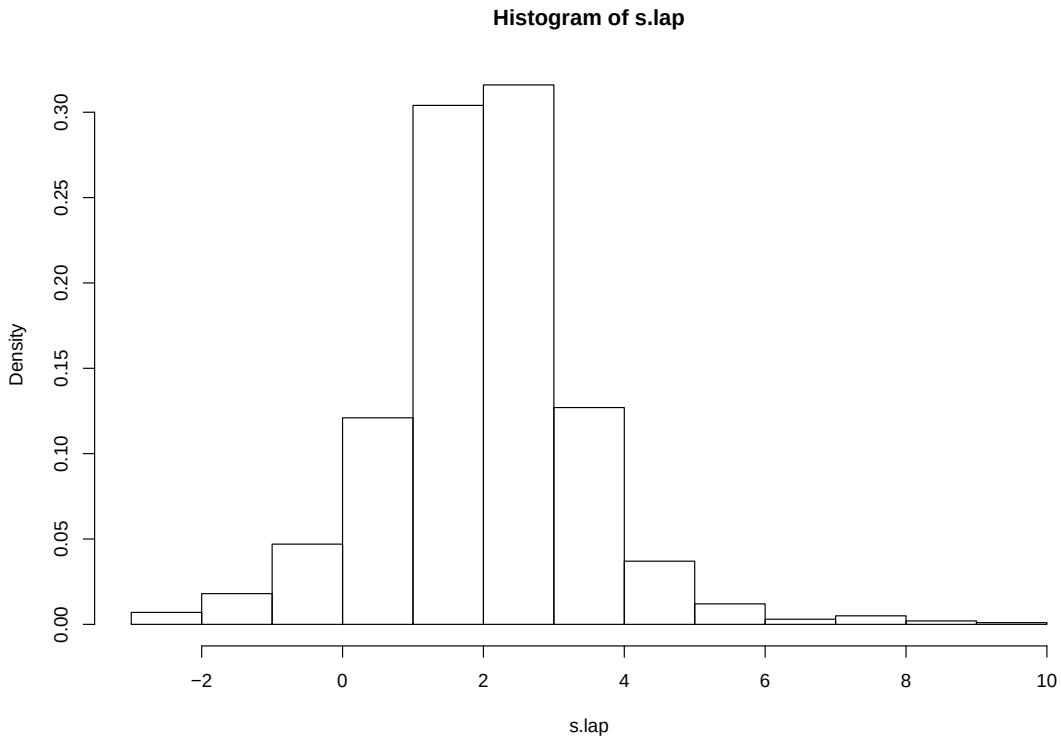
$$f(x) = \frac{\lambda}{2} e^{-\lambda|x-m|}.$$

It is thus the mixture of an Exponential rv with parameter λ and minus an Exponential rv with rate parameter λ with equal weights and shifted by m units.

1. Generate U_1, U_2 indep. random numbers in $(0, 1)$.
2. Return $X = \text{sign}(U_1 - 0.5) \ln(U_2/\lambda) + m$.

Generating a Laplace random variable

```
MC <- 1000 ; set.seed(2)
m <- 2 ; lmbd <- 1
s.lap <- sign(runif(MC)-0.5)*log(runif(MC)/lmbd)+m
hist(s.lap,prob=T)
```

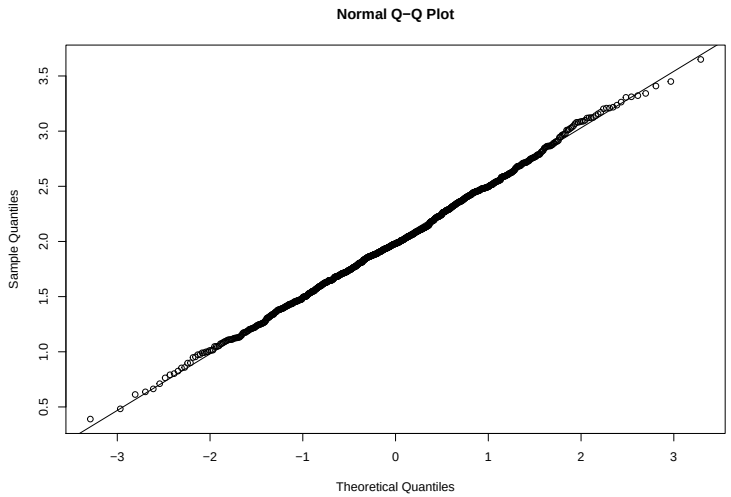


Normal rv (rejection + composition)

```
MC <- 1000 ; set.seed(1)
mu <- 2 ; sigma <- 0.5
s.norm <- sigma*(sign(runif(MC)-0.5)*s.absnorm(MC))+mu
shapiro.test(s.norm)$p.value
```

```
## [1] 0.6076152
```

```
qqnorm(s.norm) ; qqline(s.norm)
```



Generate random vector with known conditional cdfs

$$F_{X_1,X_2,X_3}(x_1,x_2,x_3) = F_{X_1}(x_1)F_{X_2|X_1}(x_2|x_1)F_{X_3|X_1,X_2}(x_3|x_1,x_2)$$

If we can simulate from the conditional distributions of a random vector, we can do as follows

1. Generate X_1 with cdf F_{X_1}.
2. Generate X_2 with cdf $F_{X_2|X_1}$.
3. Generate X_3 with cdf $F_{X_3|X_1,X_2}$.
4. Set $\mathbf{X} = (X_1, X_2, X_3)^t$ and stop.

Multinomial distribution

We use the property that the univariate marginal and conditional distributions of a *Multinomial* random vector are *Binomial* in order to simulate the *Multinomial* model.

$$\mathbf{X} = (X_1, X_2, X_3, X_4)^t \sim \mathrm{M}(n, (p_1, p_2, p_3, 1 - p_1 - p_2 - p_3))$$

1. Generate $X_1 \sim \mathrm{B}(n, p_1)$.
2. Generate $X_2 \sim \mathrm{B}(n - X_1, p2/(1 - p_1)$.
3. Generate $X_3 \sim \mathrm{B}(n - X_1, p3/(1 - p_1 - p_2)$.
4. Set $X_4 = n - X_1 - X_2 - X_3$.
5. Set $\mathbf{X} = (X_1, X_2, X_3, X_4)^t$ and stop.

Alternatively use `rmultinom(n, size, prob)`.

Random permutations

A **permutation** π of $S = \{1, 2, \ldots, n\}$ is any of the $n!$ sortings of S. That is $\pi(i) \in S$ for $i \in S$ and $\pi(i) \neq \pi(j)$ for $i, j \in S$ with $i \neq j$.

We can use the condtional cdfs argument to generate a permutation at random from $S = \{1, 2, 3, 4, 5\}$ as follows.

1. Set $\pi(1)$ random number from S.
2. Set $\pi(2)$ random number from $S \setminus \{\pi(1)\}$
3. Set $\pi(3)$ random number from $S \setminus \{\pi(1), \pi(2)\}$
4. Set $\pi(4)$ random number from $S \setminus \{\pi(1), \pi(2), \pi(3)\}$
5. Set $\pi(5) \in S \setminus \{\pi(1), \pi(2), \pi(3), \pi(4)\}$ and stop.

If A and B are two sets, $A \setminus B$ is the set formed by the elements in A that are not in B.

Random permutations

```
k <- 10 ; perm <- (1:k)
for(i in 1:(k-1)){
  interchange <- ceiling(runif(1)*(k-i+1))
  aux <- perm[i]
  perm[i] <- perm[interchange+(i-1)]
  perm[interchange+(i-1)] <- aux
}
perm
```

```
##  [1]  2 10  4  6  3  5  7  8  1  9
```

Alternatively

```
rank(runif(10)) ; sample(10,x=1:10,replace=F)
```

```
##  [1]  5  1  3  7  6 10  9  2  8  4
```

2.5 Multivariate normal distribution

Bivariate normal generation (Box-Muller's polar method)

The transformation algorithm below can be used to simulate a bivariate standard normal rv.

1. Generate U_1, U_2 independent random numbers in $(0, 1)$.
2. Set $(X_1, X_2) = \sqrt{-2 \log U_1}(\cos(2\pi U_2), \sin(2\pi U_2))$ and stop.

- If (X_1, X_2) represents a random point in the plane with X_1, X_2 independent standard normal rvs.
- The polar coordinates of (X_1, X_2) are $\sqrt{X_1^2 + X_2^2}$ and $\arctan 2(X_2, X_1)$, where $\arctan 2(\cdot, \cdot)$ is the 2-argument arctangent.
- The distance to the origin $\sqrt{X_1^2 + X_2^2}$ is the square root of a chi-square rv with 2 degrees of freedom (equivalently the square root of an exponential with $\lambda = 1/2$).
- The angle $\arctan 2(X_2, X_1)$ is a random angle in $(0, 2\pi)$.

Bivariate normal

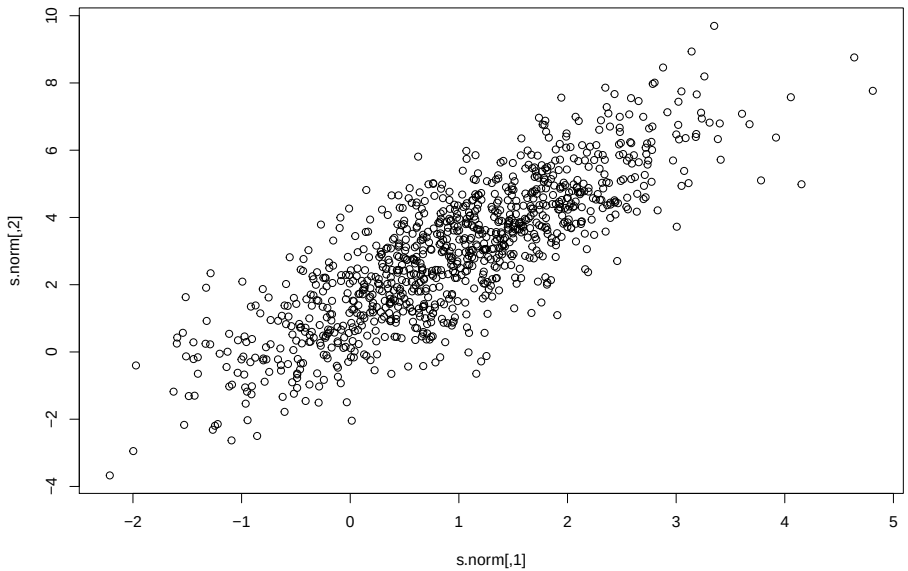
If we transform a bivariate standard normal rv $(Z_1, Z_2) \sim \mathrm{N}_2(\mathbf{0}, \mathbf{I}_2)$ as $X_1 = \mu_1 + \sigma_1 Z_1$ and $X_2 = \mu_2 + \sigma_2(Z_1\rho + Z_2\sqrt{1-\rho^2})$ then

- $\mathbb{E}[X_1] = \mathbb{E}[\mu_1 + \sigma_1 Z_1] = \mu_1$;
- $\mathrm{Var}[X_1] = \mathrm{Var}[\mu_1 + \sigma_1 Z_1] = \sigma_1^2$;
- $\mathbb{E}[X_2] = \mathbb{E}[\mu_2 + \sigma_2(Z_1\rho + Z_2\sqrt{1-\rho^2}] = \mu_2$;
- $\mathrm{Var}[X_2] = \mathrm{Var}[\mu_2 + \sigma_2(Z_1\rho + Z_2\sqrt{1-\rho^2}] = \sigma_2^2(\rho^2 + 1 - \rho^2) = \sigma_2^2$;
- $Cov(X_1, X_2) = \mathbb{E}[X_1X_2] - \mathbb{E}[X_1]\mathbb{E}[X_2] = \mu_1\mu_2 + \rho\sigma_1\sigma_2 - \mu_1\mu_2 = \rho\sigma_1\sigma_2$;
- The joint distribution of (X_1, X_2) is bivariate normal.

1. Generate Z_1 and Z_2 independent standard normal rvs.
2. Return $X_1 = \mu_1 + \sigma_1 Z_1$ and $X_2 = \mu_2 + \sigma_2(Z_1\rho + Z_2\sqrt{1-\rho^2})$.

Bivariate normal

```
mu1 <- 1 ; mu2 <- 3 ; sigma1 <- 1 ; sigma2 <- 2; rho <-0.8
MC <- 1000 ; set.seed(1)
s.bnorm <- matrix(rnorm(2*MC),nrow=2)
m.trans <- matrix(c(sigma1,0,sigma2*rho,sigma2*sqrt(1-rho^2)),
                  ncol=2,byrow=T)
s.norm <- t(m.trans%*%s.bnorm+c(mu1,mu2))
plot(s.norm)
```



Multivariate normal

Consider a d-variate normal distribution with covariance matrix Σ and mean vector μ.

1. Determine matrix C such that $CC^t = \Sigma$.
2. Generate $Z_1, \ldots, Z_d$ independent standard normal rvs.
3. Set $\mathbf{X} = C(Z_1, \ldots, Z_d)^t + \mu$ and stop.

The **Choleski decomposition** of the covariance matrix Σ is a *triangular* matrix L such that $\Sigma = LL^t$, where L is computed in R as `chol(`Σ`)` and the ouput is an *upper triangular* matrix. In case we need a lower triangular matrix, we simply take the Choleski decomposition of Σ and traspose it.

Multivariate normal

The matrix that we computed in the bivariate example equals the one we obtain with the Choleski decomposition.

```
m.trans
```

```
##      [,1] [,2]
## [1,]  1.0  0.0
## [2,]  1.6  1.2
```

```
Sigma<-matrix(c(sigma1^2,rho*sigma1*sigma2,
                rho*sigma1*sigma2,sigma2^2),ncol=2,byrow=T)
t(chol(Sigma)) # Use C=t(chol(Sigma))
```

```
##      [,1] [,2]
## [1,]  1.0  0.0
## [2,]  1.6  1.2
```

Multivariate normal with mvnorm

Alternatively use `rmvnorm(n,mean,sigma)` from package `mvtnorm`.

```
require(mvtnorm)
mu1 <- 1 ; mu2 <- 3 ; sigma1 <- 1 ; sigma2 <- 2; rho <-0.8
Sigma<-matrix(c(sigma1^2,rho*sigma1*sigma2,
                rho*sigma1*sigma2,sigma2^2),ncol=2,byrow=T)
MC <- 1000 ; set.seed(1)
plot(rmvnorm(MC,mean=c(mu1,mu2),sigma=Sigma))
```