MATH 3190 Homework 6

Focus: Notes 8

Due March 30, 2024

Your homework should be completed in R Markdown or Quarto and Knitted to an html or pdf document. You will "turn in" this homework by uploading to your GitHub Math_3190_Assignment repository in the Homework directory.

Some of the parts in problems 1 and 2 require writing down some math-heavy expressions. You may either type it up using LaTeX style formatting in R Markdown, or you can write it by hand (neatly) and include pictures or scans of your work in your R Markdown document.

Problem 1 (10 points)

Three airlines serve a small town in Ohio. Airline A has 52% of all scheduled flights, airline B has 35% and airline C has the remaining 13%. Their on-time rates are 85%, 67%, and 41%, respectively. A flight just left on-time. What is the probability that it was a flight of airline A?

The method I used here is based on the example from page 4 of Notes 8.

$$\frac{(0.52)(0.85)}{(0.35)(0.67) + (0.52)(0.85) + (0.13)(0.41)}$$

$$= \frac{0.442}{0.7298}$$

$$= 0.6056$$

Problem 2 (13 points)

Suppose we have a data set with each observation x_i independent and identically exponentially distributed for $i = 1, 2, \ldots, n$. That is, $x_i \sim \text{Exp}(\lambda)$ where λ is the rate parameter. We would like to find a posterior (or at least a function proportional to it) for λ.

Part a (5 points)

Write down the likelihood function (or a function proportional to it) in this situation. We would call this $p(x|\lambda)$.

$p(x|\lambda) = \prod_{i=1}^{n} \lambda e^{-\lambda x_i}$

Part b (5 points)

Now let λ have a normal prior with mean 0.1 and variance 1: $\lambda \sim N(1/10, 1)$. Use this and the likelihood from part a to write down a function that is proportional to the posterior of λ given $\boldsymbol{x}$. We call this $p(\lambda|\boldsymbol{x})$.

$exp\{-\frac{1}{2}(\lambda - 0.1)^2\} * \prod_{i=1}^{n} \lambda e^{-\lambda x_i}$

Part c (3 points)

Which would be more appropriate here to obtain samples of λ, the Gibbs or Metropolis algorithm? Explain why. You may want to look on page 8 of Notes 8 in the conjugate prior table.

Metropolis sampling, because we do not have a conjugate prior

Problem 3 (26 points)

Suppose we have the vector `x = c(1.83, 1.72, 2.13, 2.49, 0.90, 2.01, 1.51, 3.12, 1.29, 1.54, 2.94, 3.02, 0.93, 2.78)` that we believe comes from a gamma distribution with shape of 10 and some rate β: $x_i \sim \text{Gam}(10, \beta)$. We will use sampling to obtain some information about β. Let's put a gamma prior on β with a shape of α_0 and a rate of 1: $\beta \sim \text{Gam}(\alpha_0, 1)$.

Part a (5 points)

Use the fact that this is a conjugate prior to write down what kind of distribution the posterior of β, which is $p(\beta|\boldsymbol{x})$, is.

```r
x <-  c(1.83, 1.72, 2.13, 2.49, 0.90, 2.01, 1.51, 3.12, 1.29, 1.54, 2.94, 3.02, 0.93, 2.78)
n <- length(x)
beta0 <- 1
alpha <- 10
rate <- beta0 + sum(x)
shape <- n*alpha
```

$p(\beta|\boldsymbol{x}) \sim gam(\alpha_0 + 140, 29.21)$

Part b (5 points)

Let $\alpha_0 = 1$. In an **R** code chunk, sample 10,000 β values from the distribution you wrote down in part a using the `rgamma()` function and report the 95% credible interval for β using the 2.5th and 97.5th percentiles.

```r
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.3.2

## Warning: package 'ggplot2' was built under R version 4.3.2

## Warning: package 'tibble' was built under R version 4.2.3

## Warning: package 'tidyr' was built under R version 4.3.2

## Warning: package 'readr' was built under R version 4.3.2

## Warning: package 'purrr' was built under R version 4.3.2

## Warning: package 'dplyr' was built under R version 4.2.3
```

```
## Warning: package 'stringr' was built under R version 4.3.2

## Warning: package 'forcats' was built under R version 4.3.2

## Warning: package 'lubridate' was built under R version 4.3.2

## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
credInts1 <- tibble(alpha0 = numeric(),
               lBound = numeric(),
               uBound = numeric())

alphas <- c(1, 10, 100)
alpha <- 10
beta <- 1
for (i in alphas){
  shape <- i + length(x)*alpha
  rate <- beta0 + sum(x)
  beta <- rgamma(10000, shape, rate)
  credInt <- quantile(beta, c(0.025, 0.975))
  credInts1 <- add_row(credInts1,
    alpha0 =  i,
    lBound = credInt[1],
    uBound = credInt[2]
  )

}
print(credInts1)
```

```
## # A tibble: 3 x 3
##   alpha0 lBound uBound
##    <dbl>  <dbl>  <dbl>
## 1      1   4.06   5.66
## 2     10   4.34   5.99
## 3    100   7.21   9.27
```

Part c (3 points)

Repeat part b with $\alpha_0 = 10$.

See credInts1 table from part a

Part d (3 points)

Repeat part b with $\alpha_0 = 100$.

See credInts1 table from part a

Part e (7 points)

Now suppose we have twice as much data (given in the **R** code chunk below). Repeat parts b, c, and d using this `x` vector instead and report the three 95% credible intervals. Note, this new vector x will change the shape and rate parameters used in the `rgamma()` functions.

```
x <- c(1.83, 1.72, 2.13, 2.49, 0.90, 2.01, 1.51, 3.12, 1.29, 1.54,
       2.94, 3.02, 0.93, 2.78, 2.76, 1.70, 1.42, 2.16, 1.07, 2.21,
       2.38, 2.27, 1.72, 1.44, 1.54, 1.72, 1.87, 1.39)
```

```
credInts2 <- tibble(alpha0 = numeric(),
               lBound = numeric(),
               uBound = numeric())

alphas <- c(1, 10, 100)
alpha <- 10
beta <- 1
for (i in alphas){
  shape <- i + length(x)*alpha
  rate <- beta0 + sum(x)
  beta <- rgamma(10000, shape, rate)
  credInt <- quantile(beta, c(0.025, 0.975))
  credInts2 <- add_row(credInts2,
    alpha0 =  i,
    lBound = credInt[1],
    uBound = credInt[2]
  )

}
print(credInts2)
```

```
## # A tibble: 3 x 3
##   alpha0 lBound uBound
##    <dbl>  <dbl>  <dbl>
## 1      1   4.54   5.73
## 2     10   4.70   5.91
## 3    100   6.24   7.64
```

Part f (3 points)

In this problem, the true β value is 5. Write a sentence or two about the effect adding more data has to these credible intervals by comparing the intervals from parts b-d to the intervals from part e.

It looks like adding more data made the bounds of the credible intervals narrower. The intervals made with $\alpha = 1$ and $\alpha = 10$ both contain 5, but the ones with $\alpha = 100$ do not contain 5. It seems like adding more data makes the intervals more precise for all values of α.

Problem 4 (51 points)

Let's apply the Bayesian framework to a regression problem. In the GitHub data folder, there is a file called `treeseeds.txt` that contains information about species of tree, the count of seeds it produces, and the average weight of those seeds in mg.

Part a (3 points)

Read in the `treeseeds.txt` file and take the log of the counts and weights. Fit an OLS regression model using log(weight) to predict log(count).

```r
tree <- read_csv("treeseeds.txt")
```

```
## Rows: 19 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): species
## dbl (2): count, weight
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
tree['count'] = log(tree['count'])
tree['weight'] = log(tree['weight'])
```

```r
treeMod <- lm(count ~ weight, data = tree)
summary(treeMod)
```

```
##
## Call:
## lm(formula = count ~ weight, data = tree)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.4429 -0.3877  0.1778  0.6167  1.2691
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  9.37903    0.39720  23.613 1.95e-14 ***
## weight      -0.51491    0.07185  -7.166 1.58e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9363 on 17 degrees of freedom
## Multiple R-squared:  0.7513, Adjusted R-squared:  0.7367
## F-statistic: 51.35 on 1 and 17 DF,  p-value: 1.58e-06
```

Part b (15 points)

We will walk through the mathematics of obtaining the posterior together here since this problem will focus on coding the Metropolis algorithm. Assuming the true errors are normal with mean 0 and variance σ^2,

$\epsilon_i \sim N(0, \sigma^2)$, it can be shown that each y_i has the distribution

$$p(y_i|x_i, \beta_0, \beta_1\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y_i - \beta_0 - \beta_1 x_i)^2\right)$$

So, we can write the likelihood is

$$p(y_i|x_i, \beta_0, \beta_1\sigma^2) =\propto \exp\left(-\frac{1}{2\sigma^2}(y_i - \beta_0 - \beta_1 x_i)^2\right)$$

where y_i is the log(count) for observation i and x_i is the log(weight) for observation i. Note that here we think of $\boldsymbol{y}$ as being random and $\boldsymbol{x}$ as being fixed. We could, in theory, think of the vector $\boldsymbol{x}$ as also being random and put a prior on it. But we won't do that here.

Now, let's just put uniform priors on β_0 and β_1 so the priors are proportional to 1. Also, let's assume $\sigma^2 = 1$. This seems reasonable since s_e^2, the MSE, is 0.877. Of course, we could put a prior on σ^2 as well and sample it too, but we will focus on only sampling β_0 and β_1.

Now, with those uniform priors, and plugging in 1 for σ^2, we have that the joint posterior of β_0 and β_1 is:

$$p(\beta_0, \beta_1|\boldsymbol{x}, \boldsymbol{y}) =\propto \exp\left(-\frac{1}{2}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2\right) = f(\beta_0, \beta_1|\boldsymbol{x}, \boldsymbol{y}).$$

Then, we can take the log to get

$$\ln(f(\beta_0, \beta_1|\boldsymbol{x}, \boldsymbol{y})) = -\frac{1}{2}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2.$$

Our goal now is to obtain samples of β_0 and β_1. Let's use the Metropolis algorithm to do this. Using the log of the function proportional to the joint posterior of β_0 and β_1, $\ln(f(\beta_0, \beta_1|\boldsymbol{x}, \boldsymbol{y})))$, write a Metropolis algorithm in **R**. For β_0, you can use a normal proposal distribution centered at the previous value, $\beta_0^{(i)}$, with a standard deviation of 0.8 and for β_1, you can use a normal proposal distribution centered at the previous value, $\beta_1^{(i)}$, with a standard deviation of 0.1. The starting values don't matter too much, but we can use $\beta_0^{(0)} = 10$ and $\beta_1^{(0)} = -0.5$. It may be useful to look at the `Notes 8 Script.R` file that is on GitHub in the Notes 8 folder and is on Canvas.

Obtain at least 10,000 samples (set a seed, please) and plot the chains for β_0 and β_1. For this problem, include:

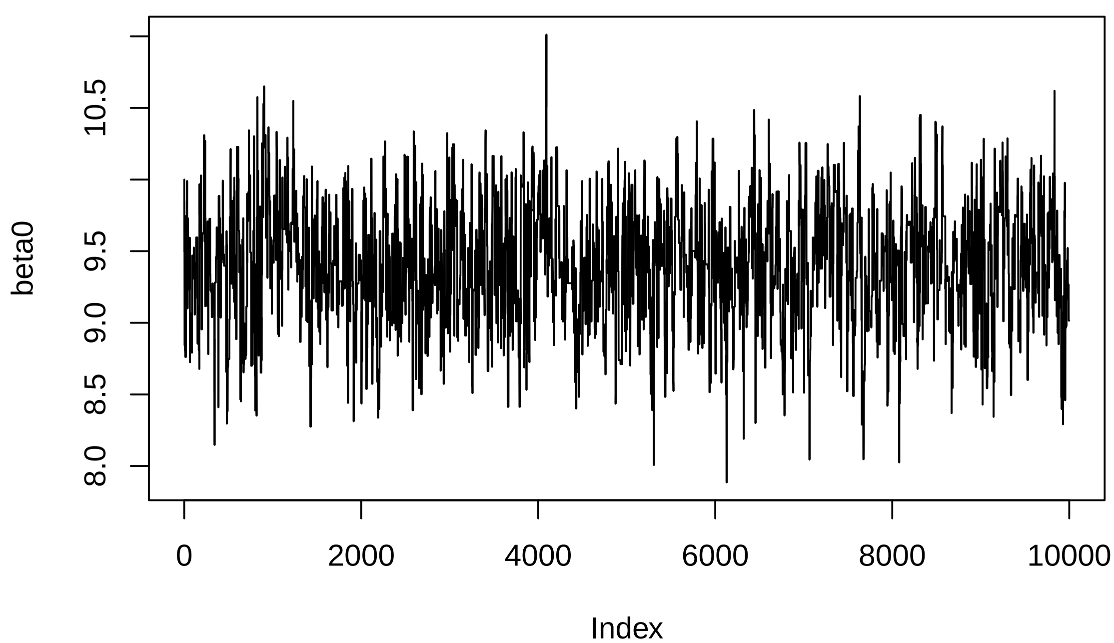
1. The plot for the β_0 chain.
2. The plot for the β_1 chain.
3. The 95% credible interval for β_0 based on the 2.5th and 97.5th percentiles.
4. The 95% credible interval for β_1 based on the 2.5th and 97.5th percentiles.

```r
set.seed(1)
nsamps <- 10000

beta1 <- rep(0, nsamps)
beta0 <- rep(0, nsamps)

beta1[1] <- -0.5
beta0[1] <- 10
```

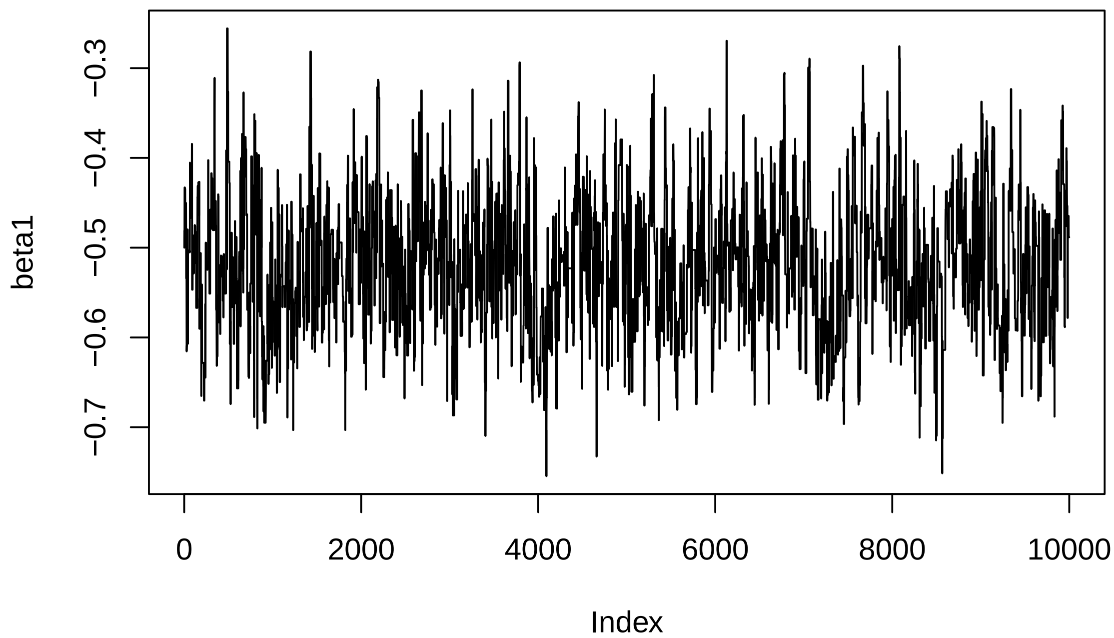
```r
n <- length(tree)
x <- tree$weight
y <- tree$count
```

```r
for(i in 1:(nsamps - 1)){
  star0 <- rnorm(1, beta0[i], 0.8)
  star1 <- rnorm(1, beta1[i], 0.1)

  logF1 <- -1/2*sum((y - star0 - star1*x)^2)
  logF2 <- -1/2*sum((y - beta0[i] - beta1[i]*x)^2)

  if (log(runif(1)) < (logF1 - logF2)){
    beta0[i + 1] <- star0
    beta1[i + 1] <- star1
  } else{
    beta0[i + 1] <- beta0[i]
    beta1[i + 1] <- beta1[i]
  }

}
```

```r
plot(beta0, type = 'l')
```




```r
plot(beta1, type = 'l')
```




```r
int0 <- quantile(beta0, c(.025, .975))
int0
```

```
##      2.5%     97.5%
##  8.545065 10.211279
```

```r
int1 <- quantile(beta1, c(.025, .975))
int1
```

```
##       2.5%      97.5%
## -0.6629883 -0.3576381
```

Part c (3 points)

Based on the plots of the chains from part b, does it look like the Metropolis sampling worked fairly well?

They do look pretty random, which is good. They're not getting stuck at any points. I feel like it could be noisier but I think they're perfectly acceptable.

Part d (4 points)

Interpret both of the credible intervals from part b.

There is a 95% chance that the true β_0 is between 8.545 and 10.211.

There is 95% chance that the true β_1 is between -0.663 and -0.358

Part e (5 points)

Find and report the integrated autocorrelation time for the β_0 and β_1 chains. Each chain will have their own $\hat{\tau}_{int}$ value, so you should report two (although they will be similar).

```
tInt0 <- 1 + 2 * sum(abs(acf(beta0, lag.max = 100, plot = F)$acf))
tInt1 <- 1 + 2 * sum(abs(acf(beta1, lag.max = 100, plot = F)$acf))
tInt0
```

```
## [1] 22.44775
```

```
tInt1
```

```
## [1] 28.65064
```

Part f (3 points)

Based on the integrated autocorrelation time for the β_0 and β_1 chains, how many MCMC samples would you need to generate to get the equivalent of 10,000 independent samples?

```
print(tInt0 * 10000)
```

```
## [1] 224477.5
```

```
print(tInt1 * 10000)
```

```
## [1] 286506.4
```

To get the equivalent of 10,000 independent samples for β_0 we would need 224478 samples and for β_1 we would need 286507 samples.

Part g (3 points)

Let's compare these credible intervals to some other intervals. First, obtain the 95% t confidence intervals for β_0 and β_1 just using the `confint()` function and report them here.

```
tconf <- confint(treeMod, level = 0.95)
tconf0 <- tconf["(Intercept)", ]
tconf1 <- tconf["weight", ]
```

Part h (10 points)

Now let's obtain confidence intervals using bootstrapping in a similar way we did with regularization in Notes 7 and HW 4 (this is known as bootstrapping the cases). Set a seed and then using at least 10,000 bootstrap samples, report the 95% percentile confidence intervals for β_0 and β_1 using the `quantile()` function on the values of β_0 and β_1 that you obtained in the bootstrap.

```r
set.seed(1)
n <- 10000
betas <- matrix(rep(0, 2*n), nrow = n)

for(i in 1:n){
  index <- sample(1:nrow(tree), nrow(tree), replace = T)
  betas[i, ] <- coef(
    lm(count ~ weight, data = tree[index, ])
  )
}
```

```r
bStrap0 <- quantile(betas[,1], c(0.025, 0.975))
bStrap1 <- quantile(betas[,2], c(0.025, 0.975))
```

Part i (5 points)

Write a couple sentences comparing all of the intervals in parts b, g, and h.

```r
ciDf <- data.frame(
  Variable = c("Beta0 (credible)", "Beta1 (credible)",
               "Beta0 (T Interval)", "Beta1 (T Interval)",
               "Beta0 (bootstrap)", "Beta1 (boostrap)"),
  Lower = c(int0[1], int1[1], tconf0[1], tconf1[1], bStrap0[1], bStrap1[1]),
  Upper = c(int0[2], int1[2], tconf0[2], tconf1[2], bStrap0[2], bStrap1[2])
)

# Print the table
print(ciDf)
```

```
##             Variable      Lower      Upper
## 1   Beta0 (credible)  8.5450650 10.2112793
## 2   Beta1 (credible) -0.6629883 -0.3576381
## 3 Beta0 (T Interval)  8.5410148 10.2170375
## 4 Beta1 (T Interval) -0.6665051 -0.3633051
## 5  Beta0 (bootstrap)  8.3897056 10.3120985
## 6   Beta1 (boostrap) -0.6682477 -0.3524438
```

The intervals are actually pretty similar, especially the prediction intervals and the T-intervals. They differ only at roughly 3 decimal places, except for the β_1 intervals. Even then, they are not too different. The bootstrap intervals are also pretty similar to the T-intervals and the prediction intervals, although not as similar. They seem to agree on where the true values of β_0 and β_1 are.