STA 478 Assignment #3 Solutions

Dr. Robert Buscaglia

September 15, 2025

Exercises

Exercise 1.

We will estimate properties of the Beta distribution using the sampling methods discussed in class. The Beta Distribution is supported on $x \in [0, 1]$ with density function

$$f(x) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)}$$

The distribution is characterized by two shape parameters $\alpha > 0$ and $\beta > 0$. The beta function, $B(\alpha, \beta)$, is a normalization parameter with $B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$. The the Gamma-function is used to denote $\Gamma(k) = (k-1)!$ for positive integers. Assuming $X \sim Beta(\alpha, \beta)$, it can be shown that:

$$E[X] = \frac{\alpha}{\alpha+\beta}$$
$$Var[X] = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$$

I encourage you to explore as far as you wish, but please complete a discussion of the sub-exercises below.

Using shape parameters $\alpha = 12$ and $\beta = 8$, simulate and discuss the following:

I wrote two more functions to make my life a little easier:

```r
summary.stats <- function(x)
{
   mean.temp <- mean(x)
   var.temp <- var(x)
   median.temp <- quantile(x, .50)
   temp.5 <- quantile(x, 0.05)
   temp.10 <- quantile(x, 0.10)
   temp.90 <- quantile(x, 0.90)
   temp.95 <- quantile(x, 0.95)
   return(data.frame(Mean = mean.temp, Var = var.temp,
                     Median = median.temp, Perc5 = temp.5,
                     Perc10 = temp.10, Perc90 = temp.90,
                     Perc95 = temp.95))
}

dbeta.overlay <- function(x, title)
{
   require(ggplot2)
   x <- data.frame(x=x)
```

```r
  xx <- seq(0, 1, length.out = 1e3)
  yy <- dbeta(xx, 12, 8) ## Plots a specific beta
  dbeta.df <- data.frame(xx, yy)
  ggplot(x, aes(x=x)) +
    geom_histogram(aes(y=after_stat(density)), color="white") +
    geom_line(data=dbeta.df, aes(x=xx, y=yy), col='red', lwd=2) +
    labs(x = 'Beta Draws', y='Density') +
    ggtitle(title)+
    theme(plot.title = element_text(hjust = 0.5))
}
```

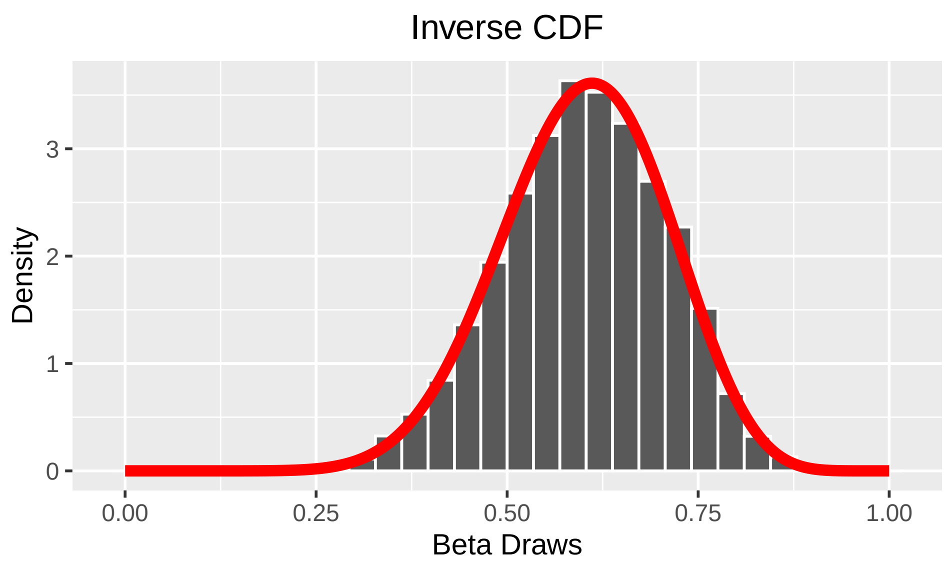
a

Generate 10000 samples using the inverse CDF method. The inverse-CDF function can be set to be `qbeta()`. Determine the mean, variance, median, and 5, 10, 90, and 95th percentiles. Display the estimate density against the target density.

```r
inv.cdf <- function(u) qbeta(u, 12, 8)
inv.out <- inverse.CDF(inv.cdf, 1e4)
inv.sum <- summary.stats(inv.out)
inv.sum %>% kable(align='c')
```

	Mean	Var	Median	Perc5	Perc10	Perc90	Perc95
50%	0.6002227	0.011724	0.6031586	0.4130887	0.4576338	0.7383512	0.7697048

```r
dbeta.overlay(inv.out, 'Inverse CDF')
```



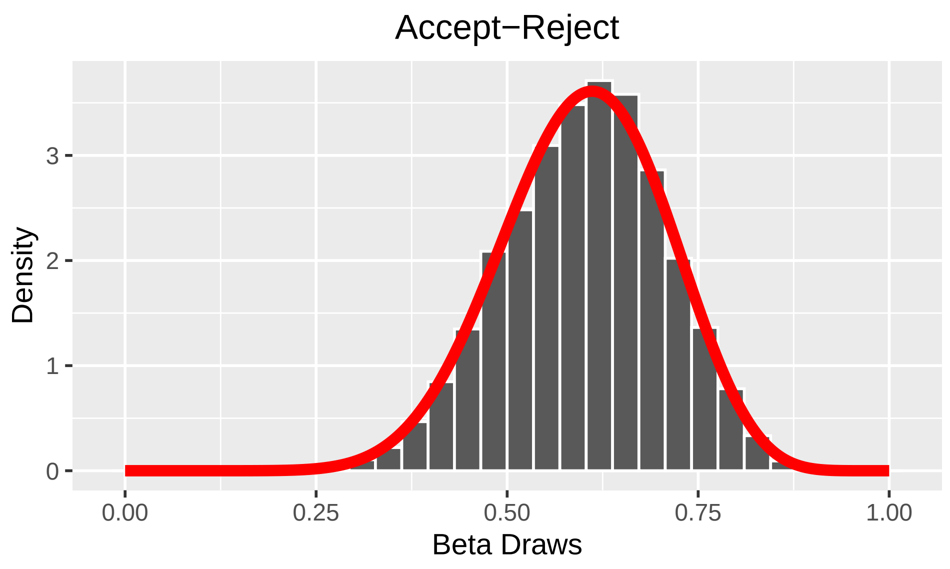
b

Generate 10000 samples using the Accept/Reject method. Determine the mean, variance, median, and 5, 10, 90, and 95th percentiles. Display the estimate density against the target density.

```r
target.density <- function(x) dbeta(x, 12, 8)
AR.out <- accept.reject(target.density, 0, 1, 1e4)
AR.sum <- summary.stats(AR.out)
AR.sum %>% kable(align='c')
```

	Mean	Var	Median	Perc5	Perc10	Perc90	Perc95
50%	0.6015368	0.0111735	0.6064404	0.4232924	0.4627369	0.7368818	0.7680616

```r
dbeta.overlay(AR.out, 'Accept-Reject')
```



c

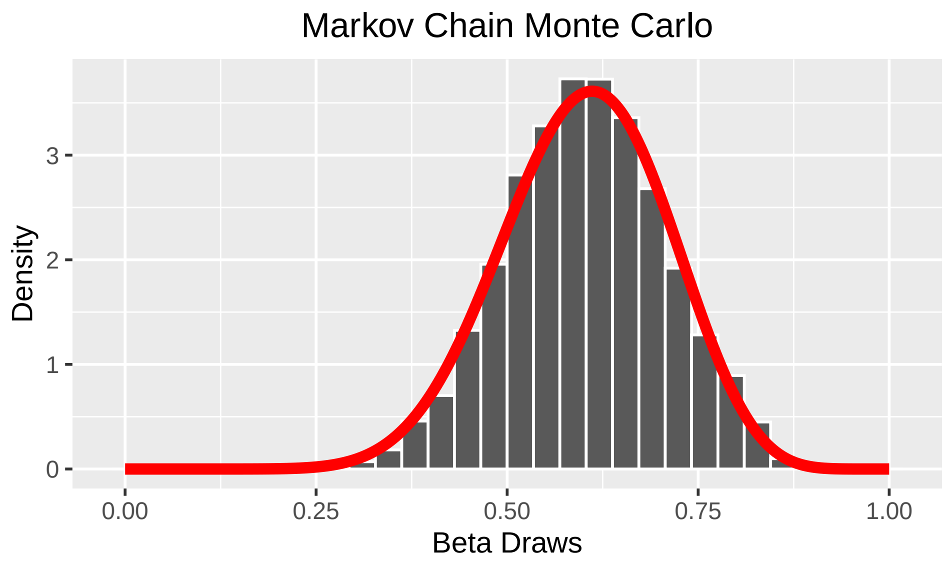
Generate 20000 steps using your MCMC function, using the final 10000 steps as your numerical draw. Discuss your choice of proposal functions. Determine the mean, variance, median, and 5, 10, 90, and 95th percentiles. Display the estimate density against the target density.

I chose to use the normal distribution as it is symmetrical and has a relatively useful 'shape' for this case. I adjusted the variance to a final value of 0.5 after some minimal tuning to ensure the sampling was somewhat effective. We would want to tune the proposal more carefully if our main tool here was to be MCMC. I was only looking that you properly chose a symmetrical distribution and implemented it correctly. In this case, without working on the proposal much more and using a basic MCMC setup, we see the MCMC is one of the worse performing methods. However, remember that MCMC works in all cases and has theory to support its convergence when drawing large samples.

```r
target.density <- function(x) dbeta(x, 12, 8)
proposal.func <- function(x) x + rnorm(1, 0, 0.5)
MCMC.out <- MCMC.metropolis(target.density, proposal.func, runif(1, 0, 1), 2e4)
MCMC.out <- MCMC.out[-(1:1e4),]
MCMC.sum <- summary.stats(MCMC.out)
MCMC.sum %>% kable(align='c')
```

	Mean	Var	Median	Perc5	Perc10	Perc90	Perc95
50%	0.6021907	0.0107826	0.602544	0.4305604	0.4670707	0.7377462	0.7767971

```r
dbeta.overlay(MCMC.out, 'Markov Chain Monte Carlo')
```



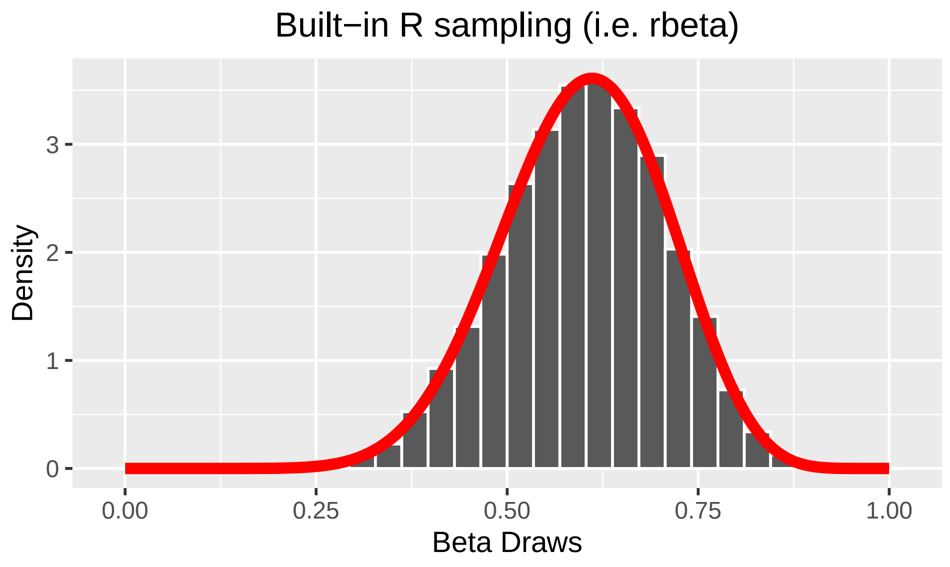
d

Draw 10000 samples using the R function `rbeta()`. Determine the mean, variance, median, and 5, 10, 90, and 95th percentiles. Display the estimate density against the target density.

```r
R.out <- rbeta(1e4, 12, 8)
R.sum <- summary.stats(R.out)
R.sum %>% kable(align='c')
```

	Mean	Var	Median	Perc5	Perc10	Perc90	Perc95
50%	0.599697	0.0114431	0.6034262	0.4172798	0.4593607	0.7364734	0.769473

```r
dbeta.overlay(R.out, 'Built-in R sampling (i.e. rbeta)')
```



e

Summarize the estimated statistics from each method against the known statistics for the $Beta(12, 8)$ distribution.

We need to both find the theoretical values for this distribution and create a table to display the results nicely.

```r
# Calculate theoretical values from given information or using
# quantile function for beta distribution.
beta.theoretical <- data.frame(
   Mean = (12/(12+8)), Var = (12*8)/((12+8+1)*(12+8)^2),
   Median = qbeta(0.5, 12, 8), Perc5 = qbeta(0.05, 12, 8),
   Perc10 = qbeta(0.10, 12, 8), Perc90 = qbeta(0.90, 12, 8),
   Perc95 = qbeta(0.95, 12, 8)
)
```

Next I create a table to display my results.

```r
Exer.1.final <- rbind(inv.sum, AR.sum, MCMC.sum, R.sum, beta.theoretical)
rownames(Exer.1.final) <- c('Inverse', 'A/R', 'MCMC', 'rbeta', 'Theoretical')
kable(Exer.1.final)
```

	Mean	Var	Median	Perc5	Perc10	Perc90	Perc95
Inverse	0.6002227	0.0117240	0.6031586	0.4130887	0.4576338	0.7383512	0.7697048
A/R	0.6015368	0.0111735	0.6064404	0.4232924	0.4627369	0.7368818	0.7680616
MCMC	0.6021907	0.0107826	0.6025440	0.4305604	0.4670707	0.7377462	0.7767971
rbeta	0.5996970	0.0114431	0.6034262	0.4172798	0.4593607	0.7364734	0.7694730
Theoretical	0.6000000	0.0114286	0.6033970	0.4180645	0.4586791	0.7367272	0.7702791

The learning objective here is to see that these different techniques all have a similar role in computational statistics. We want to be able to draw samples from a distribution to estimate properties. With 1e4 draws, we can see that all methods give us at least 2 digits of significance with the true answer. If we continued to draw larger samples, we could estimate more precise values. We could also use iterative resampling here, and do the experiment many many times, and make a distribution of our estimates. This would allow us to not only build an estimate (like the above table), but also investigate the variability of that property, allowing for the construction of a confidence interval. The big modern workhorse is the MCMC, but be aware these 'older' techniques still see plenty of use in situations where the distributions can be defined analytically.

Exercise 2.

The Rayleigh distribution is supported on the positive real line ($x \geq 0$) and is characterized by a single shape parameter ($\sigma > 0$). The PDF for the Rayleigh distribution is given by

$$f(x) = \frac{x}{\sigma^2} e^{\frac{-x^2}{2\sigma^2}} \quad x \geq 0, \quad \sigma > 0$$

Consider using MCMC to produce draws of this distribution.

a

Create a function that produces the target density of the Rayleigh distribution. The function should take as input x and the scale parameter σ and return the height of the density function. Be sure that the function understands how to handle negative values of x.

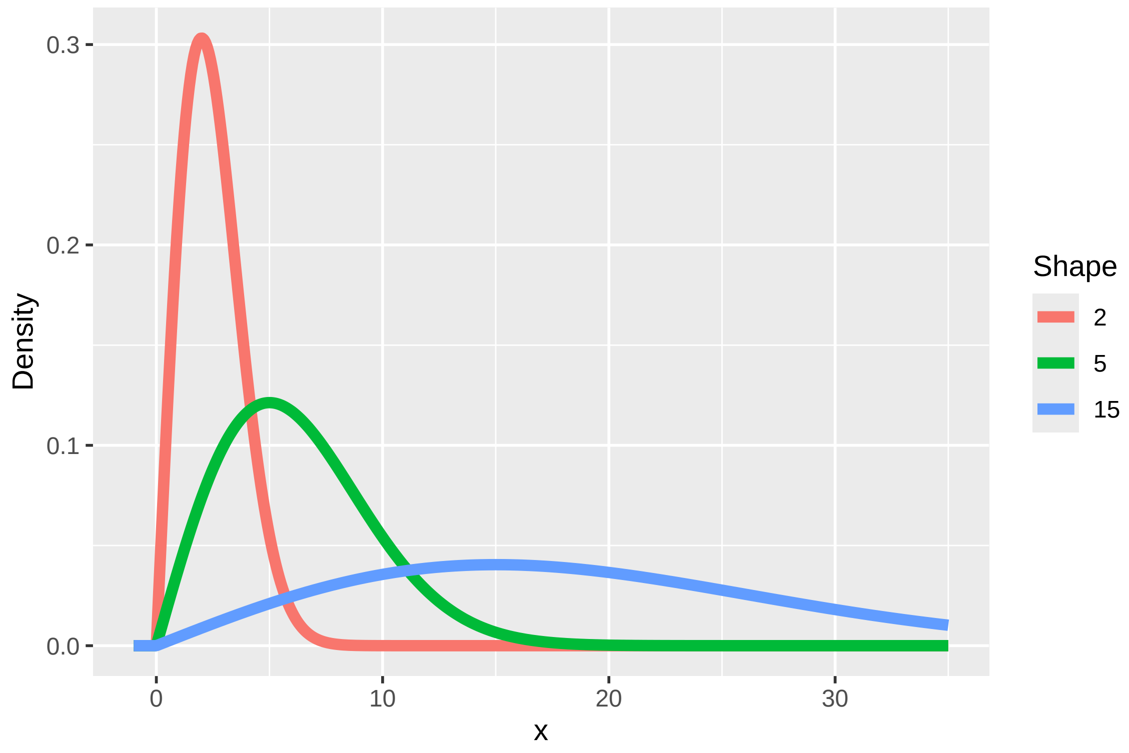
My function allows me to control sigma, which I will deal with when producing the MCMC analysis.

```r
rayleigh.pdf <- function(x, sigma){
  output <- ifelse(x >= 0, (x/sigma^2)*exp((-x^2)/(2*sigma^2)), 0)
  return(output)
}
```

b

Produce a graphical inspection of the Rayleigh density for scale parameters $\sigma = 2, 5, 15$. Show at least some part of the negative half of the real line, to ensure your function is handling negative values.

```r
rayleigh.graph <- data.frame(x = seq(-1, 35, length.out = 5e2)) %>%
  mutate(sigma2 = rayleigh.pdf(x, 2), sigma5 = rayleigh.pdf(x, 5),
         sigma15 = rayleigh.pdf(x, 15))
rayleigh.long <- rayleigh.graph %>% gather(Shape, Density, 2:4) %>%
  mutate(Shape = factor(Shape, levels=c('sigma2', 'sigma5', 'sigma15'),
                              labels=c('2', '5', '15')))
ggplot(rayleigh.long, aes(x=x, col=Shape, y=Density)) + geom_line(lwd=2)
```



c

Using your MCMC function produce samples of the Rayleigh distribution. Start by considering the case $\sigma = 2$, the proposal function $Uniform(-0.1, 0.1)$, and a starting point within the distribution. Generate 10 chains each with 2000 steps.

```r
target.pdf <- function(x) rayleigh.pdf(x, 2) ### Sigma = 2 target
proposal.func <- function(x) x + runif(1, -0.1, 0.1)
sigma2small <- matrix(ncol = 10, nrow=2e3)
```

```r
for(j in 1:10)
  sigma2small[,j] <- MCMC.metropolis(target.pdf, proposal.func, runif(1, 0, 10), 2e3)
```

All you were asked to do here is generate the chains.

d

Evaluate the Gelman diagnostics to assess chain convergence. Comment on the results.

```r
Gelman(sigma2small)
```

```
##          W        B sigma2.hat    R.hat    n.eff
## 1 1.307845 6444.795   4.529588 1.861021 14.05658
```

The chains in this case seem to have poor convergence. With a small variance within the proposal steps, it does not seem as though this MCMC properly explores the distribution. The Gelman R statistic is much too large, and we only obtain 20 *independent draws in* $2e3 \cdot 10 = 2e4$ *total steps. This is not a useful MCMC.*

I would like to start appending this all to keep for displaying later, so I create a new object here that will allow me to just append all the iterations I need to do below.

```r
Ex2.output <- data.frame('Sigma = 2', 'Unif(-0.1, 0.1)', as.vector(Gelman(sigma2small)))
colnames(Ex2.output)[c(1,2)] <- c('Shape', 'Proposal')
```

e

Repeat steps **c.** and **d.** using the proposal function $Uniform(-1, 1)$

We can be more sophisticated, I will just copy/paste and make updates.

```r
target.pdf <- function(x) rayleigh.pdf(x, 2) ### Sigma = 2 target
proposal.func <- function(x) x + runif(1, -1, 1)
sigma2med <- matrix(ncol = 10, nrow=2e3)
for(j in 1:10)
  sigma2med[,j] <- MCMC.metropolis(target.pdf, proposal.func, runif(1, 0, 10), 2e3)
Ex2.output <- rbind(Ex2.output,
                    data.frame(`Shape` = 'Sigma = 2', `Proposal` = 'Unif(-1, 1)',
                               as.vector(Gelman(sigma2med))))
kable(Ex2.output, align='c')
```

Shape	Proposal	W	B	sigma2.hat	R.hat	n.eff
Sigma = 2	Unif(-0.1, 0.1)	1.307845	6444.7948	4.529588	1.861021	14.05658
Sigma = 2	Unif(-1, 1)	1.903636	55.1771	1.930273	1.006972	699.66463

We immediately see a large improvement by allowing the variance of our proposal to increase. This MCMC seems to be much more well defined and has improved Gelman diagnostics.

f

Repeat steps **c.** and **d.** using the proposal function $Uniform(-25, 25)$

```r
target.pdf <- function(x) rayleigh.pdf(x, 2) ### Sigma = 2 target
proposal.func <- function(x) x + runif(1, -25, 25)
sigma2med <- matrix(ncol = 10, nrow=2e3)
for(j in 1:10)
  sigma2med[,j] <- MCMC.metropolis(target.pdf, proposal.func, runif(1, 0, 10), 2e3)
Ex2.output <- rbind(Ex2.output,
                    data.frame(`Shape` = 'Sigma = 2', `Proposal` = 'Unif(-25, 25)',
                               as.vector(Gelman(sigma2med))))
kable(Ex2.output, align='c')
```

Shape	Proposal	W	B	sigma2.hat	R.hat	n.eff
Sigma = 2	Unif(-0.1, 0.1)	1.307845	6444.79481	4.529588	1.861021	14.05658
Sigma = 2	Unif(-1, 1)	1.903636	55.17710	1.930273	1.006972	699.66463
Sigma = 2	Unif(-25, 25)	1.793602	13.38045	1.799396	1.001614	2689.58815

My intention here was to get you to produce something with 'too high' variance in the proposal. However, the uniform here actually is competitive with the smaller variance version. We still see decent convergence and nothing that is alarming in our diagnostics. Depending on the run, you may see either being the more effective sampler. We can improve this with 1) more chains and 2) more steps.

To show you the behavior I was after, I tack on one more fairly ridiculous proposal.

```r
target.pdf <- function(x) rayleigh.pdf(x, 2) ### Sigma = 2 target
proposal.func <- function(x) x + runif(1, -100, 100)
sigma2med <- matrix(ncol = 10, nrow=2e3)
for(j in 1:10)
  sigma2med[,j] <- MCMC.metropolis(target.pdf, proposal.func, runif(1, 0, 10), 2e3)
Ex2.output <- rbind(Ex2.output,
                    data.frame(`Shape` = 'Sigma = 2', `Proposal` = 'Unif(-100, 100)',
                               as.vector(Gelman(sigma2med))))
kable(Ex2.output, align='c')
```

Shape	Proposal	W	B	sigma2.hat	R.hat	n.eff
Sigma = 2	Unif(-0.1, 0.1)	1.307845	6444.79481	4.529588	1.861021	14.05658
Sigma = 2	Unif(-1, 1)	1.903636	55.17710	1.930273	1.006972	699.66463
Sigma = 2	Unif(-25, 25)	1.793602	13.38045	1.799396	1.001614	2689.58815
Sigma = 2	Unif(-100, 100)	1.472887	107.23721	1.525769	1.017794	284.55968

This last table makes it clear that Unif(-1, 1) and Unif(-25, 25) are fairly competitive, but that the larger proposal size is starting to cause problems. Let's go one even MORE ridiculous step to see this continue.

```r
target.pdf <- function(x) rayleigh.pdf(x, 2) ### Sigma = 2 target
proposal.func <- function(x) x + runif(1, -500, 500)
sigma2med <- matrix(ncol = 10, nrow=2e3)
for(j in 1:10)
  sigma2med[,j] <- MCMC.metropolis(target.pdf, proposal.func, runif(1, 0, 10), 2e3)
Ex2.output <- rbind(Ex2.output,
                    data.frame(`Shape` = 'Sigma = 2', `Proposal` = 'Unif(-500, 500)',
                                   as.vector(Gelman(sigma2med))))
kable(Ex2.output, align='c')
```

Shape	Proposal	W	B	sigma2.hat	R.hat	n.eff
Sigma = 2	Unif(-0.1, 0.1)	1.307845	6444.79481	4.529588	1.861021	14.05658
Sigma = 2	Unif(-1, 1)	1.903636	55.17710	1.930273	1.006972	699.66463
Sigma = 2	Unif(-25, 25)	1.793602	13.38045	1.799396	1.001614	2689.58815
Sigma = 2	Unif(-100, 100)	1.472887	107.23721	1.525769	1.017794	284.55968
Sigma = 2	Unif(-500, 500)	2.109017	794.65860	2.505292	1.089906	63.05329

This better illustrates there is a 'tuning' to this procedure, with the ***`Unif(-25,25)`*** *seemingly the most competitive proposal we have used.*

g

Repeat steps **c.** through **f.** using $\sigma = 15$.

We want to reproduce the analysis but for the updated shape parameter. I will continue to append everything to the output data.frame for discussion in part ***h***.

```
target.pdf <- function(x) rayleigh.pdf(x, 15) ### Sigma = 15
proposal.func <- function(x) x + runif(1, -0.1, 0.1)
sigma2med <- matrix(ncol = 10, nrow=2e3)
for(j in 1:10)
  sigma2med[,j] <- MCMC.metropolis(target.pdf, proposal.func, runif(1, 0, 30), 2e3)
Ex2.output <- rbind(Ex2.output,
                    data.frame(`Shape` = 'Sigma = 15', `Proposal` = 'Unif(-0.1, 0.1)',
                               as.vector(Gelman(sigma2med))))
kable(Ex2.output, align='c')
```

Shape	Proposal	W	B	sigma2.hat	R.hat	n.eff
Sigma = 2	Unif(-0.1, 0.1)	1.307845	6444.79481	4.529588	1.861021	14.05658
Sigma = 2	Unif(-1, 1)	1.903636	55.17710	1.930273	1.006972	699.66463
Sigma = 2	Unif(-25, 25)	1.793602	13.38045	1.799396	1.001614	2689.58815
Sigma = 2	Unif(-100, 100)	1.472887	107.23721	1.525769	1.017794	284.55968
Sigma = 2	Unif(-500, 500)	2.109017	794.65860	2.505292	1.089906	63.05329
Sigma = 15	Unif(-0.1, 0.1)	1.035082	236028.26100	119.048695	10.724447	10.08766

The lowest variance is equally as bad for our larger shape parameter.

```
target.pdf <- function(x) rayleigh.pdf(x, 15) ### Sigma = 15
proposal.func <- function(x) x + runif(1, -1, 1)
sigma2med <- matrix(ncol = 10, nrow=2e3)
for(j in 1:10)
  sigma2med[,j] <- MCMC.metropolis(target.pdf, proposal.func, runif(1, 0, 30), 2e3)
Ex2.output <- rbind(Ex2.output,
                    data.frame(`Shape` = 'Sigma = 15', `Proposal` = 'Unif(-1, 1)',
                               as.vector(Gelman(sigma2med))))
kable(Ex2.output, align='c')
```

Shape	Proposal	W	B	sigma2.hat	R.hat	n.eff
Sigma = 2	Unif(-0.1, 0.1)	1.307845	6444.79481	4.529588	1.861021	14.05658
Sigma = 2	Unif(-1, 1)	1.903636	55.17710	1.930273	1.006972	699.66463
Sigma = 2	Unif(-25, 25)	1.793602	13.38045	1.799396	1.001614	2689.58815
Sigma = 2	Unif(-100, 100)	1.472887	107.23721	1.525769	1.017794	284.55968
Sigma = 2	Unif(-500, 500)	2.109017	794.65860	2.505292	1.089906	63.05329
Sigma = 15	Unif(-0.1, 0.1)	1.035082	236028.26100	119.048695	10.724447	10.08766
Sigma = 15	Unif(-1, 1)	49.133963	157216.43677	127.717615	1.612258	16.24736

We now also observe the ***`Unif(-1, 1)`*** *is too small as well. Let us try these larger variances, including the more ridiculous cases I am trying here.*

```
target.pdf <- function(x) rayleigh.pdf(x, 15) ### Sigma = 15
proposal.func <- function(x) x + runif(1, -25, 25)
sigma2med <- matrix(ncol = 10, nrow=2e3)
for(j in 1:10)
  sigma2med[,j] <- MCMC.metropolis(target.pdf, proposal.func, runif(1, 0, 30), 2e3)
Ex2.output <- rbind(Ex2.output,
                    data.frame(`Shape` = 'Sigma = 15', `Proposal` = 'Unif(-25, 25)',
                               as.vector(Gelman(sigma2med))))
kable(Ex2.output, align='c')
```

Shape	Proposal	W	B	sigma2.hat	R.hat	n.eff
Sigma = 2	Unif(-0.1, 0.1)	1.307845	6444.79481	4.529588	1.861021	14.05658
Sigma = 2	Unif(-1, 1)	1.903636	55.17710	1.930273	1.006972	699.66463
Sigma = 2	Unif(-25, 25)	1.793602	13.38045	1.799396	1.001614	2689.58815
Sigma = 2	Unif(-100, 100)	1.472887	107.23721	1.525769	1.017794	284.55968
Sigma = 2	Unif(-500, 500)	2.109017	794.65860	2.505292	1.089906	63.05329
Sigma = 15	Unif(-0.1, 0.1)	1.035082	236028.26100	119.048695	10.724447	10.08766
Sigma = 15	Unif(-1, 1)	49.133963	157216.43677	127.717615	1.612258	16.24736
Sigma = 15	Unif(-25, 25)	98.933316	265.99201	99.016846	1.000422	7445.09937

We start to observe convergence when using ***`Unif(-25, 25)`****.*

```
target.pdf <- function(x) rayleigh.pdf(x, 15) ### Sigma = 15
proposal.func <- function(x) x + runif(1, -100, 100)
sigma2med <- matrix(ncol = 10, nrow=2e3)
for(j in 1:10)
  sigma2med[,j] <- MCMC.metropolis(target.pdf, proposal.func, runif(1, 0, 30), 2e3)
Ex2.output <- rbind(Ex2.output,
                    data.frame(`Shape` = 'Sigma = 15', `Proposal` = 'Unif(-100, 100)',
                               as.vector(Gelman(sigma2med))))
kable(Ex2.output, align='c')
```

Shape	Proposal	W	B	sigma2.hat	R.hat	n.eff
Sigma = 2	Unif(-0.1, 0.1)	1.307845	6444.79481	4.529588	1.861021	14.05658
Sigma = 2	Unif(-1, 1)	1.903636	55.17710	1.930273	1.006972	699.66463

Shape	Proposal	W	B	sigma2.hat	R.hat	n.eff
Sigma = 2	Unif(-25, 25)	1.793602	13.38045	1.799396	1.001614	2689.58815
Sigma = 2	Unif(-100, 100)	1.472887	107.23721	1.525769	1.017794	284.55968
Sigma = 2	Unif(-500, 500)	2.109017	794.65860	2.505292	1.089906	63.05329
Sigma = 15	Unif(-0.1, 0.1)	1.035082	236028.26100	119.048695	10.724447	10.08766
Sigma = 15	Unif(-1, 1)	49.133963	157216.43677	127.717615	1.612258	16.24736
Sigma = 15	Unif(-25, 25)	98.933316	265.99201	99.016846	1.000422	7445.09937
Sigma = 15	Unif(-100, 100)	99.660460	406.56551	99.813913	1.000770	4910.10231

The `Unif(-100, 100)` might be a little too big, as the diagnostics begin to turn against us.

```r
target.pdf <- function(x) rayleigh.pdf(x, 15) ### Sigma = 15
proposal.func <- function(x) x + runif(1, -500, 500)
sigma2med <- matrix(ncol = 10, nrow=2e3)
for(j in 1:10)
  sigma2med[,j] <- MCMC.metropolis(target.pdf, proposal.func, runif(1, 0, 30), 2e3)
Ex2.output <- rbind(Ex2.output,
                    data.frame(`Shape` = 'Sigma = 15', `Proposal` = 'Unif(-500, 500)',
                               as.vector(Gelman(sigma2med))))
kable(Ex2.output, align='c')
```

Shape	Proposal	W	B	sigma2.hat	R.hat	n.eff
Sigma = 2	Unif(-0.1, 0.1)	1.307845	6444.79481	4.529588	1.861021	14.05658
Sigma = 2	Unif(-1, 1)	1.903636	55.17710	1.930273	1.006972	699.66463
Sigma = 2	Unif(-25, 25)	1.793602	13.38045	1.799396	1.001614	2689.58815
Sigma = 2	Unif(-100, 100)	1.472887	107.23721	1.525769	1.017794	284.55968
Sigma = 2	Unif(-500, 500)	2.109017	794.65860	2.505292	1.089906	63.05329
Sigma = 15	Unif(-0.1, 0.1)	1.035082	236028.26100	119.048695	10.724447	10.08766
Sigma = 15	Unif(-1, 1)	49.133963	157216.43677	127.717615	1.612258	16.24736
Sigma = 15	Unif(-25, 25)	98.933316	265.99201	99.016846	1.000422	7445.09937
Sigma = 15	Unif(-100, 100)	99.660460	406.56551	99.813913	1.000770	4910.10231
Sigma = 15	Unif(-500, 500)	99.808510	6487.75666	103.002484	1.015875	317.52881

h

How does tuning the proposal distribution influence our sampling from the target distribution? Prepare summary tables or figures to support your discussion.

The table above serves as a good set of summary information for the discussion. The largest take home message here is that the proposal function is what is driving the effectiveness of the Metropolis algorithm. Here, we use a proposal that doesn't have a useful 'shape' in comparison with our target. Thus, shape is less important than variance, although both can matter when performing MCMC.

When we have too small of a variance for our proposal steps, we see a lack of exploration within the distribution. This causes us to highly underestimate the variance with the `Within-chain` estimates, and often our `between-chain` variance will also explode because the chains explore so slowly they do not properly mix. Thus, too small of a perturbation variance causes the distributions to not be fully explored and our estimated variance to be too small.

When we have a perturbation with variance that is much too high, we see much of the same problem. The within chain variance will often become a much worse under-estimate, due to too many stationary movements (remember we can always view the chains). We also see the between chain variance suffers consequences as well from these stationary movements, starting to become much larger again. Thus, proposals with variance that are much too high also give poor results due to many stationary movements, although they do tend to explore okay, they are not doing it effectively.

What we want is a well tuned proposal. When we use proposals that have a better variance in comparison with the target (this is determined through evaluation of diagnostics such as shown in this work), we start to see the MCMC becomes effective for drawing samples, with improved estimation of the target density properties. In this work the **`Unif(-1, 1)`** *and* **`Unif(-25, 25)`** *tended to have these properties. Our Gelman ratio starts converging toward* `1`*, and we see an improvement in our effective sampling.*

We should always be cautious and try several proposals. This work has tried to show the importance of turning the proposal variance.

Exercise 3.

Consider studying the distribution with the following target density,

```
mu <- c(-3, 0, 3)
sd <- 1

f <- function(x)
  0.25  * dnorm(x, mu[1], sd) +
  0.5 * dnorm(x, mu[2], sd) +
  0.25 * dnorm(x, mu[3], sd)
```

The density is the summation of three normals, each with standard deviation $\sigma = 1$. We will want to evaluate how our sampling using MCMC is effected when we change the `sd` value in the above command. We will produce several MCMC chains and evaluate convergence based on Gelman diagnostics.

For each of the following, we will hold constant the proposal function to be $Normal(0, \sigma = 0.5)$ and starting point to be drawn from a $Uniform(-10, 10)$.

a

Ensure `sd <- 1` for the target density. Produce 5 chains each of 10000 (i.e. `1e4`) steps.

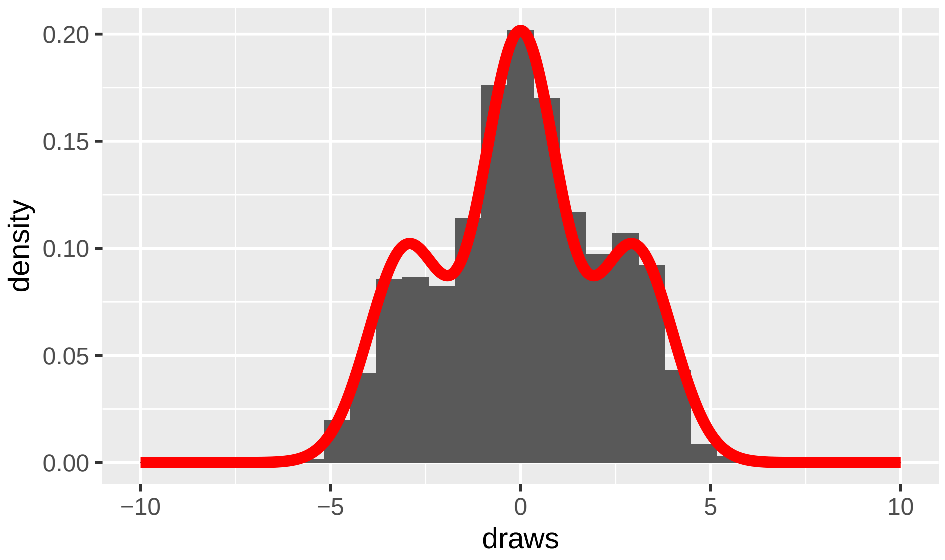
```
proposal.func <- function(x) x + rnorm(1, 0, 0.5)
sd <- 1
mu <- c(-3, 0, 3)
f <- function(x)
  0.25  * dnorm(x, mu[1], sd) +
  0.5 * dnorm(x, mu[2], sd) +
  0.25 * dnorm(x, mu[3], sd)
```

```
results <- matrix(ncol = 5, nrow=1e4)
for(j in 1:5)
  results[,j] <- MCMC.metropolis(f, proposal.func, runif(1, -10, 10), 1e4)
```

b

Compare a chain to the target density graphically.

```r
xx <- seq(-10, 10, length.out = 1e3)
yy <- f(xx)
temp.frame <- data.frame(x = xx, y = yy)
temp.chain <- data.frame(draws = results[,1])
ggplot(temp.chain) + geom_histogram(aes(x = draws, y=after_stat(density))) +
  geom_line(data=temp.frame, aes(x=x, y=y), col='red', lwd=2)
```



c

Evaluate and comment on Gelman diagnostics.

```r
Gelman(results) %>% kable(align='c')
```

W	B	sigma2.hat	R.hat	n.eff
5.139522	105.9707	5.149606	1.000981	2429.731

Our simple visualization made it seem we were doing pretty well, and the Gelman diagnostics agree. This set of chains seems to show convergence, with only 0.3% *variance reduction with continued sampling. The only negative may be that in* 5e3 *steps, we only achieved* 772 *independent draws. This is a difficulty when drawing from a complex distribution; our example isn't even that complex!*

d

Set `sd <- 0.5` and repeat steps **a.** through **c.**.

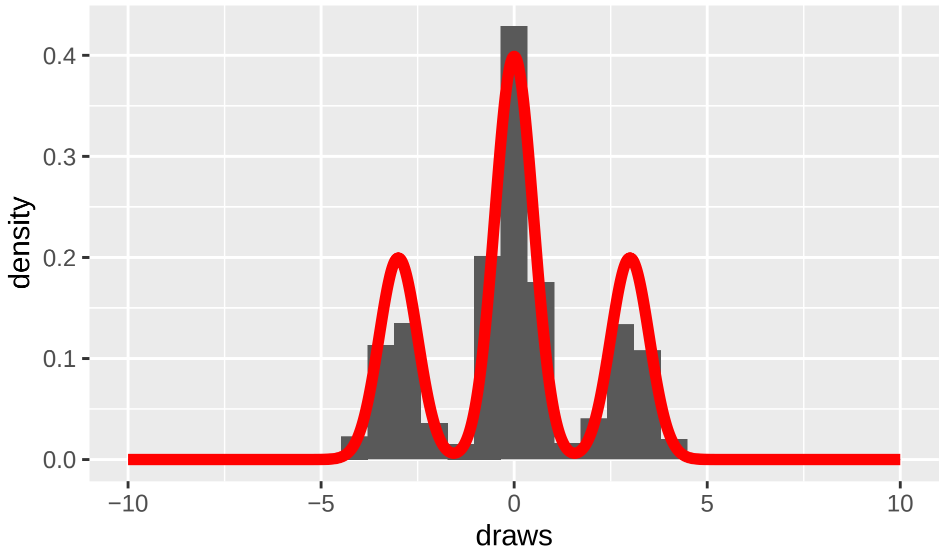
Run my chains.

```r
proposal.func <- function(x) x + rnorm(1, 0, 0.5)
sd <- 0.5
mu <- c(-3, 0, 3)
f <- function(x)
  0.25  * dnorm(x, mu[1], sd) +
  0.5 * dnorm(x, mu[2], sd) +
  0.25 * dnorm(x, mu[3], sd)

results <- matrix(ncol = 5, nrow=1e4)
for(j in 1:5)
  results[,j] <- MCMC.metropolis(f, proposal.func, runif(1, -10, 10), 1e4)
```

Graph and visualize performance.

```r
xx <- seq(-10, 10, length.out = 1e3)
yy <- f(xx)
temp.frame <- data.frame(x = xx, y = yy)
temp.chain <- data.frame(draws = results[,1])
ggplot(temp.chain) + geom_histogram(aes(x = draws, y=after_stat(density))) +
  geom_line(data=temp.frame, aes(x=x, y=y), col='red', lwd=2)
```



Diagnostics.

```r
Gelman(results) %>% kable(align='c')
```

W	B	sigma2.hat	R.hat	n.eff
4.766809	1086.622	4.874994	1.011284	224.3187

Visualization again shows we seem to be sampling well, however the diagnostics have gotten a little worse. Even by spreading these bells apart just a bit, it seems we are no longer obtaining adequate convergence. There is becoming difficulty for the perturbations to adequately move us between the bells.

e

Set `sd <- 0.25` and repeat steps **a.** through **c.**.

Run my chains.

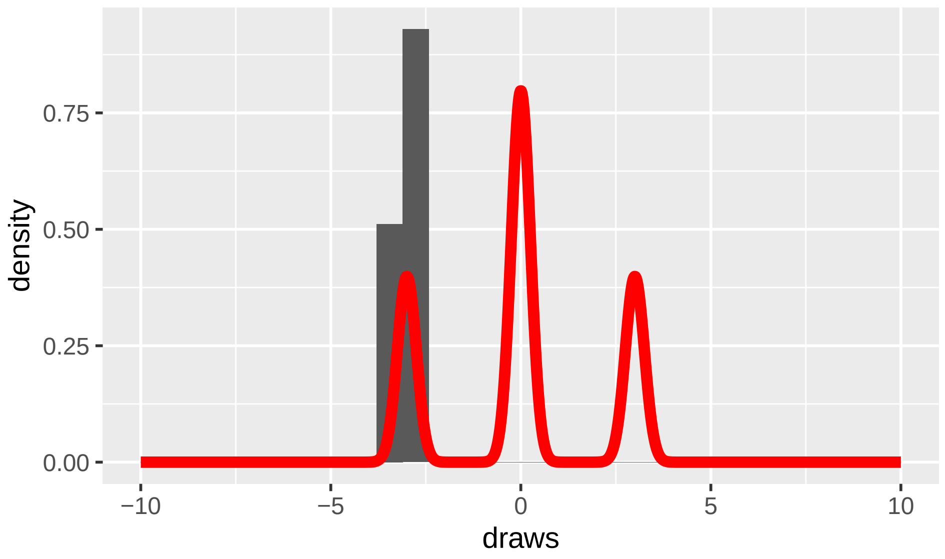
```r
proposal.func <- function(x) x + rnorm(1, 0, 0.5)
sd <- 0.25
```

```r
mu <- c(-3, 0, 3)
f <- function(x)
  0.25  * dnorm(x, mu[1], sd) +
  0.5 * dnorm(x, mu[2], sd) +
  0.25 * dnorm(x, mu[3], sd)

results <- matrix(ncol = 5, nrow=1e4)
for(j in 1:5)
  results[,j] <- MCMC.metropolis(f, proposal.func, runif(1, -10, 10), 1e4)
```

Graph and visualize performance.

```r
xx <- seq(-10, 10, length.out = 1e3)
yy <- f(xx)
temp.frame <- data.frame(x = xx, y = yy)
temp.chain <- data.frame(draws = results[,1])
ggplot(temp.chain) +
  geom_histogram(aes(x = draws, y=after_stat(density))) +
  geom_line(data=temp.frame, aes(x=x, y=y), col='red', lwd=2)
```



Diagnostics.

```r
Gelman(results) %>% kable(align='c')
```

W	B	sigma2.hat	R.hat	n.eff
0.9589984	71634.83	8.122386	2.910267	5.669299

Now we begin to see the MCMC miss completely. This is both a lesson in why we need to randomize our starting points, and the importance of the proposal. If we hadn't been randomly starting in different areas, we might actually find the diagnostics say we are doing good. Why? Because we would just see the result of exploring the same bell over and over. Instead, when we alter our starting points, we tend to end up stuck in a different bell each time, therefore the diagnostics are clear this is not a useful MCMC.

f

Discuss a comparison of MCMC sampling using the three different target density functions. What is going wrong? How are things effected by the proposal function? Comment on the importance of the proposal function when using the Metropolis algorithm. What updates would be required to improve the MCMC sampling?

We could improve sampling in this case by allowing our proposal to have a much larger variance. The key here is that when we moved the bells apart, there are now large regions where we will practically always reject a step into those regions. What we need is a variance that will allow us to jump from bell to bell, the difficulty there is then you may be jumping between the bells too fast. Last year we took the time to explore the proposal in detail, and there is an optimal perturbation, where you will sit in a bell, explore a while, then jump out to a new bell, explore a while, and this continues as the chain grows.

The most important take home messages here: if we don't know the shape/profile of the density, we need randomization in our starting points. We cannot just trust a single starting point will be adequate. The proposal variance is again a major player, just like problem 2, in that variance can control our ability to move between large regions of empty density. Thus, always try a few proposals with small iterations to get an idea of how best to sample with MCMC. Once you do some tuning/exploration, chose a final set of parameters and head to the super computer for lots of iterations.

Exercise 4

Consider simulating the random variable $Z = X + Y$, which is the sum of two independent random variables X and Y. The distribution for Z is unknown, but it is given that

$$X \sim N(\mu = 10, \sigma^2 = 4) \quad \text{and} \quad Y \sim Exp(\lambda = 0.2)$$

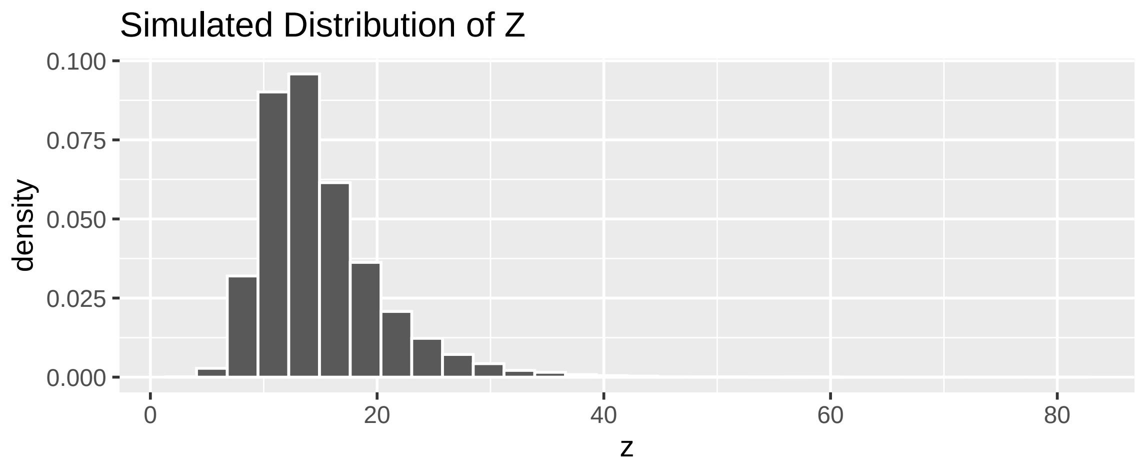
Using the concepts for simulation taught in-class, provide a simulation that estimates the mean, variance, 5th percentile, and 95th percentile of the random variable Z. Base your final answer on $1e5$ iterations. *Hint: You can use* ***`rnorm()`*** *and* ***`rexp()`*** *to draw random samples for X and Y.*

Solution.

```r
samples <- 1e5
x <- rnorm(samples, 10, 2)
y <- rexp(samples, rate = 0.2)
z <- x+y
```

This generates $1e5$ replicates of the given random experiment based on the distributional assumptions of X and Y. We can then extract sample information as estimates of the parameters of Z. I start with visualization of the distribution.

```r
z.df <- data.frame(z = z)
ggplot(z.df, aes(x = z, y = after_stat(density))) + geom_histogram(color = 'white') +
  labs(title = 'Simulated Distribution of Z')
```



```
summary.output <-
  z.df %>% summarize(Mean = mean(z), Variance = var(z),
                     `Q.5th` = quantile(z, 0.05), `Q.95th` = quantile(z, 0.95))
kable(summary.output)
```

Mean	Variance	Q.5th	Q.95th
14.97186	29.11978	8.629776	25.38849

We can calculate the true mean and variance of Z using statistical theory. Expectation is a linear operator thus,

$$E[Z] = E[X + Y] = E[X] + E[Y] = 10 + \frac{1}{0.2} = 10 + 5 = 15$$

We obtained an estimate of 14.97186 after $1e5$ iterations, giving an absolute percent error of

$$\frac{|14.97186 - 15|}{15} \times 100\% \approx 0.188\%$$

Because the random variables X and Y are independent, we can quickly find the variance as well

$$Var[Z] = Var[X + Y] \overset{\text{independent}}{=} Var[X] + Var[Y] = 4 + \frac{1}{(0.2)^2} = 4 + 25 = 29$$

We obtained an estimate of 29.11978 after $1e5$ iterations, giving an absolute percent error of

$$\frac{|29.11978 - 29|}{29} \times 100\% \approx 0.413\%$$

This basic simulation using $1e5$ iterations gives the mean and variance within about 0.5% of the true value. It is not possible to check the quantiles in any easy fashion, but we could expect that our quantiles are likely within $1 - 3\%$ of the population estimate as well. It should be noted that more error is present in the tails of a distribution when simulating, which is often where we want to extract values for a confidence interval. We may still have decent estimates, but the errors will grow as we move away from the mean of the distribution.