MCMC for Bayesian Inference

Our Goal

- We want to generate samples from the posterior distribution of a parameter θ (or possibly a vector of parameters) given the observed data.

Miscellanea

MCMC stands for "Markov chain Monte Carlo"

- Monte Carlo: we will use randomly generated numbers to perform a task (estimation of quantities involving the posterior distribution)
- Markov chain: Our numbers will be randomly generated from a Markov chain.

Notation:

- Denote the pdf of this posterior by $f(\theta|x_1, \ldots, x_n)$

The Metropolis Algorithm

1. Pick an initial value, θ_1

2. For $i = 2, 3, \ldots, m$:

   (a) Generate a **proposal** θ_i^* for θ_i.

   - The proposal is drawn at random from a symmetric distribution that may depend on θ_{i-1}
   - Example:

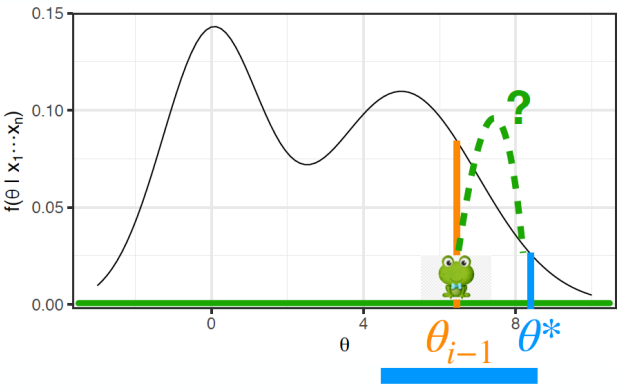
$$\theta^*|\theta_{i-1} \sim \text{Unif}(\theta_{i-1} - 2, \theta_{i-1} + 2)$$

   (b) Calculate the **acceptance probability**

$$r = \begin{cases} \frac{f(\theta^*|x_1,\ldots,x_n)}{f(\theta_{i-1}|x_1,\ldots,x_n)} \text{ if } f(\theta^*|x_1, \ldots, x_n) \leq f(\theta_{i-1}|x_1, \ldots, x_n) \\ 1 \text{ if } f(\theta^*|x_1, \ldots, x_n) > f(\theta_{i-1}|x_1, \ldots, x_n) \end{cases}$$

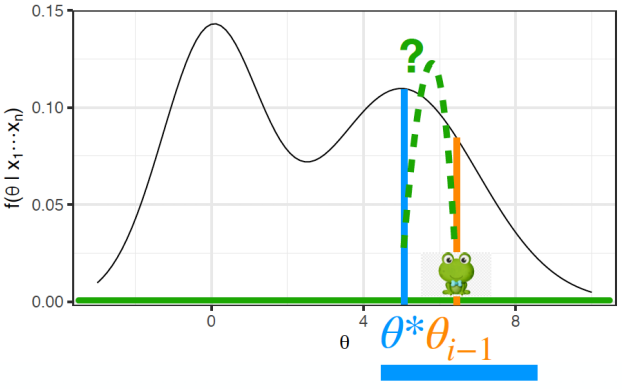
   (c) Set $\theta_i = \begin{cases} \theta^* \text{ with probability } r \\ \theta_{i-1} \text{ with probability } 1 - r \end{cases}$

Case 1: Proposal Accepted with Probability r < 1



Proposal is drawn uniformly in this interval (in this example)

Case 2: Proposal Accepted with Probability 1



Proposal is drawn uniformly in this interval (in this example)

Definition: A Markov chain is a sequence of random variables $X_1, X_2, \ldots$ with the property that for any set A,

$P(X_i \in A | X_1 = x_1, X_2 = x_2, \ldots, X_{i-1} = x_{i-1}) = P(X_i \in A | X_{i-1} = x_{i-1})$

- Intuition: the distribution of X_i depends only on X_{i-1}

Example: The sequence of random variables $\Theta_1, \Theta_2, \ldots$ from the Metropolis algorithm form a Markov chain

Remarkable fact 1: If some conditions are satisfied then the distribution of X_i converges to a given **stationary distribution** as $i \to \infty$.

- The stationary distribution depends on the transition distribution for moving from X_{i-1} to X_i.
- The stationary distribution does not depend on your choice of X_1!

For proofs of things like this, see Math 339SP: Stochastic Processes.

Remarkable fact 2: For the Metropolis algorithm above, the stationary distribution is the posterior distribution for Θ.

In practice, the sampler can take a while to converge to the stationary distribution. We typically throw away the first several hundred samples as *burn-in*.

To check for convergence, we can run multiple chains that started from different random starting points and compare the results; if the samples from all four chains look similar, that's evidence that they have converged.

Example: Cosmological Microwave Background (CMB)

This example is taken from Marin and Robert (2007). Here's a quote from them describing the figure below, also from them:

> 'Figure 2.2 is an image (in the spectral domain) of the "cosmological microwave background" (CMB) in a region of the sky: More specifically, this picture represents the electromagnetic radiation from photons dating back to the early ages of the universe, a radiation often called "fossil light," that dates back to a few hundred thousand years after the Big Bang (Chown, 1996). The grey levels are given by the differences in apparent temperature from the mean temperature and as stored in `cmb`.
>
> For astrophysical (or rather cosmological) reasons too involved to be detailed here, the repartition of the sectrum is quite isotropic (that is, independent of direction) and normal. In fact, if we treat each temperature difference in Figure 2.2 as an independent realization, the histogram of these differences ... provides a rather accurate representation of the distribution of these temperatures...'

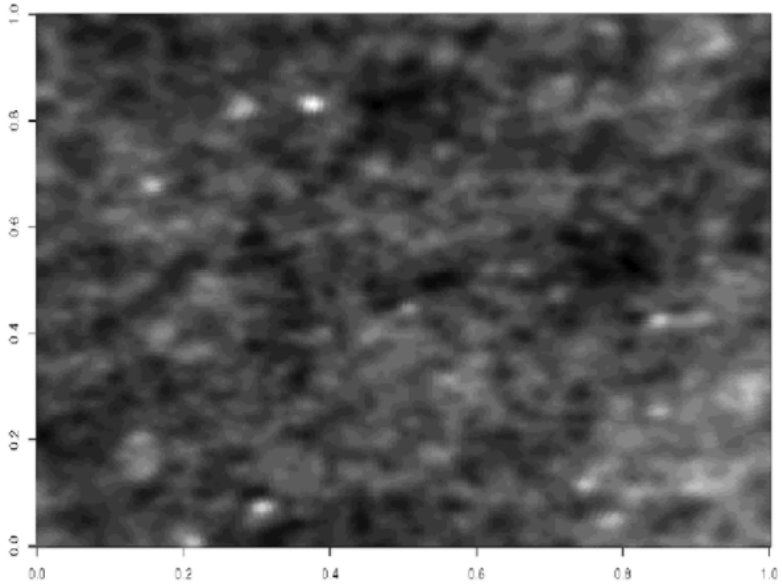


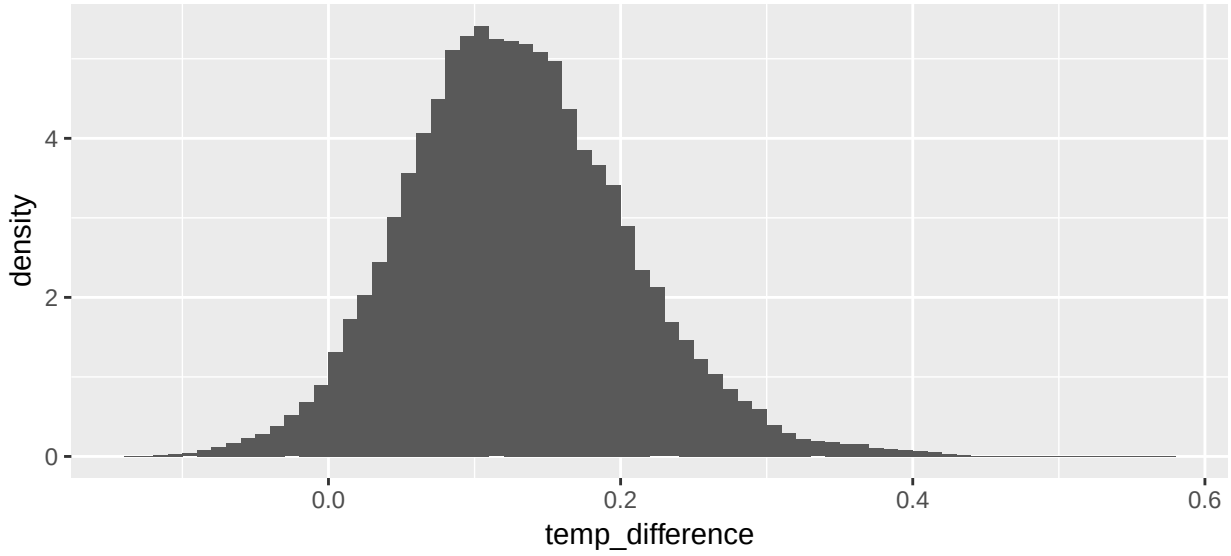
Fig. 2.2. Dataset CMBdata: Spectral image of the cosmological microwave background (CMB) of the universe. (The darker the pixel, the higher the temperature difference from the mean temperature.)

The code below reads in the data and makes an initial plot:

```
library(tidyverse)

cmb <- read_csv("http://www.evanlray.com/data/bayesian_core/CMBdata.txt",
    col_names = FALSE)
names(cmb) <- "temp_difference"

ggplot(data = cmb, mapping = aes(x = temp_difference)) +
  geom_histogram(center = 0.005, binwidth = 0.01, mapping = aes(y = ..density..))
```



Model

It appears that a normal model would be reasonable for these data. To be formal, let $X_1, \ldots, X_n$ denote the $n = 640000$ temperature differences. We model these as independent, with each

$X_i \sim \text{Normal}(\mu, \sigma^2)$

This model has two parameters: μ and σ^2. We will use the following prior distributions for these parameters:

An improper, non-informative prior for μ:

$f(\mu) = 1$

A Gamma$(2, 1)$ prior for σ^2:

$f(\sigma^2 | \alpha = 2, \beta = 1) = \frac{\beta^\alpha}{\Gamma(\alpha)} (\sigma^2)^{\alpha - 1} e^{-\beta x}$

The probability density function of the posterior distribution for μ and σ^2 is therefore equal to a constant c times the prior pdfs times the data model pdf:

$$\begin{aligned} f(\mu, \sigma^2 | \alpha = 2, \beta = 1, x_1, \ldots, x_n) &= c \cdot f(\mu) \cdot f(\sigma^2 | \alpha = 2, \beta = 1) \cdot f(x_1, \ldots, x_n | \mu, \sigma^2) \\ &= c \cdot 1 \cdot \frac{\beta^\alpha}{\Gamma(\alpha)} (\sigma^2)^{\alpha - 1} e^{-\beta x} \cdot \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[\frac{-1}{2\sigma^2}(x_i - \mu)^2\right] \end{aligned}$$

The posterior distribution is involved. Rather than trying to understand it analytically, let's take a sample from the joint posterior distribution of μ and σ^2, and use that to examine the posterior distribution.

Samples from the posterior distribution

```r
# how many samples to draw from the posterior
sample_size <- 10000

# How many to throw away?  (Assuming convergence after this)
burn_in <- 100

# allocate data frame to store results
theta_posterior_sample <- data.frame(
  mu = rep(NA, sample_size + burn_in),
  log_sigma_sq = rep(NA, sample_size + burn_in),
  sigma_sq = rep(NA, sample_size + burn_in)
)

# set initial values/first sample/Markov chain starting point
theta_posterior_sample$mu[1] <- mean(cmb$temp_difference)
theta_posterior_sample$log_sigma_sq[1] <- log(var(cmb$temp_difference))
theta_posterior_sample$sigma_sq[1] <- var(cmb$temp_difference)

# Sequentially, obtain the remaining samples
for(i in seq(from = 2, to = sample_size + burn_in)) {
    # Generate a proposal for the next value of theta, from a
    # Uniform(previous_theta - 0.1, previous_theta + 0.1) distribution
    previous_mu <- theta_posterior_sample$mu[i-1]
    previous_log_sigma_sq <- theta_posterior_sample$log_sigma_sq[i-1]
    mu_proposal <- runif(1, previous_mu - .001, previous_mu + .001)
    log_sigma_sq_proposal <- runif(1, previous_log_sigma_sq - .001, previous_log_sigma_sq + .001)

    # calculate probability of accepting the proposal
    log_r_num <- dgamma(exp(log_sigma_sq_proposal), shape = 2, rate = 1, log = TRUE) +
        sum(dnorm(cmb$temp_difference,
                  mean = mu_proposal,
                  sd = sqrt(exp(log_sigma_sq_proposal)),
                  log = TRUE))
    log_r_denom <- dgamma(exp(previous_log_sigma_sq), shape = 2, rate = 1, log = TRUE) +
        sum(dnorm(cmb$temp_difference,
                  mean = previous_mu,
                  sd = sqrt(exp(previous_log_sigma_sq)),
                  log = TRUE))

    # calculate acceptance probability
    if(log_r_num > log_r_denom) {
      r <- 1
    } else {
      r <- exp(log_r_num - log_r_denom)
    }

    # accept the proposal or not, with the appropriate probability
    if(rbinom(1, 1, r) == 1) {
        theta_posterior_sample$mu[i] <- mu_proposal
        theta_posterior_sample$log_sigma_sq[i] <- log_sigma_sq_proposal
        theta_posterior_sample$sigma_sq[i] <- exp(log_sigma_sq_proposal)
    } else {
        theta_posterior_sample$mu[i] <- previous_mu
        theta_posterior_sample$log_sigma_sq[i] <- previous_log_sigma_sq
        theta_posterior_sample$sigma_sq[i] <- exp(previous_log_sigma_sq)
    }
```

```
}

# discard burn-in
theta_posterior_sample <- theta_posterior_sample[-seq_len(burn_in), ]
```

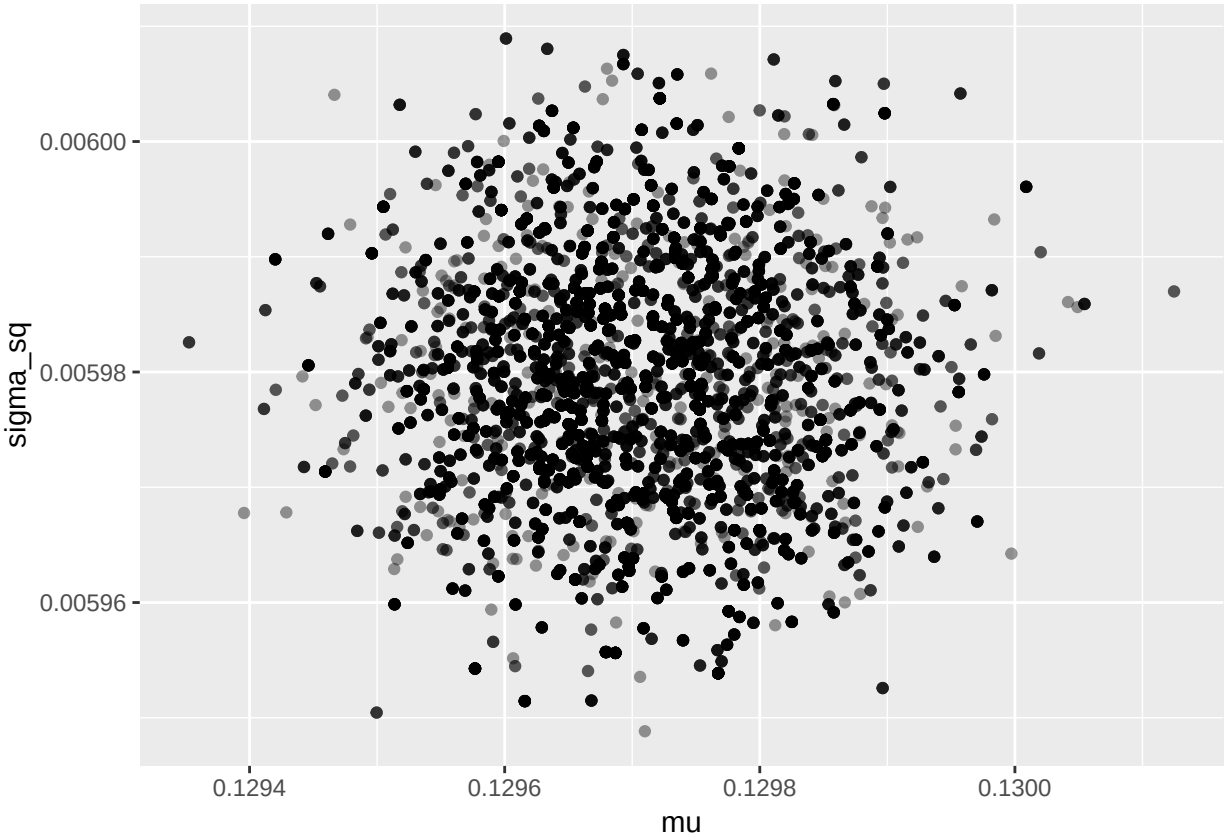
```
head(theta_posterior_sample)
```

```
##            mu log_sigma_sq    sigma_sq
## 101 0.1296762    -5.116966 0.005994183
## 102 0.1296762    -5.116966 0.005994183
## 103 0.1296762    -5.116966 0.005994183
## 104 0.1296762    -5.116966 0.005994183
## 105 0.1296762    -5.116966 0.005994183
## 106 0.1296762    -5.116966 0.005994183
```

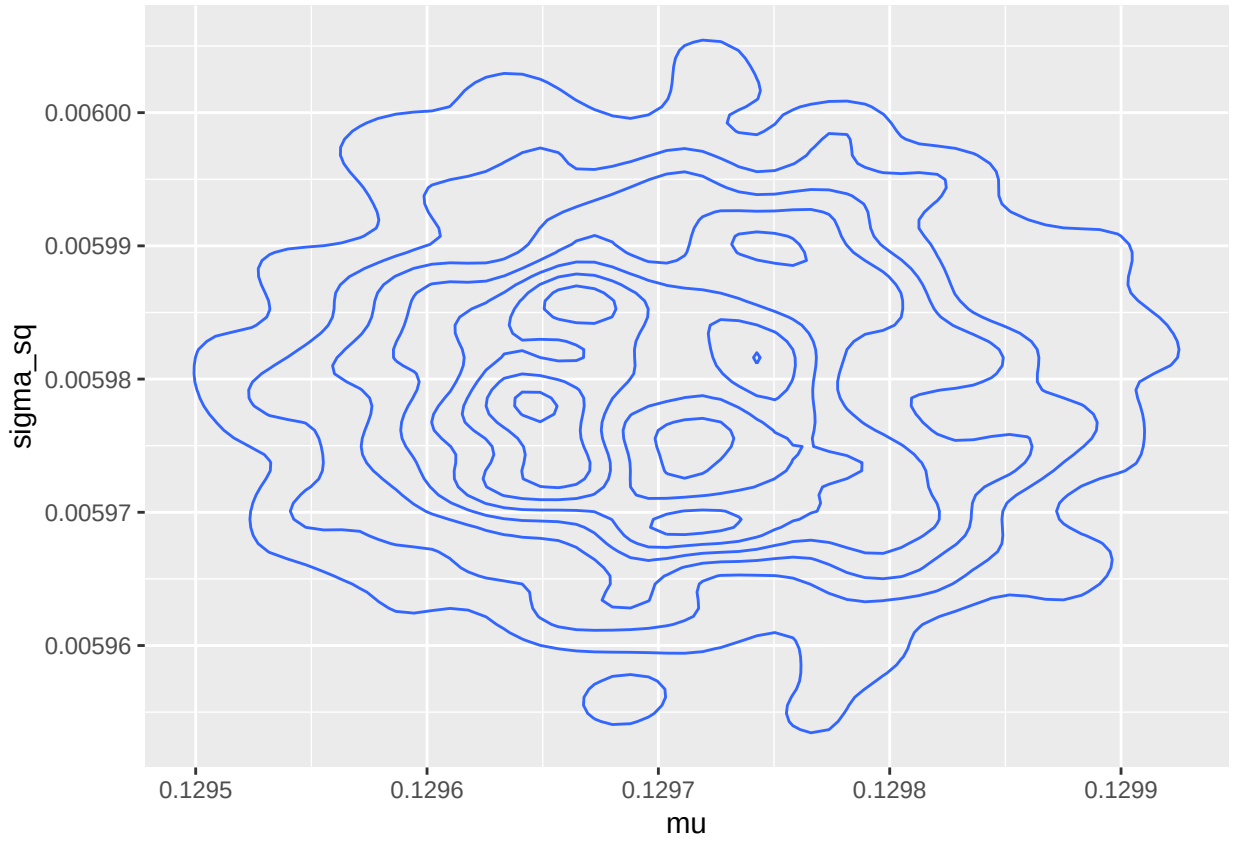
Plots to represent the approximate joint posterior distribution of μ, σ^2

Each point in the plot is a sample from the joint posterior of $\mu, \sigma^2 | x_1, \ldots, x_n$.

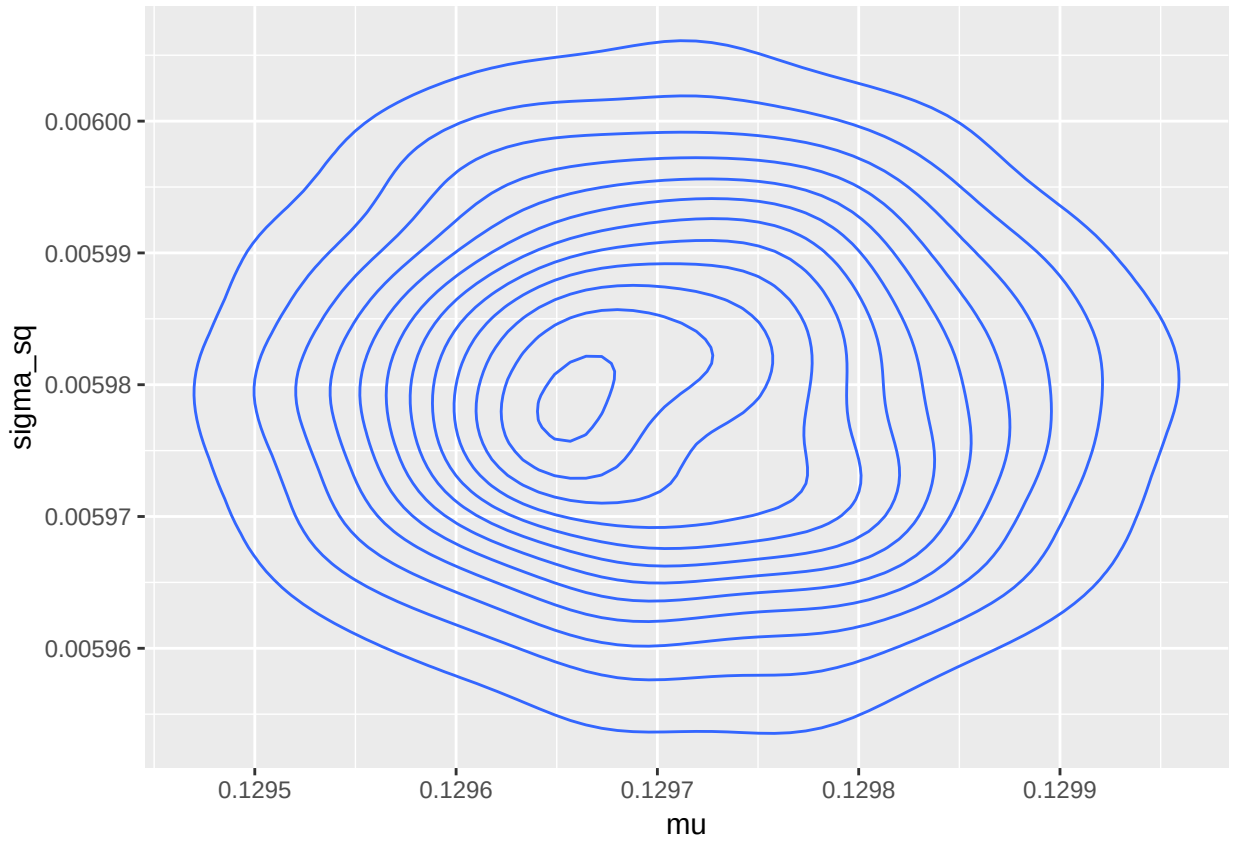
```
ggplot(data = theta_posterior_sample, mapping = aes(x = mu, y = sigma_sq)) +
  geom_point(alpha = 0.4)
```




```
ggplot(data = theta_posterior_sample, mapping = aes(x = mu, y = sigma_sq)) +
  geom_density2d()
```



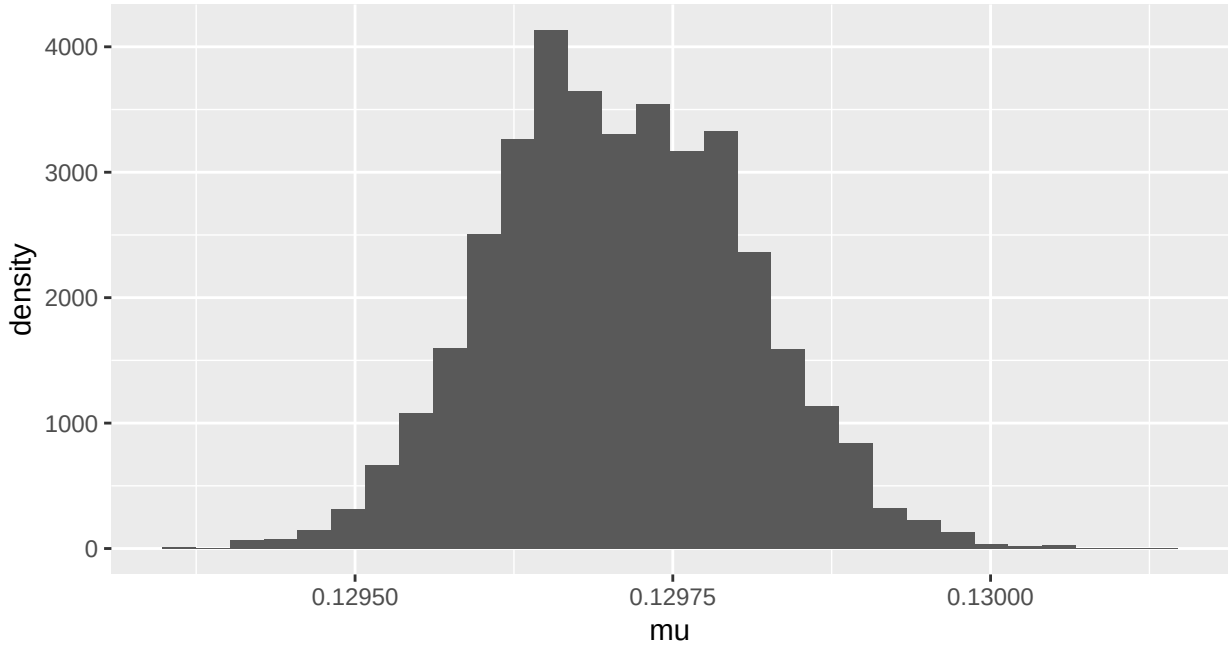

```
ggplot(data = theta_posterior_sample, mapping = aes(x = mu, y = sigma_sq)) +
  geom_density2d(h = c(0.00015, 0.000015))
```



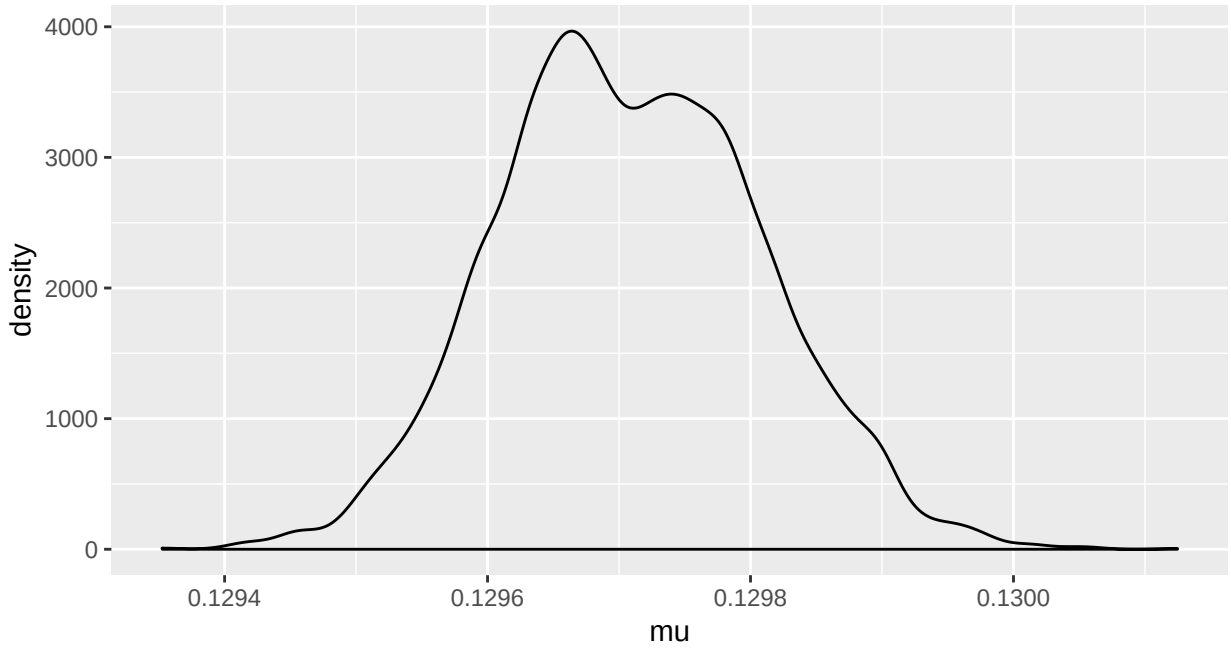
Plots to represent the approximate marginal posterior distribution of μ

```
ggplot() +
  geom_histogram(data = theta_posterior_sample, mapping = aes(x = mu, y = ..density..))
```

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



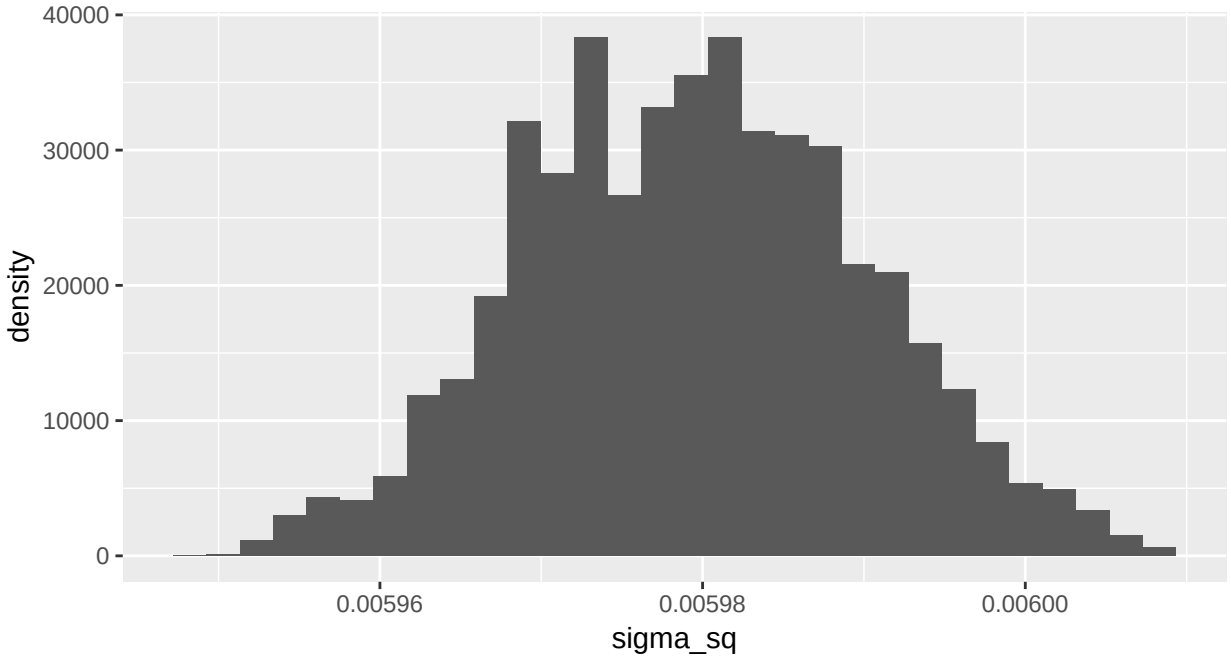

```
ggplot() +
  geom_density(data = theta_posterior_sample, mapping = aes(x = mu))
```



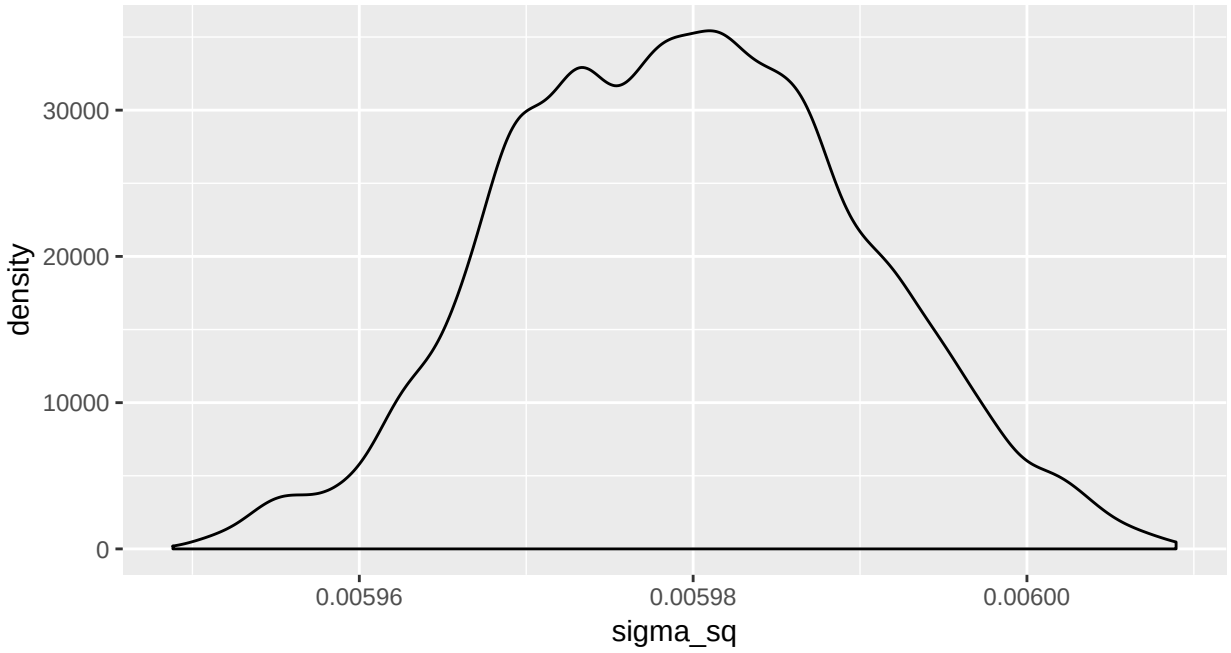
Plots to represent the approximate marginal posterior distribution of σ^2

```
ggplot() +
  geom_histogram(data = theta_posterior_sample, mapping = aes(x = sigma_sq, y = ..density..))
```

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



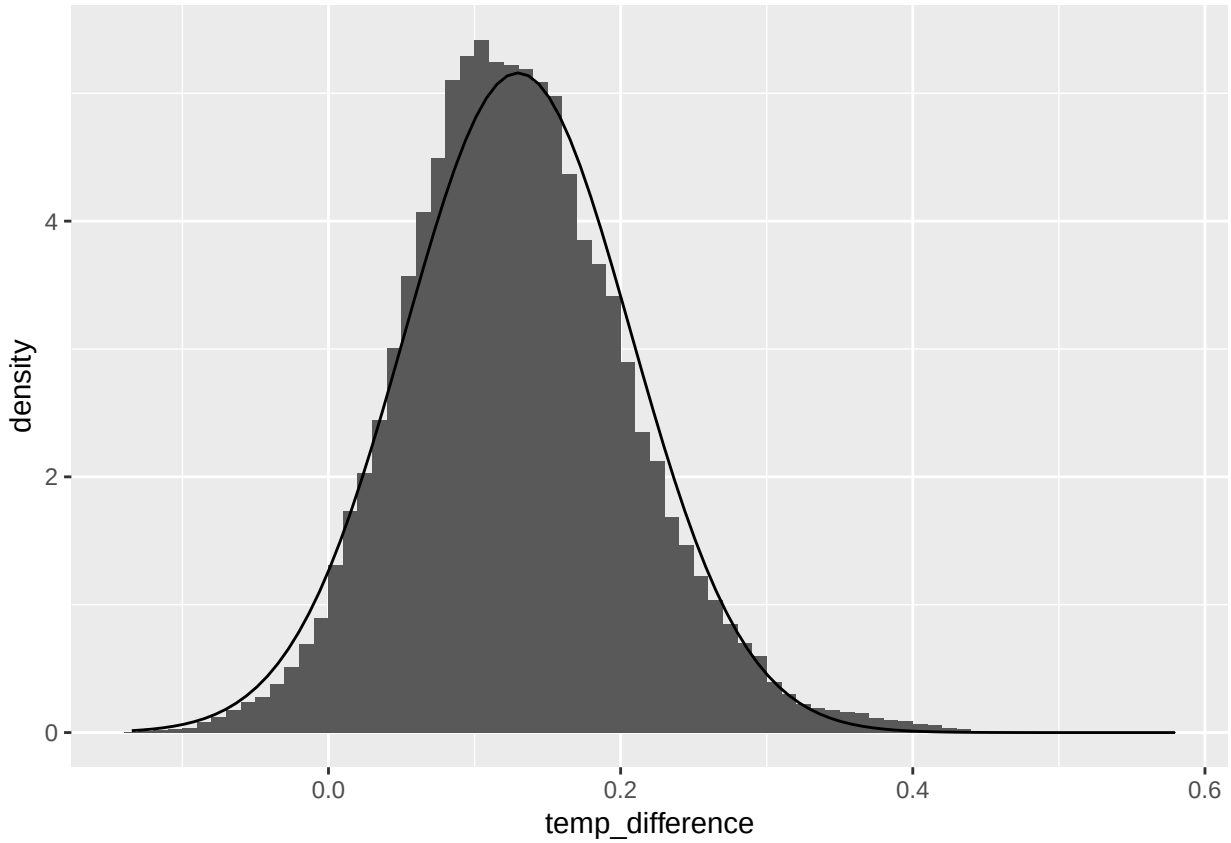

```
ggplot() +
  geom_density(data = theta_posterior_sample, mapping = aes(x = sigma_sq))
```



How does the model fit?

Compare a histogram of the data to the normal density with parameters set equal to the estimated posterior mean for μ and for σ^2.

```
ggplot(data = cmb, mapping = aes(x = temp_difference)) +
  geom_histogram(center = 0.005, binwidth = 0.01, mapping = aes(y = ..density..)) +
  stat_function(fun = dnorm,
    args = list(mean = mean(theta_posterior_sample$mu), sd = sqrt(mean(theta_posterior_sample$sigma_sq))))
```



Fit via Stan

normal.stan

Here is the content of the file normal.stan:

```
data {
  int<lower=0> n; // number of observations
  real x[n]; // data: an array of length n where each entry is a real number
}

parameters {
  real mu;
  real<lower=0> sigma;
}

model {
  mu ~ normal(0, 1000); // prior for mu: normal with a very large variance; non-informative
  sigma ~ gamma(2, 1); // prior for sigma: gamma with shape = 1 and rate = 0.01; non-informative
  x ~ normal(mu, sigma); // data model: each element of x follows a normal(mu, sigma) distribution
}
```

The `data` block in the Stan file describes fixed, known quantities that will be passed in to Stan. In this case, we have said that we will tell Stan what our sample size is (`n`) and give it a vector of length `n` with observed data values `x`.

The `parameters` block defines parameters to estimate; in this case, the mean `mu` and standard deviation `sigma` of the normal distribution.

The `model` block defines our prior distributions and data model.

Stan takes these ingredients and creates a program in C++ that will perform Bayesian estimation for this model using a MCMC approach called Hamiltonian Monte Carlo.

R code to interface with Stan

Call Stan to do the estimation

- For MCMC, just one command (`stan`) both compiles the model and performs the sampling.
- A substantial part of the run time when calling Stan comes from creating and compiling the C++ program to do estimation. The command `rstan_options(auto_write = TRUE)` ensures that this is done only the first time you call Stan, unless you've made changes to the stan file.
- The `stan` function does estimation. Here we have used 4 arguments:
  - `file`: the stan file with the model definition, created above.
  - `data`: a named list with one entry for each variable declared in the `data` block of the stan file.
  - `iter`: how many iterations to perform (how many samples to draw from the posterior in each MCMC chain).
  - `chains`: how many MCMC chains to run; here, 4 separate chains are run.

```
library(rstan)
```

```
## Warning: package 'StanHeaders' was built under R version 3.5.2
```

```
rstan_options(auto_write = TRUE)

fit <- stan(
  file = "normal.stan",
  data = list(n = nrow(cmb), x = cmb$temp_difference),
  iter = 1000,
  chains = 4)
```

```
##
## SAMPLING FOR MODEL 'normal' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.002114 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 21.14 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 1: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 1: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 1: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 1: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 1: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 1: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 1: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 1: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 13.4016 seconds (Warm-up)
## Chain 1:                15.1235 seconds (Sampling)
## Chain 1:                28.5251 seconds (Total)
## Chain 1:
```

```
##
## SAMPLING FOR MODEL 'normal' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0.002252 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 22.52 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 2: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 2: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 2: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 2: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 2: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 2: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 2: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 2: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 2: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 2: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 2: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 7.18012 seconds (Warm-up)
## Chain 2:                15.2367 seconds (Sampling)
## Chain 2:                22.4168 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'normal' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0.00231 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 23.1 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 3: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 3: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 3: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 3: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 3: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 3: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 3: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 3: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 3: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 3: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 3: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 15.9207 seconds (Warm-up)
## Chain 3:                15.3833 seconds (Sampling)
## Chain 3:                31.304 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'normal' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0.002014 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 20.14 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:   1 / 1000 [  0%]  (Warmup)
```

```
## Chain 4: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 4: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 4: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 4: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 4: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 4: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 4: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 4: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 4: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 4: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 4: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 19.4481 seconds (Warm-up)
## Chain 4:                13.0007 seconds (Sampling)
## Chain 4:                32.4487 seconds (Total)
## Chain 4:
```

View the results

The `rstan` package comes with some pretty useful default functions to display and summarize the samples from the posterior distribution:

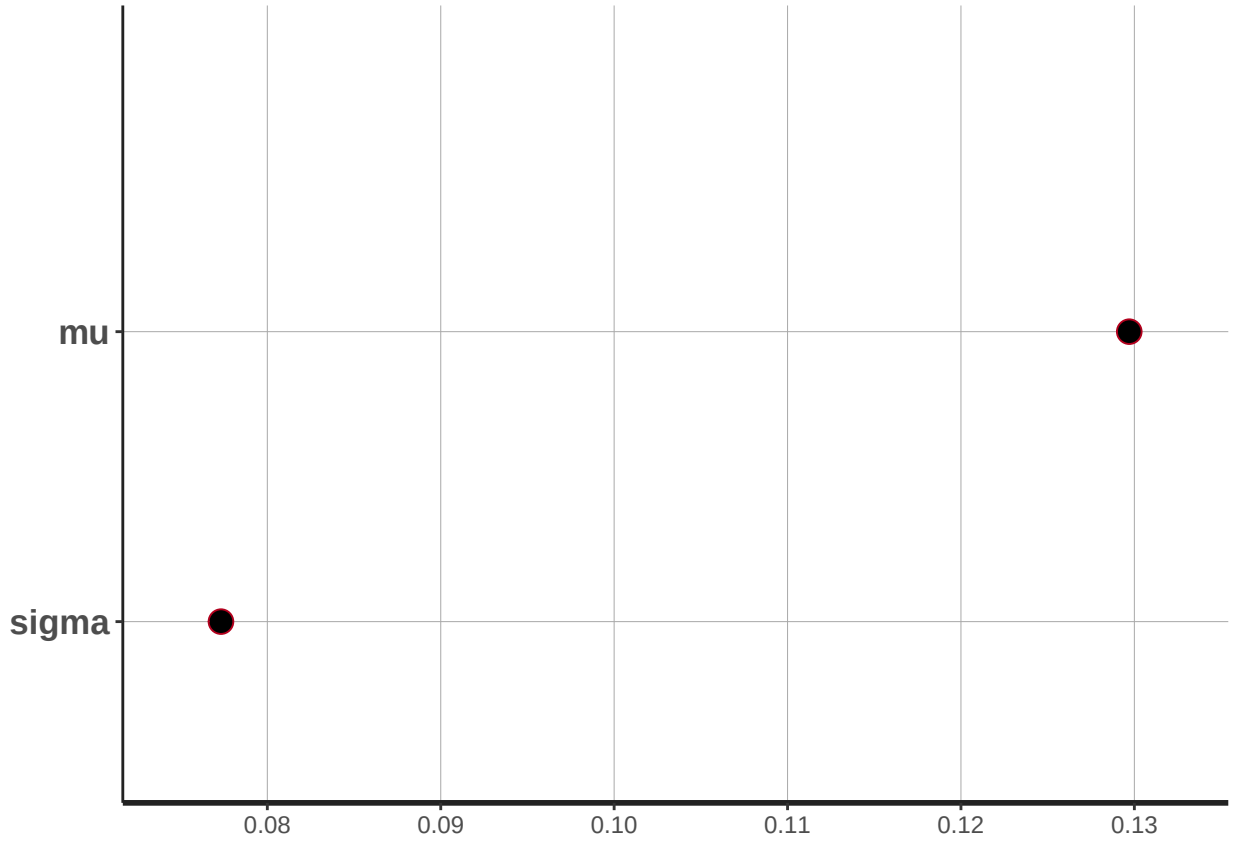
```
print(fit)
```

```
## Inference for Stan model: normal.
## 4 chains, each with iter=1000; warmup=500; thin=1;
## post-warmup draws per chain=500, total post-warmup draws=2000.
##
##             mean se_mean   sd       2.5%        25%        50%        75%
## mu          0.13    0.00 0.00       0.13       0.13       0.13       0.13
## sigma       0.08    0.00 0.00       0.08       0.08       0.08       0.08
## lp__  1318253.74    0.04 0.96 1318251.10 1318253.38 1318254.02 1318254.42
##            97.5% n_eff Rhat
## mu          0.13  1977 1.00
## sigma       0.08   296 1.02
## lp__  1318254.68   576 1.01
##
## Samples were drawn using NUTS(diag_e) at Thu Feb 28 21:48:54 2019.
## For each parameter, 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).
```

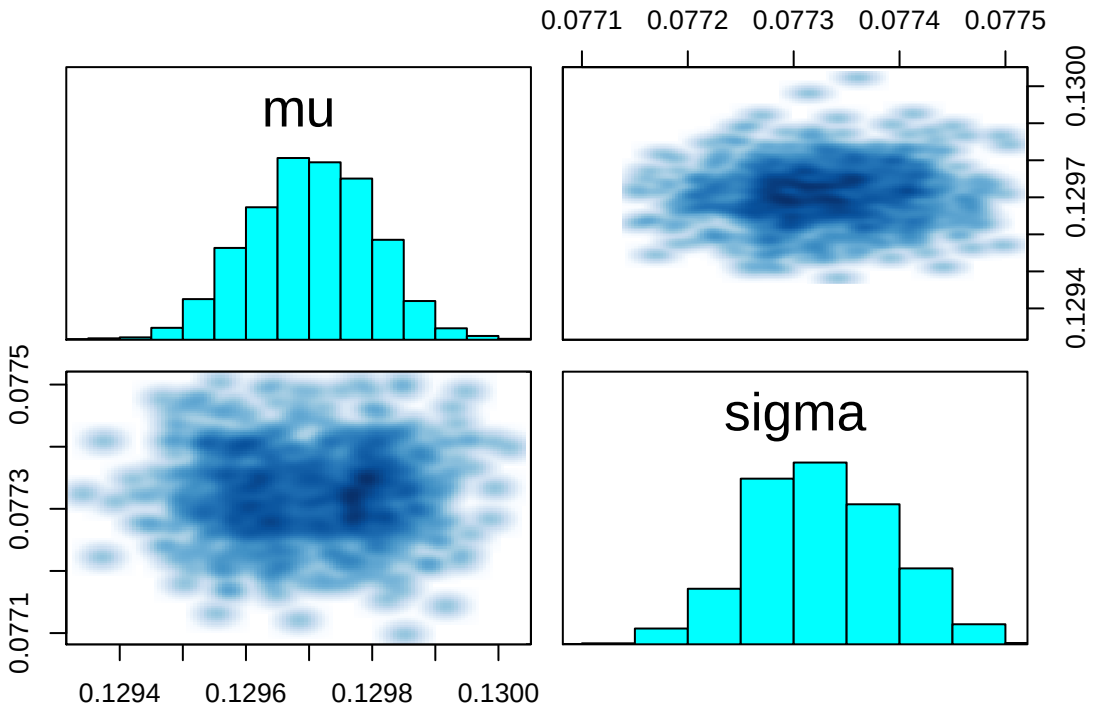
```
plot(fit)
```

```
## ci_level: 0.8 (80% intervals)
```

```
## outer_level: 0.95 (95% intervals)
```




```
pairs(fit, pars = c("mu", "sigma"))
```



We can also extract the parameter samples and compute summaries like posterior means and credible intervals by hand. Calling `as.data.frame` on our model fit object returns a data frame with the samples for each parameter defined in the stan model file.

```
param_samples <- as.data.frame(fit)

head(param_samples)
```

```
##          mu      sigma    lp__
```

```
## 1 0.1297261 0.07732078 1318255
## 2 0.1297270 0.07742988 1318253
## 3 0.1297446 0.07732818 1318255
## 4 0.1297443 0.07741279 1318254
## 5 0.1297420 0.07735710 1318254
## 6 0.1296313 0.07734655 1318254
```

```
dim(param_samples)
```

```
## [1] 2000    3
```

```
param_samples %>%
  summarize(
    mu_mean = mean(mu),
    mu_ci_lower = quantile(mu, probs = 0.025),
    mu_ci_upper = quantile(mu, probs = 0.975),
    sigma_mean = mean(sigma),
    sigma_ci_lower = quantile(sigma, probs = 0.025),
    sigma_ci_upper = quantile(sigma, probs = 0.975)
  )
```

```
##     mu_mean mu_ci_lower mu_ci_upper sigma_mean sigma_ci_lower
## 1 0.1297032    0.129516   0.1298862 0.07732648     0.07719778
##   sigma_ci_upper
## 1     0.07745808
```

```
ggplot(data = param_samples, mapping = aes(x = mu)) +
  geom_density()
```