MCMC software

Stat 340: Bayesian Statistics

Implementing MCMC in JAGS

(Problem topic 9)

Example

- Researchers are interested in modeling the survival times, measured in weeks, of patients who were diagnosed with leukemia.
- The patients were classified according to two characteristics of white bloods cells.
- The sample consists of $n = 17$ times in weeks from diagnosis to death

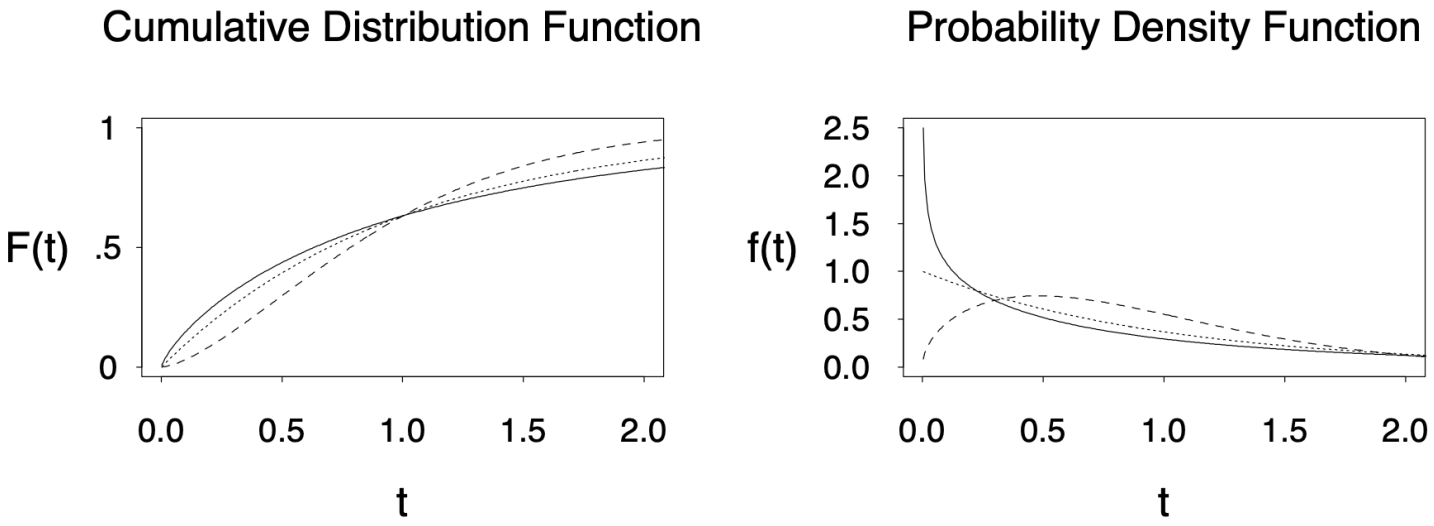
```
survival_times <- c(65, 156, 100, 134, 16, 108, 121, 4,
                    39, 143, 56, 26, 22, 1, 1, 5, 65)
```

Weibull distribution

Weibull distribution is often used as a model for survival times in biomedical, demographic, and engineering analyses

PDF: $f(y|v,\lambda) = \lambda\alpha y^{\alpha-1}e^{-\lambda y^{\alpha}}, \quad y,\alpha,\lambda > 0$

CDF: $F(y|v,\lambda) = 1 - e^{-\lambda y^{\alpha}}$



Proposed model

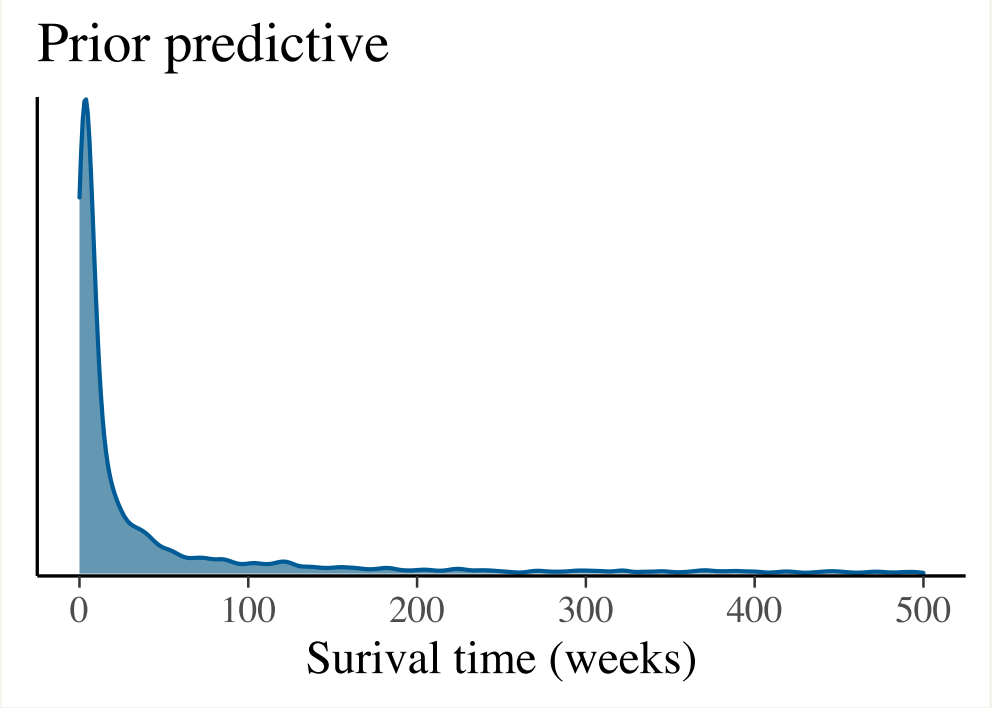
Let $Y_i =$ survival time in weeks

$$Y_i | \alpha, \lambda \overset{\text{iid}}{\sim} \text{Weibull}(\alpha, \lambda)$$

$$\alpha \sim \text{Gamma}(1, 1)$$

$$\lambda \sim \text{Gamma}(1.53, 26.3)$$

$$\alpha \propto \lambda$$



Your turn

Weibull PDF: $f(y|v,\lambda) = \lambda\alpha y^{\alpha-1}e^{-\lambda y^{\alpha}}$

Gamma PDF: $f(x|a,b) = \frac{b^a}{\Gamma(a)}x^{a-1}e^{-bx}$

1. Derive the joint posterior distribution for] α and λ
2. Derive the two conditional posterior distributions for this model
3. Are either distributions that you recognize?
4. If you get done, outline the steps of a Gibbs sampler for this posterior

05:00

Why Just Another Gibbs Sampler (JAGS)?

- You can fit virtually any model
- You can call JAGS from R, allowing for plotting and data manipulation in R
- It runs on all platforms: LINUX, Mac, Windows
- There is a lot of help online
- R has many built in packages for convergence diagnostics

Slide adapted from Brian Reich

How does JAGS work?

- You specify the model by declaring the likelihood and priors
- JAGS then sets up the MCMC sampler
  - e.g., works out the full conditional distributions for all parameters
- It returns MCMC samples in a matrix or array
- It also automatically produces posterior summaries like means, credible sets, and convergence diagnostics
- User's manual: http://people.stat.sc.edu/hansont/stat740/jags_user_manual.pdf

Slide adapted from Brian Reich

Steps for any analysis using JAGS

1. Specify the model as a string (or script)
2. Define/load the data
3. Define initial values
4. Draw posterior samples using `run.jags()`
5. Inspect/summarize the results

0. Load `runjags`

```
library(runjags)
```

1. Specify the model as a string

```
model_string <- "
model{
  # Specify the likelihood
  for(i in 1:n) {
    y[i] ~ dweib(alpha, lambda)
  }

  # Specify the priors
  alpha ~ dgamma(1, 1)
  lambda ~ dgamma(1.53, 26.3)
}
"
```

JAGS syntax

JAGS doesn't always use the same syntax as R.

But, uses the same d*, q*, and p* prefixes

Distribution	Distribution	Quantile	
Bernoulli	dbern	pbern	qbern
Beta	dbeta	pbeta	qbeta
Binomial	dbin	pbin	qbin
Chi-squared	dchisqr	pchisqr	qchisqr
Double exponential	ddexp	pdexp	qdexp
Exponential	dexp	pexp	qexp
F	df	pf	qf
Gamma	dgamma	pgamma	qgamma
Generalized gamma	dgen.gamma	pgen.gamma	qgen.gamma
Noncentral hypergeometric	dhyper	phyper	qhyper
Logistic	dlogis	plogis	qlogis
Log-normal	dlnorm	plnorm	qlnorm
Negative binomial	dnegbin	pnegbin	qnegbin
Noncentral Chi-squared	dnchisqr	pnchisqr	qnchisqr
Normal	dnorm	pnorm	qnorm
Pareto	dpar	ppar	qpar
Poisson	dpois	ppois	qpois
Student t	dt	pt	qt
Weibull	dweib	pweib	qweib

Image source: JAGS manual

2. Define/load the data

Loading the survival data

```
survival_times <- c(65, 156, 100, 134, 16, 108, 121, 4,
                    39, 143, 56, 26, 22, 1, 1, 5, 65)
```

Store your data in a list

```
survival_data <- list(
  y = survival_times,
  n = length(survival_times)
)
```

3. Define initial values

- If you don't specify initial values for your parameters, then JAGS will
- If not specified, then JAGS will use the mean or mode of the prior distribution
- More on this next time...

4. Draw posterior samples using `run.jags()`

- `model` string specifying the model

```
posterior <- run.jags(
  model = model_string,
  n.chains = 1,
  data = survival_data,
  monitor = c("alpha", "lambda"
  adapt = 1000,
  burnin = 5000,
  sample = 5000
)
```

4. Draw posterior samples using `run.jags()`

- `model` string specifying the model
- `n.chains` the number of chains to run

```
posterior <- run.jags(
  model = model_string,
  n.chains = 1,
  data = survival_data,
  monitor = c("alpha", "lambda"
  adapt = 1000,
  burnin = 5000,
  sample = 5000
)
```

4. Draw posterior samples using `run.jags()`

- `model` string specifying the model
- `n.chains` the number of chains to run
- `data` a named list or data frame include the data and prior parameter values

```
posterior <- run.jags(
  model = model_string,
  n.chains = 1,
  data = survival_data,
  monitor = c("alpha", "lambda"
  adapt = 1000,
  burnin = 5000,
  sample = 5000
)
```

4. Draw posterior samples using `run.jags()`

- `model` string specifying the model
- `n.chains` the number of chains to run
- `data` a named list or data frame include the data and prior parameter values
- `monitor` character vector of the names of variables to monitor

```
posterior <- run.jags(
  model = model_string,
  n.chains = 1,
  data = survival_data,
  monitor = c("alpha", "lambda"
  adapt = 1000,
  burnin = 5000,
  sample = 5000
)
```

4. Draw posterior samples using `run.jags()`

- `model` string specifying the model
- `n.chains` the number of chains to run
- `data` a named list or data frame include the data and prior parameter values
- `monitor` character vector of the names of variables to monitor
- `adapt` number of samples drawn during initial sampling/adaptation phase

```
posterior <- run.jags(
  model = model_string,
  n.chains = 1,
  data = survival_data,
  monitor = c("alpha", "lambda"
  adapt = 1000,
  burnin = 5000,
  sample = 5000
)
```

4. Draw posterior samples using `run.jags()`

- `model` string specifying the model
- `n.chains` the number of chains to run
- `data` a named list or data frame include the data and prior parameter values
- `monitor` character vector of the names of variables to monitor
- `adapt` number of samples drawn during initial sampling/adaptation phase
- `burnin` number of burn-in iterations, NOT including the adaptive iterations

```
posterior <- run.jags(
  model = model_string,
  n.chains = 1,
  data = survival_data,
  monitor = c("alpha", "lambda"
  adapt = 1000,
  burnin = 5000,
  sample = 5000
)
```

4. Draw posterior samples using `run.jags()`

- `model` string specifying the model
- `n.chains` the number of chains to run
- `data` a named list or data frame include the data and prior parameter values
- `monitor` character vector of the names of variables to monitor
- `adapt` number of samples drawn during initial sampling/adaptation phase
- `burnin` number of burn-in iterations, NOT including the adaptive iterations
- `sample` the total number of (additional) samples to draw

```
posterior <- run.jags(
  model = model_string,
  n.chains = 1,
  data = survival_data,
  monitor = c("alpha", "lambda"
  adapt = 1000,
  burnin = 5000,
  sample = 5000
)
```

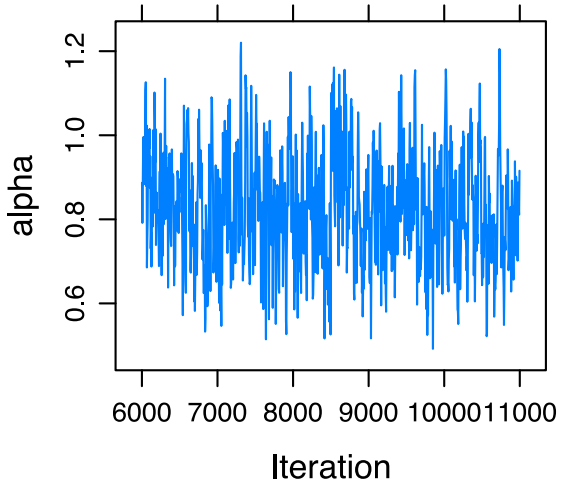
In an .Rmd file, add `silent.jags = TRUE` to avoid this issue...

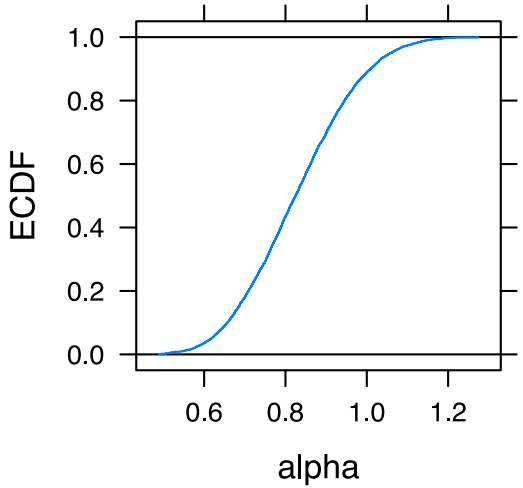
```
## Compiling rjags model...
## Calling the simulation using the rjags method...
## Adapting the model for 1000 iterations...
## Burning in the model for 5000 iterations...
## Running the model for 5000 iterations...
## Simulation complete
## Calculating summary statistics...
```

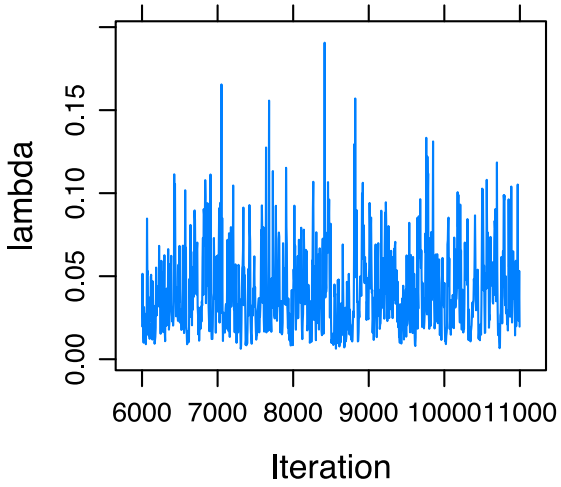
```
## Warning: Convergence cannot be assessed with only 1 chain
```

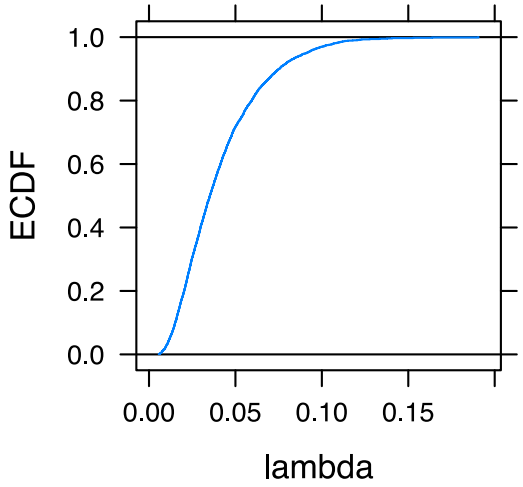
```
## Finished running the simulation
```

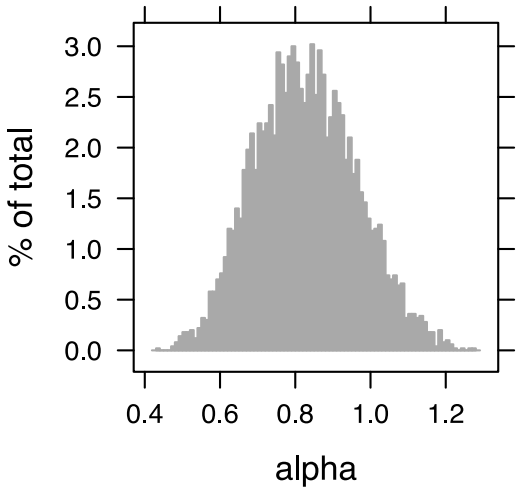
5. Inspect the results

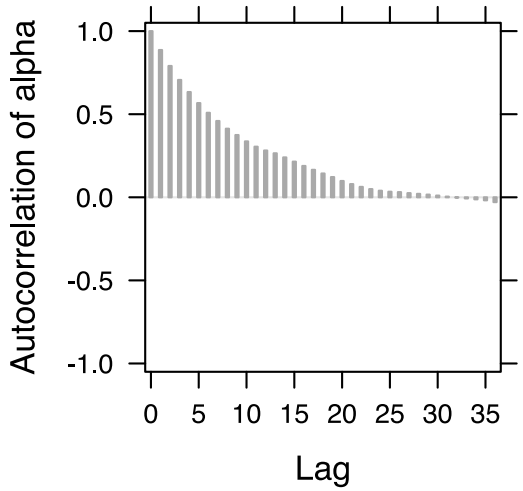


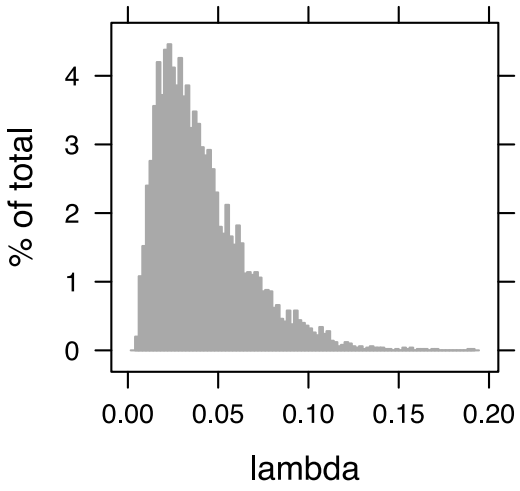


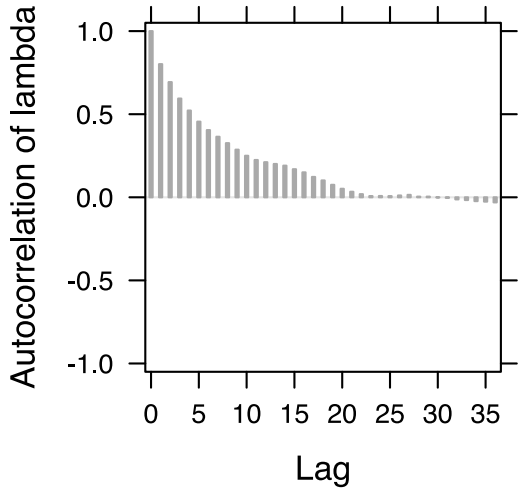












`runjags` objects are big lists

```
names(posterior)
```

```
##  [1] "mcmc"               "deviance.table"     "deviance.sum"
##  [4] "pd"                 "end.state"          "samplers"
##  [7] "burnin"             "sample"             "thin"
## [10] "model"              "data"               "monitor"
## [13] "noread.monitor"     "modules"            "factories"
## [16] "response"           "residual"           "fitted"
## [19] "method"             "method.options"     "timetaken"
## [22] "runjags.version"    "summaries"          "summary"
## [25] "HPD"                "hpd"                "mcse"
## [28] "psrf"               "autocorr"           "crosscorr"
## [31] "truestochastic"     "semistochastic"     "nonstochastic"
## [34] "discrete"           "trace"              "density"
## [37] "histogram"          "ecdfplot"           "key"
## [40] "acplot"             "ccplot"             "summary.available"
## [43] "summary.pars"       "dic"
```

mcmc element is a list of the actual draws

```
head(posterior$mcmc)
```

```
## [[1]]
## Markov Chain Monte Carlo (MCMC) output:
## Start = 6001
## End = 6007
## Thinning interval = 1
##          alpha     lambda
## 6001 0.8868662 0.01977346
## 6002 0.9252918 0.01395950
## 6003 1.0516895 0.01721196
## 6004 0.9212670 0.02472420
## 6005 0.9250907 0.02964864
## 6006 0.7915493 0.05146866
## 6007 0.7406706 0.05755608
##
## attr(,"class")
## [1] "mcmc.list"
```

5. Summarize the results

```
summary(posterior$mcmc[[1]])
##
## Iterations = 6001:11000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean      SD  Naive SE Time-series SE
## alpha  0.82868 0.13494 0.0019084       0.007950
## lambda 0.04096 0.02484 0.0003512       0.001279
##
## 2. Quantiles for each variable:
##
##           2.5%     25%     50%    75%  97.5%
## alpha  0.58207 0.73092 0.82469 0.9219 1.1017
## lambda 0.00974 0.02251 0.03521 0.0537 0.1027
```

bayesplot

- provides a variety of `ggplot2`-based plotting functions for use after fitting Bayesian models
- Visualizations of MCMC simulations and diagnostics *from any algorithm* (`mcmc_*`)
- Graphical prior and posterior predictive checks (`ppc_*`)

Convert to an `mcmc` object

`bayesplot` requires objects to be of class `mcmc`

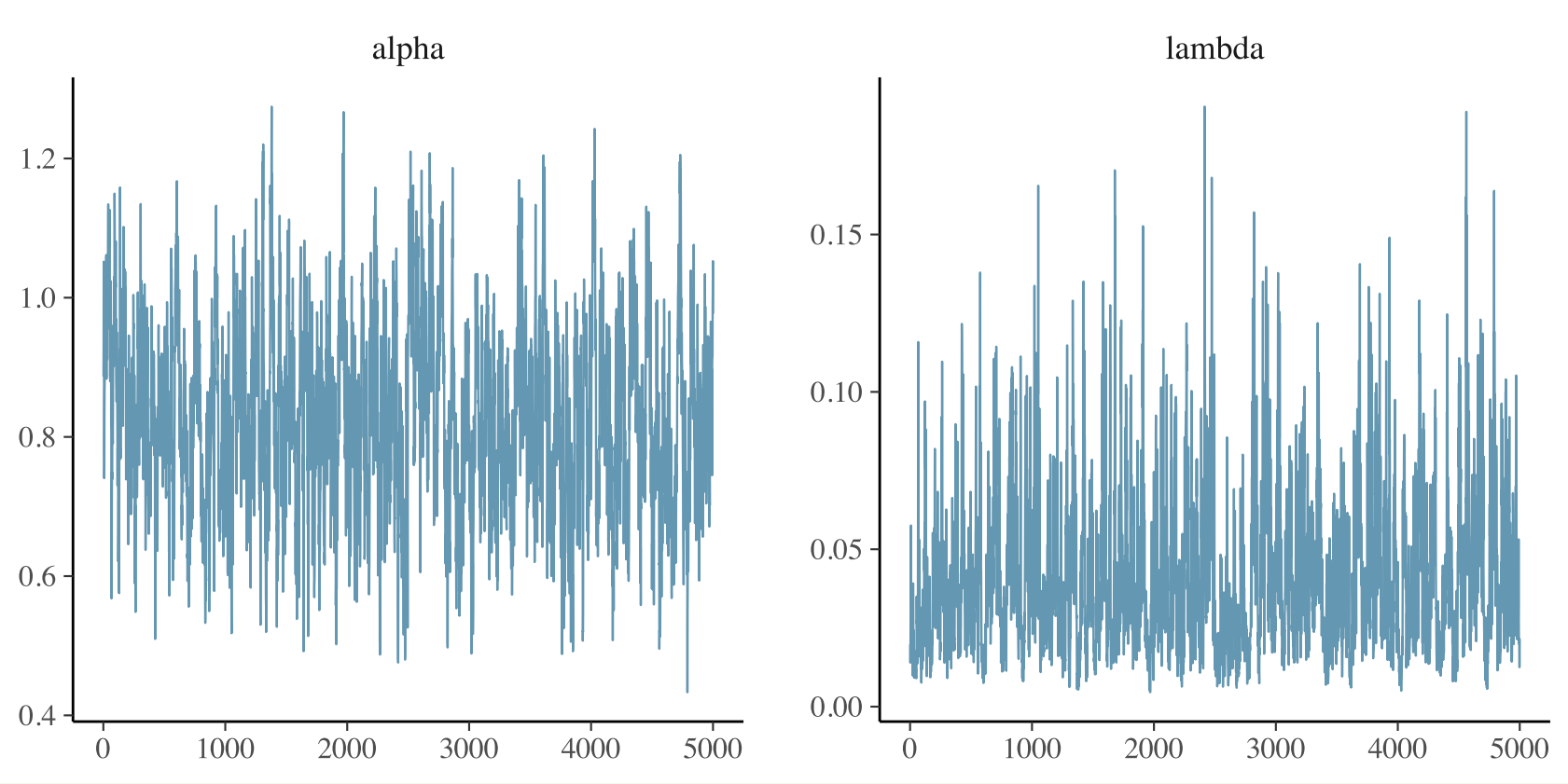
You'll need to convert your output from `run.jags()`

```
class(posterior)
## [1] "runjags"
```

```
post_mcmc <- as.mcmc(posterior)
class(post_mcmc)
## [1] "mcmc"
```

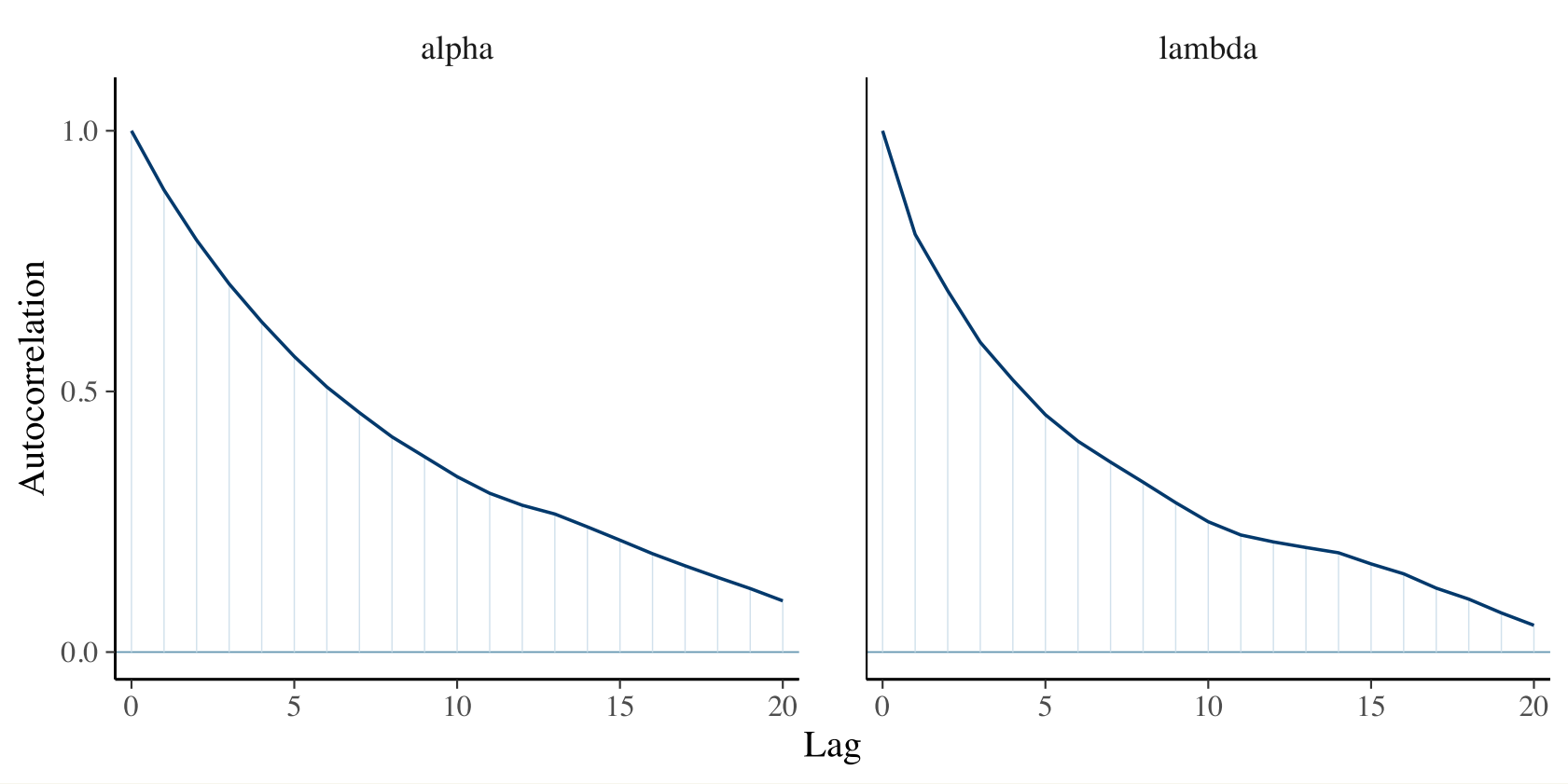
Trace plot

```
mcmc_trace(post_mcmc)
```



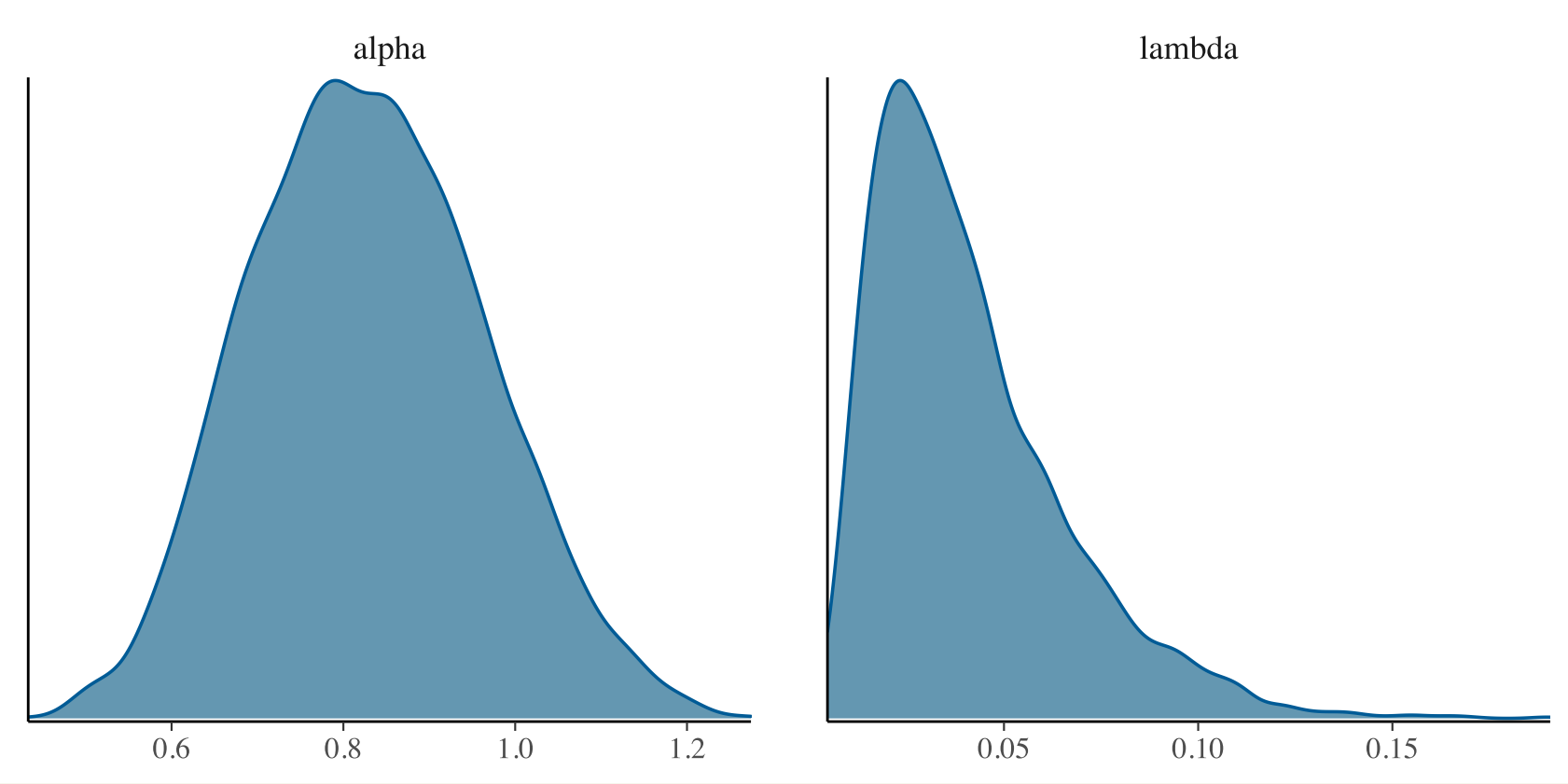
ACF plot

```
mcmc_acf(post_mcmc)
```



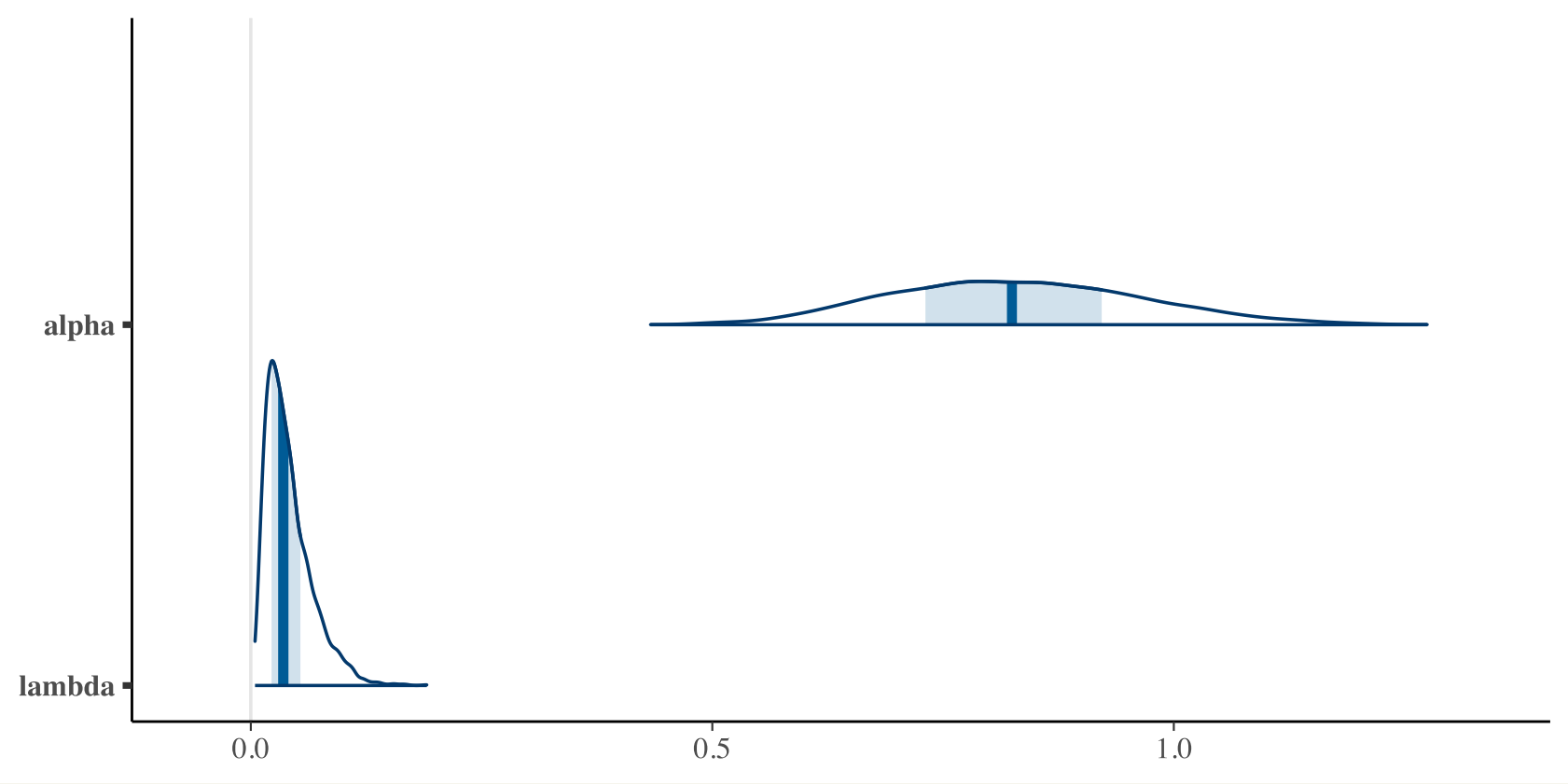
Posterior density plot

```
mcmc_dens(post_mcmc)
```



Plot equal-tail interval estimates

```
mcmc_areas(post_mcmc)
```



Your turn

The median survival time can be calculated as

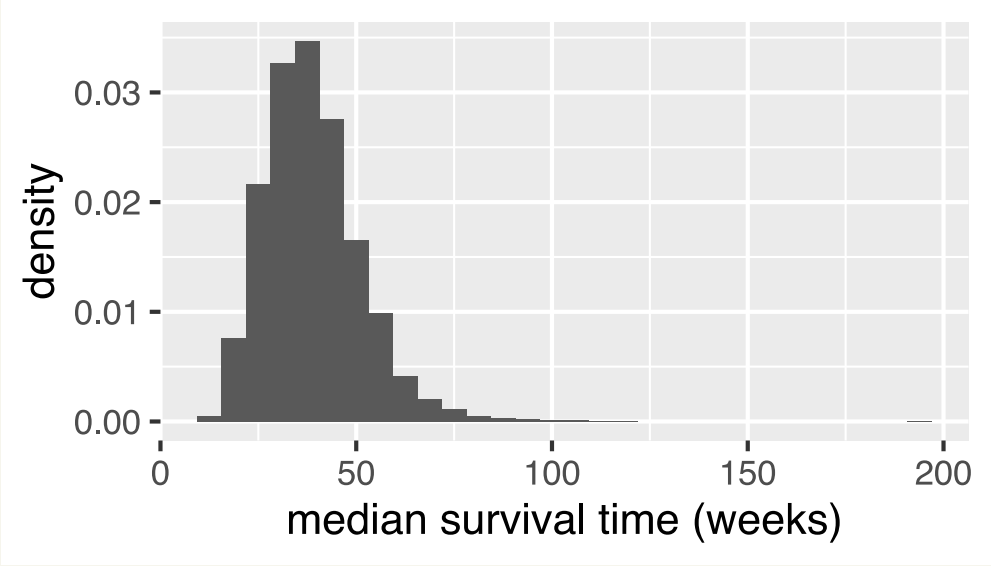
$$\left(\frac{\log(2)}{\lambda}\right)^{1/\alpha}$$

How could you calculate a 90% credible interval for the median survival time?

01:30

Posterior inference for `runjags` objects

```
median_survive <- (log(2) / posterior$mcmc[[1]][,"lambda"])^(1 / posterior$mcmc[[1]][,"alpha"])
```



90% credible interval for the median survival time

```
quantile(median_survive,
         probs = c(0.05, 0.95))
##       5%      95%
## 21.82081 60.24946
```

Monitoring functions of parameters

You can create objects in JAGS using `<-`

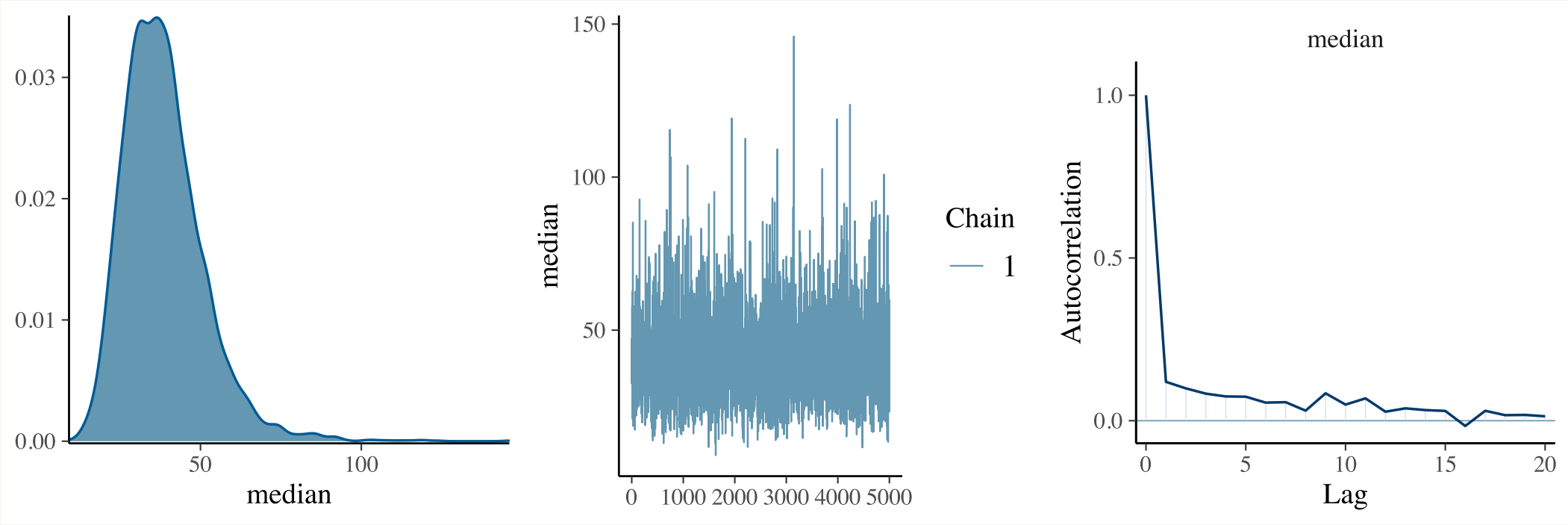
```
model_string2 <- "
model{
  # Specify the likelihood
  for(i in 1:n) {
    y[i] ~ dweib(alpha, lambda)
  }

  # Specify the priors
  alpha ~ dgamma(1, 1)
  lambda ~ dgamma(1.53, 26.3)

  # Calculate median
  median <- (log(2) / lambda)^(1 / alpha)
}
"
```

Monitoring functions of parameters

You can monitor objects you create

```
posterior2 <- run.jags(
  model = model_string2,
  n.chains = 1,
  data = survival_data,
  monitor = "median",
  adapt = 1000,
  burnin = 5000,
  sample = 5000
)
```

Did the chain converge?



Your turn

Work through the example on the JAGS handout with a neighbor.

Check the website for a .Rmd template