ESS 575: Bayes Theorem Lab

Team England

04 October, 2022

Team England:

- Caroline Blommel
- Carolyn Coyle
- Bryn Crosby
- George Woolsey

cblommel@mail.colostate.edu, carolynm@mail.colostate.edu, brcrosby@rams.colostate.edu, george.woolsey@colostate.edu

Setup

Download the R package SESYNCBayes ver. 0.6 to your computer.

Run:

```
install.packages("<pathtoSESYNCBayes>/Packages/SESYNCBayes_0.6.0.tar.gz", repos = NULL,
type = "source")
```

Aim

The purpose of this Primer is to teach the programming skills needed to approximate the marginal posterior distributions of parameters, latent variables, and derived quantities of interest using software implementing Markov chain Monte Carlo methods. Along the way, we will reinforce some of the ideas and principles that we have learned in lecture. The Primer is organized primarily as a tutorial and contains only a modicum of reference material. You will need to study the JAGS manual at some point to become a fully competent programmer.

Example

Consider a model predicting the per-capita rate of population growth (or the mass specific rate of nitrogen accumulation),

$$\frac{1}{N}\frac{dN}{dt} = r - \frac{r}{K}N, \tag{1}$$

which, of course, is a linear model with intercept r and slope $\frac{r}{K}$. Note that these quantities enjoy a sturdy biological interpretation in population ecology; r is the intrinsic rate of increase, $\frac{r}{K}$ is the strength of

the feedback from population size to population growth rate, and K is the carrying capacity, that is, the population size (o.k., o.k., the gm N per gm soil for the ecosystem scientists) at which $\frac{dN}{dt} = 0$. Presume we have some data consisting of observations of per capita rate of growth paired with observations of N. The vector $\mathbf{y}$ contains values for the rate and the vector $\mathbf{x}$ contains aligned data on N, i.e., $y_i = \frac{1}{N_i}\frac{dN_i}{dt}$, $x_i = N_i$. To keep things simple, we start out by assuming that the x_i are measured without error. A simple Bayesian model specifies the joint distribution of the parameters and data as:

$$\mu_i = r - \frac{rx_i}{K}, \tag{2}$$

$$\begin{aligned}
[r, K, \tau \mid \mathbf{y}] &\propto \prod_{i=1}^{n} [y_i \mid \mu_i, \tau]\,[r]\,[K]\,[\tau]\,, \\
[r, K, \tau \mid \mathbf{y}] &\propto \prod_{i=1}^{n} \text{normal}\,(y_i \mid \mu_i, \tau) \times \text{gamma}\,(K \mid .001, .001) \\
&\quad \text{gamma}\,(\tau \mid .001, .001)\,\text{gamma}\,(r \mid .001, .001)\,,
\end{aligned} \tag{3}$$

where the priors are vague distributions for quantities that must, by definition, be positive. Note that we have used the precision (τ) as a argument to the normal distribution rather than the variance $\left(\tau = \frac{1}{\sigma^2}\right)$ to keep things consistent with the code below, a requirement of the BUGS language. Now, we have full, abiding confidence that with a couple of hours worth of work, perhaps less, you could knock out a Gibbs sampler to estimate r, K, and τ. However, we are all for doing things nimbly in 15 minutes that might otherwise take a sweaty hour of hard labor.

```
## Logistic example for Primer
model{
  # priors
  K ~ dgamma(.001, .001) # dgamma(r,n)
  r ~ dgamma(.001, .001) # dgamma(r,n)
  tau ~ dgamma(.001, .001) # precision  # dgamma(r,n)
  sigma <- 1/sqrt(tau) # calculate sd from precision
  # likelihood
  for (i in 1:n){
    mu[i] <- r - r/K * x[i]
    y[i] ~ dnorm(mu[i], tau) # dnorm(mu,tau)
  }
}
```

Exercise 1: Factoring

There is no x in the posterior distribution in equation 4. What are assuming if x is absent? Draw the Bayesian network, or DAG, for this model. Use the chain rule to fully factor the joint distribution into sensible parts then simplify by assuming that r, K, and τ are independent.

Factoring the joint using the chain rule:

$$[r, K, \tau, y] = [y \mid r, K, \tau] \cdot [r \mid K, \tau] \cdot [K \mid \tau] \cdot [\tau]$$

Assuming r, K, τ are independent:

$$[r, K, \tau, y] = [y \mid r, K, \tau] \cdot [r] \cdot [K] \cdot [\tau]$$

There is no x because we are assuming it is measured without error.

Exercise 2: Can you improve these priors?

A recurring theme in this course will be to use priors that are informative whenever possible. The gamma priors in equation 4 include *the entire number line* > 0. Don't we know more about population biology than that? Let's, say for now that we are modeling the population dynamics of a large mammal. How might you go about making the priors on population parameters more informative?

The intrinsic rate of increase r can plausibly take on values between 0 and 1. The only requirement for a vague prior is that its "range of uncertainty should be clearly wider that the range of reasonable values of the parameter". Similarly, we could use experience and knowledge to put some reasonable bounds on K and even σ, which we can use to calculate τ as $\tau = \frac{1}{\sigma^2}$.

```
## Logistic example for Primer
  model{
  # priors
  K ~ dunif(0, 4000) # dunif(alpha = lower limit, beta = upper limit)
  r ~ dunif (0, 2) # dunif(alpha, beta)
  sigma ~ dunif(0, 2) # dunif(alpha, beta)
  tau <- 1/sigma^2
  # likelihood
  for(i in 1:n){
    mu[i] <- r - r/K * x[i]
    y[i] ~ dnorm(mu[i], tau) # dnorm(mu,tau)
  }
}
```

Exercise 3: Using `for` loops.

Write a code fragment to set vague normal priors for 5 regression coefficients – `dnorm(0, 0.000001)` – stored in the vector `b`.

```
b <- numeric(5)
for(i in 1:length(b)){
  b[i] ~ dnorm(0, 0.000001) # dnorm(mu,tau)
}
```

Stepping through a JAGS run

We will go through the R code step by step. We start by loading the package `SESYNCBayes` which has the data frame Logistic, which we then order by `PopulationSize`. Next, we specify the initial conditions for the MCMC chain in the statement `inits`. This is exactly the same thing as you did when you wrote MCMC code and assigned a guess to the first element in the chain. Initial conditions must be specified as as "list of lists", as you can see in the code. If you create a single list, rather than a list of lists, you will get an error message when you execute the `jags.model` statement and your code will not run. Second, this statement allows you to set up multiple chains, which are needed for some tests of convergence and to calculate DIC (more about these tasks later). For example, if you want three chains, you would use:

```
inits = list(
  list(K = 1500, r = .2, sigma = 1),
  list(K = 1000, r = .15, sigma = .1),
```

```
  list(K = 900, r = .3, sigma = .01)
)
```

Now it is really easy to see why we need the "list of lists" format – there is one list for each chain; but remember, you require the same structure for a single chain, that is, a list of lists.

Which variables in your JAGS code require initialization? All unknown quantities that appear on the left hand side of the conditioning in the posterior distribution require initial values. Think about it this way. When you were writing your own MCMC algorithm, every chain required a value as the first element in the vector holding the chain. That is what you are doing when you specify initial conditions here. You can get away without explicitly specifying initial values – JAGS will choose them for you if you don't specify them – however, we strongly urge you to provide explicit initial values, particularly when your priors are vague. This habit also forces you to think about what you are estimating.

The left hand side of the `=` corresponds to variable name for the data in the `JAGS` program and the right hand side of the `=` is what they are called in `R`.

```
rm(list = ls())
library(SESYNCBayes)
library(rjags)
# SESYNCBayes which has the data frame Logistic, which we then order by PopulationSize
Logistic = SESYNCBayes::Logistic[order(Logistic$PopulationSize),]
# specify the initial conditions for the MCMC chain
inits = list(
  list(K = 1500, r = .2, sigma = 1),
  list(K = 1000, r = .15, sigma = .1),
  list(K = 900, r = .3, sigma = .01)
)
# specify the data that will be used by your JAGS program
  #the execution of JAGS is about 5 times faster on double precision than on integers.
hey_data = list(
  n = nrow(SESYNCBayes::Logistic), # n is required in the JAGS program to index the for structure
  x = as.double(SESYNCBayes::Logistic$PopulationSize),
  y = as.double(SESYNCBayes::Logistic$GrowthRate)
)
# specify 3 scalars, n.adapt, n.update, and n.iter
# n.adapt = number of iterations that JAGS will use to choose the sampler
  # and to assure optimum mixing of the MCMC chain
n.adapt = 1000
# n.update = number of iterations that will be discarded to allow the chain to
#   converge before iterations are stored (aka, burn-in)
n.update = 10000
# n.iter = number of iterations that will be stored in the
  # final chain as samples from the posterior distribution
n.iter = 10000
#######################
# Call to JAGS
#######################
set.seed(1)
jm = rjags::jags.model(
  file = "LogisticJAGS.R"
  , data = hey_data
  , inits = inits
  , n.chains = length(inits)
```

```
  , n.adapt = n.adapt
)
stats::update(jm, n.iter = n.update)
zm = rjags::coda.samples(
  model = jm
  , variable.names = c("K", "r", "sigma", "tau")
  , n.iter = n.iter
  , n.thin = 1
)
```

Exercise 4: Coding the model.

Write R code (algorithm 3) to run the JAGS model (algorithm 2) and estimate the parameters, r, K, σ and τ. We suggest you insert the JAGS model into this R script using the sink command as shown in algorithm 4 because this model is small. You will find this a convenient way to keep all your code in the same R script. For larger models, you will be happier using a separate file for the JAGS code.

```
#######################################################################
# insert JAGS model code into an R script
#######################################################################
{ # Extra bracket needed only for R markdown files - see answers
  sink("LogisticJAGS.R") # This is the file name for the jags code
  cat("
  ## Logistic example for Primer
    model{
      # priors
      K ~ dunif(0, 4000) # dunif(alpha = lower limit, beta = upper limit)
      r ~ dunif (0, 2) # dunif(alpha, beta)
      sigma ~ dunif(0, 2) # dunif(alpha, beta)
      tau <- 1/sigma^2
      # likelihood
      for(i in 1:n){
        mu[i] <- r - r/K * x[i]
        y[i] ~ dnorm(mu[i], tau) # dnorm(mu,tau)
      }
    }
  ", fill = TRUE)
  sink()
}
#######################################################################
# implement model
#######################################################################
# SESYNCBayes which has the data frame Logistic, which we then order by PopulationSize
Logistic = SESYNCBayes::Logistic[order(Logistic$PopulationSize),]
# specify the initial conditions for the MCMC chain
inits = list(
  list(K = 1500, r = .2, sigma = 1),
  list(K = 1000, r = .15, sigma = .1),
  list(K = 900, r = .3, sigma = .01)
)
# specify the data that will be used by your JAGS program
  #the execution of JAGS is about 5 times faster on double precision than on integers.
```

```
hey_data = list(
  n = nrow(SESYNCBayes::Logistic), # n is required in the JAGS program to index the for structure
  x = as.double(SESYNCBayes::Logistic$PopulationSize),
  y = as.double(SESYNCBayes::Logistic$GrowthRate)
)
# specify 3 scalars, n.adapt, n.update, and n.iter
# n.adapt = number of iterations that JAGS will use to choose the sampler
  # and to assure optimum mixing of the MCMC chain
n.adapt = 1000
# n.update = number of iterations that will be discarded to allow the chain to
#   converge before iterations are stored (aka, burn-in)
n.update = 10000
# n.iter = number of iterations that will be stored in the
  # final chain as samples from the posterior distribution
n.iter = 10000
######################
# Call to JAGS
######################
set.seed(1)
jm = rjags::jags.model(
  file = "LogisticJAGS.R"
  , data = hey_data
  , inits = inits
  , n.chains = length(inits)
  , n.adapt = n.adapt
)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 50
##    Unobserved stochastic nodes: 3
##    Total graph size: 211
##
## Initializing model
```

```
stats::update(jm, n.iter = n.update)
# save the coda object (more precisely, an mcmc.list object) to R as "zm"
zm = rjags::coda.samples(
  model = jm
  , variable.names = c("K", "r", "sigma", "tau")
  , n.iter = n.iter
  , n.thin = 1
)
######################
# check output
######################
MCMCvis::MCMCsummary(zm)
```

```
##               mean           sd         2.5%          50%        97.5% Rhat
## K     1.237994e+03 6.372182e+01 1.130669e+03 1.232046e+03 1.380557e+03    1
## r     2.006627e-01 9.813994e-03 1.809543e-01 2.008046e-01 2.195696e-01    1
```

```
## sigma 2.867197e-02 3.037380e-03 2.351267e-02 2.840965e-02 3.529886e-02    1
## tau   1.256634e+03 2.596682e+02 8.025619e+02 1.238991e+03 1.808823e+03    1
##       n.eff
## K      6387
## r      7104
## sigma 14788
## tau   16463
```

```
# chain 1 first 6 iterations
zm[[1]][1:6,]
```

```
##             K         r      sigma       tau
## [1,] 1173.869 0.1959574 0.02894180 1193.8476
## [2,] 1189.095 0.2040177 0.02636004 1439.1562
## [3,] 1180.767 0.2021973 0.03181864  987.7267
## [4,] 1291.478 0.1897676 0.02584462 1497.1308
## [5,] 1311.942 0.1985844 0.03265458  937.8033
## [6,] 1367.690 0.1823589 0.03561935  788.1845
```

Exercise 5: Understanding coda objects.

1) Convert the coda object `zm`, into a data frame using `df = as.data.frame(rbind(zm[[1]], zm[[2]], zm[[3]]))` Note the double brackets, which effectively unlist each element of `zm`, allowing them to be combined. Another way to do this is `do.call(rbind,zm)`.
2) Look at the first six rows of the data frame.
3) Find the maximum value of σ.
4) Find the mean of r for the first 1000 iterations.
5) Find the mean of r after the first 1000 iterations.
6) Make two publication quality plots of the marginal posterior density of `K`, one as a smooth curve and the other as a histogram.
7) Compute the probability that $K > 1600$. Hint: what type of probability distribution would you use for this computation? Investigate the the dramatically useful R function `ecdf()`.
8) Compute the probability that $1000 < K < 1600$.
9) Compute the .025 and .975 quantiles of K. Hint–use the R `quantile()` function. This is an equal-tailed Bayesian credible interval on K.

1) Convert the coda object `zm`, into a data frame using `df = as.data.frame(rbind(zm[[1]], zm[[2]], zm[[3]]))` Note the double brackets, which effectively unlist each element of `zm`, allowing them to be combined. Another way to do this is `do.call(rbind,zm)`.

```
########################################
########################################
# only need to do this if want a column denoting the chain #
########################################
########################################
# function to transform to data frame and
  # store chain number and iteration as variable
df_fn <- function(x){
```

```
  as.data.frame(zm[[x]]) %>%
    dplyr::mutate(
      chain = x
      , iteration = dplyr::row_number()
    )
}
# pass zm to function
zm_df <- 1:length(zm) %>%
  purrr::map(df_fn) %>% # purrr::map returns a list of data frames in this case...
  dplyr::bind_rows() # ...which we bind together
```

2) Look at the first six rows of the data frame.

```
zm_df %>%
  dplyr::slice_head(n = 6)
```

```
##          K         r      sigma       tau chain iteration
## 1 1173.869 0.1959574 0.02894180 1193.8476     1         1
## 2 1189.095 0.2040177 0.02636004 1439.1562     1         2
## 3 1180.767 0.2021973 0.03181864  987.7267     1         3
## 4 1291.478 0.1897676 0.02584462 1497.1308     1         4
## 5 1311.942 0.1985844 0.03265458  937.8033     1         5
## 6 1367.690 0.1823589 0.03561935  788.1845     1         6
```

3) Find the maximum value of σ.

```
max(zm_df$sigma)
```

```
## [1] 0.04603398
```

4) Find the mean of r for the first 1000 iterations.

```
mean(zm_df$r[zm_df$iteration <= 1000])
```

```
## [1] 0.2002673
```

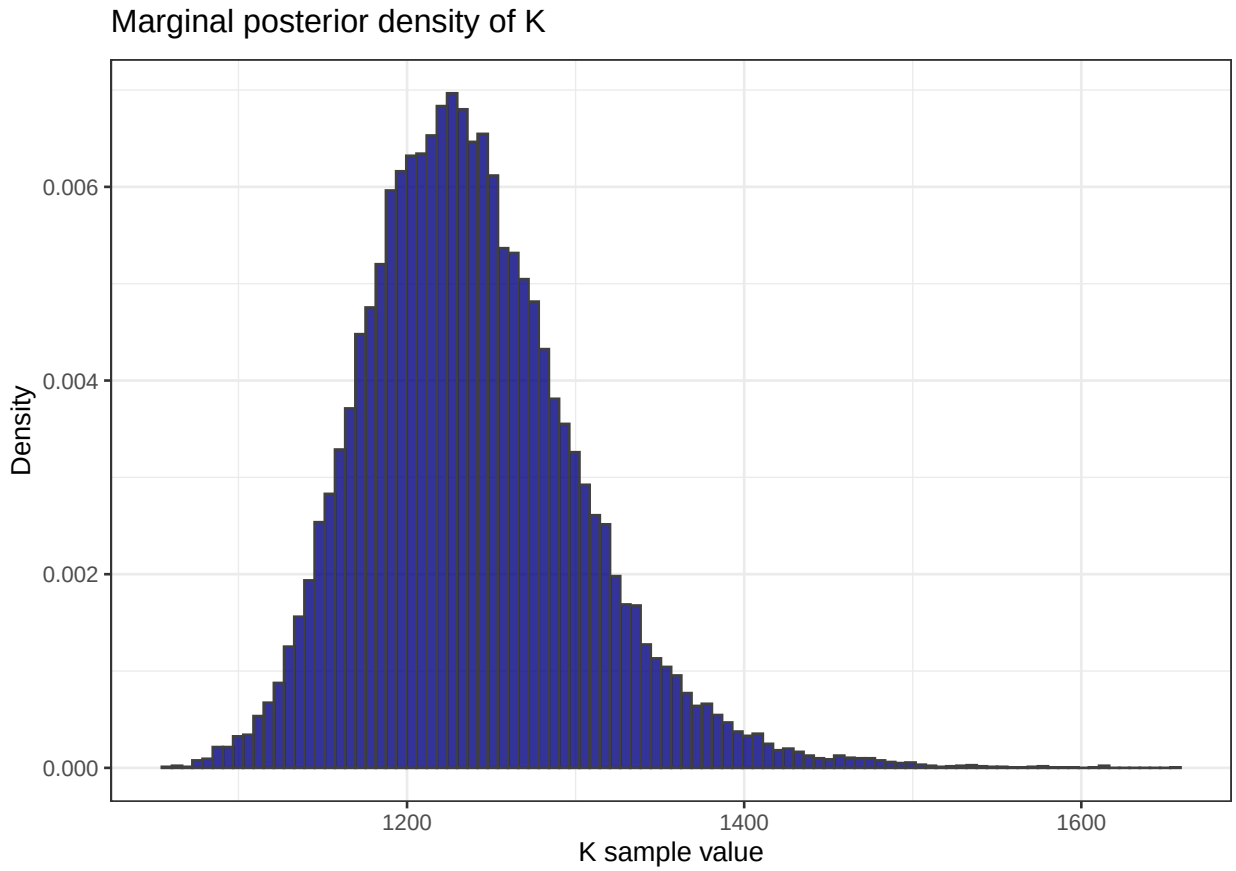
5) Find the mean of r after the first 1000 iterations.

```
mean(zm_df$r[zm_df$iteration > 1000])
```

```
## [1] 0.2007066
```

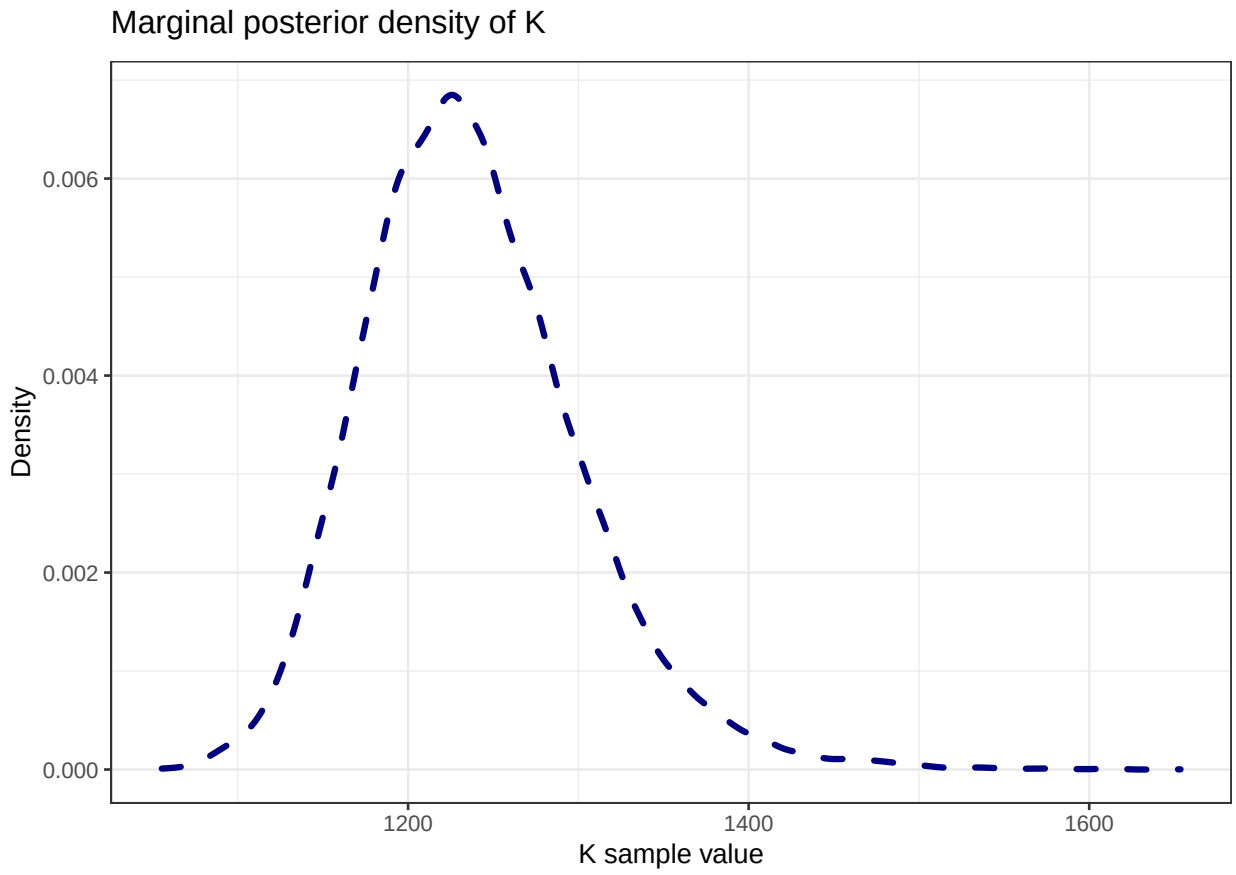
6) Make two publication quality plots of the marginal posterior density of `K`, one as a smooth curve and the other as a histogram.

```
ggplot(
    data = zm_df
    , mapping = aes(x = K)
  ) +
  geom_histogram(
    aes(y = ..density..)
    , bins = 100
    , fill = "navy"
    , alpha = 0.8
    , color = "gray25"
  ) +
  xlab(latex2exp::TeX("$K$ sample value")) +
  ylab("Density") +
  labs(
    title = latex2exp::TeX("Marginal posterior density of $K$")
  ) +
  theme_bw()
```




```
ggplot(
    data = zm_df
```

```
  , mapping = aes(x = K)
) +
geom_density(
  aes(y = ..density..)
  , linetype = 2
  , lwd = 1.2
  , color = "navy"
) +
xlab(latex2exp::TeX("$K$ sample value")) +
ylab("Density") +
labs(
  title = latex2exp::TeX("Marginal posterior density of $K$")
) +
theme_bw()
```



7) Compute the probability that $K > 1600$. Hint: what type of probability distribution would you use for this computation? Investigate the the dramatically useful R function `ecdf()`.

```
# Find the probability that the parameter K exceeds 1600
1 - stats::ecdf(zm_df$K)(1600)
```

```
## [1] 2e-04
```

8) Compute the probability that $1000 < K < 1600$.

```
# Find the probability that the parameter 1000 < K < 1300
stats::ecdf(zm_df$K)(1300) - stats::ecdf(zm_df$K)(1000)
```

```
## [1] 0.8477667
```

9) Compute the .025 and .975 quantiles of K. Hint–use the R `quantile()` function. This is an equal-tailed Bayesian credible interval on K.

```
quantile(zm_df$K, probs = c(.025, .975))
```

```
##     2.5%    97.5%
## 1130.669 1380.557
```

Exercise 6: Using `MCMCsummary`

1) Summarize the coda output from the logistic model with 4 significant digits. Include `Rhat` and effective sample size diagnostics (more about these soon).
2) Summarize the coda output for r alone.

1) Summarize the coda output from the logistic model with 4 significant digits. Include `Rhat` and effective sample size diagnostics (more about these soon).

```
MCMCvis::MCMCsummary(zm, digits = 4, Rhat = TRUE, n.eff = TRUE)
```

```
##            mean        sd      2.5%       50%     97.5% Rhat n.eff
## K     1.238e+03 6.372e+01 1.131e+03 1.232e+03 1381.0000    1  6387
## r     2.007e-01 9.814e-03 1.810e-01 2.008e-01    0.2196    1  7104
## sigma 2.867e-02 3.037e-03 2.351e-02 2.841e-02    0.0353    1 14788
## tau   1.257e+03 2.597e+02 8.026e+02 1.239e+03 1809.0000    1 16463
```

2) Summarize the coda output for r alone.

```
MCMCvis::MCMCsummary(zm, params = c("r"), digits = 4, Rhat = TRUE, n.eff = TRUE)
```

```
##     mean       sd  2.5%    50%  97.5% Rhat n.eff
## r 0.2007 0.009814 0.181 0.2008 0.2196    1  7104
```