Random Variate Generation

Canhong Wen

Agenda

- Pseudorandom numbers
- Univariate distribution
  - Common distribution
  - Inverse transform method
  - Convolution method
  - Composition method
  - Special-case techniques
- Multivariate normal distribution

Pseudorandom numbers

Pseudorandom generators produce a deterministic sequence that is indistinguishable from a true random sequence if you don't know how it started.

```
runif(1:5)
```

```
## [1] 0.64293220 0.03531451 0.29935124 0.57790069 0.02232273
```

```
set.seed(5) ; runif(1:5)
```

```
## [1] 0.2002145 0.6852186 0.9168758 0.2843995 0.1046501
```

```
set.seed(5) ; runif(1:5)
```

```
## [1] 0.2002145 0.6852186 0.9168758 0.2843995 0.1046501
```

Sample from a finite population

```
sample(0:1, size = 10, replace = TRUE) # toss some coins
```

```
##  [1] 0 0 0 1 0 0 0 0 1 1
```

```
sample(1:100, size = 6, replace = FALSE) # choose some lottery numbers
```

```
## [1] 47 53 76 16 26 27
```

```
sample(letters) # permutation of letters a-z
```

```
##  [1] "i" "v" "j" "u" "s" "q" "m" "o" "k" "n" "b" "z" "x" "w" "d" "h" "r"
## [18] "l" "y" "t" "e" "c" "g" "a" "f" "p"
```

```
# sample from a multinomial distribution
x <- sample(1:3, size = 100, replace = TRUE, prob = c(.2, .3, .5))
table(x)
```

```
## x
##  1  2  3
## 23 26 51
```

Common distributions

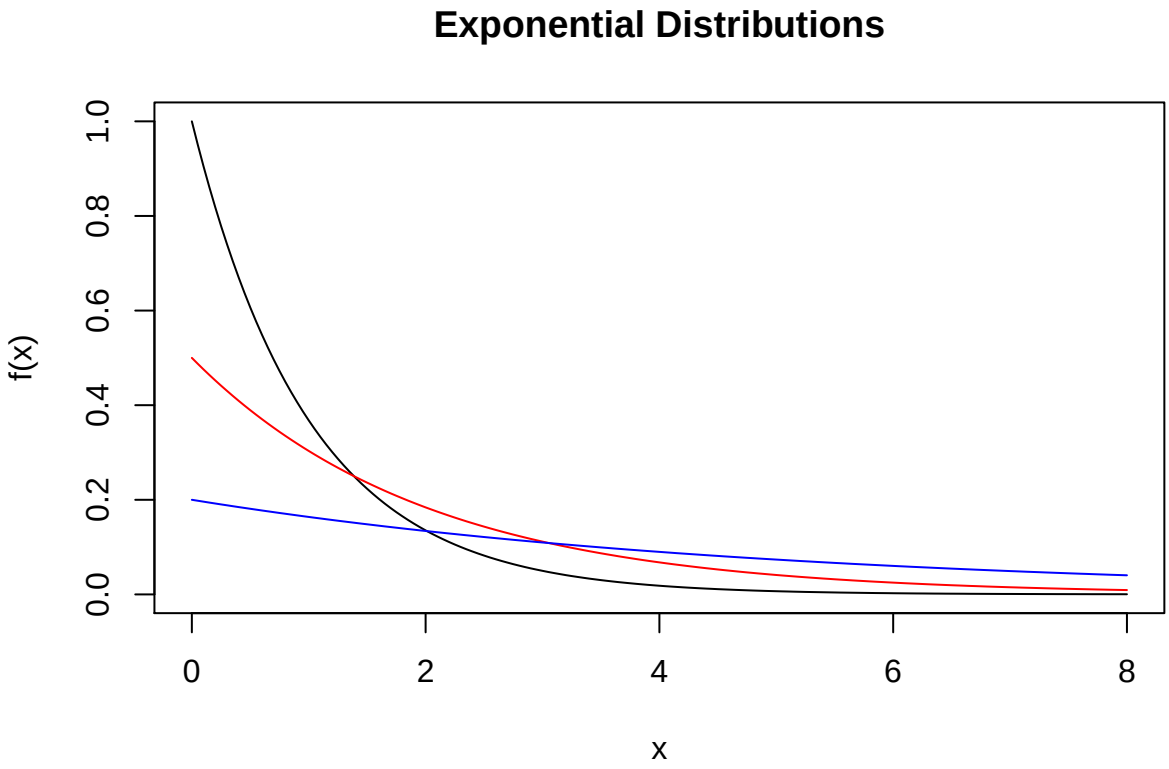
Every distribution that R handles has four functions. There is a root name, for example, the root name for the normal distribution is `norm`. This root is prefixed by one of the letters

- `p` for "probability", the cumulative distribution function (c. d. f.)
- `q` for "quantile", the inverse c. d. f.
- `d` for "density", the density function (p. f. or p. d. f.)
- `r` for "random", a random variable having the specified distribution

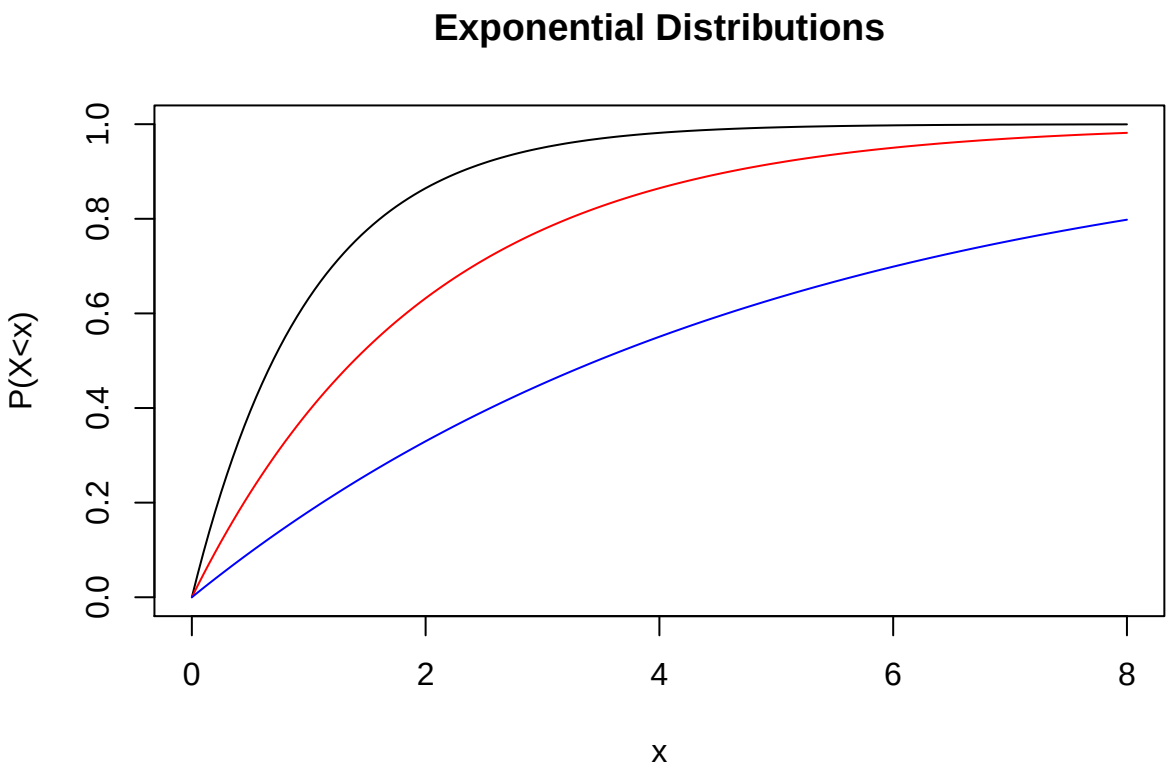
Distribution	cdf	parameter
Beta	`pbeta`	`shape1`, `shape2`
Binomial	`pbinom`	`size`, `prob`
Chi-Square	`pchisq`	`df`
Exponential	`pexp`	`rate`
F	`pf`	`df1`, `df2`
Gamma	`pgamma`	`shape`, `rate` or `shape`
Geometric	`pgeom`	`prob`
Log Normal	`plnorm`	`meanlog`, `sdlog`
Negative Binomial	`pnbinom`	`size`, `prob`
Normal	`pnorm`	`mean`, `sd`
Poisson	`ppois`	`lambda`
Student t	`pt`	`df`
Uniform	`punif`	`min`, `max`

Example: exponential distribution

```
this.range <- seq(0, 8, .05)
plot (this.range, dexp(this.range), ty="l", main="Exponential Distributions",
      xlab="x", ylab="f(x)")
lines (this.range, dexp(this.range, rate=0.5), col="red")
lines (this.range, dexp(this.range, rate=0.2), col="blue")
```




```
this.range <- seq(0, 8, .05)
plot (this.range, pexp(this.range), ty="l", main="Exponential Distributions",
      xlab="x", ylab="P(X<x)")
lines (this.range, pexp(this.range, rate=0.5), col="red")
lines (this.range, pexp(this.range, rate=0.2), col="blue")
```



Inverse transformation method

Inverse Transformation Theorem Let X be a continuous random variable with c.d.f. $F(x)$. Then $F(X) \sim \mathcal{U}(0,1)$.

- For discrete variable, we can define the inverse c.d.f. by

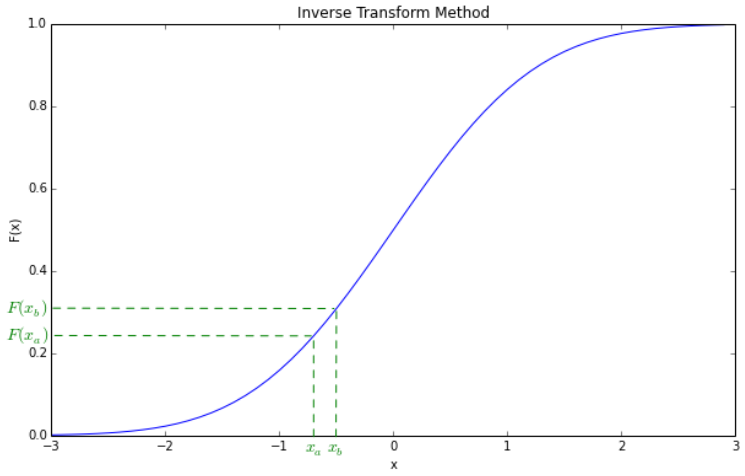
$$F^{-1}(y) = \min\{x, F(x) \geq y\}, y \in [0,1].$$

  Then the above theorem can be applied to a discrete random variable.

- Used when F^{-1} can be determined either analytically or empirically.

So here is the *inverse transformation method* for generating a RV X having c.d.f. $F(x)$:

1. Sample U from $\mathcal{U}(0,1)$.
2. Return $X = F^{-1}(U)$.



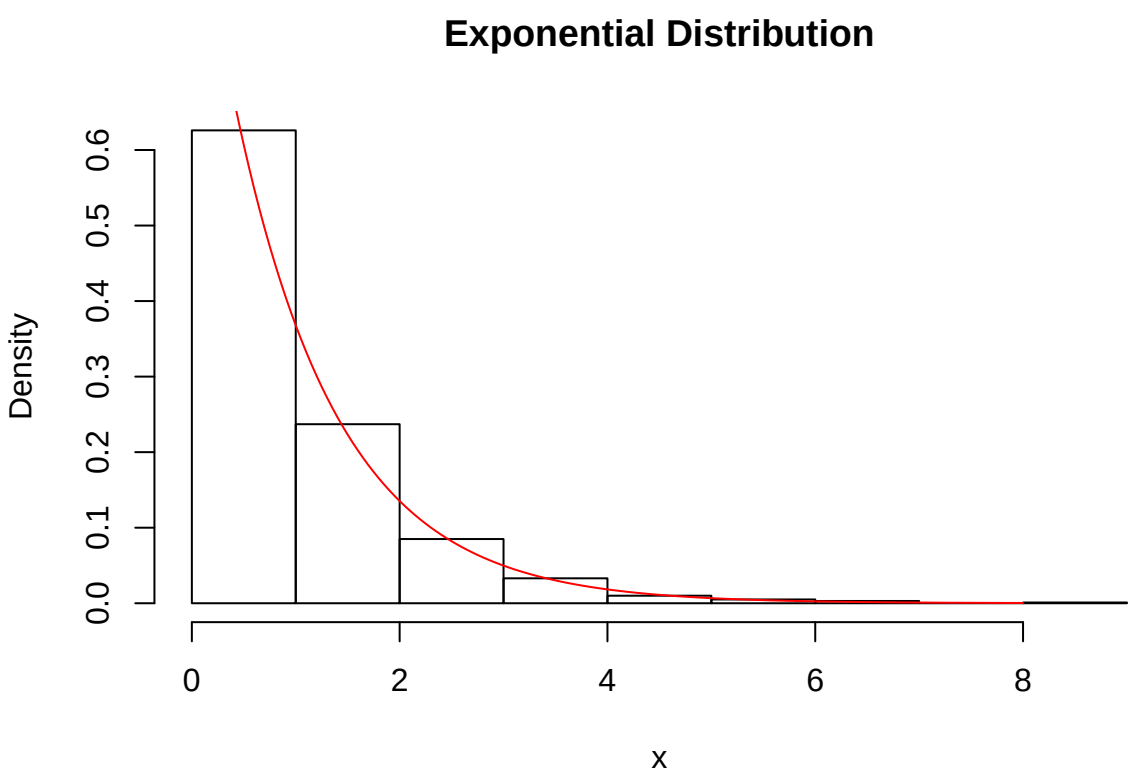
Example: continuous distribution

The $Exp(\lambda)$ distribution, with $F(x) = 1 - e^{-\lambda x}, x > 0$.

Solving $F(X) = U$ for X,

$$X = -\frac{1}{\lambda}\log(1-U) \quad \text{or} \quad X = -\frac{1}{\lambda}\log(U).$$

```
u <- runif(1000)
lambda <- 1
x <- -1/lambda * log(u)
hist(x, probability = TRUE, main="Exponential Distribution")
y <- seq(0, 8, .05)
lines(y, dexp(y, rate = lambda), col="red")
```



Example: discrete distribution

The geometric distribution with p.m.f. and c.d.f.

$$f(x) = q^x p \quad \text{and} \quad F(x) = 1 - q^{x+1}, x = 0, 1, 2, \ldots,$$

where $q = 1 - p$. Thus, after some algebra,

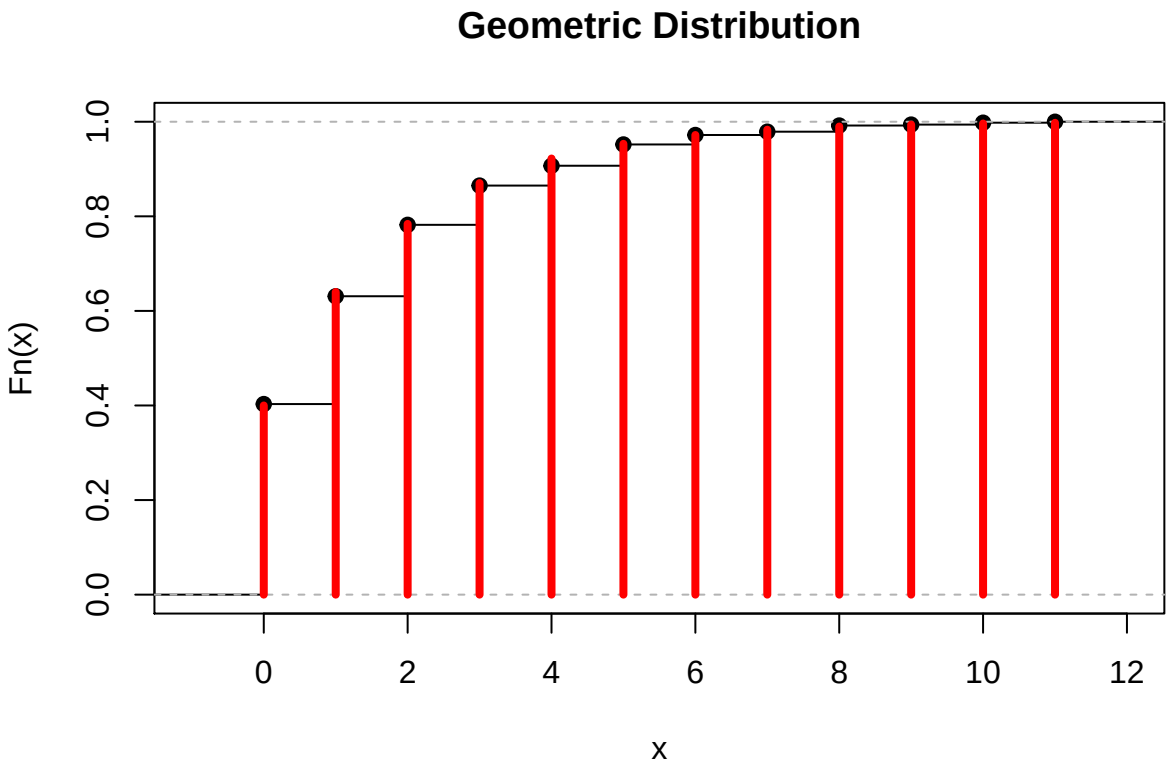
$$X = \min\{x, 1 - q^x \geq U\} = \left\lceil \frac{\log(1-U)}{\log(1-p)} \right\rceil - 1.$$

```
n <- 1000
p <- 0.4
u <- runif(n)
x <- ceiling(log(1-u) / log(1-p)) -1
table(x)
```

```
## x
##   0   1   2   3   4   5   6   7   8   9  10  11
## 403 228 151  83  42  45  20   7  13   2   4   2
```

```
# plot(table(x)/n, main="Geometric Distribution", type="h")
plot.ecdf(x, main="Geometric Distribution")
```

```
y <- seq(min(x), max(x), 1)
lines(y, pgeom(y, prob = p), col="red", type="h", lwd=4)
```



Motivation for Acceptance-Rejection Method (A-R)

The majority of c.d.f.'s cannot be inverted efficiently. A-R samples from a distribution that is "almost" the one we want, and then adjusts by "accepting" only a certain proportion of those samples.

Baby Example: Generate a $\mathcal{U}(\frac{2}{3}, 1)$ RV. (You would usually do this via inverse transform, but what the heck!) Here is the A-R algorithm:

1. Generate $U \sim U(0, 1)$.
2. If $U \geq \frac{2}{3}$, ACCEPT $X \leftarrow U$. Otherwise, REJECT and go to Step 1.

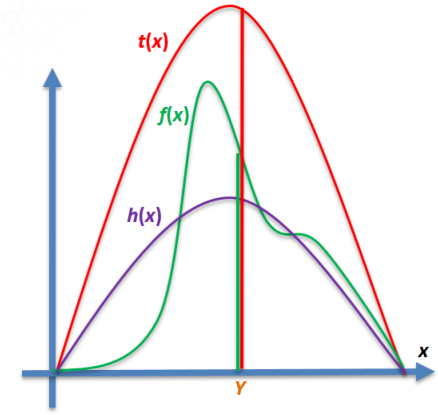
Theorem

- Suppose we want to simulate a continuous RV X with p.d.f. $f(x)$, but that is difficult to generate directly.
- A easily generated RV Y having p.d.f. $h(x) \equiv t(x)/c$, where $t(x)$ majorizes $f(x)$, i.e., $t(x) \geq f(x), x \in \mathbf{R}$.
- $g(x) \equiv f(x)/t(x)$. Note that $0 \leq g(x) \leq 1$.

Theorem (von Neumann 1951) Let $U \sim \mathcal{U}(0, 1)$ and Y be a RV (inndependent of U) with p.d.f. $h(y) = t(y)/c$. If $U \leq g(Y)$, then Y has (conditional) p.d.f. f(y).

Acceptance-Rejection (A-R) Algorithm

- Repeat
- Generate U from $\mathcal{U}(0,1)$
- Generate Y from $h(y)$ (independent of U)
- until $U \leq g(Y) = \frac{f(Y)}{t(Y)} = \frac{f(Y)}{ch(Y)}$.
- Return $X \leftarrow Y$.



Generate a point *Y* uniformly under *t*(*x*) (equivalently, sample *Y* from p.d.f. *h*(*x*)).

Accept the point with probability *f*(*Y*) / *t*(*Y*) = *f*(*Y*) / [*c h*(*Y*)].

If you accept, then set *X* = *Y* and stop.

There are two main issues:

- The ability to quickly sample from $h(y)$.
- $c \geq 1$ ought to be as close to 1 as possible, i.e., $t(x)$ must be "close" to $f(x)$. That is because

$$P(U \leq g(Y)) = \frac{1}{c}$$

and the number of trials until "success" $\{U \leq g(Y)\}$ is Geom$(1/c)$, so that the mean number of trial is c.

Example

Generate a RV with p.d.f. $f(x) = 60x^3(1-x)^2, 0 \leq x \leq 1$. Cannot invert this analytically.

The maximum occurs at $x = 0.6$ and $f(0.6) = 2.0736$.

Using the majorizing function $t(x) = 2.0736, 0 \leq x \leq 1$, get $c = \int_0^1 t(x)dx = 2.0736$, so

$$h(x) = \frac{t(x)}{c} = 1, 0 \leq x \leq 1 \quad (i.e., a\ \mathcal{U}(0,1) p.d.f.)$$

and

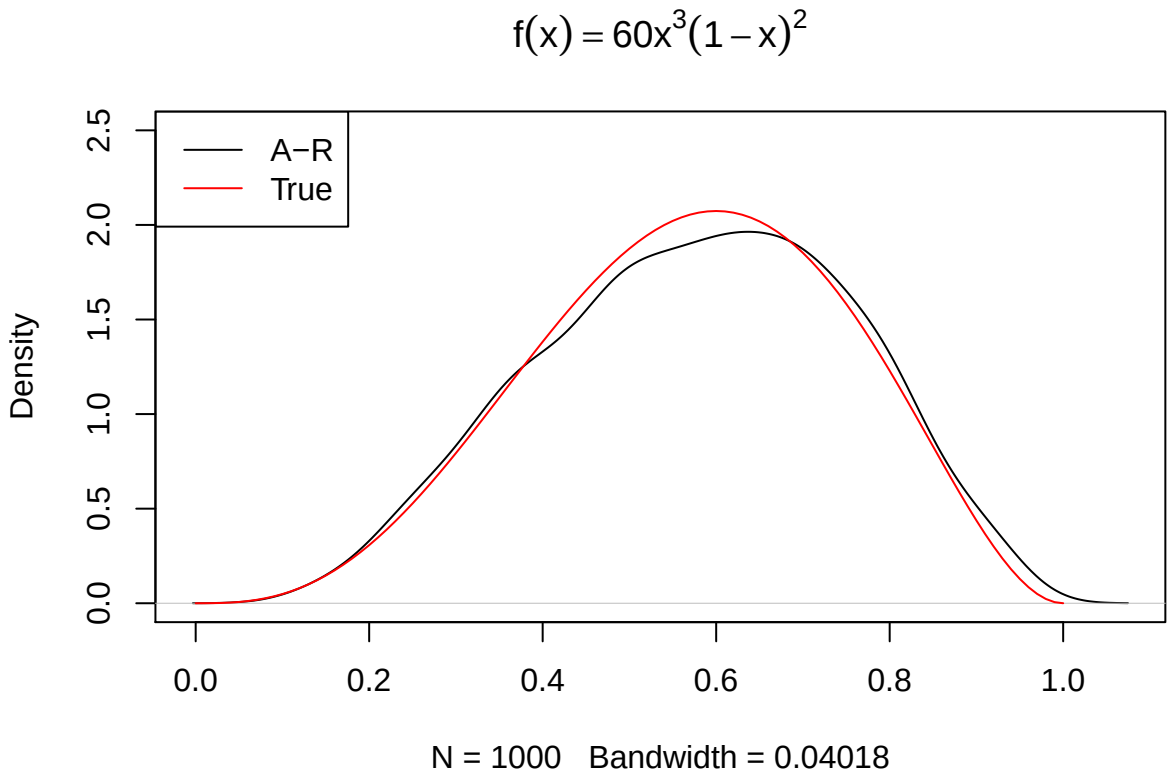
$$g(X) = \frac{f(x)}{t(x)} = 60x^3(1-x)^2/2.0736.$$

```
n <- 1000
k <- 0            # counter for acception
j <- 0            # iterations
x <- numeric(n)    # initiation

while(k < n){
  u <- runif(1)
  j <- j + 1
  y <- runif(1) # random variate from g
  if(60*y^3*(1-y)^2/2.0736 > u){ # we accept y
    k <- k + 1
    x[k] <- y
  }
}
j
```

```
## [1] 2004
```

```
plot(density(x), main = bquote(f(x)==60*x^3*(1-x)^2), ylim = c(0, 2.5))
this.range <- seq(0, 1, 0.01)
f <- function(x) 60*x^3*(1-x)^2
lines(this.range,f(this.range), col="red")
legend("topleft", legend = c("A-R", "True"), col=c("black", "red"), lty=1)
```



Exercise: half-normal distribution

Generate a standard half-normal RV with p.d.f

$$f(x) = \frac{2}{\sqrt{2\pi}} e^{-x^2/2}, \quad x \geq 0$$

Convolution Method

Convolution refers to adding things up.

Example: $\text{Bin}(r, p)$.

If $X_1, \ldots, X_r \sim$ i.i.d. $\text{Bern}(p)$, then $\sum_{i=1}^{r} X_i \sim \text{Bin}(r, p)$

```
n <- 10000
x <- rbinom(n, 1, 0.3)
y <- rbinom(n, 1, 0.3)
z <- x + y
table(z)/n
```

```
## z
##      0      1      2
## 0.4926 0.4192 0.0882
```

What if $\sum_{i=1}^{r} X_i$? (Avoid to use `for`)

```r
n <- 10000
r <- 2
x <- matrix(rbinom(n*r, 1, 0.3), n, r)
z <- rowSums(x)
table(z)/n
```

```
## z
##      0      1      2
## 0.4900 0.4208 0.0892
```

```r
z <- apply(x, MARGIN = 1, FUN = sum)
table(z)/n
```

```
## z
##      0      1      2
## 0.4900 0.4208 0.0892
```

Other examples:

- If $X_1, \ldots, X_r$ are i.i.d. Geom(p), then $\sum_{i=1}^{r} X_i \sim NegBin(r, p)$.
- If $X_1, \ldots, X_r$ are i.i.d. $\mathcal{N}(0, 1)$, then $\sum_{i=1}^{r} X_i^2 \sim \chi^2(r)$.
- If $X_1, \ldots, X_r$ are i.i.d. Exp(λ), then $\sum_{i=1}^{r} X_i \sim$ Gamma(r, λ).

Composition Method

Mixture distribution:

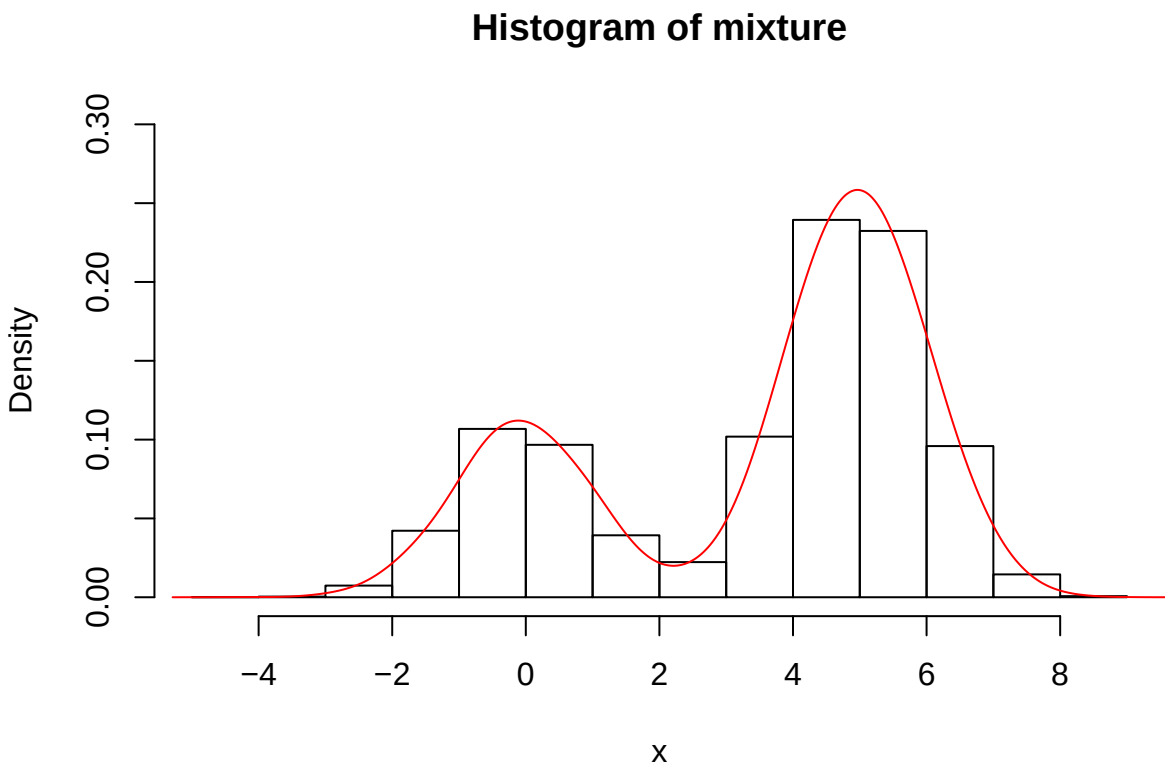
$$F(x) = \sum_{j=1} p_j F_j(x),$$

where $p_j > 0$ for all j and $\sum_j p_j = 1$.

To generate a RV with c.d.f. $F(x)$, we do

- Generate a positive integer J such that $P(J = j) = p_j$ for all j.
- Return X from c.d.f. $F_J(x)$.

```r
n <- 10000
p <- c(0.3, 0.7)
k <- sample(0:1, size = n, replace = TRUE, prob = p)
x1 <- rnorm(n)
x2 <- rnorm(n, mean = 5, sd = 1)
x <- (1-k)*x1 + k*x2
hist(x, probability = TRUE, main="Histogram of mixture", ylim = c(0,0.3))
lines(density(x), col = "red")#, main = "Density plot of mixture")
```



Special-Case Techniques

1. If $X \sim \chi^2(n)$ and $Y \sim \chi^2(m)$ and X and Y are indpendent, then $F = \frac{X/n}{Y/m} \sim F(n, m)$.
2. If $Z \sim \mathcal{N}(0,1)$ and $Y \sim \chi^2(n)$, and Z and Y are independent, then $T = \frac{Z}{\sqrt{Y/n}} \sim t(n)$.
3. Box-Muller Method: If U_1, U_2 are i.i.d. $\mathcal{U}(0,1)$, then

$$Z_1 = \sqrt{-2\log(U_1)}\cos(2\pi U_2) Z_2 = \sqrt{-2\log(U_1)}\sin(2\pi U_2)$$

   are i.i.d $\mathcal{N}(0,1)$.

Multivariate Normal Distribution

The random vector $\boldsymbol{X} = (X_1, \ldots, X_d)^\top$ has the multivariate normal distribution with mean vector $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_d)^\top$ and $d \times d$ covariance matrix $\Sigma = (\sigma_{ij})$ if it has p.d.f.

$$f(\boldsymbol{x}) = \frac{1}{(2\pi)^{k/2}|\Sigma|^{1/2}} \exp\left\{-\frac{(\boldsymbol{x}-\boldsymbol{\mu})^\top \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}{2}\right\}, \boldsymbol{x} \in \mathbb{R}^k.$$

Denote $\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$.

```
MASS:mvrnorm(n, mean = mu, sigma = sigma)
mvtnorm:rmvnorm(n, mean = mu, sigma = sigma)
```

In order to generate $\boldsymbol{X}$:

1. Start with $\boldsymbol{Z} = (Z_1, \ldots, Z_d)$ with each element i.i.d. from $\mathcal{N}(0, 1)$.
2. Decompose Σ as $\Sigma = CC^\top$.
3. Then $\boldsymbol{X} = \boldsymbol{\mu} + C\boldsymbol{Z} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$.

In R:

```
Z <- matrix(rnorm(n*d), nrow = n, ncol = d)
X <-  Z%*%C + matrix(mu, n, d, byrow = TRUE)
```

So what kinds of decomposition?

Decomposition of Σ

1. **Spectral decomposition or eigendecomposition**
   - Let $\Sigma = P\Lambda P^\top$, where Λ consists of the eigenvalues of Σ and P consists of the corresponding eigenvectors.
   - Then we have $\Sigma^{1/2} = P\Lambda^{1/2}P^\top$.
   - $C = \Sigma^{1/2}$.
   - `eigen`
2. **Choleski decomposition**
   - $\Sigma = CC^\top$, where C is a lower triangular matrix.
   - `chol`

```
rmvn.eigen <- function(n, mu, Sigma){
  # Generate n random vectors from MVN(mu, Sigma)
  # dimension is inferred from mu and Sigma
  d <- length(mu)
  ev <- eigen(Sigma, symmetric = TRUE)
  lambda <- ev$values
  V <- ev$vectors
  C <- V %*% diag(sqrt(lambda)) %*% t(V)
  Z <- matrix(rnorm(n*d), nrow = n, ncol = d)
  X <-  Z%*%C + matrix(mu, n, d, byrow = TRUE)
  X
}
```

```
rmvn.Choleski <- function(n, mu, Sigma){
  # Generate n random vectors from MVN(mu, Sigma)
  # dimension is inferred from mu and Sigma
  d <- length(mu)
  Q <- chol(Sigma) # Q is a upper triangular matrix
  Z <- matrix(rnorm(n*d), nrow = n, ncol = d)
  X <-  Z%*%t(Q) + matrix(mu, n, d, byrow = TRUE)
  X
}
```

Comparison for different methods

```
  library(MASS)
  library(mvtnorm)
  n <- 100          # Sample size
  d <- 30           # Dimension
  N <- 1000         # Iterations
  mu <- numeric(d)
  Sigma <- cov(matrix(rnorm(n*d), n, d))

  set.seed(100)
  system.time(for(i in 1:N) mvrnorm(n, mu, Sigma)) # MASS
```

```
##    user  system elapsed
##    0.52    0.00    0.51
```

```
  set.seed(100)
  system.time(for(i in 1:N) rmvn.eigen(n, mu, Sigma))
```

```
##    user  system elapsed
##    0.46    0.00    0.48
```

```
  set.seed(100)
  system.time(for(i in 1:N) rmvn.Choleski(n, mu, Sigma))
```

```
##    user  system elapsed
##    0.28    0.00    0.28
```

```
  set.seed(100)
  system.time(for(i in 1:N) rmvnorm(n, mu, Sigma)) # mvtnorm
```

```
##    user  system elapsed
##    0.65    0.00    0.67
```

```
  set.seed(100)
  system.time(for(i in 1:N) rmvnorm(n, mu, Sigma, method = "chol"))
```

```
##    user  system elapsed
##     0.5     0.0     0.5
```

```
  n <- 500
  d <- 400
  N <- 100
  mu <- numeric(d)
  Sigma <- cov(matrix(rnorm(n*d), n, d))
  set.seed(100)
  system.time(for(i in 1:N) mvrnorm(n, mu, Sigma))
```

```
##    user  system elapsed
##   15.26    0.21   15.46
```

```r
  cur.time <- Sys.time()
  ev <- eigen(Sigma, symmetric = TRUE)
  lambda <- ev$values
  V <- ev$vectors
  C <- V %*% diag(sqrt(lambda)) %*% t(V)
  Sys.time() - cur.time
```

```
## Time difference of 0.135 secs
```

```r
  set.seed(100)
  system.time(
  for(i in 1:N){
    Z <- matrix(rnorm(n*d), nrow = n, ncol = d)
    X <-  Z%*%t(C) + matrix(mu, n, d, byrow = TRUE)
  })
```

```
##    user  system elapsed
##    4.29    0.18    4.49
```