Generating Random Variables

Jonathan Navarrete

April 15, 2017

Introduction

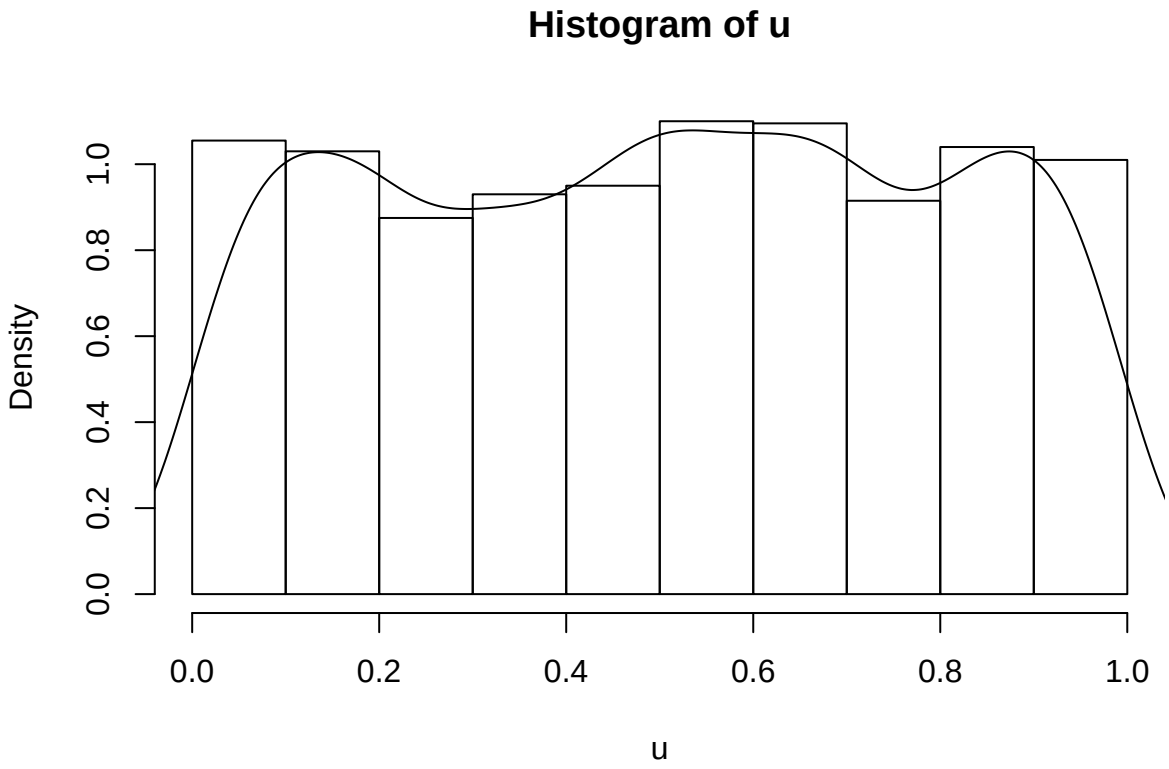
One of the fundamental tools required in monte carlo methods is the ability to generate pseudo-random variables from a specified probability distribution. We explore some of these methods starting with the Inverse Transform Method. One of the most fundamental number generation is the uniform random generation. Pseudo random number genrators rely on the assumption that computational methods can consistently generate uniform random numbers. Though, these notes do not dive deep into the mechanics of random number generation, they do provide an overview of some topics. After having the ability to generate simple random numbers transformations are used to simulate other random processes.

In this section we'll cover the following methods for generating random numbers from a target distribution.

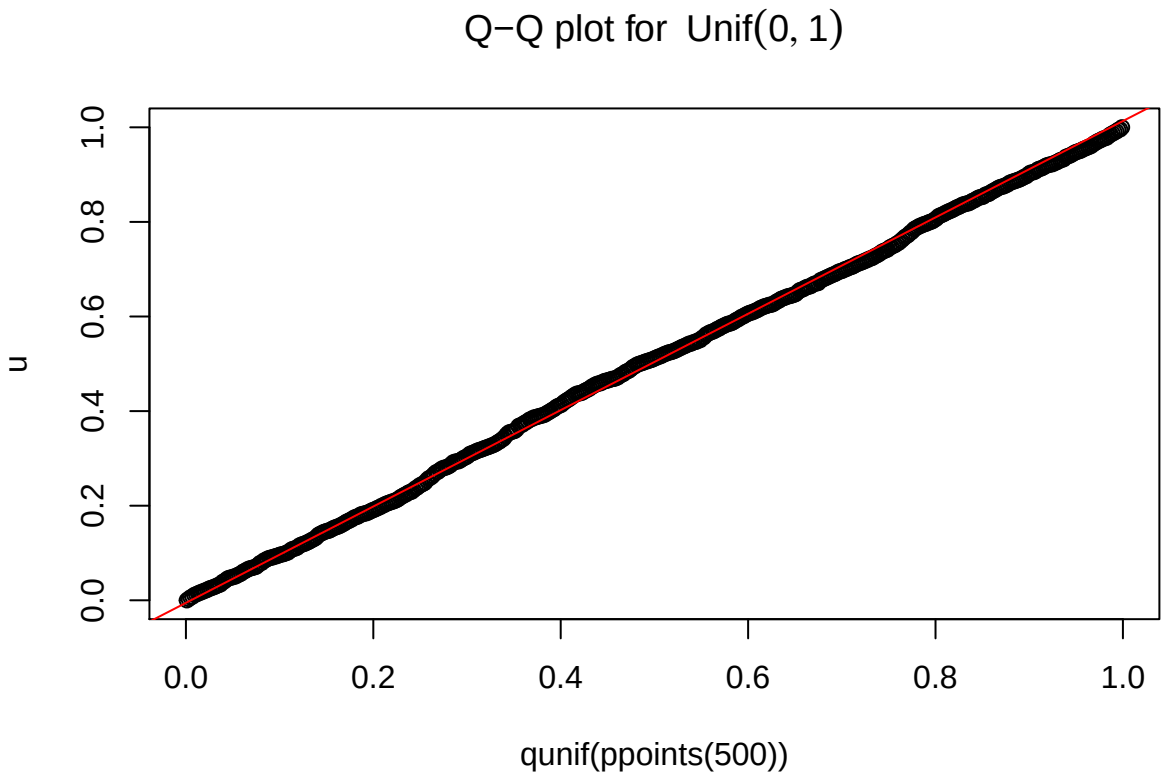
1. Inverse Transform Method
2. Accept-Reject Method
3. Transformation Method
4. Sums and Mixture distributions
5. Stochastic Processes

Generation uniform samples

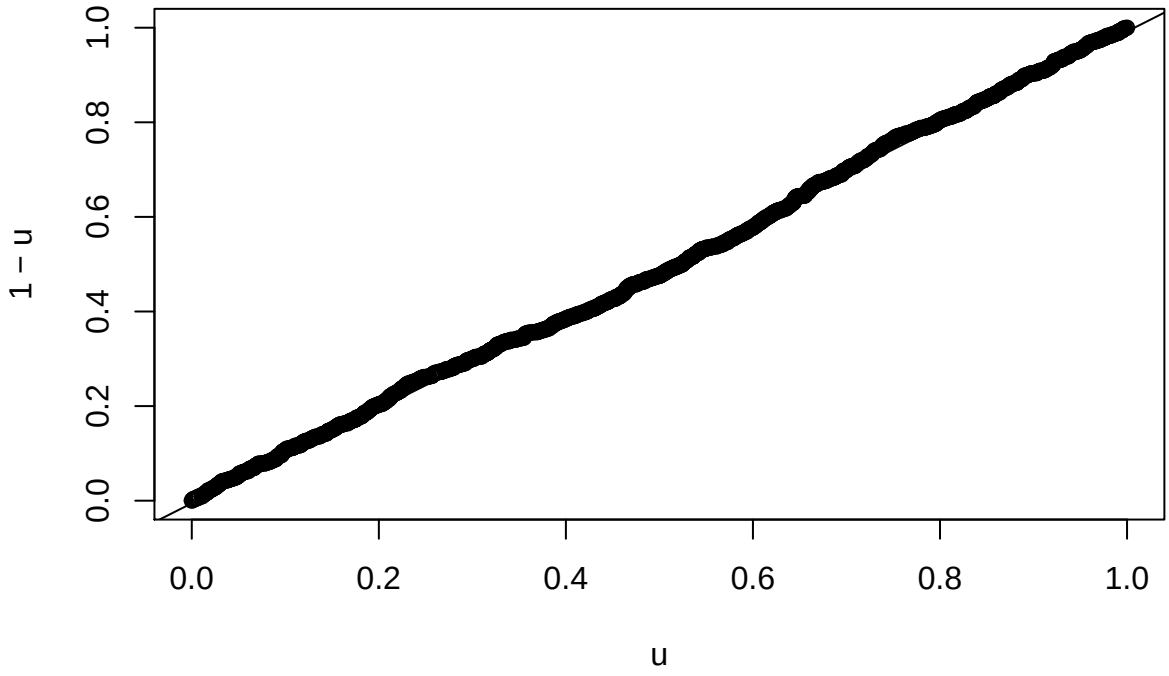
```
u = runif(2000)

#par(mfrow=c(1,2))
hist(u, probability=TRUE)
lines(density(u), xlim=c(0,1))
```



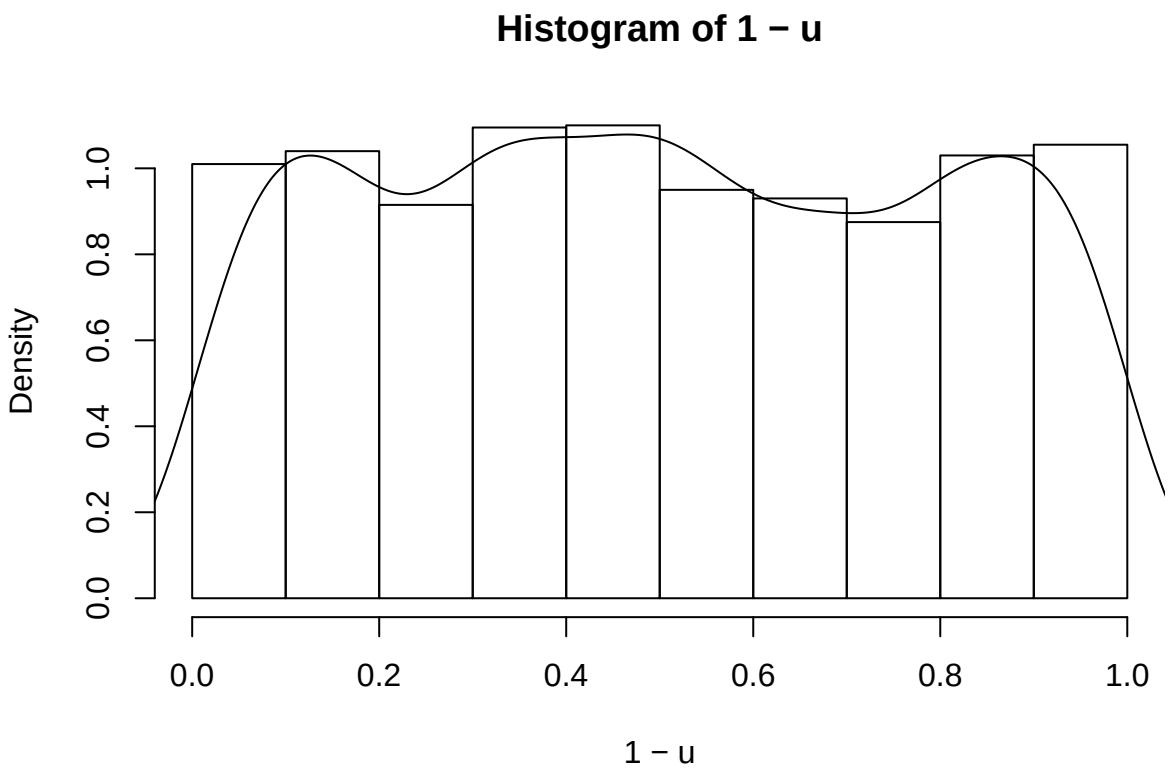

```
#par(mfrow=c(1,1))

## Q-Q plot for `runif` data against true theoretical distribution:
qqplot(qunif(ppoints(500)), u,
       main = expression("Q-Q plot for" ~~ {Unif(0,1)}))
qqline(u, distribution = qunif,
       prob = c(0.1, 0.6), col = 2)
```



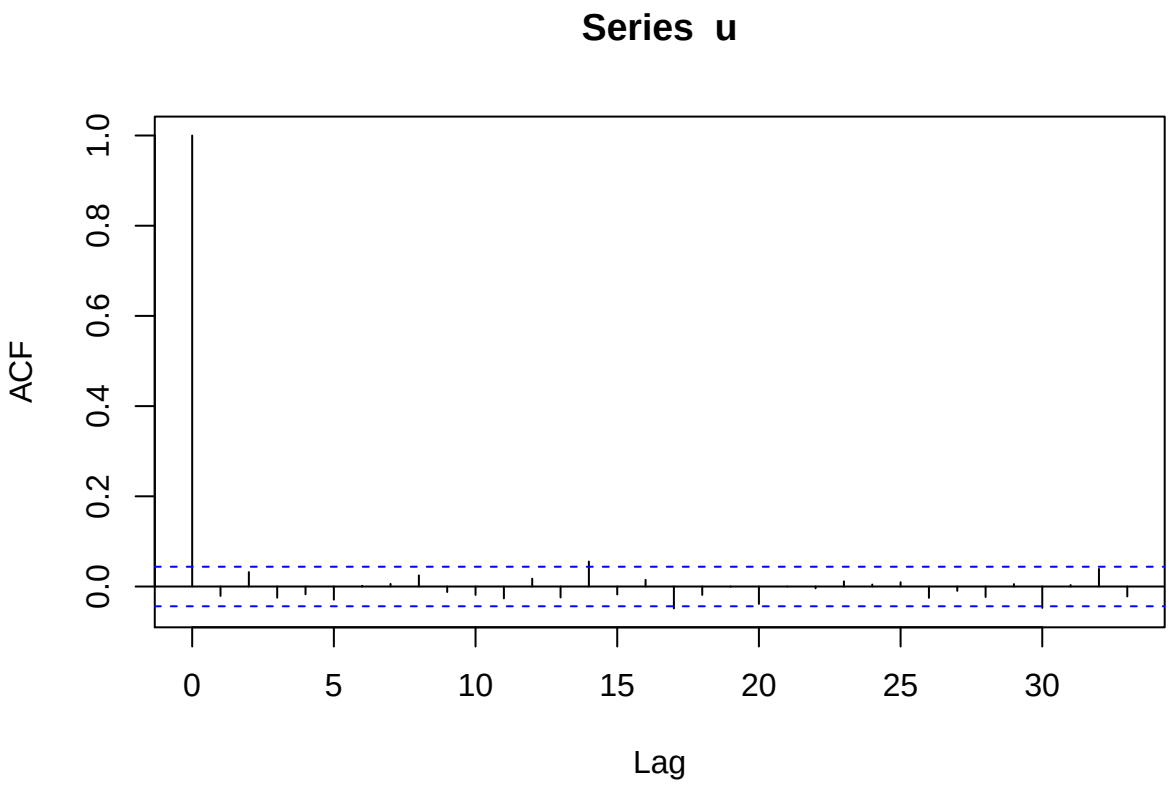

```
qqplot(u,1-u)
qqline(u, distribution = qunif)
```




```
hist(1-u, probability = TRUE) ## also follows a uniform dist
lines(density(1-u), xlim=c(0,1))
```

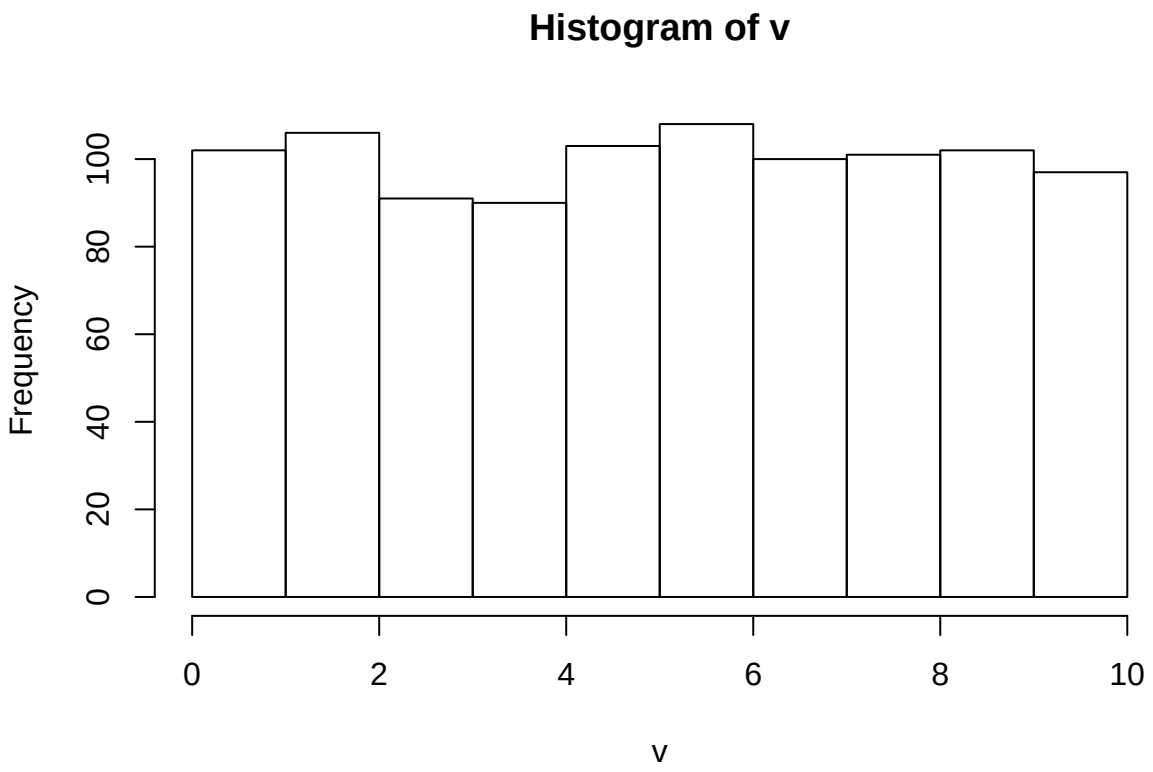



```
acf(u) ## autocorrelation of random number generation
```

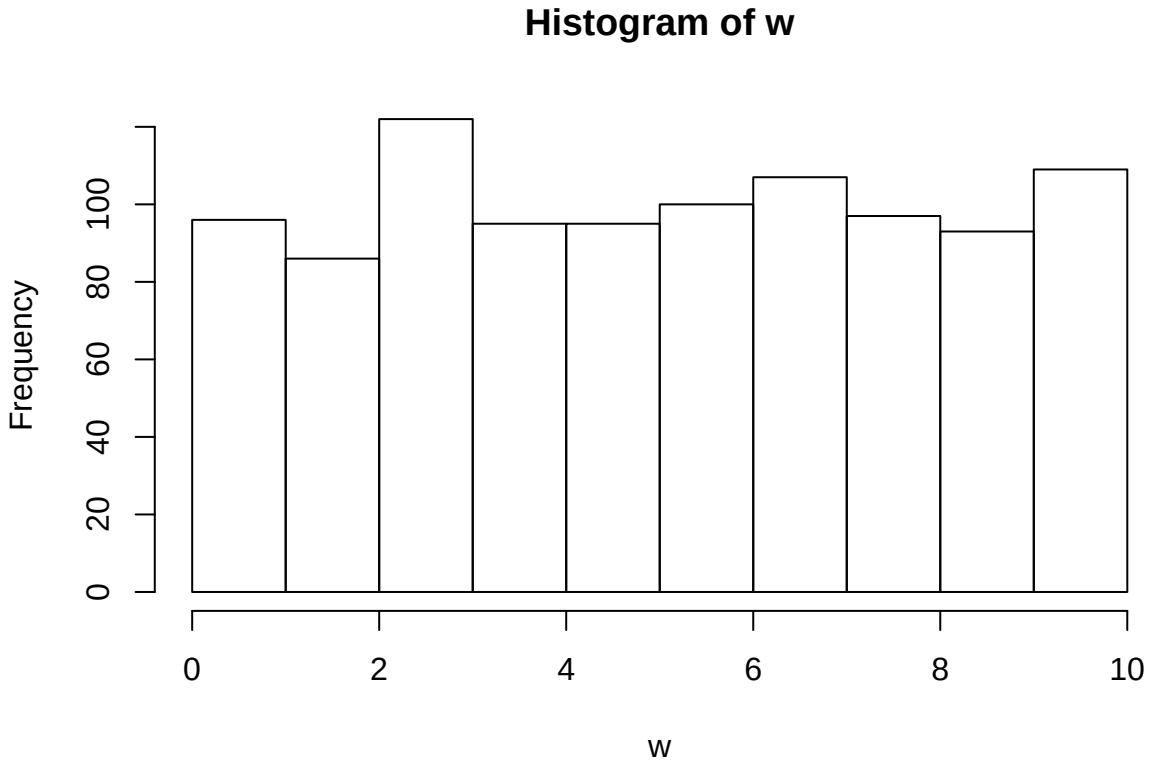


Now, say we could only simulate $u \sim Unif(0, 1)$. How could we simulate $v \sim Unif(0, 10)$? Well, we could simply include a multiplicative constant such that $v = 10 \times u$.

```r
v = 10 * runif(1000, 0, 1) ## v ~ unif(0, 10)
hist(v)
```




```r
w = runif(1000, 0, 10) ## unif(0, 10)
hist(w)
```



```r
## z = unif(0,8) + 8
z = runif(1000, 0, 8) + 2 ## v ~ unif(2, 10)
hist(z)
```

Inverse Transform Method

General idea: only using a uniform distribution, generate random values and use an inverse CDF of the target distribution for which you wish to simulate. See the following link for further discussion: How does the inverse transform method work?

Theorem (Probability Integral Transformation):

If X is a continuous random variable with CDF $F_X(X)$, then $U = F_X(X) \sim Uniform(0,1)$. If $U \sim Uniform(0,1)$, then for all $x \in \mathbb{R}$

$$P(F_X^{-1}(U) \leq x) = P(\inf\{t : F_X(t) = U\} \leq x)$$

$$= P(U \leq F_X(x))$$

$$= F_U(F_X(x))$$

$$= F_X(x)$$

and therefore $F_X^{-1}(U)$ has the same distribution as X.

Steps: 1. For target probability distribution function (pdf) f(x), calculate the inverse of the CDF by setting F(x) = U, then solving for X, for which $U \sim Unif(0,1)$.

2. Generate N random numbers from $U \sim Unif(0,1)$
3. Plug in u observed values in $F^{-1}(U)$ to obtain N x values for which $X \sim f(x)$

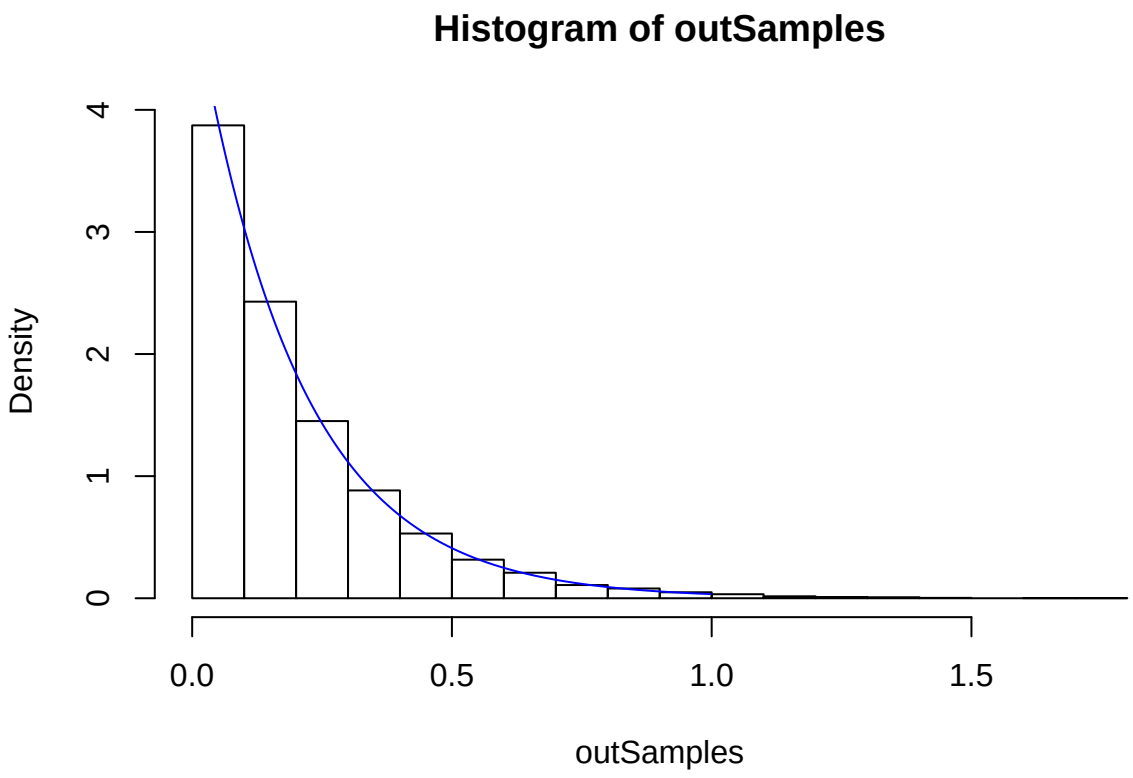
Example: Exponential distribution

Suppose we are interested in generating 10,000 random values from an Exponential distribution

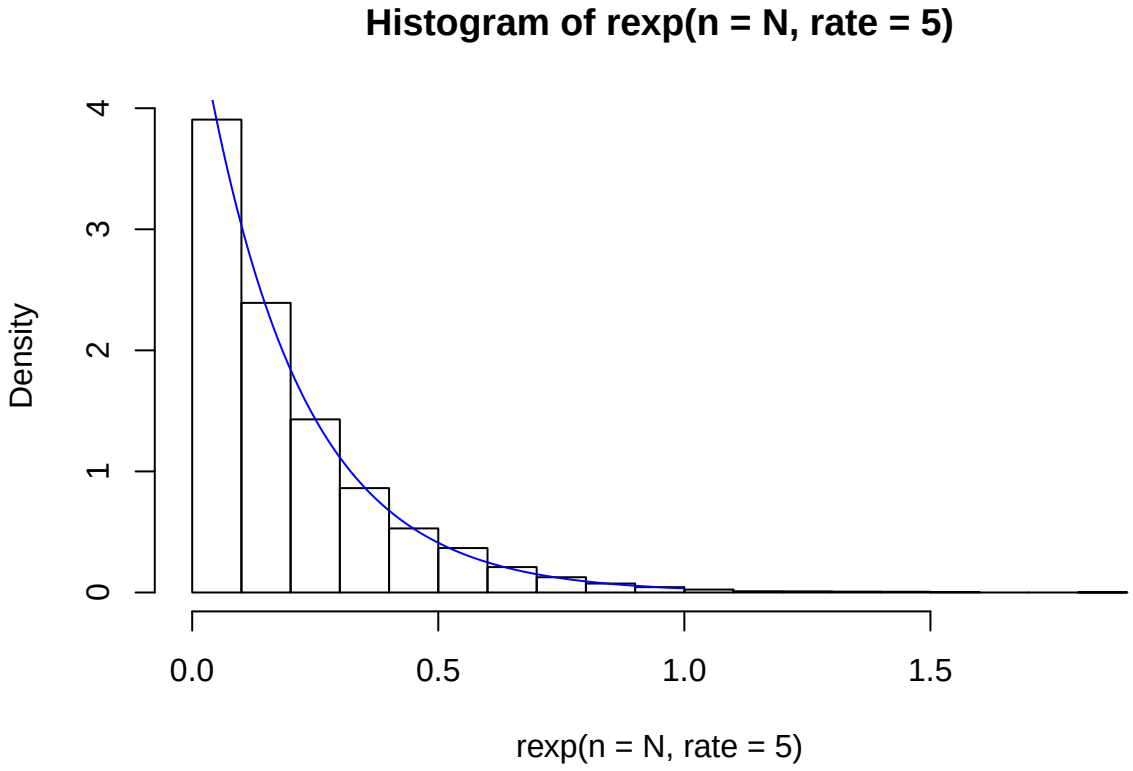
1. $f(X) = \lambda e^{-\lambda X}$
2. $F(X) = 1 - e^{-\lambda X} = U$
3. $F^{-1}(U) = -1/\lambda \; log(1-U)$; can use (1-u) or u, since both are uniformly distributed.

If we set $\lambda = 5$, then

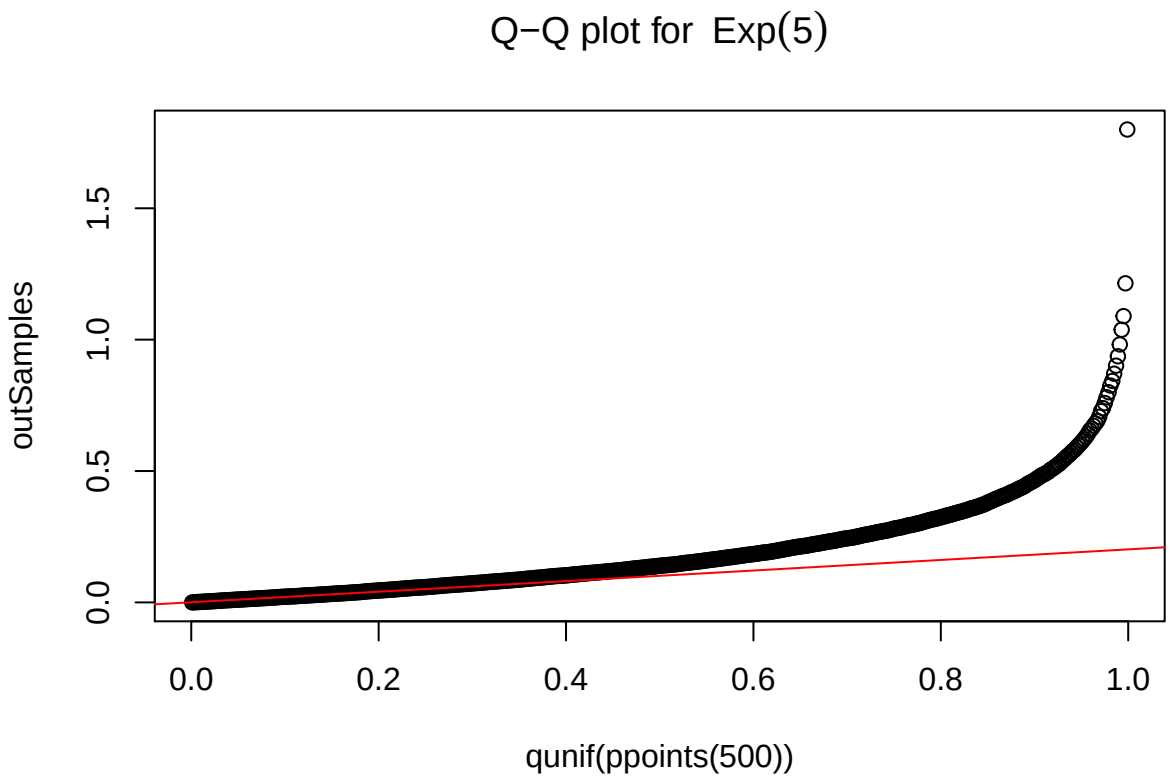
```
N = 10^4
u = runif(N)

fInv = function(u){
  (-1/5) * log(u) ## or log(1-u)
}

outSamples = fInv(u)


hist(outSamples, probability = TRUE)
lines(x = ppoints(200), y = dexp(x = ppoints(200), rate = 5),
      col = "blue")
```




```
hist(rexp(n = N, rate = 5), probability = TRUE)
lines(x = ppoints(200), y = dexp(x = ppoints(200), rate = 5),
      col = "blue")
```



```r
qqplot(qunif(ppoints(500)), outSamples,
      main = expression("Q-Q plot for" ~~ {Exp(5)}))
qqline(outSamples, distribution = qexp,
      prob = c(0.1, 0.9), col = 2)
```



Example: Pareto Distribution

For information on the Pareto distribution, please see: Pareto Distribution

The $Pareto(a, b)$ distribution has CDF $F(X \leq x) = 1 - (\frac{b}{x})^a$ for $x \geq b > 0,\ a > 0$

1. First set $F(x) = U$, where $U \sim Unif(0, 1)$

$$1 - (\frac{b}{x})^2 = 0$$

2. Solve for X

$$(\frac{b}{x})^a = 1 - U$$

3.

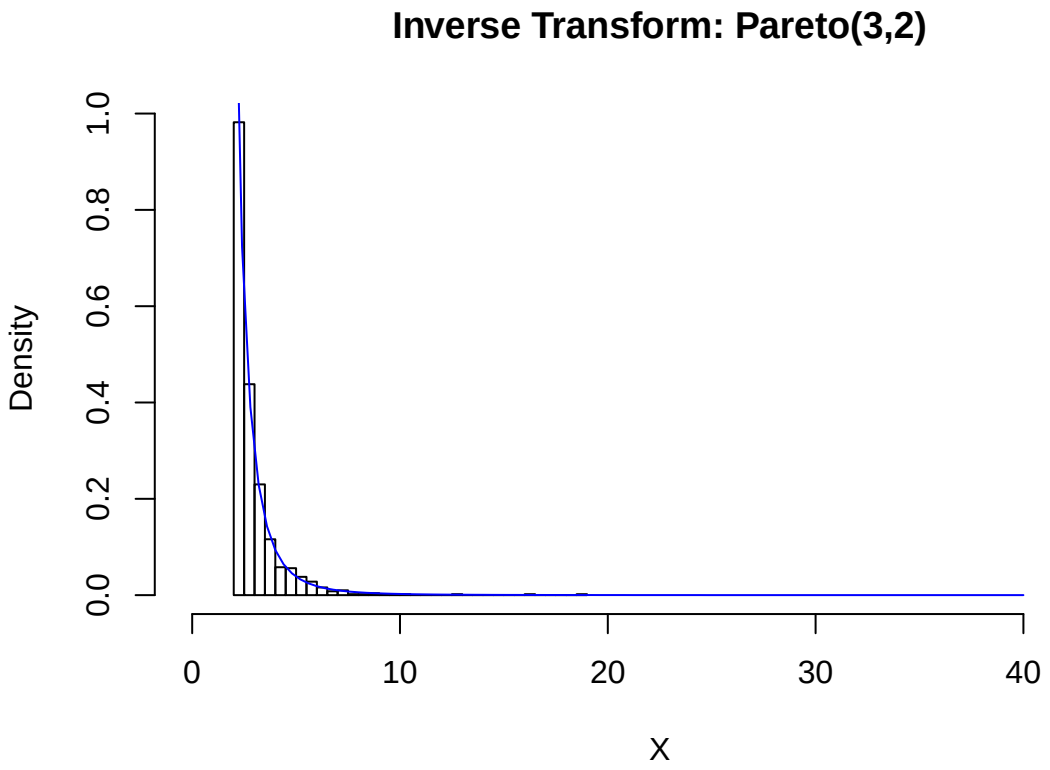
$$\frac{b}{x} = (1 - U)^{1/a}$$

4.

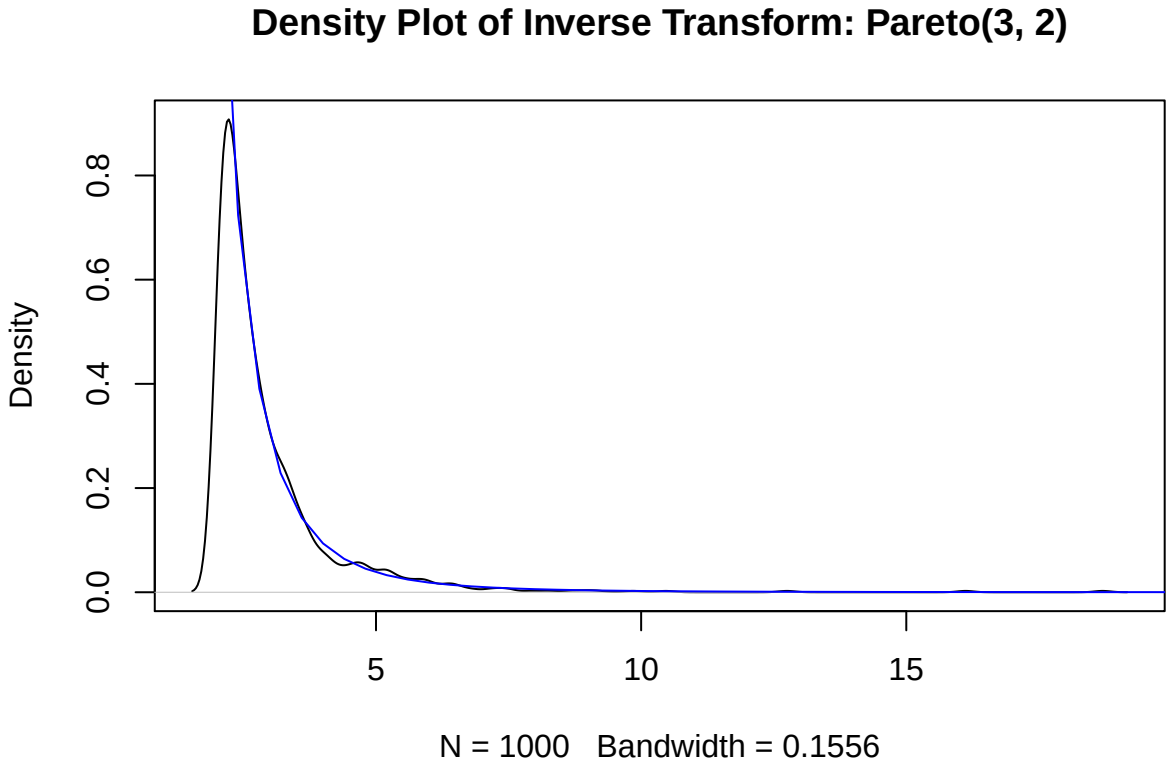
$$x = b \times (1 - U)^{-1/a}$$

```r
n = 1000

U =runif(n)
```

```r
a = 3
b = 2

X = b*(1-U)^(-1/a)

hist(X, probability = TRUE, breaks = 25, xlim =c(0, 45),main = "Inverse Transform: Pareto(3,2)")

pareto = function(x){(a*(b^a)/x^(a+1))}
curve(pareto(x), from = 0, to = 40, add = TRUE, col = "blue")
```




```r
plot(density(X), main = "Density Plot of Inverse Transform: Pareto(3, 2)")
curve(pareto(x), from = 0, to = 40, add = TRUE, col = "blue")
```



Inverse Transform Discrete scenario

For a given an ordered discrete random sample ... $< x_{i-1} < x_i < x_{i+1} <$ ... from a distribution $f(X)$, with CDF $F(x)$. Then, the inverse transformation $F_X^{-1}(u) = x_i$, where $F_X(x_{i-1}) < u \leq F_X(x_i)$. Then for each random variable desired,

1. Generate a random variable $u \sim Unif(0, 1)$
2. Deliver x_i where $F(x_{i-1}) < u \leq F(x_i)$

Example

Given the following distribution $P(X = 0) = 0.1$, $P(X = 1) = 0.2$, $P(X = 2) = 0.2$, $P(X = 3) = 0.2$, and $P(X = 4) = 0.3$, use the inverse transform method to generate a random sample of size 1000 from the distribution.

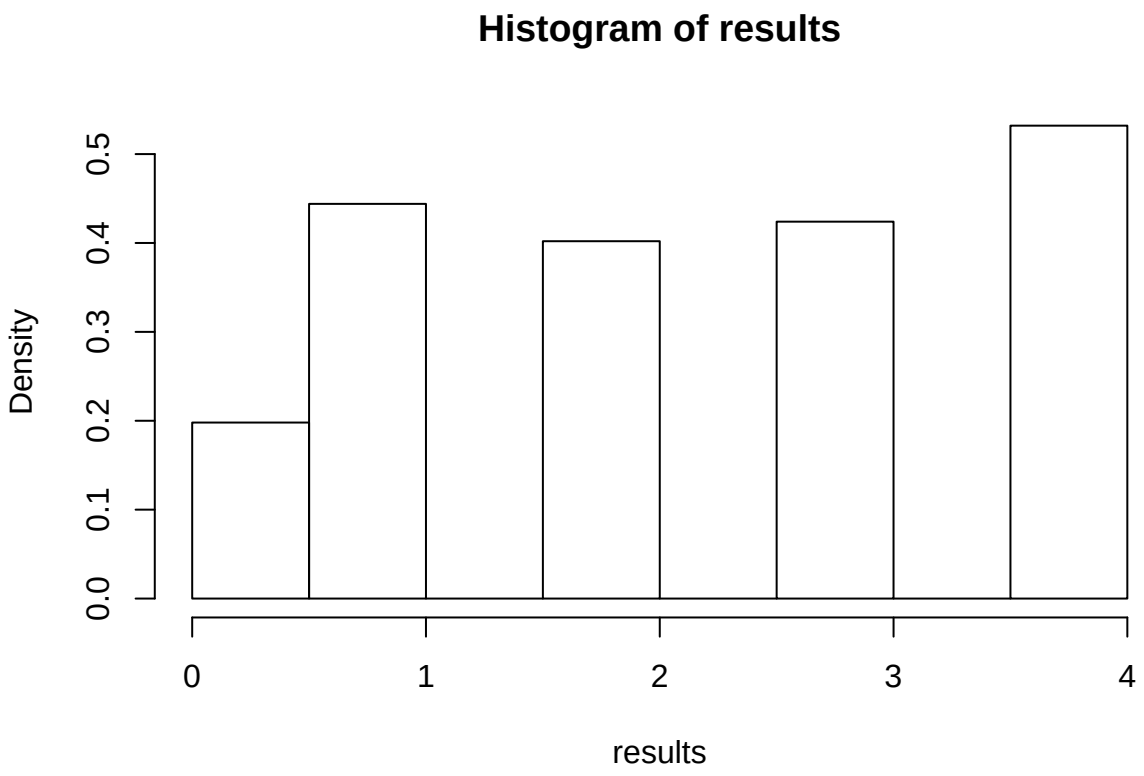
$$F(X \leq x) = \begin{cases} 0.1 & \text{if } x \leq 0 \\ 0.3 & \text{if } x \leq 1 \\ 0.5 & \text{if } x \leq 2 \\ 0.7 & \text{if } x \leq 3 \\ 1.0 & \text{if } x \leq 4 \end{cases}$$

```
cdf = c(0.0, 0.1, 0.3, 0.5, 0.7, 1.0)
results = numeric(length = 1000) ## creates a vector of zeros
u = runif(1000)
for(i in 2:6){
    ind = (cdf[i-1] < u) & (u <= cdf[i])
```

```
    results[ind] <- (i-2)
}

table(results) / 1000
```

```
## results
##     0     1     2     3     4
## 0.099 0.222 0.201 0.212 0.266
```

```
hist(results, probability = TRUE)
```



Accept-Reject

For notes on the Accept-Rejection algorithm see Accept-Reject

Suppose that X and Y are random variables with density (or pmf) f and g respectively, and there exists a constant M such that

$$\frac{f(t)}{g(t)} \leq M$$

for all t such that $f(t) > 0$. If we'd like to simulate from the target density $f(t)$, then the following algorithm can be applied to generate the random variable X.

The Accept-Reject Algorithm

1. Generate $Y \sim g_Y(t)$ and $U \sim Unif(0, 1)$
2. If $U \leq \frac{f(Y)}{M \times g(Y)}$ then we accept Y, such that $Y = X$
3. Repeat until you have sufficient samples

In order for the algorithm to work we require the following constraings:

1. f and g have to have compatible supports (i.e. $g(x) > 0$ when $f(x) > 0$)
2. There is a constant M such that $\frac{f(t)}{g(t)} \leq M$

Example: Beta(2,2)

Supposed we'd like to generate 1,000 samples from $Beta(2, 2)$, f. The density function for $Beta(2, 2)$ is simply $f(x) = 6x(1 - x)$ for $0 < x < 1$. Since our domain is between 0 and 1, we can use a simple $Unif(0, 1)$ density as our instrumental density, g. Then, by the accept-reject algorithm we can simulate a random variable $Y \sim g$, and a random variable $U \sim Unif(0, 1)$. Then, if

$$U \leq \frac{f(Y)}{M \times g(Y)}$$

we accept the candidate variable $Y \sim g$ as X, $X = Y$. Otherwise, we reject Y and simulate again until we get an appropriate sample size. Note that the target density f has a maximum of 1.5, so we can set M = 1.5; see: Max of Beta(2,2)

First, we'll generate 5 samples

```
## Accept-Reject
M = 1.5
X = rep(x = NA, 5) ## create a vector of length 5 of NAs
set.seed(123)
f <- function(x){ 6*x*(1 - x)} ## pdf of Beta(2,2)
g <- function(x){ 1 } ## pdf of Unif(0,1) is just 1

for(i in 1:5){
  print(paste("Run: ", i))
  u = runif(1)
  print(u)
  y = runif(1)
  print(y)
  accept <- u <= f(y)/(M* g(y))
```

```
  print(paste("Accept? ", accept))
  if(accept){
    X[i] <- y
  }

}
```

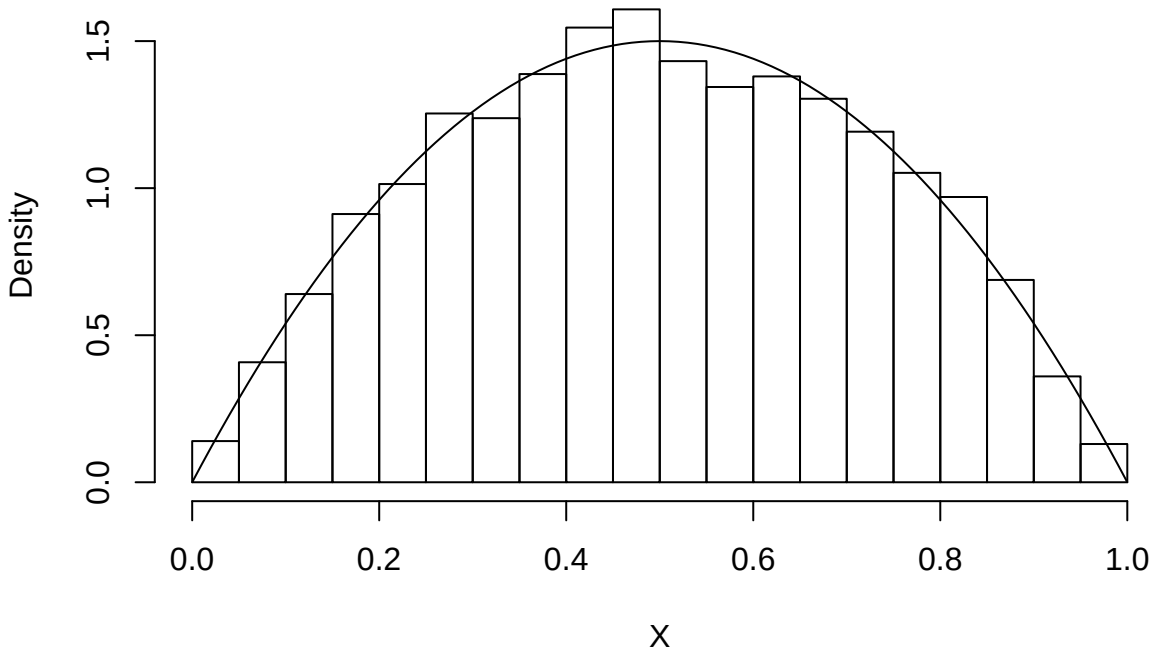
```
## [1] "Run:  1"
## [1] 0.2875775
## [1] 0.7883051
## [1] "Accept?  TRUE"
## [1] "Run:  2"
## [1] 0.4089769
## [1] 0.8830174
## [1] "Accept?  TRUE"
## [1] "Run:  3"
## [1] 0.9404673
## [1] 0.0455565
## [1] "Accept?  FALSE"
## [1] "Run:  4"
## [1] 0.5281055
## [1] 0.892419
## [1] "Accept?  FALSE"
## [1] "Run:  5"
## [1] 0.551435
## [1] 0.4566147
## [1] "Accept?  TRUE"
```

```
print(X)
```

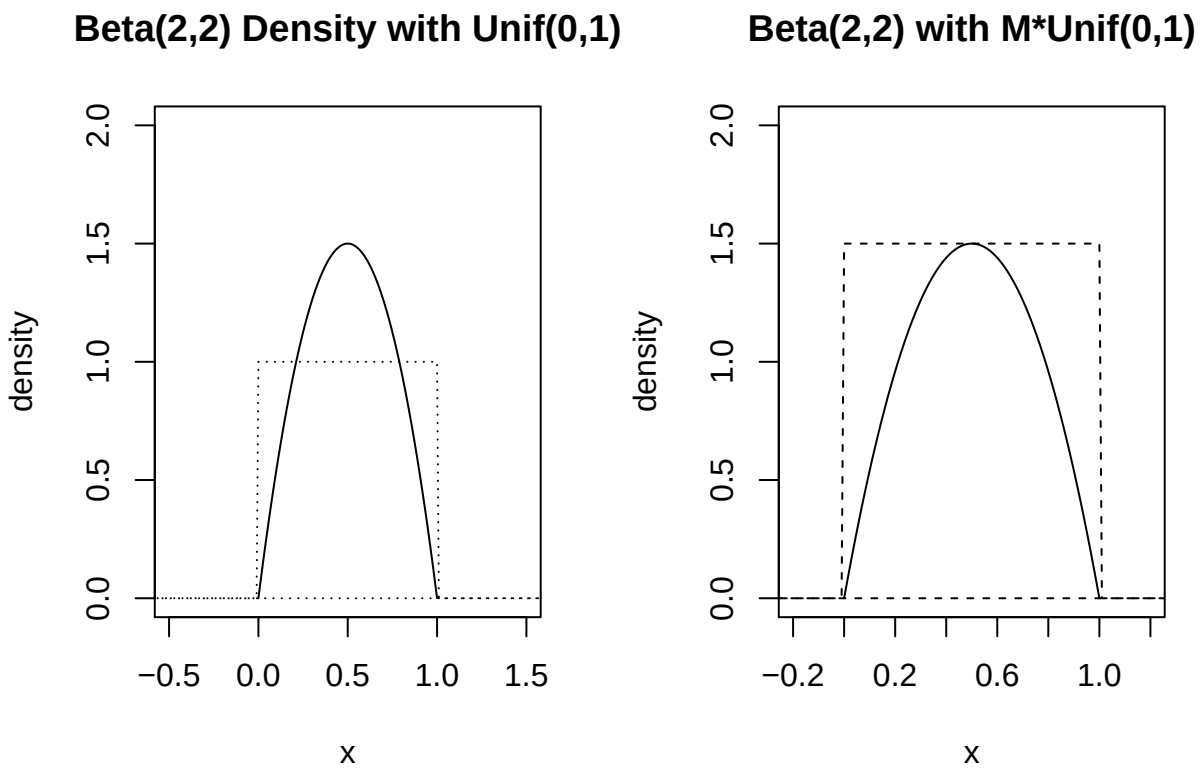
```
## [1] 0.7883051 0.8830174        NA        NA 0.4566147
```

Now, say we needed 10,000 samples from $Beta(2, 2)$, then a better implementation would be

```
X = rep(NA, 10000)
M = 1.5
i = 0 ## index set to start at 0
while(sum(is.na(X))){
  U = runif(1)
  Y = runif(1)
  accept <- U <= f(Y)/(M*g(Y))
  #print(paste("Accept? ", accept))
  if(accept){
    i = i+1 ## update the index
    X[i] <- Y
  }
}


hist(X, xlab = "X", main = "Beta(2,2) from Accept-Reject algorithm",
     probability = TRUE)
beta <- function(x) 6*x*(1-x)
curve(expr = beta, from = 0, to = 1, add = TRUE)
```




```
par(mfrow=c(1,2))
curve(expr = beta, from = 0, to = 1,
      xlim = c(-0.5, 1.5), ylim = c(0,2),
      main = "Beta(2,2) Density with Unif(0,1)",
      xlab = "x", ylab = "density")

x = seq(from = -1, to = 2, by = 0.01)
Unif1 = function(x){ ifelse(x >= 0 & x <= 1, 1, 0) }
polygon(x, Unif1(x), lty = 9)


curve(expr = beta, from = 0, to = 1,
      xlim = c(-0.2, 1.2), ylim = c(0,2),
      main = "Beta(2,2) with M*Unif(0,1)",
      xlab = "x", ylab = "density")

Unif2 = function(x){ ifelse(x >= 0 & x <= 1, 1*M, 0) }
polygon(x, Unif2(x), lty = 2)
```




```
#abline(h = 1.5, col = "red")
par(mfrow=c(1,1))
```

```
N = 10000
U = runif(N)
Y = runif(N)
M = 1.5

f <- function(x){ 6*x*(1 - x)} ## pdf of Beta(2,2)
g <- function(x){ 1 } ## pdf of Unif(0,1) is just 1

accept <- U*M < f(Y)/(g(Y))

mean(accept) ## acceptance rate
```

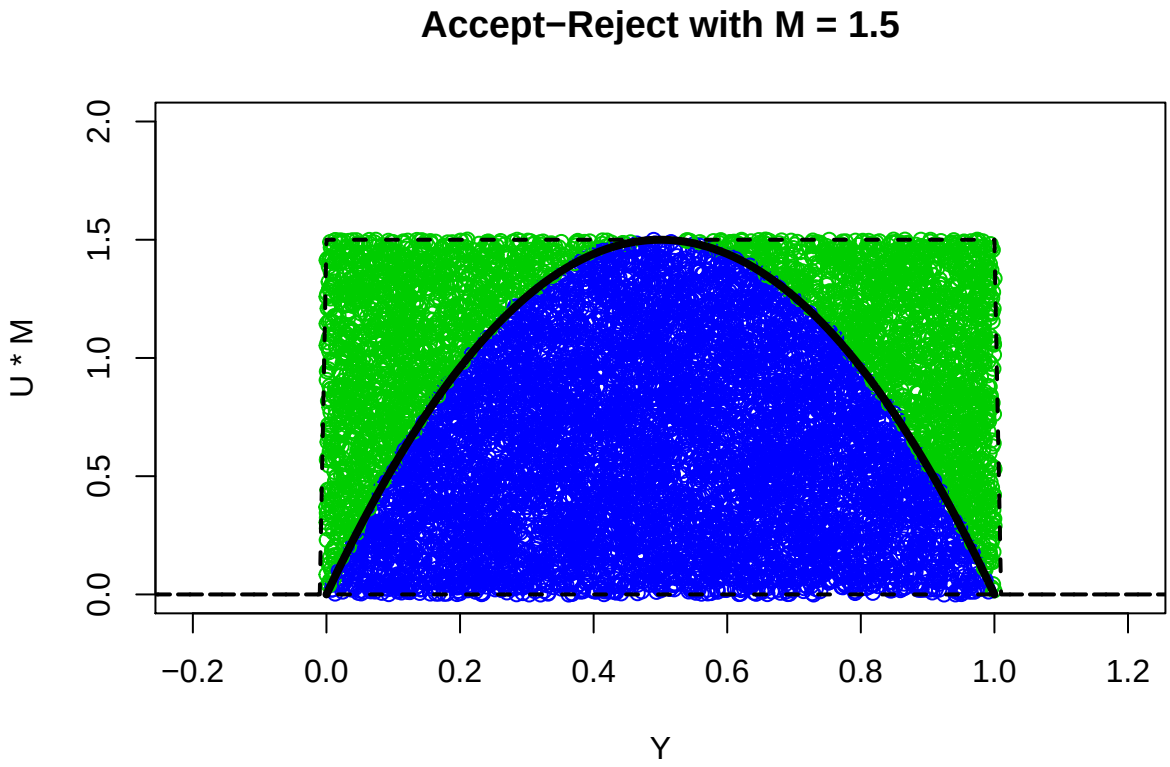
```
## [1] 0.6741
```

```
print(1/M) ## probability of acceptance
```

```
## [1] 0.6666667
```

```
plot(Y, U*M, col = as.numeric(accept)+3,
     xlim = c(-0.2, 1.2), ylim = c(0,2),
     main = "Accept-Reject with M = 1.5")



curve(expr = beta, from = 0, to = 1,
      xlim = c(-0.5, 1.5), ylim = c(0,2),
      main = "Beta(2,2) with M*Unif(0,1)",
      xlab = "x", ylab = "density", add = TRUE,
```

```
      lwd = 4)

Unif2 = function(x){ ifelse(x >= 0 & x <= 1, 1*M, 0) }
polygon(x, Unif2(x), lty = 2, lwd = 2)
```



It should be noted that the probability of acceptance is given by $\frac{1}{M}$. So, in order to make an efficient accept-reject algorithm, we should set M to be as high as needed, but no larger! As M increases, the probability of acceptance decreases, and this results in an increase in draws where we do not obtain samples from our target distribution f. This increases the computational cost.

Transformation Methods

Now we look at transformation methods for generating random variables. Distributions sometimes share relationships, and if the relationship is relatively simple, we can exploit this to generate random variables from a "simple to simulate" distribution and transform those random variables to generate samples from a "harder to simulate" distribution. I'll give some examples below. First, see the following Wikipedia page for some relationships: Relationships among probability distributions

Now that we've covered how to simulate random variables from an $Exp(\theta)$ distribution ("simple to simulate"), I'll cover how to generate some random variables from "harder to simulate" distributions.

We'll cover the following:

1. $Y = \sum_{i=1}^{N} X$ where $Y \sim Chi\ Square(2N)$ ($2N$ degress of freedom)

See: Chi-Square distribution

2. $Y = \beta \sum_{i=1}^{\alpha} X$ where $Y \sim Gamma(\alpha, \beta)$

See: Gamma distribution

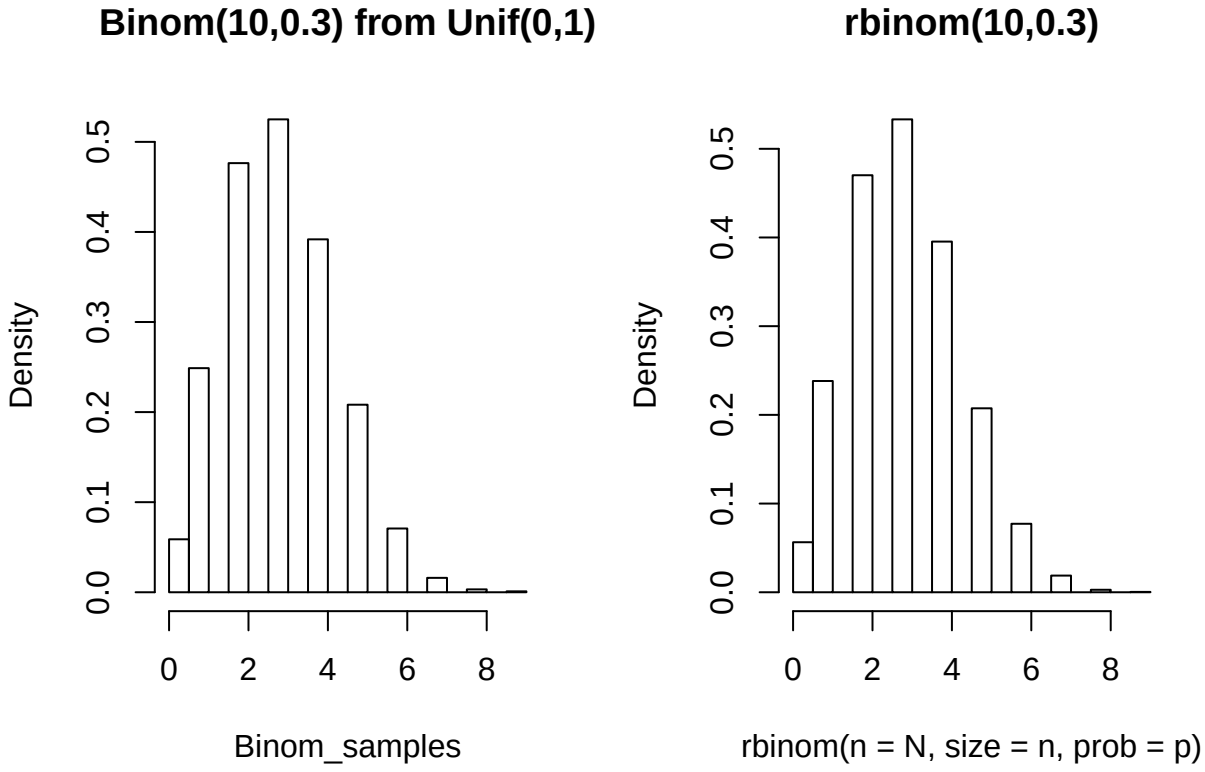
3. $Y = \frac{\sum_{i=1}^{a} X}{\sum_{i=1}^{a+b} X}$ where $Y \sim Beta(a, b)$

See: Beta distribution

4. $Y = \sum_{i=0}^{N} X_i$ where $X\ Bernoulli(p)$ such that $Y \sim Bin(n, p)$

Example: Binomial trials:

```
## let's simulate 10,000 samples from bin(n=10, p = 0.3)
N = 10000
n = 10
p = 0.3
x = as.numeric(runif(n = (n*N)) <= p) ## convert bools to 1 and 0
m = matrix(data = x, nrow = N, ncol = n)
Binom_samples = rowSums(m)

par(mfrow = c(1,2))
hist(Binom_samples, probability = TRUE, main = "Binom(10,0.3) from Unif(0,1)")
hist(rbinom(n = N, size = n, prob = p), probability = TRUE, main = "rbinom(10,0.3)")
```




```
par(mfrow=c(1,1))
```

Example: Generate R ~ Rayleigh

First attempt the Inverse Transform Method and see why it won't work.

For information on the Rayleigh distribution follow the link: Rayleigh Distribution

PDF: $f(x|\sigma) = \frac{x}{\sigma^2} exp(\frac{-x^2}{2\sigma^2})$ for $x \geq 0,\ \sigma > 0$

CDF: $F(X \leq x) = 1 - exp(\frac{-x^2}{2\sigma^2})$

Inverse Transform: Set $F(x) = U$, where $U \sim Unif(0, 1)$.

1. $$1 - exp(\frac{-x^2}{2\sigma^2}) = U$$
2. $$exp(\frac{-x^2}{2\sigma^2}) = 1 - U$$
3. $$log(exp(\frac{-x^2}{2\sigma^2})) = log(1 - U)$$
4. $$\frac{-x^2}{2\sigma^2} = log(1 - U)$$
5. $$-x^2 = 2\sigma^2 \times log(1 - U)$$

6.

$$x = \sqrt{-2\sigma^2 \times log(1-U)}$$

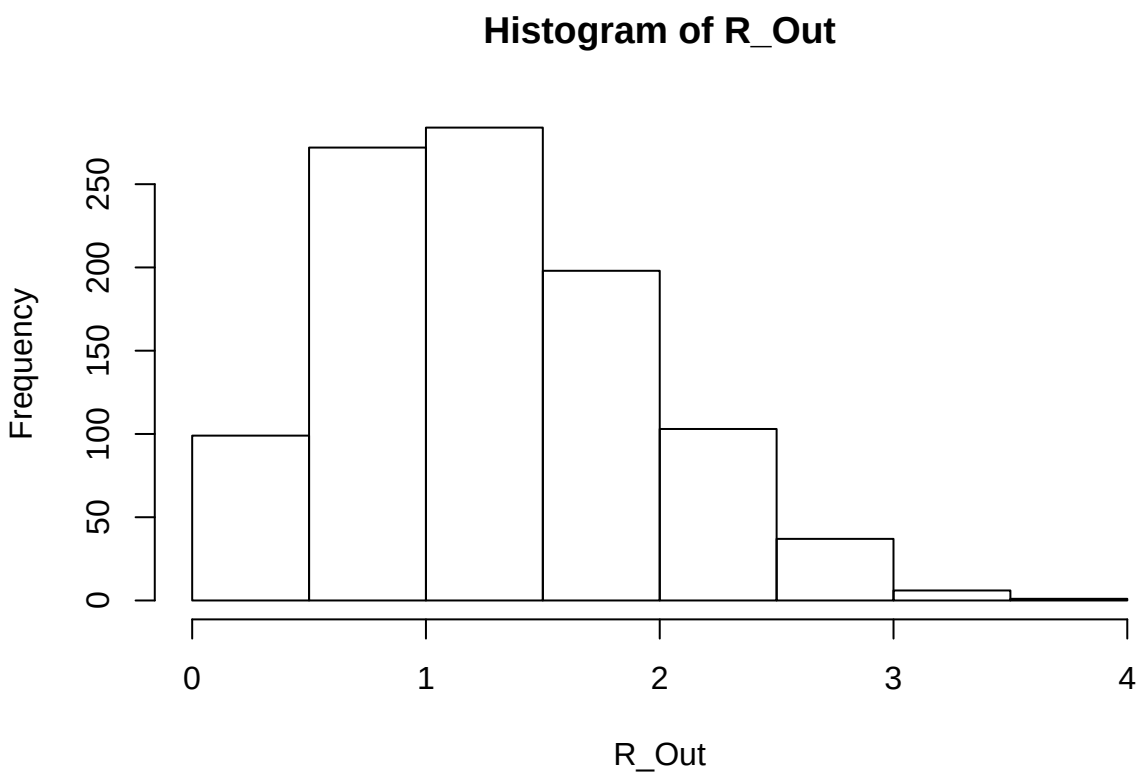
From the last equation, we see that we'd be taking the square root of negative values which would be problematic. Therefore, we need an alternative algorithm.

From information on the Rayleigh distribution, we know that given two i.i.d. random variables $Z_1, Z_2 \sim N(0, \sigma)$ then $R = \sqrt{Z_1^2 + Z_2^2} \sim Rayleigh(\sigma)$. Therefore, in order to simulate 1 random variable from $Rayleigh(\sigma)$, we first generate 2 random variables from a Normal distribution with mean 0 and standard deviation σ.

To generate N Rayleigh random variables, our algorithm would be:

1. Generate $2 \times N$ random variables $Z_i \sim N(0, \sigma)$ for $i \in (0, 2N)$
2. For each pair of $Z_i \sim N(0, \sigma)$ use the transformation $R = \sqrt{Z_1^2 + Z_2^2}$ to obtain N random variables from $Rayleigh(\sigma)$.

```
N = 1000

Z = rnorm(n = 2*N, mean = 0, sd = 1)
Z = matrix(data = Z, nrow = N, ncol = 2)

transfromation <- function(vec){
  R = sqrt(sum(vec^2))
  #R = sqrt(vec[1]^2 + vec[2]^2)
  return(R)
}


R_Out = apply(X = Z, MARGIN = 1, FUN = transfromation)

hist(R_Out)
```



```r
## compare with Rayleigh {VGAM}

sqrt(pi/2) ## theoretical mean
```

```
## [1] 1.253314
```

```r
mean(R_Out) ## calculated mean
```

```
## [1] 1.281112
```