MCMC for Bayesian Inference

Goal

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

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...'

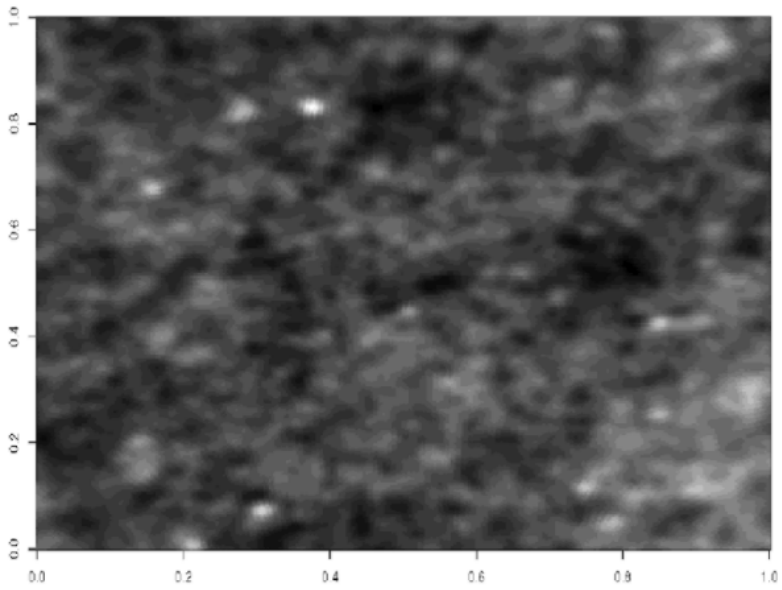


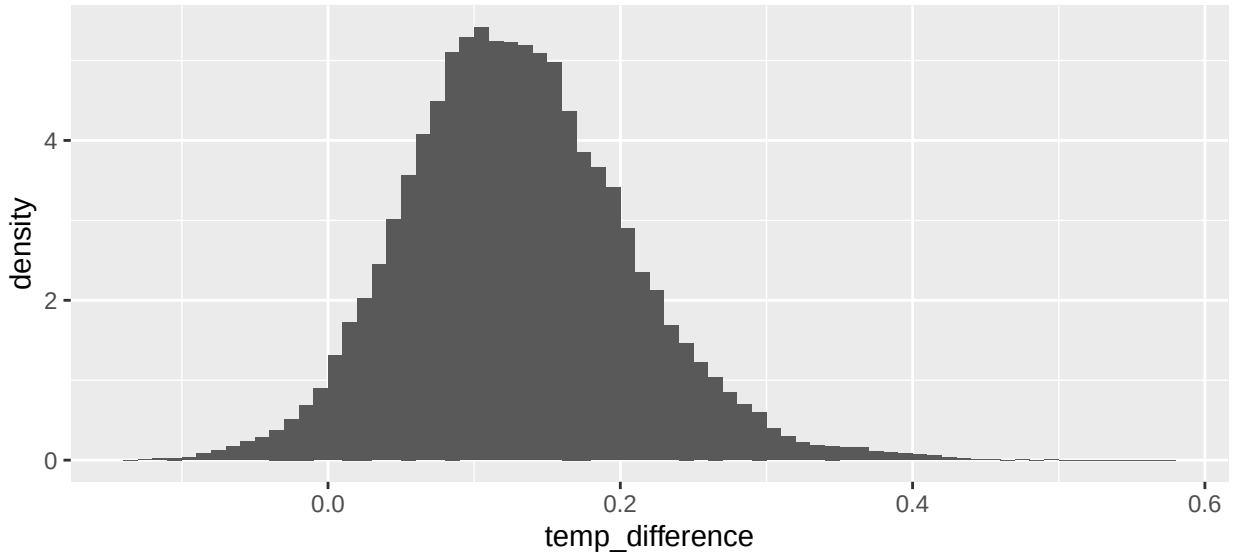
Fig. 2.2. Dataset CMBdata: Spectral image of the cosmological microwave background (CMB) of the universe. (The darker the pixel, the higher the temperature difference from the mean temperature.)

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

We'll try a normal model. 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:

A Normal$(0, 1000)$ prior for μ. This is a very high variance, so is not informative.

A Gamma$(2, 1)$ prior for σ^2:

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

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)
```

```
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 0.002222 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 22.22 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: 15.2302 seconds (Warm-up)
## Chain 1:                13.5728 seconds (Sampling)
## Chain 1:                28.803 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'normal' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0.002215 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 22.15 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: 14.4632 seconds (Warm-up)
## Chain 2:                15.0269 seconds (Sampling)
## Chain 2:                29.4901 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'normal' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0.002219 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 22.19 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: 14.536 seconds (Warm-up)
## Chain 3:                13.7104 seconds (Sampling)
## Chain 3:                28.2463 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'normal' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0.002214 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 22.14 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: 13.5766 seconds (Warm-up)
## Chain 4:                13.7854 seconds (Sampling)
## Chain 4:                27.362 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__  1318253.70    0.04 0.97 1318251.04 1318253.36 1318254.00 1318254.41
##            97.5% n_eff Rhat
## mu          0.13  2018 1.00
## sigma       0.08   326 1.01
## lp__  1318254.68   480 1.01
##
## Samples were drawn using NUTS(diag_e) at Mon Mar  2 00:43:58 2020.
## 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).
```

What's the deal with effective sample sizes?

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 0.1297553 0.07737122 1318254
## 2 0.1298299 0.07737078 1318254
## 3 0.1297188 0.07736988 1318254
## 4 0.1298187 0.07736406 1318254
## 5 0.1297696 0.07739936 1318254
## 6 0.1296375 0.07739600 1318254
```

```r
dim(param_samples)
```

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

```r
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 0.1297088   0.1295177   0.1299026 0.07732196     0.07719782
##   sigma_ci_upper
## 1     0.07745889
```

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