BAYESIAN DATA ANALYSIS

Notes by Professor Athar Ali Khan, AMU, Aligarh

MOHAMMAD WASIQ

Bayesian Data Analysis

Mohammad Wasiq

2022-05-10

Table of Contents

1 Fundamentals of Bayesian Data Analysis

1.1 The three steps of Bayesian data analysis

The process of Bayesian data analysis can be idealized by dividing it into the following three steps:

1. *Setting up a full probability model*— A joint probability distribution for all observable and unobservable quantities in a problem. The model should be consistent with knowledge about the underlying scientific problem and the data collection process.

2. *Conditioning on observed data :* Calculating and interpreting the appropriate posterior distribution—the conditional probability distribution of the unobserved quantities of ultimate interest, given the observed data.

3. *Evaluating the fit of the model and the implications of the resulting posterior distribution :* how well does the model fit the data, are the substantive conclusions reasonable, and how sensitive are the results to the modeling assumptions in step 1 ? In response, one can alter or expand the model and repeat the three steps.

A primary motivation for Bayesian thinking is that it facilitates a common-sense interpretation of statistical conclusions. For instance, a Bayesian (probability) interval for an unknown quantity of interest can be directly regarded as having a high probability of containing the unknown quantity, in contrast to a frequentist (confidence) interval, which may strictly be interpreted only in relation to a sequence of similar inferences that might be made in repeated practice.

1.2 Bayes' Rule

$$p(\theta, y) = p(\theta)p(y|\theta)$$

$$p(y, \theta) = p(y)p(\theta|y)$$

$$p(\theta)p(y|\theta) = p(y)p(\theta|y)$$

$$p(\theta|y) = \frac{p(\theta)p(y|\theta)}{p(y)}$$

$$p(\theta|y) \propto p(\theta)p(y|\theta)$$

$$Posterior \propto Prior \times Likelihood$$

this is known as *Bayes' rule*. Once we have constructed posterior density every important aspect of Bayesian analysis is done.

Prediction

Prior Prediction :

$$p(y) = \int p(\theta, y)d\theta = \int p(\theta)p(y|\theta)d\theta$$

This is known as prior predictive distribution as this is prior to data y has been observed.

Posterior Predictive Distribution :

After data y has been observed, we can predict an unknown observable, $\tilde{y}$, from the same process. The distribution of $\tilde{y}$ is called posterior predictive distribution, posterior because it is conditional on observed data y and predictive because it is a prediction for an observable $\tilde{y}$.

$$p(\tilde{y}|y) = \int p(\tilde{y}, \theta|y)d\theta = \int p(\tilde{y}|\theta, y)p(\theta|y)d\theta$$

$$p(\tilde{y}|y) = \int p(\tilde{y}|\theta)p(\theta|y)d\theta$$

The first and the second line display the posterior predictive distribution as an average of conditional prediction over the posterior distribution of θ.

Simulation

Step 1. Simulate θ from posterior density $p(\theta|y)$.

Step 2. Use these simulated values of θ into the conditional posterior $p(\tilde{y}|\theta)$ and finally simulate $\tilde{y}$ from $p(\tilde{y}|\theta)$, thus without integrating the expression we get simulation from $p(\tilde{y}|y)$.

2 Single Parameter Model

If $y_1, y_2, \cdots, y_i \sim iid\ B(1, p)$ then $\sum y_i \sim B(n, p)$

$$p(y|\theta) = Bin(y|n,\theta) = \binom{n}{y}\theta^y(1-\theta)^{n-y}$$

If we assume $p(\theta|y), \theta \in [0,1]$ then

$$p(\theta|y) \propto p(\theta)p(y|\theta)$$

$$p(\theta|y) \propto \theta^y(1-\theta)^{n-y}$$

$$p(\theta|y) \propto \theta^{y+1-1}(1-\theta)^{n-y+1-1}$$

$$\theta|y \sim Beta(y+1, n-y+1)$$

Informative Prior for $\theta$$

Let $p(y|\theta) \propto \theta^y(1-\theta)^{n-y}$

and prior

$$p(y) \propto \theta^{\alpha-1}(1-\theta)^{\beta-1}$$

$$p(\theta|y) \propto p(\theta)p(y|\theta)$$

$$= \theta^{\alpha-1}(1-\theta)^{\beta-1}\theta^y(1-\theta)^{n-y}$$

$$= \theta^{y+\alpha-1}(1-\theta)^{n-y+\beta-1}$$

$$= Beta(\theta|y+\alpha, n-y+\beta)$$

$$\theta|y \sim Beta(y+\alpha, n-y+\beta)$$

- Since Prior and Posterior belongs to the same family it is terms as conjugacy i.e. conjugate prior for binomial parameter is beta distribution.
- using properties of beta distribution we can get $E(\theta|y) = \frac{\alpha+y}{\alpha+\beta+n}$ and posterior variance $Var(\theta|y) = \frac{(\alpha+y)(\beta+n-y)}{(\alpha+\beta+n)^2(\alpha+\beta+n+1)}$ or $Var(\theta|y) = \frac{E(\theta|y)[1-E(\theta|y)]}{\alpha+\beta+n+1}$
- AS y and $n-y$ becomes large with fixed α and β, then $E(\theta|y) \approx \frac{y}{n}$ and $Var(\theta|y) \approx \frac{1}{n}\frac{y}{n}\left(1-\frac{y}{n}\right)$.

Thus in the limit, the parameters of the prior distribution have no influence on posterior.

- In fact, we will see that $p(\theta|y)$ can be $p(\theta|y) \sim N[E(\theta|y), Var(\theta|y)$ as $n \to \infty$
- In fact, for binomial parameter parameter θ, normal approximation is more accurate approximation in practice, if we transform θ to logit scale, i.e. performing inference for $log\left(\frac{\theta}{1-\theta}\right)$ instead of θ itself. Thus expanding the probability space from [0,1] to $[-\infty, \infty]$, which is more fitting for a normal approximation.

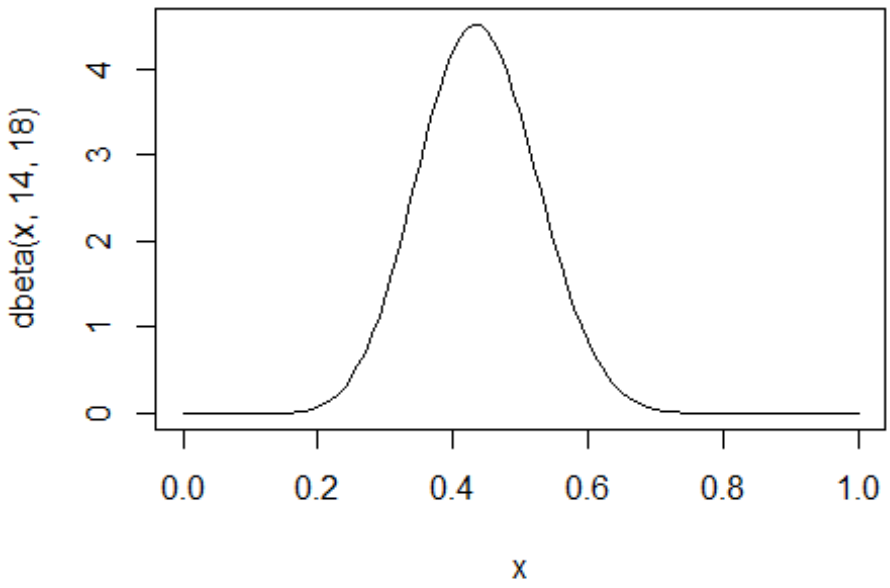
2.1 Numerical Illustration

Suppose out of n=30, y=13 successes we are obtained. Find out bayesian analysis of proportion of successes.

Solution

We know that if prior $p(\theta) = 1$ this implies $\alpha = 1, \beta = 1$ hence posterior $p(\theta|y) \propto Beta(\theta|\alpha + y, \beta + n - y) = Beta(\theta|1 + 13, 1 + 30 - 13) = Beta(\theta|14,18)$

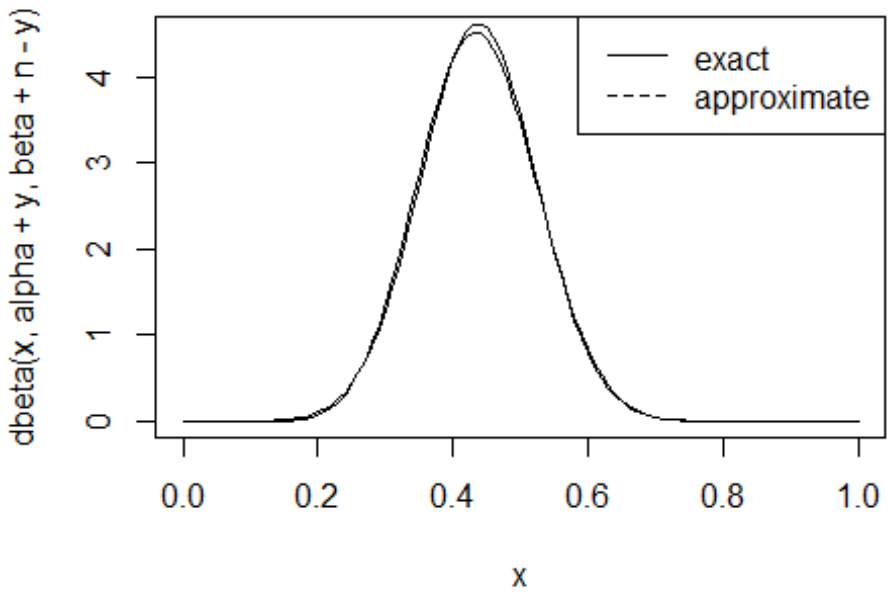
```
curve(dbeta(x, 14 , 18), from = 0 , to = 1)
```



This will represent exact distribution whose mean and variance can be express as

```
alpha <- 1
beta <- 1
n <- 30
y <- 13
curve(dbeta(x, alpha+y, beta+n-y), from = 0, to = 1)

# Compute Mean and Variance and sd
mean_theta<- (alpha+y)/(alpha+beta+n)
mean_theta

## [1] 0.4375

var_theta<- mean_theta*(1-mean_theta)/(alpha+beta+n+1)
var_theta
```

```
## [1] 0.007457386

sd_theta <- sqrt(var_theta)
sd_theta

## [1] 0.08635616

curve(dnorm(x, mean = mean_theta, sd=sd_theta), add = TRUE)
legend("topright", c("exact", "approximate"), lty = c(1, 2))
```



Approximate and exact posterior of binomial probability is so close that it is very difficult to distinguish them.

2.2 How to make use of simulation

```
theta_sim_30<- rbeta(30, alpha+y, beta+n-y)

theta_sim_100<- rbeta(100, alpha+y, beta+n-y)

theta_sim_1000<- rbeta(1000, alpha+y, beta+n-y)

mean_sim_30<- mean(theta_sim_30)

mean_sim_100<- mean(theta_sim_100)

mean_sim_1000<- mean(theta_sim_1000)

sd_sim_30<- sd(theta_sim_30)
```

```
sd_sim_100<- sd(theta_sim_100)

sd_sim_1000<- sd(theta_sim_1000)

mean_sim_30

## [1] 0.4479218

mean_sim_100

## [1] 0.4155181

mean_sim_1000

## [1] 0.4349188

sd_sim_30

## [1] 0.09686458

sd_sim_100

## [1] 0.08685048

sd_sim_1000

## [1] 0.084279
```

Thus we find that as sample size becomes large the posterior density approximate well.

2.3 Introduction to Multi-Parameter Models :

Virtually every practical problem in statistics involves more than one unknown one unobservable quantity.

$$p(y|\mu,\sigma^2), \quad \because \theta = (\mu,\sigma^2)$$

It is in dealing with such problems that the simple conceptual framework of bayesian approach reveals its principal advantage over other methods of inference. Although a problem can improve several parameters of interest conclusions will often be drawn about one or only a few parameters at a time. In this case, the ultimate aim of a bayesian analysis is to obtain the marginal posterior distribution of the parameters of interest. In principle, the routine to achieving this aim is clear : we first require the joint posterior distribution of all unknowns, and then we integrate this distribution over the unknowns that are not of immediate interest to obtain the desired marginal distribution. Or equivalently, using simulation, we draw samples from the joint posterior distribution and then look at the parameters of interest and ignore the values of the other unknowns. Parameters which are not interest are termed as **nuisance** parameters.

$$p(\mu|y) = \int p(\mu,\sigma^2|y)d\sigma^2$$

$$p(\sigma^2|y) = \int p(\mu, \sigma^2|y) d\mu$$

2.4 Averaging Over Nuisance Parameter

Suppose $\theta = (\theta_1, \theta_2) = (\mu, \sigma^2)$ then if interest centers on μ, σ^2 may be treated as nuisance parameter, thus

$$p(\theta_1, \theta_2|y) \propto p(y|\theta_1, \theta_2)\ p(\theta_1, \theta_2)$$

Thus, to integrate over θ_2, we do (Marginal of θ_1)

$$p(\theta_1|y) = \int p(\theta_1, \theta_2|y) d\theta_2$$

$$p(\theta_1|y) = \int p(\theta_1|\theta_2, y)\, p(\theta_2|y) d\theta_2$$

Similarly, for Marginal of θ_2

$$p(\theta_2|y) = \int p(\theta_1, \theta_2|y) d\theta_1$$

$$p(\theta_2|y) = \int p(\theta_2|\theta_1, y)\, p(\theta_1|y) d\theta_1$$

2.5 Normal Data with non-informative prior

$$p(\mu, \sigma^2) \propto p(p|\theta) p(\theta)$$

$p(y|\theta) \sim N(\mu, \sigma^2)$ and $p(\theta) \rightarrow$ non informative

A non-informative prior is

$$p(\mu, \sigma^2) \propto (\sigma^2)^{-1} \qquad \because p(\mu) = 1\ and\ p(\sigma^2) = \frac{1}{\sigma^2}$$

The Joint posterior density

$$p(\mu, \sigma^2) \propto p(y|\mu, \sigma^2)\, p(\mu, \sigma^2)$$

$$p(\mu, \sigma^2) \propto \sigma^{-n-2} exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \mu)^2\right)$$

$$= \sigma^{-n-2} exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \bar{y} + \bar{y} - \mu)^2\right)$$

$$= \sigma^{-n-2} exp\left(-\frac{1}{2\sigma^2}\left[\sum_{i=1}^{n}(y_i - \bar{y})^2 + n(\bar{y} - \mu)^2\right]\right)$$

$$= \sigma^{-n-2} exp\left(-\frac{1}{2\sigma^2}[(n-1)s^2 + n(\bar{y} - \mu)^2]\right)$$

where, $s^2 = \frac{1}{n-1}\sum(y_i - \bar{y})^2$

The Conditional Distribution

$$(\mu|\sigma^2, y) \sim N(\bar{y}, \sigma^2/n)$$

Note that,

$$p(\mu, \sigma^2|y) \propto p(\mu|\sigma^2, y)\ p(\sigma^2|y)$$

where

$$p(\mu|\sigma^2, y) \sim N(\bar{y}, \sigma^2/n)$$

The Marginal Posterior of σ^2

The Marginal Posterior of σ^2 is

$$p(\sigma^2|y) \propto \int \sigma^{-n-2} exp\left(-\frac{1}{2\sigma^2}[(n-1)s^2 + n(\bar{y}-\mu)^2]\right) d\mu$$

$$p(\sigma^2|y) \propto \int \sigma^{-n-2} exp\left(-\frac{1}{2\sigma^2}(n-1)s^2\right) \cdot exp\left(-\frac{n}{2\sigma^2}(\bar{y}-\mu)^2\right) d\mu$$

$$p(\sigma^2|y) \propto \sigma^{-n-2} exp\left(-\frac{1}{2\sigma^2}(n-1)s^2\right) \int exp\left(-\frac{n}{2\sigma^2}(\bar{y}-\mu)^2\right) d\mu$$

$$\because f(y|\mu.\sigma^2) = \int \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right) = 1$$

$$= \int exp\left(-\frac{1}{2\sigma^2}(y-\mu)^2\right) = \sigma\sqrt{2\pi} = \sqrt{2\pi\sigma^2}$$

$$\therefore p(\sigma^2|y) \propto \sigma^{-n-2} exp\left(-\frac{1}{2\sigma^2}(n-1)s^2\right) \cdot \sqrt{2\pi\sigma^2/n}$$

$$\therefore p(\sigma^2|y) \propto (\sigma^2)^{(-n+1)/2} exp\left(-\frac{(n-1)s^2}{2\sigma^2}\right)$$

which is **scaled inverse χ^2 density**.

$$(\sigma^2|y) \sim Inv\ \chi^2_{(n-1,s^2)}$$

3 Introduction to bayesplot

bayesplot is a package meant for plotting of data simulated using Markov chain Monte Carlo(MCMC).

```
library(bayesplot)
help(pack = bayesplot)
bayesplot::plotting-mcmc-draws
```

```r
library("bayesplot")
library("ggplot2")
library("rstanarm")

# help(mtcars)
names(mtcars)

##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
"carb"

head(mtcars)

##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1

tail(mtcars)

##                 mpg cyl  disp  hp drat    wt qsec vs am gear carb
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.7  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
## Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
## Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
## Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2

str(mtcars)

## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...

# linear regression model using stan_glm
# using '~ .' to include all variables
fit<- stan_glm(mpg ~ ., data = mtcars, seed = 1111)

##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 1).
## Chain 1:
```

```
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would
take 0 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.742 seconds (Warm-up)
## Chain 1:                0.64 seconds (Sampling)
## Chain 1:                1.382 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would
take 0 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.679 seconds (Warm-up)
## Chain 2:                0.361 seconds (Sampling)
## Chain 2:                1.04 seconds (Total)
## Chain 2:
##
```

```
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would
take 0 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.637 seconds (Warm-up)
## Chain 3:                0.553 seconds (Sampling)
## Chain 3:                1.19 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would
take 0 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.63 seconds (Warm-up)
## Chain 4:                0.344 seconds (Sampling)
```

```
## Chain 4:                0.974 seconds (Total)
## Chain 4:

print(fit)

## stan_glm
##  family:       gaussian [identity]
##  formula:      mpg ~ .
##  observations: 32
##  predictors:   11
## ------
##             Median MAD_SD
## (Intercept) 12.7   19.1
## cyl         -0.1    1.1
## disp         0.0    0.0
## hp           0.0    0.0
## drat         0.8    1.7
## wt          -3.6    1.9
## qsec         0.8    0.7
## vs           0.3    2.1
## am           2.4    2.2
## gear         0.6    1.5
## carb        -0.3    0.9
##
## Auxiliary parameter(s):
##       Median MAD_SD
## sigma 2.7    0.4
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg

summary(fit)

##
## Model Info:
##  function:     stan_glm
##  family:       gaussian [identity]
##  formula:      mpg ~ .
##  algorithm:    sampling
##  sample:       4000 (posterior sample size)
##  priors:       see help('prior_summary')
##  observations: 32
##  predictors:   11
##
## Estimates:
##               mean   sd    10%   50%   90%
## (Intercept)  12.9   19.7 -11.4  12.7  38.2
## cyl          -0.1    1.1  -1.5  -0.1   1.3
## disp          0.0    0.0   0.0   0.0   0.0
## hp            0.0    0.0   0.0   0.0   0.0
```

```
## drat          0.8    1.8  -1.4   0.8   3.0
## wt           -3.5    2.0  -6.1  -3.6  -1.0
## qsec          0.8    0.8  -0.2   0.8   1.7
## vs            0.3    2.2  -2.4   0.3   3.0
## am            2.5    2.2  -0.3   2.4   5.3
## gear          0.7    1.6  -1.3   0.6   2.6
## carb         -0.3    0.9  -1.4  -0.3   0.8
## sigma         2.8    0.4   2.3   2.7   3.4
##
## Fit Diagnostics:
##            mean   sd   10%   50%   90%
## mean_PPD 20.1    0.7 19.2  20.1  21.0
##
## The mean_ppd is the sample average posterior predictive distribution of
the outcome variable (for details see help('summary.stanreg')).
##
## MCMC diagnostics
##               mcse Rhat n_eff
## (Intercept)   0.4  1.0  3027
## cyl           0.0  1.0  2848
## disp          0.0  1.0  1845
## hp            0.0  1.0  2451
## drat          0.0  1.0  3828
## wt            0.0  1.0  1708
## qsec          0.0  1.0  2586
## vs            0.0  1.0  3381
## am            0.0  1.0  3441
## gear          0.0  1.0  3029
## carb          0.0  1.0  1594
## sigma         0.0  1.0  2026
## mean_PPD      0.0  1.0  4214
## log-posterior 0.1  1.0  1114
##
## For each parameter, mcse is Monte Carlo standard error, n_eff is a crude
measure of effective sample size, and Rhat is the potential scale reduction
factor on split chains (at convergence Rhat=1).
```

To use the posterior draws with the functions in the **bayesplot** package we'll extract them from the fitted model object :

```
# Convert the model into array
posterior<-as.array(fit)

# Dimensions of array
dim(posterior)

## [1] 1000    4   12

# Names of array
dimnames(posterior)
```

```
## $iterations
## NULL
##
## $chains
## [1] "chain:1" "chain:2" "chain:3" "chain:4"
##
## $parameters
##  [1] "(Intercept)" "cyl"         "disp"        "hp"          "drat"
"wt"
##  [7] "qsec"        "vs"          "am"          "gear"        "carb"
"sigma"
```

We've used `as.array` above (as opposed to `as.matrix`) because it keeps the Markov chains separate (`stan_glm` runs four chains by default). Most of the plots don't actually need the chains to be separate, but for a few of the plots we make in this vignette we'll want to show the chains individually.

3.1 Posterior uncertainty intervals

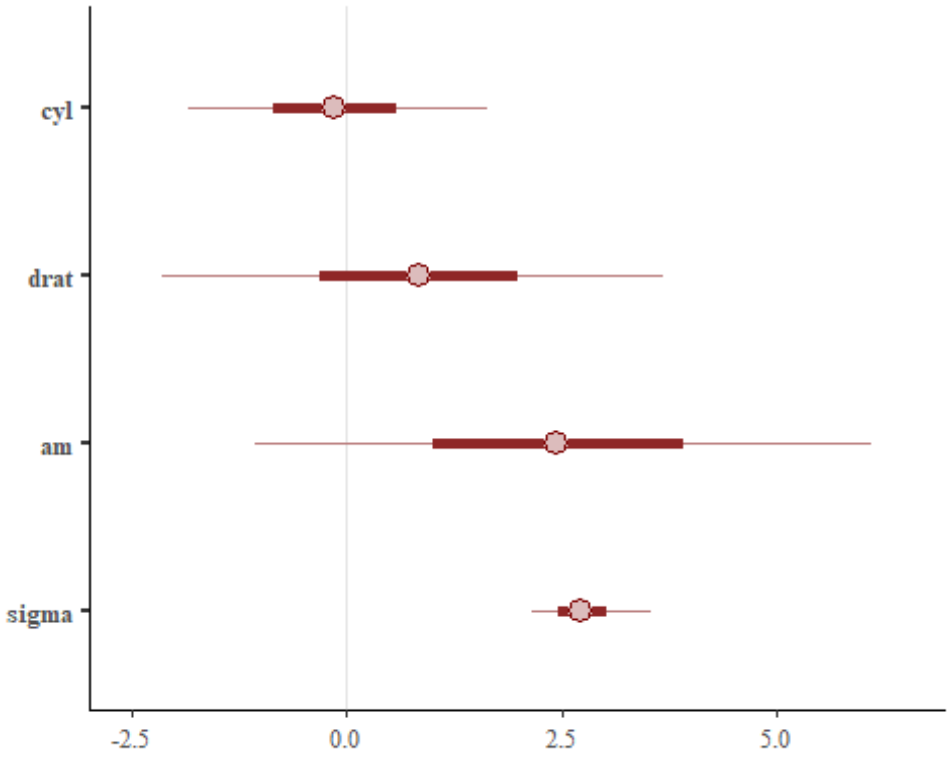
For models fit using MCMC we can compute posterior uncertainty intervals (sometimes called "credible intervals") in various ways. **bayesplot** currently provides plots of central intervals based on quantiles, although additional options may be provided in future releases (e.g., HDIs, which can be useful in particular cases).

```
help("MCMC-intervals")
```

3.1.1 mcmc_intervals, mcmc_areas

Central posterior uncertainty intervals can be plotted using the `mcmc_intervals` function.

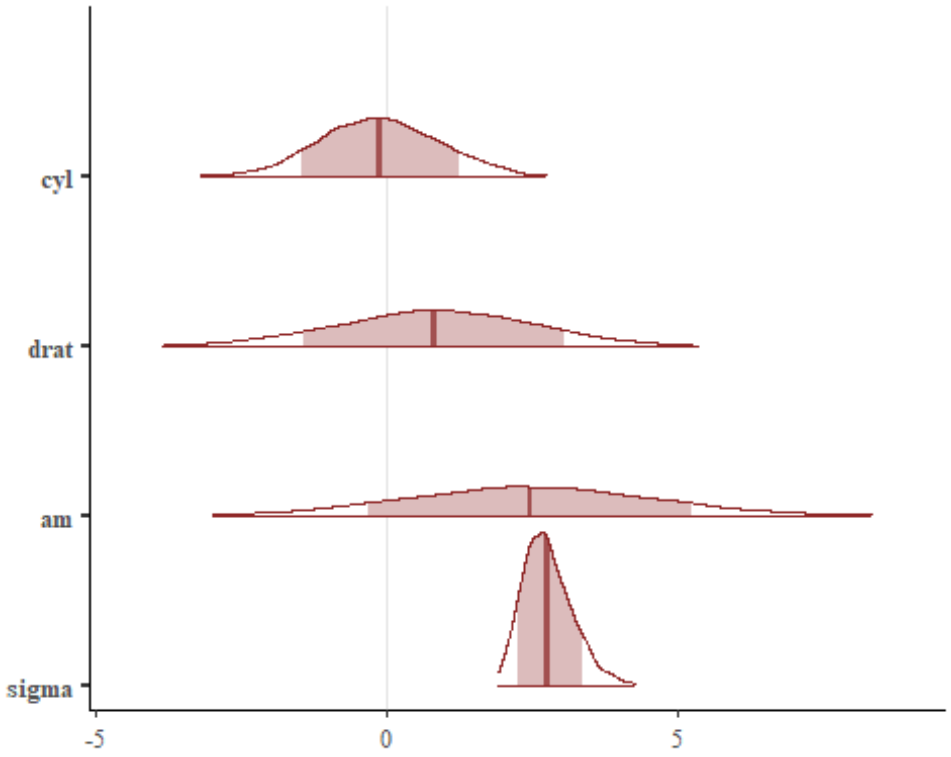
```
color_scheme_set("red")
mcmc_intervals(posterior, pars = c("cyl", "drat", "am", "sigma"))
```



The default is to show 50% intervals (the thick segments) and 90% intervals (the thinner outer lines). These defaults can be changed using the `prob` and `prob_outer` arguments, respectively. The points in the above plot are posterior medians. The `point_est` argument can be used to select posterior means instead or to omit the point estimates.

To show the uncertainty intervals as shaded areas under the estimated posterior density curves we can use the `mcmc_areas` function.

```
mcmc_areas(
  posterior,
  pars = c("cyl", "drat", "am", "sigma"),
  prob = 0.8, # 80% intervals
  prob_outer = 0.99, # 99%
  point_est = "mean"
)
```



3.2 Univariate marginal posterior distributions

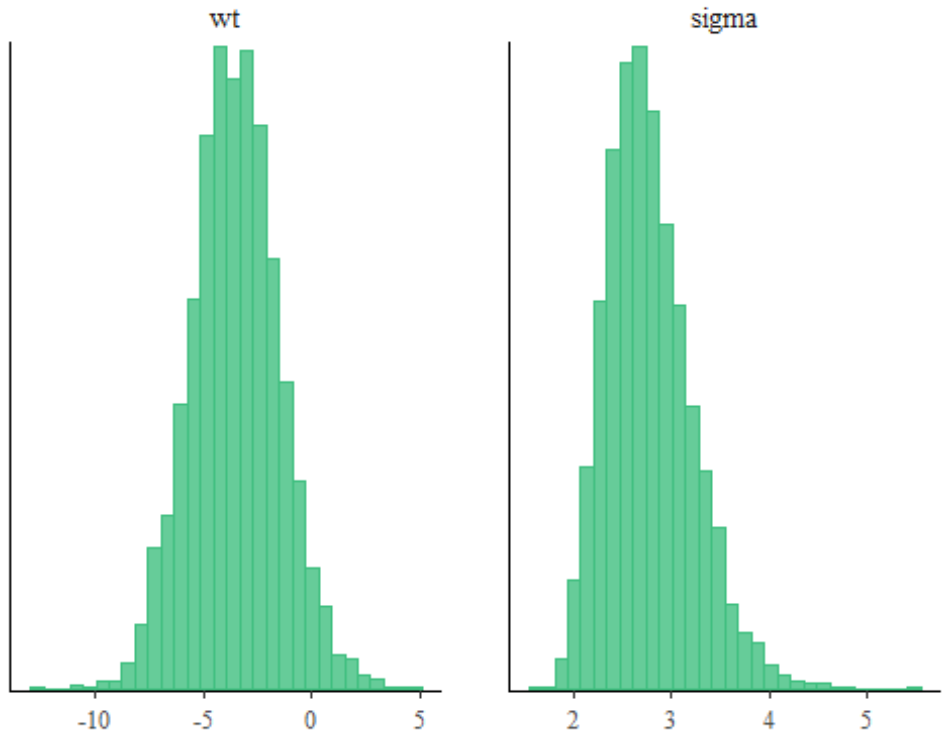
bayesplot provides functions for looking at histograms or kernel density estimates of marginal posterior distributions, either with all Markov chains combined or with the chains separate.

```
help("MCMC-distributions")
```

3.2.1 mcmc_hist

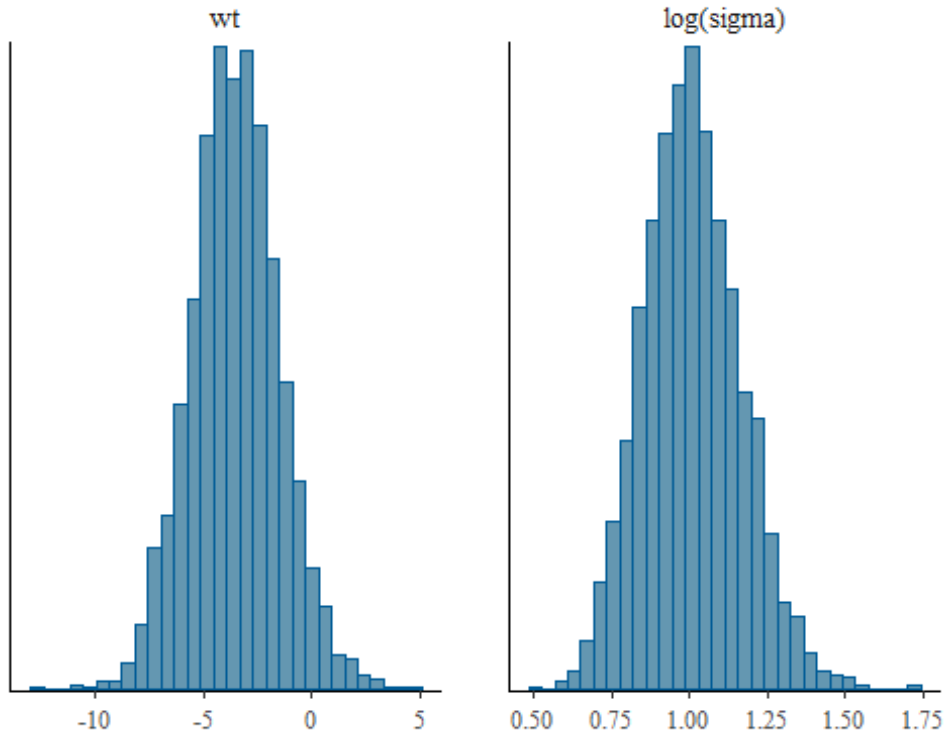
The `mcmc_hist` function plots marginal posterior distributions (combining all chains) :

```
color_scheme_set("green")
mcmc_hist(posterior, pars = c("wt", "sigma"))

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



If we want to plot `log(sigma)` rather than sigma we can either transform the draws in advance or use the `transformations` argument.

```
color_scheme_set("blue")
mcmc_hist(posterior, pars = c("wt", "sigma"),
          transformations = list("sigma" = "log"))

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

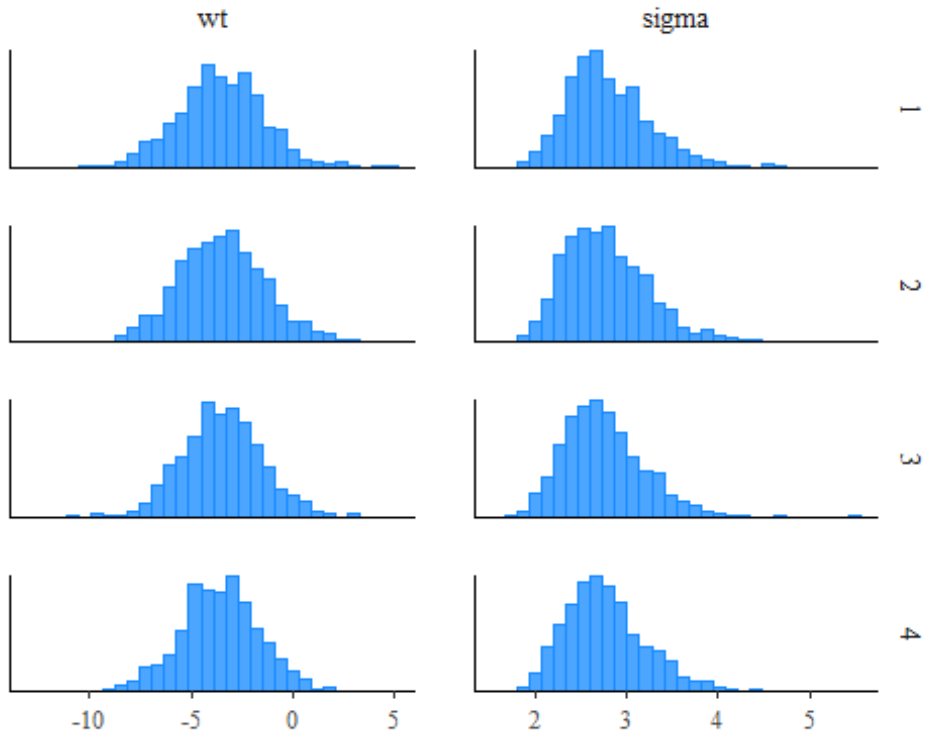


Most of the other functions for plotting MCMC draws also have a `transformations` argument.

3.2.2 mcmc_hist_by_chain

To view separate histograms of each of the four Markov chains we can use `mcmc_hist_by_chain`, which plots each chain in a separate facet in the plot.

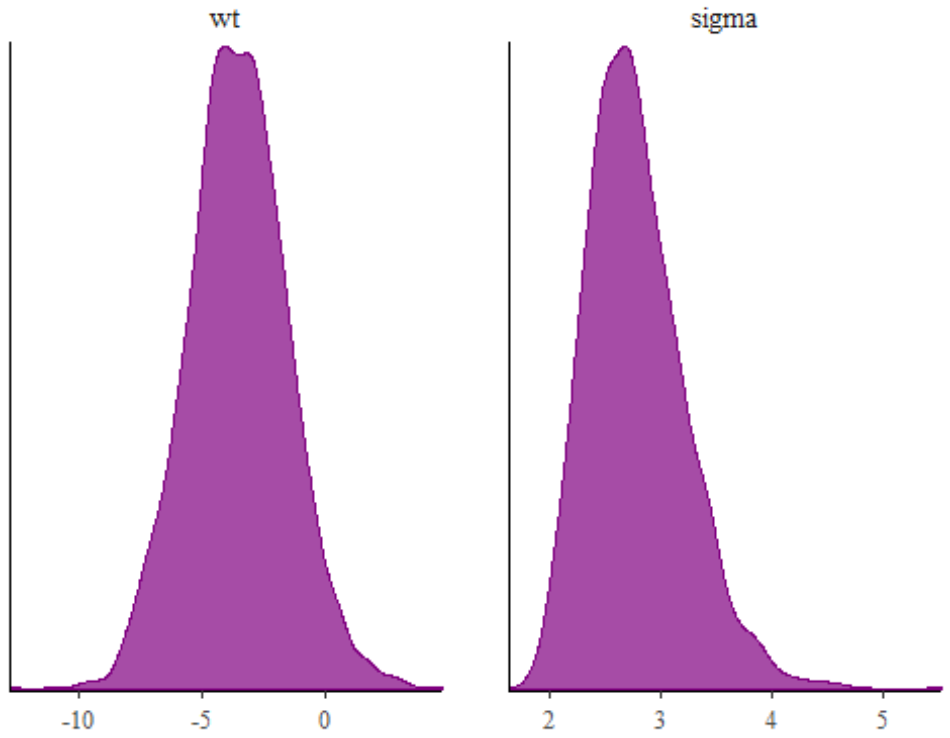
```
color_scheme_set("brightblue")
mcmc_hist_by_chain(posterior, pars = c("wt", "sigma"))

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



3.2.3 mcmc_dens

The `mcmc_dens` function is similar to `mcmc_hist` but plots kernel density estimates instead of histograms.

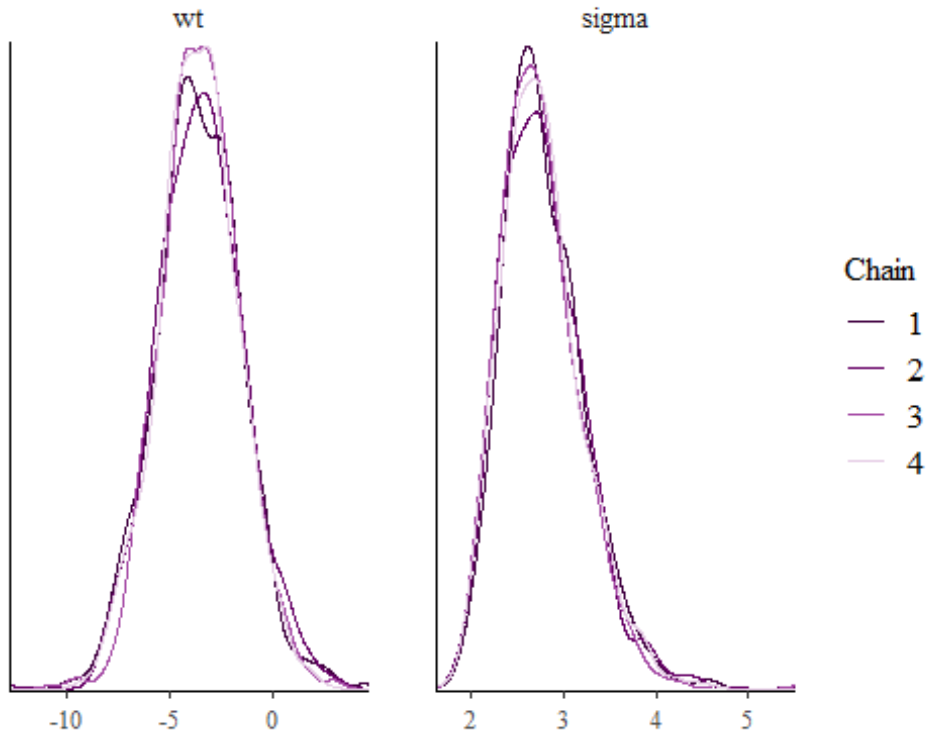
```
color_scheme_set("purple")
mcmc_dens(posterior, pars = c("wt", "sigma"))
```



3.2.4 mcmc_dens_overlay

Like `mcmc_hist_by_chain`, the `mcmc_dens_overlay` function separates the Markov chains. But instead of plotting each chain individually, the density estimates are overlaid.

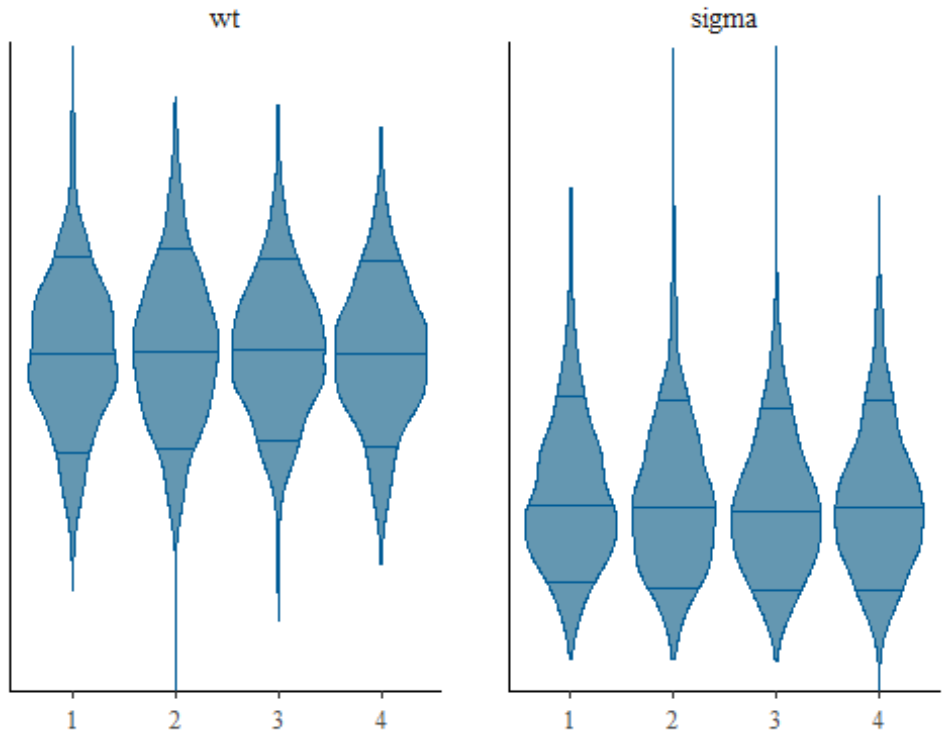
```
mcmc_dens_overlay(posterior, pars = c("wt", "sigma"))
```



3.2.5 mcmc_violin

The `mcmc_violin` function plots the density estimates of each chain as violins and draws horizontal line segments at user-specified quantiles.

```
color_scheme_set("blue")
mcmc_violin(posterior, pars = c("wt", "sigma"), probs = c(0.1, 0.5, 0.9))
```



3.3 Bivariate Plots

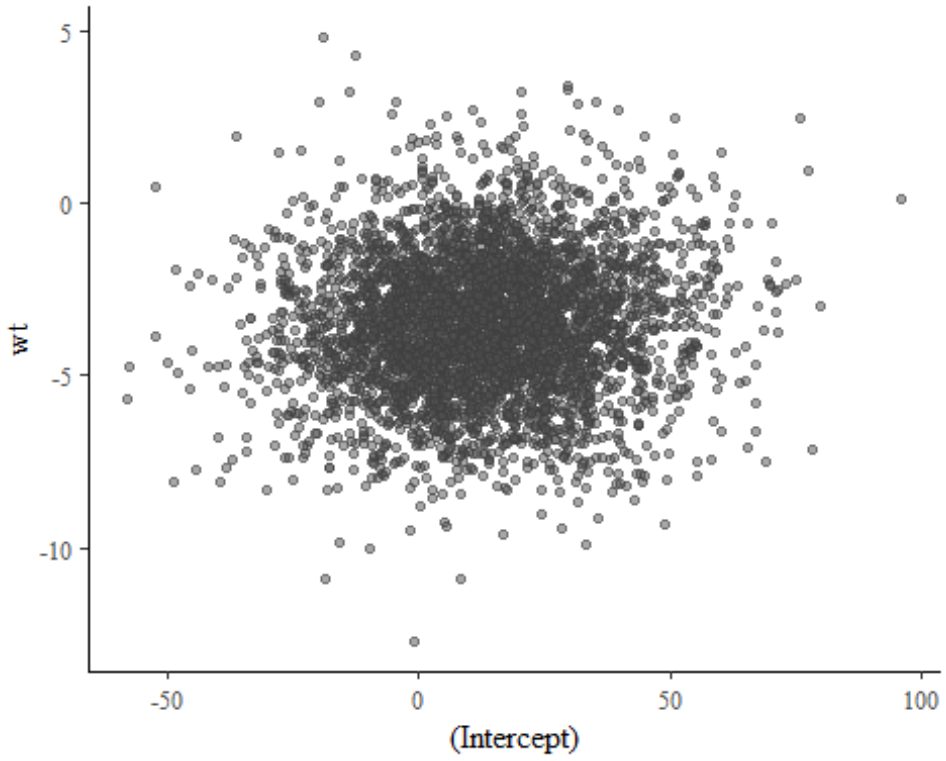
Various functions are available for plotting bivariate marginal posterior distributions.

3.3.1 mcmc_scatter

```
help("MCMC-scatterplots")
```

The `mcmc_scatter` function creates a simple scatterplot of two parameters.

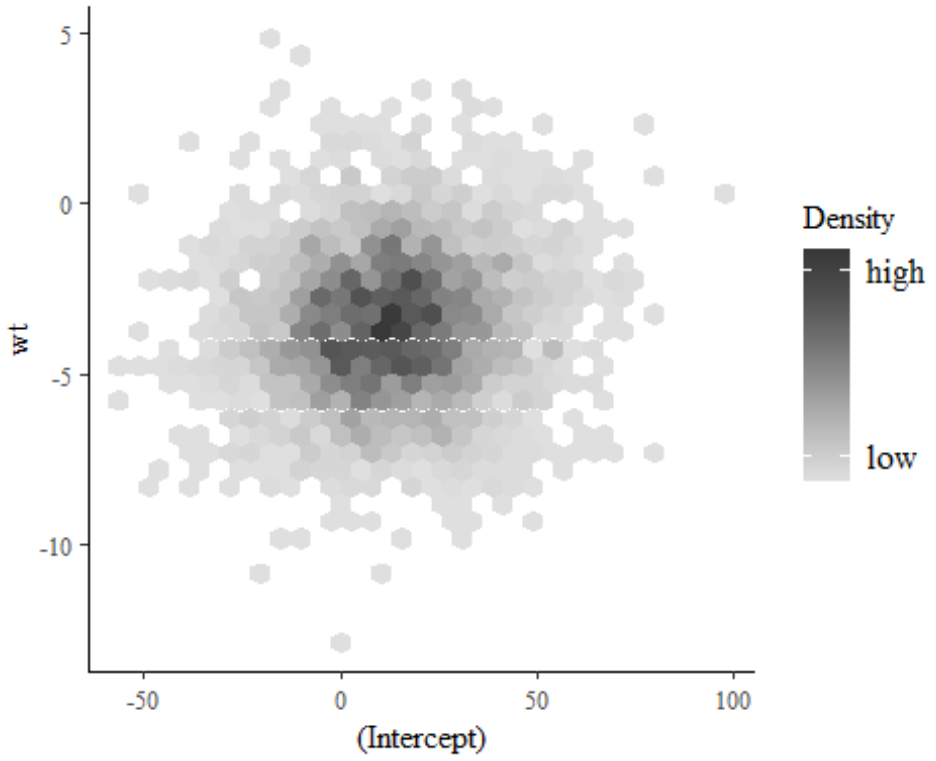
```
require(bayesplot)
color_scheme_set("gray")
mcmc_scatter(posterior, pars = c("(Intercept)", "wt"),
             size = 1.5, alpha = 0.5)
```



3.3.2 mcmc_hex

The mcmc_hex function creates a similar plot but using hexagonal binning, which can be useful to avoid overplotting.

```
# requires hexbin package
if (requireNamespace("hexbin", quietly = TRUE)) {
  mcmc_hex(posterior, pars = c("(Intercept)", "wt"))
}

# Same as above
# requires hexbin package
require(hexbin)
mcmc_hex(posterior, pars = c("(Intercept)", "wt"))
```

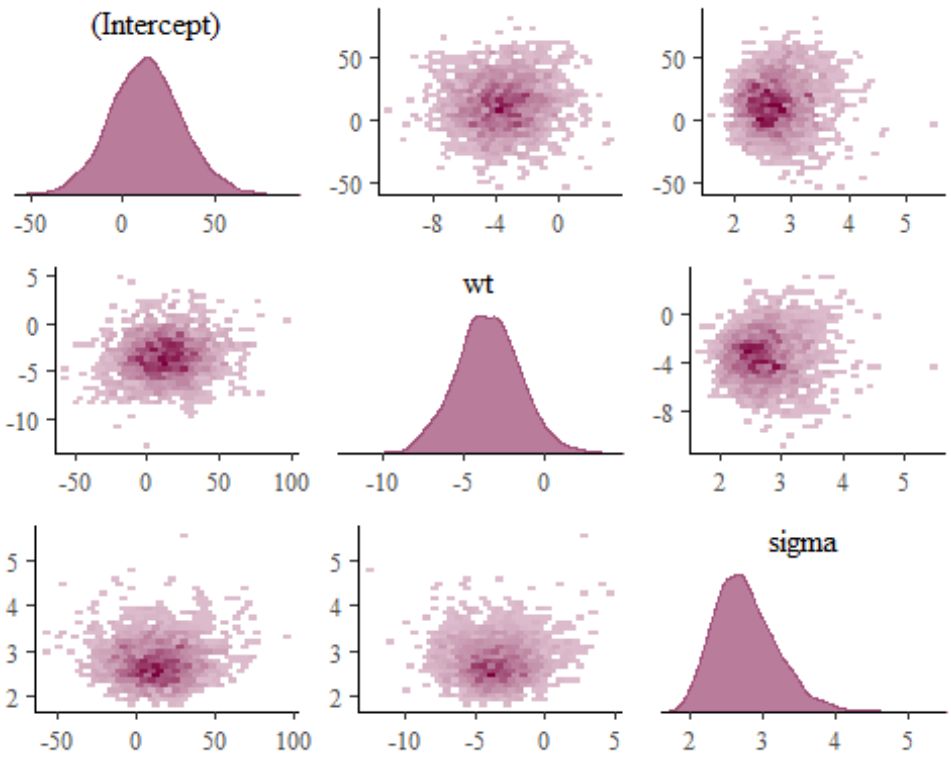


3.3.3 mcmc_pairs

In addition to mcmc_scatter and mcmc_hex, bayesplot now provides an mcmc_pairs function for creating pairs plots with more than two parameters.

```
color_scheme_set("pink")
mcmc_pairs(posterior, pars = c("(Intercept)", "wt", "sigma"),
           off_diag_args = list(size = 1.5), diag_fun="dens",
off_diag_fun="hex")
```

```
## Warning: The following arguments were unrecognized and ignored: size
## The following arguments were unrecognized and ignored: size
## The following arguments were unrecognized and ignored: size
## The following arguments were unrecognized and ignored: size
## The following arguments were unrecognized and ignored: size
## The following arguments were unrecognized and ignored: size
```



The univariate marginal posteriors are shown along the diagonal as histograms, but this can be changed to densities by setting `diag_fun="dens"`. Bivariate plots are displayed above and below the diagonal as scatterplots, but it is also possible to use hex plots by setting `off_diag_fun="hex`". By default, `mcmc_pairs` shows some of the Markov chains (half, if an even number of chains) above the diagonal and the others below. There are many more options for controlling how the draws should be split between the plots above and below the diagonal (see the documentation for the `condition` argument), but they are more useful when MCMC diagnostic information is included.

3.4 Trace Plots

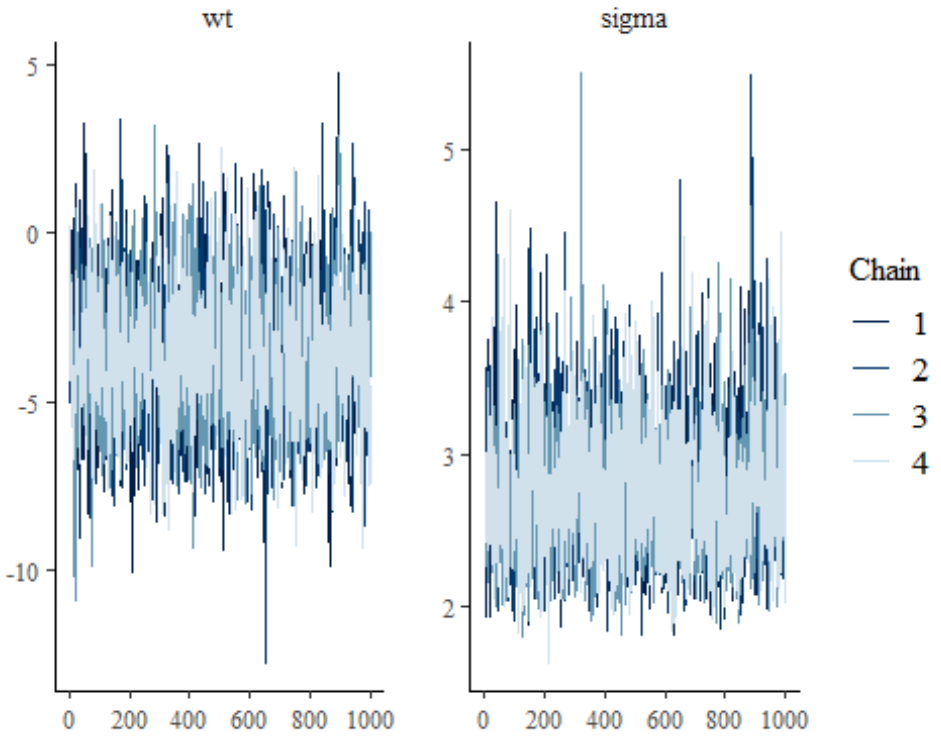
Trace plots are time series plots of Markov chains.

```
help("MCMC-traces")
```

3.4.1 mcmc_trace

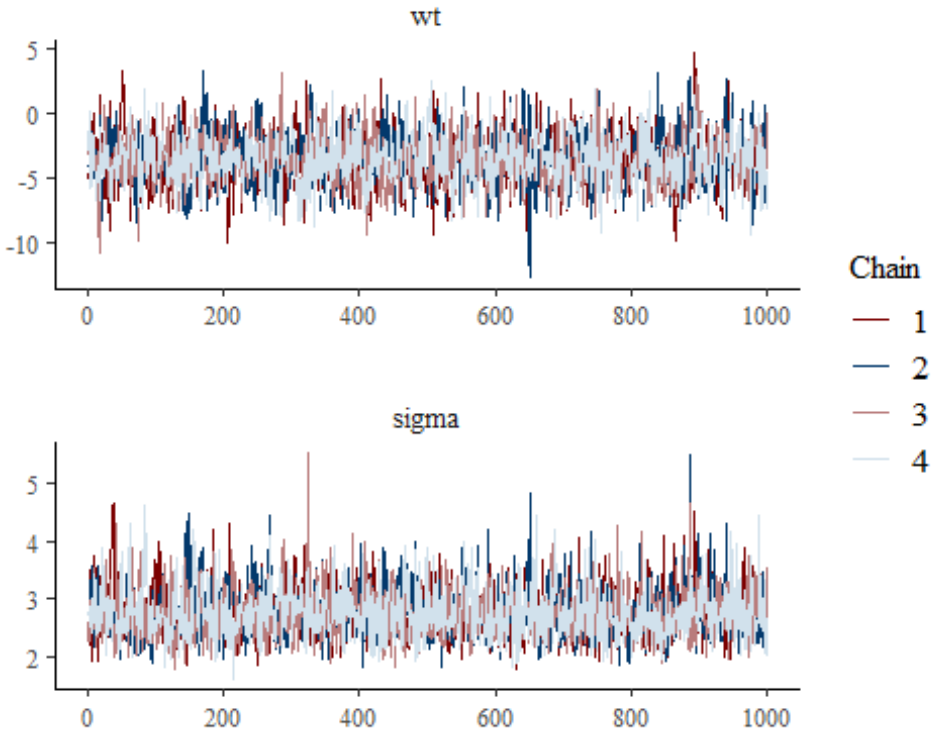
The `mcmc_trace` function creates standard trace plots:

```
color_scheme_set("blue")
mcmc_trace(posterior, pars = c("wt", "sigma"))
```



If it's hard to see the difference between the chains we can change to a mixed color scheme, for example:

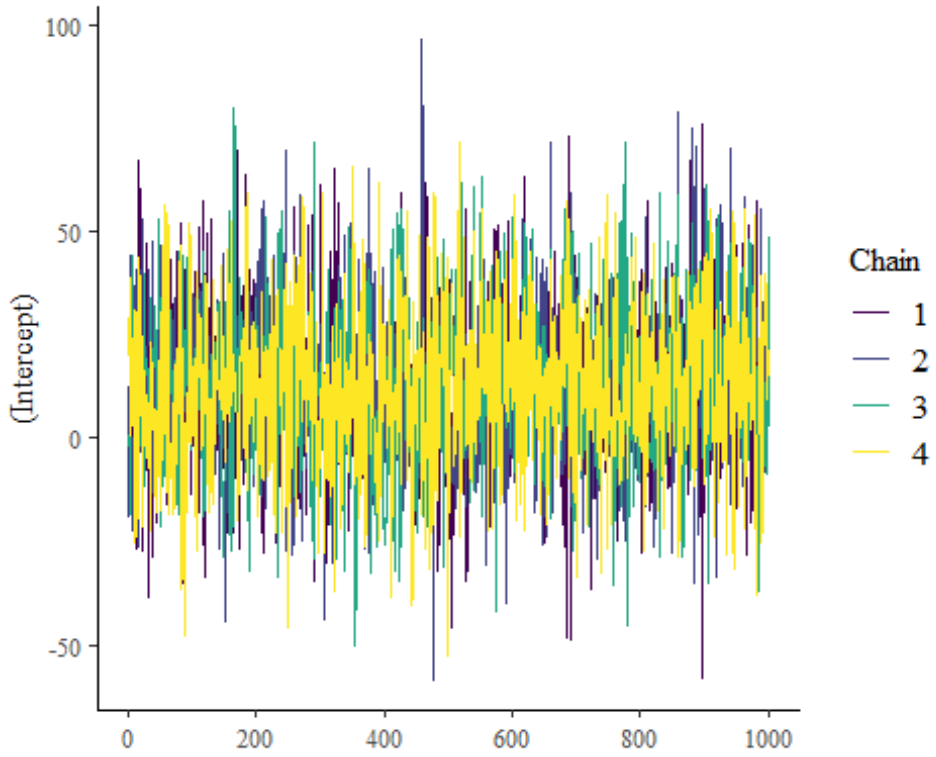
```
color_scheme_set("mix-blue-red")
mcmc_trace(posterior, pars = c("wt", "sigma"),
           facet_args = list(ncol = 1, strip.position = "top"))
```



The code above also illustrates the use of the `facet_args` argument, which is a list of parameters passed to `facet_wrap` in **ggplot2**. Specifying `ncol=1` means the trace plots will be stacked in a single column rather than placed side by side, and `strip.position="left"` moves the facet labels to the y-axis (instead of above each facet).

The `"viridis"` color scheme is also useful for trace plots because it is comprised of very distinct colors:

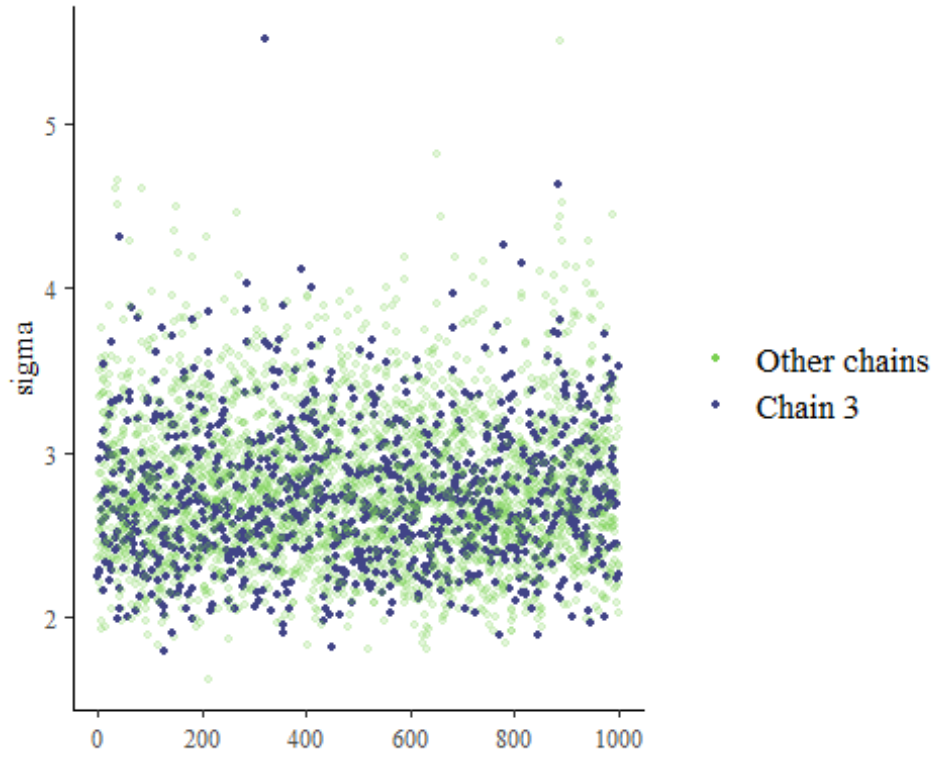
```
color_scheme_set("viridis")
mcmc_trace(posterior, pars = "(Intercept)")
```



3.4.2 mcmc_trace_highlight

The `mcmc_trace_highlight` function uses points instead of lines and reduces the opacity of all but a single chain (which is specified using the highlight argument).

```
mcmc_trace_highlight(posterior, pars = "sigma", highlight = 3)
```



4 Graphical posterior predictive checks using the bayesplot package

```
library(bayesplot)
help(pack = bayesplot)
bayesplot::graphical-ppcs
```

4.1 Graphical posterior predictive checks (PPCs)

The **bayesplot** package provides various plotting functions for graphical posterior predictive checking, that is, creating graphical displays comparing observed data to simulated data from the posterior predictive distribution (Gabry et al, 2019).

The idea behind posterior predictive checking is simple: if a model is a good fit then we should be able to use it to generate data that looks a lot like the data we observed. To generate the data used for posterior predictive checks (PPCs) we simulate from the *posterior predictive distribution*. This is the distribution of the outcome variable implied by a model after using the observed data y (a vector of N outcome values) to update our beliefs about unknown model parameters θ. The posterior predictive distribution for observation $\tilde{y}$ can be written as

$$p(\tilde{y}|y) = \int p(\tilde{y}|\theta)p(\theta|y)d\theta$$

Typically we will also condition on X (a matrix of predictor variables).

For each draw (simulation) $s = 1, \ldots, S$ of the parameters from the posterior distribution, $\theta^{(s)} \sim p(\theta|y)$, we draw an entire vector of N outcomes $\tilde{y}^{(s)}$ from the posterior predictive distribution by simulating from the data model conditional on parameters $\theta^{(s)}$. The result is an $S \times N$ matrix of draws $\tilde{y}$.

When simulating from the posterior predictive distribution we can use either the same values of the predictors X that we used when fitting the model or new observations of those predictors. When we use the same values of X we denote the resulting simulations by y^{rep}, as they can be thought of as replications of the outcome y rather than predictions for future observations $(\tilde{y}\ using\ predictors\ \tilde{X})$. This corresponds to the notation from Gelman et al. (2013) and is the notation used throughout the package documentation.

Using the replicated datasets drawn from the posterior predictive distribution, the functions in the **bayesplot** package create various graphical displays comparing the observed data y to the replications. The names of the **bayesplot** plotting functions for posterior predictive checking all have the prefix `ppc_`.

```
library("bayesplot")
library("ggplot2")
library("rstanarm")
```

Example models

To demonstrate some of the various PPCs that can be created with the bayesplot package we'll use an example of comparing Poisson and Negative binomial regression models from one of the **rstanarm** package.

First we fit a Poisson regression model with outcome variable y representing the roach count in each apartment at the end of the experiment.

```
# see help("rstanarm-datasets")
head(roaches)

##     y roach1 treatment senior exposure2
## 1 153 3.0800         1      0  0.800000
## 2 127 3.3125         1      0  0.600000
## 3   7 0.0167         1      0  1.000000
## 4   7 0.0300         1      0  1.000000
## 5   0 0.0200         1      0  1.142857
## 6   0 0.0000         1      0  1.000000
```

roaches

Data on the efficacy of a pest management system at reducing the number of roaches in urban apartments.

Source: Gelman and Hill (2007)

262 obs. of 6 variables

- y Number of roaches caught

- `roach1` Pretreatment number of roaches
- `treatment` Treatment indicator
- `senior` Indicator for only elderly residents in building
- `exposure2` Number of days for which the roach traps were used

```
# pre-treatment number of roaches (in 100s)
roaches$roach100 <- roaches$roach1 / 100

# using rstanarm's default priors. For details see the section on default
# weakly informative priors at https://mc-
stan.org/rstanarm/articles/priors.html
fit_poisson <- stan_glm(
  y ~ roach100 + treatment + senior,
  offset = log(exposure2),
  family = poisson(link = "log"),
  data = roaches,
  seed = 1111,
  refresh = 0 # suppresses all output as of v2.18.1 of rstan
)

print(fit_poisson)
```

```
## stan_glm
##  family:       poisson [log]
##  formula:      y ~ roach100 + treatment + senior
##  observations: 262
##  predictors:   4
## ------
##             Median MAD_SD
## (Intercept)  3.1    0.0
## roach100    69.8    0.9
## treatment   -0.5    0.0
## senior      -0.4    0.0
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

We'll also fit the negative binomial model that we'll compare to the Poisson:

```
fit_nb <- update(fit_poisson, family = "neg_binomial_2")

print(fit_nb)
```

```
## stan_glm
##  family:       neg_binomial_2 [log]
##  formula:      y ~ roach100 + treatment + senior
##  observations: 262
```

```
##  predictors:   4
## ------
##             Median MAD_SD
## (Intercept)   2.8    0.2
## roach100    131.0   26.1
## treatment    -0.8    0.2
## senior       -0.3    0.3
##
## Auxiliary parameter(s):
##                       Median MAD_SD
## reciprocal_dispersion 0.3    0.0
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

4.2 Defining y and yrep

In order to use the PPC functions from the **bayesplot** package we need a vector y of outcome values,

```
y <- roaches$y
```

and a matrix y^{rep} of draws from the posterior predictive distribution,

```
yrep_poisson <- posterior_predict(fit_poisson, draws = 500)

yrep_nb <- posterior_predict(fit_nb, draws = 500)

dim(yrep_poisson)

## [1] 500 262

dim(yrep_nb)

## [1] 500 262
```

Each row of the matrix is a draw from the posterior predictive distribution, i.e. a vector with one element for each of the data points in y.

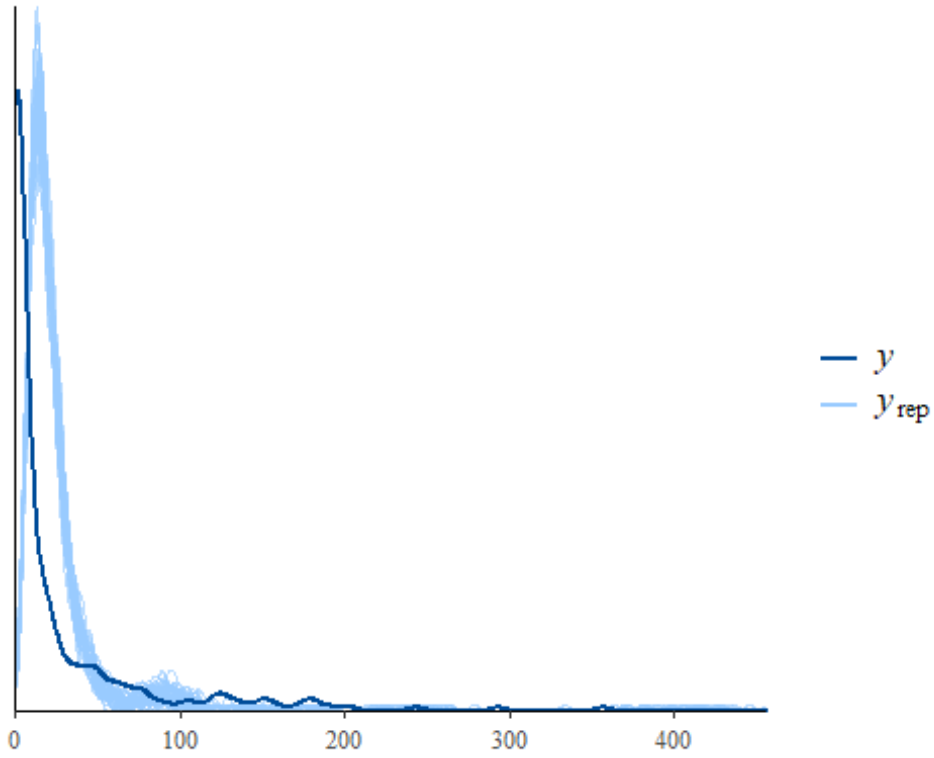
Since we fit the models using **rstanarm** we used its special `posterior_predict` function, but if we were using a model fit with the rstan package we could create yrep in the `generated quantities` block of the Stan program or by doing simulations in R after fitting the model. Draws from the posterior predictive distribution can be used with **bayesplot** regardless of whether or not the model was fit using an interface to Stan. **bayesplot** just requires a `yrep` matrix that has `number_of_draws` rows and `number_of_observations` columns.

4.3 Histograms and density estimates

4.3.1 ppc_dens_overlay

The first PPC we'll look at is a comparison of the distribution of y and the distributions of some of the simulated datasets (rows) in the yrep matrix.

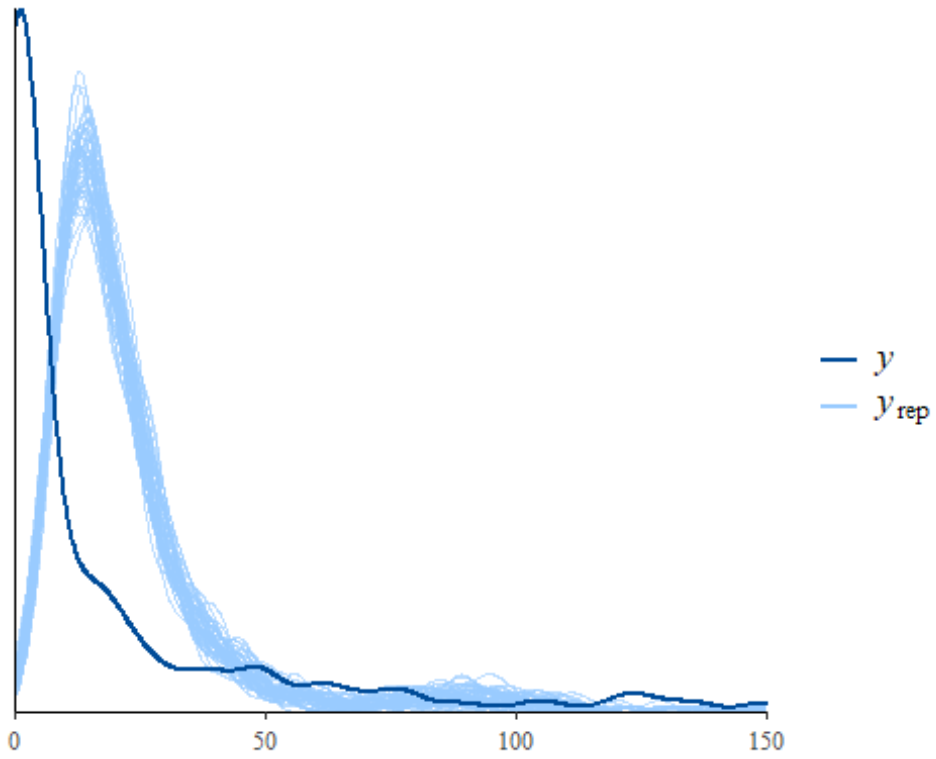
```
color_scheme_set("brightblue")
ppc_dens_overlay(y, yrep_poisson[1:50, ])
```



In the plot above, the dark line is the distribution of the observed outcomes y and each of the 50 lighter lines is the kernel density estimate of one of the replications of y from the posterior predictive distribution (i.e., one of the rows in yrep). This plot makes it easy to see that this model fails to account for the large proportion of zeros in y. That is, the model predicts fewer zeros than were actually observed.

To see the discrepancy at the lower values of more clearly we can use the xlim function from **ggplot2** to restrict the range of the x-axis:

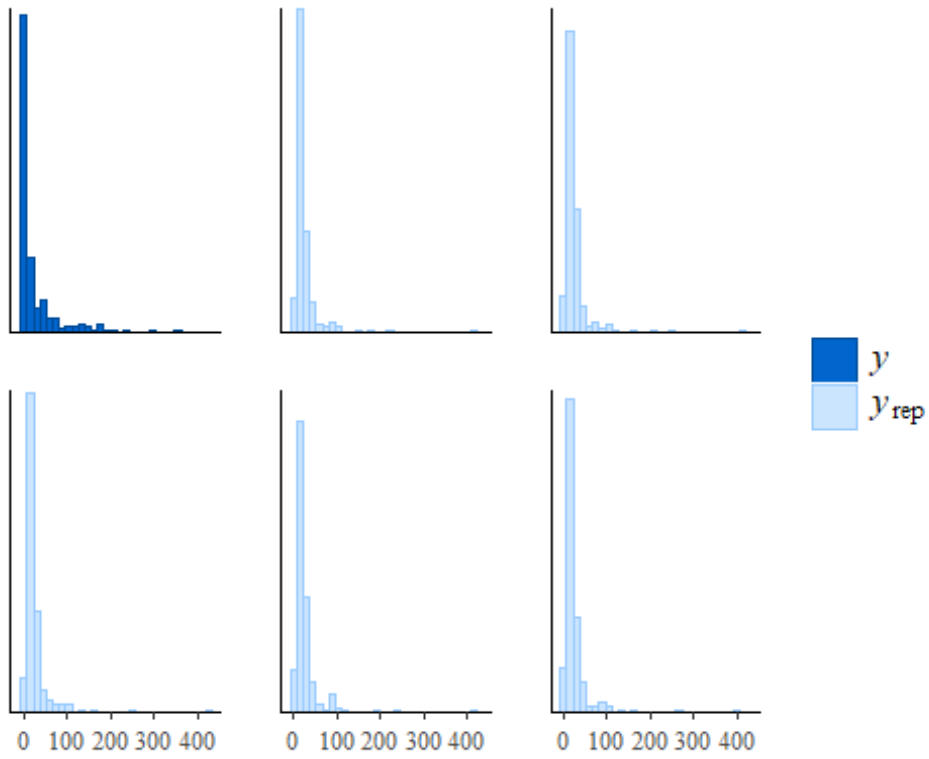
```
ppc_dens_overlay(y, yrep_poisson[1:50, ]) + xlim(0, 150)

## Warning: Removed 154 rows containing non-finite values (stat_density).

## Warning: Removed 11 rows containing non-finite values (stat_density).
```



4.3.2 ppc_hist

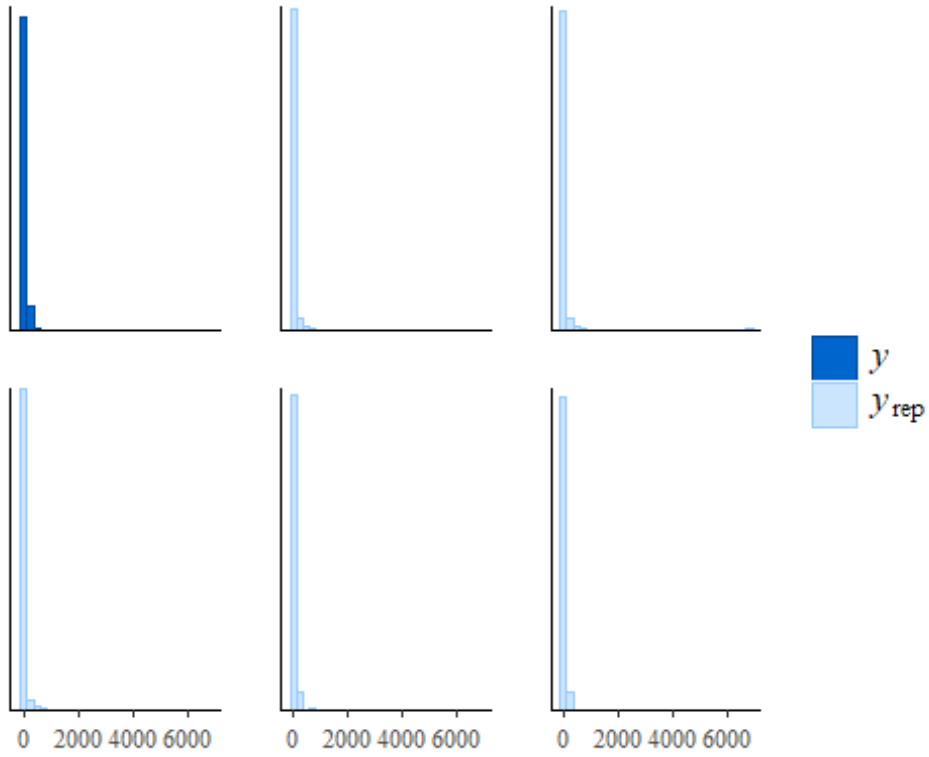
We could see the same thing from a different perspective by looking at separate histograms of y and some of the yrep datasets using the ppc_hist function :

```
ppc_hist(y, yrep_poisson[1:5, ])

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



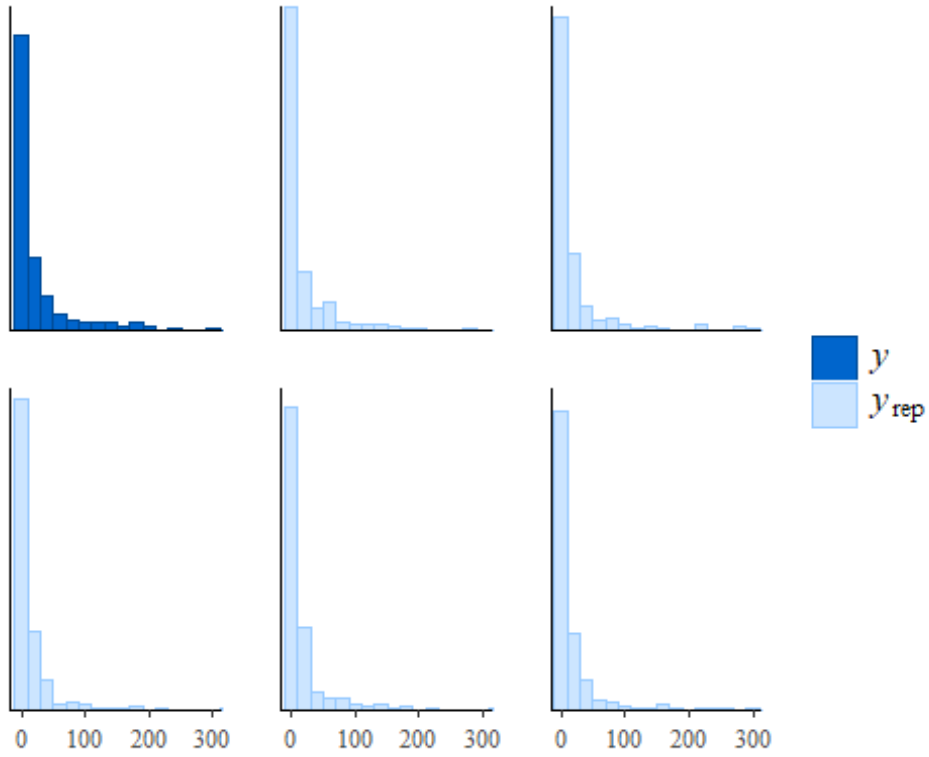
The same plot for the negative binomial model looks much different:

```
ppc_hist(y, yrep_nb[1:5, ])

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



The negative binomial model does better handling the number of zeros in the data, but it occasionally predicts values that are way too large, which is why the x-axes extend to such high values in the plot and make it difficult to read. To see the predictions for the smaller values more clearly we can zoom in :

```
ppc_hist(y, yrep_nb[1:5, ], binwidth = 20) +
  coord_cartesian(xlim = c(-1, 300))
```



4.4 Distributions of test statistics

Another way to see that the Poisson model predicts too few zeros is to look at the distribution of the proportion of zeros over the replicated datasets from the posterior predictive distribution in `yrep` and compare to the proportion of observed zeros in `y`.

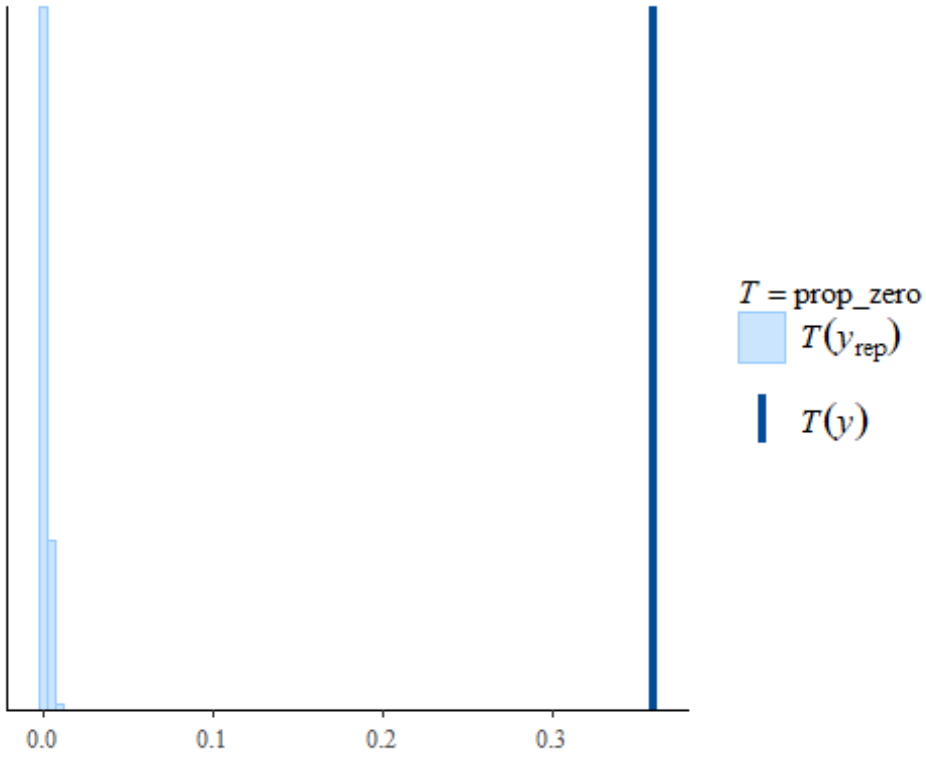
4.4.1 ppc_stat

First we define a function that takes a vector as input and returns the proportion of zeros :

```
prop_zero<- function(x) mean(x == 0)
prop_zero(y) # check proportion of zeros in y

## [1] 0.3587786
```

The `stat` argument to `ppc_stat` accepts a function or the name of a function for computing a test statistic from a vector of data. In our case we can specify `stat = "prop_zero"` since we've already defined the `prop_zero` function, but we also could have used `stat = function(x) mean(x == 0)`.

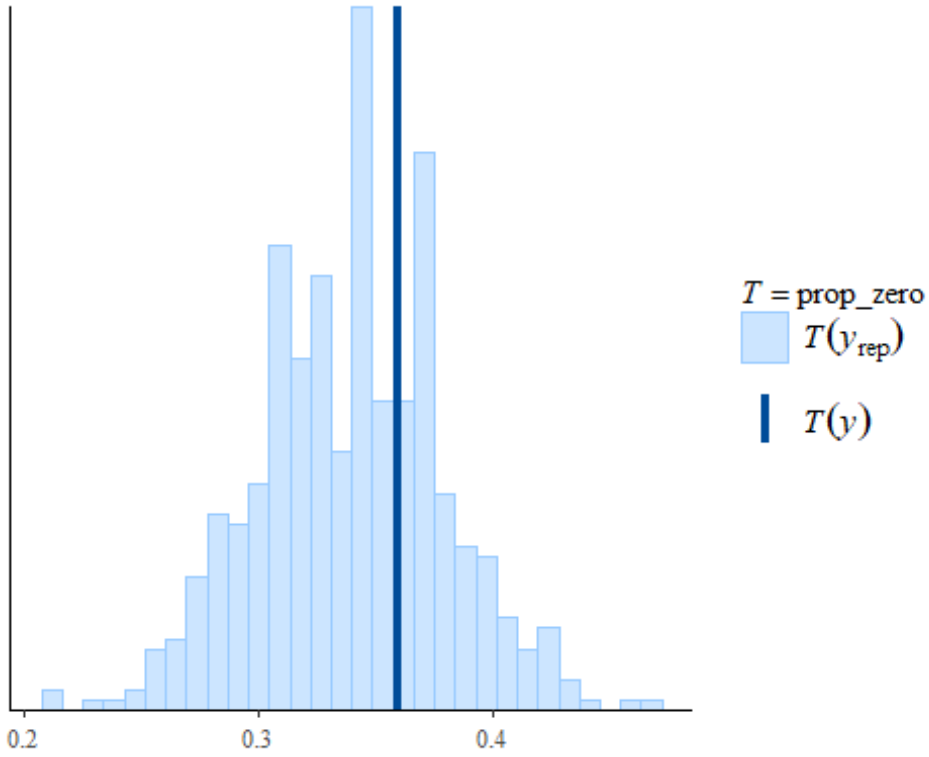
```
ppc_stat(y, yrep_poisson, stat = "prop_zero", binwidth = 0.005)
```



The dark line is at the value $T(y)$, i.e. the value of the test statistic computed from the observed y, in this case `prop_zero(y)`. The lighter area on the left is actually a histogram of the proportion of zeros in in the `yrep` simulations, but it can be hard to see because almost none of the simulated datasets in `yrep` have any zeros.

Here's the same plot for the negative binomial model:

```
ppc_stat(y, yrep_nb, stat = "prop_zero")

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

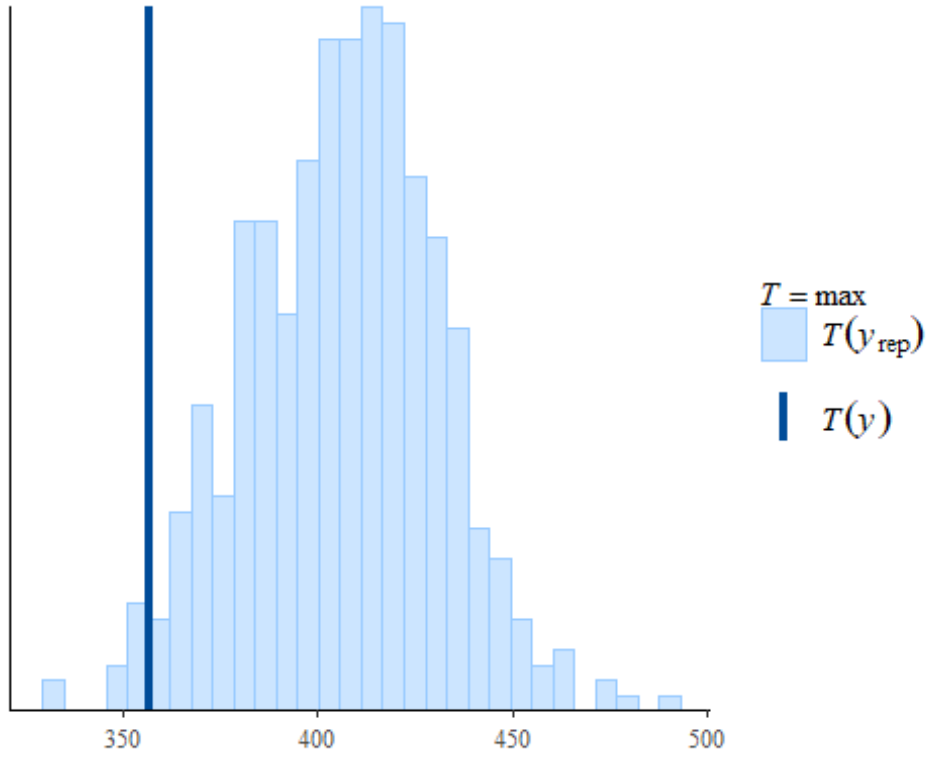


Again we see that the negative binomial model does a much better job predicting the proportion of observed zeros than the Poisson.

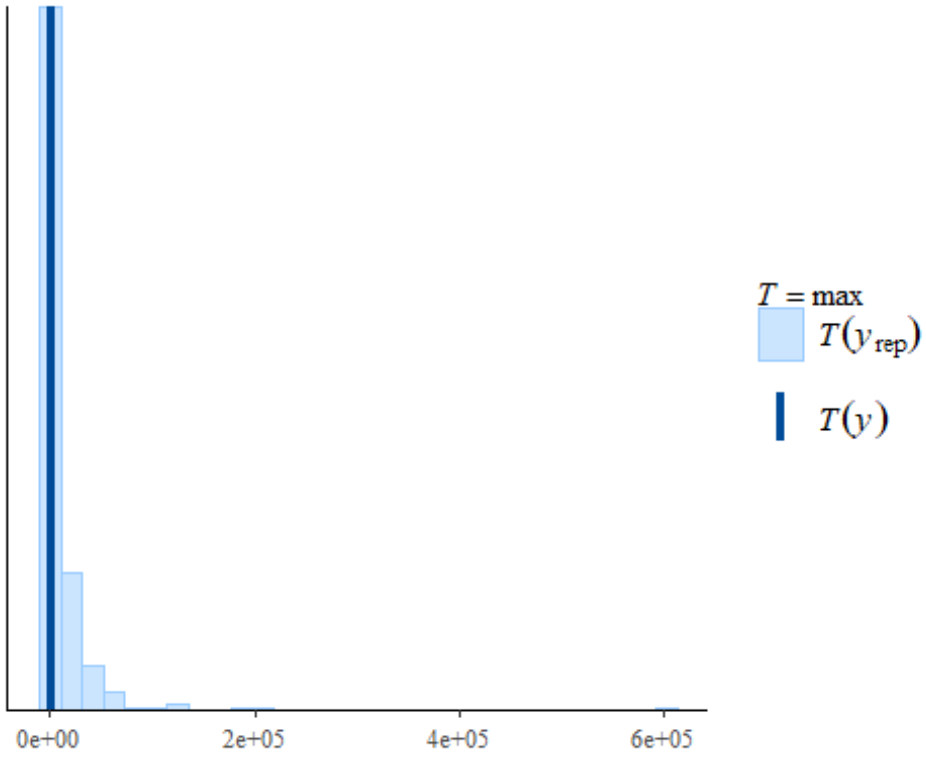
However, if we look instead at the distribution of the maximum value in the replications, we can see that the Poisson model makes more realistic predictions than the negative binomial:

```
ppc_stat(y, yrep_poisson, stat = "max")
```

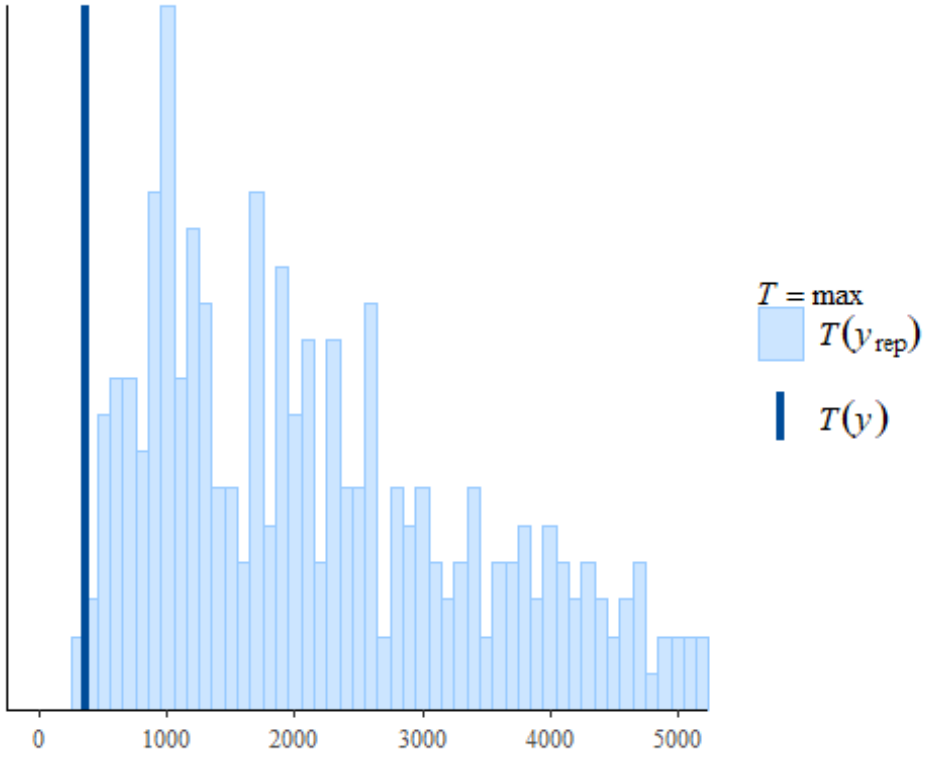
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
# For NB
ppc_stat(y, yrep_nb, stat = "max")

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
ppc_stat(y, yrep_nb, stat = "max", binwidth = 100) +
  coord_cartesian(xlim = c(-1, 5000))
```



4.5 Other PPCs and PPCs by group

There are many additional PPCs available, including plots of predictive intervals, distributions of predictive errors, and more.

The `available_ppc` function can also be used to list the names of all PPC plotting functions :

```
available_ppc()
```

```
## bayesplot PPC module:
##   ppc_bars
##   ppc_bars_grouped
##   ppc_boxplot
##   ppc_dens
##   ppc_dens_overlay
##   ppc_dens_overlay_grouped
##   ppc_ecdf_overlay
##   ppc_ecdf_overlay_grouped
##   ppc_error_binned
##   ppc_error_hist
##   ppc_error_hist_grouped
##   ppc_error_scatter
##   ppc_error_scatter_avg
##   ppc_error_scatter_avg_grouped
##   ppc_error_scatter_avg_vs_x
##   ppc_freqpoly
##   ppc_freqpoly_grouped
##   ppc_hist
##   ppc_intervals
##   ppc_intervals_grouped
##   ppc_km_overlay
##   ppc_km_overlay_grouped
##   ppc_loo_intervals
##   ppc_loo_pit
##   ppc_loo_pit_overlay
##   ppc_loo_pit_qq
##   ppc_loo_ribbon
##   ppc_ribbon
##   ppc_ribbon_grouped
##   ppc_rootogram
##   ppc_scatter
##   ppc_scatter_avg
##   ppc_scatter_avg_grouped
##   ppc_stat
##   ppc_stat_2d
##   ppc_stat_freqpoly
##   ppc_stat_freqpoly_grouped
##   ppc_stat_grouped
##   ppc_violin_grouped
```

```
available_ppc(pattern = "_grouped")
```

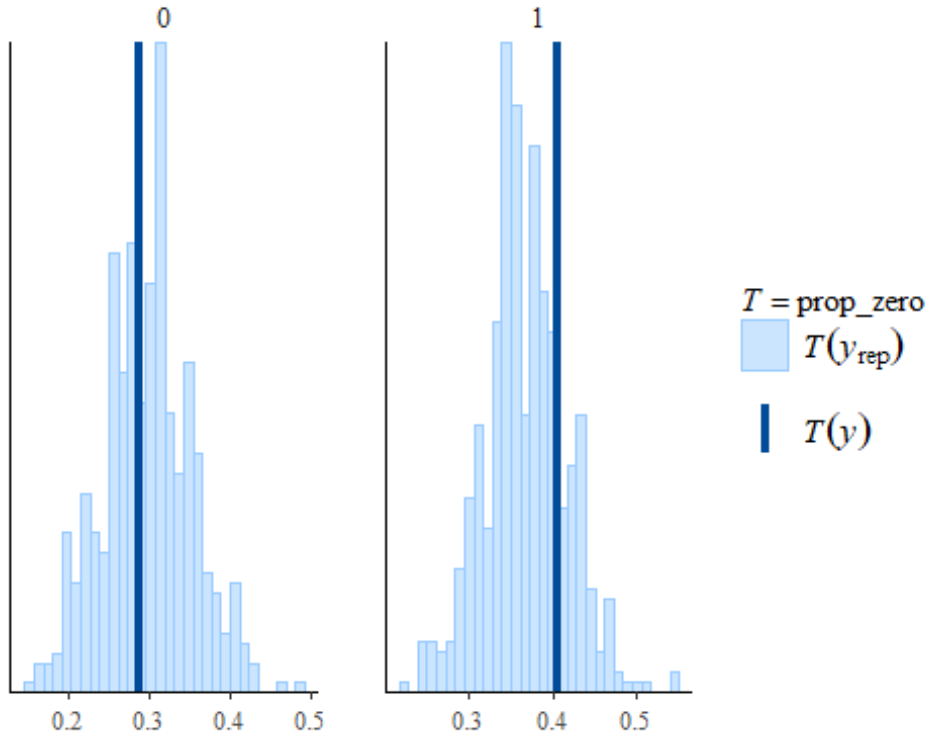
```
## bayesplot PPC module:
## (matching pattern '_grouped')
##   ppc_bars_grouped
##   ppc_dens_overlay_grouped
##   ppc_ecdf_overlay_grouped
##   ppc_error_hist_grouped
##   ppc_error_scatter_avg_grouped
##   ppc_freqpoly_grouped
##   ppc_intervals_grouped
##   ppc_km_overlay_grouped
##   ppc_ribbon_grouped
##   ppc_scatter_avg_grouped
##   ppc_stat_freqpoly_grouped
##   ppc_stat_grouped
##   ppc_violin_grouped
```

4.5.1 ppc_stat_grouped

For example, ppc_stat_grouped is the same as ppc_stat except that the test statistic is computed within levels of the grouping variable and a separate plot is made for each level :

```
ppc_stat_grouped(y, yrep_nb, group = roaches$treatment, stat = "prop_zero")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



4.6 Providing an interface to bayesplot PPCs from another package

The **bayesplot** package provides the S3 generic function `pp_check`. Authors of R packages for Bayesian inference are encouraged to define methods for the fitted model objects created by their packages. This will hopefully be convenient for both users and developers and contribute to the use of the same naming conventions across many of the R packages for Bayesian data analysis.

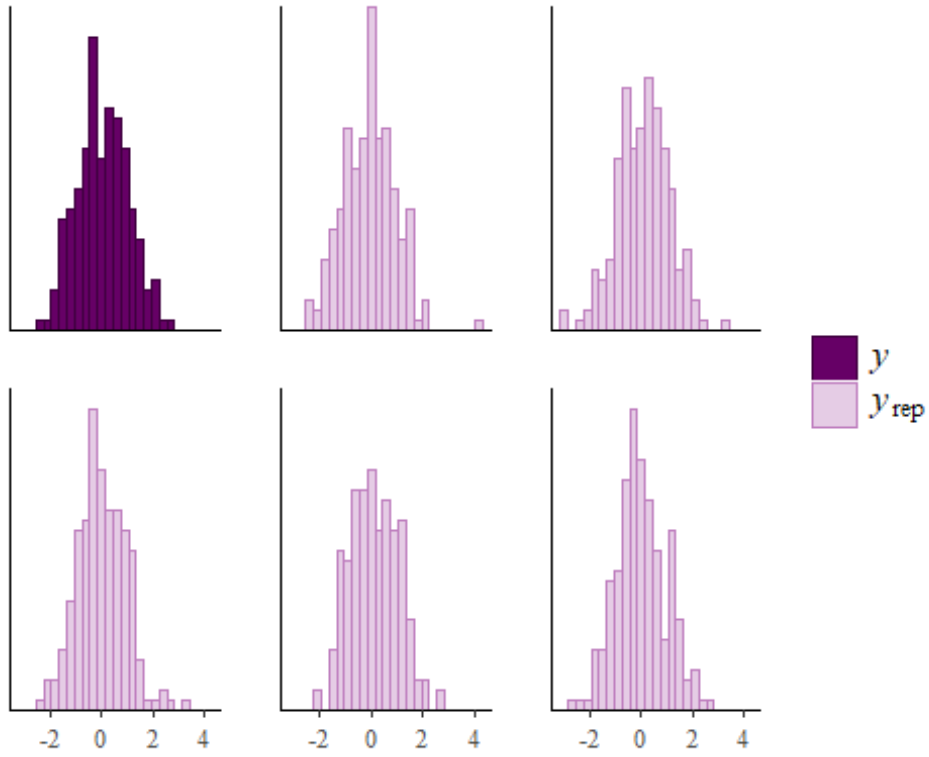
To provide an interface to **bayesplot** from your package, you can very easily define a `pp_check` method (or multiple `pp_check` methods) for the fitted model objects created by your package. All a `pp_check` method needs to do is provide the y vector and yrep matrix arguments to the various plotting functions included in **bayesplot**.

Suppose that objects of class `"foo"` are lists with named components, two of which are `y` and `yrep`. Here's a simple method `pp_check.foo` that offers the user the option of two different plots :

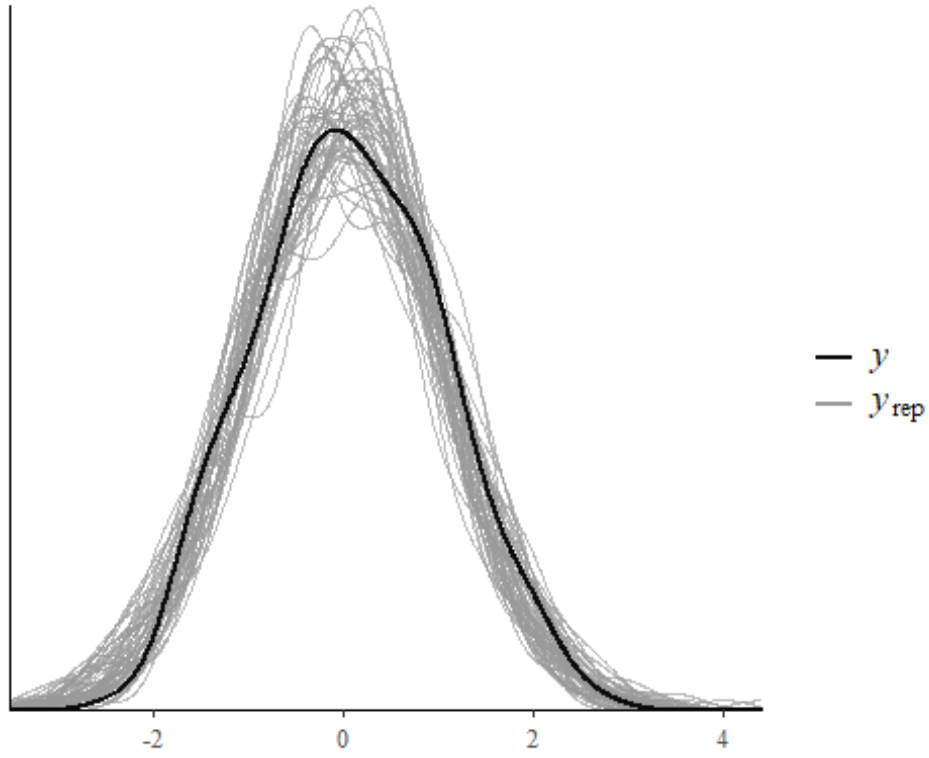
```
# @param object An object of class "foo".
# @param type The type of plot.
# @param ... Optional arguments passed on to the bayesplot plotting function.
pp_check.foo <- function(object, type = c("multiple", "overlaid"), ...) {
  type <- match.arg(type)
  y <- object[["y"]]
  yrep <- object[["yrep"]]
  stopifnot(nrow(yrep) >= 50)
  samp <- sample(nrow(yrep), size = ifelse(type == "overlaid", 50, 5))
  yrep <- yrep[samp, ]

  if (type == "overlaid") {
    ppc_dens_overlay(y, yrep, ...)
  } else {
    ppc_hist(y, yrep, ...)
  }
}
```

To try out `pp_check.foo` we can just make a list with *y* and *yrep* components and give it class foo:

```
x <- list(y = rnorm(200), yrep = matrix(rnorm(1e5), nrow = 500, ncol = 200))
class(x) <- "foo"

color_scheme_set("purple")
pp_check(x, type = "multiple", binwidth = 0.3)
```




```
color_scheme_set("darkgray")
pp_check(x, type = "overlaid")
```



4.6.1 Examples of pp_checkmethods in other packages

Several packages currently use this approach to provide an interface to bayesplot's graphical **posterior** predictive checks. See, for example, the `pp_check` methods in the **rstanarm** and **brms** packages.

5 How to use the rstanarm package

```
library(rstanarm)
help(pack = rstanarm)
rstanarm::rstanarm
```

The goal of the **rstanarm** package is to make Bayesian estimation routine for the most common regression models that applied researchers use. This will enable researchers to avoid the counter-intuitiveness of the frequentist approach to probability and statistics with only minimal changes to their existing R scripts.

The four steps of a Bayesian analysis are

1. Specify a joint distribution for the outcome(s) and all the unknowns, which typically takes the form of a marginal prior distribution for the unknowns multiplied by a likelihood for the outcome(s) conditional on the unknowns. This joint distribution is proportional to a posterior distribution of the unknowns conditional on the observed data

2. Draw from posterior distribution using Markov Chain Monte Carlo (MCMC).

3. Evaluate how well the model fits the data and possibly revise the model.

4. Draw from the posterior predictive distribution of the outcome(s) given interesting values of the predictors in order to visualize how a manipulation of a predictor affects (a function of) the outcome(s).

Step 1 is necessarily model-specific and is covered in more detail in the other vignettes that cover specific forms of the marginal prior distribution and likelihood of the outcome. It is somewhat more involved than the corresponding first step of a frequentist analysis, which only requires that the likelihood of the outcome be specified. However, the default priors in the **rstanarm** package should work well in the majority of cases. Steps 2, 3, and 4 are the focus of this vignette because they are largely not specific to how the joint distribution in Step 1 is specified.

The key concept in Step 3 and Step 4 is the posterior predictive distribution, which is the distribution of the outcome implied by the model after having used the observed data to update our beliefs about the unknown parameters. Frequentists, by definition, have no posterior predictive distribution and frequentist predictions are subtly different from what applied researchers seek. Maximum likelihood estimates do not condition on the observed outcome data and so the uncertainty in the estimates pertains to the variation in the sampling distribution of the estimator, i.e. the distribution of estimates that would occur if we could repeat the process of drawing a random sample from a well-defined population

and apply the estimator to each sample. It is possible to construct a distribution of predictions under the frequentist paradigm but it evokes the hypothetical of repeating the process of drawing a random sample, applying the estimator each time, and generating point predictions of the outcome. In contrast, the posterior predictive distribution conditions on the observed outcome data in hand to update beliefs about the unknowns and the variation in the resulting distribution of predictions reflects the remaining uncertainty in our beliefs about the unknowns.

5.1 6.1 Step 1 : Specify a Posterior Distribution

The posterior distribution — with independent priors — can be written as

$$f(\alpha, \beta_1, \beta_2 | \mathbf{y}, \mathbf{X}) \propto f(\alpha) f(\beta_1) f(\beta_2) \times \prod_{i=1}^{J} g^{-1}(\eta_i)^{y_i} (1 - g^{-1}(\eta_i))^{n_i - y_i}$$

where $\eta_i = \alpha + \beta_1 education_i + \beta_2 Female_i$ is the linear predictor and a function of an intercept α, a coefficient on the years of education β_1, and an intercept-shift β_2 for the case where the respondent is female. These data are organized such that y_i is the number of respondents who agree with the statement that have the same level of education and the same gender, and $(n_i - y_i)$ is the number of such people who disagree with the statement. The inverse link function, $p = g^{-1}(\eta_i)$, for a binomial likelihood can be one of several Cumulative Distribution Functions (CDFs) but in this case is the standard logistic CDF, $g^{-1}(\eta_i) = \frac{1}{1+e^{-\eta_i}}$.

Suppose we believe — prior to seeing the data — that α, β_1, and β_2 are probably close to zero, are as likely to be positive as they are to be negative, but have a small chance of being quite far from zero. These beliefs can be represented by Student t distributions with a few degrees of freedom in order to produce moderately heavy tails. In particular, we will specify seven degrees of freedom. Note that these purported beliefs may well be more skeptical than your actual beliefs, which are probably that women and people with more education have less conservative societal views.

5.1.1 Note on "prior beliefs" and default priors

In this vignette we use the term "prior beliefs" to refer in generality to the information content of the prior distribution (conditional on the model). Sometimes previous research on the topic of interest motivates beliefs about model parameters, but other times such work may not exist or several studies may make contradictory claims. Regardless, we nearly always have some knowledge that should be reflected in our choice of prior distributions. For example, no one believes a logistic regression coefficient will be greater than five in absolute value if the predictors are scaled reasonably. You may also have seen examples of so-called "non-informative" (or "vague", "diffuse", etc.) priors like a normal distribution with a variance of 1000. When data are reasonably scaled, these priors are almost always a bad idea for various reasons (they give non-trivial weight to extreme values, reduce computational efficiency, etc). The default priors in **rstanarm** are designed to be weakly informative, by which we mean that they avoid placing unwarranted prior

weight on nonsensical parameter values and provide some regularization to avoid overfitting, but also do allow for extreme values if warranted by the data. If additional information is available, the weakly informative defaults can be replaced with more informative priors.

5.2 Step 2: Draw from the posterior distribution

The likelihood for the sample is just the product over the J groups of

$$g^{-1}(\eta_i)^{y_i}(1 - g^{-1}(\eta_i))^{n_i - y_i},$$

which can be maximized over α, β_1, and β_2 to obtain frequentist estimates by calling

```
Data from a survey from 1974 / 1975 asking both female and male responders
about their opinion on the statement: Women should take care of running their
homes and leave running the country up to men.

data("womensrole")

education : years of education.

gender : a factor with levels Male and Female.

agree : number of subjects in agreement with the statement.
disagree : number of subjects in disagreement with the statement.
```

$$\eta_i = \alpha + \beta_1 Education_i + \beta_2 Female_i$$

```
# Load the require libraries
library(rstanarm)
library(ggplot2)
library(bayesplot)

# Set the theme
theme_set(bayesplot::theme_default())
```

5.3 Draw from posterior distribution

```
# Load the data
data("womensrole", package = "HSAUR3")

# Add a column
womensrole$total<- womensrole$agree + womensrole$disagree

# Head of data
head(womensrole)

##   education gender agree disagree total
## 1         0   Male     4        2     6
## 2         1   Male     2        0     2
## 3         2   Male     4        0     4
```

```
## 4         3   Male     6        3     9
## 5         4   Male     5        5    10
## 6         5   Male    13        7    20

# Fit the glm model (frequentist)
womensrole_glm_1<- glm(cbind(agree, disagree) ~ education + gender,
                       data = womensrole, family = binomial(link = "logit"))

round(coef(summary(womensrole_glm_1)), 3)

##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)     2.509      0.184  13.646    0.000
## education      -0.271      0.015 -17.560    0.000
## genderFemale   -0.011      0.084  -0.136    0.892
```

The p-value for the null hypothesis that $\beta_1 = 0$ is very small, while the p-value for the null hypothesis that $\beta_2 = 0$ is very large. However, frequentist p-values are awkward because they do not pertain to the probability that a scientific hypothesis is true but rather to the probability of observing a z-statistic that is so large (in magnitude) if the null hypothesis were true. The desire to make probabilistic statements about a scientific hypothesis is one reason why many people are drawn to the Bayesian approach.

A model with the same likelihood but Student t priors with seven degrees of freedom can be specified using the **rstanarm** package in a similar way by prepending `stan_` to the `glm` call and specifying priors (and optionally the number of cores on your computer to utilize) :

```
# Fit the bayesian glm model
womensrole_bglm_1<- stan_glm(cbind(agree, disagree) ~ education + gender,
                             data = womensrole,
                             family = binomial(link = "logit"),
                             prior = student_t(df = 7, 0, 5),
                             prior_intercept = student_t(df = 7, 0, 5),
                             cores = 2, seed = 12345)

womensrole_bglm_1

## stan_glm
##  family:       binomial [logit]
##  formula:      cbind(agree, disagree) ~ education + gender
##  observations: 42
##  predictors:   3
## ------
##              Median MAD_SD
## (Intercept)   2.5    0.2
## education    -0.3    0.0
## genderFemale  0.0    0.1
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

As can be seen, the "Bayesian point estimates" — which are represented by the posterior medians — are very similar to the maximum likelihood estimates. Frequentists would ask whether the point estimate is greater in magnitude than double the estimated standard deviation of the sampling distribution. But here we simply have estimates of the standard deviation of the marginal posterior distributions, which are based on a scaling of the Median Absolute Deviation (MAD) from the posterior medians to obtain a robust estimator of the posterior standard deviation. In addition, we can use the `posterior_interval` function to obtain a Bayesian uncertainty interval for β_1 :

```
ci95 <- posterior_interval(womensrole_bglm_1, prob = 0.95, pars =
"education")
round(ci95, 2)

##           2.5% 97.5%
## education -0.3 -0.24
```

Unlike frequentist confidence intervals — which are not interpretable in terms of post-data probabilities — the Bayesian uncertainty interval indicates we believe after seeing the data that there is a 0.95 probability that β2 is between `ci95[1,1]` and `ci95[1,2]`. Alternatively, we could say that there is essentially zero probability that $\beta_2 > 0$, although frequentists cannot make such a claim coherently.

Many of the post-estimation methods that are available for a model that is estimated by `glm` are also available for a model that is estimated by `stan_glm`. For example,

```
cbind(Median = coef(womensrole_bglm_1), MAD_SD = se(womensrole_bglm_1))

##                    Median     MAD_SD
## (Intercept)   2.52098276 0.18285768
## education    -0.27153061 0.01556542
## genderFemale -0.01262136 0.08463091

# not deviance residuals
summary(residuals(womensrole_bglm_1))

##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
NA's
## -0.3076575 -0.0359870 -0.0041319 -0.0003265  0.0660755  0.2822688
1

# Variance-Covariance Matrix
cov2cor(vcov(womensrole_bglm_1))

##              (Intercept)   education genderFemale
## (Intercept)    1.0000000 -0.93963167  -0.23059559
## education     -0.9396317  1.00000000  -0.02463045
## genderFemale  -0.2305956 -0.02463045   1.00000000
```

rstanarm does provide a confint method, although it is reserved for computing confidence intervals in the case that the user elects to estimate a model by (penalized) maximum likelihood. When using full Bayesian inference (the **rstanarm** default) or approximate

Bayesian inference the `posterior_interval` function should be used to obtain Bayesian uncertainty intervals.

5.4 Step 3: Criticize the model

The `launch_shinystan` function in the **shinystan** package provides almost all the tools you need to visualize the posterior distribution and diagnose any problems with the Markov chains. In this case, the results are fine and to verify that, you can call

```
launch_shinystan(womensrole_bglm_1, ppd = FALSE)
```

which will open a web browser that drives the visualizations.

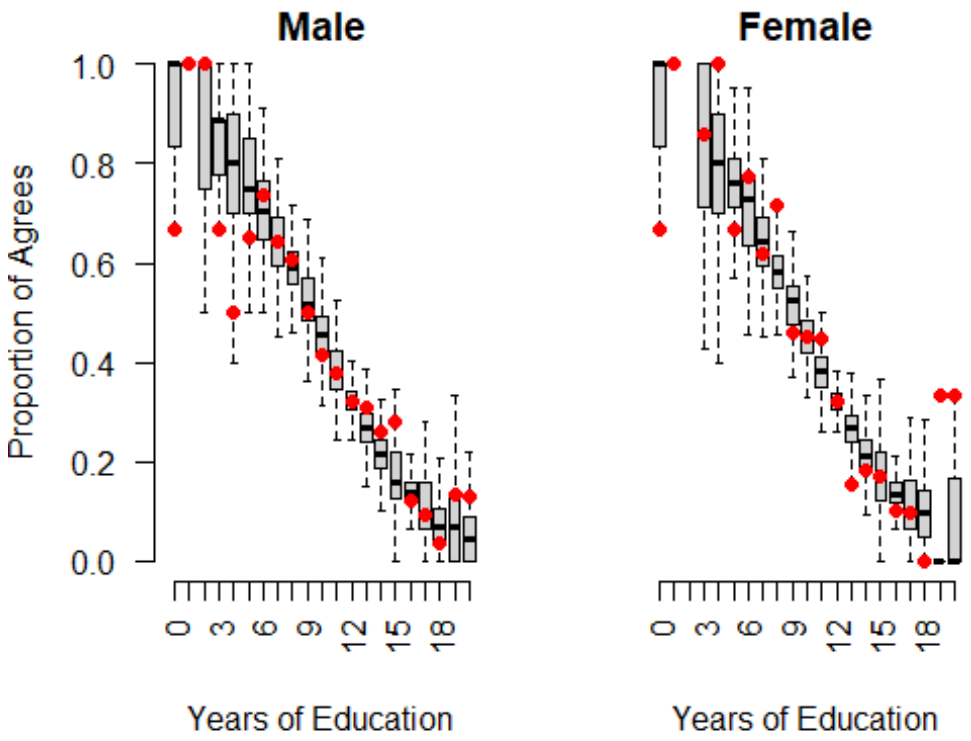
For the rest of this subsection, we focus on what users can do programmatically to evaluate whether a model is adequate. A minimal requirement for Bayesian estimates is that the model should fit the data that the estimates conditioned on. The key function here is `posterior_predict`, which can be passed a new `data.frame` to predict out-of-sample, but in this case is omitted to obtain in-sample posterior predictions :

```
y_rep <- posterior_predict(womensrole_bglm_1)
dim(y_rep)

## [1] 4000   42
```

The resulting matrix has rows equal to the number of posterior simulations, which in this case is 2000 and columns equal to the number of observations in the original dataset, which is 42 combinations of education and gender. Each element of this matrix is a predicted number of respondents with that value of education and gender who agreed with the survey question and thus should be reasonably close to the observed proportion of agreements in the data. We can create a plot to check this:

```
par(mfrow = 1:2, mar = c(5,3.7,1,0) + 0.1, las = 3)
boxplot(sweep(y_rep[,womensrole$gender == "Male"], 2, STATS =
                womensrole$total[womensrole$gender == "Male"], FUN = "/"),
        axes = FALSE, main = "Male", pch = NA,
        xlab = "Years of Education", ylab = "Proportion of Agrees")
with(womensrole, axis(1, at = education[gender == "Male"] + 1,
                      labels = 0:20))

axis(2, las = 1)
with(womensrole[womensrole$gender == "Male",],
     points(education + 1,  agree / (agree + disagree),
            pch = 16, col = "red"))
boxplot(sweep(y_rep[,womensrole$gender == "Female"], 2, STATS =
          womensrole$total[womensrole$gender == "Female"], FUN = "/"),
          axes = FALSE, main = "Female", pch = NA,
        xlab = "Years of Education", ylab = "")
with(womensrole, axis(1, at = education[gender == "Female"] + 1,
     labels = 0:20))
with(womensrole[womensrole$gender == "Female",],
```

```
    points(education + 1,  agree / (agree + disagree),
           pch = 16, col = "red"))
```



Here the boxplot provide the *median, IQR* and the *hinges* of the posterior predictive for a gender and level of education, while the red points represent the corresponding observed data, As can we seen the model the observed dat fairly well for to years education but predicts less well for very low or very high levels of education, where there are less data.

Consequently we might consider a modal where education has a quadratic effect on agreement, which is easy to specify using R formula syntax.

```
(womensrole_bglm_2 <- update(womensrole_bglm_1, formula. = . ~ . +
I(education^2)))
```

```
## stan_glm
##  family:       binomial [logit]
##  formula:      cbind(agree, disagree) ~ education + gender +
I(education^2)
##  observations: 42
##  predictors:   4
## ------
##                Median MAD_SD
## (Intercept)     2.1    0.4
## education      -0.2    0.1
## genderFemale    0.0    0.1
## I(education^2)  0.0    0.0
##
```

```
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

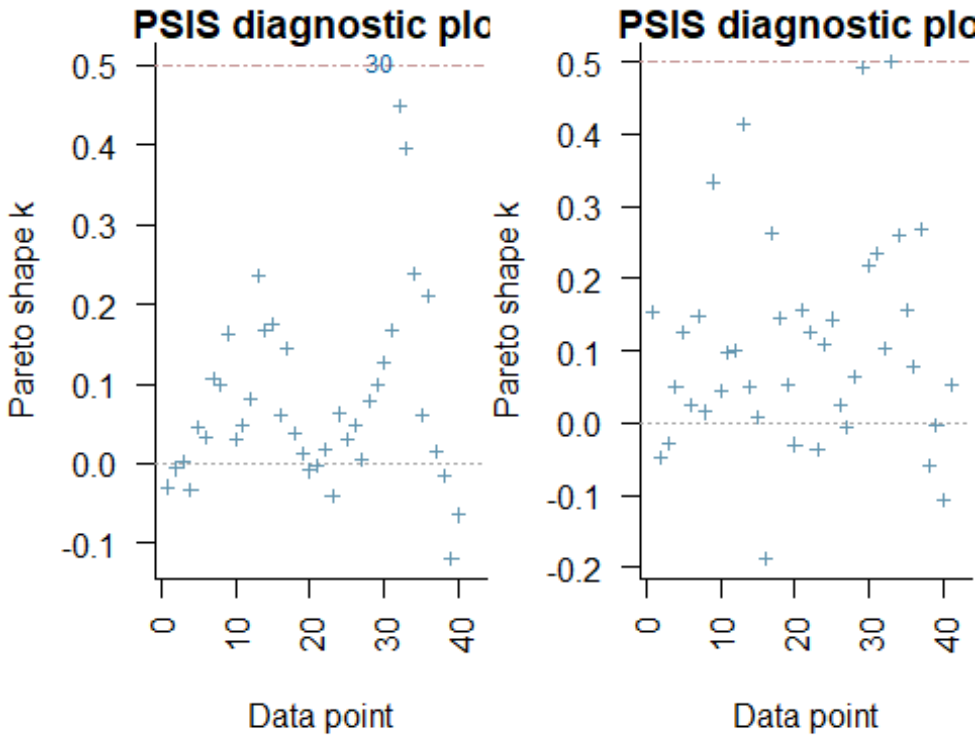
Frequentist would test the Null Hypothesis that the coeff of squared lavel of education is 0. Bayesian might also ask whether such a model is expected to produce better out of sample prediction than a model with only level of education.

The latter question can be answered using leave-one-out cross-validation or the approximation thereof provided by the loo function in the loo package, for which a method is provided by the **rstanarm** package.

```
require(loo)
loo_bglm_1<- loo(womensrole_bglm_1)

## Warning: Found 1 observation(s) with a pareto_k > 0.7. We recommend
calling 'loo' again with argument 'k_threshold = 0.7' in order to calculate
the ELPD without the assumption that these observations are negligible. This
will refit the model 1 times to compute the ELPDs for the problematic
observations directly.

loo_bglm_2<- loo(womensrole_bglm_2)

## Warning: Found 1 observation(s) with a pareto_k > 0.7. We recommend
calling 'loo' again with argument 'k_threshold = 0.7' in order to calculate
the ELPD without the assumption that these observations are negligible. This
will refit the model 1 times to compute the ELPDs for the problematic
observations directly.
```

First we verify that posterior is not to sensitive to any particular observation in the dataset.

```
par(mfrow = 1:2, mar = c(5,3.8,1,0) + 0.1, las = 3)
plot(loo_bglm_1, label_points = TRUE)

## Warning in plot.psis_loo(loo_bglm_1, label_points = TRUE): 2.4% of Pareto
k estimates are Inf/NA/NaN
## and not plotted.

plot(loo_bglm_2, label_points = TRUE)

## Warning in plot.psis_loo(loo_bglm_2, label_points = TRUE): 2.4% of Pareto
k estimates are Inf/NA/NaN
## and not plotted.
```



There are only one or two moderate outliers (whose statistics are greater than 0.5), which should not have too much of an effect on the resulting model comparison :

```
compare_models(loo_bglm_1, loo_bglm_2)

## Warning: 'compare_models' is deprecated.
## Use 'loo_compare' instead.
## See help("Deprecated")

## Warning: 'loo::compare' is deprecated.
## Use 'loo_compare' instead.
## See help("Deprecated")

## Model formulas:
##  :  NULL
##  :  NULLelpd_diff        se
##       -0.7       1.7

loo_compare(loo_bglm_1, loo_bglm_2)

##                    elpd_diff se_diff
## womensrole_bglm_1  0.0       0.0
## womensrole_bglm_2 -0.7       1.7
```

In this case there is little difference in the expected log pointwise deviance between the two models, so we are essentially indifferent between them after taking into a count that second model estimates an additional parameter. The “LOO Information Criterion (LOOIC)”

```
loo_bglm_1

##
## Computed from 4000 by 42 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -104.8  9.5
## p_loo         4.2  1.7
## looic       209.7 18.9
## ------
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     40    95.2%   1532
##  (0.5, 0.7]   (ok)        1     2.4%   209
##    (0.7, 1]   (bad)       0     0.0%   <NA>
##    (1, Inf)   (very bad)  1     2.4%   2000
## See help('pareto-k-diagnostic') for details.
```

Frequentists use AIC for diagonistic which ignores the prior and assumes that posterior distribution is multivariate normal, whereas the functions from the **loop** package used here do not assume that the posterior distribution is multivariate normal and integrate over uncertainty in the parameters. This only assumes that any one observation can be omitted without having a major effect on the posterior distribution, which can be judged using the plots above.

5.5 Step 4: Analyze manipulations of predictors

Frequentists attempt to interpret the estimates of the model, which is difficult except when the model is linear, has no inverse link function, and contains no interaction terms. Bayesians can avoid this difficulty simply by inspecting the posterior predictive distribution at different levels of the predictors. For example,

```
# note: in newdata we want agree and disagree to sum to the number of people
we
# want to predict for. the values of agree and disagree don't matter so long
as
# their sum is the desired number of trials. we need to explicitly imply the
# number of trials like this because our original data are aggregate. if we
had
# bernoulli data then it would be a given we wanted to predict for single
# individuals.
newdata <- data.frame(agree = c(0,0), disagree = c(100,100), education =
c(12,16),
                      gender = factor("Female", levels = c("Male",
"Female")))

y_rep <- posterior_predict(womensrole_bglm_2, newdata)
summary(apply(y_rep, 1, diff))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   -41.0   -24.0   -20.0   -19.7   -16.0     3.0
```

As can be seen, out of 100 women who have a college degree versus 100 women with only a high school degree, we would expect about 20 fewer college-educated women to agree with the question. There is an even chance that the difference is between 24 and 16, a one-in-four chance that it is greater, and one-in-four chance that it is less.

5.6 Troubleshooting

This section provides suggestions for how to proceed when you encounter warning messages generated by the modeling functions in the **rstanarm** package. The example models below are used just for the purposes of concisely demonstrating certain difficulties and possible remedies (we won't worry about the merit of the models themselves). The references at the end provide more information on the relevant issues.

5.6.1 Markov Chains didn't converge

Recommendation : Run the chains for more iterations.

By default, all **rstanarm** modeling functions will run four randomly initialized Markov chains, each for 2000 iterations (including a warmup period of 1000 iterations that is discarded). All chains must converge to the target distribution for inferences to be valid. For most models, the default settings are sufficient, but if you see a warning message about Markov chains not converging, the first thing to try is increasing the number of iterations. This can be done by specifying the `iter` argument (e.g. `iter = 3000`).

One way to monitor whether a chain has converged to the equilibrium distribution is to compare its behavior to other randomly initialized chains. This is the motivation for the Gelman and Rubin potential scale reduction statistic Rhat. The Rhat statistic measures the ratio of the average variance of the draws within each chain to the variance of the pooled draws across chains; if all chains are at equilibrium, these will be the same and Rhat will be one. If the chains have not converged to a common distribution, the Rhat statistic will tend to be greater than one.

Gelman and Rubin's recommendation is that the independent Markov chains be initialized with diffuse starting values for the parameters and sampled until all values for *Rhat* are below 1.1. When any Rhat values are above 1.1 rstanarm will print a warning message like this:

```
Markov chains did not converge! Do not analyze results!
```

To illustrate how to check the Rhat values after fitting a model using **rstanarm** we'll fit two models and run them for different numbers of iterations.

```
bad_rhat<- stan_glm(mpg~. , data = mtcars, iter = 20)

##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 1).
## Chain 1:
```

```
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would
take 0 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: WARNING: No variance estimation is
## Chain 1:          performed for num_warmup < 20
## Chain 1:
## Chain 1: Iteration:  1 / 20 [  5%]  (Warmup)
## Chain 1: Iteration:  2 / 20 [ 10%]  (Warmup)
## Chain 1: Iteration:  4 / 20 [ 20%]  (Warmup)
## Chain 1: Iteration:  6 / 20 [ 30%]  (Warmup)
## Chain 1: Iteration:  8 / 20 [ 40%]  (Warmup)
## Chain 1: Iteration: 10 / 20 [ 50%]  (Warmup)
## Chain 1: Iteration: 11 / 20 [ 55%]  (Sampling)
## Chain 1: Iteration: 12 / 20 [ 60%]  (Sampling)
## Chain 1: Iteration: 14 / 20 [ 70%]  (Sampling)
## Chain 1: Iteration: 16 / 20 [ 80%]  (Sampling)
## Chain 1: Iteration: 18 / 20 [ 90%]  (Sampling)
## Chain 1: Iteration: 20 / 20 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.002 seconds (Warm-up)
## Chain 1:                0.002 seconds (Sampling)
## Chain 1:                0.004 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would
take 0 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: WARNING: No variance estimation is
## Chain 2:          performed for num_warmup < 20
## Chain 2:
## Chain 2: Iteration:  1 / 20 [  5%]  (Warmup)
## Chain 2: Iteration:  2 / 20 [ 10%]  (Warmup)
## Chain 2: Iteration:  4 / 20 [ 20%]  (Warmup)
## Chain 2: Iteration:  6 / 20 [ 30%]  (Warmup)
## Chain 2: Iteration:  8 / 20 [ 40%]  (Warmup)
## Chain 2: Iteration: 10 / 20 [ 50%]  (Warmup)
## Chain 2: Iteration: 11 / 20 [ 55%]  (Sampling)
## Chain 2: Iteration: 12 / 20 [ 60%]  (Sampling)
## Chain 2: Iteration: 14 / 20 [ 70%]  (Sampling)
## Chain 2: Iteration: 16 / 20 [ 80%]  (Sampling)
## Chain 2: Iteration: 18 / 20 [ 90%]  (Sampling)
## Chain 2: Iteration: 20 / 20 [100%]  (Sampling)
```

```
## Chain 2:
## Chain 2:  Elapsed Time: 0.008 seconds (Warm-up)
## Chain 2:                0.001 seconds (Sampling)
## Chain 2:                0.009 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would
take 0 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: WARNING: No variance estimation is
## Chain 3:          performed for num_warmup < 20
## Chain 3:
## Chain 3: Iteration:  1 / 20 [  5%]  (Warmup)
## Chain 3: Iteration:  2 / 20 [ 10%]  (Warmup)
## Chain 3: Iteration:  4 / 20 [ 20%]  (Warmup)
## Chain 3: Iteration:  6 / 20 [ 30%]  (Warmup)
## Chain 3: Iteration:  8 / 20 [ 40%]  (Warmup)
## Chain 3: Iteration: 10 / 20 [ 50%]  (Warmup)
## Chain 3: Iteration: 11 / 20 [ 55%]  (Sampling)
## Chain 3: Iteration: 12 / 20 [ 60%]  (Sampling)
## Chain 3: Iteration: 14 / 20 [ 70%]  (Sampling)
## Chain 3: Iteration: 16 / 20 [ 80%]  (Sampling)
## Chain 3: Iteration: 18 / 20 [ 90%]  (Sampling)
## Chain 3: Iteration: 20 / 20 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.003 seconds (Warm-up)
## Chain 3:                0.002 seconds (Sampling)
## Chain 3:                0.005 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would
take 0 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: WARNING: No variance estimation is
## Chain 4:          performed for num_warmup < 20
## Chain 4:
## Chain 4: Iteration:  1 / 20 [  5%]  (Warmup)
## Chain 4: Iteration:  2 / 20 [ 10%]  (Warmup)
## Chain 4: Iteration:  4 / 20 [ 20%]  (Warmup)
## Chain 4: Iteration:  6 / 20 [ 30%]  (Warmup)
```

```
## Chain 4: Iteration:  8 / 20 [ 40%]  (Warmup)
## Chain 4: Iteration: 10 / 20 [ 50%]  (Warmup)
## Chain 4: Iteration: 11 / 20 [ 55%]  (Sampling)
## Chain 4: Iteration: 12 / 20 [ 60%]  (Sampling)
## Chain 4: Iteration: 14 / 20 [ 70%]  (Sampling)
## Chain 4: Iteration: 16 / 20 [ 80%]  (Sampling)
## Chain 4: Iteration: 18 / 20 [ 90%]  (Sampling)
## Chain 4: Iteration: 20 / 20 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.002 seconds (Warm-up)
## Chain 4:                0.049 seconds (Sampling)
## Chain 4:                0.051 seconds (Total)
## Chain 4:

## Warning: There were 4 divergent transitions after warmup. See
## https://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.

## Warning: There were 1 chains where the estimated Bayesian Fraction of
Missing Information was low. See
## https://mc-stan.org/misc/warnings.html#bfmi-low

## Warning: Examine the pairs() plot to diagnose sampling problems

## Warning: The largest R-hat is 3.37, indicating chains have not mixed.
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#r-hat

## Warning: Bulk Effective Samples Size (ESS) is too low, indicating
posterior means and medians may be unreliable.
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#bulk-ess

## Warning: Tail Effective Samples Size (ESS) is too low, indicating
posterior variances and tail quantiles may be unreliable.
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#tail-ess

## Warning: Markov chains did not converge! Do not analyze results!

good_rhat<- stan_glm(mpg ~. , data = mtcars, iter = 500)

##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.001 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would
take 10 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 500 [  0%]  (Warmup)
```

```
## Chain 1: Iteration:  50 / 500 [ 10%]  (Warmup)
## Chain 1: Iteration: 100 / 500 [ 20%]  (Warmup)
## Chain 1: Iteration: 150 / 500 [ 30%]  (Warmup)
## Chain 1: Iteration: 200 / 500 [ 40%]  (Warmup)
## Chain 1: Iteration: 250 / 500 [ 50%]  (Warmup)
## Chain 1: Iteration: 251 / 500 [ 50%]  (Sampling)
## Chain 1: Iteration: 300 / 500 [ 60%]  (Sampling)
## Chain 1: Iteration: 350 / 500 [ 70%]  (Sampling)
## Chain 1: Iteration: 400 / 500 [ 80%]  (Sampling)
## Chain 1: Iteration: 450 / 500 [ 90%]  (Sampling)
## Chain 1: Iteration: 500 / 500 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.137 seconds (Warm-up)
## Chain 1:                0.09 seconds (Sampling)
## Chain 1:                0.227 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would
take 0 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:   1 / 500 [  0%]  (Warmup)
## Chain 2: Iteration:  50 / 500 [ 10%]  (Warmup)
## Chain 2: Iteration: 100 / 500 [ 20%]  (Warmup)
## Chain 2: Iteration: 150 / 500 [ 30%]  (Warmup)
## Chain 2: Iteration: 200 / 500 [ 40%]  (Warmup)
## Chain 2: Iteration: 250 / 500 [ 50%]  (Warmup)
## Chain 2: Iteration: 251 / 500 [ 50%]  (Sampling)
## Chain 2: Iteration: 300 / 500 [ 60%]  (Sampling)
## Chain 2: Iteration: 350 / 500 [ 70%]  (Sampling)
## Chain 2: Iteration: 400 / 500 [ 80%]  (Sampling)
## Chain 2: Iteration: 450 / 500 [ 90%]  (Sampling)
## Chain 2: Iteration: 500 / 500 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.146 seconds (Warm-up)
## Chain 2:                0.131 seconds (Sampling)
## Chain 2:                0.277 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would
take 0 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
```

```
## Chain 3:
## Chain 3: Iteration:   1 / 500 [  0%]  (Warmup)
## Chain 3: Iteration:  50 / 500 [ 10%]  (Warmup)
## Chain 3: Iteration: 100 / 500 [ 20%]  (Warmup)
## Chain 3: Iteration: 150 / 500 [ 30%]  (Warmup)
## Chain 3: Iteration: 200 / 500 [ 40%]  (Warmup)
## Chain 3: Iteration: 250 / 500 [ 50%]  (Warmup)
## Chain 3: Iteration: 251 / 500 [ 50%]  (Sampling)
## Chain 3: Iteration: 300 / 500 [ 60%]  (Sampling)
## Chain 3: Iteration: 350 / 500 [ 70%]  (Sampling)
## Chain 3: Iteration: 400 / 500 [ 80%]  (Sampling)
## Chain 3: Iteration: 450 / 500 [ 90%]  (Sampling)
## Chain 3: Iteration: 500 / 500 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.115 seconds (Warm-up)
## Chain 3:                0.107 seconds (Sampling)
## Chain 3:                0.222 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would
take 0 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:   1 / 500 [  0%]  (Warmup)
## Chain 4: Iteration:  50 / 500 [ 10%]  (Warmup)
## Chain 4: Iteration: 100 / 500 [ 20%]  (Warmup)
## Chain 4: Iteration: 150 / 500 [ 30%]  (Warmup)
## Chain 4: Iteration: 200 / 500 [ 40%]  (Warmup)
## Chain 4: Iteration: 250 / 500 [ 50%]  (Warmup)
## Chain 4: Iteration: 251 / 500 [ 50%]  (Sampling)
## Chain 4: Iteration: 300 / 500 [ 60%]  (Sampling)
## Chain 4: Iteration: 350 / 500 [ 70%]  (Sampling)
## Chain 4: Iteration: 400 / 500 [ 80%]  (Sampling)
## Chain 4: Iteration: 450 / 500 [ 90%]  (Sampling)
## Chain 4: Iteration: 500 / 500 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.132 seconds (Warm-up)
## Chain 4:                0.096 seconds (Sampling)
## Chain 4:                0.228 seconds (Total)
## Chain 4:

## Warning: Bulk Effective Samples Size (ESS) is too low, indicating
posterior means and medians may be unreliable.
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#bulk-ess
```

Here the first model leads to the warning message about convergence but the second model does not. Indeed, we can see that many Rhat values are much bigger than 1 for the first model :

```
rhat_b<- summary(bad_rhat)[, "Rhat"]
rhat_b
```

```
##   (Intercept)           cyl          disp            hp          drat
wt          qsec
##     0.9324166     0.9478906     1.1394097     0.9794994     0.9323501
1.4949257     1.1296911
##            vs            am          gear          carb         sigma
mean_PPD log-posterior
##     1.0949525     1.0605379     0.9288911     1.0113121     3.1920901
2.4149361     9.1135879
```

```
rhat_b[rhat_b > 1.1]
```

```
##          disp            wt          qsec         sigma      mean_PPD log-
posterior
##      1.139410      1.494926      1.129691      3.192090      2.414936
9.113588
```

Since we didn't get a warning for the second model we shouldn't find any parameters with an Rhat far from 1:

```
rhat_g<- summary(good_rhat)[, "Rhat"]
rhat_g
```

```
##   (Intercept)           cyl          disp            hp          drat
wt          qsec
##     1.0069948     1.0021167     1.0185325     1.0042223     1.0029628
1.0239669     1.0103576
##            vs            am          gear          carb         sigma
mean_PPD log-posterior
##     1.0013535     1.0016348     1.0053044     1.0213740     0.9996486
1.0027922     1.0034404
```

```
any(summary(good_rhat)[, "Rhat"] > 1.1)
```

```
## [1] FALSE
```

5.6.2 Divergent transitions

Recommendation : increase the target acceptance rate `adapt_delta`.

Hamiltonian Monte Carlo (HMC), the MCMC algorithm used by Stan, works by simulating the evolution of a Hamiltonian system. Stan uses a symplectic integrator to approximate the exact solution of the Hamiltonian dynamics. When the step size parameter is too large relative to the curvature of the log posterior this approximation can diverge and threaten the validity of the sampler. **rstanarm** will print a warning if there are any divergent transitions after the warmup period, in which case the posterior sample may be biased. The

recommended method is to increase the `adapt_delta` parameter – target average proposal acceptance probability in the adaptation – which will in turn reduce the step size. Each of the modeling functions accepts an `adapt_delta` argument, so to increase `adapt_delta` you can simply change the value from the default value to a value closer to 1. To reduce the frequency with which users need to manually set `adapt_delta`, the default value depends on the prior distribution used (see `help("adapt_delta", package = "rstanarm")` for details).

The downside to increasing the target acceptance rate – and, as a consequence, decreasing the step size – is that sampling will tend to be slower. Intuitively, this is because a smaller step size means that more steps are required to explore the posterior distribution. Since the validity of the estimates is not guaranteed if there are any post-warmup divergent transitions, the slower sampling is a minor cost.

5.6.3 Maximum treedepth exceeded

Recommendation : increase the maximum allowed treedepth `max_treedepth`.

Configuring the No-U-Turn-Sampler (the variant of HMC used by Stan) involves putting a cap on the depth of the trees that it evaluates during each iteration. This is controlled through a maximum depth parameter `max_treedepth`. When the maximum allowed tree depth is reached it indicates that NUTS is terminating prematurely to avoid excessively long execution time. If **rstanarm** prints a warning about transitions exceeding the maximum treedepth you should try increasing the `max_treedepth` parameter using the optional `control` argument. For example, to increase `max_treedepth` to 20 (the default used **rstanarm** is 15) you can provide the argument `control = list(max_treedepth = 20)` to any of the **rstanarm** modeling functions. If you do not see a warning about hitting the maximum treedepth (which is rare), then you do not need to worry.

5.7 Conclusion

In this vignette, we have gone through the four steps of a Bayesian analysis. The first step — specifying the posterior distribution — varies considerably from one analysis to the next because the likelihood function employed differs depending on the nature of the outcome variable and our prior beliefs about the parameters in the model varies not only from situation to situation but from researcher to researcher. However, given a posterior distribution and given that this posterior distribution can be drawn from using the **rstanarm** package, the remaining steps are conceptually similar across analyses. The key is to draw from the posterior predictive distribution of the outcome, which is the what the model predicts the outcome to be after having updated our beliefs about the unknown parameters with the observed data. Posterior predictive distributions can be used for model checking and for making inferences about how manipulations of the predictors would affect the outcome.

Of course, all of this assumes that you have obtained draws from the posterior distribution faithfully. The functions in the **rstanarm** package will throw warnings if there is evidence that the draws are tainted, and we have discussed some steps to remedy these problems.

For the most part, the model-fitting functions in the **rstanarm** package are unlikely to produce many such warnings, but they may appear in more complicated models.

If the posterior distribution that you specify in the first step cannot be sampled from using the **rstanarm** package, then it is often possible to create a hand-written program in the the Stan language so that the posterior distribution can be drawn from using the **rstan** package. See the documentation for the **rstan** package or https://mc-stan.org for more details about this more advanced usage of Stan. However, many relatively simple models can be fit using the **rstanarm** package without writing any code in the Stan language, which is illustrated for each estimating function in the **rstanarm** package in the other vignettes.

6 Estimating Generalized Linear Models for Count Data with rstanarm

```
library(rstanarm)
help(pack = rstanarm)
rstanarm::count

library(ggplot2)
library(bayesplot)
theme_set(bayesplot::theme_default())
```

6.1 Introduction

This vignette explains how to estimate generalized linear models (GLMs) for count data using the `stan_glm` function in the **rstanarm** package.

The four steps of a Bayesian analysis are

1. Specify a joint distribution for the outcome(s) and all the unknowns, which typically takes the form of a marginal prior distribution for the unknowns multiplied by a likelihood for the outcome(s) conditional on the unknowns. This joint distribution is proportional to a posterior distribution of the unknowns conditional on the observed data
2. Draw from posterior distribution using Markov Chain Monte Carlo (MCMC).
3. Evaluate how well the model fits the data and possibly revise the model.
4. Draw from the posterior predictive distribution of the outcome(s) given interesting values of the predictors in order to visualize how a manipulation of a predictor affects (a function of) the outcome(s).

Steps 3 and 4 are covered in more depth by the vignette entitled "How to Use the **rstanarm** Package". This vignette focuses on Step 1 for Poisson and negative binomial regression models using the `stan_glm` function.

6.2 Likelihood

If the outcome for a single observation y is assumed to follow a Poisson distribution, the likelihood for one observation can be written as a conditionally Poisson PMF

$$\frac{1}{y!}\lambda^y e^{-\lambda}$$

where $\lambda = E(y|\mathbf{x}) = g^{-1}(\eta)$ and $\eta = \alpha + \mathbf{x}^\top\boldsymbol{\beta}$ is a linear predictor. For the Poisson distribution it is also true that $\lambda = Var(y|\mathbf{x})$, i.e. the mean and variance are both λ. Later in this vignette we also show how to estimate a negative binomial regression, which relaxes this assumption of equal conditional mean and variance of y.

Because the rate parameter λ must be positive, for a Poisson GLM the link function g maps between the positive real numbers $\mathbb{R}^+$ (the support of λ) and the set of all real numbers $\mathbb{R}$. When applied to a linear predictor η with values in $\mathbb{R}$, the inverse link function $g^{-1}(\eta)$ therefore returns a positive real number.

Although other link functions are possible, the canonical link function for a Poisson GLM is the log link $g(x) = \ln(x)$. With the log link, the inverse link function is simply the exponential function and the likelihood for a single observation becomes

$$\frac{g^{-1}(\eta)^y}{y!}e^{-g^{-1}(\eta)} = \frac{e^{\eta y}}{y!}e^{-e^{\eta}}$$

Priors A full Bayesian analysis requires specifying prior distributions $f(\alpha)$ and $f(\boldsymbol{\beta})$ for the intercept and vector of regression coefficients. When using `stan_glm`, these distributions can be set using the `prior_intercept` and `prior` arguments. The `stan_glm` function supports a variety of prior distributions, which are explained in the **rstanarm** documentation (`help(priors, package = 'rstanarm')`.

As an example, suppose we have K predictors and believe — prior to seeing the data — that $\alpha, \beta_1, \ldots, \beta_K$ are as likely to be positive as they are to be negative, but are highly unlikely to be far from zero. These beliefs can be represented by normal distributions with mean zero and a small scale (standard deviation). To give α and each of the βs this prior (with a scale of 1, say), in the call to `stan_glm` we would include the arguments `prior_intercept = normal(0,1)` and `prior = normal(0,1)`.

If, on the other hand, we have less a priori confidence that the parameters will be close to zero then we could use a larger scale for the normal distribution and/or a distribution with heavier tails than the normal like the Student t distribution. **Step 1** in the "How to Use the **rstanarm** Package" vignette discusses one such example.

6.3 Posterior

With independent prior distributions, the joint posterior distribution for α and $\boldsymbol{\beta}$ in the Poisson model is proportional to the product of the priors and the N likelihood contributions:

$$f(\alpha, \boldsymbol{\beta}|\mathbf{y}, \mathbf{X}) \propto f(\alpha) \times \prod_{k=1}^{K} f\left(\beta_k\right) \times \prod_{i=1}^{N} \frac{g^{-1}(\eta_i)^{y_i}}{y_i!} e^{-g^{-1}(\eta_i)}$$

This is posterior distribution that `stan_glm` will draw from when using MCMC.

6.4 Poisson and Negative Binomial Regression Example

This example comes from Chapter 8.3 of Gelman and Hill (2007).

We want to make inferences about the efficacy of a certain pest management system at reducing the number of roaches in urban apartments. Here is how Gelman and Hill describe the experiment (pg. 161):

> […] the treatment and control were applied to 160 and 104 apartments, respectively, and the outcome measurement y_i in each apartment i was the number of roaches caught in a set of traps. Different apartments had traps for different numbers of days […]

In addition to an intercept, the regression predictors for the model are the pre-treatment number of roaches `roach1`, the treatment indicator `treatment`, and a variable indicating whether the apartment is in a building restricted to elderly residents `senior`. Because the number of days for which the roach traps were used is not the same for all apartments in the sample, we include it as an exposure, which slightly changes the model described in the **Likelihood** section above in that the rate parameter $\lambda_i = exp(\eta_i)$ is multiplied by the exposure u_i giving us $y_i \sim Poisson(u_i\lambda_i)$. This is equivalent to adding $\ln(u_i)$ to the linear predictor η_i and it can be specified using the `offset` argument to `stan_glm`.

```
library(rstanarm)
data(roaches)

# Rescale
roaches$roach1 <- roaches$roach1 / 100

# Estimate original model
glm1 <- glm(y ~ roach1 + treatment + senior, offset = log(exposure2),
            data = roaches, family = poisson)

# Estimate Bayesian version with stan_glm
stan_glm1 <- stan_glm(y ~ roach1 + treatment + senior, offset =
log(exposure2),
                      data = roaches, family = poisson,
                      prior = normal(0, 2.5),
                      prior_intercept = normal(0, 5),
                      seed = 12345 , refresh = 0)
```

The `formula, data, family`, and `offset` arguments to `stan_glm` can be specified in exactly the same way as for `glm`. The `poisson` family function defaults to using the log link, but to write code readable to someone not familiar with the defaults we should be explicit and use `family = poisson(link = "log")`.

We've also specified some optional arguments. The `chains` argument controls how many Markov chains are executed, the cores argument controls the number of `cores` utilized by the computer when fitting the model. We also provided a seed so that we have the option to deterministically reproduce these results at any time. The `stan_glm` function has many other optional arguments that allow for more user control over the way estimation is performed. The documentation for `stan_glm` has more information about these controls as well as other topics related to GLM estimation.

Here are the point estimates and uncertainties from the `glm` fit and `stan_glm` fit, which we see are nearly identical :

```
round(rbind(glm = coef(glm1), stan_glm = coef(stan_glm1)), digits = 2)
```

```
##          (Intercept) roach1 treatment senior
## glm             3.09    0.7     -0.52  -0.38
## stan_glm        3.09    0.7     -0.52  -0.38
```

(Note: the dataset we have is slightly different from the one used in Gelman and Hill (2007), which leads to slightly different parameter estimates than those shown in the book even when copying the `glm` call verbatim. Also, we have rescaled the `roach1` predictor. For the purposes of this example, the actual estimates are less important than the process.)

Gelman and Hill next show how to compare the observed data to replicated datasets from the model to check the quality of the fit. Here we don't show the original code used by Gelman and Hill because it's many lines, requiring several loops and some care to get the matrix multiplications right (see pg. 161-162). On the other hand, the **rstanarm** package makes this easy. We can generate replicated datasets with a single line of code using the `posterior_predict` function :

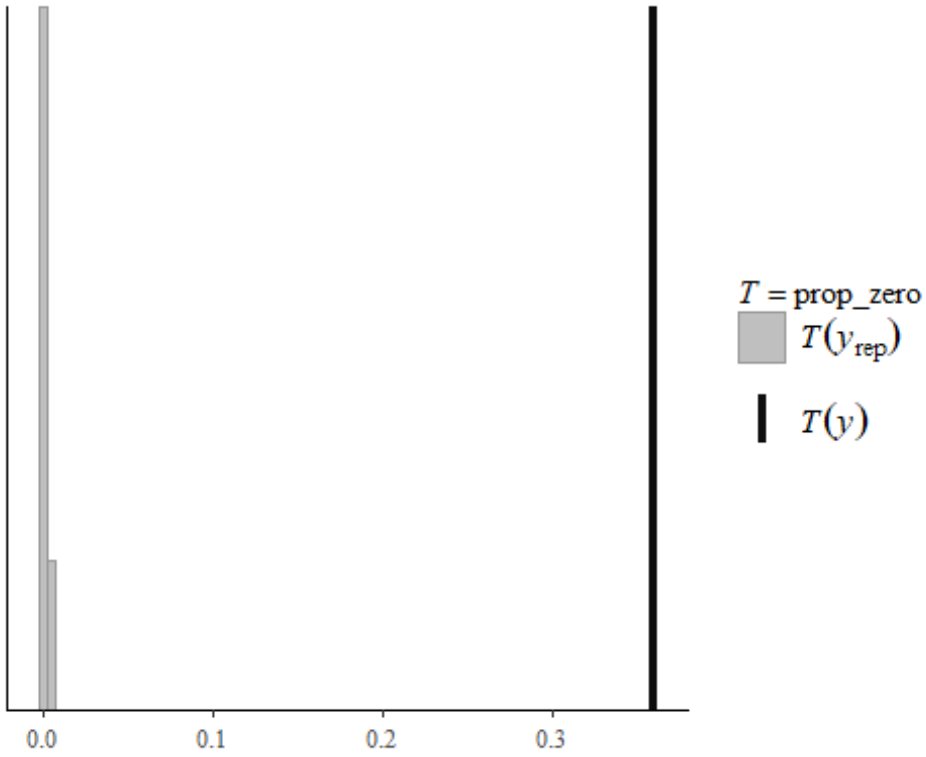
```
yrep <- posterior_predict(stan_glm1)
dim(yrep)
```

```
## [1] 4000  262
```

By default `posterior_predict` will generate a dataset for each set of parameter draws from the posterior distribution. That is, `yrep` will be an $S \times N$ matrix, where S is the size of the posterior sample and N is the number of data points. Each row of `yrep` represents a full dataset generated from the posterior predictive distribution. For more about the importance of the `posterior_predict` function, see the "How to Use the **rstanarm** Package" vignette.

Gelman and Hill take the simulated datasets and for each of them compute the proportion of zeros and compare to the observed proportion in the original data. We can do this easily using the `pp_check` function, which generates graphical comparisons of the data y and replicated datasets `yrep`.

```
prop_zero <- function(y) mean(y == 0)

(prop_zero_test1 <- pp_check(stan_glm1, plotfun = "stat", stat = "prop_zero",
binwidth = .005))
```

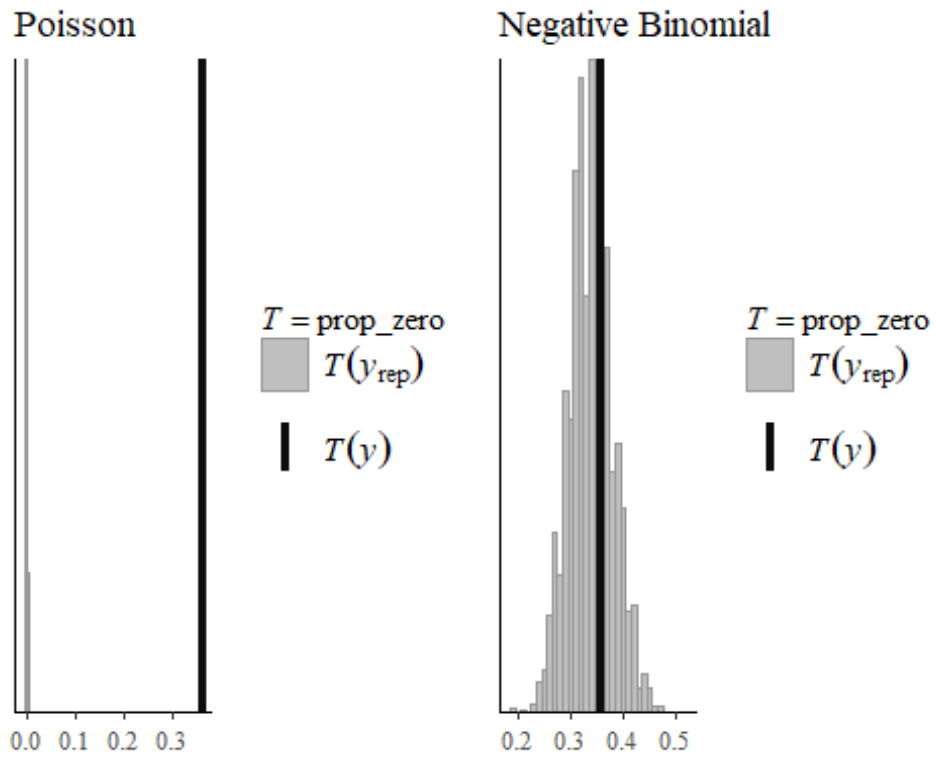


The value of the test statistic (in this case the proportion of zeros) computed from the sample y is the dark blue vertical line. More than 30% of these observations are zeros, whereas the replicated datasets all contain less than 1% zeros (light blue histogram). This is a sign that we should consider a model that more accurately accounts for the large proportion of zeros in the data. Gelman and Hill show how we can do this using an overdispersed Poisson regression. To illustrate the use of a different `stan_glm` model, here we will instead try negative binomial regression, which is also used for overdispersed or zero-inflated count data. The negative binomial distribution allows the (conditional) mean and variance of y to differ unlike the Poisson distribution. To fit the negative binomial model can either use the `stan_glm.nb` function or, equivalently, change the family we specify in the call to `stan_glm` to `neg_binomial_2` instead of `poisson`. To do the latter we can just use `update` :

```
stan_glm2 <- update(stan_glm1, family = neg_binomial_2)
```

We now use `pp_check` again, this time to check the proportion of zeros in the replicated datasets under the negative binomial model:

```
prop_zero_test2 <- pp_check(stan_glm2, plotfun = "stat", stat = "prop_zero",
                            binwidth = 0.01)
# Show graphs for Poisson and negative binomial side by side
bayesplot_grid(prop_zero_test1 + ggtitle("Poisson"),
               prop_zero_test2 + ggtitle("Negative Binomial"),
               grid_args = list(ncol = 2))
```



This is a much better fit, as the proportion of zeros in the data falls nicely near the center of the distribution of the proportion of zeros among the replicated datasets. The observed proportion of zeros is quite plausible under this model.

We could have also made these plots manually without using the `pp_check` function because we have the `yrep` datasets created by `posterior_predict`. The `pp_check` function takes care of this for us, but `yrep` can be used directly to carry out other posterior predictive checks that aren't automated by `pp_check`.

When we comparing the models using the **loo** package we also see a clear preference for the negative binomial model

```
loo1 <- loo(stan_glm1)

## Warning: Found 15 observations with a pareto_k > 0.7. With this many
problematic observations we recommend calling 'kfold' with argument 'K=10' to
perform 10-fold cross-validation rather than LOO.

loo2 <- loo(stan_glm2)
loo_compare(loo1, loo2)

##           elpd_diff se_diff
## stan_glm2     0.0       0.0
## stan_glm1 -5345.4     706.8

loo1
```

```
##
## Computed from 4000 by 262 log-likelihood matrix
##
##          Estimate     SE
## elpd_loo  -6241.0  725.4
## p_loo       283.6   71.7
## looic     12482.0 1450.8
## ------
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     239   91.2%   347
##  (0.5, 0.7]   (ok)         8    3.1%   121
##    (0.7, 1]   (bad)        8    3.1%   16
##    (1, Inf)   (very bad)   7    2.7%   1
## See help('pareto-k-diagnostic') for details.
```

```
loo2
```

```
##
## Computed from 4000 by 262 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -895.5 37.7
## p_loo         6.6  2.5
## looic      1791.1 75.4
## ------
## Monte Carlo SE of elpd_loo is 0.1.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     260   99.2%   3090
##  (0.5, 0.7]   (ok)         2    0.8%   121
##    (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.
```

which is not surprising given the better fit we've already observed from the posterior predictive checks.

Lower looic is better fit of the model. Simce looic for poisson is greater than the looic of negative binomial. Hence Negative Binomial is better fit.

$$looic = -2 \times elpd\_diff$$

7 8 Introduction to Bayesian computation

7.1 Notes

Bayesian computation revolves around two steps : 1. Computation of the posterior distribution, $p(\theta|y)$

2. Computation of the posterior predictive distribution, $p(\tilde{y}|y)$

7.2 Normalized and Unnormalized Densities

$p(\theta|y)$ is the target distribution and $q(\theta|y)$ is the unnormalized density.

- $p(y) = \frac{1}{\sqrt{2\pi}} e^{-y^2/2}$, since $\int p(y)dy = 1 \Rightarrow p(y)$ is normalize.
- $q(y) = e^{-y^2/2}$, since $\int p(y)dy \neq 1 \Rightarrow q(y)$ is not normalize.

Our requirement is that $\frac{q(\theta|y)}{p(\theta|y)}$ should be constant.

7.3 8.2 Log Densities

To avoid computational overflows or underflows one should compute with $log\ p(\theta|y)$

7.4 Numerical Integration

$$E(h(\theta)|y) = \int h(\theta)\, p(\theta|y)d\theta$$

which is often interactable and one has to rely to simulation.

7.5 Simulation Methods

Simulation (stochastic) methods are based on obtaining random samples θ^s from the desired distribution $p(\theta)$ and estimating the expectation of any function $h(\theta|y)$ as :

$$E(h(\theta)|y) = \int h(\theta)\, p(\theta|y)d\theta = \frac{1}{S}\sum_{s=1}^{S} h\,(\theta^s)$$

7.6 Distribution Approximations

7.6.1 Normal Approximation to Posterior $p(\theta|y)$

If the posterior distribution $p(\theta|y)$ is unimodal and roughly symmetric, it can be convenient to approximate it by a normal distribution that is the logorithm of the posterior density is approximated by a quadratic function of θ.

A **Taylor Series expansion of** $log\ p(\theta|y)$ centered at posterior mode $\hat{\theta}$, gives

$$log\ p(\theta|y) = log\ p\left(\hat{\theta}|y\right) + \frac{d}{d\theta}p(\theta|y)\Big|_{\theta=\hat{\theta}}(\theta-\hat{\theta}) + \frac{1}{2}(\theta-\hat{\theta})^{\top}\left[\frac{d^2}{d\theta^2}log\ p(\theta|y)\right]_{\theta=\hat{\theta}}(\theta-\hat{\theta})$$

$$p(\theta|y) = exp\left[log\ p(\hat{\theta}|y) + \frac{1}{2}(\theta-\hat{\theta})^{\top}\left[\frac{d^2}{d\theta^2}log\ p(\theta|y)\right]_{\theta=\hat{\theta}}(\theta-\hat{\theta})\right]$$

$$\Rightarrow p(\theta|y) \propto exp\left[-\frac{1}{2}(\theta-\hat{\theta})^{\top}I(\hat{\theta})(\theta-\hat{\theta})\right]$$

where, $I(\hat{\theta}) = -\left[\frac{d^2}{d\theta^2}log\ p(\theta|y)\right]_{\theta=\hat{\theta}}$

$$p(\theta|y) \sim N\left(\hat{\theta}, I^{-1}(\theta)\right)$$

Note that posterior node $\hat{\theta}$ and negative of the *Hesian Matrix* need to the computed in this approximation.

7.7 8.6 Simulating from the Predictive Distribution

$$p(\tilde{y}|y) = \int p(\tilde{y}|\theta)p(\theta|y)d\theta$$

Once we have assemble from posterior distribution $p(\theta|y)$, it is typically easy to draw from the predictive distribution of unobserved or future data $\tilde{y}$. For each draw θ from posterior distribution just draw on $\tilde{y}$ from predictive distribution $p(\tilde{y}|\theta)$.

The set of simulated $\tilde{y}$ from all $\theta's$ characterizes from the posterior distribution.

7.7.1 Rejection Sampling

Suppose we want to obtain a single random draw from a density $p(\theta|y)$ or perhaps an unnormalized density $q(\theta|y)$ (with $p(\theta|y) = q(\theta|y)/\int q(\theta|y)d\theta$).

To perform rejection sampling we require a positive function $g(\theta)$ defined for all θ for which $p(\theta|y) > 0$ that has the following properties :

1. We can draw from the probability density proportional to g. It is not required that $g(\theta)$ integrate to 1, but $g(\theta)$ must have a finite integral.
2. The importance ratio $\frac{p(\theta|y)}{g(\theta)}$ must have a known bound; that is, there must be some known constant M for which $\frac{p(\theta|y)}{g(\theta)} \leq M$ for all θ.

The rejection sampling algorithm proceeds in two steps :

1. Sample θ at random from the probability density proportional to $g(\theta)$.

2. With probability $\frac{p(\theta|y)}{N\,g(\theta)}$, accept θ as a draw from p. If the drawn θ is rejected, return to step 1.

7.8 Importance Sampling

Importance sampling is a method, related to rejection sampling and a precursor to the Metropolis algorithm that is used for computing expectations using a random sample drawn from an approximation to the target distribution.

If $g(\theta)$ is a probability density from which we can generate random draws, then we can write,

$$e(h(\theta)|y) = \frac{\int h(\theta)q(\theta|y)d\theta}{\int q(\theta|y)d\theta} = \frac{\int [h(\theta)q(\theta|y)/g(\theta)]g(\theta)d\theta}{\int [q(\theta|y)/g(\theta)]g(\theta)d\theta}$$

which can be estimated using S draws $\theta^1, \theta^2, \ldots, \theta^S$ from $g(\theta)$ by the expression,

$$\frac{\frac{1}{S}\sum_{s=1}^{S} h\,(\theta^s)w(\theta^s)}{\frac{1}{S}\sum_{s=1}^{S} w\,(\theta^s)}$$

where the factors

$$w(\theta^s) = \frac{q(\theta^s|y)}{g(\theta^s)}$$

are called importance ratios or importance weights.

If $g(\theta)$ can be chosen such that $\frac{hq}{g}$ is roughly constant, then fairly precise estimates of the integral can be obtained.

7.9 Matropolis Algorithm

1. Draw a starting point θ^0 from a starting distribution $p_0(\theta)$.

2. For $t = 1,2,\ldots$

(a) Sample a proposal θ^* from a jumping or proposal distribution distinct at time $t, J_t(\theta^*|\theta^{t-1})$. J_t must be symmatric.

(b) Calculate the ratio of densities

$$\frac{p(\theta^*|y)}{p(\theta^{t-1}|y)}$$

(c) Set

$$\theta^t = \begin{cases} \theta^*, & \text{with probability min(r, 1)} \\ \theta^{t-1}, & \text{otherwhere.} \end{cases}$$

are equivalent $u \sim U(0,1)$, if $u < r$ accept $\theta^* = \theta^t$ otherwise $\theta^* = \theta^{t-1}$.

Example : Bivariate Unit Normal with Normal Jumping :- The Target density

$$p(\theta|y) = N(0, I), \quad I_{2\times 2}$$

The jumping or proposal $J_t(\theta^*|\theta^{t-1}) = N(\theta^*|\theta^{t-1}), 0.2^2 I$. Hence

$$r = \frac{N(\theta^*|0, I)}{N(\theta^{t-1}|0, I)}$$

Simulate $u \sim U(0,1)$, if $u < r$ accept $\theta^* = \theta^t$ otherwise $\theta^* = \theta^{t-1}$.

7.10 8.11 Matropolis Hestings Algorithm

Note that in Matropolis Algorithm, proposal density is always symmetric, hence a correction for symmatry. i.e $I_E(\theta_a|\theta_b) = J_t(\theta_b|\theta_a)$ is required. Thus

$$r = \frac{p(\theta^*|y)/J_t(\theta^*|\theta^{t-1})}{p(\theta^{t-1}|y)/J_t(\theta^{t-1}|\theta^*)}$$

rest is same as in Matropolis Algorithm.

7.10.1 Good Properties of Jumping / Proposal Distribution

A good jumping / proposal distribution has the following properties :

- For any θ, it is easy to sample from $J(\theta*|\theta)$.
- It is easy to compute ratio r.
- Each jump goes a reasonable distance in parametric space.
- The jump are not rejected to frequently otherwise random walk waste time standing still.

7.10.2 Gibbs Sampler

A particular Markov chain algorithm that has been found useful in many multidimensional problems is the Gibbs sampler, also called alternating conditional sampling, which is defined in terms of subvectors of θ.

Suppose the parameter vector θ has been divided into d components or subvectors, $\theta = (\theta_1, \theta_2, \ldots, \theta_d)$

- Each iteration of the Gibbs sampler cycles through the subvectors of θ, drawing each subset conditional on the value of all the others.
- There are thus d steps in iteration t. At each iteration t, an ordering of the d subvectors of θ is chosen and, in turn, each θ_j^t is sampled from the conditional distribution given all the other components of θ :

$$p(\theta_j|\theta_{-j}^{t-1}, y)$$

where θ_{-j}^{t-1} represents all the components of θ, except for θ_j, at their current values :

$$\theta_{-j}^{t-1} = (\theta_1^t, \ldots, \theta_{j-1}^t, \theta_{j+1}^t, \ldots, \theta_d^{t-1})$$

- Thus, each subvector θ_j is updated conditional on the latest values of the other components of θ_j.

Example : Bivariate Noraml Distribution Consider a single observation (y_1, y_2) from a bivariate normally distributed population with unknown mean $\theta = (\theta_1, \theta_2$ and known covariance matrix $\begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}$. That is

$$p(y|\theta) = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \sim N_2\left(\begin{pmatrix} y_1 \\ y_2 \end{pmatrix}, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}\right)$$

With a uniform prior distribution on θ, the posterior distribution

$$p(\theta|y) \sim p(y|\theta)\, p(\theta)$$

$$\Rightarrow y\left|\theta = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}\right| y \sim N_2\left(\begin{pmatrix} y_1 \\ y_2 \end{pmatrix}, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}\right)$$

For Gibbs Sampler, we need conditional distributions

$$\theta_1|\theta_2, y \sim N(y_1 + \rho(\theta_2 - y_2), 1 - \rho^2)$$

$$\theta_2|\theta_1, y \sim N(y_2 + \rho(\theta_1 - y_1), 1 - \rho^2)$$

The Gibbs sampler proceeds by alternately sampling from these two normal distribution.

7.10.3 Assessing Convergence of MCMC Algorithm

Convergence of an MCMC Algorithm is used in the sense of convergence to the target posterior density. Various convergence criteria, quantitative and visual have been proposed to assess convergence. However, the most widely used quantitative convergence criterion is the potential scale reduction factor, $\hat{R}$ and visual criteria are mainly **traceplot** and **auto-correlation plot**.

7.10.4 Potential Scale Reduction Factor – $\hat{R}$

For multiple MCMC chains, the criterion is based on comparing the overall variance of the chains to the within chain variance if V_1 is between the chain variance and V_2 is the within the chain variance and the length of each MCMC chain is m, then posterior variance is estimated as

$$\widehat{Var(\theta|y)} = \frac{1}{m}V_1 + \frac{m-1}{m}V_2$$

and $\hat{R}$ is estimated by

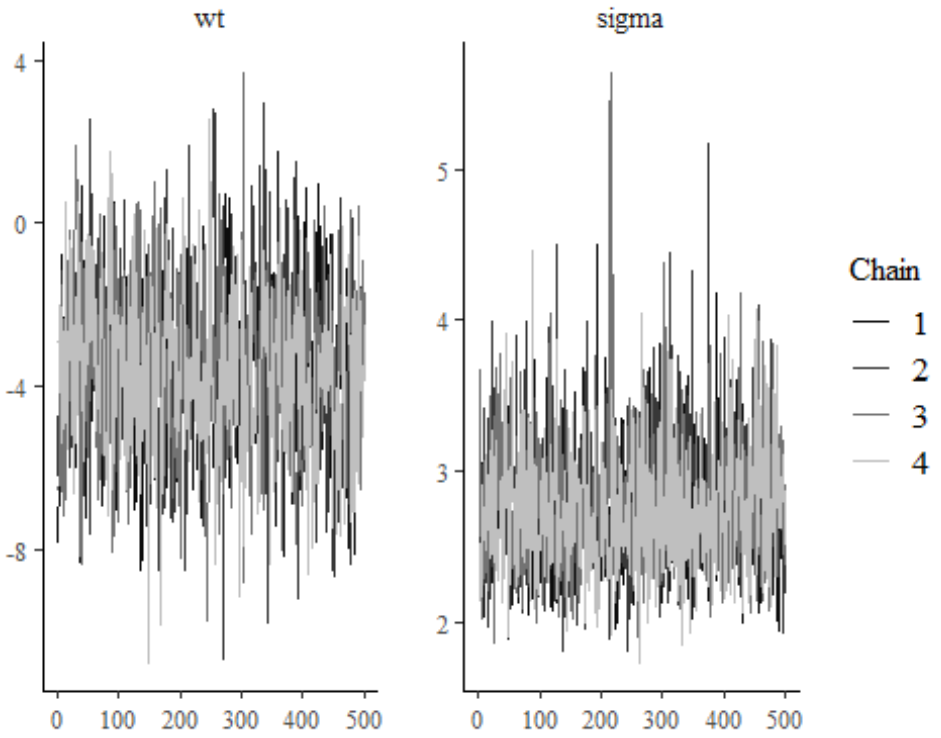
$$\hat{R} = \sqrt{\frac{\widehat{Var(\theta|y)}}{V_2}}$$

which is reduce to 1 as $m \to \infty$

Gelman et al(2014) recommended acceptable limit of potential scale reduction factor $\hat{R} < 1.1$

7.10.5 Trace Plots

```
library(bayesplot)
library(ggplot2)
library(rstanarm)
data("mtcars")


# If we decrease the iteration / iter then the mixing is not good. # To check this we can see this at iter = 100

# For iter = 1000
fit<- stan_glm(mpg ~. , data = mtcars, refresh = 0, iter = 1000)
posterior<- as.array(fit)

dim(posterior)

## [1] 500   4  12

# Trace Plot
mcmc_trace(posterior, pars = c("wt", "sigma"))
```

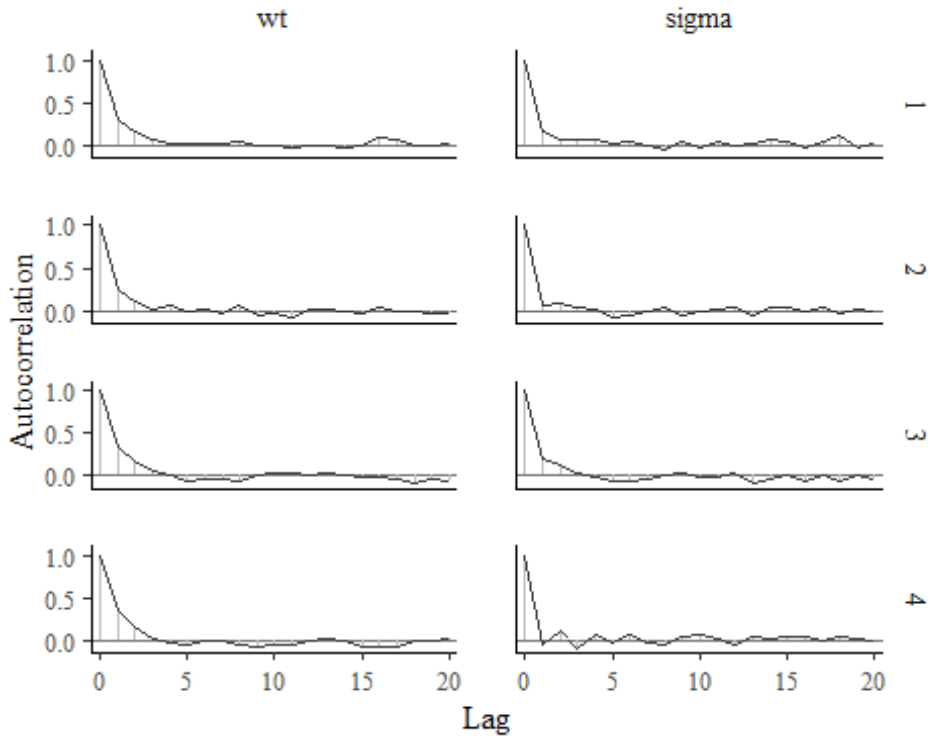


Trace Plot is simply a two-dimensional plot in which *X-axis* represents the *iteration number* and the *Y-axis* represents the corresponding *simulated values* of the random variable. If the trace plot shows a stationary pattern fluctuating within a band and looks like a *"fact hairy caterpillar"*, then algorithm is said to converge to the posterior distribution and mixing well throughout the support of the posterior density.

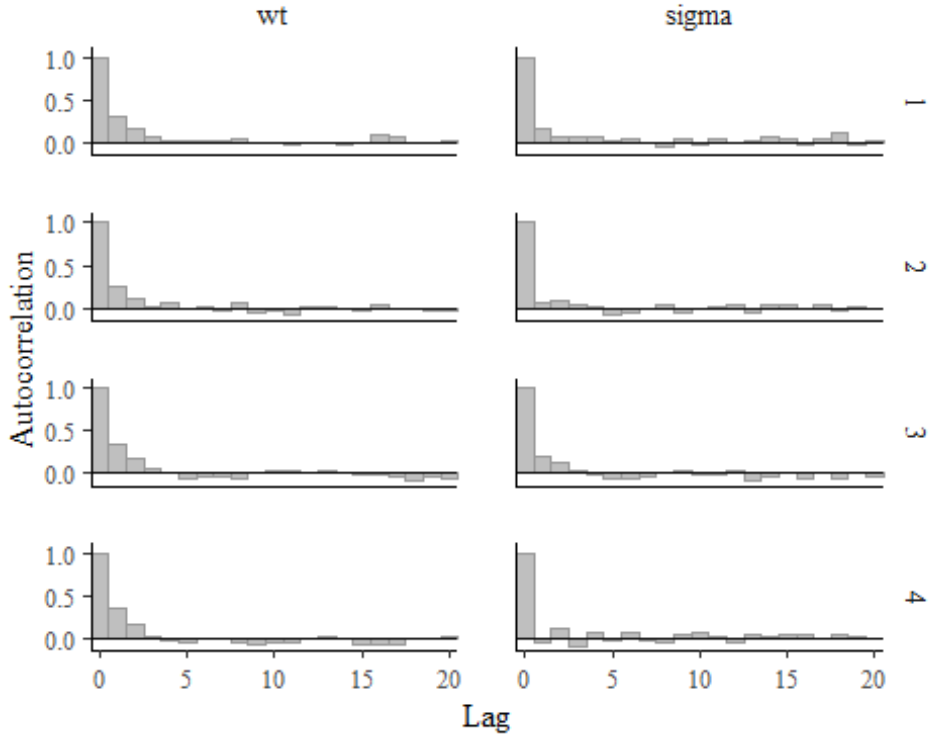
When convergence is achieved, different chains overlay within the same plot, as such the chains are indistinguishable. A non-convergence MCMC algorithm will never levels-off to a stable, stationary state or it may show some noticeable periodic pattern.

7.10.6 Auto-Correlation Plot

```
# If we decrease the iteration / iter then the result is not good. # To check
this we can see this at iter = 100

# For iter = 1000
mcmc_acf(posterior, pars = c("wt", "sigma"))
```




```
mcmc_acf_bar(posterior, pars = c("wt", "sigma"))
```



MCMC methods generate dependent sequence of values of the parameters, the dependence is measured by correlation between pairs θ^j, θ^{j+1} within a chain, which is known as **auto-correlation** with lag period 1.

In an auto-correlation plot *X-axis* represents *lags* and *auto-correlation* is plotted along the *Y-axis* and the height of each bar express the magnitude of the auto-correlation at that lag. A well mixed chain produces auto-correlation plot where the correlation values quickly drop close to $zero^{(0)}$ as the lag increases. A Markov Chain in which auto-correlation is large at lag 1 and decays slowly as the lag increases is said to mix slowly and it will converge slowly. That is, it will take a long time to find its stationary distribution, even after reaching stationary distribution, it will take many iterations to explore the entire support of the distribution and a very large number of iterations will be required to obtain usefully precise estimation of the characteristics of the posterior distribution.

7.10.7 Effective Sample Size

```
# If we decrease the iteration / iter then the n_eff is not good.
# To check this we can see this at iter = 100

# For iter = 1000
summary(fit)

##
## Model Info:
##  function:     stan_glm
##  family:       gaussian [identity]
##  formula:      mpg ~ .
##  algorithm:    sampling
##  sample:       2000 (posterior sample size)
##  priors:       see help('prior_summary')
##  observations: 32
##  predictors:   11
##
## Estimates:
##               mean   sd    10%   50%   90%
## (Intercept)  11.5   19.4 -13.6  11.3  36.0
## cyl          -0.1    1.1  -1.4  -0.1   1.3
## disp          0.0    0.0   0.0   0.0   0.0
## hp            0.0    0.0   0.0   0.0   0.0
## drat          0.8    1.6  -1.2   0.8   2.8
## wt           -3.6    2.0  -6.1  -3.6  -1.1
## qsec          0.8    0.7  -0.1   0.8   1.8
## vs            0.2    2.1  -2.4   0.3   2.9
## am            2.6    2.1  -0.1   2.6   5.3
## gear          0.7    1.5  -1.3   0.8   2.7
## carb         -0.3    0.9  -1.4  -0.3   0.8
## sigma         2.8    0.4   2.3   2.7   3.4
##
## Fit Diagnostics:
##            mean   sd   10%   50%   90%
```

```
## mean_PPD 20.1    0.7 19.2  20.1  21.0
##
## The mean_ppd is the sample average posterior predictive distribution of
the outcome variable (for details see help('summary.stanreg')).
##
## MCMC diagnostics
##               mcse Rhat n_eff
## (Intercept)   0.5  1.0  1626
## cyl           0.0  1.0  1336
## disp          0.0  1.0  1044
## hp            0.0  1.0  1341
## drat          0.0  1.0  2026
## wt            0.1  1.0   998
## qsec          0.0  1.0  1430
## vs            0.1  1.0  1379
## am            0.1  1.0  1335
## gear          0.0  1.0  1639
## carb          0.0  1.0   993
## sigma         0.0  1.0  1347
## mean_PPD      0.0  1.0  2187
## log-posterior 0.1  1.0   636
##
## For each parameter, mcse is Monte Carlo standard error, n_eff is a crude
measure of effective sample size, and Rhat is the potential scale reduction
factor on split chains (at convergence Rhat=1).
```

The **effective sample size** or effective number of samples (`n_eff`) is a measure of number of independent samples from the posterior distribution. The larger the effective sample size, the greater the precision of the MCMC estimates. MCMC samples are generally auto-correlated and rest independent. Auto-correlation reduces the effective sample size.

Usually $n\_eff > 400$ is sufficient for reliable diagnostics and accurate parameter estimates.

Stan provides an estimate of effective sample size for estimates of the parameters.

7.10.8 Monte Carlo Error

Monte Carlo (MC) error, $mcse$ is a measure of variability of each of the parameter estimates that is due to simulation. It can be said as the measure of performance of Markov Chain.

MCMC samples are not independent rather correlated and as such standard independent sample methods can not be used to compute MCMC error. It is defined as posterior standard deviation (SD) divided by the square root of the effective sample size, that is, MC error

$$mcse = \frac{sd}{\sqrt{n\_eff}}$$

It should be less than 0.5 of the posterior SD, Stan provides MC error.

```r
# Sd_intercept = 19.9   and n_eff_interecept = 1624
mcse_itercept<- 19.9/sqrt(1624) ;   mcse_itercept

## [1] 0.4938102
```

7.10.9 Thinning

MCMC generated samples are not independent rather correlated. To reduce auto-correlation, only k^{th} sampled value is saved discarding the rest where $\mu > 1$ is the lag period beyond which auto-correlation is low. So thinning reduces auto-correlation and increases effective sample size and whereas effective sample size (`n_eff`). It also saves storage space and increases computational speed in high dimensional problems.

7.10.10 Warm-up Period

The starting period, that is the number of iterations before convergence, which are discarded in order to avoid the inflence of the initial values are called **burn-in** or **warm-up period**. The effect of this period becomes less if the sample is large enough. In Stan, this Starting period is known as warm-up period. During this warm-up period. **Stan*** does not draw samples from the posterior distribution.

7.11 Hamilton Mante Carlo (HMC)

HMC borrows an idea from physics to suppress the local random walk behavior in the Metropolis algorithm, thus allowing it to move much more rapidly through the target distribution.

Target Density is $p(\theta|y)$.

7.11.1 Auxiliary Momentum Variable phi

HMC introduces auxiliary momentum variables ϕ and draw from a joint density

$$p(\phi, \theta) = p(\phi|\theta)\ p(\theta)$$

In most application

$$\phi \sim Multi\_normal(0, \mu)$$

By default Stan sets M^{-1} equal to a diagonal estimate of the covariance computed during warm-up.

7.11.2 The Hamilton

The joint density $p(\phi, \theta)$ defines a Hamilton

$$H(\phi, \theta) = -log\ p(\phi, \theta)$$

$$H(\phi, \theta) = -log\ p(\phi, \theta) - log\ p(\theta)$$

$$H(\phi, \theta) = \text{Kinetic Energy + Potential Energy}$$

7.12 The Three Steps of an HMC Iteration

1. The iteration begins by updating ϕ with a random draw from its posterior distribution.

2. The main part of the Hamiltonian Monte Carlo iteration is a simultaneous update of (ϕ, θ). This update involves **L'leapfrog steps'**, which is defined as follows :

Repeat the following steps L times :

(a) Use the gradient of the log-posterior density of θ to make a half-step of ϕ :

$$\phi \leftarrow \phi + \frac{1}{2}\epsilon \frac{d}{d\theta} log\ p(\theta|y)$$

(b) Use the 'momentum' vector ϕ to update the 'position' vector θ :

$$\phi \leftarrow \phi + \epsilon M^{-1}\phi$$

(c) Again use the gradient of $theta$ to half-update ϕ :

$$\phi \leftarrow \phi + \frac{1}{2}\epsilon \frac{d}{d\theta} log\ p(\theta|y)$$

3. Label θ^{t-1}, ϕ^{t-1} as the value of the parameter and momentum vectors at the start of the leapfrog process and θ^*, ϕ^* as the value after the L steps. In the accept-reject step, we compute

$$r = \frac{p(\theta^*|y)p(\phi^*)}{p(\theta^{t-1}|y)p(\phi^{t-1}}$$

4. Set

$$\theta^t = \begin{cases} \theta^*, & \text{with probability min(r, 1)} \\ \theta^{t-1}, & \text{otherwhere.} \end{cases}$$

This is the step 4, generate $u \sim U(0,1)$, if $u < r$ accept $\theta^* = \theta^t$ otherwise $\theta^* = \theta^{t-1}$.

8 Evaluating Mode Fitting

For fitting a Bayesian Model to the data y, the posterior distribution $p(\theta|y)$ is obtained. Evaluating model fit means :

(i) Assessing a Single Model

(ii) Comparing two or more completing models.

Posterior predictive distribution is used to check model fit in Bayesian scenario. That is, if a model fits the current data well, future data simulated from the model should look like similar to the current data. The posterior predictive density

$$p(y_{n+1}|y) = \hat{p}(y_{new}|y) = p(y_{rep}|y)$$

is obtained by the equation

$$p(y_{n+1}|y) = \int p(y_{n+1}|\theta)\ p(\theta|y)d\theta$$

Then drawing samples from $p(y_{n+1}|y)$ can be performed in two steps :

(i) Simulate θ_i from the posterior distribution $p(\theta|y)$

(ii) For a fixed θ_i, simulate from the data density $p(y_{n+1}|\theta_i)$

A model is said to fit well if the model predicts the future observations or predict the past observations that are consistent with current data.

8.1 Leave-One-Out Information : (LOOIC)

In Bayesian cross-validation, the data are repeatedly partioned into a training set y_{train} and a testing set y_{test}. The idea of cross-validation is to fit the model using y_{train} and make predictions based on y_{test}.

For a sample of size n, if y_{train} contains $n-1$ observations and y_{test} contains only the remaining 1 data point, then, it is called **Leave-One-Out Cross- Validation (LOOCV)**, it is based on *log pointwise predictive density (lppd)* and defined as follows :

$$lppd_{loocv} = \sum_{i=1}^{n} log\left(\int p(y_i|\theta)\ p(\theta|y_{-i})d\theta = \sum_{i=1}^{n} log\left(E\left(p(y_i|\theta)\right)\ p(\theta|y_{-i})\right)d\theta\right.$$

where y_{-i} is the data set without the i^{th} data point. An can be computed as

$$llpd_{loocv} = \sum_{i=1} log\left(\frac{1}{T}\sum_{i=1} p\left(y_i|\theta^{it}\right)\right)$$

where $\theta^{it} = 1,2,\ldots,T$ are draws from the posterior distribution $p(\theta|y_{-i})$.

8.1.1 The LOOIC

The leave-one-out (cross-validation) information criterian (LOOIC) is defined as

$$LOOIC = -2 \times llpd_{loocv}$$

$$= -2\sum_{i=1}^{n} log\left(\int p(y_i|\theta)\, p(\theta|y_{-i})d\theta\right)$$

The LOOIC is heavily completing intensive because n models have to fitted and for each model, the posterior distributions of the parameters are required to be simulated.

However, Vehtrieted(2012) estimated (LOOIC) as

$$LOOIC = -2\sum_{i=1}^{n} log\left(\frac{\sum_{s=1}^{S} w_i^s\, p(y_i|\theta^s)}{\sum_{s=1}^{S} w_i^s}\right)$$

where $s = 1,2,\ldots,S$, simulated sample size, θ^s is the s^{th} value of θ in the joint posterior sample and w_i^s is a Parets smoothed importance sampling weight. The R package **loo** can be called to compute LOOIC.

Least LOOIC is the best fitted model

8.2 Widely Applicable Information Criterion (WAIC)

Watanabe and Opper(2010) proposed a measure, Widely Applicable Information Criterion (WAIC), which is used for comparing models under Bayesian framework. It is also known as Watanable Akaike information criterion. WAIC is based on log pointwise predictive density (lppd) and define as :

$$WAIC = -2 \times lppd + 2\sum_{i=1}^{n} Var\, log\, p(y_i|\theta)$$

where, $lppd = \sum_{i=1}^{n} log\left(\int p(y_i|\theta)\, p(\theta|y_{-i})d\theta\right)$ and $Var\, log\, p(y_i|\theta)$ is known as *penalty* term which is the seem of posterior variance in log probabilities for each observation y_i.

Least WAIC is the least fitted model.

9 Using the loo package (version >= 2.0.0)

```
library(loo)
help(pack = loo)
loo::loo2-example
```

9.1 Introduction

This vignette demonstrates how to use the **loo** package to carry out Pareto smoothed importance-sampling leave-one-out cross-validation (PSIS-LOO) for purposes of model checking and model comparison.

In this vignette we can't provide all necessary background information on PSIS-LOO and its diagnostics (Pareto k and effective sample size)

9.2 Example : Poisson vs negative binomial for the roaches dataset

Background and model fitting

The Poisson and negative binomial regression models used below in our example, as well as the `stan_glm` function used to fit the models, are covered in more depth in the **rstanarm** vignette Estimating Generalized Linear Models for Count Data with rstanarm. In the rest of this vignette we will assume the reader is already familiar with these kinds of models.

Roaches data the treatment and control were applied to 160 and 104 apartments, respectively, and the outcome measurement yi in each apartment i was the number of roaches caught in a set of traps. Different apartments had traps for different numbers of days

In addition to an intercept, the regression predictors for the model are `roach1`, the pre-treatment number of roaches (rescaled above to be in units of hundreds), the `treatment` indicator treatment, and a variable indicating whether the apartment is in a building restricted to elderly residents `senior`. Because the number of days for which the roach traps were used is not the same for all apartments in the sample, we use the `offset` argument to specify that `log(exposure2)` should be added to the linear predictor.

```
library("rstanarm")
library("bayesplot")
library("loo")

# the 'roaches' data frame is included with the rstanarm package
data(roaches)

# rescale to units of hundreds of roaches
roaches$roach1 <- roaches$roach1 / 100
head(roaches)
```

```
##     y roach1 treatment senior exposure2
## 1 153 3.0800         1      0  0.800000
## 2 127 3.3125         1      0  0.600000
## 3   7 0.0167         1      0  1.000000
## 4   7 0.0300         1      0  1.000000
## 5   0 0.0200         1      0  1.142857
## 6   0 0.0000         1      0  1.000000
```

9.2.1 Fit Poisson model

We'll fit a simple Poisson regression model using the `stan_glm` function from the **rstanarm** package.

```
fit1 <-
  stan_glm(
    formula = y ~ roach1 + treatment + senior,
    offset = log(exposure2),
    data = roaches,
    family = poisson(link = "log"),
    prior = normal(0, 2.5, autoscale = TRUE),
    prior_intercept = normal(0, 5, autoscale = TRUE),
    seed = 12345, refresh = 0
  )
```

Usually we would also run posterior predictive checks as shown in the **rstanarm** vignette (Estimating Generalized Linear Models for Count Data with rstanarm), but here we focus only on methods provided by the **loo** package.

9.2.2 Using the loo package for model checking and comparison

Although cross-validation is mostly used for model comparison, it is also useful for model checking.

9.2.3 Computing PSIS-LOO and checking diagnostics

We start by computing PSIS-LOO with the `loo` function. Since we fit our model using rstanarm we can use the `loo` method for `stanreg` objects (fitted model objects from **rstanarm**), which doesn't require us to first extract the pointwise log-likelihood values. If we had written our own Stan program instead of using **rstanarm** we would pass an array or matrix of log-likelihood values to the `loo` function (see, e.g. `help("loo.array", package = "loo")`). We'll also use the argument `save_psis = TRUE` to save some intermediate results to be re-used later.

```
loo1 <- loo(fit1, save_psis = TRUE)

## Warning: Found 17 observations with a pareto_k > 0.7. With this many
problematic observations we recommend calling 'kfold' with argument 'K=10' to
perform 10-fold cross-validation rather than LOO.
```

`loo` gives us warnings about the Pareto diagnostics, which indicate that for some observations the leave-one-out posteriors are different enough from the full posterior that importance-sampling is not able to correct the difference. We can see more details by printing the `loo` object.

```
print(loo1)

##
## Computed from 4000 by 262 log-likelihood matrix
##
##          Estimate     SE
## elpd_loo  -6247.8  728.0
## p_loo       292.4   73.3
## looic     12495.5 1455.9
## ------
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     239   91.2%   200
##  (0.5, 0.7]   (ok)         6    2.3%   56
##    (0.7, 1]   (bad)        8    3.1%   25
##    (1, Inf)   (very bad)   9    3.4%   1
## See help('pareto-k-diagnostic') for details.
```

The table shows us a summary of Pareto k diagnostic, which is used to assess the reliability of the estimates. In addition to the proportion of leave-one-out folds with k values in different intervals, the minimum of the effective sample sizes in that category is shown to give idea why higher k values are bad. Since we have some $k > 1$, we are not able to

compute an estimate for the Monte Carlo standard error (SE) of the expected log predictive density (`elpd_loo`) and `NA` is displayed.

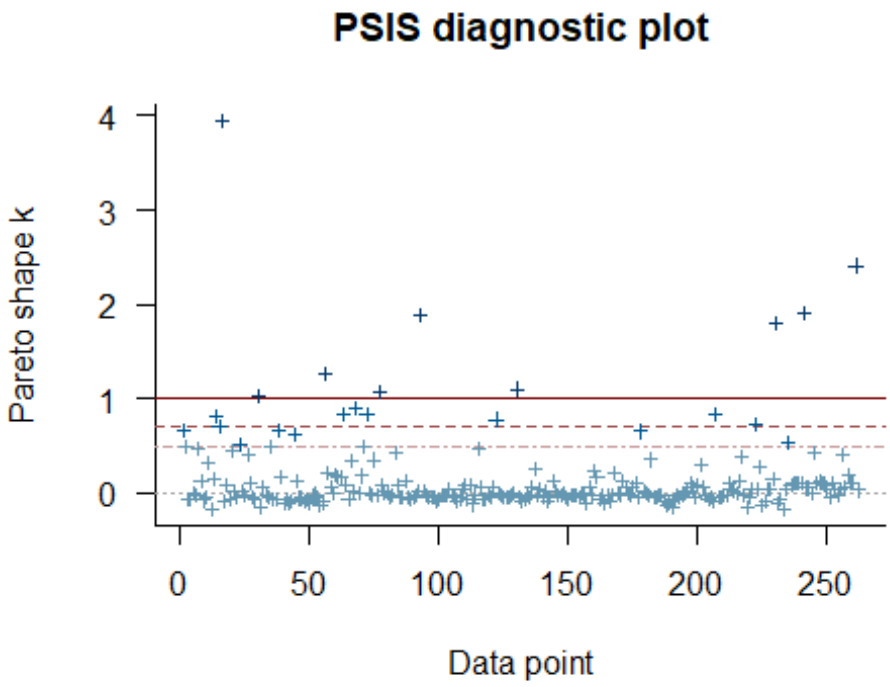
(Full details on the interpretation of the Pareto k diagnostics are available in the Vehtari, Gelman, and Gabry (2017) and Vehtari, Simpson, Gelman, Yao, and Gabry (2019) papers referenced at the top of this vignette.)

In this case the elpd_loo estimate should not be considered reliable. If we had a well-specified model we would expect the estimated effective number of parameters (p_loo) to be smaller than or similar to the total number of parameters in the model. Here p_loo is almost 300, which is about 70 times the total number of parameters in the model, indicating severe model misspecification.

9.2.4 Plotting Pareto k diagnostics

Using the `plot` method on our `loo1` object produces a plot of the k values (in the same order as the observations in the dataset used to fit the model) with horizontal lines corresponding to the same categories as in the printed output above.

```
plot(loo1)
```

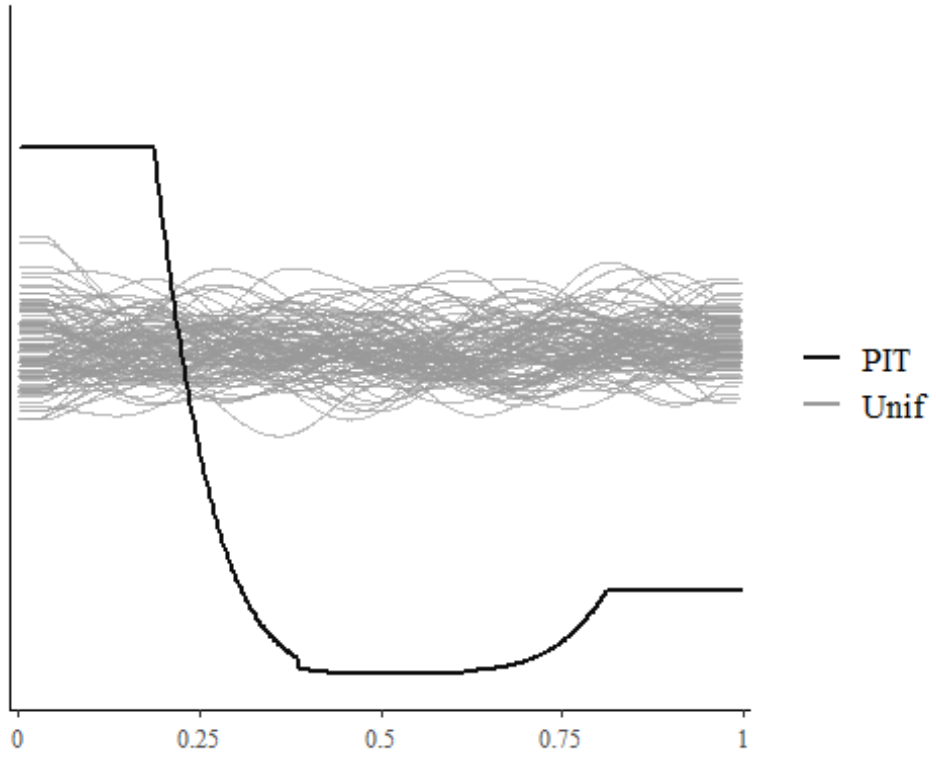


This plot is useful to quickly see the distribution of k values, but it's often also possible to see structure with respect to data ordering. In our case this is mild, but there seems to be a block of data that is somewhat easier to predict (indices around 90–150). Unfortunately even for these data points we see some high k values.

9.2.5 Marginal posterior predictive checks

The loo package can be used in combination with the **bayesplot** package for leave-one-out cross-validation marginal posterior predictive checks Gabry et al (2018). LOO-PIT values are cumulative probabilities for yi computed using the LOO marginal predictive distributions $p(y_i|y_{-i})$. For a good model, the distribution of LOO-PIT values should be uniform. In the following plot the distribution (smoothed density estimate) of the LOO-PIT values for our model (thick curve) is compared to many independently generated samples (each the same size as our dataset) from the standard uniform distribution (thin curves).

```
yrep <- posterior_predict(fit1)

ppc_loo_pit_overlay(
  y = roaches$y,
  yrep = yrep,
  lw = weights(loo1$psis_object)
)

## NOTE: The kernel density estimate assumes continuous observations and is
not optimal for discrete observations.
```



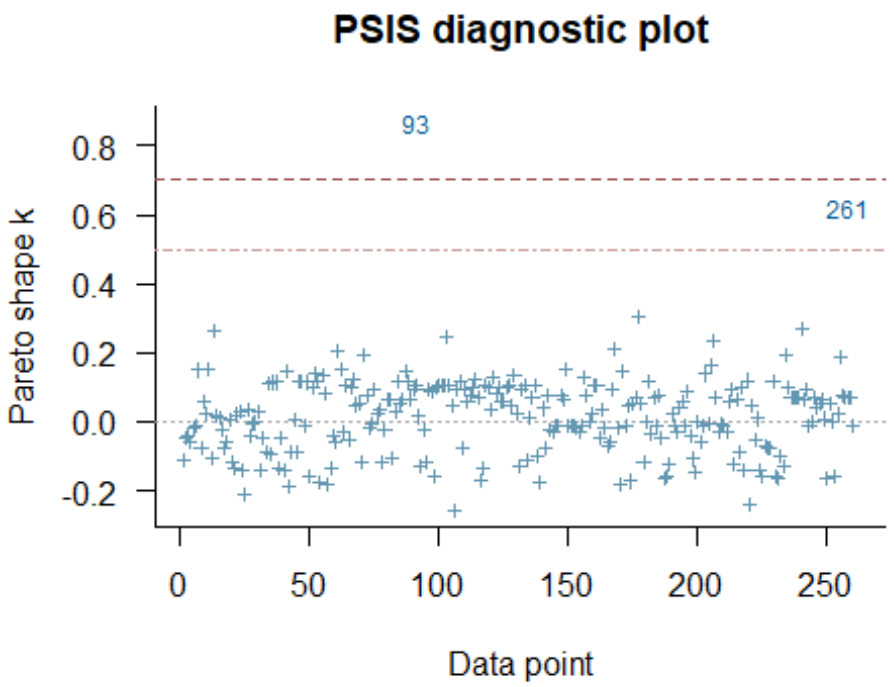
The excessive number of values close to 0 indicates that the model is under-dispersed compared to the data, and we should consider a model that allows for greater dispersion.

9.2.6 Try alternative model with more flexibility

Here we will try *negative binomial* regression, which is commonly used for overdispersed count data.

Unlike the Poisson distribution, the negative binomial distribution allows the conditional mean and variance of y to differ.

```
fit2 <- update(fit1, family = neg_binomial_2)

loo2 <- loo(fit2, save_psis = TRUE, cores = 2)

## Warning: Found 1 observation(s) with a pareto_k > 0.7. We recommend
calling 'loo' again with argument 'k_threshold = 0.7' in order to calculate
the ELPD without the assumption that these observations are negligible. This
will refit the model 1 times to compute the ELPDs for the problematic
observations directly.

print(loo2)

##
## Computed from 4000 by 262 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -895.6 37.8
## p_loo         6.7  2.7
## looic      1791.3 75.5
## ------
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     260   99.2%   2615
##  (0.5, 0.7]   (ok)         1    0.4%   377
##    (0.7, 1]   (bad)        1    0.4%   28
##    (1, Inf)   (very bad)   0    0.0%   <NA>
## See help('pareto-k-diagnostic') for details.

plot(loo2, label_points = TRUE)
```



Using the label_points argument will label any *k* values larger than 0.7 with the index of the corresponding data point. These high values are often the result of model misspecification and frequently correspond to data points that would be considered "outliers" in the data and surprising according to the model *Gabry et al (2019)*. Unfortunately, while large *k* values are a useful indicator of model misspecification, small k values are not a guarantee that a model is well-specified.

If there are a small number of problematic *k* values then we can use a feature in **rstanarm** that lets us refit the model once for each of these problematic observations. Each time the model is refit, one of the observations with a high *k* value is omitted and the LOO calculations are performed exactly for that observation. The results are then recombined with the approximate LOO calculations already carried out for the observations without problematic *k* values :

```
if (any(pareto_k_values(loo2) > 0.7)) {
  loo2 <- loo(fit2, save_psis = TRUE, k_threshold = 0.7)
}
```

```
## 1 problematic observation(s) found.
## Model will be refit 1 times.

##
## Fitting model 1 out of 1 (leaving out observation 93)
```

```
print(loo2)
```

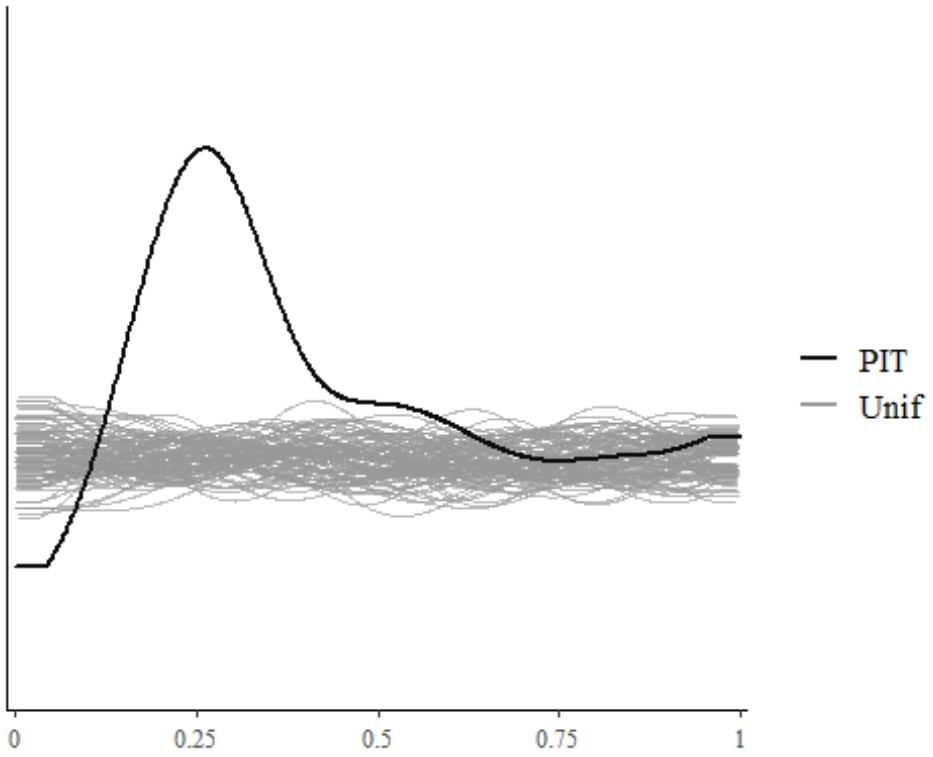
```
##
## Computed from 4000 by 262 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -895.5 37.7
## p_loo         6.6  2.6
## looic      1791.1 75.4
## ------
## Monte Carlo SE of elpd_loo is 0.2.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     260   99.6%   2615
##  (0.5, 0.7]   (ok)         1    0.4%   377
##    (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.
```

In the print output we can see that the Monte Carlo SE is small compared to the other uncertainties.

On the other hand, `p_loo` is about 7 and still a bit higher than the total number of parameters in the model. This indicates that there is almost certainly still some degree of model misspecification, but this is much better than the `p_loo` estimate for the Poisson model.

For further model checking we again examine the LOO-PIT values.

```
yrep <- posterior_predict(fit2)

ppc_loo_pit_overlay(roaches$y, yrep, lw = weights(loo2$psis_object))

## NOTE: The kernel density estimate assumes continuous observations and is
not optimal for discrete observations.
```



The plot for the negative binomial model looks better than the Poisson plot, but we still see that this model is not capturing all of the essential features in the data.

9.2.7 Comparing the models on expected log predictive density

We can use the `loo_compare` function to compare our two models on expected log predictive density (ELPD) for new data:

```
loo_compare(loo1, loo2)
```

```
##      elpd_diff se_diff
## fit2     0.0       0.0
## fit1 -5352.2     709.2
```

The difference in ELPD is much larger than several times the estimated standard error of the difference again indicating that the negative-binomial model is expected to have better predictive performance than the Poisson model. However, according to the LOO-PIT checks there is still some misspecification, and a reasonable guess is that a hurdle or zero-inflated model would be an improvement (we leave that for another case study).

10 Estimating ANOVA Models with rstanarm

```
library(rstanarm)
help(pack = rstanarm)
rstanarm::aov
```

10.1 Introduction

This vignette explains how to estimate ANalysis Of VAriance (ANOVA) models using the `stan_aov` function in the **rstanarm** package

The four steps of a Bayesian analysis are

1. Specify a joint distribution for the outcome(s) and all the unknowns, which typically takes the form of a marginal prior distribution for the unknowns multiplied by a likelihood for the outcome(s) conditional on the unknowns. This joint distribution is proportional to a posterior distribution of the unknowns conditional on the observed data
2. Draw from posterior distribution using Markov Chain Monte Carlo (MCMC).
3. Evaluate how well the model fits the data and possibly revise the model.
4. Draw from the posterior predictive distribution of the outcome(s) given interesting values of the predictors in order to visualize how a manipulation of a predictor affects (a function of) the outcome(s).

Steps 3 and 4 are covered in more depth by the vignette entitled "How to Use the **rstanarm** Package". This vignette focuses on Step 1 when the likelihood is the product of independent normal distributions. We also demonstrate that Step 2 is not entirely automatic because it is sometimes necessary to specify some additional tuning parameters in order to obtain optimally efficient results.

10.2 Likelihood

The likelihood for one observation under a linear model can be written as a conditionally normal PDF

$$\frac{1}{\sigma_\epsilon \sqrt{2\pi}} e^{-\frac{1}{2}(\frac{y-\mu}{\sigma_\epsilon})^2},$$

where $\mu = \alpha + x^\top \beta$ is a linear predictor and σ_ϵ is the standard deviation of the error in predicting the outcome, y. The likelihood of the entire sample is the product of N individual likelihood contributions.

An ANOVA model can be considered a special case of the above linear regression model where each of the K predictors in $\mathbf{x}$ is a dummy variable indicating membership in a group. An equivalent linear predictor can be written as $\mu_j = \alpha + \alpha_j$, which expresses the conditional expectation of the outcome in the j^{th} group as the sum of a common mean, α, and a group-specific deviation from the common mean, α_j.

10.3 Priors

If we view the ANOVA model as a special case of a linear regression model with only dummy variables as predictors, then the model could be estimated using the prior

specification in the `stan_lm` function. In fact, this is exactly how the `stan_aov` function is coded. These functions require the user to specify a value for the prior location (by default the mode) of the R^2, the proportion of variance in the outcome attributable to the predictors under a linear model. This prior specification is appealing in an ANOVA context because of the fundamental identity

$$SS_{total} = SS_{model} + SS_{error}$$

,

where SS stands for sum-of-squares. If we normalize this identity, we obtain the tautology $1 = R^2 + (1 - R^2)$ but it is reasonable to expect a researcher to have a plausible guess for R^2 before conducting an ANOVA. See the vignette for the `stan_lm` function (regularized linear models) for more information on this approach.

If we view the ANOVA model as a difference of means, then the model could be estimated using the prior specification in the `stan_lmer` function. In the syntax popularized by the **lme4** package, `y ~ 1 + (1|group)` represents a likelihood where $\mu_j = \alpha + \alpha_j$ and α_j is normally distributed across the J groups with mean zero and some unknown standard deviation. The `stan_lmer` function specifies that this standard deviation has a Gamma prior with, by default, both its shape and scale parameters equal to 1, which is just an standard exponential distribution. However, the shape and scale parameters can be specified as other positive values. This approach also requires specifying a prior distribution on the standard deviation of the errors that is independent of the prior distribution for each α_j. See the vignette for the `stan_glmer` function (**lme4**-style models using **rstanarm**) for more information on this approach.

10.4 Example

We will utilize an example from the **HSAUR3** package. *A Handbook of Statistical Analyses Using R* (3rd Edition). This book is frequentist in nature and we will show how to obtain the corresponding Bayesian results.

The model in section 4.3.1 analyzes an experiment where rats were subjected to different diets in order to see how much weight they gained. The experimental factors were whether their diet had low or high protein and whether the protein was derived from beef or cereal. Before seeing the data, one might expect that a moderate proportion of the variance in weight gain might be attributed to protein (source) in the diet. The frequentist ANOVA estimates can be obtained:

```
library(ggplot2)
library(bayesplot)
theme_set(bayesplot::theme_default())

# Load the Data
data("weightgain", package = "HSAUR3")

coef(aov(weightgain ~ source * type, data = weightgain))
```

```
##          (Intercept)         sourceCereal              typeLow
sourceCereal:typeLow
##                100.0               -14.1                -20.8
18.8
```

To obtain Bayesian estimates we can prepend `stan_` to `aov` and specify the prior location of the R^2 as well as optionally the number of cores that the computer is allowed to utilize :

```
library(rstanarm)
post1 <- stan_aov(weightgain ~ source * type,
                  data = weightgain,
                  prior = R2(location = 0.5),
                  adapt_delta = 0.999,
                  seed = 12345, refresh = 0)
post1
```

```
## stan_aov
##  family:       gaussian [identity]
##  formula:      weightgain ~ source * type
##  observations: 40
##  predictors:   4
## ------
##                      Median MAD_SD
## (Intercept)           98.8    4.8
## sourceCereal         -12.7    6.3
## typeLow              -18.6    6.5
## sourceCereal:typeLow  16.9    8.7
##
## Auxiliary parameter(s):
##               Median MAD_SD
## R2             0.2    0.1
## log-fit_ratio  0.0    0.1
## sigma         14.7    1.7
##
## ANOVA-like table:
##                     Median MAD_SD
## Mean Sq source      547.3  424.6
## Mean Sq type        967.5  593.4
## Mean Sq source:type 714.8  693.7
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

Here we have specified `adapt_delta = 0.999` to decrease the stepsize and largely prevent divergent transitions. See the Troubleshooting section in the main rstanarm vignette for more details about `adapt_delta`. Also, our prior guess that $R^2 = 0.5$ was overly optimistic. However, the frequentist estimates presumably overfit the data even more.

Alternatively, we could prepend `stan_` to `lmer` and specify the corresponding priors

```
post2 <- stan_lmer(weightgain ~ 1 + (1|source) + (1|type) + (1|source:type),
                   data = weightgain, prior_intercept = cauchy(),
                   prior_covariance = decov(shape = 2, scale = 2),
                   adapt_delta = 0.999, seed = 12345, refresh = 0)

post2
```

```
## stan_lmer
##  family:       gaussian [identity]
##  formula:      weightgain ~ 1 + (1 | source) + (1 | type) + (1 |
source:type)
##  observations: 40
## ------
##             Median MAD_SD
## (Intercept) 1.0    5.0
##
## Auxiliary parameter(s):
##       Median MAD_SD
## sigma 14.9    1.7
##
## Error terms:
##  Groups      Name        Std.Dev.
##  source:type (Intercept) 39
##  type        (Intercept) 65
##  source      (Intercept) 64
##  Residual                15
## Num. levels: source:type 4, type 2, source 2
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

Comparing these two models using the `loo` function in the **loo** package reveals a negligible preference for the first approach that is almost entirely due to its having a smaller number of effective parameters as a result of the more regularizing priors. However, the difference is so small that it may seem advantageous to present the second results which are more in line with a mainstream Bayesian approach to an ANOVA model.

```
library(loo)
loo(post1)
```

```
##
## Computed from 4000 by 40 log-likelihood matrix
##
##          Estimate  SE
## elpd_loo   -167.7 4.1
## p_loo         4.4 0.9
## looic       335.5 8.1
## ------
## Monte Carlo SE of elpd_loo is 0.1.
```

```
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.

loo(post2)

##
## Computed from 4000 by 40 log-likelihood matrix
##
##          Estimate  SE
## elpd_loo   -168.1 4.0
## p_loo         4.7 0.9
## looic       336.2 8.0
## ------
## Monte Carlo SE of elpd_loo is 0.0.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.

# loo(post1, post2)
```

10.5 Conclusion

This vignette has compared and contrasted two approaches to estimating an ANOVA model with Bayesian techniques using the **rstanarm** package. They both have the same likelihood, so the (small in this case) differences in the results are attributable to differences in the priors.

The `stan_aov` approach just calls `stan_lm` and thus only requires a prior location on the R^2 of the linear model. This seems rather easy to do in the context of an ANOVA decomposition of the total sum-of-squares in the outcome into model sum-of-squares and residual sum-of-squares.

The `stan_lmer` approach just calls `stan_glm` but specifies a normal prior with mean zero for the deviations from α across groups. This is more in line with what most Bayesians would do naturally — particularly if the factors were considered "random" — but also requires a prior for α, σ, and the standard deviation of the normal prior on the group-level intercepts. The `stan_lmer` approach is very flexible and might be more appropriate for more complicated experimental designs.

11 JAGS : Just Another Gibs Sampler

BUGS : Bayesian Analysis Using Gibs Sampling

Hilbe et.al(2017) *Bayesian Models for astrophysical data using R, JAGS, Python and Stan*, Cambridge University Press.

11.1 Bayesian Synthetic Normal Model in R using JAGS

11.1.1 Normal linear model in R using JAGS

```
require(R2jags)
set.seed(1056) # set seed to replicate example

nobs = 5000 # number of observations in model
x1 <- runif(nobs) # random uniform variable

beta0 = 2.0 # intercept
beta1 = 3.0 # slope or coefficient

xb <- beta0 + beta1 * x1 # linear predictor, xb
y <- rnorm(nobs, xb, sd=1) # create y as adjusted random normal variate

# Construct data dictionary
X <- model.matrix(~ 1 + x1)
K <- ncol(X)

model.data <- list(Y = y, # response variable
X = X, # predictors
K = K, # number of predictors including the intercept
N = nobs # sample size
)

# Model set up with JAGS
NORM <- "model{
# Diffuse normal priors for predictors
for (i in 1:K) { beta[i] ~ dnorm(0, 0.0001) }
# Uniform prior for standard deviation
tau <- pow(sigma, -2) # precision
sigma ~ dunif(0, 100) # standard deviation
# Likelihood function
for (i in 1:N){
Y[i]~dnorm(mu[i],tau)
mu[i] <- eta[i]
eta[i] <- inprod(beta[], X[i,])
}
}"

# Initial values
inits <- function () {
list(beta = rnorm(K, 0, 0.01))
}

# Parameters to be displayed
params <- c("beta", "sigma")

# MCMC
```

```
normfit <- jags(data = model.data,
inits = inits,
parameters = params,
model = textConnection(NORM),
n.chains = 3,
n.iter = 15000,
n.thin = 1,
n.burnin = 10000)

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 5000
##    Unobserved stochastic nodes: 3
##    Total graph size: 25012
##
## Initializing model
##
##   |                                                                      |
|   0%  |                                                                      |++
|   3%  |                                                                      |+++
|   6%  |                                                                      |++++
|   9%  |                                                                      |++++++
|  12%  |                                                                      |++++++++
|  15%  |                                                                      |+++++++++
|  18%  |
|+++++++++                                                 |  21%  |
|++++++++++++                                              |  24%  |
|++++++++++++++                                            |  27%  |
|+++++++++++++++                                           |  30%  |
|++++++++++++++++                                          |  33%  |
|++++++++++++++++++                                        |  36%  |
|++++++++++++++++++++                                      |  39%  |
|++++++++++++++++++++++                                    |  42%  |
|++++++++++++++++++++++++                                  |  45%  |
|++++++++++++++++++++++++++                                |  48%  |
|++++++++++++++++++++++++++++                              |  51%  |
|++++++++++++++++++++++++++++++                            |  54%  |
|++++++++++++++++++++++++++++++++                          |  57%  |
|++++++++++++++++++++++++++++++++++                        |  60%  |
|++++++++++++++++++++++++++++++++++++                      |  63%  |
|++++++++++++++++++++++++++++++++++++++                    |  66%  |
|++++++++++++++++++++++++++++++++++++++++                  |  69%  |
|++++++++++++++++++++++++++++++++++++++++++                |  72%  |
|++++++++++++++++++++++++++++++++++++++++++++              |  75%  |
|++++++++++++++++++++++++++++++++++++++++++++++            |  78%  |
|++++++++++++++++++++++++++++++++++++++++++++++++          |  81%  |
|++++++++++++++++++++++++++++++++++++++++++++++++++        |  84%  |
|++++++++++++++++++++++++++++++++++++++++++++++++++++++    |  87%  |
```

```
|++++++++++++++++++++++++++++++++++++++++++++++      |  90%  |
|++++++++++++++++++++++++++++++++++++++++++++++++    |  93%  |
|++++++++++++++++++++++++++++++++++++++++++++++++++  |  96%  |
|++++++++++++++++++++++++++++++++++++++++++++++++++++|  99%  |
|++++++++++++++++++++++++++++++++++++++++++++++++++++| 100%
##   |                                                                      |
|   0%  |                                                                      |***
|   6%  |                                                                      |******
|  12%  |                                                                      |*********
|  18%  |
|************                                                          |  24%  |
|***************                                                       |  30%  |
|******************                                                    |  36%  |
|*********************                                                 |  42%  |
|************************                                              |  48%  |
|****************************                                          |  54%  |
|*******************************                                       |  60%  |
|**********************************                                    |  66%  |
|*************************************                                 |  72%  |
|****************************************                              |  78%  |
|*******************************************                           |  84%  |
|**********************************************                        |  90%  |
|**************************************************                    |  96%  |
|***************************************************| 100%
```

```
print(normfit, intervals = c(0.025, 0.975), digits = 2)
```

```
## Inference for Bugs model at "5", fit using jags,
##  3 chains, each with 15000 iterations (first 10000 discarded)
##  n.sims = 15000 iterations saved
##           mu.vect sd.vect     2.5%    97.5% Rhat n.eff
## beta[1]      1.99    0.03     1.94     2.05    1 15000
## beta[2]      3.01    0.05     2.91     3.10    1 15000
## sigma        1.00    0.01     0.98     1.02    1 15000
## deviance 14175.64    2.39 14172.91 14181.84    1 15000
##
## 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 = 2.9 and DIC = 14178.5
## DIC is an estimate of expected predictive error (lower deviance is
better).
```

```
# Names of Model
names(normfit)
```

```
## [1] "model"              "BUGSoutput"         "parameters.to.save"
"model.file"
## [5] "n.iter"             "DIC"
```

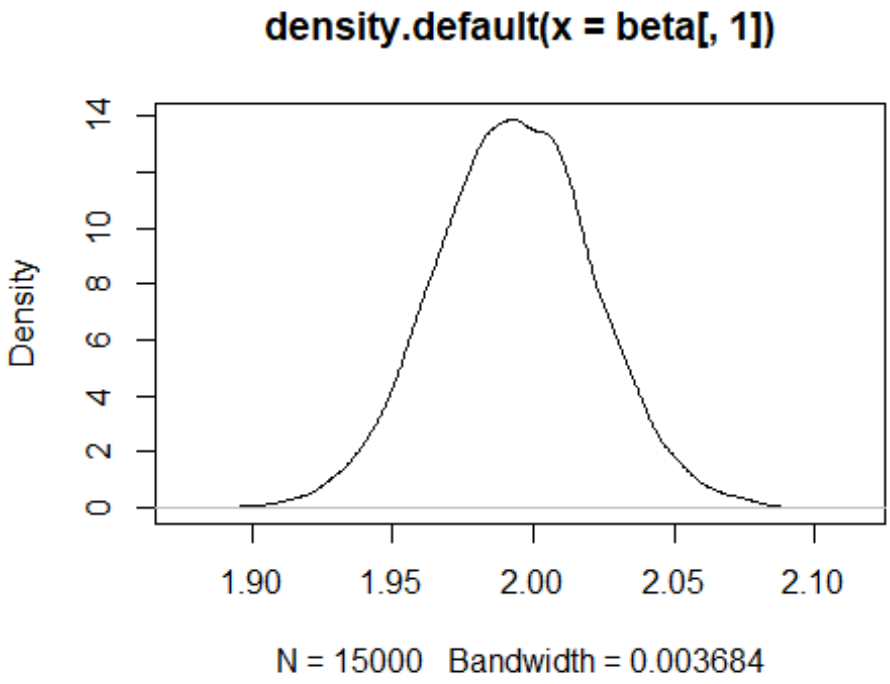
```r
# Names of BUGSoutput
out1<- normfit$BUGSoutput
names(out1)
```

```
##  [1] "n.chains"        "n.iter"          "n.burnin"        "n.thin"
"n.keep"
##  [6] "n.sims"          "sims.array"      "sims.list"       "sims.matrix"
"summary"
## [11] "mean"            "sd"              "median"          "root.short"
"long.short"
## [16] "dimension.short" "indexes.short"   "last.values"     "program"
"model.file"
## [21] "isDIC"           "DICbyR"          "pD"              "DIC"
```

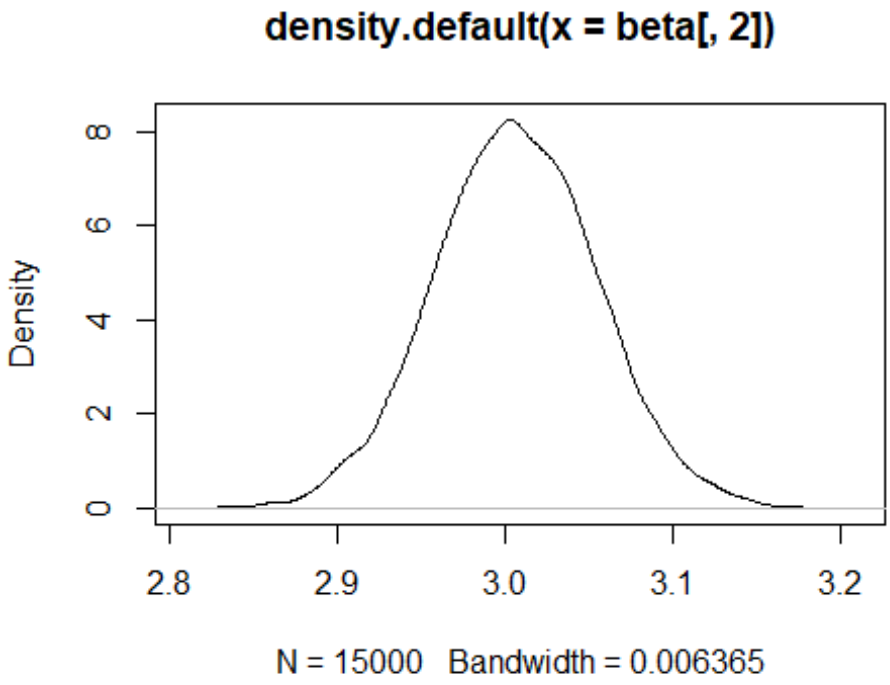
```r
# beta's
beta<- out1$sims.list$beta
sigma<- out1$sims.list$sigma
head(beta)
```

```
##          [,1]     [,2]
## [1,] 1.994523 3.045283
## [2,] 1.958757 3.058549
## [3,] 1.971816 3.071811
## [4,] 2.013860 2.988573
## [5,] 2.028180 2.957303
## [6,] 2.021864 2.935898
```

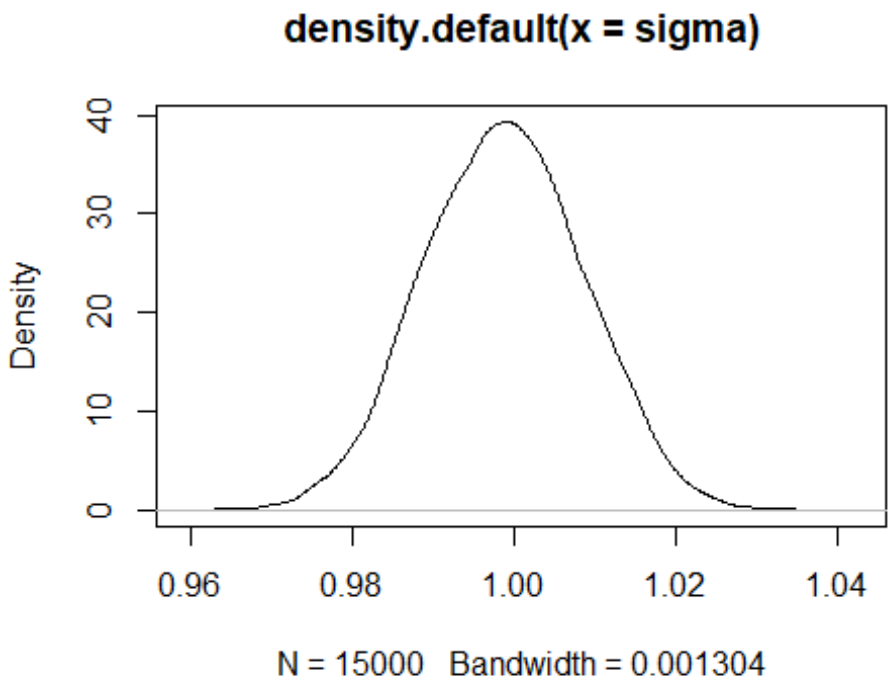
```r
# Plot for beta_0
plot(density(beta[,1]))
```




```
# Plot for beta_1
plot(density(beta[,2]))
```



```
# Plot for sigma
plot(density(sigma))
```



11.2 Bayesian fitting of binomial model using JAGS

Simulation of data

```
set.seed(33559)

nobs<- 2000

m<- 1 + rpois(nobs, 5)

x1<- runif(nobs)

x2<- runif(nobs)

xb<- -2 - 1.5 * x1 + 3 * x2

exb<- exp(xb)

p<- exb/(1 + exb) # prob of p=0

y<- rbinom(nobs, prob=p, size=m)

bindata<- data.frame(y=y, m=m, x1, x2)
```

11.2.1 ML estimation of binomial model

To get ML estimator we shall make use of `glm()` function of R

```
noty<- m - y

mybin<- glm(cbind(y, noty) ~ x1 + x2, family=binomial, data=bindata)

summary(mybin)

##
## Call:
## glm(formula = cbind(y, noty) ~ x1 + x2, family = binomial, data = bindata)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.0346  -0.9230  -0.1723   0.6087   3.6340
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.99406    0.06354  -31.38   <2e-16 ***
## x1          -1.59785    0.08162  -19.58   <2e-16 ***
## x2           3.09066    0.08593   35.97   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4035.8  on 1999  degrees of freedom
## Residual deviance: 2131.4  on 1997  degrees of freedom
## AIC: 4951.5
##
## Number of Fisher Scoring iterations: 4
```

It may be observed that the estimates are very close to the constants provided.

11.2.2 Bayesian Analysis with JAGS

```
library(R2jags)

X<- model.matrix(~ x1 + x2, data = bindata)

K<- ncol(X)

model.data <- list(Y = bindata$y,
                   N = nrow(bindata),
                   X = X,
                   K = K,
                   m = bindata$m)

cat("
```

```
model{
# Priors
# Diffuse normal priors betas
for (i in 1:K) { beta[i] ~ dnorm(0, 0.0001)}
# Likelihood
for (i in 1:N){
Y[i] ~ dbin(p[i],m[i])
logit(p[i]) <- eta[i]
eta[i] <- inprod(beta[], X[i,])
}
}
",file='binom.txt')

inits<- function () {
list(
  beta = rnorm(K, 0, 0.1)
) }

params<- c("beta")

BINL<- jags(data = model.data,
            inits = inits,
            parameters = params,
            model.file = "binom.txt",
            n.thin = 1,
            n.chains = 3,
            n.burnin = 3000,
            n.iter = 5000)

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 2000
##    Unobserved stochastic nodes: 3
##    Total graph size: 16008
##
## Initializing model
##
##   |                                                                      |
|   0%  |                                                                      |**
|   5%  |                                                                      |*****
|  10%  |                                                                      |********
|  15%  |
|**********                                                    |  20%  |
|************                                                  |  25%  |
|***************                                               |  30%  |
|******************                                            |  35%  |
|*********************                                         |  40%  |
|************************                                      |  45%  |
```

```
|************************                         |  50%  |
|*************************                        |  55%  |
|**************************                       |  60%  |
|***************************                      |  65%  |
|****************************                     |  70%  |
|*****************************                    |  75%  |
|******************************                   |  80%  |
|*******************************                  |  85%  |
|********************************                 |  90%  |
|*********************************                |  95%  |
|**********************************************| 100%

print(BINL, intervals=c(0.025, 0.975), digits=3)

## Inference for Bugs model at "binom.txt", fit using jags,
##  3 chains, each with 5000 iterations (first 3000 discarded)
##  n.sims = 6000 iterations saved
##           mu.vect sd.vect     2.5%    97.5%  Rhat n.eff
## beta[1]    -1.995   0.065   -2.117   -1.868 1.001  4300
## beta[2]    -1.597   0.083   -1.760   -1.436 1.002  2100
## beta[3]     3.091   0.090    2.920    3.262 1.002  2500
## deviance 4948.690   6.486 4945.717 4954.962 1.005  6000
##
## 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 = 21.0 and DIC = 4969.7
## DIC is an estimate of expected predictive error (lower deviance is
better).
```

12 Hierarchical Models

Bayesian models we have discussed so for have been based on likelihood and the mixture of prior and likelihood. One of the key assumption of a likelihood that each component of observation in the distribution is independent to other observation.

When there is correlation are time-series auto-correlation in the data caused by clustered, nested of pannel structured data, statistician must make adjustment to the model in order to avoid bias in the interpretation of the parameter.

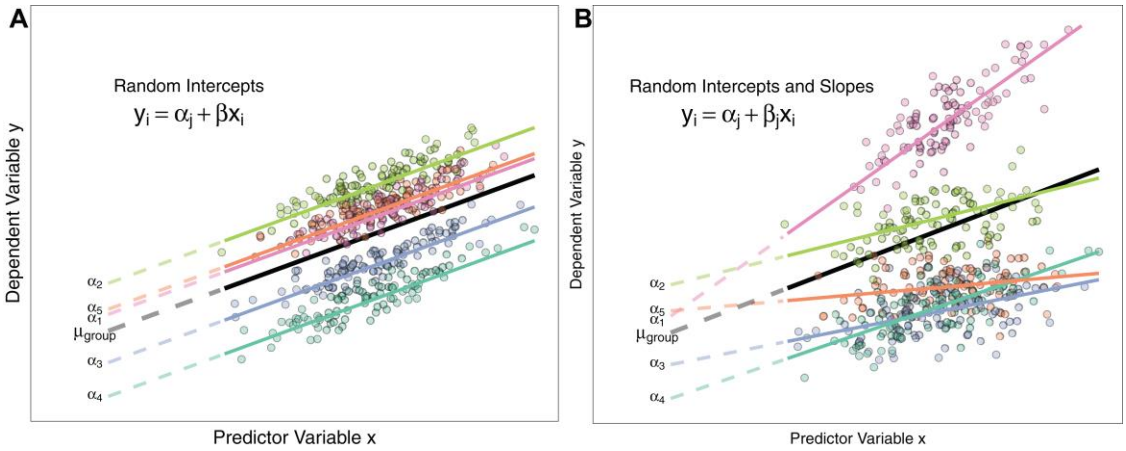
Example : Suppose that a study is made of student scores on the graduate record examination (GRE) subject area test in Physics. The score are taking from applicants for *Ph.D* program in *Astronomy* and *Astrophysics*. We wish to model the test score on the bases of applicant cummulative *GPA* on under-graduates courses taken is *Mathematics* and *Physics*. The model can thus appears as

$$score = \beta_0 + \beta_1 \times GPA + \epsilon$$

where β_0 and β_1 are parameters and ϵ is the error term.

What may concern us, though, is that the *GPA* results could differ depending on where the applicant did their under- graduate work.

Students taking courses at one university may differ from students from at another university. Moreover the type of courses and expectation what must be learn to achieve a given grade may differ from university in other words *GPA* results within a university may be more correlated than the result between the university.



Note that **slope** are *common* but **intercept** are *random*.

The random intercept model conceptually divides the data into separate model for each university that is part of study. The coefficients are assumed to remain same but the intercept varies between universities. If the mean *GPA* result within each university are nearly the same, as the overall mean *GPA* results then the universities will differ in their results. If they differ considerably however we know that there is university effect.

The general formula for random intercept model for a single predictor can be given as

$$y_{ij} = \beta_0 + \beta_1 \times x_{ij} + \xi_j + \epsilon$$

where ξ_j is the *university effect* or *location effect*.

```
set.seed(1656)
N <- 4500
NGroups <- 20
x1 <- runif(N)
x2 <- runif(N)
Groups <- rep(1:20, each = 225) # 20 groups, each with 225 observations
a <- rnorm(NGroups, mean = 0, sd = 0.5)
mu <- 1 + 0.2 * x1 - 0.75 * x2 + a[Groups]
y <- rnorm(N, mean=mu, sd=2)
normr <- data.frame(
y = y,
x1 = x1,
x2 = x2,
```

```
Groups = Groups,
RE = a[Groups]
)

head(normr)

##            y        x1        x2 Groups        RE
## 1 -1.7908844 0.4748350 0.8209906      1 0.5786043
## 2  0.6513202 0.2725096 0.3062738      1 0.5786043
## 3  0.5886806 0.9555390 0.8619673      1 0.5786043
## 4  0.4948763 0.6180860 0.1885107      1 0.5786043
## 5  1.2084331 0.3918778 0.6929373      1 0.5786043
## 6  1.0519377 0.5826335 0.8278024      1 0.5786043

library(rstanarm)
m1<- stan_lmer(y~x1+x2+(1|Groups), data = normr, refresh = 0)

print(m1)

## stan_lmer
##  family:       gaussian [identity]
##  formula:      y ~ x1 + x2 + (1 | Groups)
##  observations: 4500
## ------
##             Median MAD_SD
## (Intercept)  0.7    0.2
## x1           0.2    0.1
## x2          -0.7    0.1
##
## Auxiliary parameter(s):
##       Median MAD_SD
## sigma 2.0    0.0
##
## Error terms:
##  Groups   Name        Std.Dev.
##  Groups   (Intercept) 0.8
##  Residual             2.0
## Num. levels: Groups 20
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg

# Accuracy
0.8^2/(0.8^2+2^2)

## [1] 0.137931

names(m1)
```

```
##  [1] "coefficients"      "ses"               "fitted.values"
"linear.predictors"
##  [5] "residuals"         "df.residual"       "covmat"            "y"
##  [9] "x"                 "model"             "data"              "family"
## [13] "offset"            "weights"           "prior.weights"
"contrasts"
## [17] "na.action"         "formula"           "terms"
"prior.info"
## [21] "dropped_cols"      "algorithm"         "stan_summary"      "stanfit"
## [25] "rstan_version"     "call"              "stan_function"     "glmod"

m1$stan_summary

##                                          mean      se_mean
sd          2.5%
## (Intercept)                             6.998654e-01 0.0100348722
0.20031757  2.932273e-01
## x1                                       2.044828e-01 0.0017996893
0.10427360 -3.457459e-03
## x2                                      -6.904762e-01 0.0017814823
0.10657424 -8.893489e-01
## b[(Intercept) Groups:1]                  8.693745e-01 0.0100568946
0.22365847  4.437142e-01
## b[(Intercept) Groups:2]                  3.563621e-01 0.0101245549
0.22642196 -8.762767e-02
## b[(Intercept) Groups:3]                  4.494643e-02 0.0102310271
0.22306426 -3.962626e-01
## b[(Intercept) Groups:4]                  4.105903e-01 0.0100863157
0.22537074 -4.089599e-02
## b[(Intercept) Groups:5]                 -1.971971e-01 0.0103584038
0.22446235 -6.381343e-01
## b[(Intercept) Groups:6]                 -1.086843e+00 0.0102873494
0.22081259 -1.523163e+00
## b[(Intercept) Groups:7]                 -9.300441e-01 0.0098227879
0.22483529 -1.370617e+00
## b[(Intercept) Groups:8]                 -1.967510e-02 0.0102129245
0.22343093 -4.599989e-01
## b[(Intercept) Groups:9]                  4.635555e-01 0.0102378096
0.22431239  2.292156e-02
## b[(Intercept) Groups:10]                 3.604525e-01 0.0102150331
0.22412876 -7.782598e-02
## b[(Intercept) Groups:11]                -5.414124e-01 0.0100862554
0.22367575 -9.756372e-01
## b[(Intercept) Groups:12]                -4.652187e-03 0.0100931928
0.22312643 -4.453002e-01
## b[(Intercept) Groups:13]                -9.927815e-01 0.0099979247
0.22324313 -1.435665e+00
## b[(Intercept) Groups:14]                 1.243382e+00 0.0099440057
0.22158012  8.199812e-01
## b[(Intercept) Groups:15]                 1.285328e+00 0.0099112790
```

```
0.22218501  8.574053e-01
## b[(Intercept) Groups:16]               7.383185e-02 0.0101777131
0.22230130 -3.553375e-01
## b[(Intercept) Groups:17]              -3.890306e-01 0.0100036473
0.22344766 -8.334678e-01
## b[(Intercept) Groups:18]               3.382127e-03 0.0099895831
0.22347136 -4.310050e-01
## b[(Intercept) Groups:19]              -1.282616e+00 0.0101942298
0.22133479 -1.727912e+00
## b[(Intercept) Groups:20]               6.193786e-01 0.0102415822
0.22163080  1.832895e-01
## b[(Intercept) Groups:_NEW_Groups]      1.655032e-02 0.0158211832
0.81026005 -1.507711e+00
## sigma                                  1.994049e+00 0.0003715233
0.02021602  1.954373e+00
## Sigma[Groups:(Intercept),(Intercept)]  6.419501e-01 0.0105311065
0.24642146  3.217026e-01
## mean_PPD                               4.700048e-01 0.0006785590
0.04203483  3.878332e-01
## log-posterior                         -9.525621e+03 0.1616602503
4.67665211 -9.535702e+03
##                                                 10%           25%
50%           75%
## (Intercept)                            4.514669e-01  5.742321e-01
6.981443e-01  8.316957e-01
## x1                                     7.180069e-02  1.353248e-01
2.040080e-01  2.738808e-01
## x2                                    -8.276409e-01 -7.627247e-01 -
6.910523e-01 -6.185837e-01
## b[(Intercept) Groups:1]                5.878675e-01  7.218910e-01
8.694075e-01  1.012685e+00
## b[(Intercept) Groups:2]                6.928715e-02  2.068203e-01
3.576246e-01  5.044075e-01
## b[(Intercept) Groups:3]               -2.297364e-01 -1.044687e-01
4.336993e-02  1.930414e-01
## b[(Intercept) Groups:4]                1.334925e-01  2.635153e-01
4.102874e-01  5.608832e-01
## b[(Intercept) Groups:5]               -4.853139e-01 -3.449349e-01 -
1.928147e-01 -4.713914e-02
## b[(Intercept) Groups:6]               -1.365221e+00 -1.229077e+00 -
1.086030e+00 -9.427711e-01
## b[(Intercept) Groups:7]               -1.214418e+00 -1.076911e+00 -
9.276050e-01 -7.846275e-01
## b[(Intercept) Groups:8]               -3.010120e-01 -1.672480e-01 -
2.185092e-02  1.250671e-01
## b[(Intercept) Groups:9]                1.881844e-01  3.131540e-01
4.654106e-01  6.121505e-01
## b[(Intercept) Groups:10]               7.645346e-02  2.137751e-01
3.606504e-01  5.099226e-01
## b[(Intercept) Groups:11]              -8.306827e-01 -6.840420e-01 -
```

```
5.372992e-01 -3.989869e-01
## b[(Intercept) Groups:12]              -2.825084e-01 -1.515210e-01 -
5.155782e-03  1.404510e-01
## b[(Intercept) Groups:13]              -1.279777e+00 -1.134739e+00 -
9.941357e-01 -8.467125e-01
## b[(Intercept) Groups:14]               9.582846e-01  1.097872e+00
1.243092e+00  1.384207e+00
## b[(Intercept) Groups:15]               9.991660e-01  1.140331e+00
1.286042e+00  1.427683e+00
## b[(Intercept) Groups:16]              -2.038405e-01 -7.364222e-02
7.256908e-02  2.169067e-01
## b[(Intercept) Groups:17]              -6.823130e-01 -5.315097e-01 -
3.881243e-01 -2.420937e-01
## b[(Intercept) Groups:18]              -2.825936e-01 -1.439430e-01
4.529531e-03  1.510944e-01
## b[(Intercept) Groups:19]              -1.562242e+00 -1.428922e+00 -
1.280833e+00 -1.136554e+00
## b[(Intercept) Groups:20]               3.312017e-01  4.801161e-01
6.225063e-01  7.621549e-01
## b[(Intercept) Groups:_NEW_Groups]     -9.940374e-01 -5.235697e-01
2.616590e-02  5.324696e-01
## sigma                                  1.968288e+00  1.980710e+00
1.994222e+00  2.006959e+00
## Sigma[Groups:(Intercept),(Intercept)]  3.911536e-01  4.718768e-01
5.882196e-01  7.532801e-01
## mean_PPD                               4.154634e-01  4.402206e-01
4.702420e-01  4.983075e-01
## log-posterior                         -9.531821e+03 -9.528559e+03 -
9.525261e+03 -9.522340e+03
##                                                 90%         97.5%
n_eff      Rhat
## (Intercept)                            9.494080e-01  1.086525e+00
398.4872 1.0047749
## x1                                     3.375629e-01  4.075119e-01
3357.0177 0.9992754
## x2                                    -5.538771e-01 -4.800703e-01
3578.8333 1.0004803
## b[(Intercept) Groups:1]                1.142761e+00  1.332444e+00
494.5873 1.0031084
## b[(Intercept) Groups:2]                6.415100e-01  8.226875e-01
500.1326 1.0037539
## b[(Intercept) Groups:3]                3.292260e-01  4.852657e-01
475.3588 1.0024564
## b[(Intercept) Groups:4]                6.923806e-01  8.563748e-01
499.2637 1.0030908
## b[(Intercept) Groups:5]                8.067256e-02  2.419021e-01
469.5711 1.0032374
## b[(Intercept) Groups:6]               -8.125991e-01 -6.474251e-01
460.7238 1.0036752
## b[(Intercept) Groups:7]               -6.487678e-01 -4.758068e-01
```

```
523.9133 1.0028688
## b[(Intercept) Groups:8]                 2.590316e-01  4.213102e-01
478.6150 1.0026415
## b[(Intercept) Groups:9]                 7.456417e-01  9.102542e-01
480.0566 1.0033490
## b[(Intercept) Groups:10]                6.438478e-01  8.142929e-01
481.4106 1.0037911
## b[(Intercept) Groups:11]               -2.634512e-01 -8.742337e-02
491.7880 1.0038166
## b[(Intercept) Groups:12]                2.750607e-01  4.392347e-01
488.7029 1.0031105
## b[(Intercept) Groups:13]               -7.171734e-01 -5.463674e-01
498.5819 1.0032226
## b[(Intercept) Groups:14]                1.519032e+00  1.700525e+00
496.5224 1.0030612
## b[(Intercept) Groups:15]                1.563950e+00  1.735904e+00
502.5394 1.0023403
## b[(Intercept) Groups:16]                3.582228e-01  5.239924e-01
477.0716 1.0021813
## b[(Intercept) Groups:17]               -1.096636e-01  5.379628e-02
498.9245 1.0033969
## b[(Intercept) Groups:18]                2.889624e-01  4.439656e-01
500.4365 1.0033001
## b[(Intercept) Groups:19]               -1.003982e+00 -8.436580e-01
471.4010 1.0040941
## b[(Intercept) Groups:20]                8.953265e-01  1.057923e+00
468.3021 1.0043110
## b[(Intercept) Groups:_NEW_Groups]       1.010351e+00  1.658312e+00
2622.8348 1.0000523
## sigma                                   2.020067e+00  2.034173e+00
2960.8690 1.0002890
## Sigma[Groups:(Intercept),(Intercept)]   9.525495e-01  1.265483e+00
547.5314 1.0039773
## mean_PPD                                5.243673e-01  5.519714e-01
3837.4553 1.0003855
## log-posterior                          -9.519962e+03 -9.517463e+03
836.8809 1.0015032

library(R2jags)

# Data
X<- model.matrix(~ x1 + x2, data = normr)

K<- ncol(X)

re<- as.numeric(normr$Groups)

Nre<- length(unique(normr$Groups))

model.data<- list(
```

```
  Y = normr$y, # response
  X = X, # covariates
  N = nrow(normr), # rows in model
  re = re, # random effect
  b0 = rep(0,K), # parameter priors with initial 0 value
  B0 = diag(0.0001, K), # priors for V-C matrix
  a0 = rep(0,Nre), # priors for scale parameters
  A0 = diag(Nre)) # hyperpriors for scale parameters

# Fit
cat("
model {
# Diffuse normal priors for regression parameters
beta ~ dmnorm(b0[], B0[,])
# Priors for random intercept groups
a ~ dmnorm(a0, tau.plot * A0[,])
# Priors for the two sigmas and taus
tau.plot <- 1 / (sigma.plot * sigma.plot)
tau.eps <- 1 / (sigma.eps * sigma.eps)
sigma.plot ~ dunif(0.001, 10)
sigma.eps ~ dunif(0.001, 10)
# Likelihood
for (i in 1:N) {
Y[i] ~ dnorm(mu[i], tau.eps)
mu[i] <- eta[i]
eta[i] <- inprod(beta[], X[i,]) + a[re[i]]
}
}
",file = "lmm.txt")

inits<- function () {
  list(beta = rnorm(K, 0, 0.01),
       a = rnorm(Nre, 0, 1),
       sigma.eps = runif(1, 0.001, 10),
       sigma.plot = runif(1, 0.001, 10)
)}

params<- c("beta","a", "sigma.plot", "sigma.eps")

NORM0<- jags(data = model.data,
             inits = inits,
             parameters = params,
             model.file = "lmm.txt",
             n.thin = 10,
             n.chains = 3,
             n.burnin = 6000,
             n.iter = 10000)

## Compiling model graph
##    Resolving undeclared variables
```

```
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 4500
##    Unobserved stochastic nodes: 4
##    Total graph size: 36469
##
## Initializing model
##
##   |                                                  |
|   0%  |                                                  |++
|   3%  |                                                  |+++
|   7%  |                                                  |+++++
|  10%  |                                                  |+++++++
|  13%  |                                                  |++++++++
|  17%  |
|++++++++++                                        |  20%  |
|++++++++++++                                      |  23%  |
|++++++++++++++                                    |  27%  |
|+++++++++++++++++                                 |  30%  |
|++++++++++++++++++                                |  33%  |
|++++++++++++++++++++                              |  37%  |
|++++++++++++++++++++++                            |  40%  |
|++++++++++++++++++++++++                          |  43%  |
|++++++++++++++++++++++++++                        |  47%  |
|++++++++++++++++++++++++++++                      |  50%  |
|++++++++++++++++++++++++++++++                    |  53%  |
|++++++++++++++++++++++++++++++++                  |  57%  |
|++++++++++++++++++++++++++++++++++                |  60%  |
|++++++++++++++++++++++++++++++++++++              |  63%  |
|++++++++++++++++++++++++++++++++++++++            |  67%  |
|++++++++++++++++++++++++++++++++++++++++          |  70%  |
|++++++++++++++++++++++++++++++++++++++++++        |  73%  |
|++++++++++++++++++++++++++++++++++++++++++++      |  77%  |
|++++++++++++++++++++++++++++++++++++++++++++++    |  80%  |
|++++++++++++++++++++++++++++++++++++++++++++++++  |  83%  |
|++++++++++++++++++++++++++++++++++++++++++++++++++|  87%  |
|++++++++++++++++++++++++++++++++++++++++++++++++++|  90%  |
|++++++++++++++++++++++++++++++++++++++++++++++++++|  93%  |
|++++++++++++++++++++++++++++++++++++++++++++++++++|  97%  |
|++++++++++++++++++++++++++++++++++++++++++++++++++| 100%
##   |                                                  |
|   0%  |                                                  |**
|   5%  |                                                  |*****
|  10%  |                                                  |********
|  15%  |
|**********                                        |  20%  |
|***********                                       |  25%  |
|***************                                   |  30%  |
|*****************                                 |  35%  |
|*******************                               |  40%  |
```

```
|**********                                              |  45%  |
|**********                                              |  50%  |
|**********                                              |  55%  |
|**********                                              |  60%  |
|**********                                              |  65%  |
|**********                                              |  70%  |
|**********                                              |  75%  |
|**********                                              |  80%  |
|**********                                              |  85%  |
|**********                                              |  90%  |
|**********                                              |  95%  |
|**********                                              | 100%
```

```
# Output
print(NORM0, intervals=c(0.025, 0.975), digits=3)
```

```
## Inference for Bugs model at "lmm.txt", fit using jags,
##  3 chains, each with 10000 iterations (first 6000 discarded), n.thin = 10
##  n.sims = 1200 iterations saved
##              mu.vect sd.vect      2.5%     97.5%  Rhat n.eff
## a[1]           0.841   0.225     0.396     1.278 1.001  1200
## a[2]           0.328   0.224    -0.114     0.748 1.003   660
## a[3]           0.026   0.225    -0.423     0.468 1.003   750
## a[4]           0.392   0.224    -0.060     0.822 1.001  1200
## a[5]          -0.221   0.225    -0.681     0.211 1.000  1200
## a[6]          -1.119   0.222    -1.557    -0.704 1.000  1200
## a[7]          -0.958   0.225    -1.386    -0.488 1.003   650
## a[8]          -0.041   0.224    -0.471     0.400 1.001  1200
## a[9]           0.436   0.222    -0.013     0.870 1.002  1100
## a[10]          0.342   0.227    -0.132     0.779 1.002  1000
## a[11]         -0.563   0.225    -1.006    -0.092 1.002   770
## a[12]         -0.021   0.223    -0.445     0.404 1.000  1200
## a[13]         -1.009   0.227    -1.485    -0.569 1.000  1200
## a[14]          1.221   0.227     0.772     1.685 1.002   730
## a[15]          1.264   0.228     0.818     1.701 1.001  1200
## a[16]          0.053   0.217    -0.402     0.471 1.002   930
## a[17]         -0.415   0.226    -0.859     0.022 1.003   580
## a[18]         -0.017   0.218    -0.459     0.427 1.002  1100
## a[19]         -1.304   0.227    -1.767    -0.885 1.000  1200
## a[20]          0.601   0.223     0.159     1.016 1.000  1200
## beta[1]        0.723   0.198     0.347     1.138 1.001  1200
## beta[2]        0.203   0.103     0.001     0.390 1.000  1200
## beta[3]       -0.694   0.100    -0.889    -0.490 1.000  1200
## sigma.eps      1.993   0.022     1.949     2.037 1.002   860
## sigma.plot     0.793   0.143     0.570     1.122 1.001  1100
## deviance   18978.911   7.089 18967.168 18995.011 1.000  1200
##
## 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 = 25.2 and DIC = 19004.1
## DIC is an estimate of expected predictive error (lower deviance is
better).
```