# Generating Random Variables

Jonathan Navarrete

## Introduction

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

1. Uniform random samples
2. Transformation Method
3. Inverse Transform Method
4. Accept-Reject Method

## Pseudo-random samples

One of the requirements for Monte Carlo methods is the ability to generate consistent *pseudo*-random samples from a specified probability distribution. One of the most fundamental tools is a uniform random number generater

- Applying transformations to uniform samples, we can sample from many different distributions
- Computers on their own can't generate *randomness*, but they can simulate what *random* may look like

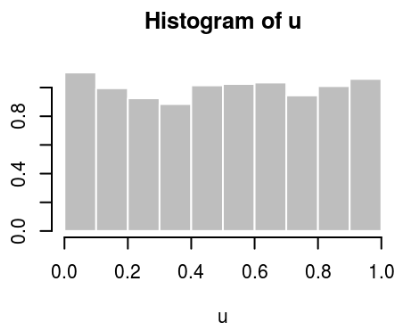
In R, we can control how random samples are generated. For example, we can specify seed for random number generation, see the function `set.seed` (https://stat.ethz.ch/R-manual/R-devel/library/base/html/Random.html)

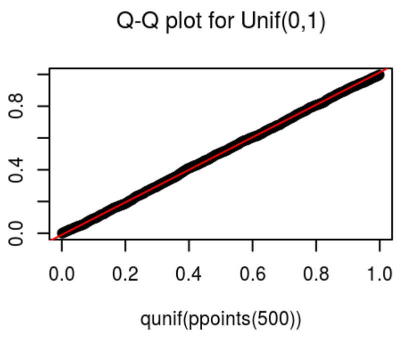
- `set.seed` allows us to control the start and flow (*state*) of random number generation in a program
- Useful for reproducibility
- On StackOverflow: Reasons for using the set.seed function (https://stackoverflow.com/questions/13605271/reasons-for-using-the-set-seed-function)

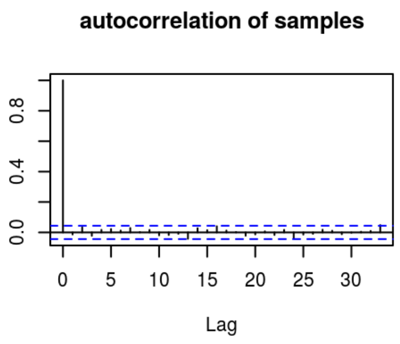
# Generation uniform samples

Random number generation for statistical purposes heavily relies on the assumption that computational methods can consistently generate *independent* uniform random numbers.

```
u = runif(2000) ## generate a sample of unif(0,1) samples

hist(u, probability=TRUE, col = "gray", border = "white") ## plot histogram
## Q-Q plot for `runif` data against true theoretical distribution:
qqplot(x = qunif(ppoints(500)), y = u,  main = expression("Q-Q plot for Unif(0,1)"))
qqline(y = u, distribution = qunif, prob = c(0.1, 0.6), col = 2)

acf(x = u, main = "autocorrelation of samples")  ## autocorrelation function
```







4/40

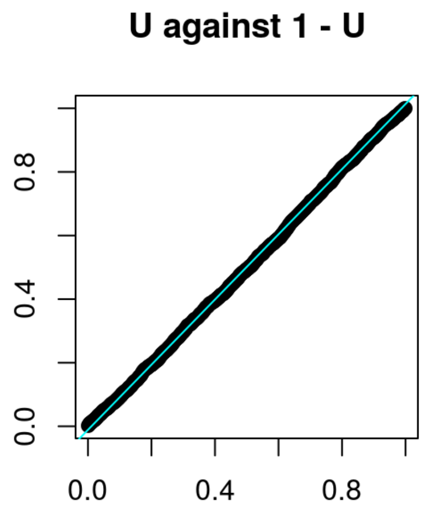
## Generation uniform samples

### References

- QQ Plot (http://www.itl.nist.gov/div898/handbook/eda/section3/qqplot.htm)
- Autocorrelation plot (http://www.itl.nist.gov/div898/handbook/eda/section3/autocopl.htm)

## Generation of uniform samples

For a random variable $u \sim Unif(0,1)$, then $(1-u) \sim Unif(0,1)$



```
qqplot(qunif(ppoints(500)),1-u, main = "U against 1 - U")
qqline(u, distribution = qunif, col = 2)
```

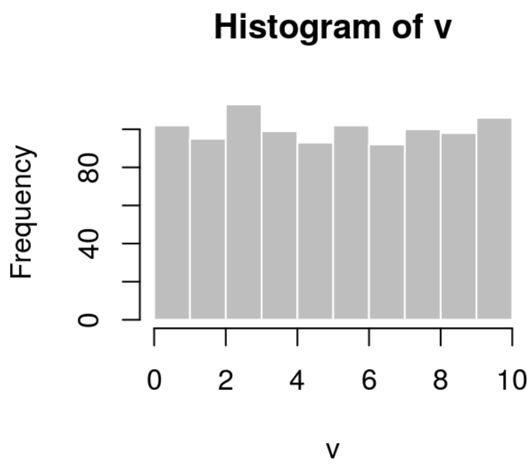
It is important to generate samples that are not correlated with each other!

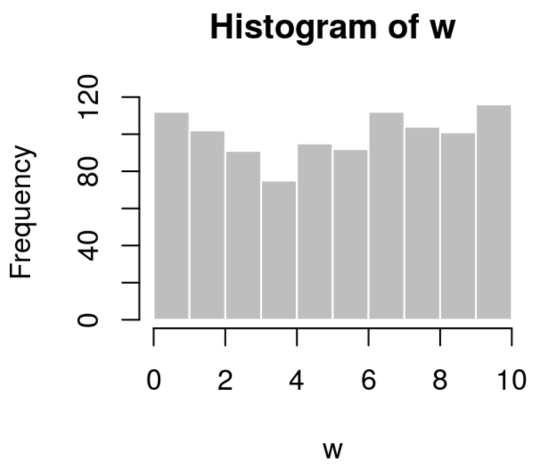
6/40

## Generation of uniform samples, pt. 2

Once we can simulate $u \sim Unif(0,1)$, we can begin generating samples from other target distributions. How could we simulate $v \sim Unif(0,10)$? Well, we could simply include a multiplicative constant such that $v = 10 \times u$.

```
v = 10 * runif(1000, 0, 1) ## v ~ unif(0, 10)
hist(v, col = "gray", border = "white")
w = runif(1000, 0, 10) ## unif(0, 10)
hist(w, col = "gray", border = "white")
```





7/40

## Transformation Methods

- Distributions sometimes share direct relationships.
- If the relationship between the instrumental and target distribution is simple, then we can exploit it to generate random variables from the target distribution
- We can sample from an "easy to simulate" distribution and then transform those random samples to generate samples from a "difficult to simulate" distribution.

To review some common relationships see the following: Relationships among probability distributions (https://en.wikipedia.org/wiki/Relationships_among_probability_distributions)

- Some easy *instrumental* distributions which are easy to sample from include $Unif(a, b)$, $Exp(\theta)$.
- Some distributions that are more difficult to sample from include the $N(\mu, \sigma)$ and **MVN** distributions.

8/40

## Example: Binomial samples

Another use of generating samples $U \sim U(0,1)$ is to generate Bernoulli. Once we've generated $n$ Bernoulli samples, we can sum the Bernouilli samples to generate Binomial samples. Reference: Notes (http://www.stat.ufl.edu/~abhisheksaha/sta4321/lect12.pdf)

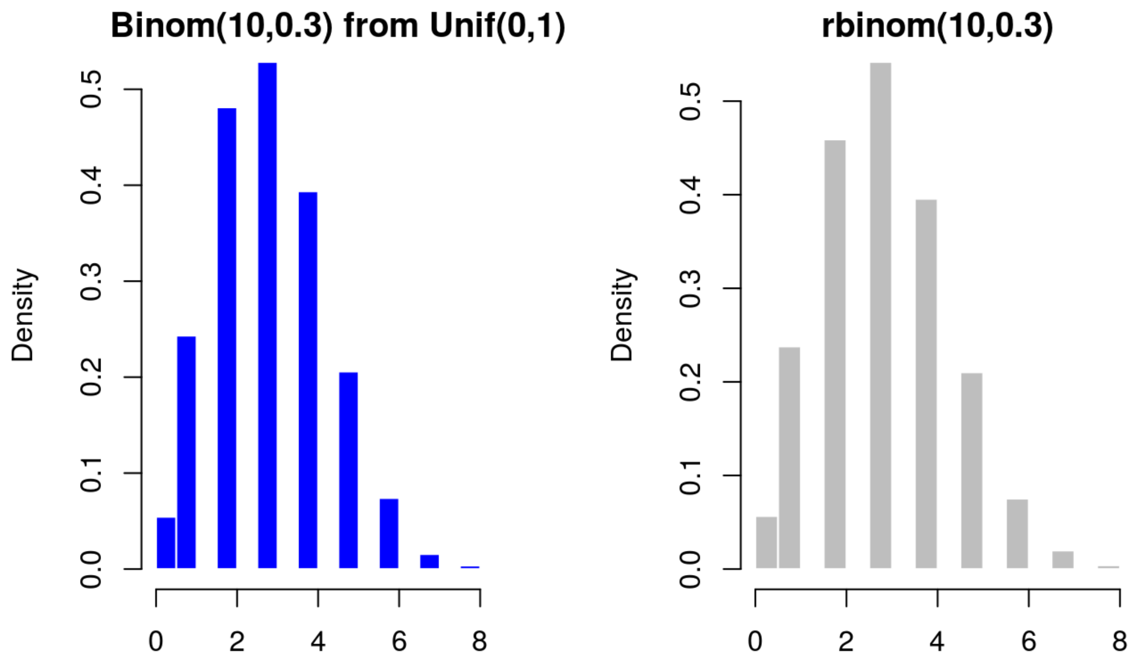
$$u \sim Unif(0,1)$$
$$x = I(u \leq p) \text{ such that } x \sim Bernoulli(p)$$
$$y = \sum_{i=1}^{n} x_i \text{ such that } y \sim Bin(n,p)$$

```
## let's simulate 10,000 samples from bin(n=10, p = 0.3)
N = 10000 ## number of Unif and Binomial samples
n = 10 ## number of bernoulli trials for each bernoulli sample
p = 0.3 ## theoretical p
u =  runif(n = (n*N))
x = as.numeric(u <= p) ## convert bools to 1 and 0; Bernoulli samples
m = matrix(data = x , nrow = N, ncol = n)
Binom_samples = rowSums(m) ## binomial samples
```

9/40

## Example: Binomial samples, pt. 2

```
par(mfrow = c(1,2), mar = c(3, 4, 1, 2))
hist(Binom_samples, probability = TRUE, main = "Binom(10,0.3) from Unif(0,1)",
     col = "blue", border = "white")
hist(rbinom(n = N, size = n, prob = p), probability = TRUE, main = "rbinom(10,0.3)",
     col = "gray", border = "white")
```



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

## Inverse Transform Method

The inverse transform method is a simple algorithm for generating random variables $x$ from a *continuous* target distribution $f(x)$ using random samples from a $Unif(0,1)$ distribution.

General idea: evaluate random variables $u \sim Unif(0,1)$ using the inverse CDF of the target distribution for which you wish to simulate from. See the following link for further discussion: How does the inverse transform method work? (https://stats.stackexchange.com/questions/184325/how-does-the-inverse-transform-method-work)

**Theorem (Probability Integral Transformation):** If $X$ is a *continuous* random variable with CDF $F_X(X)$, then $U = F_X(X) \sim Unif(0,1)$. If $U \sim Unif(0,1)$, then for all $x \in \mathbb{R}$

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

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

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

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

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

11/40

## Inverse Transform Method, pt. 2

Algorithm:

1. For target probability density function (*pdf*) $f(X)$, calculate the CDF, $F(X)$
2. Set the CDF equal to $U$, $F(X) = U$, then solving for $X$, obtaining $F^{-1}(U) = X$
3. Generate $n$ random variables from $u \sim Unif(0, 1)$
4. Plug in $u$ observed values in $F^{-1}(U = u)$ to obtain $n$ values for which $x \sim f(X)$

12/40

## Example: Exponential distribution

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

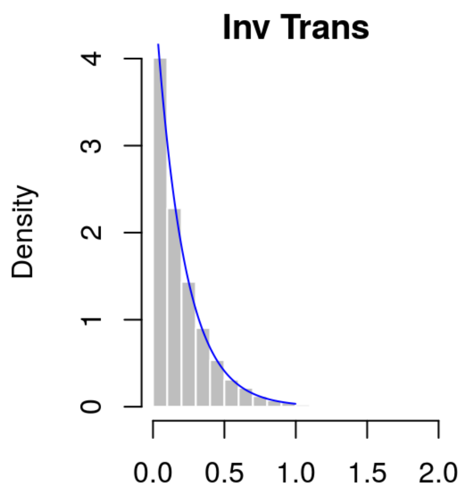
1. $f(X) = \lambda e^{-\lambda X}$
2. $F(X) = 1 - e^{-\lambda X} = U$
3. $F^{-1}(U) = -1/\lambda \; log(1-U)$; can use $(1-u)$ or $u$, since both are uniformly distributed.

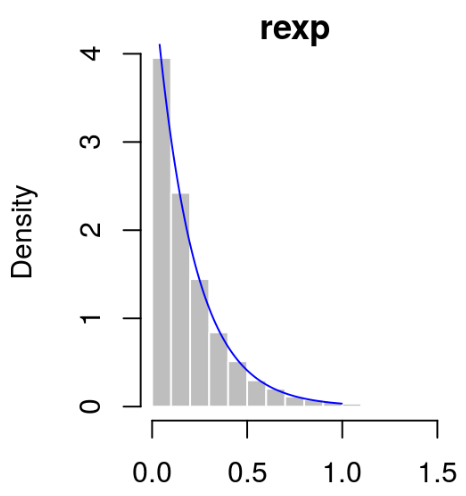
If we set $\lambda = 5$, then

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

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

outSamples = fInv(u)
```

# Example: Exponential distribution, pt. 2

```
hist(outSamples, probability = TRUE, main = "Inv Trans", col = "gray", border = "white")
lines(x = ppoints(200), y = dexp(x = ppoints(200), rate = 5),
      col = "blue")
hist(rexp(n = N, rate = 5), probability = TRUE, main = "rexp", border = "gray")
lines(x = ppoints(200), y = dexp(x = ppoints(200), rate = 5),
      col = "blue")
```





## Example: Pareto Distribution

For information on the Pareto distribution, please see: Pareto Distribution (http://www.math.uah.edu/stat/special/Pareto.html)

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 $U \sim Unif(0,1)$, then solve for $X$

$$\begin{aligned} 1 - \left(\frac{b}{x}\right)^2 &= U \\ \left(\frac{b}{x}\right)^a &= 1 - U \\ \frac{b}{x} &= (1-U)^{1/a} \\ x &= b \times (1-U)^{-1/a} \\ &= F_X^{-1}(U) \end{aligned}$$

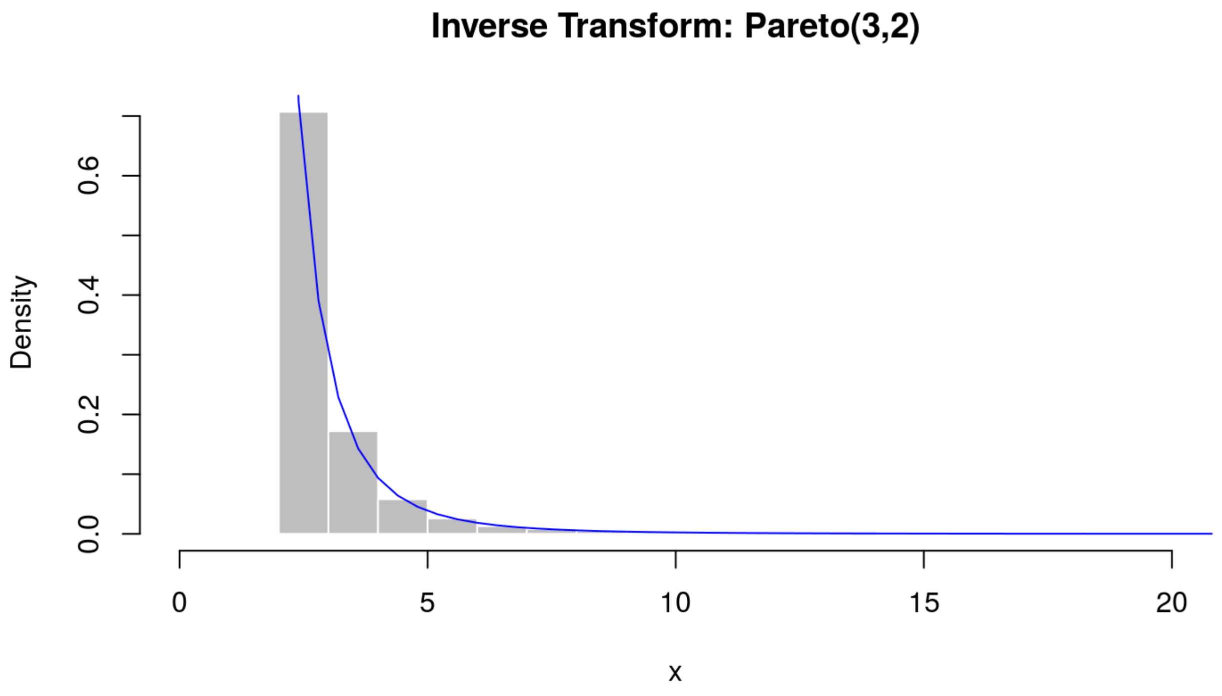
## Example: Pareto Distribution, pt. 2

```
set.seed(123)
n = 1000
U =runif(n)
a = 3
b = 2
X = b*(1-U)^(-1/a)
pareto = function(x){(a*(b^a)/x^(a+1))}

summary(X)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.000   2.205   2.503   2.969   3.161  23.773
```

## Example: Pareto Distribution, pt. 3

```
hist(X, probability = TRUE, breaks = 25, xlim =c(0, 20),
     col = "gray", border = "white",
     main = "Inverse Transform: Pareto(3,2)", xlab = "x")
curve(pareto(x), from = 0, to = 40, add = TRUE, col = "blue")
```



17/40

## Inverse Transform Discrete scenario

For a given ordered discrete random sample $x_1 < \ldots < x_{i-1} < x_i < x_{i+1} < \ldots < x_n$ 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 $F(x_{i-1}) < u \leq F(x_i)$

As an example, take 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.

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

18/40

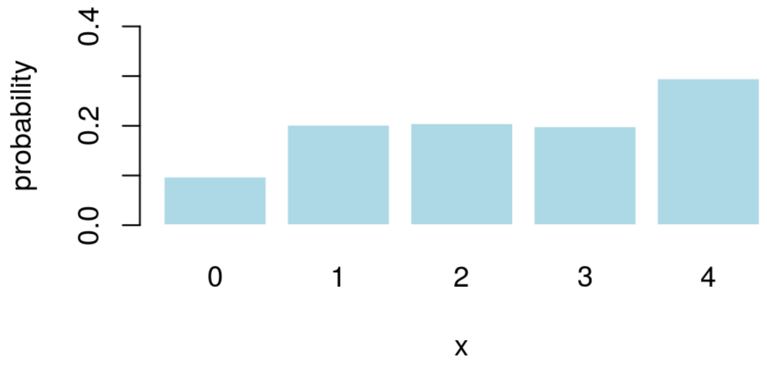
## Inverse Transform Discrete scenario, pt. 2

```
n = 10000
u = runif(n)

fInv <- function(u){
  if(u <= 0.1) x <- 0
  if(u > 0.1 && u <= 0.3) x <- 1
  if(u > 0.3 && u <= 0.5) x <- 2
  if(u > 0.5 && u <= 0.7) x <- 3
  if(u > 0.7 && u <= 1) x <- 4
  return(x)
}

results = sapply(X = u, fInv)
table(results) / n
```

```
## results
##      0      1      2      3      4
## 0.0979 0.2022 0.2053 0.1991 0.2955
```

## Inverse Transform Discrete scenario, pt. 3

```
z = table(results) / n
barplot(z, border = "white", col = "lightblue", xlab = "x", ylab = "probability", ylim = c(0, 0.4))
```



$P(X = 0) = 0.1$, $P(X = 1) = 0.2$, $P(X = 2) = 0.2$, $P(X = 3) = 0.2$, and $P(X = 4) = 0.3$

## Accept-Reject

For notes on the Accept-Rejection algorithm see Accept-Reject (http://www.columbia.edu/~ks20/4703-Sigman/4703-07-Notes-ARM.pdf)

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

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

or

$$M \times g(t) \geq f(t)$$

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

To generate $n$ samples, `for(i in 1:n)`

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:

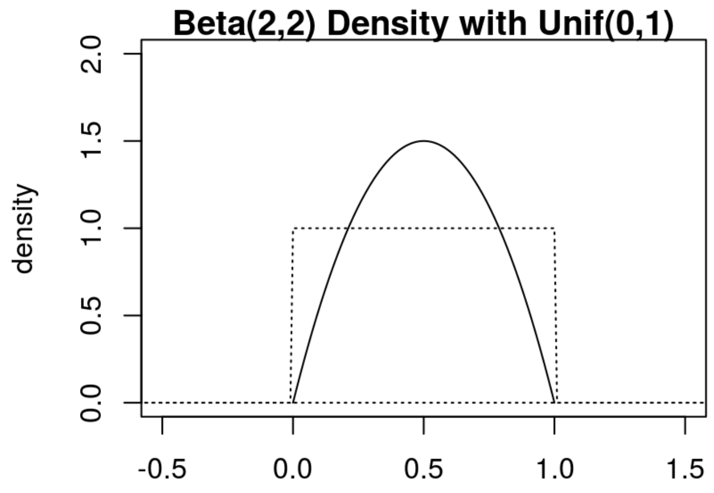
1. $f$ and $g$ have to have compatible supports (i.e. $g(x) > 0$ when $f(x) > 0$)
2. There is a constant $M$ such that $\frac{f(t)}{g(t)} \leq M$

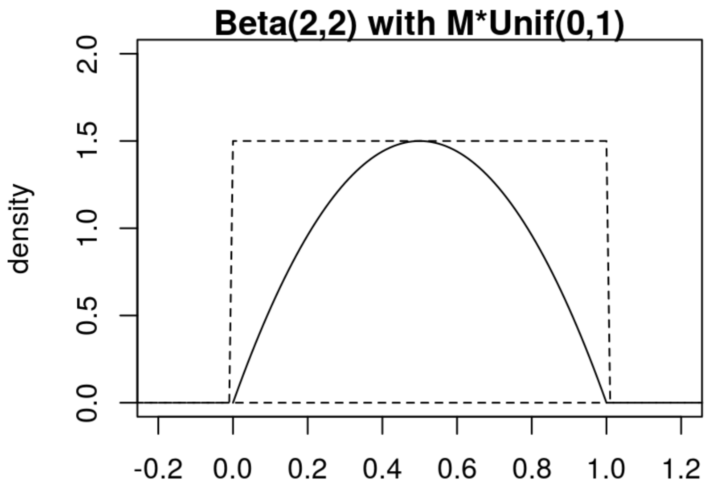
## Example: Beta(2,2)

Suppose we'd like to generate samples from $Beta(2,2)$ distribution. 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

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

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.





23/40

## Example: Beta(2,2), pt. 2

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) (http://www.wolframalpha.com/input/?i=max+6x(1+-+x))

```
## 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

n = 10000
```

24/40

## Example: Beta(2,2), pt. 2

First, we'll generate 5 samples to test the algorithm

```
for(i in 1:5){
  print(paste("Run: ", i))
  u = round(runif(1), 5)
  y = round(runif(1), 5)
  accept <- u <= f(y)/(M* g(y))
  print(paste("U: ", u, "and Y:", y, "and f/M*g: ",f(y)/(M* g(y))))
  print(paste("Accept? ", accept))
  if(accept){
    X[i] <- y
  }
}
```

```
## [1] "Run:  1"
## [1] "U:  0.28758 and Y: 0.78831 and f/M*g:  0.6675093756"
## [1] "Accept?  TRUE"
## [1] "Run:  2"
## [1] "U:  0.40898 and Y: 0.88302 and f/M*g:  0.4131827184"
## [1] "Accept?  TRUE"
## [1] "Run:  3"
## [1] "U:  0.94047 and Y: 0.04556 and f/M*g:  0.1739371456"
## [1] "Accept?  FALSE"
## [1] "Run:  4"
## [1] "U:  0.52811 and Y: 0.89242 and f/M*g:  0.3840261744"
## [1] "Accept?  FALSE"
## [1] "Run:  5"
## [1] "U:  0.55144 and Y: 0.45661 and f/M*g:  0.9924692316"
## [1] "Accept?  TRUE"
```

## Example: Beta(2,2), pt. 3

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

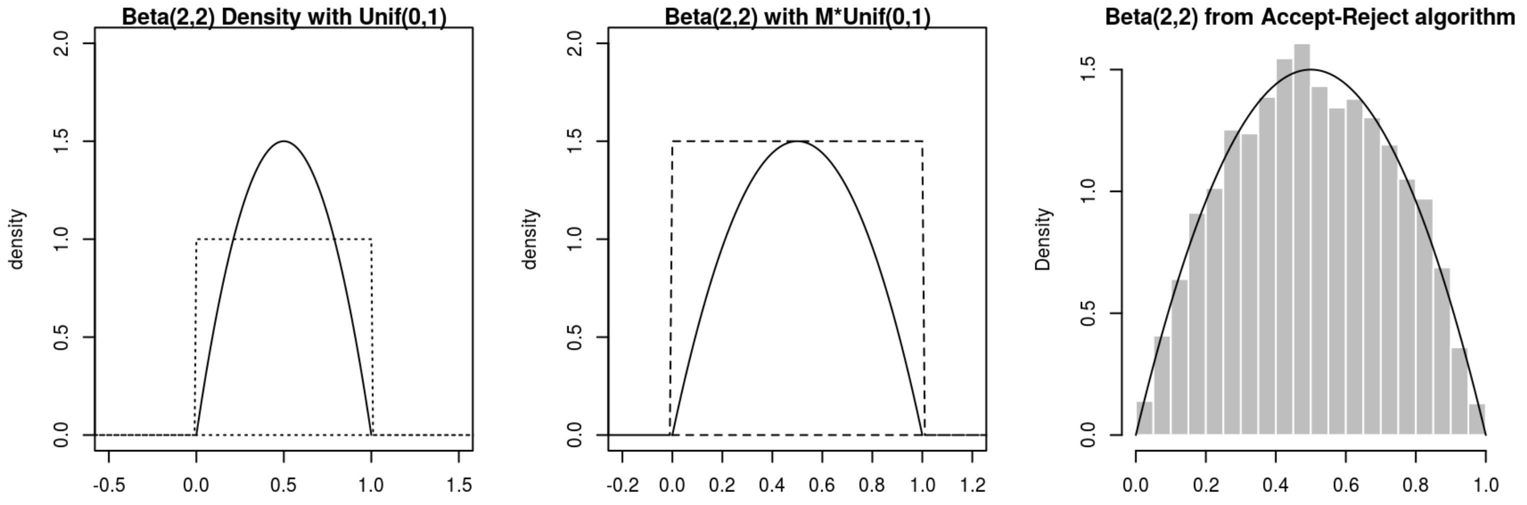
```
X = rep(NA, n); M = 1.5
i = 0 ## index set to start at 0
while(sum(is.na(X))){
  U = runif(1); Y = runif(1)
  accept <- U <= f(Y)/(M*g(Y))
  if(accept){
    i = i+1 ## update the index
    X[i] <- Y
  }
}

round(summary(X), 4)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0037  0.3250  0.4962  0.5000  0.6767  0.9930
```

```
round(qbeta(p = c(0, 0.25, 0.5, 0.75, 1), 2, 2), 4)
```

```
## [1] 0.0000 0.3264 0.5000 0.6736 1.0000
```

## Example: Beta(2,2), pt. 4



- The success of the Accept-Reject algorithm is to choose $M$ to be as small as possible while still maintaining an envelop over the target distribution

27/40

## Example: Beta(2,2), pt. 5

Code

```
## plot 1
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)

## plot 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)

## plot 3
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)
```

28/40

## Example: Beta(2,2), pt. 6

```
N = 1000; 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] 0.671
```

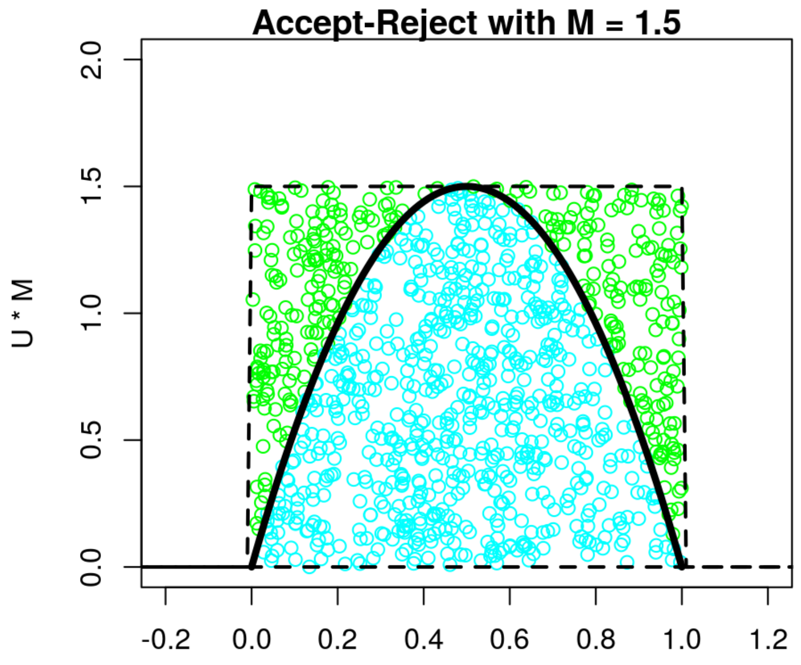
```
print(1/M) ## probability of acceptance
```

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

Exercise: Change the value of M to see how the performance of the algorithm changes

29/40

# Example: Beta(2,2), pt. 7



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.

## Example: Beta(2,2), pt. 7

### Code

```
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)
```

## Other Transformation Methods

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 for these "harder to simulate" distributions using transformation methods.

From a uniform distribution, $U \sim Unif(0,1)$ we can generate $X \sim Exp(1)$ to then generate the following:

1. $Y = \sum_{i=1}^{N} X$ where $Y \sim \chi^2_{2N}$ ($2N$ degress of freedom). See: Chi-Square distribution (http://www.math.uah.edu/stat/special/ChiSquare.html)
2. $Y = \beta \sum_{i=1}^{\alpha} X$ where $Y \sim Gamma(\alpha, \beta)$. See: Gamma distribution (http://www.math.uah.edu/stat/special/Gamma.html)
3. $Y = \frac{\sum_{i=1}^{a} X}{\sum_{i=1}^{a+b} X}$ where $Y \sim Beta(a,b)$. See: Beta distribution (http://www.math.uah.edu/stat/special/Beta.html)

Try these yourself!

32/40

## Example: Generate Rayleigh samples

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

For information on the Rayleigh distribution follow the link: Rayleigh Distribution (http://www.math.uah.edu/stat/special/Rayleigh.html)

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

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

$$\begin{aligned}
1 - exp\left(\frac{-x^2}{2\sigma^2}\right) &= U \\
exp\left(\frac{-x^2}{2\sigma^2}\right) &= 1 - U \\
log\left(exp\left(\frac{-x^2}{2\sigma^2}\right)\right) &= log(1 - U) \\
\frac{-x^2}{2\sigma^2} &= log(1 - U) \\
-x^2 &= 2\sigma^2 \times log(1 - U) \\
x &= \sqrt{-2\sigma^2 \times log(1 - U)}
\end{aligned}$$

- can't take the square root of a negative value

33/40

## Example: Generate Rayleigh samples

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 $\sigma$.

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 $Rayleigh(\sigma)$.

34/40

## Example: Generate Rayleigh samples

```
N = 10000
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)

summary(R_Out)
```

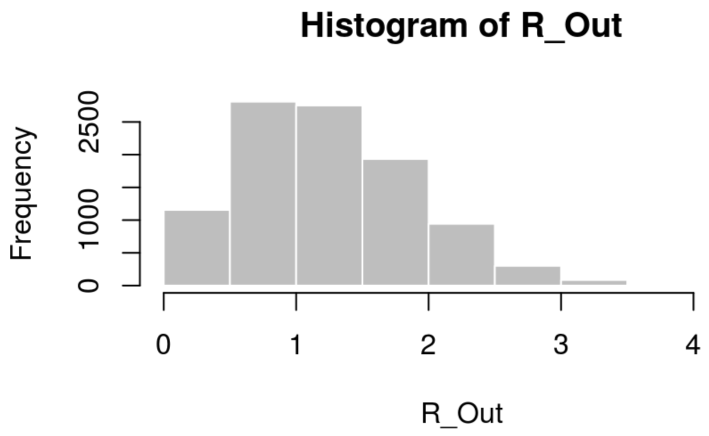
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.03023 0.75916 1.17814 1.25367 1.67331 4.41913
```

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

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

## Example: Generate Rayleigh samples, pt.2

```
hist(R_Out, col = "gray", border = "white")
```

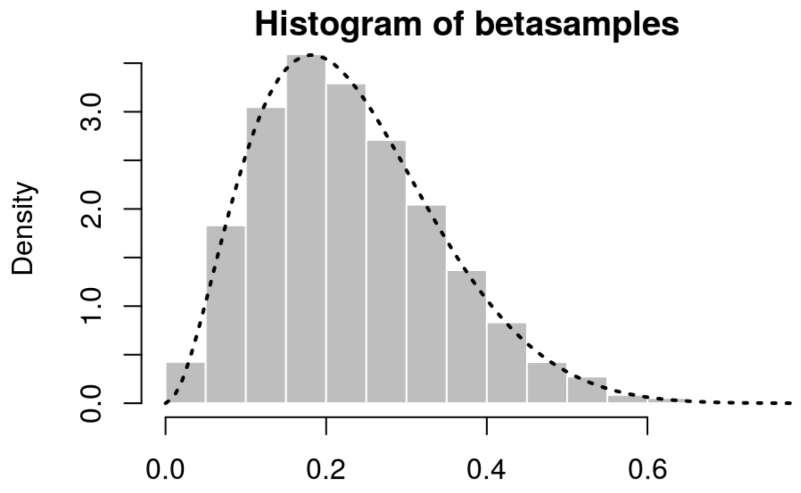


```
## compare with Rayleigh {VGAM}
```

## Appendix

## A Solution to Beta transformation

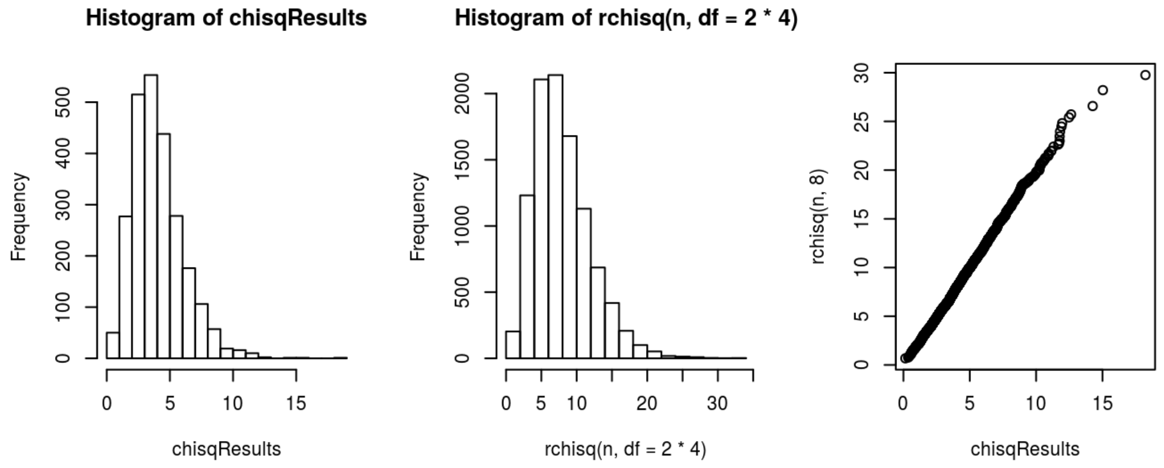
```
par(mar = c(3, 4, 1, 2))
a = 3
b = 10

A = matrix(rexp(10000*a, 1), ncol = a)
B = matrix(rexp(10000*b, 1), ncol = b)
numerator = rowSums(A)
denom = rowSums(A) + rowSums(B)
betasamples = numerator/denom

hist(betasamples, probability=TRUE, col = "gray", border = "white")
curve(dbeta(x, a, b), from=0, to=1, add=TRUE, lty=3, lwd = 2)
```



38/40

## A solution to Chisq transformation

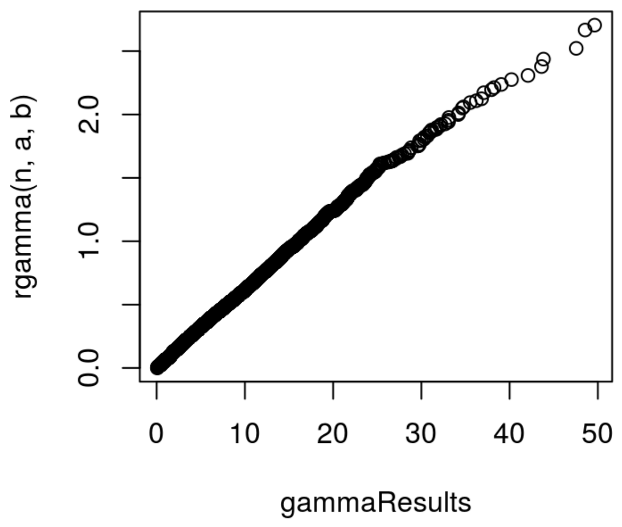
```
n = 10000
x = rexp(n=n, rate = 1)
mat = matrix(data = x, ncol = 4)
chisqResults = rowSums(mat)
par(mfrow = c(1, 3))
hist(chisqResults)
hist(rchisq(n, df = 2*4))
qqplot(x = chisqResults, y = rchisq(n, 8))
```



39/40

## A solution go Gamma transformation

```
b = 4
a = 2
n = 10000
x = rexp(n, 1)
mat = matrix(x, ncol = a)
gammaResults = b*rowSums(mat)
qqplot(x = gammaResults, y = rgamma(n,a,b))
```


rgamma(n, a, b)
gammaResults