homework3

Amy Kuang, Shravan Shenoy - PSTAT 115, Spring 2021

Due on May 23, 2021 at 11:59 pm

Problem 1. Rejection Sampling the Beta distribution. (15 pts)

Assume we did not have access to the `rbeta` function for sampling from a Beta, but we were able to evaluate the density, `dbeta`. This is a very common setting in Bayesian statistics, since we can always evaluate the (proportional) posterior density $p(\theta \mid y) \propto p(y \mid \theta)p(\theta)$ but we don't have immediate access to a method for sampling from this distribution.

1. Let p(x) be a Beta(3, 9) density, $q_1(x)$ a Uniform(0, 1) density, and $q_2(x)$ a Normal($\mu = 0.25, \sigma = 0.15$) density.

```r
sampVec = rbeta(1000,3,9)
p_func <-function(x){dbeta(x, 3, 9)}

Q1 = function(x){dunif(x, 0, 1)}
Q2 = function(x){dnorm(x, 0.25, 0.15)}
```

1. Use rejection sampling to sample from p(x) by proposing samples from $q_1(x)$. To do so, first find $M_1 = \max_x p(x)/q_1(x)$ using the `optimize` function and set `lower=0`, `upper=1`, and `maximum = TRUE` (since we are maximizing not minimizing, the default). M will be the value in the `objective` argument returned by optimize (`maximum` tells us where the maximum occurs, but not what height it achieves). Propose 10000 samples and keep only the accepted samples.

```r
DBetaFunc = function(x){dbeta(x, 3, 9)/dunif(x, 0, 1)}

Optim = optimize(DBetaFunc, lower = 0, upper = 1, maximum = TRUE)
OptObj = Optim$objective

sampleVar = numeric()
Reject = numeric()

for (i in 1:10000){sampleFunc <- runif(1)
  if (runif(1)<DBetaFunc(sampleFunc)/OptObj){
    sampleVar = append(sampleVar, sampleFunc)}
  else{
    Reject = append(Reject, sampleFunc)}}

head(sampleVar)
```

```
## [1] 0.3493505 0.1338830 0.1777596 0.3406022 0.1957928 0.2094330
```

```r
head(Reject)
```

```
## [1] 0.9581892 0.7895988 0.1027326 0.4874408 0.5996962 0.4928667
```

1. Use rejection sampling to sample from p(x) by proposing samples from $q_2(x)$. To do this you need to find $M_2 = \max_x p(x)/q_2(x)$ as above. Propose 10000 samples and keep only the accepted samples.

```r
DBetaFuncAlt = function(x){
  dbeta(x, 3, 9)/dnorm(x, 0.25, 0.15)}

OptDBeta = optimize(DBetaFuncAlt, lower = 0, upper = 1, maximum = TRUE)
OptDBetaObj = OptDBeta$objective

sampleVar2 = numeric()
Reject2 = numeric()

for (i in 1:10000){sample <- runif(1)
  if (runif(1)< DBetaFuncAlt(sample)/OptDBetaObj){
    sampleVar2 = append(sampleVar2, sample)}
  else{
    rejects2 = append(Reject2, sample)}}

head(sampleVar2)
```

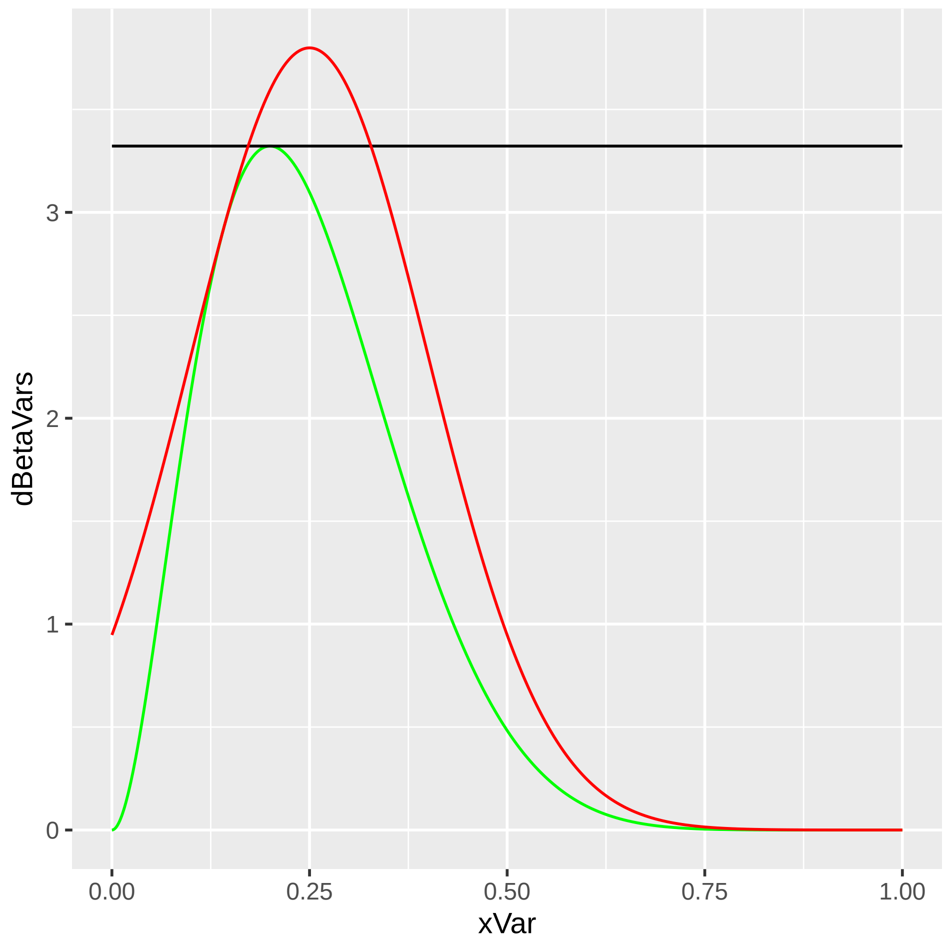
```
## [1] 0.06577053 0.81837427 0.24108655 0.22710431 0.23948399 0.70534238
```

```r
head(rejects2)
```

```
## [1] 0.5943422
```

1. Plot the p(x), $M_1 q_1(x)$ and $M_2 q_2(x)$ all on the same plot and verify visually that the scaled proposal densities "envelope" the target, p(x). Set the xlimits of the plot from 0 to 1. Use different color lines for the various densities so are clearly distinguishable. (5 pts)

```r
xVar = seq(0, 1, by = .001)
dBetaVars = dbeta(xVar, 3, 9)

plotObj1 = OptObj * dunif(xVar)
plotObj2 = OptDBetaObj * dnorm(xVar, 0.25, 0.15)
dat = data.frame(xVar, dBetaVars, plotObj1,plotObj2)

ggplot(data=dat, aes(x=xVar)) +
  geom_line(aes(y=dBetaVars), col = "green") +geom_line(aes(y=plotObj1), col = "black") +
  geom_line(aes(y=plotObj2), col = "red")
```



1. Which rejection sampler had the higher rejection rate? Why does this make sense given the plot from the previous part? This means when proposing 10000 samples from each proposal, the Monte Carlo error of our approximation will be higher when proposing from ______ (choose q_1 or q_2). (5 pts)

```
if (length(Reject) > length(Reject2)){print("Q1 will end up with a higher Monte Carlo Error")}
```

```
## [1] "Q1 will end up with a higher Monte Carlo Error"
```

```
if (length(Reject) < length(Reject2)){print("Q2 will end up with a higher Monte Carlo Error")}
```

Given the plot from the previous part, this does make sense. This is indicated by the value for the Q2 curve being closer to p(x) than that of Q1.

1. Report the variance of Beta(3, 9) distribution by computing the variance of the beta samples. How does this compare to the theoretical variance (refer to the probability cheatsheet). (5 pts)

```
var(sampleVar)
```

```
## [1] 0.01376463
```

```
var(sampleVar2)
```

```
## [1] 0.0442327
```

The theoretical variance as shown in the probability cheatsheet is roughly 0.0144. The variance for Q1 is closer to the theoretical yield than that of Q2.

Problem 2. Frequentist Coverage of The Bayesian Posterior Interval. (35 pts)

Suppose that $y_1, .., y_n$ is an IID sample from a $Normal(\mu, 1)$. We wish to estimate μ.

2a. For Bayesian inference, we will assume the prior distribution $\mu \sim Normal(0, \frac{1}{\kappa_0})$ for all parts below. Remember, from lecture that we can interpret κ_0 as the pseudo-number of prior observations with sample mean $\mu_0 = 0$. State the posterior distribution of μ given $y_1, .., y_n$. Report the lower and upper bounds of the 95% quantile-based posterior credible interval for μ, using the fact that for a normal distribution with standard eviation σ, approximately 95% of the mass is between $\pm 1.96\sigma$. (5 pts)

The posterior distribution of_ μ given $y_1, .., y_n$:

$$p(\mu|y;\sigma) \propto L(\mu) * p(\mu)$$

where L(μ) is likelihood and p(μ) is the prior distribution. So,

$$p(\mu|y;\sigma) \propto L(\mu) * p(\mu)$$

$$\propto exp(-\frac{(\bar{y}-\mu)^2}{\frac{2\sigma^2}{n}} * exp(-\frac{(\mu-\mu_0)^2}{2\tau^2}))$$

$$\propto exp(-\frac{(\frac{n}{\sigma^2}+\frac{1}{\tau^2})(\mu-\frac{\frac{n\bar{y}}{\sigma^2}+\frac{\mu_0}{\tau^2}}{\frac{n}{\sigma^2}+\frac{1}{\tau^2}})^2}{2})$$

where $\tau^2 = \frac{1}{k_0}, \sigma^2 = 1$, and $\mu_0 = 0$. Further,

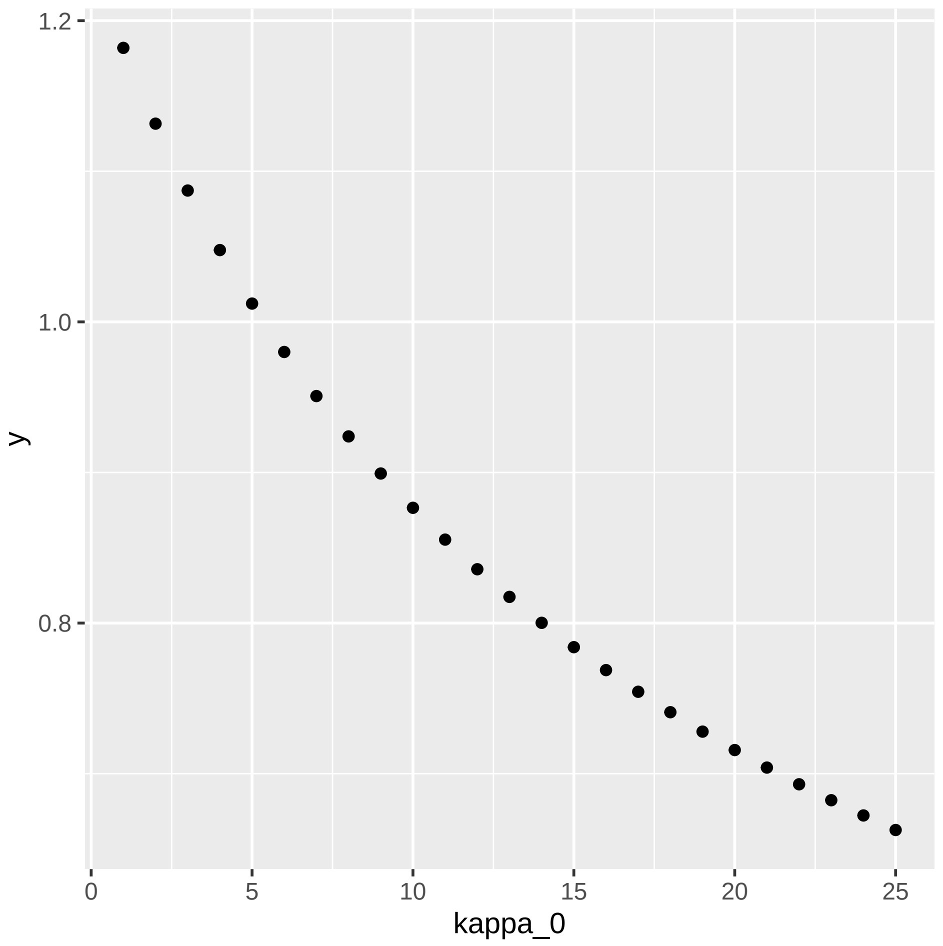
$$\propto exp(-\frac{(\frac{n}{1}+\frac{1}{\frac{1}{k_0}})(\mu-\frac{\frac{n\bar{y}}{1}+\frac{0}{\frac{1}{k_0}}}{\frac{n}{1}+\frac{1}{\frac{1}{k_0}}})^2}{2})$$

$$\propto exp(-\frac{(n+k_0)(\mu-\frac{n\bar{y}}{n+k_0})}{2})$$

$$= N(\frac{n\bar{y}}{n+k_0}, \frac{1}{n+k_0})$$

2b. Plot the length of the posterior credible interval as a function of κ_0, for $\kappa_0 = 1, 2, ..., 25$ assuming $n = 10$. Report how this prior parameter effects the length of the posterior interval and why this makes intuitive sense. (10 pts)

```r
# YOUR CODE HERE
kappa_0 <- c(1:25)
n <- 10
values <- c()

length_post_ci <- function(kappa_0){
    2*1.96*sqrt(1/(n+kappa_0))}

ggplot(data.frame(kappa_0), aes(kappa_0)) +
  stat_function(fun = length_post_ci,
                geom = 'point', n = length(kappa_0))
```



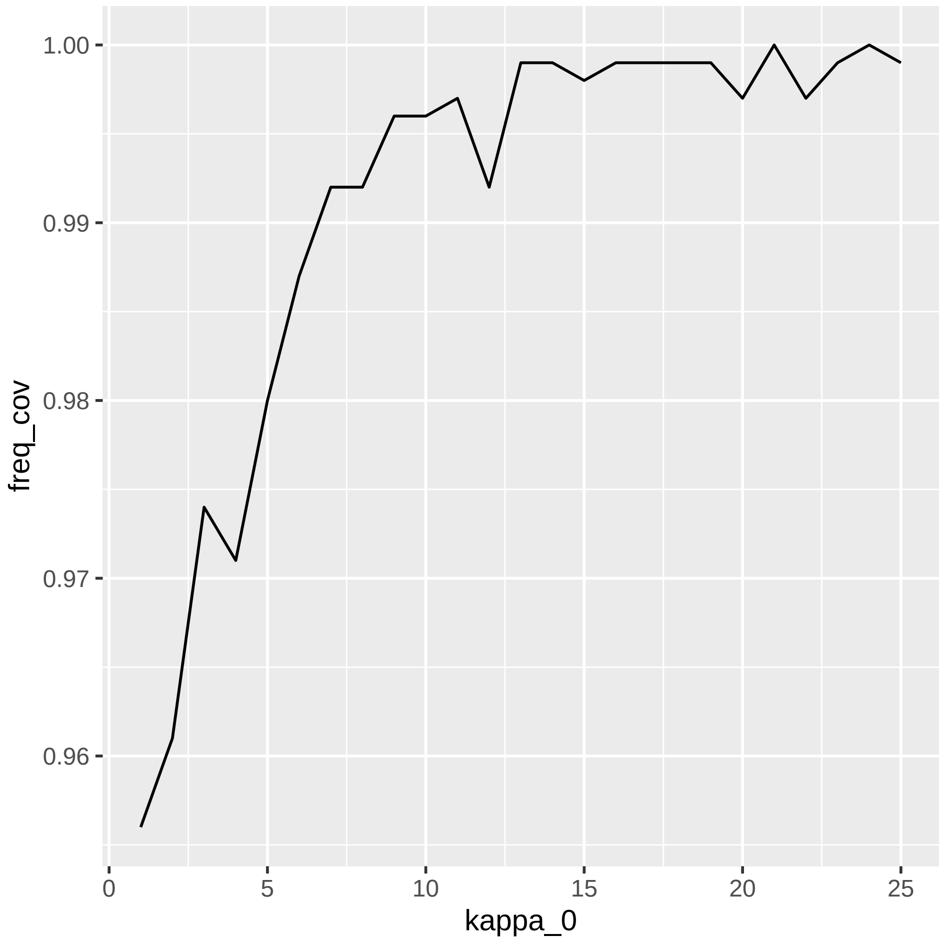
As κ_0 increases, the variance decreases which then leads to the interval length decreasing as well.

2c. Now we will evaluate the *frequentist coverage* of the posterior credible interval on simulated data. Generate 1000 data sets where the true value of $\mu = 0$ and $n = 10$. For each dataset, compute the posterior 95% interval endpoints (from the previous part) and see if it the interval covers the true value of $\mu = 0$. Compute the frequentist coverage as the fraction of these 1000 posterior 95% credible intervals that contain $\mu = 0$. Do this for each value of $\kappa_0 = 1, 2, ..., 25$. Plot the coverage as a function of κ_0. (5 pts)

```
# YOUR CODE HERE
true_mu <- 0
nsims <- 1000
dataset <- matrix(0, nsims, length(kappa_0))
freq_cov <- rep(0, length(kappa_0))

# function w/ posterior mean, frequentist coverage, and
for (k in kappa_0) {
    for (data in seq(nsims)) {
        y <- rnorm(n, true_mu, 1)
        post_mu <- (mean(y)*n/(k+n))
        cred_int <- qnorm(c(0.025,0.975), post_mu, sqrt(1/(k+n)))
        if(between(true_mu,cred_int[1],cred_int[2])==TRUE){
            dataset[data,k] <- 1
        }
    }
    freq_cov[k] <- sum(dataset[,k])/nsims
}

cred_int
```

```
## [1] -0.3628778  0.2997109
```

```
# plot
plot_df <- data.frame(x=kappa_0,y=freq_cov)
ggplot(data=plot_df, aes(x=kappa_0, y=freq_cov)) + geom_line()
```



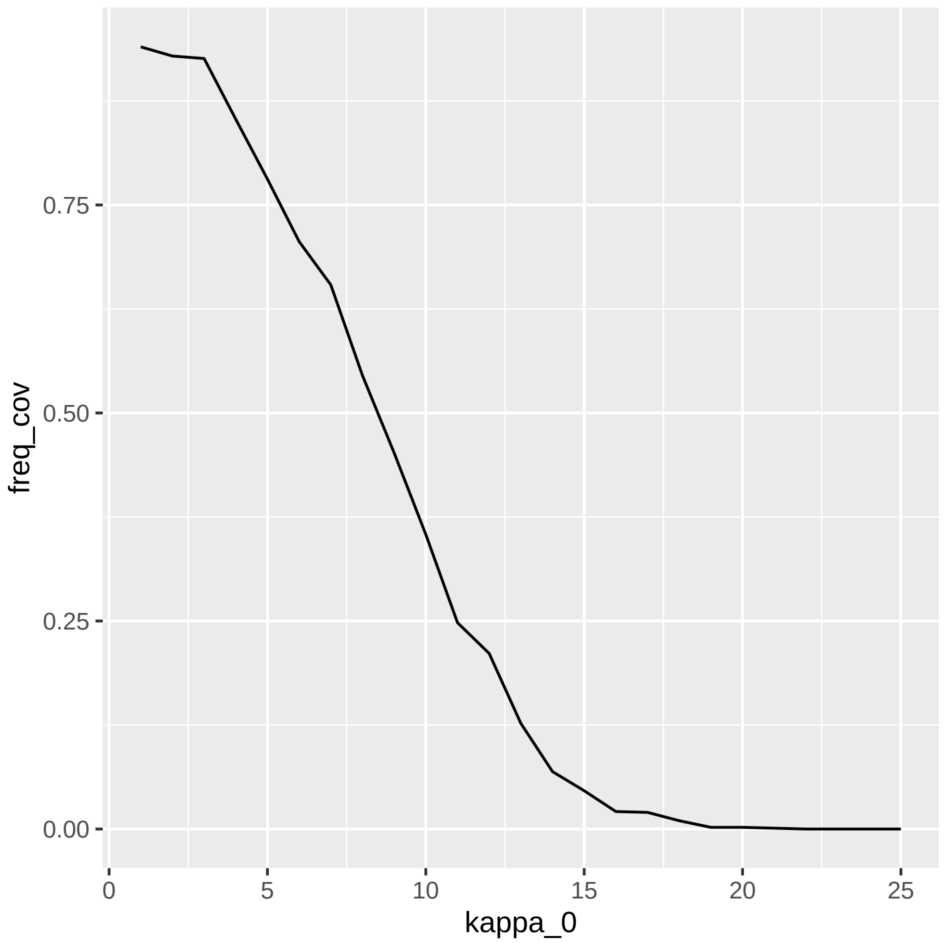
Yes, the interval covers the true value of $\mu = 0$.

2d. Repeat 2c but now generate data assuming the true $\mu = 1$. (5 pts)

```
# YOUR CODE HERE
true_mu <- 1
nsims <- 1000
dataset <- matrix(0, nsims, length(kappa_0))
freq_cov <- rep(0, length(kappa_0))

# function w/ posterior mean, frequentist coverage, and
for (k in kappa_0) {
    for (data in seq(nsims)) {
        y <- rnorm(n, true_mu, 1)
        post_mu <- (mean(y)*n/(k+n))
        cred_int <- qnorm(c(0.025,0.975), post_mu, sqrt(1/(k+n)))
        if(between(true_mu,cred_int[1],cred_int[2])==TRUE){
            dataset[data,k] <- 1
        }
    }
    freq_cov[k] <- sum(dataset[,k])/nsims
}

cred_int
```

```
## [1] 0.1443588 0.8069475
```

```
# plot
plot_df <- data.frame(x=kappa_0,y=freq_cov)
ggplot(data=plot_df, aes(x=kappa_0, y=freq_cov)) + geom_line()
```



No, the interval does not cover the true value of $\mu = 1$.

2e. Explain the differences between the coverage plots when the true $\mu = 0$ and the true $\mu = 1$. For what values of κ_0 do you see closer to nominal coverage (i.e. 95%)? For what values does your posterior interval tend to overcover (the interval covers the true value more than 95% of the time)? Undercover (the interval covers the true value less than 95% of the time)? Why does this make sense? (10 pts)

When μ equals one, the posterior value will cover far more than needed when κ_0 is more than 2. When μ equals zero, the posterior value's interval of coverage becomes less as the value approaches 25, and will result in far less being covered over time. As a result, we are able to conclude that μ is equal to zero.

Problem 3. Bayesian inference for the normal distribution in Stan. (50pts)

Create a new Stan file by selecting "Stan file" in the Rstudio menu. Save it as `IQ_model.stan`. We will make some basic modifications to the template example in the default Stan file for this problem. Consider the IQ example used from class. Scoring on IQ tests is designed to yield a N(100, 15) distribution for the general population. We observe IQ scores for a sample of n individuals from a particular town, $y_1, \ldots y_n \sim N(\mu, \sigma^2)$. Our goal is to estimate the population mean in the town. Assume the $p(\mu, \sigma) = p(\mu \mid \sigma)p(\sigma)$, where $p(\mu \mid \sigma)$ is $N(\mu_0, \sigma/\sqrt{\kappa_0})$ and $p(\sigma)$ is Gamma(a, b). Before you administer the IQ test you believe the town is no different than the rest of the population, so you assume a prior mean for μ of $\mu_0 = 100$, but you aren't to sure about this a priori and so you set $\kappa_0 = 1$ (the effective number of pseudo-observations). Similarly, a priori you assume σ has a mean of 15 (to match the intended standard deviation of the IQ test) and so you decide on setting $a = 15$ and $b = 1$ (remember, the mean of a Gamma is a/b). Assume the following IQ scores are observed:

```r
y <- c(70, 85, 111, 111, 115, 120, 123)
n <- length(y)
```

```r
input_dat = list(N= n, y= y)
stan_fit = stan(file = "IQ_model.stan", data=input_dat)
```

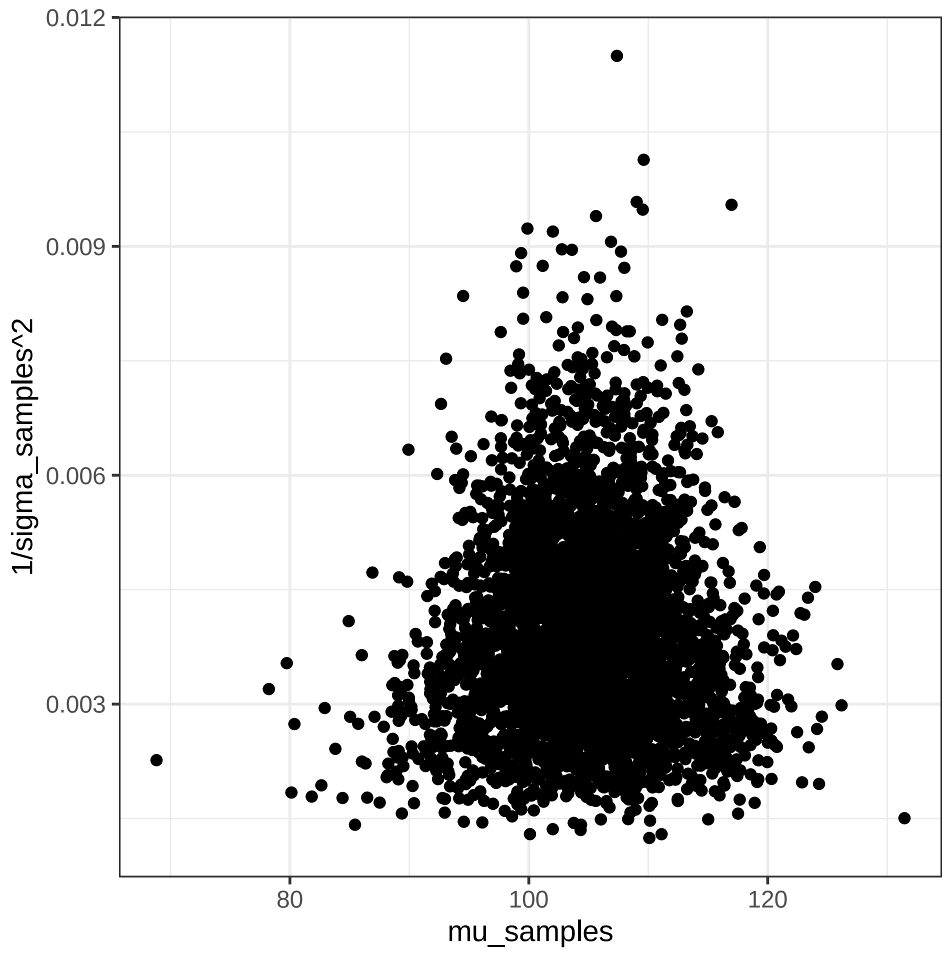
```
## Trying to compile a simple C file

## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Library/Frameworks/R.frame
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/inclu
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
##                ^
##                ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/Core:96:10: fa
## #include <complex>
##          ^~~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL 'IQ_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 1.3e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.13 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
```

```
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.018764 seconds (Warm-up)
## Chain 1:                0.010985 seconds (Sampling)
## Chain 1:                0.029749 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'IQ_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 5e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.05 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.017244 seconds (Warm-up)
## Chain 2:                0.011313 seconds (Sampling)
## Chain 2:                0.028557 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'IQ_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 5e-06 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.05 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.018323 seconds (Warm-up)
## Chain 3:                0.01298 seconds (Sampling)
## Chain 3:                0.031303 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'IQ_model' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 4e-06 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.04 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.022237 seconds (Warm-up)
## Chain 4:                0.011741 seconds (Sampling)
## Chain 4:                0.033978 seconds (Total)
## Chain 4:
```

```r
samples = extract(stan_fit)
mu_samples = samples$mu
sigma_samples = samples$sigma
```

3a. Make a scatter plot of the posterior distribution of the median, μ, and the precision, $1/\sigma^2$. Put μ on the x-axis and $1/\sigma^2$ on the y-axis. What is the posterior relationship between μ and $1/\sigma^2$? Why does this make sense? *Hint:* review the lecture notes. (10pts)

```r
ggplot(data = data.frame(samples)) +
geom_point(aes(x=mu_samples,y=1/sigma_samples^2)) +
theme_bw()
```



When_ $1/\sigma^2$ is high, μ has low variance in the posterior. Similarly, if $1/\sigma^2$ is low, μ _has high variance. This makes sense since variance tells us how spread out the data is.

3b. You are interested in whether the mean IQ in the town is greater than the mean IQ in the overall population. Use Stan to find the posterior probability that μ is greater than 100. (20pts)

```
library(rstan)
y <- c(70, 85, 111, 111, 115, 120, 123)
n <- length(y)

# YOUR CODE HERE
mean(mu_samples > 100)
```

```
## [1] 0.78575
```

3c. You notice that two of the seven scores are significantly lower than the other five. You think that the normal distribution may not be the most appropriate model, in particular because you believe some people in this town are likely have extreme low and extreme high scores. One solution to this is to use a model that is more robust to these kinds of outliers. The Student's t distribution and the Laplace distribution are two so called "heavy-tailed distribution" which have higher probabilities of outliers (i.e. observations further from the mean). Heavy-tailed distributions are useful in modeling because they are more robust to outliers. Fit the model assuming now that the IQ scores in the town have a Laplace distribution, that is $y_1, \ldots, y_n \sim Laplace(\mu, \sigma)$. Create a copy of the previous stan file, and name it "IQ_laplace_model.stan". *Hint:* In the Stan file you can replace `normal` with `double_exponential` in the model section, another name for the Laplce distribution. Like the normal distribution it has two arguments, μ and σ. Keep the same prior distribution, $p(\mu, \sigma)$ as used in the normal model. Under the Laplace model, what is the posterior probability that the median IQ in the town is greater than 100? How does this compare to the probability under the normal model? Why does this make sense? (20pts)

```
# YOUR CODE HERE
input_dat = list(N= n, y= y)
stan_fit2 = stan(file = "IQ_laplace_model.stan", data=input_dat)
```

```
## Trying to compile a simple C file

## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Library/Frameworks/R.fram
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/inclu
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
##                ^
##                ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/Core:96:10: fa
## #include <complex>
##          ^~~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL 'IQ_laplace_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 1.6e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.16 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.020492 seconds (Warm-up)
## Chain 1:                0.014499 seconds (Sampling)
## Chain 1:                0.034991 seconds (Total)
```

```
## Chain 1:
##
## SAMPLING FOR MODEL 'IQ_laplace_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 5e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.05 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.019178 seconds (Warm-up)
## Chain 2:                0.013296 seconds (Sampling)
## Chain 2:                0.032474 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'IQ_laplace_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 4e-06 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.04 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.017712 seconds (Warm-up)
## Chain 3:                0.012861 seconds (Sampling)
## Chain 3:                0.030573 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'IQ_laplace_model' NOW (CHAIN 4).
## Chain 4:
```

```
## Chain 4: Gradient evaluation took 3e-06 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.03 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.019835 seconds (Warm-up)
## Chain 4:                0.012486 seconds (Sampling)
## Chain 4:                0.032321 seconds (Total)
## Chain 4:
```

```r
samples2 = extract(stan_fit2)
mu_samples2 = samples2$mu
sigma_samples2 = samples2$sigma
```

```r
mean(mu_samples2 > 100)
```

```
## [1] 0.955
```

The probability under the normal model is ~0.79 and the probability under the Laplace model is ~0.96, so the probability under the normal model is less than the probability under the Laplace model. This is normal since we believe that some people in this town are likely to have extreme low and extreme high scores. Thus, the Laplace model would be more accurate since it's more robust to these kinds of outliers.