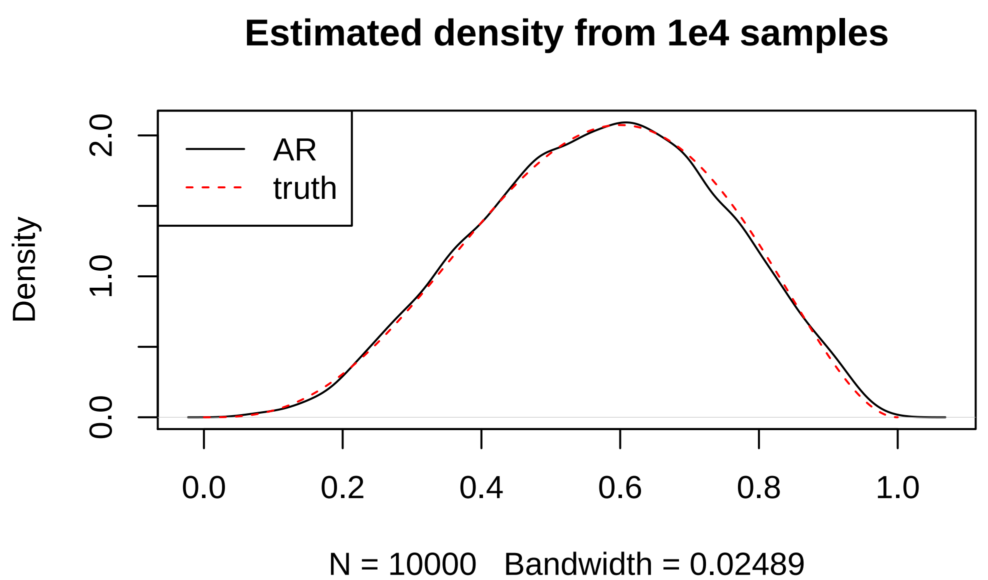
MTH210a: Lab 2 Solutions

1. **The file `BetaAR.R` contains partial code to implement an AR algorithm for a Beta$(4, 3)$ target. Complete the code and analyse the results.**

   Below is the complete code, along with the code for plots.

```r
########################
## Accept-reject for
## Beta(4,3) distribution
## Using U(0,1) proposal
########################
set.seed(1)
beta_ar <- function()
{
  c <- 60 *(3/5)^3 * (2/5)^2
  accept <- 0
  counter <- 0   # count the number of loop
  while(accept == 0)
  {
    counter <- counter + 1

    prop <- runif(1)
    ratio <- dbeta(prop, 4, 3)/c

    U <- runif(1)

    if(U <= ratio)
    {
      accept <- 1
      return(c(prop, counter))
    }
  }
}



### Obtaining 10^4 samples from Beta() distribution
```

```r
N <- 1e4
samp <- numeric(length = N)
counts <- numeric(length = N)
for(i in 1:N)
{
  rep <-   beta_ar() ## fill in
  samp[i] <-  rep[1] ## fill in
  counts[i] <-  rep[2]## fill in
}

# Make a plot of the estimated density from the samples
# versus the true density
x <- seq(0, 1, length = 500)
plot(density(samp), main = "Estimated density from 1e4 samples")
lines(x, dbeta(x, 4, 3), col = "red", lty = 2) ## Complete this
legend("topleft", lty = 1:2, col = c("black", "red"), legend = c("AR", "truth"))
```




```r
# This is c
(c <- 60 *(3/5)^3 * (2/5)^2)
```

```
[1] 2.0736
```

```r
# This is the mean number of loops required
mean(counts)
```

```
[1] 2.0936
```

```r
#They should be almost the same!
```

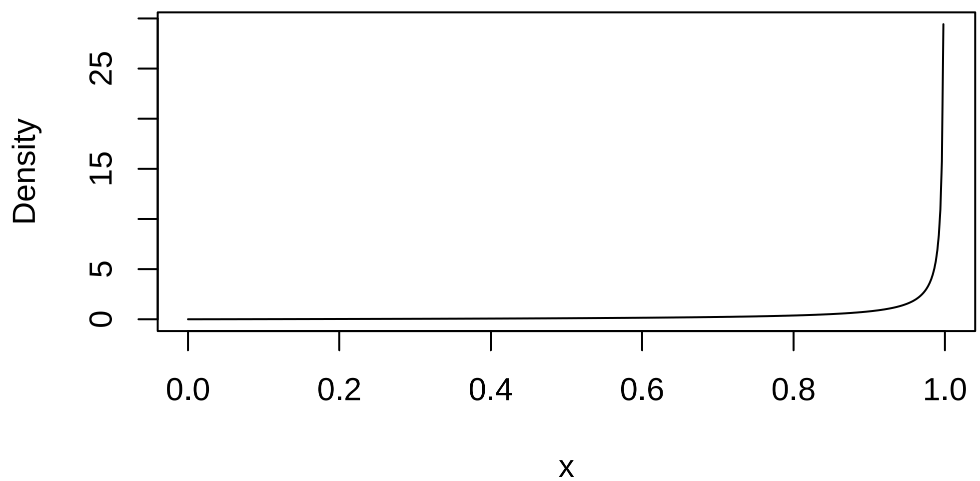
2. **Modify the `BetaAR.R` appropriately so that it can implement an AR algorithm for Beta$(2, .1)$.**

   Let us do some theory for this first. The target density is:

$$f(x) = \frac{\Gamma(2.1)}{\Gamma(2)\Gamma(1)} x^1 (1-x)^{.1-1} \qquad ; 0 \le x \le 1 .$$

   The shape of the target density is

```
x <- seq(0, 1, length = 5e2)
plot(x, dbeta(x, 2, .1), type = 'l', ylab = "Density")
```



   As you can see, as $x \to 1$, the target density diverges. Thus, using a uniform proposal will not work in this problem. Instead, we do the same trick as we did in class:

$$f(x) \le \frac{\Gamma(2.1)}{\Gamma(2)\Gamma(1)} (1-x)^{(.1-1)} \Rightarrow g(x) \propto (1-x)^{(.1-1)} .$$

   This means, $g(x)$ is the density of Beta$(1, .1)$. Since the task is to draw from Beta, I do not want to use `rbeta` command. Instead will us inverse transform.

$$g(x) = .1(1-x)^(.1-1) \Rightarrow G(x) = \int_0^x g(x)dx = 1 - (1-t)^{(.1)} .$$

   So the inverse is

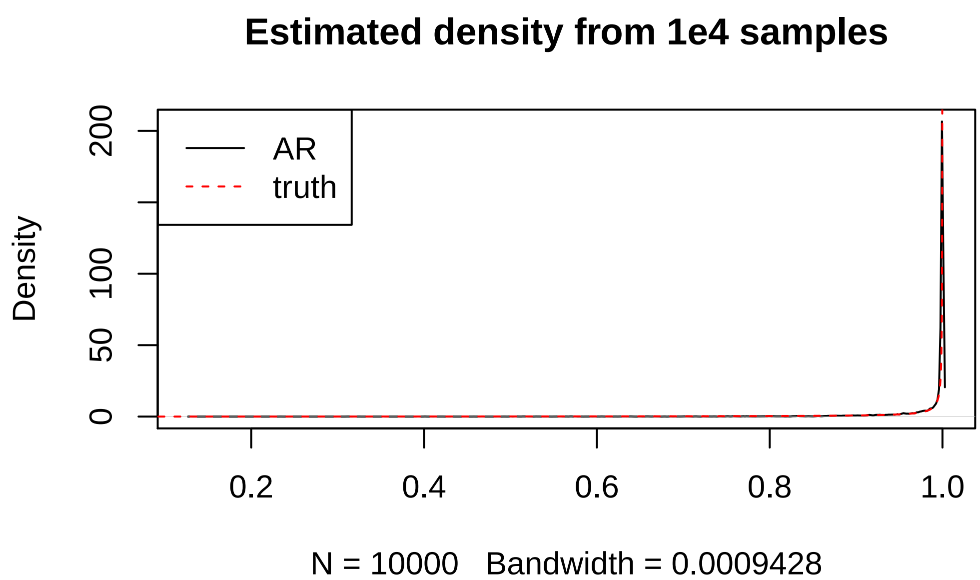
$$G^{-1}(u) = 1 + (1-u)^{1/.1} .$$

   Using the above, we can draw our proposal. However, we have to be careful about numerical inaccuracies, so we will do the following:

$$\log(1 - G^{-1}(u)) = 10 \log(1 - u)$$

```
########################
## Accept-reject for
## Beta(2,.1) distribution
## Using Beta(1,.1) proposal
########################
beta_ar <- function(m = 2, n = .1)
{
  c <- gamma(m + n)/(gamma(n)*gamma(m))/n
  accept <- 0
  counter <- 0   # count the number of loop
  while(accept == 0)
  {
    counter <- counter + 1

    # Inverse transform to draw from proposal
    U <- runif(1)
    foo <- (1/n) * log(1 - U)
    prop <- 1 - exp(foo)

    # log ratio
    log.ratio <- dbeta(prop, m, n, log = TRUE) - log(c)
    - dbeta(prop, 1, n, log = TRUE)
    if(log(runif(1)) <= log.ratio)
    {
      accept <- 1
      return(c(prop, counter))
    }
  }
}


### Obtaining 10^4 samples from Beta() distribution
N <- 1e4
samp <- numeric(length = N)
counts <- numeric(length = N)
for(i in 1:N)
{
  rep <-   beta_ar() ## fill in
  samp[i] <-  rep[1] ## fill in
```

```r
  counts[i] <-  rep[2]## fill in
}

# Make a plot of the estimated density from the samples
# versus the true density
x <- seq(0, 1, length = 5000)
plot(density(samp), main = "Estimated density from 1e4 samples")
lines(x, dbeta(x, 2, .1), col = "red", lty = 2) ## Complete this
legend("topleft", lty = 1:2, col = c("black", "red"), legend = c("AR", "truth"))
```



Since the estimated density from my 10^4 samples matches the true density, I can be confident that I have coded this correctly. In the above, to avoid numerical instability, instead of calculating

$$\frac{f(y)}{cg(y)}$$

I calculate:

$$\log(f(y)) - \log(c) - \log(g(y))$$

And compare log of the ratio to log of a uniform random variable.

3. **The file `circleAR.R` contains partial code to implement the accept-reject sampler to draw from the uniform distribution over the circle. Complete the code.**

```r
###################################
## Accept-reject for obtaining
## sample uniformlyfrom a standard circle
```

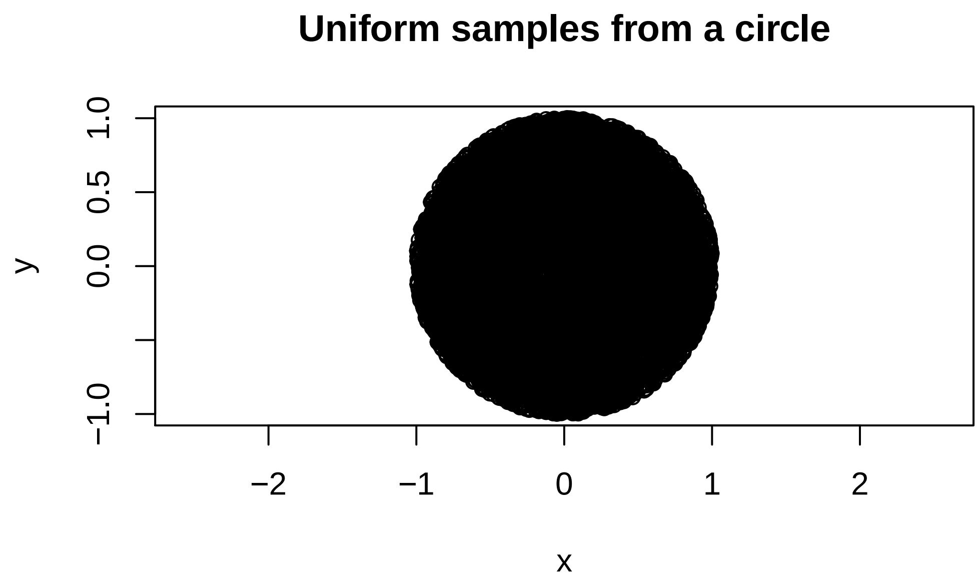
```r
## using a box as a proposal
##############################
set.seed(1)
circle_ar <- function()
{
  accept <- 0
  counter <- 0   # count the number of loop
  while(accept == 0)
  {
    counter <- counter + 1

    prop.temp <- runif(2) # from U(0,1)
    prop <- -1 + 2*prop.temp # from U(-1,1)

    if(prop[1]^2 + prop[2]^2 <= 1) # fill condition
    {
      accept <- 1
      return(c(prop, counter))
    }
  }
}

# Simulation 10^4 samples from circle
N <- 1e4
samp <- matrix(0, ncol = 2, nrow = N)
counts <- numeric(length = N)
for(i in 1:N)
{
  foo <- circle_ar()  # I use foo as a dummy name
  samp[i,] <- foo[1:2]
  counts[i] <- foo[3]
}


4/pi
```

```
[1] 1.27324
```

```r
# [1] 1.27324
mean(counts)  # should be very close
```

```
[1] 1.2783
```

```r
# Plotting the obtained samples
# no paritcular part of the circle is favored more
# than any other part.
plot(samp[,1], samp[,2], xlab = "x", ylab = "y",
  main = "Uniform samples from a circle", asp = 1)
```



4. **Taking inspiration from `circleAR.R`, implement Problem 16 from Section 4 Exercises of the notes.**

   a. **Implement an accept-reject sampler to sample uniformly from the circle $\{x^2 + y^2 \leq 1\}$** and obtain 10000 samples and estimate the probability of acceptance. Does it approximately equal $\pi/4$**?**

      We have already done this.

   b. **Now consider sampling uniformly from a p-dimensional sphere (a circle is $p = 2$). Consider a p-vector $\mathbf{x} = (x_1, x_2, \dots, x_p)$ and let $\|\cdot\|$ denote the Euclidean norm. The pdf of this distribution is**

$$
f(\mathbf{x}) = \frac{\Gamma\left(\frac{p}{2}+1\right)}{\pi^{p/2}} I\{\|\mathbf{x}\| \leq 1\}\,.
$$

      Use a uniform \$p\$-dimensional hypercube to sample uniformly from this sphere. Implement this for p = 3, 4, 5, and 6. What happens as p increases?

      To code this question, we will first have to do some theory to ensure that c remains finite and to understand what the value of c will be. We consider a p dimensional box as the proposal, centered at the origin. The pdf of the uniform distribution over this box is

$$g(\mathbf{x}) = \frac{1}{2^p} I(-1 \leq x_i \leq 1, i = 1, \dots, p)\,.$$

For this, we can find c since

$$\sup_{\mathbf{x}} \frac{f(\mathbf{x})}{g(\mathbf{x})} = \frac{\Gamma\left(\frac{p}{2}+1\right) 2^p}{\pi^{p/2}} I\{\|\mathbf{x}\| \leq 1\} \leq \frac{\Gamma\left(\frac{p}{2}+1\right) 2^p}{\pi^{p/2}}\,.$$

The above value of c increases rapidly as a function of p

```
c_sphere <- function(p)
{
  gamma(p/2 + 1)* 2^p/ (pi^(p/2))
}
c_sphere(c(2:6, 10, 30))
```

```
[1] 1.273240e+00 1.909859e+00 3.242278e+00 6.079271e+00 1.238459e+01
[6] 4.015428e+02 4.899496e+13
```

The value of c increases rapidly with p. So we can see that the algorithm will slow down incredibly in higher dimensions.

```
###################################
## Accept-reject for obtaining
## sample uniformly from a sphere
## using a box as a proposal
###############################
sphere_ar <- function(p = 3)
{
  accept <- 0
  counter <- 0   # count the number of loop
  while(accept == 0)
  {
    counter <- counter + 1

    prop.temp <- runif(p) # from U(0,1)
    prop <- -1 + 2*prop.temp # from U(-1,1)

    if(sum(prop^2) <= 1) # fill condition
    {
      accept <- 1
      return(c(prop, counter))
    }
  }
}
```

```r
# Simulation 10^3 samples from circle
N <- 1e3
p <- 4
samp <- matrix(0, ncol = p+1, nrow = N)
counts <- numeric(length = N)
for(i in 1:N)
{
  foo <- sphere_ar(p = p)  # I use foo as a dummy name
  samp[i, 1:p] <- foo[1:p]
  counts[i] <- foo[p+1]
}


c_sphere(p = 4)
```

```
[1] 3.242278
```

```r
mean(counts)  # should be very close
```

```
[1] 3.136
```

5. **Write `R` code for Problem 7 in Exercises from Section 4 of the notes.**

   Let X be an Exp(1). Provide an efficient algorithm for simulating a random variable whose distribution is the conditional distribution of X given that $X < 0.05$. That is, its density function is

$$f(x) = \frac{e^{-x}}{1 - e^{-0.05}} \qquad 0 < x < 0.05\,.$$

   Using R generate 1000 such random variables and use them to estimate $E[X \mid X < 0.05]$.

   First, we will do the theory for this. Note that that the target density if the truncated expoential(1) truncated to be between 0 and 0.05. Just like the previous truncation examples, an AR is easy, if we use an Exponential proposal. I will not show the math for this; please do this by your self. We will get finally for $Y \sim Exp(1)$

$$\frac{f(y)}{cg(y)} = I(0 \leq y \leq .05$$

   After we get samples from the truncated exponential, we need to return the mean of this truncated exponential. We can estimate the population mean with the sample mean: so finally when we get $X_1, X_2, \dots, X_n$ from Truncated Exponential

$$\frac{1}{n}\sum_{t=1}^{n} X_t$$

```r
#### sample from truncated exp
truncExp <- function()
{
  accept <- 0
  count <- 0
  # inverse transform
  # to sample from exp
  while(!accept)
  {
    count <- count + 1
    U <- runif(1)
    expo <- -log(U)
    if(expo <= .05)
    {
      accept <- 1
      return(c(expo, count))
    }
  }
}

## Obtaining multiple samples
N <- 1e4
samples <- numeric(length = N)
try <- numeric(length = N)
for(i in 1:N)
{
  rep <- truncExp()
  samples[i] <- rep[1]
  try[i] <- rep[2]
}
mean(samples)  # answer
```

```
[1] 0.02476285
```

6. **Using only $U(0,1)$ draws, draw samples from Gamma$(4,3)$ using Accept-Reject and an exponential proposal. Compare the performance of the sampler using the optimal exponential proposal, versus $\lambda = 2$.**

   We have done this problem theoretically in the class. First note that $\alpha > 1$ and so the optimal value of λ in this proposal is

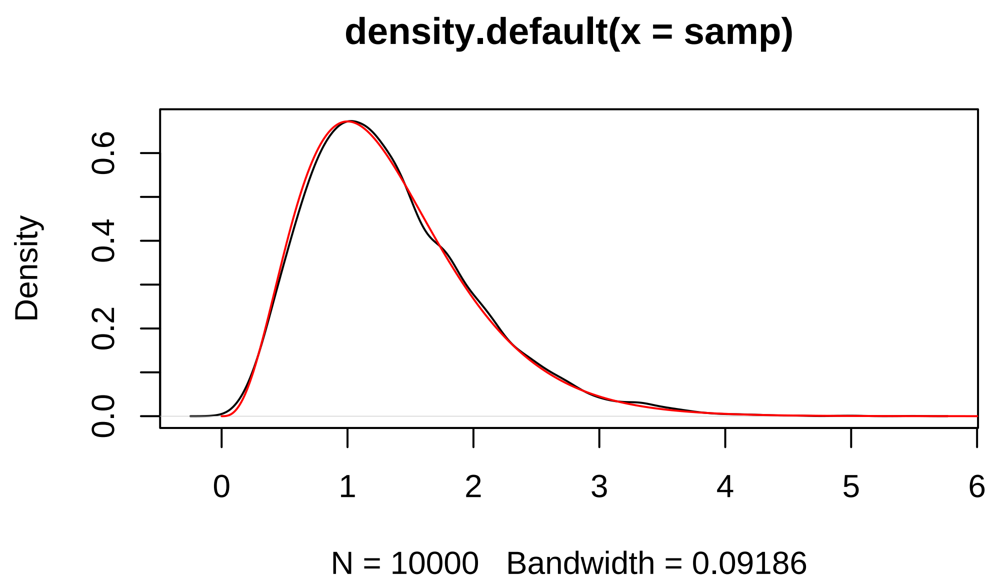
$$\lambda = \frac{\beta}{\alpha} = \frac{4}{3}$$

```
########################
## Accept-reject for
## Gamma(4,3) distribution
## Using Exp(lambda) proposal
########################
gamma_ar <- function(alpha, beta, lambda)
{
  if(alpha < 1)
  {
    stop("alpha less than 1. AR not possible with Exponential proposal")
  }
  accept <- 0
  counter <- 0   # count the number of loop

  # value of x where f/g is max
  max.x <- (alpha - 1)/(beta - lambda)

  # calculating c at that value +some little thing
  c.lambda <- dgamma(max.x, alpha, beta)/ dexp(max.x, rate = lambda) + .00001
  while(accept == 0)
  {
    counter <- counter + 1

    # from Exp(lambda)
    # using inverse-transform
    foo <- runif(1)
    prop <- -log(foo)/lambda

    log.ratio <- dgamma(prop, alpha, beta, log = TRUE) - dexp(prop, rate = lambda, log = TRUE) -

    if(log(runif(1)) <= log.ratio)
    {
      accept <- 1
      return(c(prop, counter))
    }
  }
}


### Obtaining 10^4 samples from Beta() distribution
N <- 1e4
samp <- numeric(length = N)
```

```r
counts <- numeric(length = N)
for(i in 1:N)
{
  rep <-   gamma_ar(alpha = 4, beta = 3, lambda = 4/3) # at optimal value of lambda
  samp[i] <-  rep[1]
  counts[i] <-  rep[2]
}
mean(counts)
```

```
[1] 2.9592
```

```r
plot(density(samp), ylab = "Density")
x <- seq(0, 6, length = 1e3)
lines(x, dgamma(x, 4, 3), col = "red")
```



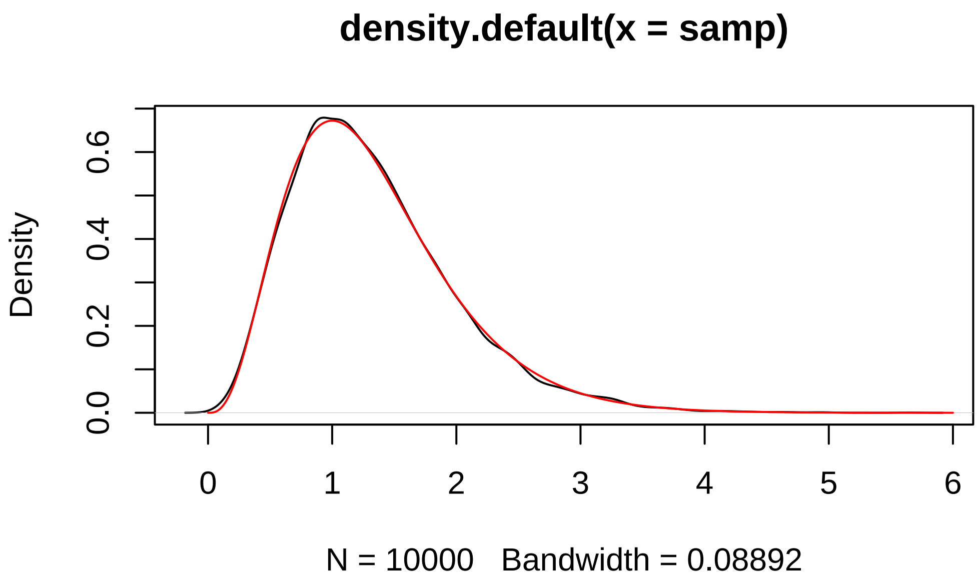
The mean counts is roughly 2.9 which matches the theoretical optimal, and the density function of the sampled values matches the true density. Now we repeat for $\lambda = 2$ which should be less efficient (c will be larger). But notice how the density matches similarly.

```r
# For lambda = 2 now
N <- 1e4
samp <- numeric(length = N)
counts <- numeric(length = N)
for(i in 1:N)
{
  rep <-   gamma_ar(alpha = 4, beta = 3, lambda = 2) # at value 2
  samp[i] <-  rep[1]
```

```
  counts[i] <-  rep[2]
}
mean(counts)
```

```
[1] 8.9764
```

```
plot(density(samp), ylab = "Density")
x <- seq(0, 6, length = 1e3)
lines(x, dgamma(x, 4, 3), col = "red")
```



7. **Suppose** $Y = \sum_{i=1}^{5} X_i$ **where** $X_i \sim \text{Weibull}(\alpha_i, \lambda)$**. Here density of Weibull**(α, λ) **is**

$$f(x) = \alpha\lambda^{-\alpha}x^{\alpha-1}e^{-\lambda x^{\alpha}}, \qquad x > 0\,.$$

Using only $U(0,1)$ **draws, estimate** $\text{E}(Y^2)$. Assume $\alpha_i = i$ and $\lambda = 5$.

Unfortunately there were a few typos in the question. The above is the corrected density. First, I need to figure out how to sample from Weibull distribution. Since inverse transform is fairly easy method, I want to try that first. Using change of variables trick, it can be shown that

$$F(x) = 1 - e^{-(\lambda x)^{\alpha}}\,.$$

Inverting this function, I obtain that

$$F^{-1}(u) = \frac{[-\log(1-u)]^{1/\alpha}}{\lambda}\,.$$

This means, we can sample from Weibull easily. Soo in order to estimate $\text{E}(Y^2)$, I note that if I can obtain $Y_1, Y_2, \dots, Y_n \overset{iid}{\sim}$ Distribution of Y, then I can estimate this expectation with:

$$\frac{1}{n}\sum_{t=1}^{n} Y_t^2$$

Simulating from the distribution of Y is possible by sampling Weibulls and adding them up as the formula indicates. Below, the function `distY` obtains one draw from Y given a vector of α and λ.

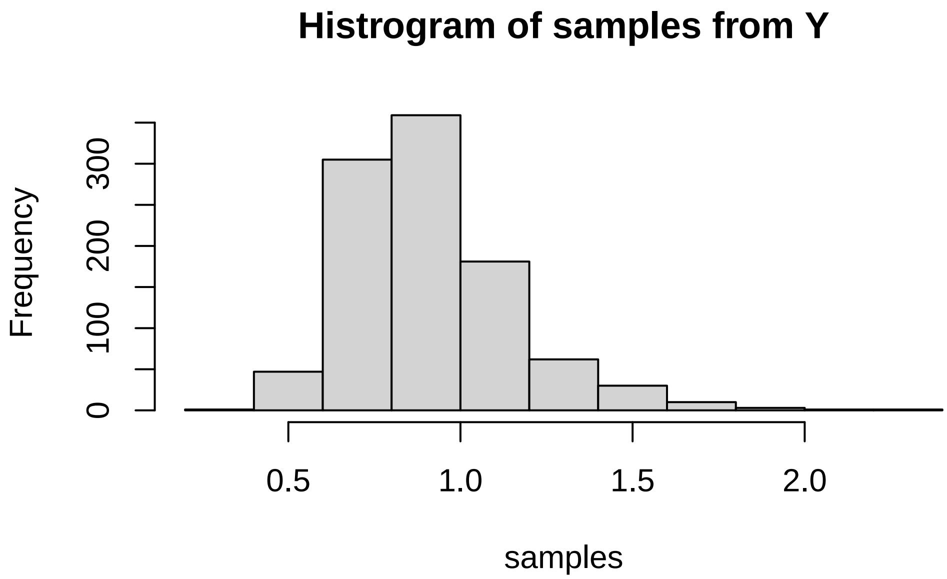
```
####################
# Sample from dist of Y
####################
distY <- function(alpha, lambda)
{
  l <- length(alpha)
  Wi <- numeric(length = l)
  for(i in 1:l)
  {
    U <- runif(1)
    Wi[i] <- (-log(1-U))^(1/alpha[i])/lambda
  }
  return(sum(Wi))
}
```

Now, I will call this function $n = 1e3$ times to estimate $E(Y^2)$ from a sample average of these 1000 Yi's.

```
### Estimate expectation with average
samples <- replicate(1e3, distY(alpha = 1:5, lambda = 5))

## Final answer
mean(samples^2)
```

```
[1] 0.8901571
```

```
## Just for information, here is a
## hist of samples of Y
hist(samples, main = "Histrogram of samples from Y")
```


Histrogram of samples from Y
Frequency
0
100
200
300
0.5
1.0
1.5
2.0
samples