MTH210: Lab 4 Solutions

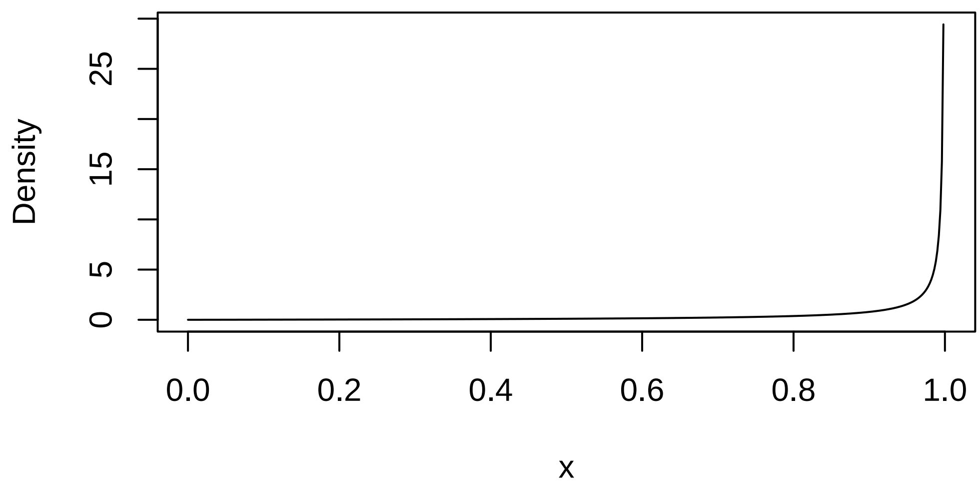
1. **Implement an AR algorithm to sample from a Beta(2, .1) distribution. Follow the theory from the notes.**

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

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

   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:

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

   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

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

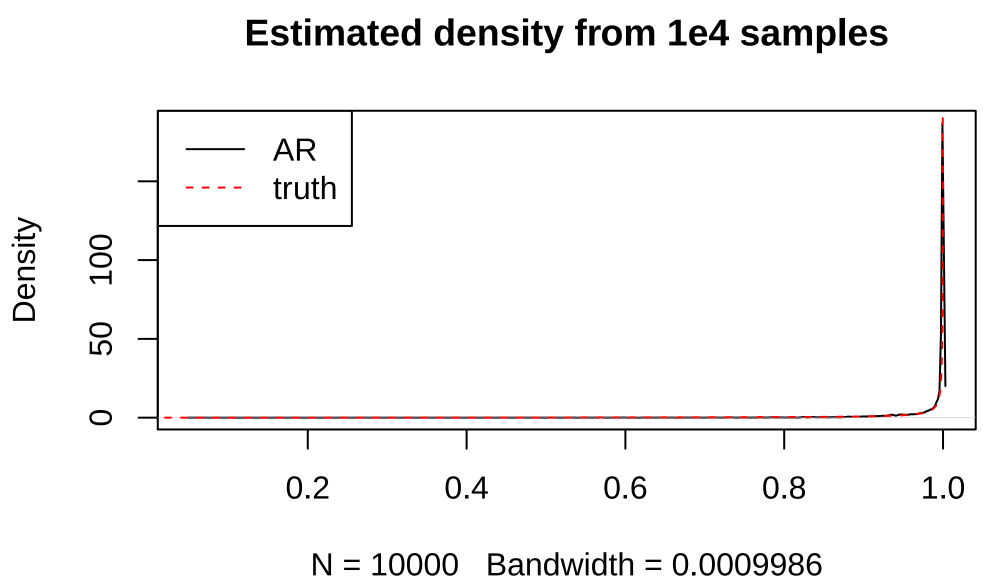
Using the above, we can draw our proposal. However, we have to be careful about numerical inaccuracies, so we will do the following:

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

and then exponentiate.

```
#########################
## 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
  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.

2. **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. The way we found the optimal value for an Exponential Proposal was by finding the bound c as a function of $c(\lambda)$, and then determining which λ minimizes $c(\lambda)$. First note that $\alpha > 1$ and so the optimal value of λ in this proposal is

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

   Further, from our notes, recall that for any $\lambda < \beta$, the sup $f(x)/g(x)$ occurs at

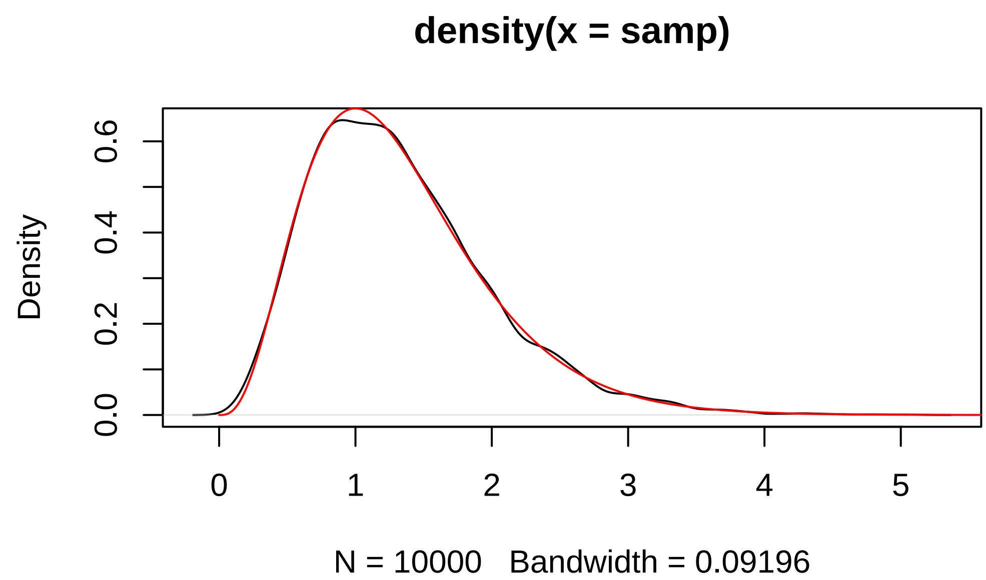
$$x = \frac{\alpha - 1}{\beta - \lambda}\,.$$

```
#########################
## 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)
```

```r
    # 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)
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] 2.8656
```

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

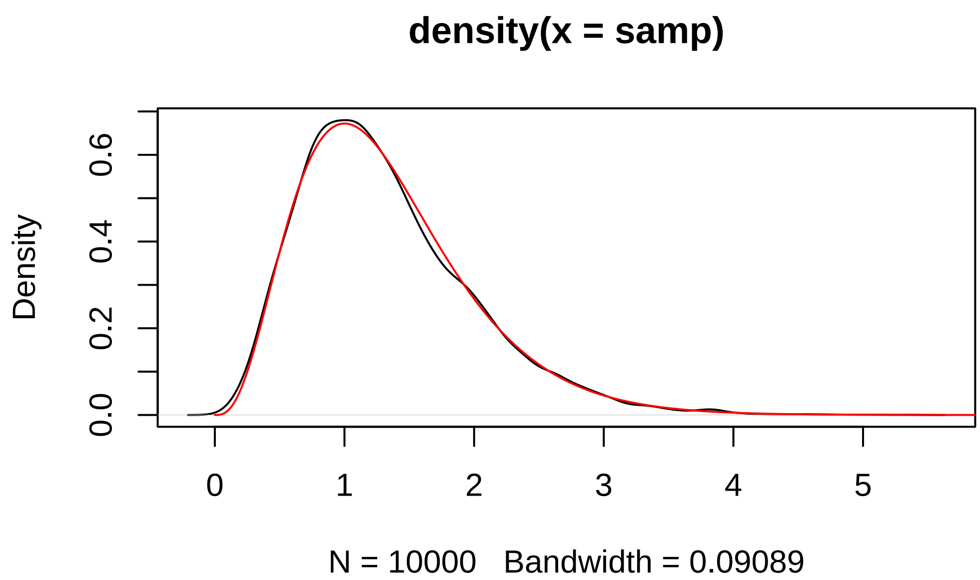


The mean counts is roughly 2.9 which matches the theoretical optimal, and the density function of the sampled values matches the true density. Now we repeat for $\lambda = 2$ which should be less efficient (c will be larger). But notice how the density matches similarly.

```
# 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] 9.1226
```

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



3. **For a $N(0,1)$ target, consider a Cauchy proposal with scale parameter σ, where the pdf of such a proposal is**

$$g(x) = \frac{1}{\pi\sigma}\frac{1}{(1+(x/\sigma)^2)}\,.$$

Find the optimal value of σ, and implement the AR algorithm for this value.

Before we do any implementation, we will have to do some theory. First, note that the target density is

$$f(x) = \frac{1}{\sqrt{2\pi}}e^{-x^2/2}\,.$$

For this target and this proposal, we get

$$\begin{aligned}
r(x) &:= \frac{f(x)}{g(x)} = \frac{\pi\sigma}{\sqrt{2\pi}}\left(1+\frac{x^2}{\sigma^2}\right)e^{-x^2/2} \\
\log r(x) &= \log\frac{\pi\sigma}{\sqrt{2\pi}} + \log\left(1+\frac{x^2}{\sigma^2}\right) - \frac{x^2}{2} \\
\Rightarrow \frac{d\log r(x)}{dx} &= \frac{2x}{\sigma^2+x^2} - x \overset{set}{=} 0 \\
\Rightarrow x &= 0, x = \pm\sqrt{2-\sigma^2}(\text{ for } \sigma^2 < 2)
\end{aligned}$$

By checking the second derivatives, we can see that for $\sigma^2 < 2$, the maxima occurs at $x = \pm\sqrt{2-\sigma^2}$, and for $\sigma^2 > 2$, the maxima occurs at $x = 0$. Thus, we obtain that the bound is

$$c(\sigma) = \sup r(x) = \max\left\{r(0), r(\sqrt{2-\sigma^2})\right\} = \max\left\{\frac{\pi\sigma}{\sqrt{2\pi}}, \frac{\pi\sigma}{\sqrt{2\pi}}\left(1+\frac{2-\sigma^2}{\sigma^2}\right)e^{-(2-\sigma^2)/2}\right\}$$

In order to find the optimal value of σ^2, we will have to minimize $c(\sigma)$. We will find the minimum in two different cases:

Case 1: for $\sigma^2 \geq 2$, $c(\sigma)$ is an increasing function of σ, and thus the minimum occurs as $\sigma^2 = 2$, which yields the best c in this case to be $c(\sqrt{2}) = \sqrt{\pi}$.

Case 2: for $\sigma^2 < 2$, the minimum occurs at $\sigma = 1$ (you can show), at which point $c(1) = \sqrt{2\pi/e}$.

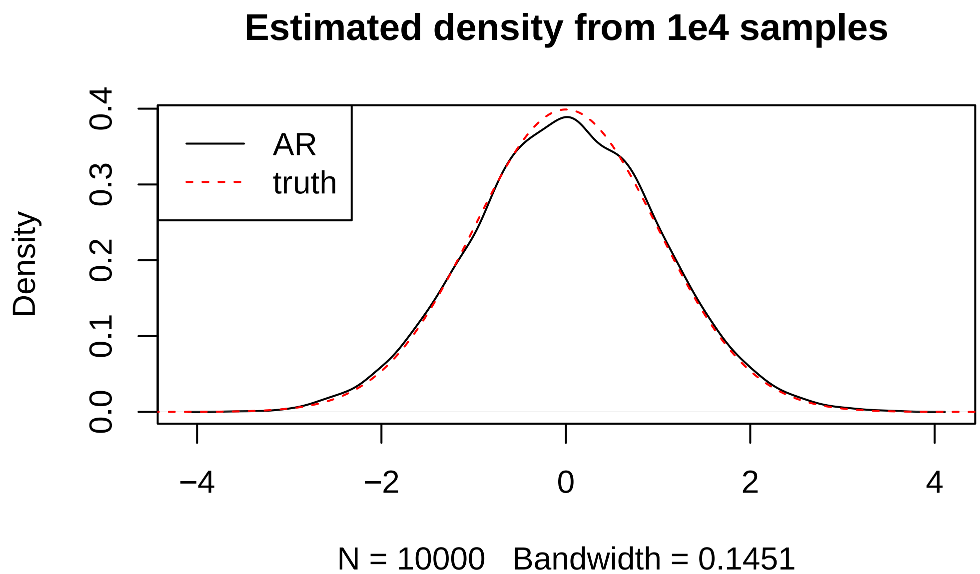
Since $\sqrt{\pi} > \sqrt{2\pi/e}$, we conclude that the optimal Cauchy proposal is with $\sigma = 1$ (the standard Cauchy)!

```
#########################
## Accept-reject for
## N((2,.1))0,1) distribution
## Using Cauchy(0, sigma) proposal
#########################
ARnormCauchy <- function(prop.scale = 1)
{
  if(prop.scale^2 >= 2) c <- pi*prop.scale/(sqrt(2*pi))
  if(prop.scale^2 < 2)
  {
    c <- dnorm(sqrt(2 - prop.scale^2))/dcauchy(sqrt(2 - prop.scale^2), scale = prop.scale)
  }

  accept <- 0
  counter <- 0   # count the number of loop
  while(accept == 0)
  {
    counter <- counter + 1

    # Inverse transform to draw from proposal
    prop.U <- runif(1)
    # generating from proposal -- using location scale trick
    prop <- prop.scale*tan(pi*(prop.U - .5))

    # log ratio
    log.ratio <- dnorm(prop, log = TRUE) - dcauchy(prop, scale = prop.scale, log = TRUE)
    log.ratio <- log.ratio - log(c)

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

```
  }
}



### Obtaining 10^4 samples from Normal() distribution
N <- 1e4
samp <- numeric(length = N)
counts <- numeric(length = N)
for(i in 1:N)
{
  rep <-   ARnormCauchy(prop.scale = 1)
  samp[i] <-  rep[1]
  counts[i] <-  rep[2]
}

# mean counts
mean(counts)
```

```
[1] 1.5201
```

```
# Make a plot of the estimated density from the samples
# versus the true density
x <- seq(-5, 5, length = 5000)
plot(density(samp), main = "Estimated density from 1e4 samples")
lines(x, dnorm(x), col = "red", lty = 2) ## Complete this
legend("topleft", lty = 1:2, col = c("black", "red"), legend = c("AR", "truth"))
```



```r
## Checking for other values to see if it is optimum
rowMeans(replicate(1e3, ARnormCauchy(prop.scale = 2)))[2]
```

```
[1] 2.511
```

```r
rowMeans(replicate(1e3, ARnormCauchy(prop.scale = .8)))[2]
```

```
[1] 1.589
```

```r
rowMeans(replicate(1e3, ARnormCauchy(prop.scale = 10)))[2]
```

```
[1] 12.09
```

```r
rowMeans(replicate(1e3, ARnormCauchy(prop.scale = .5)))[2]
```

```
[1] 2.035
```

Empirically as well, we see that $\sigma = 1$ gives the optimal proposal.