MCMC(Metropolis-Hastings & Gibbs) & JAGS

Justin Kao

RICE

Course Tools

RStudio: Your Data Science Playground

> *I code the data, and I tell the story.*

1. Install R
2. Install Rstudio

Markdown & Quarto: Your Storytelling Superpowers

- 🧠 **RMarkdown or Quarto:** Make sure it *knits*, *renders*, and *saves* without RStudio crying.
  (If it knits, you're officially a data scientist.)
- 📝 **Markdown Basics** — Learn how to make text bold, italic, draw diagrams, or show off anything you want to demonstrate.
- 🚀 **Hello, Quarto** by Dr. Mine Çetinkaya-Rundel (Duke University) — From *"What's a chunk?"* to *"Look, I just published my work on GitHub!"*

Markov Chain Monte Carlo(MCMC)

MCMC - Metropolis-Hastings Algorithm

Let's recap of Key Concepts

- **Step 1:** Propose a candidate sample from the proposal (transition) distribution

$$g(x_{t+1} \mid x_t)$$

  For example, if the proposal is Normal,

$$g(x_{t+1} \mid x_t) = \mathcal{N}(x_t, \sigma^2)$$

- **Step 2:** Accept the proposed value with probability

$$A(x_t \to x_{t+1})$$

Detail Balance Condition to reach stationary: $p(a)T(a \to b) = p(b)T(b \to a), \quad \forall a, b$

$$\frac{f(a)}{NC} g(b \mid a) A(a \to b) = \frac{f(b)}{NC} g(a \mid b) A(b \to a)$$

MCMC - Metropolis-Hastings Algorithm

$$\frac{f(a)}{NC} g(b \mid a) A(a \to b) = \frac{f(b)}{NC} g(a \mid b) A(b \to a)$$

Rewrite the balance condition:

$$\frac{A(a \to b)}{A(b \to a)} = \underbrace{\frac{f(b)}{f(a)}}_{r_f} \times \underbrace{\frac{g(a \mid b)}{g(b \mid a)}}_{r_g}$$

- **Scenario 1**: $r_f r_g < 1, A(a \to b) = r_f r_g, A(b \to a) = 1$
- **Scenario 1**: $r_f r_g > 1, A(a \to b) = 1, A(b \to a) = \frac{1}{r_f r_g}$

So $A(a \to b) = \min\left(r_f r_g, 1\right)$

This matches the expression shown in Dr. Marina Vannucci's slides:

$$A(\theta_{t-1} \to \theta^* = \theta_t) = \rho\left(\theta^{(t-1)}, \theta^*\right) = \min\left\{\frac{p\left(\theta^* \mid x\right)}{p\left(\theta^{(t-1)} \mid x\right)} \frac{q\left(\theta^{(t-1)} \mid \theta^*\right)}{q\left(\theta^* \mid \theta^{(t-1)}\right)}, 1\right\}$$

$$\theta^{(t)} = \begin{cases} \theta^* & \text{with probability } \rho\left(\theta^{(t-1)}, \theta^*\right) \\ \theta^{(t-1)} & \text{with probability } \quad 1 - \rho\left(\theta^{(t-1)}, \theta^*\right) \end{cases}$$

MCMC - Metropolis-Hastings Algorithm - Intuition

Let's assume the distribution is symmetric

$$\frac{A(a \to b)}{A(b \to a)} = \underbrace{\frac{f(b)}{f(a)}}_{r_f} \times \underbrace{\frac{g(a \mid b)}{g(b \mid a)}}_{r_g} = \underbrace{\frac{f(b)}{f(a)}}_{r_f} = \frac{p(b)}{p(a)} = \frac{p\left(\theta^* \mid y\right)}{p\left(\theta^{(t-1)} \mid y\right)} = r$$

$$A(a \to b) = \min\left(1, \frac{f(b)}{f(a)}\right) = \min\left(1, \frac{p(b)}{p(a)}\right) = \min\left(1, \frac{p(\theta^*|y)}{p\left(\theta^{(t-1)}|y\right)} = r\right)$$

- If $r > 1$:
  - Intuition: Since $\theta^{(t-1)}$ is already in our set, we should include θ^* as it has a higher probability than $\theta^{(t-1)}$.
  - Procedure: Accept θ^* into our set, i.e. set $\theta^{(s)} = \theta^*$.
- If $r < 1$:
  - Intuition: The relative frequency of θ-values in our set equal to θ^* compared to those equal to $\theta^{(t-1)}$ should be $p\left(\theta^* \mid y\right) / p\left(\theta^{(t-1)} \mid y\right) = r$. This means that for every instance of $\theta^{(t-1)}$, we should have only a "fraction" of an instance of a θ^* value.
  - Procedure: Set $\theta^{(t)}$ equal to either θ^* or $\theta^{(t-1)}$, with probability r and $1 - r$ respectively.

MCMC - Metropolis-Hastings Algorithm(HW problem 1)

Consider the file `school1.` dat (from the Book website) which contains data on the amount of times students at a high school spent on studying or homework during an exam period. Suppose we fit the following model:

$$Y_i \sim \mathcal{N}\left(\mu, \sigma^2\right), \quad i = 1, \ldots, n$$
$$\mu \sim \mathcal{N}\left(\mu_0, \tau_0^2\right)$$

with $\mu_0 = 5, \tau_0^2 = 0.5$. Let us fix $\sigma^2 = 1$. Implement a Metropolis-Hastings:

```r
library(coda)

# Read data
y <- scan(url("https://www2.stat.duke.edu/~pdh10/FCBS/Exercises/school1.dat"))
n <- length(y)

# Known hyperparameters
mu0 <- 5
tau0_sq <- 0.5
sigma_sq <- 1

# Log-posterior: log p(mu | y) ∝ log p(y | mu) + log p(mu)
log_post <- function(mu, y, mu0, tau0_sq, sigma_sq) {
  log_lik <- sum(dnorm(y, mean = mu, sd = sqrt(sigma_sq), log = TRUE))
  log_prior <- dnorm(mu, mean = mu0, sd = sqrt(tau0_sq), log = TRUE)
  log_lik + log_prior
}
```

MCMC - Metropolis-Hastings Algorithm(HW problem 1, cont'd)

- A proposal function for μ centered at the previously sampled value.
- My proposal $\mu^* \sim q\left(\mu^* \mid \mu^{(0)}\right) = N\left(\mu^{(0)}, \sigma_p^2\right)$, where we can set $\mu^{(0)} = \bar{y}$

```
mean(y)
```

```
[1] 9.464
```

- $\mu^* = \mu^{(t-1)} + \varepsilon, \quad \varepsilon \sim N\left(0, \sigma_p^2\right)$ Hence, $q\left(\mu^* \mid \mu^{(t-1)}\right) = N\left(\mu^{(t-1)}, \sigma_p^2\right)$

```
# Metropolis–Hastings sampler (Random-Walk proposal)
mh_sampler <- function(sigma_p, n_iter = 5000, burn_in = 1000) {
  mu <- numeric(n_iter)
  mu[1] <- mean(y)

  for (t in 2:n_iter) {
    mu_prop <- rnorm(1, mean = mu[t - 1], sd = sigma_p)
    log_r <- log_post(mu_prop, y, mu0, tau0_sq, sigma_sq) -
             log_post(mu[t - 1], y, mu0, tau0_sq, sigma_sq)
    if (log(runif(1)) < log_r) {
      mu[t] <- mu_prop
    } else {
      mu[t] <- mu[t - 1]
    }
  }

  # Drop burn-in and return MCMC object
  chain <- mcmc(mu[(burn_in + 1):n_iter])
  return(chain)
}
```

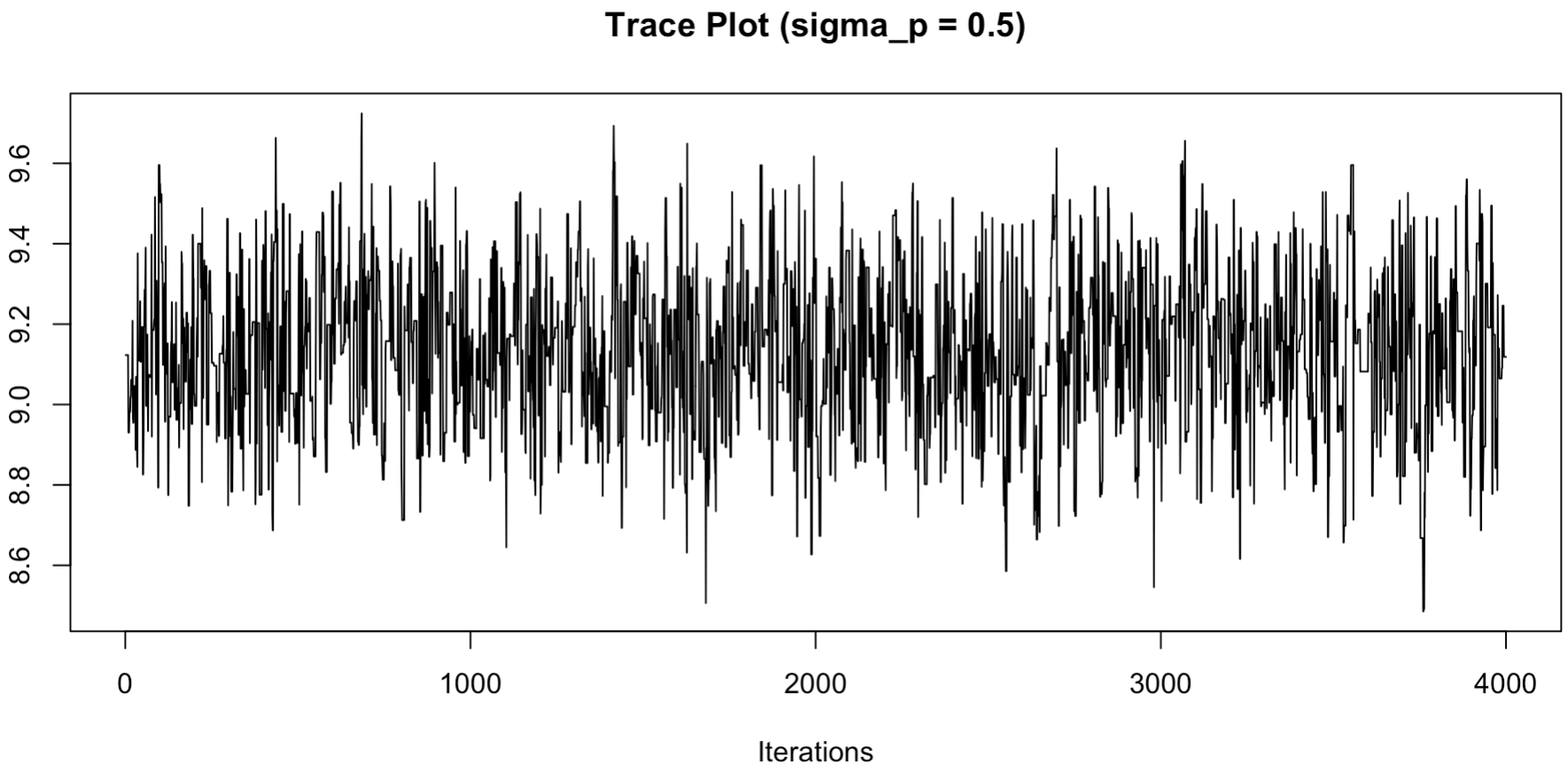
MCMC - Metropolis-Hastings Algorithm(HW problem 1, contd)

```r
# Run chain with sigma_p = 0.5
set.seed(425)
chain_small <- mh_sampler(sigma_p = 0.5)

# Summaries and plots
summary(chain_small)
```

```
Iterations = 1:4000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 4000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

          Mean             SD       Naive SE Time-series SE
      9.132470       0.193392       0.003058       0.006434

2. Quantiles for each variable:

 2.5%   25%   50%   75% 97.5%
8.768 9.002 9.133 9.264 9.505
```

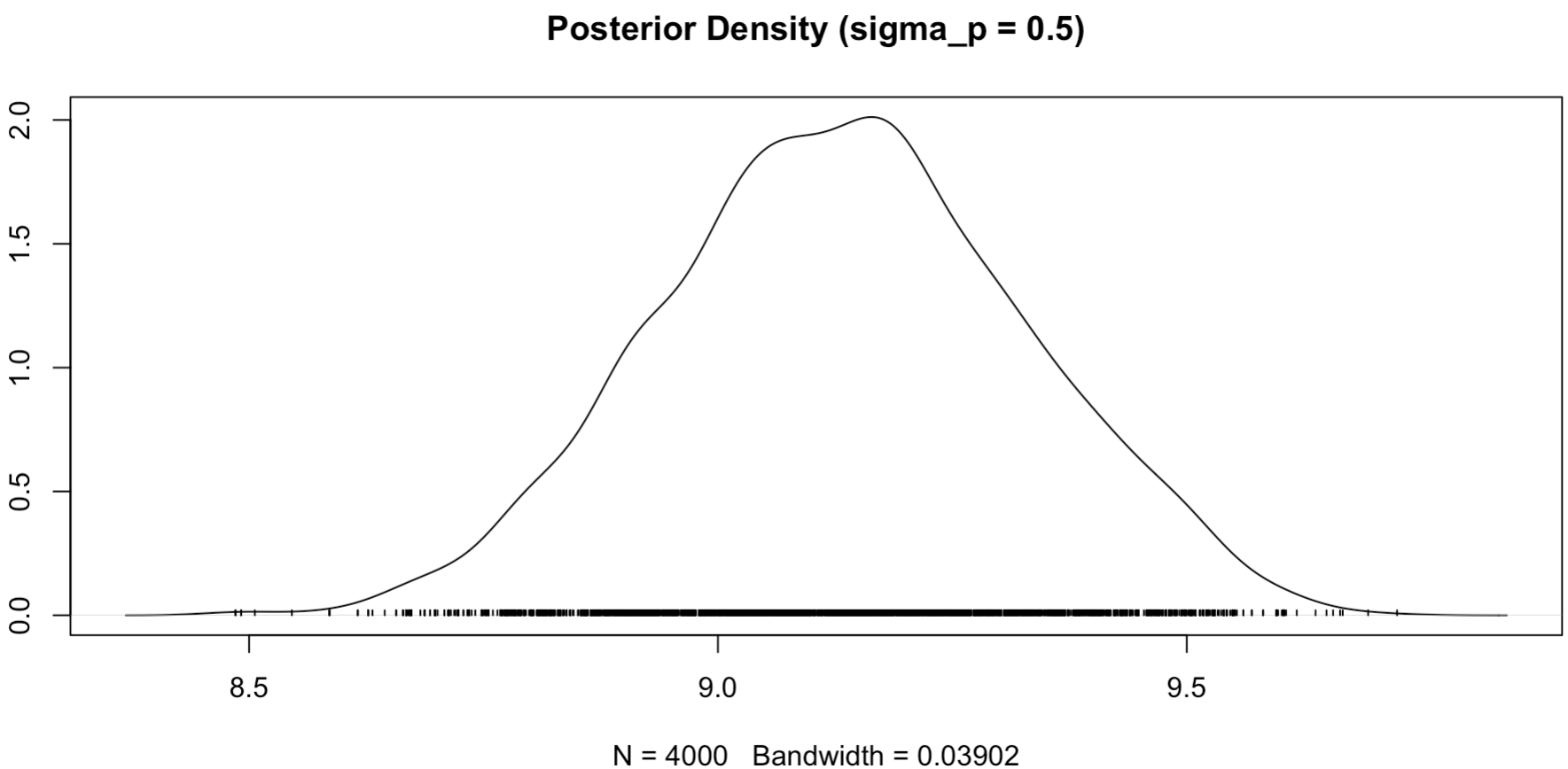
MCMC - Metropolis-Hastings Algorithm(HW problem 1, contd)

```
1 traceplot(chain_small, main = "Trace Plot (sigma_p = 0.5)")
```



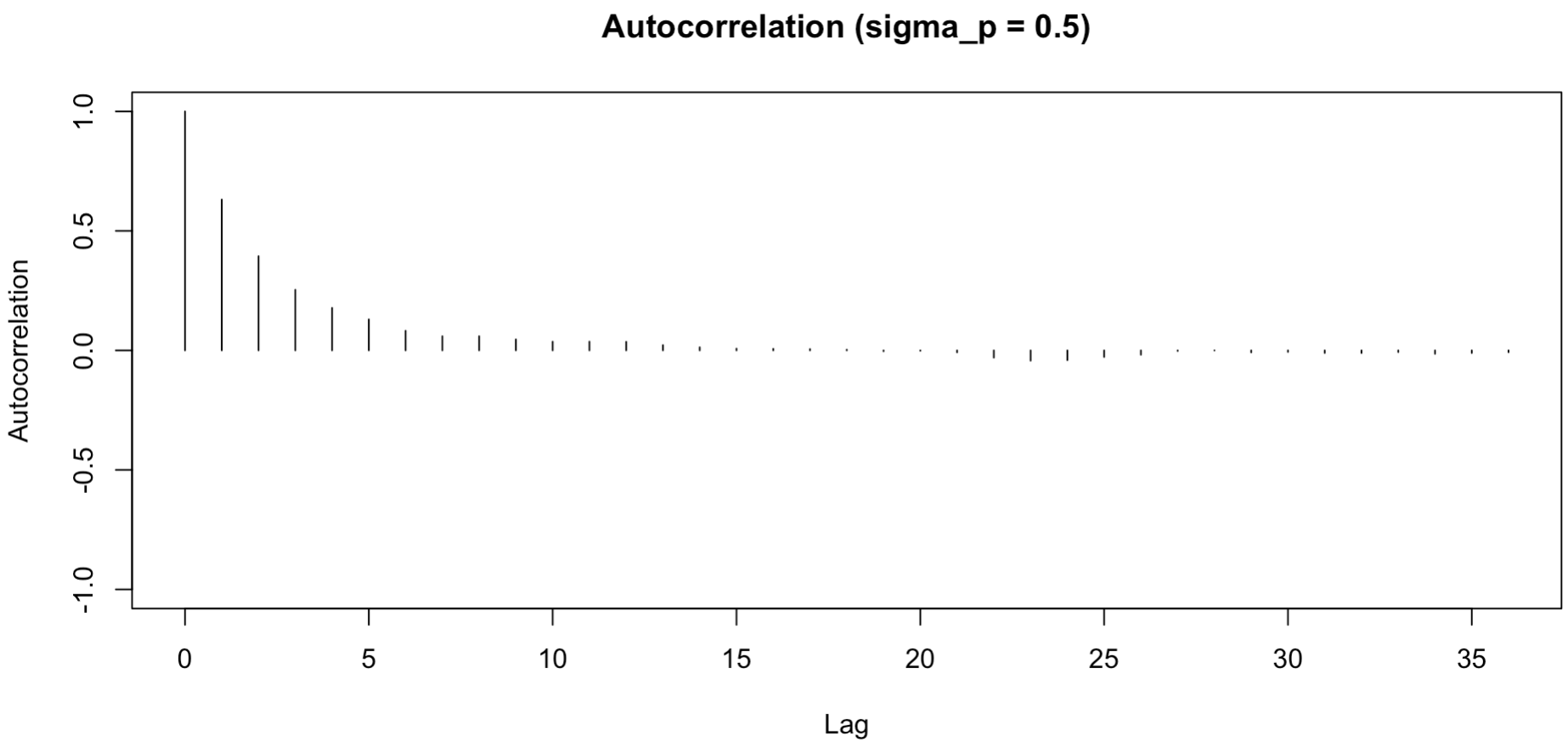
MCMC - Metropolis-Hastings Algorithm(HW problem 1, contd)

```
1 densplot(chain_small, main = "Posterior Density (sigma_p = 0.5)")
```



MCMC - Metropolis-Hastings Algorithm(HW problem 1, contd)

```
1 autocorr.plot(chain_small, main = "Autocorrelation (sigma_p = 0.5)")
```



MCMC - Metropolis-Hastings Algorithm(HW problem 1, contd)

(b) Provide a trace plot and a plot of the histogram and/or kernel density estimate for each chain.

> To answer this, you can run the following code and then compare the plots:

```
set.seed(123)
chain1 <- mh_sampler(sigma_p = 0.5)
set.seed(456)
chain2 <- mh_sampler(sigma_p = 0.5)
set.seed(789)
chain3 <- mh_sampler(sigma_p = 0.5)
```

- `summary()`,`autocorr.plot()`,`densplot()`
- I have demonstrated the case when $\sigma_p = 0.5$. Don't forget to try $\sigma_p = 2$!

MCMC - Metropolis-Hastings Algorithm(HW problem 1, cont'd)

(c) After removing an appropriate burn-in, evaluate the acceptance rate and the autocorrelation for varying lags. Which value of σ_p provides a better choice for the proposal function?

```r
# Acceptance rate function
acceptance_rate <- function(chain) {
  draws <- as.numeric(chain)
  mean(diff(draws) != 0)
}
acceptance_rate(chain_small)
```

```
[1] 0.4211053
```

(This is just an arbitrary number I came up with — you should check your own chain.)

The pattern is likely to be:

- $\sigma_p = 0.5$ accepts $\sim 82\%$ proposals $\rightarrow$ too conservative.
- $\sigma_p = 2$ accepts $\sim 27\%$ proposals $\rightarrow$ better exploration.

MCMC - Gibbs sampler(HW problem 2)

Gibbs sampler. Consider the file `school1.` dat (from the Book) which contains data on the amount of times students at a high school spent on studying or homework during an exam period. Suppose we fit the following model:

$$
\begin{aligned}
Y_i &\sim \mathcal{N}\left(\mu, \sigma^2\right), \quad i = 1, \ldots, n \\
\mu &\sim \mathcal{N}\left(\mu_0, \tau_0^2\right) \\
\sigma^2 &\sim \text{Inv-Gamma}(\alpha, \beta)
\end{aligned}
$$

with $\mu_0 = 5, \tau_0^2 = 0.5, \alpha = 1, \beta = 4$.

(a) Write down the full conditionals, $p\left(\mu \mid \sigma^2, \boldsymbol{y}\right)$ and $p\left(\sigma^2 \mid \mu, \boldsymbol{y}\right)$.

$$
\begin{aligned}
p\left(\mu, \sigma^2 \mid \mathbf{y}\right) &\propto p\left(\mathbf{y} \mid \mu, \sigma^2\right) p(\mu) p\left(\sigma^2\right) \\
&\propto \left(\sigma^2\right)^{-n/2} \exp\left[-\frac{1}{2\sigma^2} \sum \left(y_i - \mu\right)^2\right] \times \exp\left[-\frac{1}{2\tau_0^2}\left(\mu - \mu_0\right)^2\right] \times \left(\sigma^2\right)^{-(\alpha+1)} \exp\left(-\frac{\beta}{\sigma^2}\right) \\
&\propto \left(\sigma^2\right)^{-(\alpha+n/2+1)} \exp\left[-\frac{1}{2\sigma^2} \sum \left(y_i - \mu\right)^2 - \frac{\left(\mu - \mu_0\right)^2}{2\tau_0^2} - \frac{\beta}{\sigma^2}\right].
\end{aligned}
$$

MCMC - Gibbs sampler(HW problem 2, cont'd)

Say the full conditioonals are:

- $\mu \mid \sigma^2, \mathbf{y} \sim N\left(\mu_n, \tau_n^2\right), \quad \text{where } \mu_n = \dfrac{\frac{n\bar{y}}{\sigma^2} + \frac{\mu_0}{\tau_0^2}}{\frac{n}{\sigma^2} + \frac{1}{\tau_0^2}}, \quad \tau_n^2 = \left(\dfrac{n}{\sigma^2} + \dfrac{1}{\tau_0^2}\right)^{-1}$
- $\sigma^2 \mid \mu, \mathbf{y} \sim \text{Inv-Gamma}\left(\alpha + \dfrac{n}{2}, \beta + \dfrac{1}{2}\sum (y_i - \mu)^2\right)$

Let's again setup our parameters first: $\mu_0 = 5, \tau_0^2 = 0.5, \alpha = 1, \beta = 4$

```r
# Parameters
mu0 <- 5
tau0_sq <- 0.5
alpha <- 1
beta <- 4

# Data
y <- scan(url("https://www2.stat.duke.edu/~pdh10/FCBS/Exercises/school1.dat"))
n <- length(y)
ybar <- mean(y)
```

MCMC - Gibbs sampler(HW problem 2, cont'd)

```r
# Storage
n_iter <- 10000
mu <- numeric(n_iter)
sigma2 <- numeric(n_iter)

# Initial values
mu[1] <- mean(y)
sigma2[1] <- var(y)

# Gibbs sampling
for (t in 2:n_iter) {
  # 1. Sample mu | sigma2, y
  tau_n_sq <- 1 / (n / sigma2[t-1] + 1 / tau0_sq)
  mu_n <- tau_n_sq * (n * ybar / sigma2[t-1] + mu0 / tau0_sq)
  mu[t] <- rnorm(1, mu_n, sqrt(tau_n_sq))

  # 2. Sample sigma2 | mu, y
  alpha_n <- alpha + n/2
  beta_n <- beta + 0.5 * sum((y - mu[t])^2)
  sigma2[t] <- 1 / rgamma(1, shape = alpha_n, rate = beta_n)
}

# Burn-in and analysis
burn_in <- 2000
mu_samp <- mu[(burn_in + 1):n_iter]
sigma2_samp <- sigma2[(burn_in + 1):n_iter]

```

MCMC - Gibbs sampler(HW problem 2, cont'd)

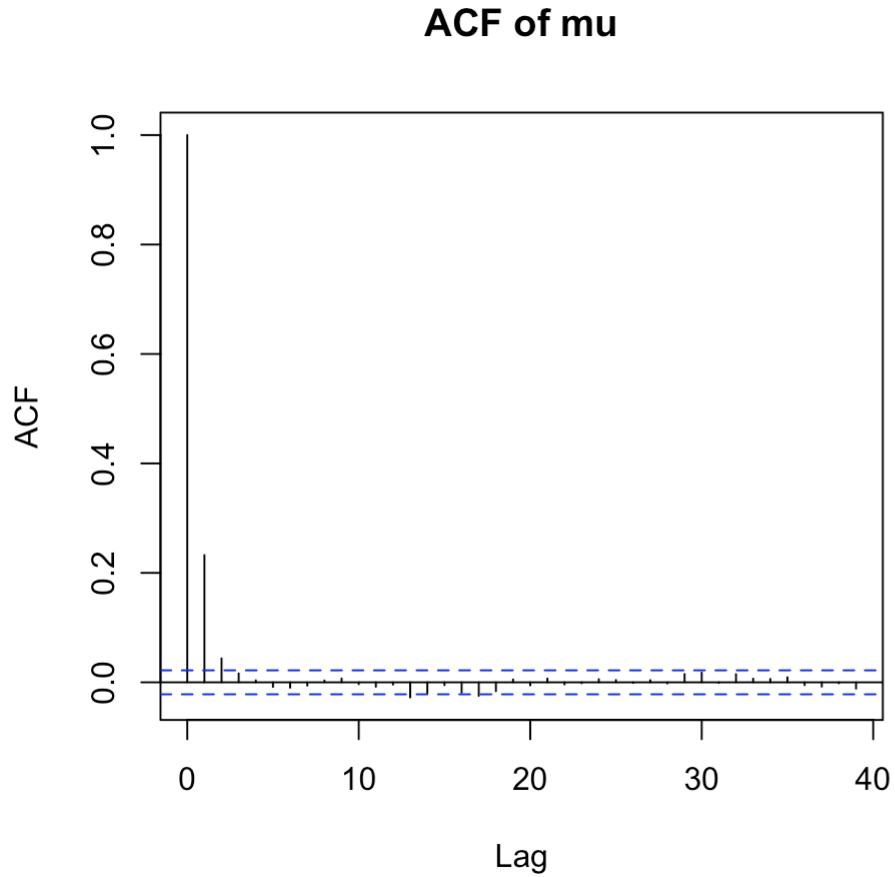
```
1 effectiveSize(mcmc(mu_samp))
```

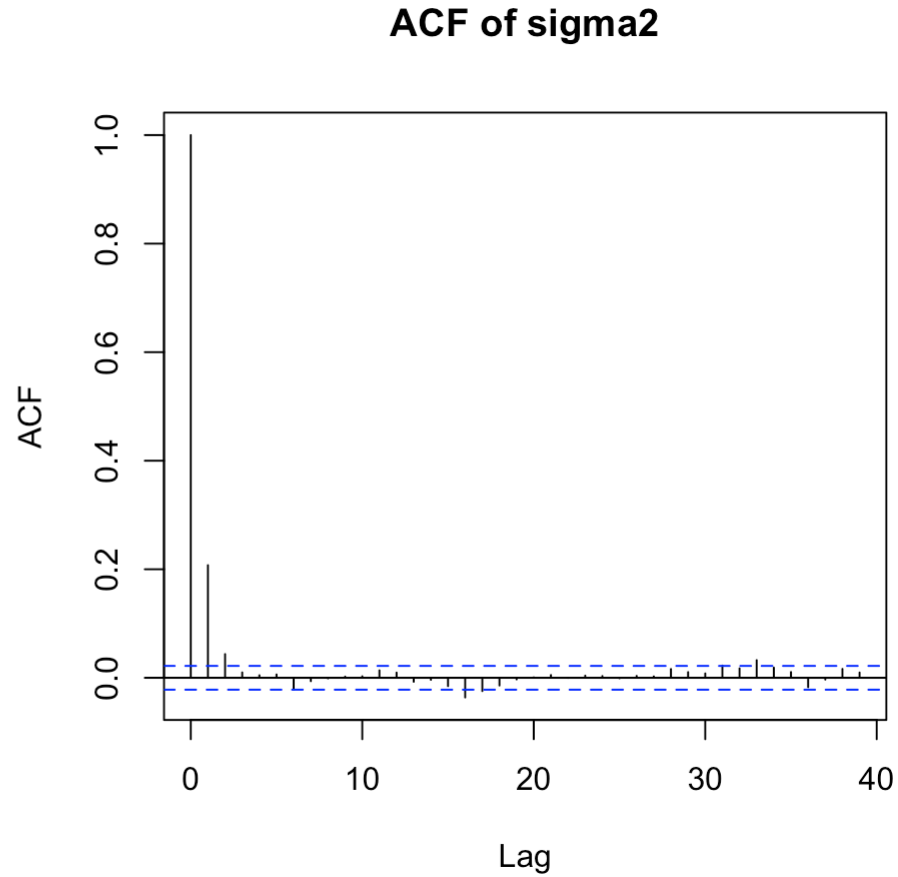
```
  var1
4980.3
```

```
1 effectiveSize(mcmc(sigma2_samp))
```

```
    var1
5249.669
```

```
1 par(mfrow = c(1, 2))
2 acf(mu_samp, main = "ACF of mu")
3 acf(sigma2_samp, main = "ACF of sigma2")
```

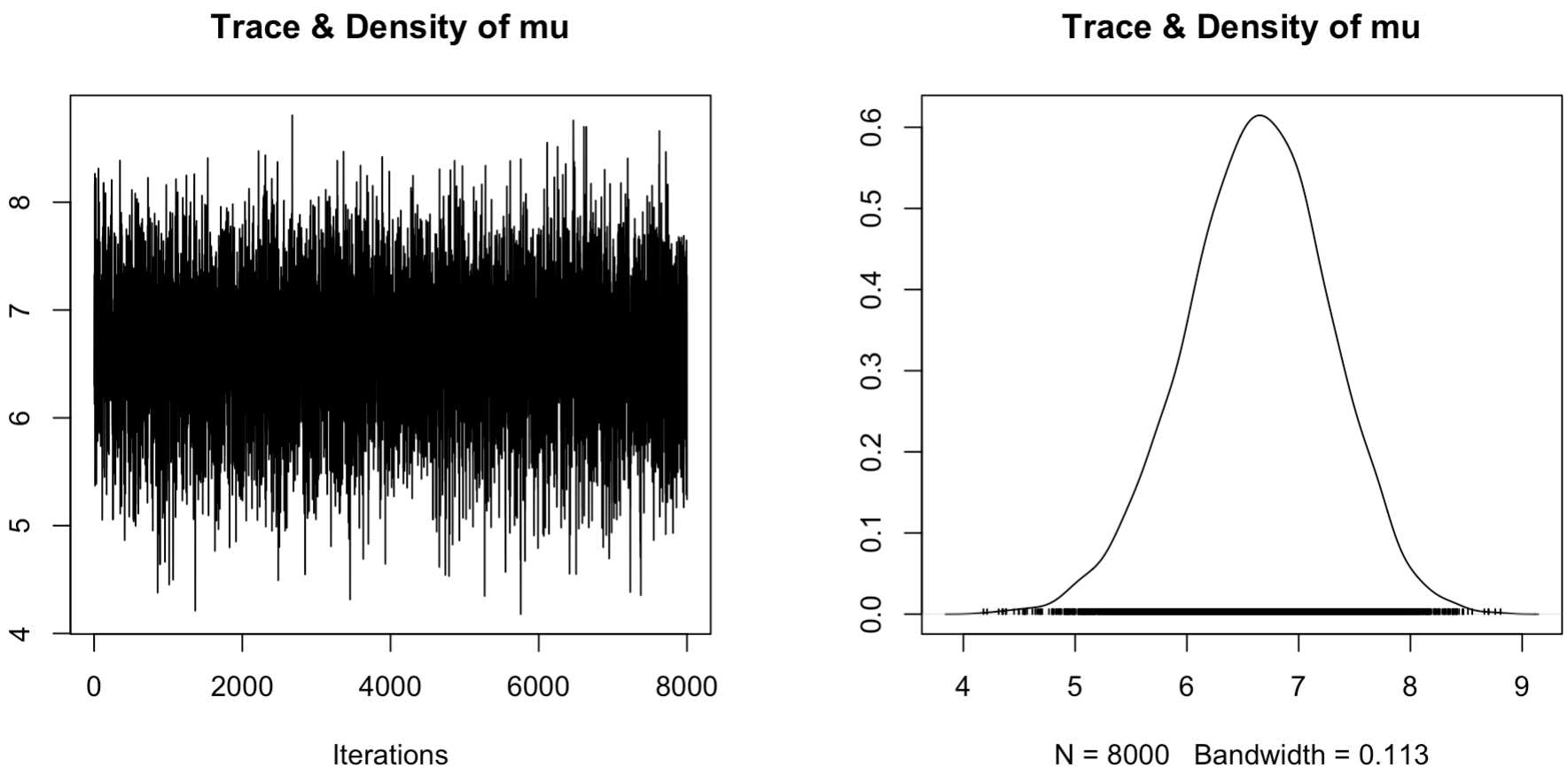





```r
par(mfrow = c(1, 1))  # reset layout
```

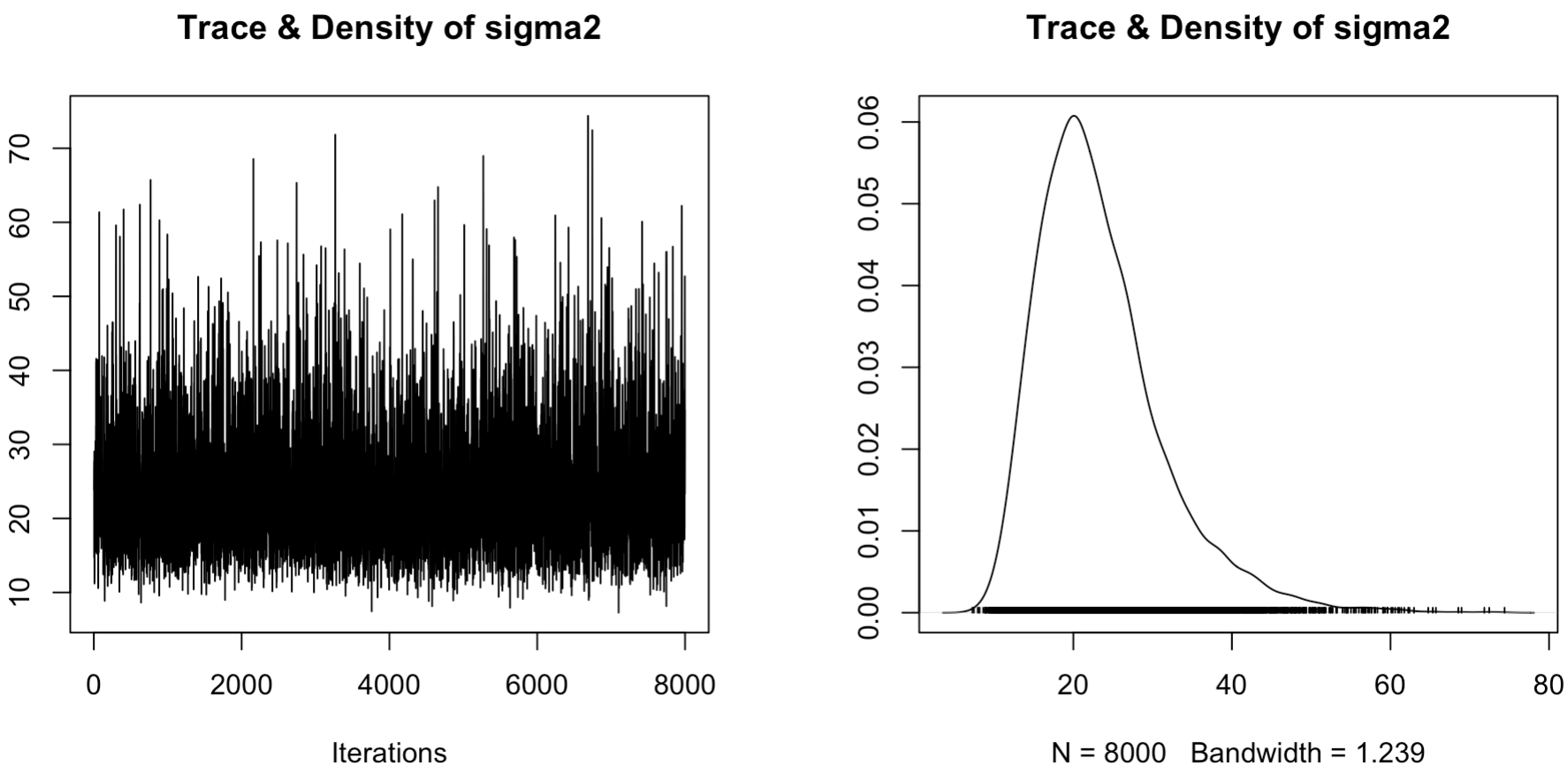
MCMC - Gibbs sampler(HW problem 2, cont'd)

```
1  plot(mu_mcmc, main = "Trace & Density of mu")
```



MCMC - Gibbs sampler(HW problem 2, cont'd)

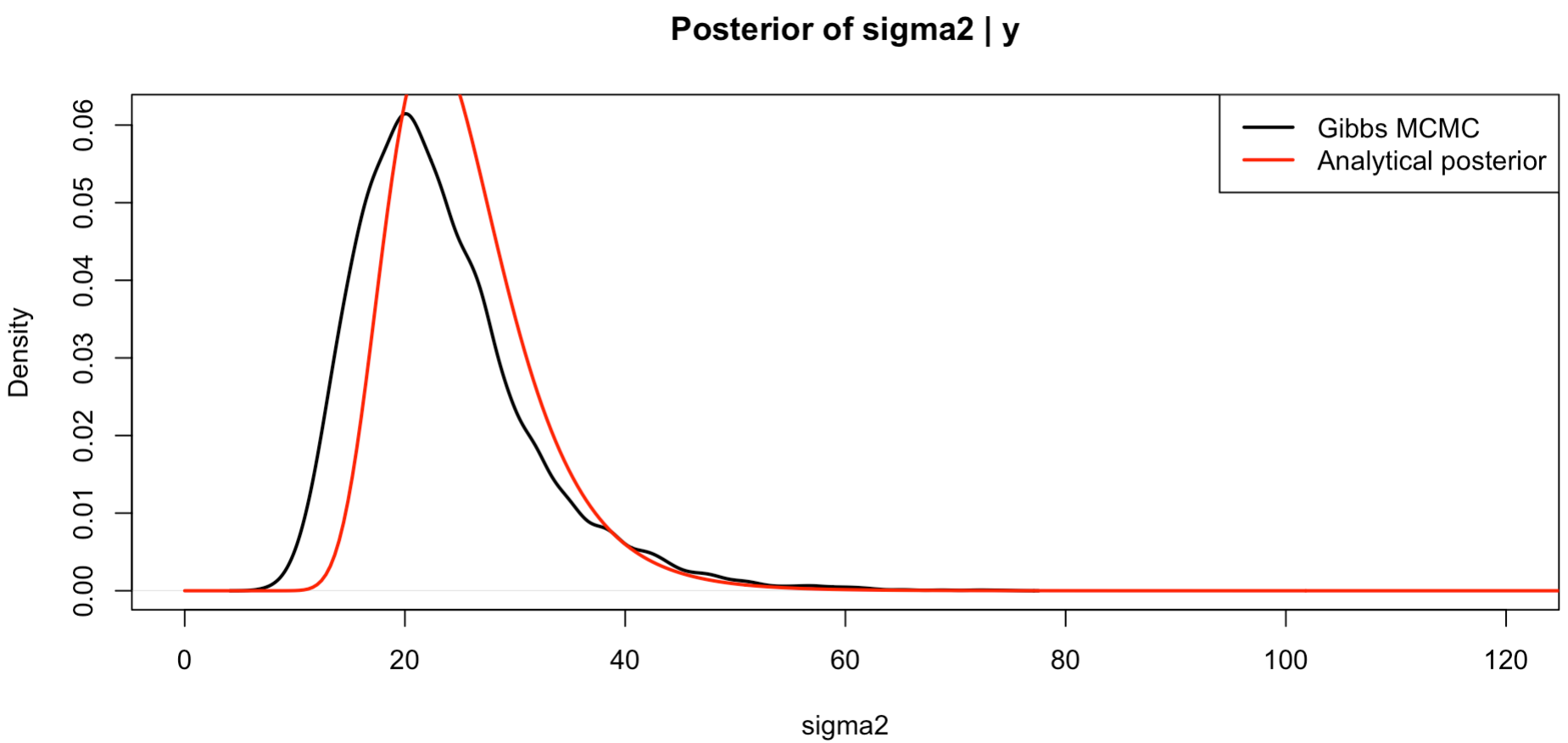
```
1  plot(sigma2_mcmc, main = "Trace & Density of sigma2")
```



RICE

MCMC - Gibbs sampler(HW problem 2, cont'd)

(d) Evaluate the marginal posterior density $p\left(\sigma^2 \mid \boldsymbol{y}\right)$ analytically up to a normalizing constant and overlay this density on your plot from (c) [Hint: you can perform the comparison by evaluating the unnormalized density on a grid of values].



RICE

MCMC - Gibbs sampler & MH(package(coda))

- You can technically wrap any numeric vector or matrix(`object` here) in `mcmc_obj <- mcmc(object)` , even if it isn't a "real" MCMC sequence. However - and this is key - doing so doesn't make it an MCMC chain in a statistical sense. It just labels it as one so that functions like `plot()`, `acf()`, and `effectiveSize()` can run without error.

Function	Purpose
`effectiveSize(mcmc_obj)`	Computes **Effective Sample Size (ESS)** — measures chain independence.
`gelman.diag(mcmc.list_obj)`	**Gelman–Rubin diagnostic** for checking convergence across multiple chains.
`geweke.diag(mcmc_obj)`	**Geweke diagnostic** for within-chain convergence (compares early vs. late samples).
`heidel.diag(mcmc_obj)`	**Heidelberger–Welch test** for stationarity and convergence.
`raftery.diag(mcmc_obj)`	**Raftery–Lewis diagnostic** estimating chain length needed for precision.

JAGS - Just Another Gibbs Sampler

The Posterior That Looks Easy but Isn't — JAGS Is Your Best Friend!

Model:

$$y_i \sim \text{Cauchy}(\theta, \sigma), \quad \theta \sim \mathcal{N}(0, 10^2), \quad \sigma \sim \text{Uniform}(0, 10)$$

Posterior:

$$p(\theta, \sigma \mid y) \propto \left[\prod_{i=1}^{n} \frac{1}{\pi\sigma \left(1 + \frac{(y_i - \theta)^2}{\sigma^2}\right)} \right] \exp\left(-\frac{\theta^2}{2 \times 10^2}\right)$$

It only has **two parameters**, yet the posterior is:

- ❌ No full conditional — not a known distribution
- ❌ Non-conjugate — prior and likelihood don't simplify
- ❌ No closed-form normalization — integral can't be solved

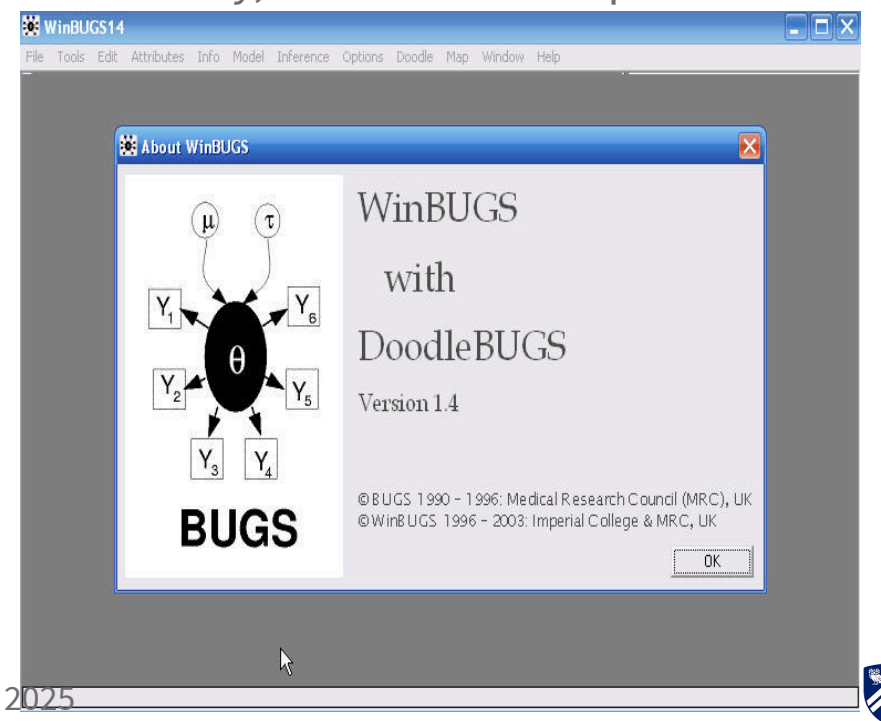
Result: $p(\theta, \sigma|y)$ has **no analytic form**, no direct sampler, and no closed expression.

> 🤢 Manual derivation stops here.

What Are BUGS and JAGS?

- **BUGS (Bayesian inference Using Gibbs Sampling):**
  Early Bayesian software (1990s, Cambridge) using Gibbs sampling. Versions include *Classic BUGS(1990s)*, *WinBUGS(Windows-only)*, and *OpenBUGS(open-source successor)*.
- **JAGS (Just Another Gibbs Sampler):**
  Developed by Martyn Plummer in 2003 as a **cross-platform**, **open-source**, and **R-integrated** version of BUGS.

> 💡 JAGS keeps the same model syntax as BUGS but adds flexibility, automatic Metropolis–within–Gibbs sampling, and modern extensibility.



Why JAGS

1. **Doing MCMC by hand means:**
    - Writing out **priors** and **likelihood**
    - Deriving **full conditional posteriors** for Gibbs sampling
    - Specifying a **proposal distribution** for Metropolis–Hastings
    - Tracking the **update order** for every parameter
2. As models grow complex, this becomes
    - 🔁 *tedious*,
    - 🧮 *algebraically messy*, and
    - 🚫 *often intractable.*
3. You want to spend time **analyzing**, not deriving.

> JAGS can help you solve this once you specify the prior and the likelihood!

Enter JAGS

- **JAGS** automates MCMC sampling.
- You define your model using **BUGS-like syntax**.
- It automatically:
  - Constructs the **posterior**
  - Performs **Gibbs** / **Metropolis-Hastings sampling**
  - Returns results to **R** through `rjags` or `R2jags`.

How JAGS Samples Behind the Scenes

Even within Gibbs sampling, not all parameters have closed-form conditionals.

JAGS handles this **automatically**:

Parameter Type	Sampling Strategy	Description
Conjugate blocks	**Pure Gibbs**	Exact draws from known conditionals
Non-conjugate blocks	**Metropolis–within–Gibbs**	Uses adaptive random-walk MH inside Gibbs
Difficult continuous nodes	**Slice sampling**	Efficient for highly nonlinear posteriors

So Gibbs and Metropolis **run in parallel**:

- Each iteration updates some parameters by **Gibbs**, others by **Metropolis**
- JAGS adapts proposal scales automatically during burn-in
- You don’t need to code or tune anything

> 💡 *You define the model; JAGS decides the best sampler for each part.*

How to install JAGS(Mac)

- Option 1: Download JAGS manually
- Option 2: Homebrew (Mac)

```
brew install jags #run this command in terminal
brew list --versions jags #check version
brew uninstall jags #uninstall jags
brew upgrade #upgrade
```

- If you download manually, how to remove?

```
#run this command in terminal
which jags

#if you get this, then JAGS was installed manually
/usr/local/bin/jags

# To remove it, do
sudo rm -rf /usr/local/bin/jags
sudo rm -rf /usr/local/lib/JAGS
sudo rm -rf /usr/local/share/jags
sudo rm -rf /usr/local/include/jags

# Verify JAGS is removed
which jags
```

How to install JAGS(Windows)

1. Download JAGS manually

2. Using Command Prompt (CMD)

```
# Step 1 – Check if JAGS is installed
jags

# If JAGS is installed, you will see: Welcome to JAGS 4.3.2 (official binary)

# Type "exit" to close JAGS
exit

# Step 2 – Check the installation path (optional)
where jags

# Step 3 – To uninstall JAGS
# Go to Control Panel → Programs → Programs and Features
# Find "JAGS 4.x.x" → Click Uninstall

# Or delete manually (if Control Panel fails)
rmdir /S /Q "C:\Program Files\JAGS"

# Step 4 – Verify removal. It should show: 'jags' is not recognized as an internal or external command
jags
```

Loading JAGS(library(rjags)) in R

```r
# install.packages(c("rjags", "coda"))

# Ensure dependencies are installed
# (coda must be loaded before rjags, since rjags depends on it)
install.packages(
  "rjags",
  type = "source",
  repos = "https://cloud.r-project.org",
  configure.args = "--with-jags-prefix=/opt/homebrew/opt/jags"
)

# Load required packages
library(coda)   # For MCMC diagnostics and output handling
library(rjags)  # Interface to the JAGS (Just Another Gibbs Sampler) engine
```

BUGS syntax - Example 1

$$y_i \sim N\left(\mu, \sigma^2\right), \quad \mu \sim N(0, 100), \quad \sigma \sim \text{Uniform}(0, 10)$$

```
# BUGS syntax (general form)
  "
  model {
    for (i in 1:N) {
      # Likelihood
      y[i] ~ distribution(parameter1, parameter2)   # stochastic: likelihood for data
    }

    # Priors for parameters
    parameter1 ~ prior_distribution1(hyperparameter_set1)   # prior for parameter1
    parameter2_raw ~ prior_distribution2(hyperparameter_set2)  # prior for parameter2 (before transformation)

    # Deterministic relationship (if needed)
    parameter2 <- g(parameter2_raw)    # e.g., transformation such as 1 / variance
  }
  "
```

Then, append the `model{}` block into the R code

```
# Model specification
model_code <-
  "
  model {
    for (i in 1:N) {
      y[i] ~ dnorm(mu, tau)  # likelihood for data
    }
    # Priors for parameters
    mu ~ dnorm(0, 0.01)      # Prior: mu ~ N(0, 100), in JAGS, always use precision = 1/100 = 0.01
    sigma ~ dunif(0, 10)     # Prior: sigma ~ Uniform(0, 10)
    tau <- pow(sigma, -2)    # Precision tau = 1 / sigma^2
  }
"

# Write model to file
writeLines(model_code, "model.txt")
```

BUGS syntax - Example 2

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i, \quad \varepsilon_i \sim N\left(0, \sigma^2\right), \quad \beta_0 \sim N(0, 1000), \beta_1 \sim N(0, 1000), \sigma^2 \sim \text{Inv-Gamma}(0.001, 0.001)$$

```
# BUGS syntax (general linear regression form)
  "
  model {
    for (i in 1:N) {
      # Likelihood
      y[i] ~ dnorm(mu[i], tau)             # stochastic: likelihood for data
      mu[i] <- beta0 + beta1 * x[i]        # deterministic: mean structure
    }

    # Priors for regression coefficients
    beta0 ~ dnorm(0, 0.001)                # Prior: β₀ ~ N(0, 1000)
    beta1 ~ dnorm(0, 0.001)                # Prior: β₁ ~ N(0, 1000)

    # Prior for error variance
    tau <- 1 / pow(sigma, 2)               # Precision = 1 / σ²
    sigma ~ dunif(0, 100)                  # Equivalent to Inv-Gamma(0.001, 0.001)
  }
  "
```

Then, append the `model{}` block into the R code

```
# Model specification
model_code_2 <-
  "
  model {
    for (i in 1:N) {
      y[i] ~ dnorm(mu[i], tau)             # likelihood for data
      mu[i] <- beta0 + beta1 * x[i]        # mean structure
    }
    beta0 ~ dnorm(0, 0.001)                # prior for intercept
    beta1 ~ dnorm(0, 0.001)                # prior for slope
    sigma ~ dunif(0, 100)                  # prior for standard deviation
    tau <- pow(sigma, -2)                  # precision = 1 / sigma^2
  }
  "

# Write model to file
writeLines(model_code_2, "model_2.txt")
```

🚀 Hands-On with rjags: Let’s Build a Bayesian Model

Example 1 (Cont'd)

Suppose we would like to run a model to draw inference about the mean for the following model:

$$y_i \sim N\left(\mu, \sigma^2\right)$$

where y_i are the data include 30 observations.

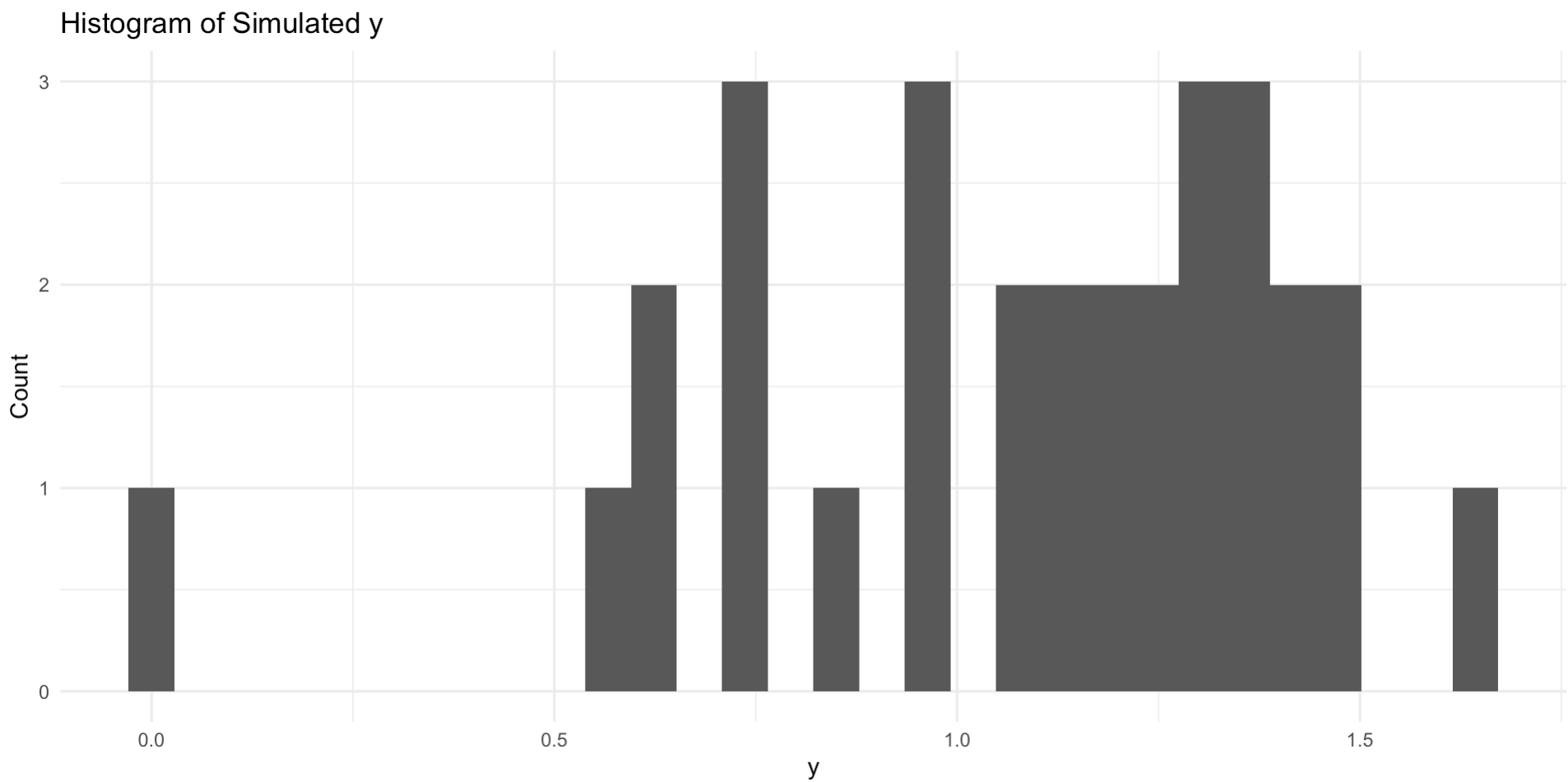
Priors are:

$$\mu \sim N(0, 100)$$
$$\sigma \sim \text{Uniform}(0, 10)$$

```r
# Ground Truth
rm(list=ls())
n = 30
mu = 1.12
sigma = 0.38

# Generate values (stochastic part of the model)
# In reality, you only observe 30 data points, and the true population mean
# and variance are unknown. Our goal is to estimate these parameters.
set.seed(425)
y = rnorm(n, mean = mu, sd = sigma)
```

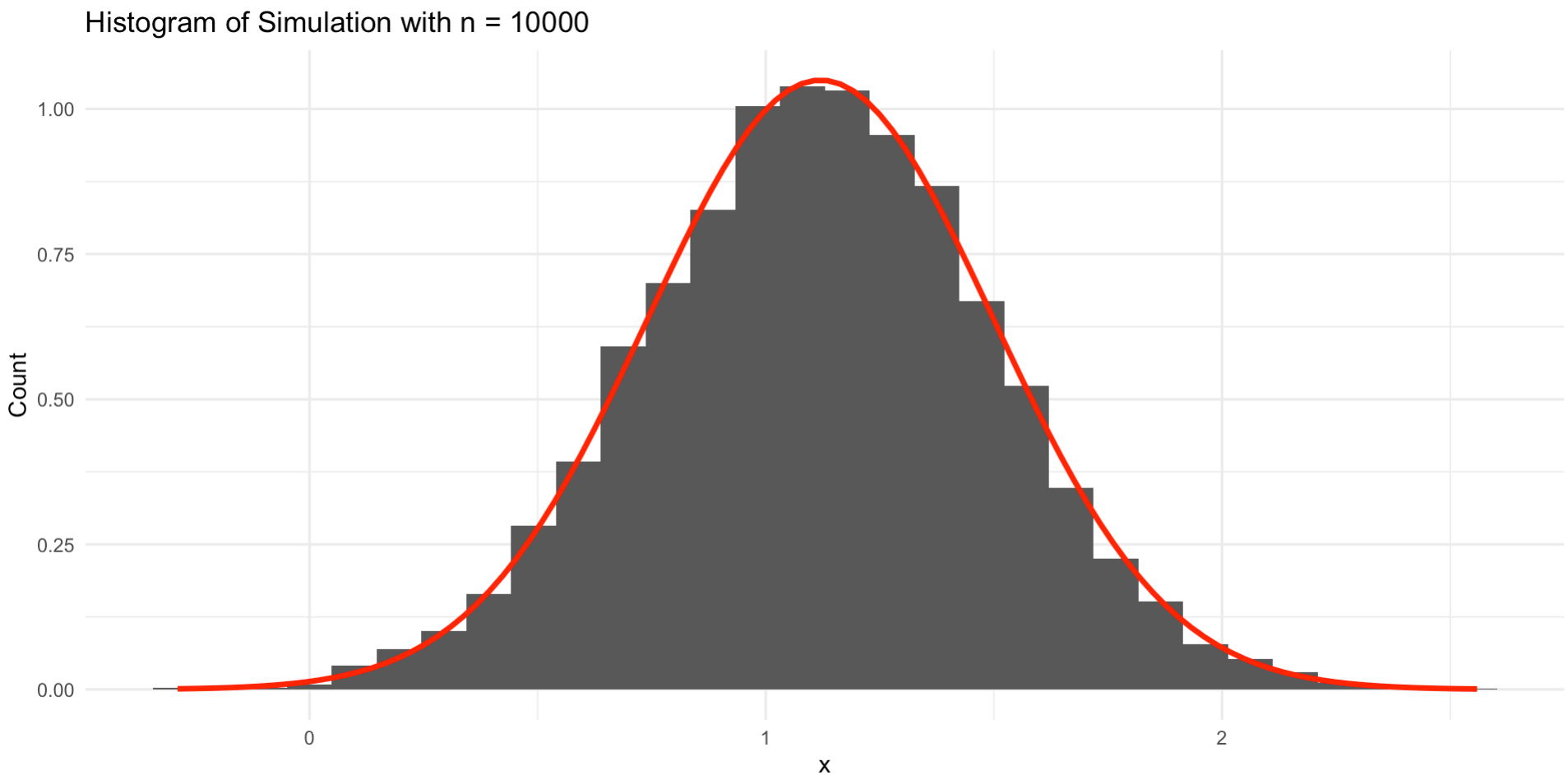
Example 1 (Cont'd)

```r
df = data.frame(y)
library(ggplot2)
ggplot(df, aes(x = y)) +
  geom_histogram(bins = 30) +
  labs(title = "Histogram of Simulated y", x = "y", y = "Count") +
  theme_minimal()
```



Example 1 (Cont'd)

If we put `n = 10000`, it is indeed the bell shape!



Example 1 (Cont'd)

- Recall our BUGS syntax: `writeLines(model_code, "model.txt")`
- You don't really need to specify `inits`

```r
# Prepare data for JAGS
jagsData <- list(
  y = y,
  N = length(y)
)

# Initial values for the model
jags_inits <- function() {
  list(mu = rnorm(1, 0, 10), sigma = runif(1, 0, 10)) # priors
}

# Parameters to monitor
jags_params <- c("mu", "sigma","tau")

# Load rjags and run the model
library(rjags)
model_1 <- jags.model(file = "model.txt", data = jagsData, inits = jags_inits, n.chains = 3)
```

```
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 30
   Unobserved stochastic nodes: 2
   Total graph size: 39

Initializing model
```

Example 1 (Cont'd)

```r
# Sample from posterior
samples <- coda.samples(model_1, variable.names = jags_params, n.iter = 2000, n.burnin = 1000)

# Convert to matrix for easier manipulation
output <- as.matrix(samples)

# Extract results
mu_samples <- output[, "mu"]
sigma_samples <- output[, "sigma"]
tau_samples <- output[,"tau"]

# Summary statistics
# posterior prediction
mean(mu_samples)
```

```
[1] 1.082436
```

```r
mean(sigma_samples)
```

```
[1] 0.3665139
```

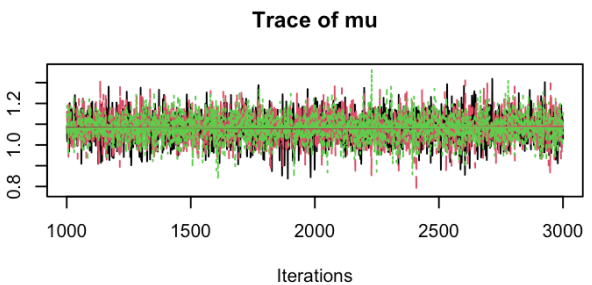
```r
mean(tau_samples)
```

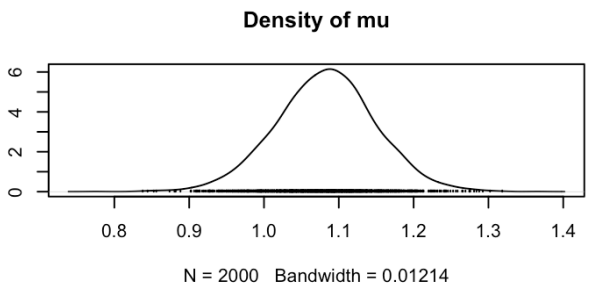
```
[1] 7.876675
```

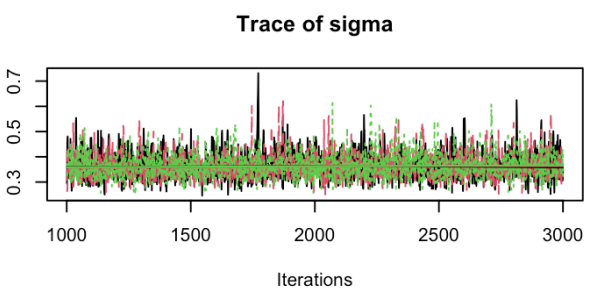
- Recall our ground truth: `mu = 1.12`, `sigma = 0.38`. What will you conclude about the result from JAGS.

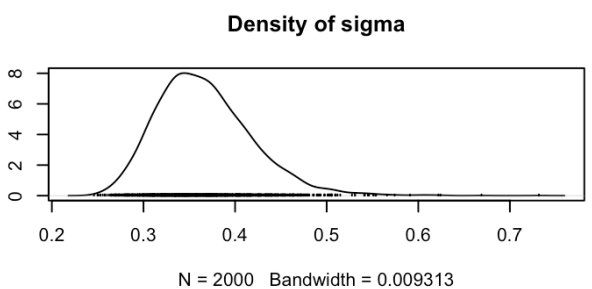
Example 1 (Cont'd) - Trace Plot

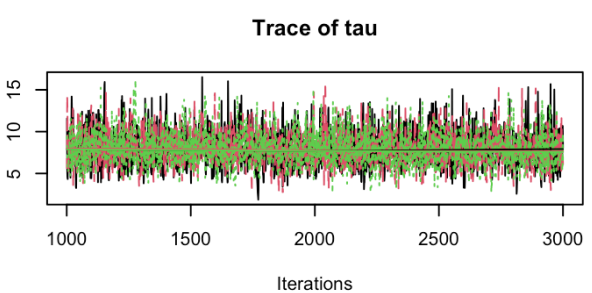
```
1 # Plot MCMC chains
2 plot(samples)
```

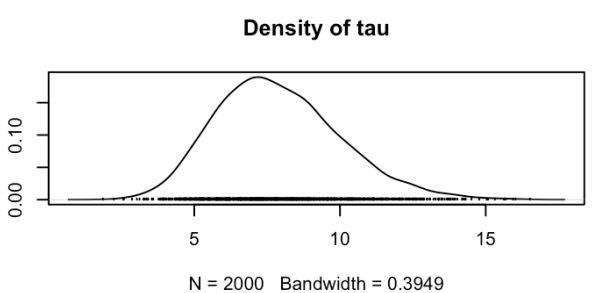












Example 1 (Cont'd) - Autocorrelation Plot & Effective Sample Size

```
1 autocorr.diag(samples)
```

```
                  mu       sigma          tau
Lag 0   1.000000e+00  1.00000000  1.000000000
Lag 1   1.772588e-02  0.33924252  0.262195963
Lag 5   9.288274e-03  0.01052557  0.002459708
Lag 10 -3.721319e-05 -0.01503355 -0.022877238
Lag 50 -7.350899e-04 -0.03318907 -0.027529983
```

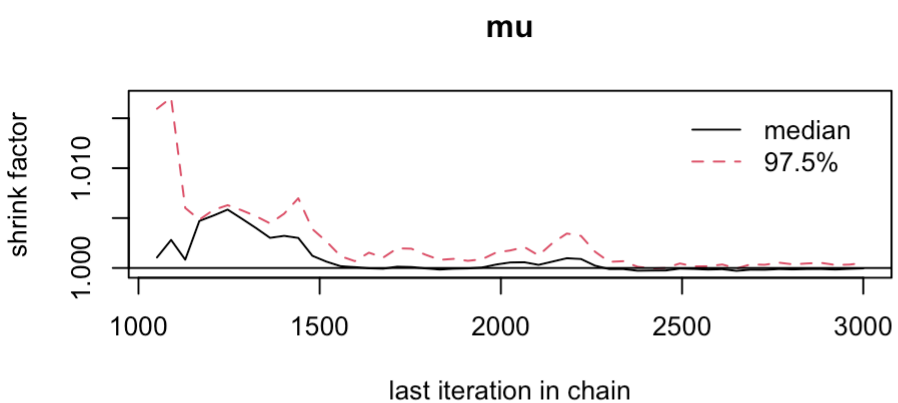
```
1 effectiveSize(samples)
```

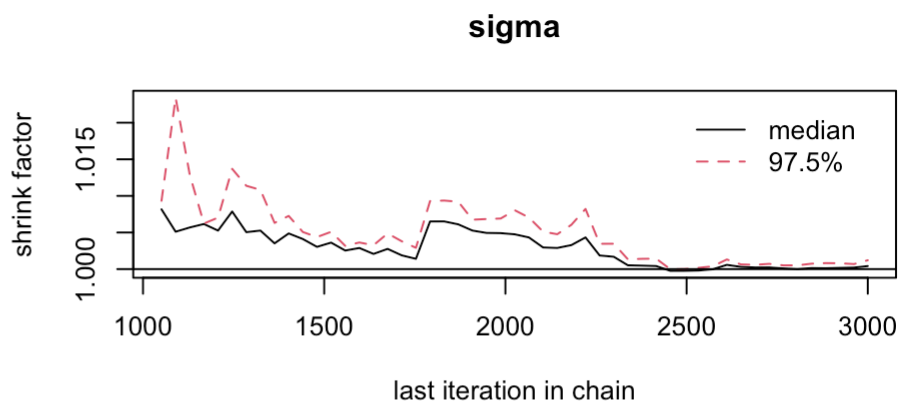
```
      mu    sigma      tau
5835.195 2712.066 3251.849
```

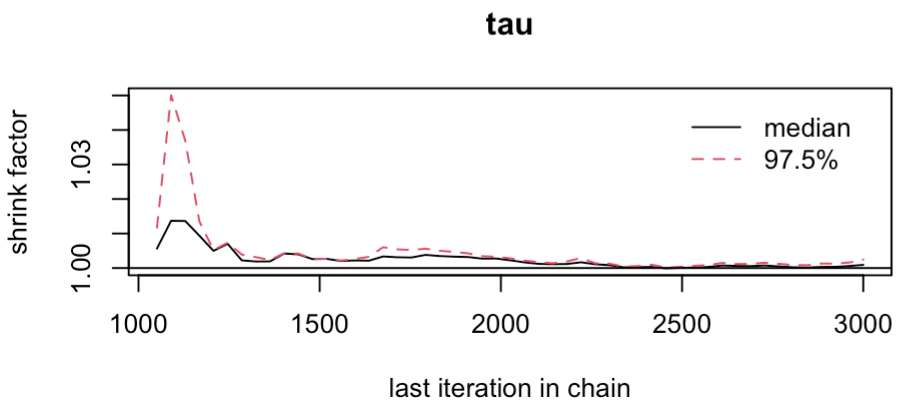
```
1 # autocorr.plot(samples)
```

Example 1 (Cont'd) - Gelman-Rubin

```
#Examine convergence of the Markov chains using the Gelman-Brooks-Rubin diagnostic
gelman.plot(samples)
```







Example 2 (Cont'd)

Simple Linear Regression: We have the data of 48 pigs with body weight measured at 9 successive weeks. Suppose we would like to fit the model: for $i = 1, \cdots, 432$

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i \quad \epsilon_i \sim N\left(0, \sigma^2\right)$$

Priors are:

$$\begin{aligned} \beta_0 &\sim N(0, 1000) \\ \beta_1 &\sim N(0, 1000) \\ \sigma^2 &\sim \text{Inv- Gamma}(0.001, 0.001) \end{aligned}$$

```r
rm(list=ls())
# Ground Truth
set.seed(425)
N <- 48
beta0_true <- 2
beta1_true <- 0.5
sigma_true <- 1
# Generate synthetic data
x <- rnorm(N, 5, 2)
y <- rnorm(N, beta0_true + beta1_true * x, sigma_true)
```

Example 2 (Cont'd)

Recall our BUGS syntax: `writeLines(model_code_2, "model_2.txt")`

```r
# Prepare data for JAGS
jags_data <- list(
  N = N,
  x = x,
  y = y
)

# Initial values for chains
inits <- function() {
  list(beta0 = rnorm(1, 0, 1),
       beta1 = rnorm(1, 0, 1),
       sigma2 = runif(1, 0, 5))
}

# Parameters to monitor
params <- c("beta0", "beta1", "sigma2", "tau2")

# Load rjags and run the model
library(rjags)
model_2 <- jags.model("model_2.txt",
                      data = jags_data,
                      inits = inits,
                      n.chains = 3)
```

```
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 48
   Unobserved stochastic nodes: 3
```

```
   Total graph size: 202

Initializing model
```

Example 2 (Cont'd)

```r
samples_2 <- coda.samples(model_2, variable.names = params, n.iter = 3000, n.burnin = 1000)

# Convert to matrix for easier manipulation
output_2 <- as.matrix(samples_2)

# Extract results
beta0_samples <- output_2[, "beta0"]
beta1_samples <- output_2[, "beta1"]
sigma_samples_2 <- output_2[,"sigma2"]
tau_samples_2 <- output_2[,"tau2"]

# Summary statistics
# posterior prediction
mean(beta0_samples)
```

```
[1] 1.978377
```

```r
mean(beta1_samples)
```

```
[1] 0.51849
```

```r
mean(sigma_samples_2)
```

```
[1] 0.9782063
```

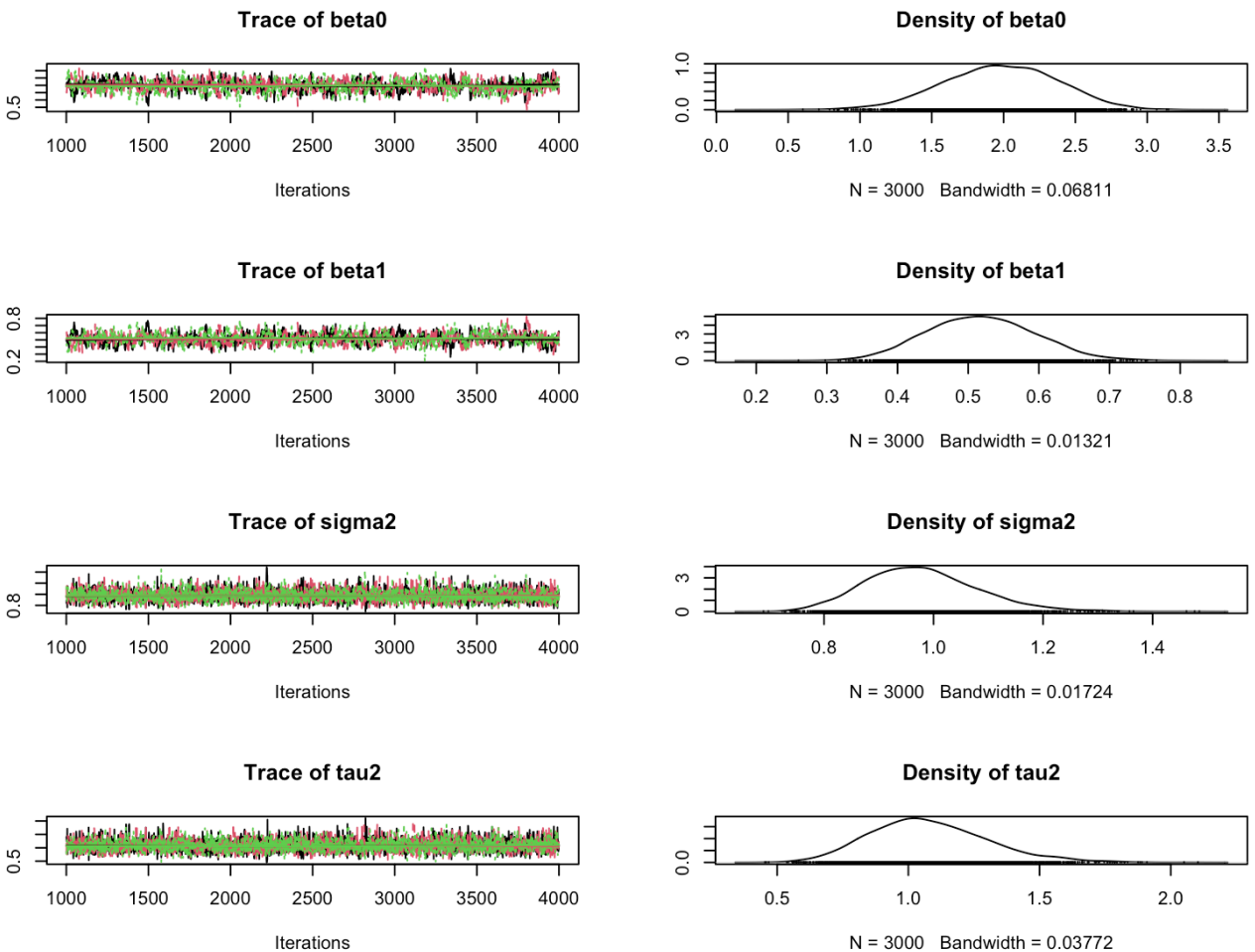
```r
mean(tau_samples_2)
```

```
[1] 1.079328
```

- Recall our ground truth: `beta0_true = 2`, `beta1_true = 0.5`, `sigma_true=1`. What will you conclude about the result from JAGS.

Example 2 (Cont'd) - Trace Plot

```
1 # Plot MCMC chains
2 plot(samples_2)
```



RICE

Example 2 (Cont'd) - Autocorrelation Plot & Effective Sample Size

```
1 autocorr.diag(samples_2)
```

```
             beta0      beta1        sigma2         tau2
Lag 0  1.00000000 1.00000000  1.0000000000  1.000000000
Lag 1  0.87054836 0.87023150  0.2882158506  0.254044303
Lag 5  0.49285648 0.49501826  0.0233869895  0.022817377
Lag 10 0.22422971 0.22620577  0.0002910298 -0.002729308
Lag 50 0.02637208 0.02833683 -0.0314088728 -0.030339463
```

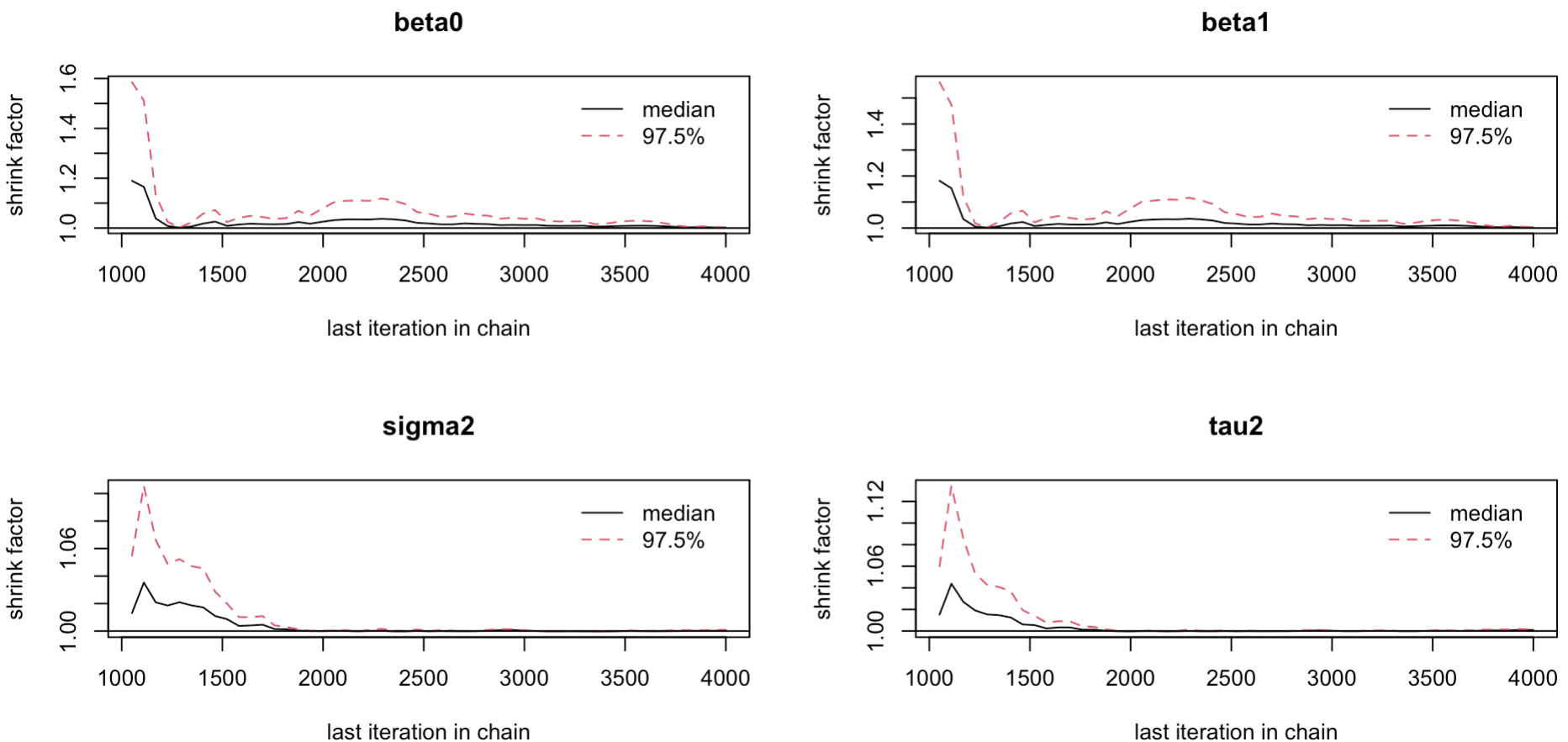
```
1 effectiveSize(samples_2)
```

```
    beta0     beta1    sigma2      tau2
629.0657  624.3691 4972.6959 5351.8985
```

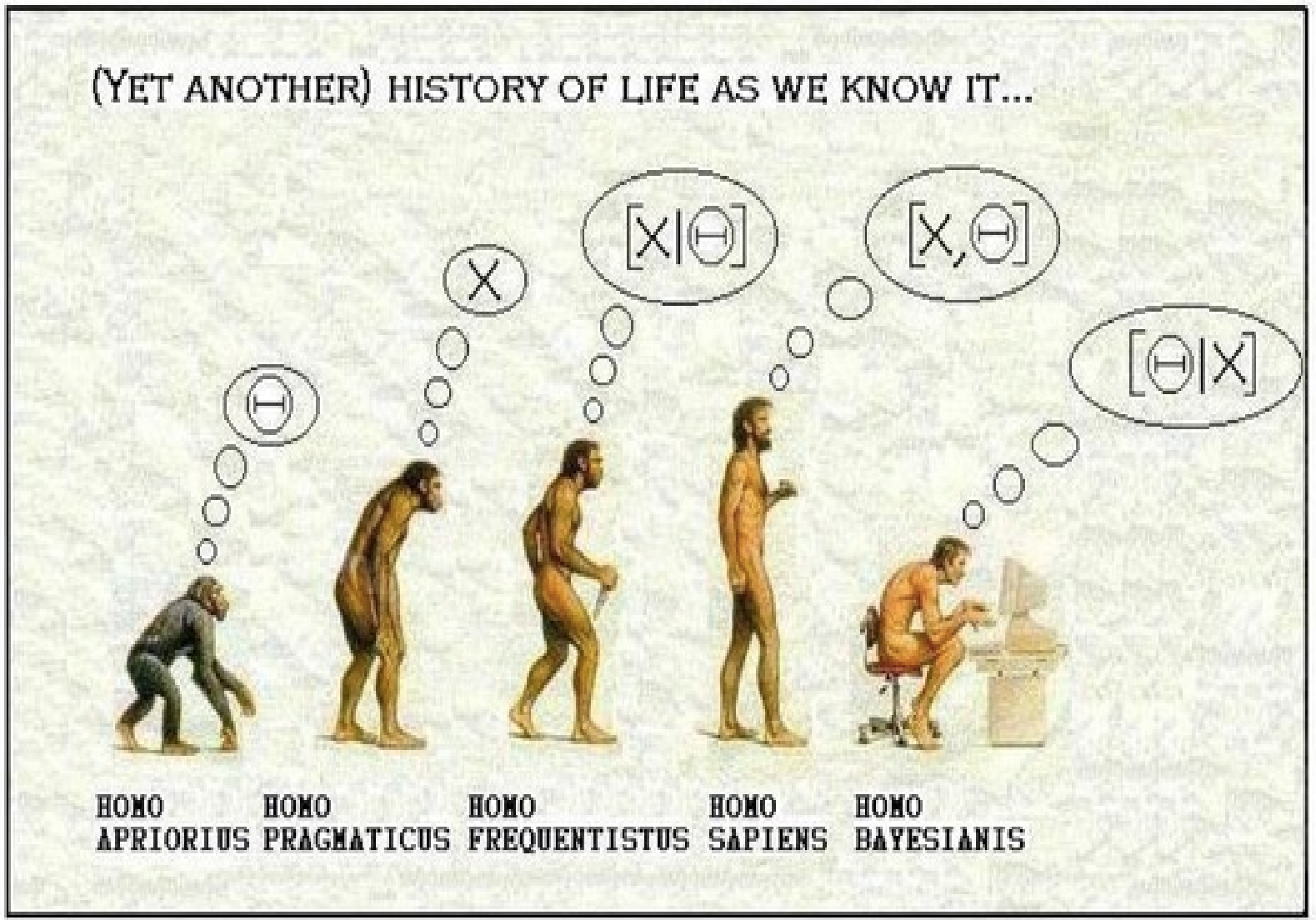
```
1 # autocorr.plot(samples)
```

Example 2 (Cont'd) - Gelman-Rubin

```
#Examine convergence of the Markov chains using the Gelman-Brooks-Rubin diagnostic
gelman.plot(samples_2)
```



Finally



RICE

Resources

- JAGS (official site)
- *JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling* by Martyn Plummer — University of Warwick
- Rebecca Steorts, *Introduction to Just Another Gibbs Sampler (JAGS)* — Duke University
- *An Introduction to Bayesian Reasoning and Methods* by Kevin Davis
- *A First Course in Bayesian Statistical Methods* by Dr. Peter Hoff
- Ritvik Kharkar
- Original slide materials were prepared by Dr. Scott Liang and Enyu Li.
- Updated and expanded under the supervision of Dr. Marina Vannucci.

Example 3: Random-Intercept Model in Action (Optional practice)

Same pig dataset, assume intercept is different for each pig Data likelihood: for $i = 1, \cdots, 48, j = 1, \cdots, 9$

$$y_{ij} = \beta_0 + \theta_i + \beta_1 x_{ij} + \varepsilon_{ij}, \quad \theta_i \sim N\left(0, \tau^2\right), \quad \varepsilon_{ij} \sim N\left(0, \sigma^2\right)$$

$$\beta_0 \sim N(0, 1000), \quad \beta_1 \sim N(0, 1000)$$

$$\tau^2 \sim \text{Inv-Gamma}(0.001, 0.001), \quad \sigma^2 \sim \text{Inv-Gamma}(0.001, 0.001)$$

```
# Model specification
model_code <-
  "
  model {
    for (i in 1:N_pig) {
      for (j in 1:N_obs) {
        y[i, j] ~ dnorm(mu[i, j], tau_eps)
        mu[i, j] <- beta0 + theta[i] + beta1 * x[i, j]
      }
      theta[i] ~ dnorm(0, tau_theta)
    }
    beta0 ~ dnorm(0, 0.001)
    beta1 ~ dnorm(0, 0.001)
    sigma ~ dunif(0, 100)
    tau_eps <- pow(sigma, -2)
    sigma_theta ~ dunif(0, 100)
    tau_theta <- pow(sigma_theta, -2)
  }
  "

# Write model to file
writeLines(model_code, "model.txt")
```

Note

Feel free to email me at jk201@rice.edu if you want to check your answer or go over the solution together.

Thank you!

Have a great rest of your day!