Carr Exam 1

Garrett Carr

Due: 2/22/2022 At Midnight

This is a take-home exam. The exam is due on Thursday, February 22, at midnight. Please complete the exam using a Markdown file. Please email your completed exam to mckayc.647@gmail.com prior to the deadline by replying to the email chain that this exam arrived in. Please email both your Markdown file (.Rmd) and your output file. Please knit the document as a PDF. Your files should be named 'lastname_exam1.Rmd' and 'lastname_exam1.PDF' where lastname is replaced with your last name. Make sure you show your code as well as your answers. Include necessary comments. I am not concerned about how you choose to do this, as long as all code and output are included in the exam document in the appropriate order. Double check that all of your code and output is visible in your output file.

Take-home exams should be your own work. However, you are welcome to use class notes, and help documentation publicly available for all the programs we have used. You should not search the web for similar problems which someone else may have solved. You should not discuss the exam with any living person. The data for the exam are in data files that I will email to you.

For the first set of problems use the data file 'exam1-1.dat'. There are 24 data points in 8 treatments. We will assume the likelihood for the data is normal. You should assume that the variance is homoscedastic across treatments (that is, the variance is the same in all the treatments). Use the following priors: Normal(mean=5,prec=.0001) for the cell means, and a gamma(shape=1.5,rate=.5) for the variance. Besides examining the posteriors of the eight cell means and the variance, you will also be examining three other functions of the parameters: (1) the average of the first four cell means, (2) the average of the last four cell means, and (3) the average of the last four cell means minus the average of the first four cell means.

1. Write the JAGS code necessary to produce posterior chains for the eight cell means and the variance. Put a set.seed(1234) command in the file prior to running the JAGS code so that we will all get the same answers. Run 4 chains with 11000 iterations per chain, a burnin of 1000 and thin by 4. This will result in 10000 samples. Print out the JAGS output file.

```
mdl1 <- "
  model {

    for (i in 1:24) {
      y[i] ~ dnorm(mu[tmt[i]], 1/vr)
    }

    for (i in 1:8) {
      mu[i] ~ dnorm(5, 0.0001)
    }

    vr ~ dgamma(1.5, 0.5)
}
"
```

```
writeLines(mdl1, 'exam1.txt')
y <- dat$y
tmt <- dat$tmt

data.jags <- c('y', 'tmt')
parms <- c('mu', 'vr')

set.seed(1234)
dat.sim <- jags(data = data.jags, inits = NULL, parameters.to.save = parms, model.file='exam1.txt',
                n.chains = 4, n.iter = 11000, n.burnin = 1000, n.thin = 4)
```

```
## module glm loaded

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 24
##    Unobserved stochastic nodes: 9
##    Total graph size: 63
##
## Initializing model
```

```
dat.sim
```

```
## Inference for Bugs model at "exam1.txt", fit using jags,
##  4 chains, each with 11000 iterations (first 1000 discarded), n.thin = 4
##  n.sims = 10000 iterations saved
##          mu.vect sd.vect   2.5%    25%    50%    75%  97.5%  Rhat n.eff
## mu[1]      2.111   0.850  0.436  1.563  2.100  2.670  3.796 1.001 10000
## mu[2]      3.099   0.845  1.425  2.524  3.108  3.651  4.806 1.001  7000
## mu[3]      3.045   0.830  1.410  2.507  3.048  3.590  4.678 1.001 10000
## mu[4]      4.712   0.835  3.064  4.152  4.705  5.256  6.356 1.001 10000
## mu[5]      5.197   0.842  3.523  4.647  5.202  5.740  6.888 1.001 10000
## mu[6]      6.097   0.842  4.435  5.564  6.097  6.633  7.785 1.001 10000
## mu[7]      5.881   0.845  4.175  5.338  5.880  6.428  7.552 1.001  8900
## mu[8]      5.721   0.838  4.046  5.175  5.724  6.277  7.346 1.001  8700
## vr         2.129   0.799  1.042  1.572  1.967  2.504  4.067 1.001 10000
## deviance  83.918   5.258 75.948 80.022 83.152 87.072 96.095 1.001  5300
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 13.8 and DIC = 97.7
## DIC is an estimate of expected predictive error (lower deviance is better).
```

2. Using coda, verify that the chains produced for the eight cell means, the variance, and the three functions of the parameters described above are appropriate for further analysis by showing that the upper confidence interval of the Gelman-Rubin diagnostic is close to 1.

```
library(coda)
sims <- as.mcmc(dat.sim)

gelman.diag(sims)
```

```
## Potential scale reduction factors:
##
##          Point est. Upper C.I.
## deviance          1          1
## mu[1]             1          1
## mu[2]             1          1
## mu[3]             1          1
## mu[4]             1          1
## mu[5]             1          1
## mu[6]             1          1
## mu[7]             1          1
## mu[8]             1          1
## vr                1          1
##
## Multivariate psrf
##
## 1
```

3. Using coda, verify that the chains produced for the eight cell means, the variance, and the three functions of the parameters described above are appropriate for further analysis by showing that the effective sample size for each chain exceeds 5000.

```
effectiveSize(sims)
```

```
##  deviance     mu[1]     mu[2]     mu[3]     mu[4]     mu[5]     mu[6]     mu[7]
##  8563.954 10000.000 10294.170  9855.434 10618.021  9851.026 10000.000 10409.513
##     mu[8]        vr
## 10000.000  7311.326
```

4. Using coda, verify that the chains produced for the eight cell means, the variance, and the three functions of the parameters described above are appropriate for further analysis by showing that the Raftery-Lewis diagnostic for each chain is smaller than 3.

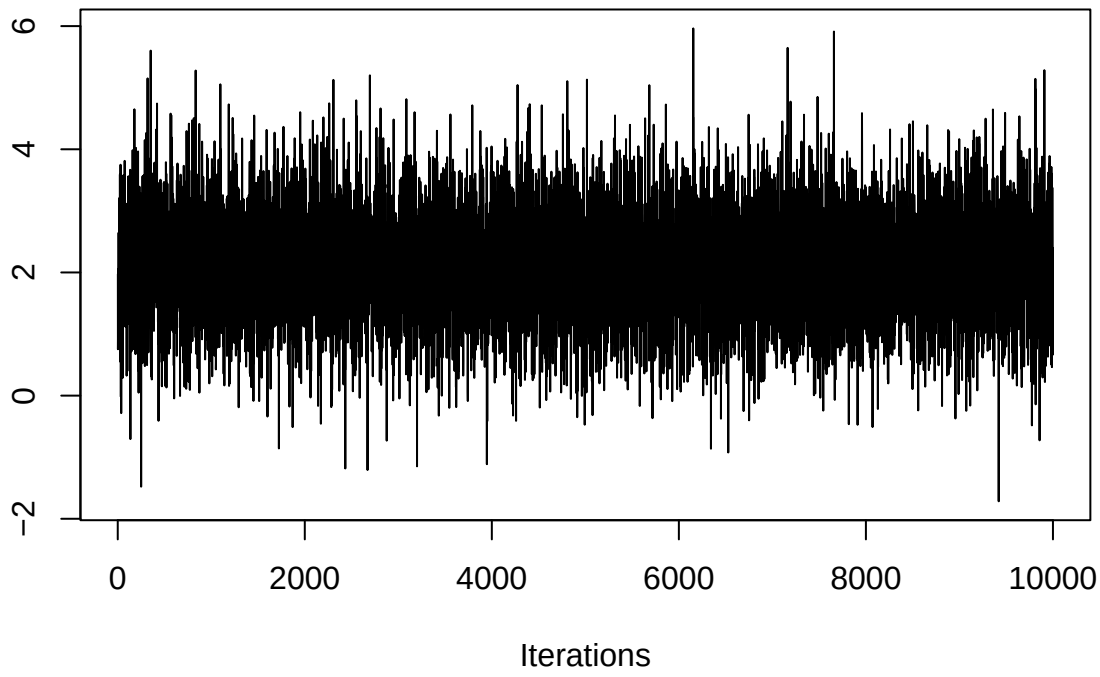
```
chains <- as.matrix(sims)
tmt14 <- as.mcmc(rowMeans(chains[,2:5]))
tmt58 <- as.mcmc(rowMeans(chains[,6:9]))
tmtdiff <- as.mcmc(rowMeans(chains[,6:9])-rowMeans(chains[,2:5]))
chains <- cbind(chains,as.matrix(tmt14), as.matrix(tmt58), as.matrix(tmtdiff))
raftery.diag(chains)
```

```
##
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.005
## Probability (s) = 0.95
##
##           Burn-in  Total Lower bound  Dependence
```

```
##           (M)      (N)   (Nmin)       factor (I)
##  deviance 2        3680  3746         0.982
##  mu[1]    2        3865  3746         1.030
##  mu[2]    2        3802  3746         1.010
##  mu[3]    2        3962  3746         1.060
##  mu[4]    2        3834  3746         1.020
##  mu[5]    2        3710  3746         0.990
##  mu[6]    2        3771  3746         1.010
##  mu[7]    2        3771  3746         1.010
##  mu[8]    2        3802  3746         1.010
##  vr       2        3834  3746         1.020
##  var1     2        3680  3746         0.982
##  var1     2        3650  3746         0.974
##  var1     2        3680  3746         0.982
```

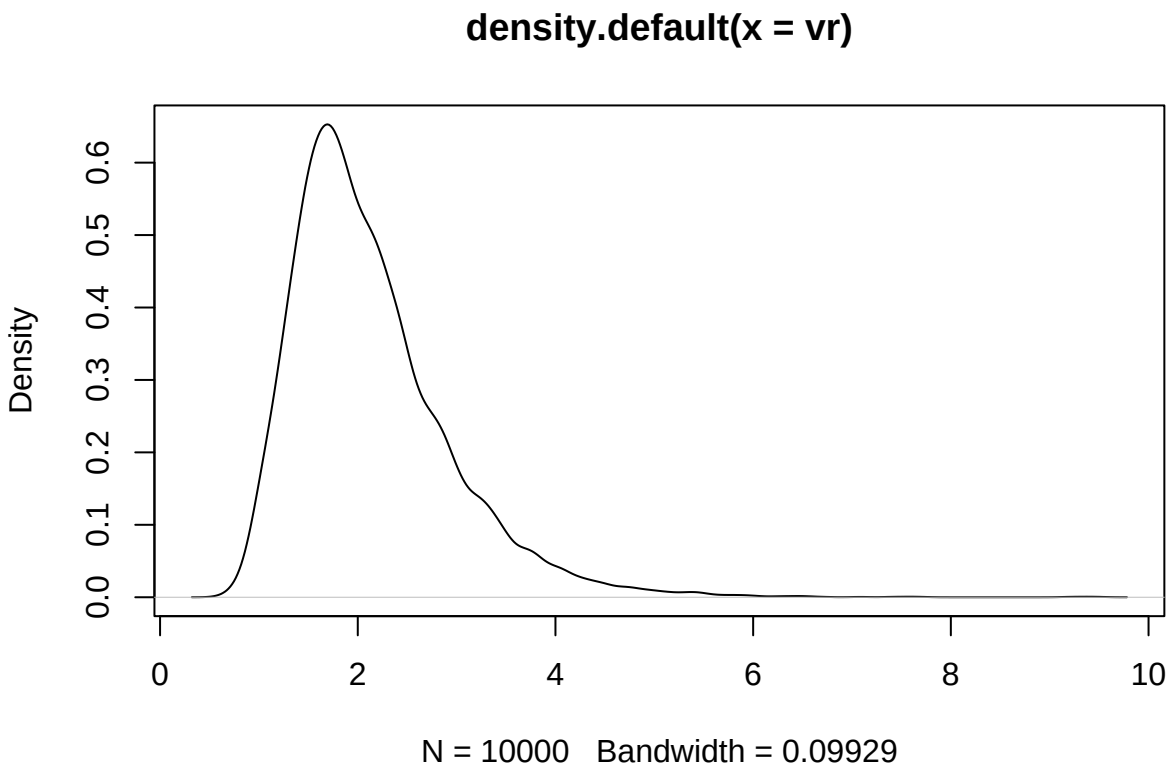
5. Produce the trace plot for the mean parameter of treatment 1.

```
tmt1 <- as.mcmc(chains[,2])
traceplot(tmt1)
```



6. Produce the density plot for the variance parameter.

```
vr <- chains[,10]
plot(density(vr))
```



7. What is the equal tail 95% posterior probability interval of the variance.

```
quantile(vr, c(0.025, 0.975))
```

```
##     2.5%    97.5%
## 1.041901 4.066660
```

8. What is the highest posterior density 95% interval of the variance.

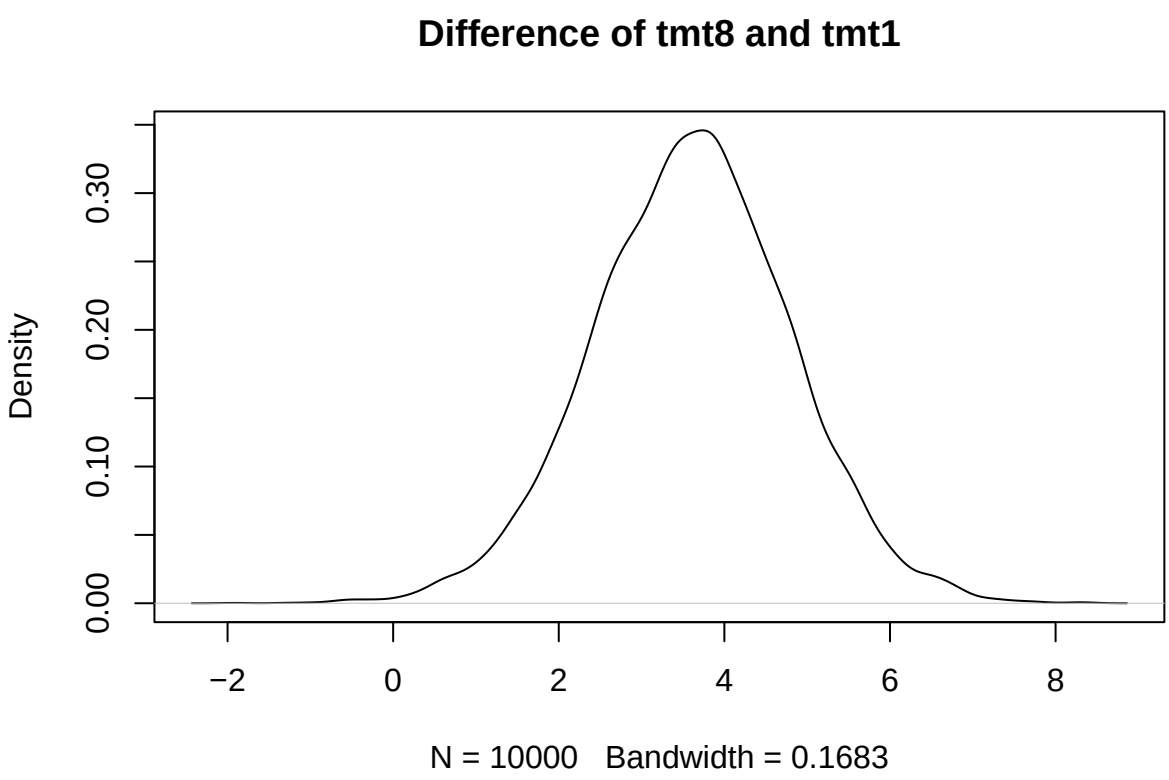
```
HPDinterval(as.mcmc(vr), prob = 0.95)
```

```
##          lower    upper
## var1 0.8905975 3.701743
## attr(,"Probability")
## [1] 0.95
```

9. Say we want to know if the mean for treatment 1 is different than the mean of treatment 8. Compute the chain that represents the difference of the mean of treatment 8 minus the mean of treatment 1. Plot the density of this chain.

```
tmt8 <- chains[,9]
tmt81 <- tmt8 - tmt1

plot(density(tmt81), main = 'Difference of tmt8 and tmt1')
```



10. Would you conclude the mean of treatment 8 exceeds the mean of treatment 1? Why?

Yes, because the bulk of the posterior distribution is above zero. In fact, we could conclude that with an estimated probability of 0.9977.

11. Compute pD using the JAGS formula using one of the chains you have already produced.

```
jags.samples(dat.sim$model, 'pD', n.iter = 11000)
```

```
## $pD
## mcarray:
## [1] 10.03316
##
## Marginalizing over: iteration(11000)
```

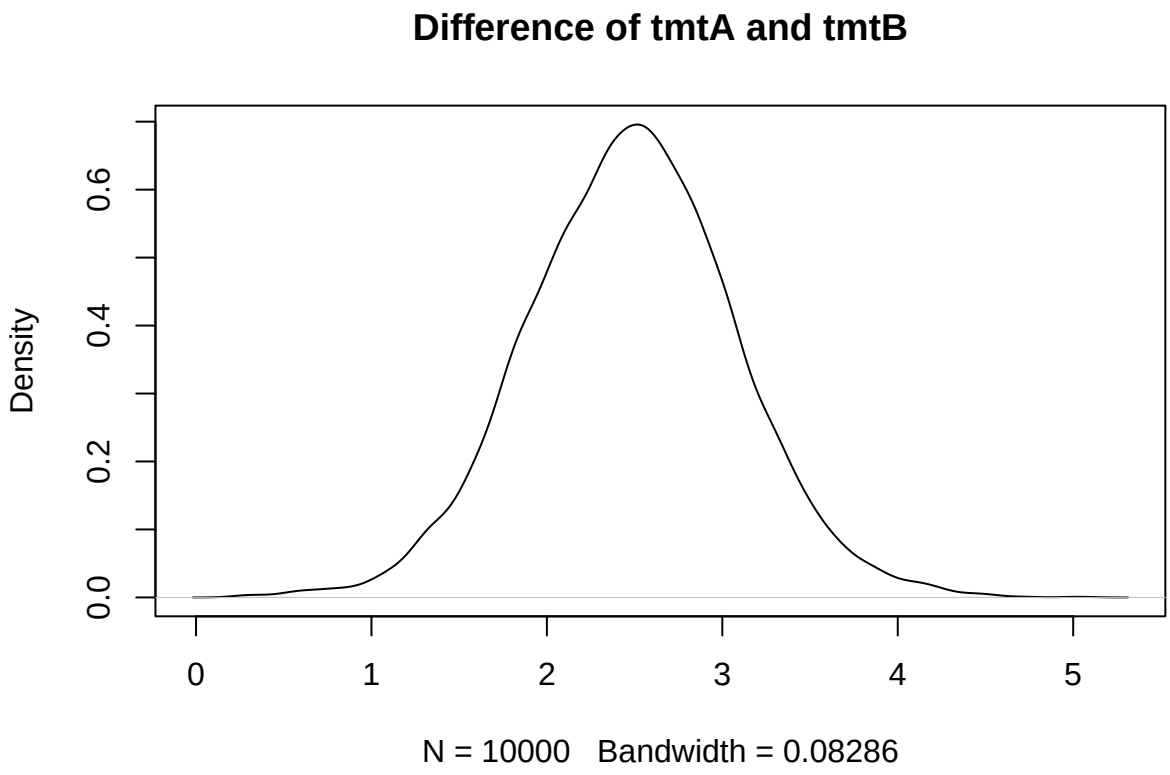
This is the effective number of parameters, based on the deviance.

12. How is the DIC for the model computed.

Deviance Information criteria is calculated based on the log likelihood for each iteration. It's $0.5\text{var}(-2\log(\text{LL})) + \text{mean}(-2 * \log(\text{LL}))$.

13. Say the first four treatment means represent 4 levels of a treatment. We'll call this treatment A. Say treatment means five through eight represent 4 levels of another treatment that we will call treatment B. Plot the posterior density for the combination of the parameters that you would use to test the assertion that treatment B yields higher responses that treatment A.

```
plot(density(tmtdiff), main = 'Difference of tmtA and tmtB')
```



14. Would you conclude treatment B yields higher responses than treatment A? Why?

Yes, because the posterior probability of Treatment B yielding higher responses than A is approximately 1

For the next set of problems use the data file 'exam1-3.dat'. Use a normal likelihood as you would with a standard frequentist multiple regression. For these data we are attempting to predict Defective using Temperature, Density, and Rate. Use the square root of Defective as the dependent variable, and please standardize all variables (ie, subtract the mean and divide by the standard deviation, including the square root of Defective) prior to running any model.

15. For the model include only main effects for Temperature, Density, and Rate. Write code to solve the problem in Stan. Produce posterior chains for the parameters you are estimating. Use normal priors for the β parameters, N(0,sd=10), and use a gamma with shape of 1.1 and a scale of 1 for the variance. Also, set the seed value as 4321 so all output will be identical. Use 5500 for the number of iterations, 500 for the burn in iterations, use 5 chains, and don't thin. Make sure the code has enough in it so that WAIC may be computed. The answer to this question is your Stan code.

```
code <- "
data {
  int N;
  real y[N];
  real Temperature[N];
  real Density[N];
  real Rate[N];
}
parameters {
  real <lower=0> sigma;
  real b0;
  real bTemp;
  real bDens;
  real bRate;
}
transformed parameters{
  real mu[N];
  real <lower=0> sig2;
  sig2 = sigma^2;
  for (i in 1:N){
    mu[i] = b0 + bTemp*Temperature[i] + bDens*Density[i] + bRate*Rate[i];
  }
}
model {
  for (i in 1:N){
    y[i] ~ normal(mu[i],sigma);
  }
  b0 ~ normal(0, 10);
  bTemp ~ normal(0, 10);
  bDens ~ normal(0, 10);
  bRate ~ norma(0, 10);
  sig2 ~ gamma(1.1, 1);

}

"

data_list <- list(
  N = dim(standat)[1],
  y = standat$DefectiveSqrt,
  Temperature = standat$Temperature,
  Density = standat$Density,
  Rate = standat$Rate
)

model2 <- cmdstan_model('midtermstan.stan')

fit <- model2$sample(
  data = data_list,
  seed = 4321,
  chains = 5,
  parallel_chains = 4,
  iter_warmup = 500,
```

```
  iter_sampling = 5500,

)
```

```
## Running MCMC with 5 chains, at most 4 in parallel...
##
## Chain 4 Iteration:    1 / 6000 [  0%]  (Warmup)
## Chain 4 Iteration:  100 / 6000 [  1%]  (Warmup)

## Chain 4 Informational Message: The current Metropolis proposal is about to be rejected because of the

## Chain 4 Exception: normal_lpdf: Scale parameter is 0, but must be positive! (in 'C:/Users/fyref/AppDa

## Chain 4 If this warning occurs sporadically, such as for highly constrained variable types like covar

## Chain 4 but if this warning occurs often then your model may be either severely ill-conditioned or mi

## Chain 4

## Chain 2 Informational Message: The current Metropolis proposal is about to be rejected because of the

## Chain 2 Exception: normal_lpdf: Scale parameter is 0, but must be positive! (in 'C:/Users/fyref/AppDa

## Chain 2 If this warning occurs sporadically, such as for highly constrained variable types like covar

## Chain 2 but if this warning occurs often then your model may be either severely ill-conditioned or mi

## Chain 2

## Chain 3 Iteration:    1 / 6000 [  0%]  (Warmup)
## Chain 3 Iteration:  100 / 6000 [  1%]  (Warmup)
## Chain 3 Iteration:  200 / 6000 [  3%]  (Warmup)
## Chain 3 Iteration:  300 / 6000 [  5%]  (Warmup)
## Chain 3 Iteration:  400 / 6000 [  6%]  (Warmup)
## Chain 3 Iteration:  500 / 6000 [  8%]  (Warmup)
## Chain 3 Iteration:  501 / 6000 [  8%]  (Sampling)
## Chain 3 Iteration:  600 / 6000 [ 10%]  (Sampling)
## Chain 4 Iteration:  200 / 6000 [  3%]  (Warmup)
## Chain 4 Iteration:  300 / 6000 [  5%]  (Warmup)
## Chain 4 Iteration:  400 / 6000 [  6%]  (Warmup)
## Chain 4 Iteration:  500 / 6000 [  8%]  (Warmup)
## Chain 4 Iteration:  501 / 6000 [  8%]  (Sampling)
## Chain 4 Iteration:  600 / 6000 [ 10%]  (Sampling)
## Chain 4 Iteration:  700 / 6000 [ 11%]  (Sampling)
## Chain 4 Iteration:  800 / 6000 [ 13%]  (Sampling)
## Chain 4 Iteration:  900 / 6000 [ 15%]  (Sampling)
## Chain 1 Iteration:    1 / 6000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 6000 [  1%]  (Warmup)
## Chain 1 Iteration:  200 / 6000 [  3%]  (Warmup)
## Chain 1 Iteration:  300 / 6000 [  5%]  (Warmup)
```

```
## Chain 1 Iteration:  400 / 6000 [  6%]  (Warmup)
## Chain 1 Iteration:  500 / 6000 [  8%]  (Warmup)
## Chain 1 Iteration:  501 / 6000 [  8%]  (Sampling)
## Chain 1 Iteration:  600 / 6000 [ 10%]  (Sampling)
## Chain 1 Iteration:  700 / 6000 [ 11%]  (Sampling)
## Chain 1 Iteration:  800 / 6000 [ 13%]  (Sampling)
## Chain 1 Iteration:  900 / 6000 [ 15%]  (Sampling)
## Chain 1 Iteration: 1000 / 6000 [ 16%]  (Sampling)
## Chain 1 Iteration: 1100 / 6000 [ 18%]  (Sampling)
## Chain 2 Iteration:    1 / 6000 [  0%]  (Warmup)
## Chain 2 Iteration:  100 / 6000 [  1%]  (Warmup)
## Chain 2 Iteration:  200 / 6000 [  3%]  (Warmup)
## Chain 2 Iteration:  300 / 6000 [  5%]  (Warmup)
## Chain 2 Iteration:  400 / 6000 [  6%]  (Warmup)
## Chain 2 Iteration:  500 / 6000 [  8%]  (Warmup)
## Chain 2 Iteration:  501 / 6000 [  8%]  (Sampling)
## Chain 2 Iteration:  600 / 6000 [ 10%]  (Sampling)
## Chain 2 Iteration:  700 / 6000 [ 11%]  (Sampling)
## Chain 2 Iteration:  800 / 6000 [ 13%]  (Sampling)
## Chain 2 Iteration:  900 / 6000 [ 15%]  (Sampling)
## Chain 2 Iteration: 1000 / 6000 [ 16%]  (Sampling)
## Chain 2 Iteration: 1100 / 6000 [ 18%]  (Sampling)
## Chain 3 Iteration:  700 / 6000 [ 11%]  (Sampling)
## Chain 3 Iteration:  800 / 6000 [ 13%]  (Sampling)
## Chain 3 Iteration:  900 / 6000 [ 15%]  (Sampling)
## Chain 3 Iteration: 1000 / 6000 [ 16%]  (Sampling)
## Chain 3 Iteration: 1100 / 6000 [ 18%]  (Sampling)
## Chain 4 Iteration: 1000 / 6000 [ 16%]  (Sampling)
## Chain 4 Iteration: 1100 / 6000 [ 18%]  (Sampling)
## Chain 4 Iteration: 1200 / 6000 [ 20%]  (Sampling)
## Chain 4 Iteration: 1300 / 6000 [ 21%]  (Sampling)
## Chain 4 Iteration: 1400 / 6000 [ 23%]  (Sampling)
## Chain 1 Iteration: 1200 / 6000 [ 20%]  (Sampling)
## Chain 1 Iteration: 1300 / 6000 [ 21%]  (Sampling)
## Chain 1 Iteration: 1400 / 6000 [ 23%]  (Sampling)
## Chain 1 Iteration: 1500 / 6000 [ 25%]  (Sampling)
## Chain 1 Iteration: 1600 / 6000 [ 26%]  (Sampling)
## Chain 1 Iteration: 1700 / 6000 [ 28%]  (Sampling)
## Chain 1 Iteration: 1800 / 6000 [ 30%]  (Sampling)
## Chain 2 Iteration: 1200 / 6000 [ 20%]  (Sampling)
## Chain 2 Iteration: 1300 / 6000 [ 21%]  (Sampling)
## Chain 2 Iteration: 1400 / 6000 [ 23%]  (Sampling)
## Chain 2 Iteration: 1500 / 6000 [ 25%]  (Sampling)
## Chain 2 Iteration: 1600 / 6000 [ 26%]  (Sampling)
## Chain 2 Iteration: 1700 / 6000 [ 28%]  (Sampling)
## Chain 3 Iteration: 1200 / 6000 [ 20%]  (Sampling)
## Chain 3 Iteration: 1300 / 6000 [ 21%]  (Sampling)
## Chain 3 Iteration: 1400 / 6000 [ 23%]  (Sampling)
## Chain 3 Iteration: 1500 / 6000 [ 25%]  (Sampling)
## Chain 3 Iteration: 1600 / 6000 [ 26%]  (Sampling)
## Chain 3 Iteration: 1700 / 6000 [ 28%]  (Sampling)
## Chain 4 Iteration: 1500 / 6000 [ 25%]  (Sampling)
## Chain 4 Iteration: 1600 / 6000 [ 26%]  (Sampling)
## Chain 4 Iteration: 1700 / 6000 [ 28%]  (Sampling)
```

```
## Chain 4 Iteration: 1800 / 6000 [ 30%]  (Sampling)
## Chain 4 Iteration: 1900 / 6000 [ 31%]  (Sampling)
## Chain 4 Iteration: 2000 / 6000 [ 33%]  (Sampling)
## Chain 4 Iteration: 2100 / 6000 [ 35%]  (Sampling)
## Chain 1 Iteration: 1900 / 6000 [ 31%]  (Sampling)
## Chain 1 Iteration: 2000 / 6000 [ 33%]  (Sampling)
## Chain 1 Iteration: 2100 / 6000 [ 35%]  (Sampling)
## Chain 2 Iteration: 1800 / 6000 [ 30%]  (Sampling)
## Chain 2 Iteration: 1900 / 6000 [ 31%]  (Sampling)
## Chain 2 Iteration: 2000 / 6000 [ 33%]  (Sampling)
## Chain 2 Iteration: 2100 / 6000 [ 35%]  (Sampling)
## Chain 3 Iteration: 1800 / 6000 [ 30%]  (Sampling)
## Chain 3 Iteration: 1900 / 6000 [ 31%]  (Sampling)
## Chain 3 Iteration: 2000 / 6000 [ 33%]  (Sampling)
## Chain 3 Iteration: 2100 / 6000 [ 35%]  (Sampling)
## Chain 4 Iteration: 2200 / 6000 [ 36%]  (Sampling)
## Chain 4 Iteration: 2300 / 6000 [ 38%]  (Sampling)
## Chain 4 Iteration: 2400 / 6000 [ 40%]  (Sampling)
## Chain 4 Iteration: 2500 / 6000 [ 41%]  (Sampling)
## Chain 1 Iteration: 2200 / 6000 [ 36%]  (Sampling)
## Chain 1 Iteration: 2300 / 6000 [ 38%]  (Sampling)
## Chain 1 Iteration: 2400 / 6000 [ 40%]  (Sampling)
## Chain 1 Iteration: 2500 / 6000 [ 41%]  (Sampling)
## Chain 2 Iteration: 2200 / 6000 [ 36%]  (Sampling)
## Chain 2 Iteration: 2300 / 6000 [ 38%]  (Sampling)
## Chain 2 Iteration: 2400 / 6000 [ 40%]  (Sampling)
## Chain 3 Iteration: 2200 / 6000 [ 36%]  (Sampling)
## Chain 3 Iteration: 2300 / 6000 [ 38%]  (Sampling)
## Chain 3 Iteration: 2400 / 6000 [ 40%]  (Sampling)
## Chain 4 Iteration: 2600 / 6000 [ 43%]  (Sampling)
## Chain 4 Iteration: 2700 / 6000 [ 45%]  (Sampling)
## Chain 4 Iteration: 2800 / 6000 [ 46%]  (Sampling)
## Chain 1 Iteration: 2600 / 6000 [ 43%]  (Sampling)
## Chain 1 Iteration: 2700 / 6000 [ 45%]  (Sampling)
## Chain 1 Iteration: 2800 / 6000 [ 46%]  (Sampling)
## Chain 2 Iteration: 2500 / 6000 [ 41%]  (Sampling)
## Chain 2 Iteration: 2600 / 6000 [ 43%]  (Sampling)
## Chain 3 Iteration: 2500 / 6000 [ 41%]  (Sampling)
## Chain 3 Iteration: 2600 / 6000 [ 43%]  (Sampling)
## Chain 4 Iteration: 2900 / 6000 [ 48%]  (Sampling)
## Chain 4 Iteration: 3000 / 6000 [ 50%]  (Sampling)
## Chain 4 Iteration: 3100 / 6000 [ 51%]  (Sampling)
## Chain 1 Iteration: 2900 / 6000 [ 48%]  (Sampling)
## Chain 1 Iteration: 3000 / 6000 [ 50%]  (Sampling)
## Chain 1 Iteration: 3100 / 6000 [ 51%]  (Sampling)
## Chain 2 Iteration: 2700 / 6000 [ 45%]  (Sampling)
## Chain 2 Iteration: 2800 / 6000 [ 46%]  (Sampling)
## Chain 2 Iteration: 2900 / 6000 [ 48%]  (Sampling)
## Chain 3 Iteration: 2700 / 6000 [ 45%]  (Sampling)
## Chain 3 Iteration: 2800 / 6000 [ 46%]  (Sampling)
## Chain 3 Iteration: 2900 / 6000 [ 48%]  (Sampling)
## Chain 4 Iteration: 3200 / 6000 [ 53%]  (Sampling)
## Chain 4 Iteration: 3300 / 6000 [ 55%]  (Sampling)
## Chain 4 Iteration: 3400 / 6000 [ 56%]  (Sampling)
```

```
## Chain 1 Iteration: 3200 / 6000 [ 53%]  (Sampling)
## Chain 1 Iteration: 3300 / 6000 [ 55%]  (Sampling)
## Chain 1 Iteration: 3400 / 6000 [ 56%]  (Sampling)
## Chain 2 Iteration: 3000 / 6000 [ 50%]  (Sampling)
## Chain 2 Iteration: 3100 / 6000 [ 51%]  (Sampling)
## Chain 2 Iteration: 3200 / 6000 [ 53%]  (Sampling)
## Chain 3 Iteration: 3000 / 6000 [ 50%]  (Sampling)
## Chain 3 Iteration: 3100 / 6000 [ 51%]  (Sampling)
## Chain 3 Iteration: 3200 / 6000 [ 53%]  (Sampling)
## Chain 4 Iteration: 3500 / 6000 [ 58%]  (Sampling)
## Chain 4 Iteration: 3600 / 6000 [ 60%]  (Sampling)
## Chain 4 Iteration: 3700 / 6000 [ 61%]  (Sampling)
## Chain 1 Iteration: 3500 / 6000 [ 58%]  (Sampling)
## Chain 1 Iteration: 3600 / 6000 [ 60%]  (Sampling)
## Chain 1 Iteration: 3700 / 6000 [ 61%]  (Sampling)
## Chain 2 Iteration: 3300 / 6000 [ 55%]  (Sampling)
## Chain 2 Iteration: 3400 / 6000 [ 56%]  (Sampling)
## Chain 2 Iteration: 3500 / 6000 [ 58%]  (Sampling)
## Chain 3 Iteration: 3300 / 6000 [ 55%]  (Sampling)
## Chain 3 Iteration: 3400 / 6000 [ 56%]  (Sampling)
## Chain 3 Iteration: 3500 / 6000 [ 58%]  (Sampling)
## Chain 4 Iteration: 3800 / 6000 [ 63%]  (Sampling)
## Chain 4 Iteration: 3900 / 6000 [ 65%]  (Sampling)
## Chain 4 Iteration: 4000 / 6000 [ 66%]  (Sampling)
## Chain 1 Iteration: 3800 / 6000 [ 63%]  (Sampling)
## Chain 1 Iteration: 3900 / 6000 [ 65%]  (Sampling)
## Chain 1 Iteration: 4000 / 6000 [ 66%]  (Sampling)
## Chain 1 Iteration: 4100 / 6000 [ 68%]  (Sampling)
## Chain 2 Iteration: 3600 / 6000 [ 60%]  (Sampling)
## Chain 2 Iteration: 3700 / 6000 [ 61%]  (Sampling)
## Chain 2 Iteration: 3800 / 6000 [ 63%]  (Sampling)
## Chain 2 Iteration: 3900 / 6000 [ 65%]  (Sampling)
## Chain 3 Iteration: 3600 / 6000 [ 60%]  (Sampling)
## Chain 3 Iteration: 3700 / 6000 [ 61%]  (Sampling)
## Chain 3 Iteration: 3800 / 6000 [ 63%]  (Sampling)
## Chain 4 Iteration: 4100 / 6000 [ 68%]  (Sampling)
## Chain 4 Iteration: 4200 / 6000 [ 70%]  (Sampling)
## Chain 4 Iteration: 4300 / 6000 [ 71%]  (Sampling)
## Chain 4 Iteration: 4400 / 6000 [ 73%]  (Sampling)
## Chain 1 Iteration: 4200 / 6000 [ 70%]  (Sampling)
## Chain 1 Iteration: 4300 / 6000 [ 71%]  (Sampling)
## Chain 1 Iteration: 4400 / 6000 [ 73%]  (Sampling)
## Chain 2 Iteration: 4000 / 6000 [ 66%]  (Sampling)
## Chain 2 Iteration: 4100 / 6000 [ 68%]  (Sampling)
## Chain 3 Iteration: 3900 / 6000 [ 65%]  (Sampling)
## Chain 3 Iteration: 4000 / 6000 [ 66%]  (Sampling)
## Chain 3 Iteration: 4100 / 6000 [ 68%]  (Sampling)
## Chain 4 Iteration: 4500 / 6000 [ 75%]  (Sampling)
## Chain 4 Iteration: 4600 / 6000 [ 76%]  (Sampling)
## Chain 4 Iteration: 4700 / 6000 [ 78%]  (Sampling)
## Chain 1 Iteration: 4500 / 6000 [ 75%]  (Sampling)
## Chain 1 Iteration: 4600 / 6000 [ 76%]  (Sampling)
## Chain 2 Iteration: 4200 / 6000 [ 70%]  (Sampling)
## Chain 2 Iteration: 4300 / 6000 [ 71%]  (Sampling)
```

```
## Chain 2 Iteration: 4400 / 6000 [ 73%]  (Sampling)
## Chain 3 Iteration: 4200 / 6000 [ 70%]  (Sampling)
## Chain 3 Iteration: 4300 / 6000 [ 71%]  (Sampling)
## Chain 4 Iteration: 4800 / 6000 [ 80%]  (Sampling)
## Chain 4 Iteration: 4900 / 6000 [ 81%]  (Sampling)
## Chain 4 Iteration: 5000 / 6000 [ 83%]  (Sampling)
## Chain 1 Iteration: 4700 / 6000 [ 78%]  (Sampling)
## Chain 1 Iteration: 4800 / 6000 [ 80%]  (Sampling)
## Chain 1 Iteration: 4900 / 6000 [ 81%]  (Sampling)
## Chain 2 Iteration: 4500 / 6000 [ 75%]  (Sampling)
## Chain 2 Iteration: 4600 / 6000 [ 76%]  (Sampling)
## Chain 2 Iteration: 4700 / 6000 [ 78%]  (Sampling)
## Chain 3 Iteration: 4400 / 6000 [ 73%]  (Sampling)
## Chain 3 Iteration: 4500 / 6000 [ 75%]  (Sampling)
## Chain 3 Iteration: 4600 / 6000 [ 76%]  (Sampling)
## Chain 4 Iteration: 5100 / 6000 [ 85%]  (Sampling)
## Chain 4 Iteration: 5200 / 6000 [ 86%]  (Sampling)
## Chain 1 Iteration: 5000 / 6000 [ 83%]  (Sampling)
## Chain 1 Iteration: 5100 / 6000 [ 85%]  (Sampling)
## Chain 1 Iteration: 5200 / 6000 [ 86%]  (Sampling)
## Chain 2 Iteration: 4800 / 6000 [ 80%]  (Sampling)
## Chain 2 Iteration: 4900 / 6000 [ 81%]  (Sampling)
## Chain 2 Iteration: 5000 / 6000 [ 83%]  (Sampling)
## Chain 3 Iteration: 4700 / 6000 [ 78%]  (Sampling)
## Chain 3 Iteration: 4800 / 6000 [ 80%]  (Sampling)
## Chain 3 Iteration: 4900 / 6000 [ 81%]  (Sampling)
## Chain 4 Iteration: 5300 / 6000 [ 88%]  (Sampling)
## Chain 4 Iteration: 5400 / 6000 [ 90%]  (Sampling)
## Chain 4 Iteration: 5500 / 6000 [ 91%]  (Sampling)
## Chain 4 Iteration: 5600 / 6000 [ 93%]  (Sampling)
## Chain 1 Iteration: 5300 / 6000 [ 88%]  (Sampling)
## Chain 1 Iteration: 5400 / 6000 [ 90%]  (Sampling)
## Chain 1 Iteration: 5500 / 6000 [ 91%]  (Sampling)
## Chain 1 Iteration: 5600 / 6000 [ 93%]  (Sampling)
## Chain 2 Iteration: 5100 / 6000 [ 85%]  (Sampling)
## Chain 2 Iteration: 5200 / 6000 [ 86%]  (Sampling)
## Chain 2 Iteration: 5300 / 6000 [ 88%]  (Sampling)
## Chain 3 Iteration: 5000 / 6000 [ 83%]  (Sampling)
## Chain 3 Iteration: 5100 / 6000 [ 85%]  (Sampling)
## Chain 3 Iteration: 5200 / 6000 [ 86%]  (Sampling)
## Chain 4 Iteration: 5700 / 6000 [ 95%]  (Sampling)
## Chain 4 Iteration: 5800 / 6000 [ 96%]  (Sampling)
## Chain 4 Iteration: 5900 / 6000 [ 98%]  (Sampling)
## Chain 4 Iteration: 6000 / 6000 [100%]  (Sampling)
## Chain 4 finished in 2.5 seconds.
## Chain 1 Iteration: 5700 / 6000 [ 95%]  (Sampling)
## Chain 1 Iteration: 5800 / 6000 [ 96%]  (Sampling)
## Chain 1 Iteration: 5900 / 6000 [ 98%]  (Sampling)
## Chain 2 Iteration: 5400 / 6000 [ 90%]  (Sampling)
## Chain 2 Iteration: 5500 / 6000 [ 91%]  (Sampling)
## Chain 2 Iteration: 5600 / 6000 [ 93%]  (Sampling)
## Chain 3 Iteration: 5300 / 6000 [ 88%]  (Sampling)
## Chain 3 Iteration: 5400 / 6000 [ 90%]  (Sampling)
## Chain 3 Iteration: 5500 / 6000 [ 91%]  (Sampling)
```

```
## Chain 3 Iteration: 5600 / 6000 [ 93%]  (Sampling)
## Chain 1 Iteration: 6000 / 6000 [100%]  (Sampling)
## Chain 1 finished in 2.6 seconds.
## Chain 2 Iteration: 5700 / 6000 [ 95%]  (Sampling)
## Chain 2 Iteration: 5800 / 6000 [ 96%]  (Sampling)
## Chain 2 Iteration: 5900 / 6000 [ 98%]  (Sampling)
## Chain 2 Iteration: 6000 / 6000 [100%]  (Sampling)
## Chain 3 Iteration: 5700 / 6000 [ 95%]  (Sampling)
## Chain 3 Iteration: 5800 / 6000 [ 96%]  (Sampling)
## Chain 3 Iteration: 5900 / 6000 [ 98%]  (Sampling)
## Chain 2 finished in 2.7 seconds.
## Chain 3 Iteration: 6000 / 6000 [100%]  (Sampling)
## Chain 5 Iteration:    1 / 6000 [  0%]  (Warmup)
## Chain 5 Iteration:  100 / 6000 [  1%]  (Warmup)
## Chain 5 Iteration:  200 / 6000 [  3%]  (Warmup)
## Chain 5 Iteration:  300 / 6000 [  5%]  (Warmup)
## Chain 5 Iteration:  400 / 6000 [  6%]  (Warmup)
## Chain 5 Iteration:  500 / 6000 [  8%]  (Warmup)
## Chain 5 Iteration:  501 / 6000 [  8%]  (Sampling)
## Chain 5 Iteration:  600 / 6000 [ 10%]  (Sampling)
## Chain 5 Iteration:  700 / 6000 [ 11%]  (Sampling)
## Chain 3 finished in 2.7 seconds.
## Chain 5 Iteration:  800 / 6000 [ 13%]  (Sampling)
## Chain 5 Iteration:  900 / 6000 [ 15%]  (Sampling)
## Chain 5 Iteration: 1000 / 6000 [ 16%]  (Sampling)
## Chain 5 Iteration: 1100 / 6000 [ 18%]  (Sampling)
## Chain 5 Iteration: 1200 / 6000 [ 20%]  (Sampling)
## Chain 5 Iteration: 1300 / 6000 [ 21%]  (Sampling)
## Chain 5 Iteration: 1400 / 6000 [ 23%]  (Sampling)
## Chain 5 Iteration: 1500 / 6000 [ 25%]  (Sampling)
## Chain 5 Iteration: 1600 / 6000 [ 26%]  (Sampling)
## Chain 5 Iteration: 1700 / 6000 [ 28%]  (Sampling)
## Chain 5 Iteration: 1800 / 6000 [ 30%]  (Sampling)
## Chain 5 Iteration: 1900 / 6000 [ 31%]  (Sampling)
## Chain 5 Iteration: 2000 / 6000 [ 33%]  (Sampling)
## Chain 5 Iteration: 2100 / 6000 [ 35%]  (Sampling)
## Chain 5 Iteration: 2200 / 6000 [ 36%]  (Sampling)
## Chain 5 Iteration: 2300 / 6000 [ 38%]  (Sampling)
## Chain 5 Iteration: 2400 / 6000 [ 40%]  (Sampling)
## Chain 5 Iteration: 2500 / 6000 [ 41%]  (Sampling)
## Chain 5 Iteration: 2600 / 6000 [ 43%]  (Sampling)
## Chain 5 Iteration: 2700 / 6000 [ 45%]  (Sampling)
## Chain 5 Iteration: 2800 / 6000 [ 46%]  (Sampling)
## Chain 5 Iteration: 2900 / 6000 [ 48%]  (Sampling)
## Chain 5 Iteration: 3000 / 6000 [ 50%]  (Sampling)
## Chain 5 Iteration: 3100 / 6000 [ 51%]  (Sampling)
## Chain 5 Iteration: 3200 / 6000 [ 53%]  (Sampling)
## Chain 5 Iteration: 3300 / 6000 [ 55%]  (Sampling)
## Chain 5 Iteration: 3400 / 6000 [ 56%]  (Sampling)
## Chain 5 Iteration: 3500 / 6000 [ 58%]  (Sampling)
## Chain 5 Iteration: 3600 / 6000 [ 60%]  (Sampling)
## Chain 5 Iteration: 3700 / 6000 [ 61%]  (Sampling)
## Chain 5 Iteration: 3800 / 6000 [ 63%]  (Sampling)
## Chain 5 Iteration: 3900 / 6000 [ 65%]  (Sampling)
```

```
## Chain 5 Iteration: 4000 / 6000 [ 66%]  (Sampling)
## Chain 5 Iteration: 4100 / 6000 [ 68%]  (Sampling)
## Chain 5 Iteration: 4200 / 6000 [ 70%]  (Sampling)
## Chain 5 Iteration: 4300 / 6000 [ 71%]  (Sampling)
## Chain 5 Iteration: 4400 / 6000 [ 73%]  (Sampling)
## Chain 5 Iteration: 4500 / 6000 [ 75%]  (Sampling)
## Chain 5 Iteration: 4600 / 6000 [ 76%]  (Sampling)
## Chain 5 Iteration: 4700 / 6000 [ 78%]  (Sampling)
## Chain 5 Iteration: 4800 / 6000 [ 80%]  (Sampling)
## Chain 5 Iteration: 4900 / 6000 [ 81%]  (Sampling)
## Chain 5 Iteration: 5000 / 6000 [ 83%]  (Sampling)
## Chain 5 Iteration: 5100 / 6000 [ 85%]  (Sampling)
## Chain 5 Iteration: 5200 / 6000 [ 86%]  (Sampling)
## Chain 5 Iteration: 5300 / 6000 [ 88%]  (Sampling)
## Chain 5 Iteration: 5400 / 6000 [ 90%]  (Sampling)
## Chain 5 Iteration: 5500 / 6000 [ 91%]  (Sampling)
## Chain 5 Iteration: 5600 / 6000 [ 93%]  (Sampling)
## Chain 5 Iteration: 5700 / 6000 [ 95%]  (Sampling)
## Chain 5 Iteration: 5800 / 6000 [ 96%]  (Sampling)
## Chain 5 Iteration: 5900 / 6000 [ 98%]  (Sampling)
## Chain 5 Iteration: 6000 / 6000 [100%]  (Sampling)
## Chain 5 finished in 1.5 seconds.
##
## All 5 chains finished successfully.
## Mean chain execution time: 2.4 seconds.
## Total execution time: 4.6 seconds.
```

I chose to run the stan code using CmdStanR, which precompiles the Stan program into C++ as an executable, has the latest features of stan, and runs much quicker than RStan and crashes far less frequently.

Execution time on my computer was about 7 seconds.

16. Verify that the chains you have produced have converged appropriately by examining (and reproducing) the effective sample size using coda.

```
library(rstan)
```

```
## Loading required package: StanHeaders

## Loading required package: ggplot2

## rstan (Version 2.21.3, GitRev: 2e1f913d3ca3)

## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)

## Do not specify '-march=native' in 'LOCAL_CPPFLAGS' or a Makevars file

##
## Attaching package: 'rstan'
```

```
## The following object is masked from 'package:R2jags':
##
##     traceplot

## The following object is masked from 'package:coda':
##
##     traceplot
```

```
stanfit <- rstan::read_stan_csv(fit$output_files())

samps <- extract(stanfit)
names(samps)
```

```
## [1] "sigma"   "b0"      "bTemp"   "bDens"   "bRate"   "mu"      "sig2"
## [8] "log_lik" "lp__"
```

```
chains <- cbind(samps[[1]], samps[[2]], samps[[3]], samps[[4]], samps[[5]])
colnames(chains) <- c("sigma","b0","bTemp","bDens","bRate")
sims <- as.mcmc(chains)
effectiveSize(sims)
```

```
##    sigma       b0    bTemp    bDens    bRate
## 27500.00 27500.00 27500.00 26436.27 27500.00
```

17. Verify that the chains you have produced have converged appropriately by examining (and reproducing) the Raftery-Lewis diagnostics.

```
raftery.diag(sims)
```

```
##
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.005
## Probability (s) = 0.95
##
##        Burn-in  Total Lower bound  Dependence
##        (M)      (N)   (Nmin)       factor (I)
##  sigma 2        3816  3746         1.020
##  b0    2        3737  3746         0.998
##  bTemp 2        3850  3746         1.030
##  bDens 2        3850  3746         1.030
##  bRate 1        3747  3746         1.000
```

18. Using coda, find 95% highest posterior density intervals for the parameters you have estimated.

```
HPDinterval(sims, prob = 0.95)
```

```
##            lower      upper
## sigma  0.1955470  0.3442060
## b0    -0.0936236  0.0989041
## bTemp  0.0514414  0.7755340
```

```
## bDens -0.8081890 -0.0640410
## bRate -0.0948177  0.4087830
## attr(,"Probability")
## [1] 0.95
```

19. What is the WAIC for the model?

```
LLa <- as.array(stanfit,pars='log_lik')
library(loo) # Leave One out cross validation for waic
```

```
## This is loo version 2.4.1

## - Online documentation and vignettes at mc-stan.org/loo

## - As of v2.0.0 loo defaults to 1 core but we recommend using as many as possible. Use the 'cores' ar

## - Windows 10 users: loo may be very slow if 'mc.cores' is set in your .Rprofile file (see https://gi

##
## Attaching package: 'loo'

## The following object is masked from 'package:rstan':
##
##     loo
```

```
waic(LLa)
```

```
## Warning:
## 4 (13.3%) p_waic estimates greater than 0.4. We recommend trying loo instead.

##
## Computed from 27500 by 30 log-likelihood matrix
##
##           Estimate  SE
## elpd_waic     -4.5 3.9
## p_waic         4.5 1.1
## waic           9.0 7.8
##
## 4 (13.3%) p_waic estimates greater than 0.4. We recommend trying loo instead.
```

20. What is the leave one out information criterion for the model?

```
fit$loo()
```

```
## Warning: Some Pareto k diagnostic values are slightly high. See help('pareto-k-diagnostic') for deta
```

```
##
## Computed from 27500 by 30 log-likelihood matrix
##
##          Estimate  SE
## elpd_loo     -4.7 3.9
## p_loo         4.7 1.2
## looic         9.3 7.9
## ------
## Monte Carlo SE of elpd_loo is 0.0.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     29    96.7%   3259
##  (0.5, 0.7]   (ok)        1     3.3%   4566
##    (0.7, 1]   (bad)       0     0.0%   <NA>
##    (1, Inf)   (very bad)  0     0.0%   <NA>
##
## All Pareto k estimates are ok (k < 0.7).
## See help('pareto-k-diagnostic') for details.
```