Chapter 16

Stan

16.1 Discoveries data revisited

The file `evaluation_discoveries.csv` contains data on the numbers of "great" inventions and scientific discoveries (X_t) in each year from 1860 to 1959 [1]. In this question you will develop a model to explain the variation in scientific inventions over time. The simplest model here is to assume that (a.) one discovery is independent of all others, and (b.) the rate of occurrence of discoveries is the same in all years (λ). Since the data is discrete, these assumptions suggest the use a Poisson likelihood,

$$X_t \sim Poisson(\lambda) \tag{16.1}$$

Problem 16.1.1. Open a text editor and create a file called "discoveries.stan" in your working directory. In the file create three parameter blocks:

```
data {

}
parameters {

}
model {

}
```

Problem 16.1.2. Fill in the data and parameter blocks for the above model.

The completed model can be written as:

```
data {
  int N; // number of observations
  int<lower=0> X[N]; // vector of discoveries
}
```

```
parameters {
  real<lower=0> lambda;
}

model {
  X ~ poisson(lambda); // likelihood
  lambda ~ lognormal(2, 1); // prior
}
```

Problem 16.1.3. Using a $log-\mathcal{N}(2,1)$ prior for λ code up the `model` block; making sure to save your file afterwards.

Problem 16.1.4. Open your statistical software (R, Python, Matlab, etc.) and load any packages necessary to use Stan. (Hint: in R this is done by using "library(rstan)'; in Python this is done using "import pystan".)

Problem 16.1.5. Load the data into your software then put it into a structure that can be passed to Stan. (Hint: in R create a list of the data; in Python create a dictionary where the 'key' for each variable is the desired variable name.)

```
aDF <- read.csv('evaluation_discoveries.csv')
X <- aDF[, 2]
N <- length(X)
dataList <- list(N=N, X=X)
```

Problem 16.1.6. Run your model using Stan, with 4 chains, each with a sample size of 1000, and a warm-up of 500 samples. Set seed=1 to allow for reproducibility of your results. Store your result in an object called "fit".

```
fit <- stan(file='discoveries.stan', data=dataList, iter=1000, chains=4, seed=1)
```

Problem 16.1.7. Diagnose whether your model has converged by printing "'fit".

Lambda and lp should both have a value of $\hat{R} \approx 1$.

Problem 16.1.8. For your sample what is the equivalent number of samples for an independent sampler?

This is just the value of "n_eff" which I get to be around 600-900.

Problem 16.1.9. Find the central posterior 80% credible interval for λ.

This can be done in R by the following:

```
print(fit, pars='lambda', probs = c(0.1, 0.9))
```

Alternatively, you can extract lambda from the fit object then find its quantiles:

```
lLambda <- extract(fit, 'lambda')[[1]]
c(quantile(lLambda, 0.1), quantile(lLambda, 0.9))
```

Or in Python by doing:

```
import numpy as np
lambda_samples = fit.extract('lambda')['lambda']
np.percentile(lambda_samples, (10, 90))
```

In any case I get an interval of approximately $2.9 \leq \lambda \leq 3.3$.

Problem 16.1.10. Draw a histogram of your posterior samples for λ.

To do this in R, first extract the parameter, then draw the plot.

```
library(ggplot2)

lLambda <- extract(fit, 'lambda')[[1]]
qplot(lLambda)
```

Or in python:

```
import numpy as np
import matplotlib.pyplot as plt
lambda_samples = fit.extract('lambda')['lambda']
np.percentile(lambda_samples, (10, 90))
plt.hist(lambda_samples)
plt.xlabel("lambda")
plt.ylabel("frequency")
```

Problem 16.1.11. Load the `evaluation_discoveries.csv` data and graph the data. What does this suggest about our model's assumptions?

Both a time series and histogram are useful here (see Figure 16.1). To me the left hand plot suggests that there is some temporal autocorrelation in the data (perhaps invalidating an assumption of independence, and/or identical distribution). The histogram would seem to support this claim, since the variance is fairly obviously greater than the mean. I also plot an autocorrelogram of the data which suggests that there is autocorrelation in the series.

Problem 16.1.12. Create a `generated quantities` block in your Stan file, and use it to sample from the posterior predictive distribution. Then carry out appropriate posterior predictive checks to evaluate your model. (Hint: use the function `poisson_rng` to generate independent samples from your lambda).

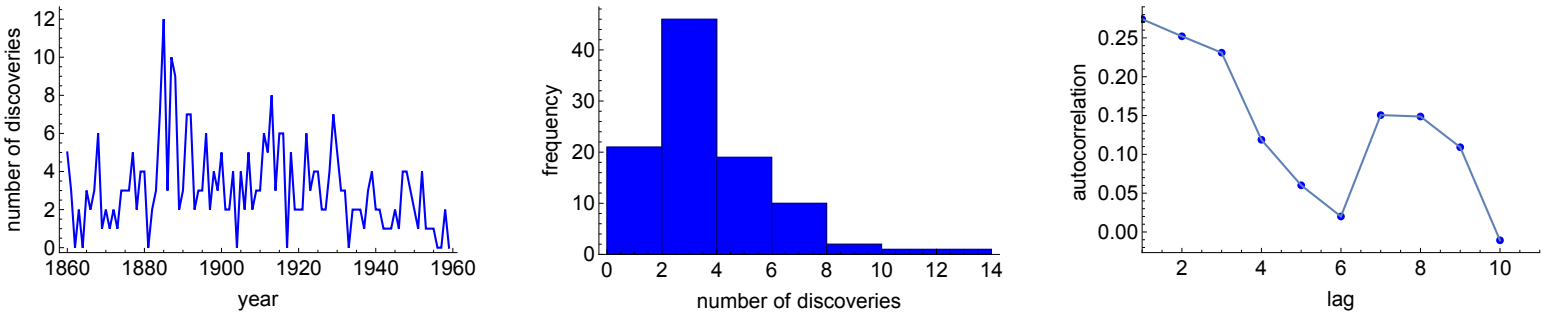

Figure 16.1: Characteristics of the "discoveries" data set.

```
generated quantities{
  int<lower=0> XSim[N];
  for (i in 1:N)
    XSim[i] <- poisson_rng(lambda);
}
```

One simple check is to compare the maximum of your posterior predictive simulations with that of the real data (which is 12 discoveries in 1885.) To do this in R do something like the following:

```
lXSim <- extract(fit, 'XSim')[[1]]
lMax <- apply(lXSim, 1, max)

library(ggplot2)
qplot(lMax)

sum(lMax >= 12) / length(lMax)
```

Or in Python by:

```
posterior_checks = fit.extract('XSim')['XSim']
posterior_checks_max = np.amax(posterior_checks, axis=1)
(posterior_checks_max >= 12).sum() / float(len(posterior_checks_max))
```

From the resultant histogram it is evident that our model would only very rarely produce 12 discoveries. I get 12+ discoveries in only about 1% of simulations.

Problem 16.1.13. A more robust sampling distribution is a negative binomial model:

$$X_i \sim NB(\mu,\ \kappa) \tag{16.2}$$

where μ is the mean number of discoveries per year, and $var(X) = \mu + \frac{\mu^2}{\kappa}$. Here κ measures the degree of over-dispersion of your model; specifically if $\kappa \uparrow$ then over-dispersion$\downarrow$.

Write a new Stan file called "discoveries_negbin.stan" that uses this new sampling model (Hint: use the Stan manual section on discrete distributions to search for the correct negative binomial

function name; be careful there are two different parameterisations of this function available in Stan!) Assume that we are using the following priors:

$$\mu \sim log - \mathcal{N}(2, 1) \tag{16.3}$$
$$\kappa \sim log - \mathcal{N}(2, 1) \tag{16.4}$$
$$\tag{16.5}$$

Draw 1000 samples across 4 chains for your new model. Has it converged to the posterior?

The full code should be something like:

```
data {
  int N;
  int<lower=0> X[N];
}

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

model {
  X ~ neg_binomial_2(mu, kappa);
  mu ~ lognormal(2, 1);
  kappa ~ lognormal(2, 1);
}
```

After 1000 samples across 4 chains I get a value of $\hat{R} \approx 1$, and an effective sample size around 1000.

Problem 16.1.14. Carry out posterior predictive checks on the new model. What do you conclude about the use of a negative binomial here versus the simpler Poisson?

The Stan code to generate the posterior predictive distribution samples is:

```
generated quantities{
  int<lower=0> XSim[N];
  for (i in 1:N)
    XSim[i] <- neg_binomial_2_rng(mu, kappa);
}
```

Now I obtain about 20% of posterior predictive samples that have a maximum value greater than or equal to 12 (that of the real data). This is much more reasonable than that from the Poisson model.

Problem 16.1.15. Find the central posterior 80% credible interval for the mean rate of discoveries μ from the negative binomial model. How does it compare with your results from the Poisson model? Why is this the case?

I get a similar answer to before, around: $2.8 \leq \lambda \leq 3.4$; it should be slightly wider because there is more sampling variability $\implies$ greater uncertainty over the parameter's value.

Problem 16.1.16. Calculate the autocorrelation in the residuals between the actual and simulated data series. What do these suggest about our current model?

If you look at the autocorrelation you see that there is a slight, yet persistent, positive autocorrelation (looking at the first lag). You can obtain these in R by the following:

```
lXSim <- extract(fit, 'XSim')[[1]]
mResid <- sweep(lXSim, 2, X)
lCorr <- sapply(seq(1, 200, 1),
                function(i) acf(mResid[i, ], lag.max=1)$acf[[2]])
qplot(lCorr)
```

Problem 16.1.17. Following on from the above suggest an alternative model formulation.

Clearly there is some persistence in the rate of discoveries over time. In other words, discoveries tend to clump together. This is clear from a simple time series plot of the data. One idea would be to allow an AR1 process for the mean of the process:

$$\mu(t) = \rho\mu(t-1) + (1-\rho)\bar{\mu} + \epsilon(t) \tag{16.6}$$

This allows for persistence in the rate over time and might remedy some of the issues. However, implementing it will be a bit tricky since $\mu(t)$ must always be positive. Perhaps the best thing to do here is to use a transformed parameter $exp(\mu)$ as the mean of the distribution. This will prevent it from being non-negative.

16.2 Hungover holiday regressions

The data in file `stan_hangover.csv` contains a series of Google Trends estimates of the search traffic volume for the term "hangover cure" in the UK between February 2012 to January 2016. The idea behind this problem is to determine how much more hungover are people in the "holiday season" period, defined here as the period between 10th December and 7th January, than the average for the rest of the year.

Problem 16.2.1. Graph the search volume over time, and try to observe the uplift in search volume around the holiday season.

```
plot(as.Date(hangover$date), hangover$volume, type='l', xlab='date',
     ylab='volume')
```

Shows obvious spikes during the holiday season period.

Problem 16.2.2. The variable "holiday" is a type of indicator variable that takes the value 1 if the given week is *all* holiday season, 0 if it contains none of it, and $0 < X < 1$ for a week that contains a fraction X of days that fall in the holiday season. Graph this variable over time so that you understand how it works.

Problem 16.2.3. A simple linear regression is proposed of the form,

$$V_t \sim \mathcal{N}(\beta_0 + \beta_1 h_t,\ \sigma) \tag{16.7}$$

where V_t is the search volume in week t and h_t is the holiday season indicator variable. Interpret β_0 and β_1 and explain how these can be used to estimate the increased percentage of hangovers in the holiday season.

β_0 is the average hangover search volume in weeks that aren't in the holiday season, and β_1 shows the uplift for a week that falls in the holiday season. The percentage increase is hence β_1/β_0.

Problem 16.2.4. Assuming $\beta_i \sim \mathcal{N}(0, 50)$ and $\sigma \sim$ half-$\mathcal{N}(0, 10)$ priors write a Stan model to estimate the percentage increase in hangoverness over the holiday period.

An example Stan program is shown below (not the most efficient since doesn't vectorise, but the below is simplest to understand),

```
data{
  int T;
  real V[T];
  real h[T];
}

parameters{
  real beta0;
  real beta1;
  real<lower=0> sigma;
}

model{
  for(t in 1:T)
    V[t] ~ normal(beta0 + beta1 *h[t], sigma);

  beta0 ~ normal(0, 50);
  beta1 ~ normal(0, 50);
  sigma ~ normal(0, 10);
```

```
}

generated quantities{
  real uplift;
  uplift = beta1 / beta0;
}
```

From this we estimate that with 50% probability, 77% $\leq$ uplift $\leq$ 86%!

16.3 Coding up a bespoke probability density

In the file `stan_survival.csv` there is data for a variable Y that we believe comes from a probability distribution $p(Y) = \frac{\sqrt[3]{b}}{\Gamma(\frac{4}{3})} exp(-bY^3)$ where $b > 0$ is a parameter of interest. In this question we are going to write a Stan program to estimate the parameter b even though this distribution is not amongst Stan's implemented distributions!

Problem 16.3.1. Explain what is meant by the following statement in Stan,

```
theta ~ beta(1, 1);
```

In particular, explain why this is equivalent to the following,

```
target += beta_lpf(theta | 1, 1);
```

where `target` is a Stan variable that stores the overall log-probability, and `+=` increments `target` by an amount corresponding to the RHS.

`~` statements in Stan do not mean sampling! They always mean increment the log probability by something, since HMC works in the (negative) log probability space. In this case it means increment the log probability by an amount corresponding to the probability density of a value "theta" from a beta(1,1) distribution. This is why it is equivalent to the second piece of code where we explicitly increment the log probability.

Note there is a subtle difference between the two which is that `~` statements drop all constant terms from the log probability update, whereas the target statements keep these. However for most purposes this is not important.

Problem 16.3.2. Work out by hand an expression for the log-probability of the density $p(Y) = \frac{\sqrt[3]{b}}{\Gamma(\frac{4}{3})} exp(-bY^3)$.

Just take the log of the expression,

$$log\ p = log\left(\frac{\sqrt[3]{b}}{\Gamma(4/3)}\right) - by^3 \tag{16.8}$$

Problem 16.3.3. Write a Stan function that for a given value of y and b calculates the log probability (ignoring any constant terms). Hint: Stan functions are declared as follows,

```
functions{
  real anExample(real a, real b){
    ...
    return(something);
  }
}
```

where in this example the function takes two reals as inputs and outputs something of type real.

A function to do this is shown below,

```
functions{
  real fCustomProb(real aY, real aB){
    real aConst;
    aConst = (aB ^ (1.0 / 3.0));
    return(log(aConst) - aB * (aY ^ 3));
  }
}
```

Problem 16.3.4. Use your previously created function to write a Stan program that estimates b, and then use it to do so with the y series contained within `stan_survival.csv`. (Hint: Stan functions must be declared at the top of a Stan program.)

An example Stan file is given below,

```
functions{
  real fCustomProb(real aY, real aB){
    real aConst;
    aConst = (aB ^ (1.0 / 3.0));
    return(log(aConst) - aB * (aY ^ 3));
  }
}

data{
  int N;
  real Y[N];
}

parameters{
  real<lower=0> b;
}

model{
  for(i in 1:N)
    target += fCustomProb(Y[i], b);
```

```
}
```

which when run estimates a posterior mean for $b \approx 2.42$.

16.4 Is a tumour benign or malignant?

Suppose that if a tumour is benign the result of a clinical test for the disease for individual i is $X_i \sim \mathcal{B}(20, \theta_b)$, whereas if the tumour is malignant $X_i \sim \mathcal{B}(20, \theta_m)$, where $\theta_b < \theta_m$. Suppose that we collect data on 10 patients' scores on this clinical test $X = \{4, 18, 6, 4, 5, 6, 4, 6, 16, 7\}$ and would like to infer the disease status for each individual, as well as the parameters (θ_b, θ_m).

Problem 16.4.1. Write down in pseudo-code the full model, where we suppose that we use uniform priors on (θ_b, θ_m) and discrete uniform priors on the disease status s_i of individual i.

This looks like,

$$s_i \sim \text{discrete-uniform}(1, 2) \tag{16.9}$$
$$\text{if} \quad (s_i = 1) \tag{16.10}$$
$$X_i \sim \mathcal{B}(10, \theta_b) \tag{16.11}$$
$$\text{else} \tag{16.12}$$
$$X_i \sim \mathcal{B}(10, \theta_m) \tag{16.13}$$

Problem 16.4.2. Assuming that $s_i \in [1, 2]$ is the disease status of each individual (1 corresponding to a benign growth, and 2 to a malignant one), use the `transformed parameters` block to calculate the log probability of each individual's data. (Hint: this will be a 10×2 matrix, where the 2 corresponds to two possible disease statuses for each individual.)

```
transformed parameters{
  matrix[10, 2] lp;
  for(i in 1:10)
    for(s in 1:2)
      lp[i,s] = log(0.5) + binomial_lpmf(X[i] | N, theta[s]);
}
```

Problem 16.4.3. The disease status of each individual $s_i \in [1, 2]$ is a discrete variable, and because Stan does not support discrete parameters directly it is not as straightforward to code up these problems as for continuous parameter problems. The way that to do this is by marginalising out s_i from the joint distribution,

$$p(\theta_b, \theta_m | X) = \sum_{s_1=1}^{2} p(\theta_b, \theta_m, s_1 | X) \tag{16.14}$$

where we have illustrated this for the disease status of individual 1. This then allows us to find an expression for the posterior density which we log to give lp, and then use `target+=lp` to increment the log probability. However, because we do this on the log density scale we instead do the following,

$$\log p(\theta_b, \theta_m|X) = \log \sum_{s_1=1}^{2} p(\theta_b, \theta_m, s_1|X) \tag{16.15}$$

$$= \log \sum_{s_1=1}^{2} exp\left(log\ p(\theta_b, \theta_m, s_1|X)\right) \tag{16.16}$$

$$= \text{log\_sum\_exp}_{s_1=1}^{2}(\log p(\theta_b, \theta_m, s_1|X)) \tag{16.17}$$

where `log_sum_exp(.)` (a function available in Stan) is defined as,

$$\text{log\_sum\_exp}_{i=1}^{K}\alpha = log \sum_{i=1}^{K} exp(\alpha) \tag{16.18}$$

and is a more numerically-stable way of doing the above calculation. Using this knowledge, write a full Stan model that implements this marginalisation, and use it to estimate θ_b and θ_m. (Hint: use the `binomial_logit_lpmf(X[i]|N,alpha[s])` function in Stan and define `ordered[2] alpha`, then transform from the unconstrained alpha to theta using `inv_logit`.)

The below code is one such implementation of this model,

```
data {
  int<lower=1> nStudy; // number studies
  int<lower=1> N; // samples per study
  int<lower=0, upper=N> X[nStudy]; // number successes
}

parameters {
  ordered[2] alpha;
}

transformed parameters{
  real<lower=0,upper=1> theta[2];
  matrix[nStudy, 2] lp;
  for(i in 1:2)
    theta[i] = inv_logit(alpha[i]);

for(n in 1:nStudy)
  for(s in 1:2)
    lp[n,s] = log(0.5) + binomial_logit_lpmf(X[n] | N, alpha[s]);
}

model {
```

```
  for(n in 1:nStudy)
    target += log_sum_exp(lp[n]);
}
```

This should yield $\theta_b \approx 0.2 - 0.3$ and $\theta_m \approx 0.8 - 0.9$.

Problem 16.4.4. We use the `generated quantities` block to estimate the probabilities of state $s = 1$ in each different experiment by averaging over all L posterior draws,

$$q(s = 1|X) \approx \frac{1}{L} \sum_{i=1}^{L} q(s = 1, \text{alpha}[s = 1]|X) \tag{16.19}$$

where $q(.)$ is the un-normalised posterior density. The averaging over all posterior draws is necessary to marginalize out the alpha parameter. To normalise the posterior density we therefore divide the above by the sum of the un-normalised probability across both states,

$$Pr(s = 1|X) = \frac{q(s = 1|X)}{q(s = 1|X) + q(s = 2|X)} \tag{16.20}$$

Using the above knowledge add a `generated quantities` block to your Stan model that does this, and hence estimate the probability that each individual's tumour is benign.

```
generated quantities{
  matrix[nStudy, 2] pstate;
  for(n in 1:nStudy)
    pstate[n] = exp(lp[n] - log_sum_exp(lp[n]));
}
```

Apart from unlucky individuals 2 and 9 where $p(benign) \approx 0$ all other individual's $p(benign) \approx 1$.

Problem 16.4.5. An alternative way to code this problem is to derive a Gibbs sampler. As a first step in this process write out the full joint posterior numerator. (Hint: now use a slightly-altered definition of $s_i \in [0, 1]$, where 1 indicates a benign tumour for individual i.)

Since the priors are uniform for all variables we have,

$$p(\theta_b, \theta_m|X, S) \propto \left(\prod_{i=1}^{10} \theta_b^{s_i X_i} (1 - \theta_b)^{s_i (20 - X_i)} \right) \left(\prod_{i=1}^{10} \theta_m^{(1-s_i) X_i} (1 - \theta_m)^{(1-s_i)(20-X_i)} \right) \tag{16.21}$$

$$= \left(\theta_b^{\sum_{i=1}^{10} s_i X_i} (1 - \theta_b)^{\sum_{i=1}^{10} s_i (20 - X_i)} \right) \left(\theta_m^{\sum_{i=1}^{10} (1-s_i) X_i} (1 - \theta_m)^{\sum_{i=1}^{10} (1-s_i)(20-X_i)} \right) \tag{16.22}$$

Problem 16.4.6. By removing those terms that don't depend on θ_b derive the conditional distribution $\theta_b|\theta_m, S, X$. Hence write down $\theta_m|\theta_b, S, X$

This amounts to removing the second half of the above yielding,

$$p(\theta_b|\theta_m, S, X) \propto \theta_b^{\sum_{i=1}^{10} s_i X_i}(1-\theta_b)^{20\sum_{i=1}^{10} s_i - \sum_{i=1}^{10} s_i X_i} \tag{16.23}$$

$$\sim beta(1 + \sum_{i=1}^{10} s_i X_i, 1 + 20\sum_{i=1}^{10} s_i - \sum_{i=1}^{10} s_i X_i) \tag{16.24}$$

Hence by symmetry we have that,

$$\theta_m|\theta_b, S, X \sim beta(1 + \sum_{i=1}^{10}(1-s_i)X_i, 1 + 20\sum_{i=1}^{10}(1-s_i) - \sum_{i=1}^{10}(1-s_i)X_i) \tag{16.25}$$

Problem 16.4.7. Show that the distribution for $s_i|s_{-i}, \theta_b, \theta_m, X$ can be written as,

$$s_i|s_{-i}, \theta_b, \theta_m, X \sim Bernoulli\left(\frac{1}{1 + \left[\frac{\theta_m}{1-\theta_m}/\frac{\theta_b}{1-\theta_b}\right]^{X_i}\left[\frac{1-\theta_m}{1-\theta_b}\right]^{20}}\right) \tag{16.26}$$

All the s_{-i} terms drop out leaving,

$$p(s_i|s_{-i}, \theta_b, \theta_m, X) \propto \theta_b^{s_i X_i}(1-\theta_b)^{s_i(20-X_i)}\theta_m^{(1-s_i)X_i}(1-\theta_m)^{(1-s_i)(20-X_i)} \tag{16.27}$$

This distribution only has two discrete values corresponding to $s_i = 0, 1$, so it can be deduced that this conditional density is a Bernoulli. This can written compactly after some algebra as,

$$s_i|s_{-i}, \theta_b, \theta_m, X \sim Bernoulli\left(\frac{1}{1 + \left[\frac{\theta_m}{1-\theta_m}/\frac{\theta_b}{1-\theta_b}\right]^{X_i}\left[\frac{1-\theta_m}{1-\theta_b}\right]^{20}}\right) \tag{16.28}$$

where as $\theta_m \to 1 \implies p(s_i = 1) \to 0$, or if $\theta_b \to 1 \implies p(s_i = 1) \to 1$, and *ceteris paribus* that as $X_i \uparrow \implies p(s_i) \downarrow$.

Problem 16.4.8. Using your three derived conditional distributions create a Gibbs sampler in R, and use it to estimate $(\theta_b, \theta_m, s_1, ..., s_{10})$.

I use three functions to do this,

```
fSampleThetaB <- function(X, S){
  aSumXS <- sum(X * S)
  aSumS <- sum(S)
  return(rbeta(1, (1 + aSumXS), (1 + 20 * aSumS - aSumXS)))
}

fSampleThetaM <- function(X, S){
  aSumXS <- sum(X * (1 - S))
  aSum1S <- sum(1 - S)
  return(rbeta(1, (1 + aSumXS),(1 + 20 * aSum1S - aSumXS)))
}

fSampleS <- function(thetaB, thetaM, X){
  S <- vector(length=10)
  logOddsM <- thetaM / (1 - thetaM)
  logOddsB <- thetaB / (1 - thetaB)
  aExtra <- ((1 - thetaM) / (1 - thetaB))
  for(i in 1:10){
    aProb <- 1 / (1 + ((logOddsM / logOddsB) ^ X[i]) * (aExtra ^ 20))
    S[i] <- rbinom(1, 1, aProb)
  }
  return(S)
}
```

Again you should to obtain a mean of around 0.3 for θ_b and 0.9 for θ_m, with all individuals apart from 2 and 9 having benign growths.

16.5 How many times did I flip the coin?

Suppose that I have a coin with θ denoting the probability of it landing heads-up. In each experiment I flip the coin N times, where N is unknown to the observer, and record the number of heads obtained Y. I repeat the experiment 10 times, each time flipping the coin the same N times, and record $Y = \{9, 7, 11, 10, 10, 9, 8, 11, 9, 11\}$ heads.

Problem 16.5.1. Write down an expression for the likelihood, stating any assumptions you make.

Assuming independent and identically-distributed observations we obtain,

$$Y_i \sim \mathcal{B}(N, \theta) \tag{16.29}$$

Problem 16.5.2. Suppose that the maximum number of times the coin could be flipped is 20, and that all other (allowed) values we regard *a priori* as equally probable. Further suppose that based on previous coin flipping fun that we specify a prior $\theta \sim beta(7, 2)$. Write down the model as a whole (namely, the likelihood and the priors).

$$Y_i \sim \mathcal{B}(N, \theta) \tag{16.30}$$
$$N \sim \text{discrete-uniform}(11, 20) \tag{16.31}$$
$$\theta \sim beta(7, 2) \tag{16.32}$$

Problem 16.5.3. This problem can be coded in Stan by marginalising out the discrete parameter N. The key to doing this is writing down an expression for the log-probability for each result Y_i conditional on an assumed value of N, and θ. Explain why this can be written in Stan as,

```
log(0.1) + binomial_lpmf(Y[i] | N[s], theta);
```

where `N[s]` is the sth element of a vector N containing all possible values for this variable.

Problem 16.5.4. In the `transformed parameters` block write code that calculates the log probability for each experiment and each possible value of N.

```
transformed parameters{
  vector[10] lp;
  for(s in 1:10)
    lp[s] = log(0.1) + binomial_lpmf(Y | N[s], theta);
}
```

The above uses the vectorised form of the RHS, which is important because it allows N to be the same across all experiments (I fell foul of this when I initially coded it up!)

Problem 16.5.5. Write a Stan program to estimate θ. (Hint: in the `model` block use `target+= log_sum_exp(lp)` to marginalise out N and increment the log probability.)

The Stan program is,

```
data{
  int K;
  int Y[K];
}

transformed data{
  int N[10];
  for(s in 1:10)
    N[s] = 10 + s;
}

parameters{
  real<lower=0, upper=1> theta;
}

transformed parameters{
```

```
  vector[10] lp;
  for(s in 1:10)
    lp[s] = log(0.1) + binomial_lpmf(Y | N[s], theta);
}

model{
  target += log_sum_exp(lp);
  theta ~ beta(7,2);
}
```

For a definition of `log_sum_exp(.)` see the answer to the next question. The posterior mean of θ is about 0.78, with a 95% HDI of $0.54 \leq \theta \leq 0.91$.

Problem 16.5.6. Use the `generated quantities` block to estimate the probabilities of each state.

This relies on us estimating the un-normalised density for the number of coin flips by averaging over all samples for θ,

$$q(N = 11|Y) \approx \frac{1}{L} \sum_{i=1}^{L} q(N = 11, \theta_i|Y) \tag{16.33}$$

where $q(.)$ is the un-normalised posterior density and L is the number of posterior samples. To normalise this density we then divide the above by the un-normalised density for all other possible values for N,

$$Pr(N = 11|Y) = \frac{q(N = 11|Y)}{\sum_{N=11}^{20} q(N = i|Y)} \tag{16.34}$$

To do this in Stan we use `log_sum_exp(.)` which is defined as,

$$\text{log\_sum\_exp}_{i=1}^{K} \alpha = log \sum_{i=1}^{K} exp(\alpha) \tag{16.35}$$

which allows us to marginalise out any dependence on N in log prob space because,

$$\log p(\theta) = \log \sum_{N=1}^{K} p(\theta, N) \tag{16.36}$$

$$= \log \sum_{N=1}^{K} exp\left(log\ p(\theta, N)\right) \tag{16.37}$$

$$= \text{log\_sum\_exp}_{N=1}^{K}(\log p(\theta, N)) \tag{16.38}$$

So implementing this in Stan we have,

```
generated quantities {
  simplex[10] pState;
  pState = exp(lp - log_sum_exp(lp));
}
```

which results in probabilities of each state shown in Figure 16.2. Note that it is better to use the expectation once, rather than do exp(something) divided by exp(something else) because this risks numerical accuracy.

Problem 16.5.7. An alternative way to estimate N and θ is to derive a Gibbs sampler for this problem. To do this first show that the joint (un-normalised) posterior distribution can be written as,

$$p(\theta, N|Y) \propto \left[\prod_{i=1}^{K} \binom{N}{Y_i} \theta^{Y_i}(1-\theta)^{N-Y_i}\right] \theta^{\alpha-1}(1-\theta^{\beta-1}) \tag{16.39}$$

where $K = 10$ and $(\alpha, \beta) = (7, 2)$ are the parameters of the prior distribution for θ.

The above is self-evident if you drop all the non-constant terms in the likelihood and prior.

Problem 16.5.8. Derive the conditional distribution $\theta|N, Y$. (Hint: remove all parts of the joint distribution that do not explicitly depend on θ.)

Removing those θ-independent parts of the joint distribution we obtain,

$$p(\theta|N, Y) \propto \theta^{\alpha-1+\sum Y_i}(1-\theta)^{\beta-1+KN-\sum Y_i} \tag{16.40}$$

which is recognisable as a $\theta|N, Y \sim beta(\alpha + \sum Y_i, \beta + KN - \sum Y_i)$ distribution!

Problem 16.5.9. Write an R function that independently samples from the conditional distribution $\theta|N, Y$.

An example R function is shown below,

```
fSampleTheta <- function(Y, N, alpha, beta){
  K <- length(Y)
  aSum.Y <- sum(Y)
  aSum.N <- N * K
  return(rbeta(1, alpha + aSum.Y, beta + aSum.N - aSum.Y))
}
```

Problem 16.5.10. Show that the conditional pmf $N|\theta, Y$ can be written as,

$$p(N|\theta, Y) \propto \left[\prod_{i=1}^{K} \binom{N}{Y_i}\right] (1-\theta)^{NK} \tag{16.41}$$

Removing all those parts of the joint distribution that do not depend on N we obtain the above.

Problem 16.5.11. Using the previously-derived expression, write a function that calculates the un-normalised conditional $N|\theta, Y$ for $N = 11, ..., 20$, which when normalised can be used to sample a value for N. Hint use the `sample` function in R.

```
fSampleN <- function(Y, theta){
  lUnnorm <- vector(length=10)
  K <- length(Y)
  for(i in 1:10){
    N <- i + 10
    lUnnorm[i] <- prod(sapply(Y, function(x) choose(N, x))) *
                                            (1 - theta) ^ (N * K)
  }
  lProb <- lUnnorm / sum(lUnnorm)
  return(sample(11:20, 1, prob=lProb))
}
```

Problem 16.5.12. Write a working Gibbs sampler using your two previously-created functions, and use this to estimate the probability distribution over θ and N.

I first create a function that randomly picks an updating order for θ and N, then samples from these.

```
fGibbsSingle <- function(Y, theta, N, alpha, beta){
  aInd <- rbinom(1, 1, 0.5)
  if(aInd==1){
    theta.new <- fSampleTheta(Y, N, alpha, beta)
    N.new <- fSampleN(Y, theta.new)
  }else{
    N.new <- fSampleN(Y, theta)
    theta.new <- fSampleTheta(Y, N.new, alpha, beta)
  }
  return(list(theta=theta.new, N=N.new))
}
```

Then string these together into a Gibbs routine,

```
fGibbsTotal <- function(numIter, Y, theta, N, alpha, beta){
  lTheta <- vector(length=numIter)
  lN <- vector(length=numIter)
  lTheta[1] <- theta
  lN[1] <- N
  for(i in 2:numIter){
    lParams <- fGibbsSingle(Y, lTheta[i - 1], lN[i - 1], alpha, beta)
    lTheta[i] <- lParams$theta
    lN[i] <- lParams$N
  }
```

```
  return(list(theta=lTheta, N=lN))
}
```

Then running this we obtain a distribution of values of N and θ sampled,

```
lSamples <- fGibbsTotal(10000, Y, 0.78, 11, 7, 2)
hist(lSamples$N)
hist(lSamples$theta)
```

which should have similar means to that of HMC for θ. The distribution for N is not the same, because in HMC we are sampling $p(N)$ not N itself!

Problem 16.5.13. Compare the rate of convergence in the mean of N sampled via Gibbs with that over that estimated from the $p(N)$ distribution that you sampled in HMC. Why is the rate of convergence so much faster for HMC? (Hint: this is not due to the standard benefits of HMC that I extolled in this chapter.)

First you will need to determine the expected value of N from $p(N)$ from Stan

```
fExpectation <- function(mStates){
  lN <- apply(mStates, 1, function(x) sum(x * seq(11, 20)))
  return(lN)
}
lExpected <- fExpectation(lPState)
```

Then examining a running mean of N over time from the Gibbs versus HMC we see that the latter is much faster to converge (Figure 16.3),

```
fRunningMean <- function(lSample){
  return(cumsum(lSample) / seq_along(lSample))
}
aDF <- data.frame(time=1:2000,
                  gibbs=fRunningMean(lSamples$N),
                  hmc=fRunningMean(lExpected))
aDF <- melt(aDF, id.vars='time')
ggplot(aDF, aes(x=time, y=value, colour=as.factor(variable))) + geom_line()
```

This is due to Rao-Blackwellisation – by marginalising out any dependence on N, and using the marginal distribution $p(\theta)$ to infer $p(N)$ we get a significant speed up.

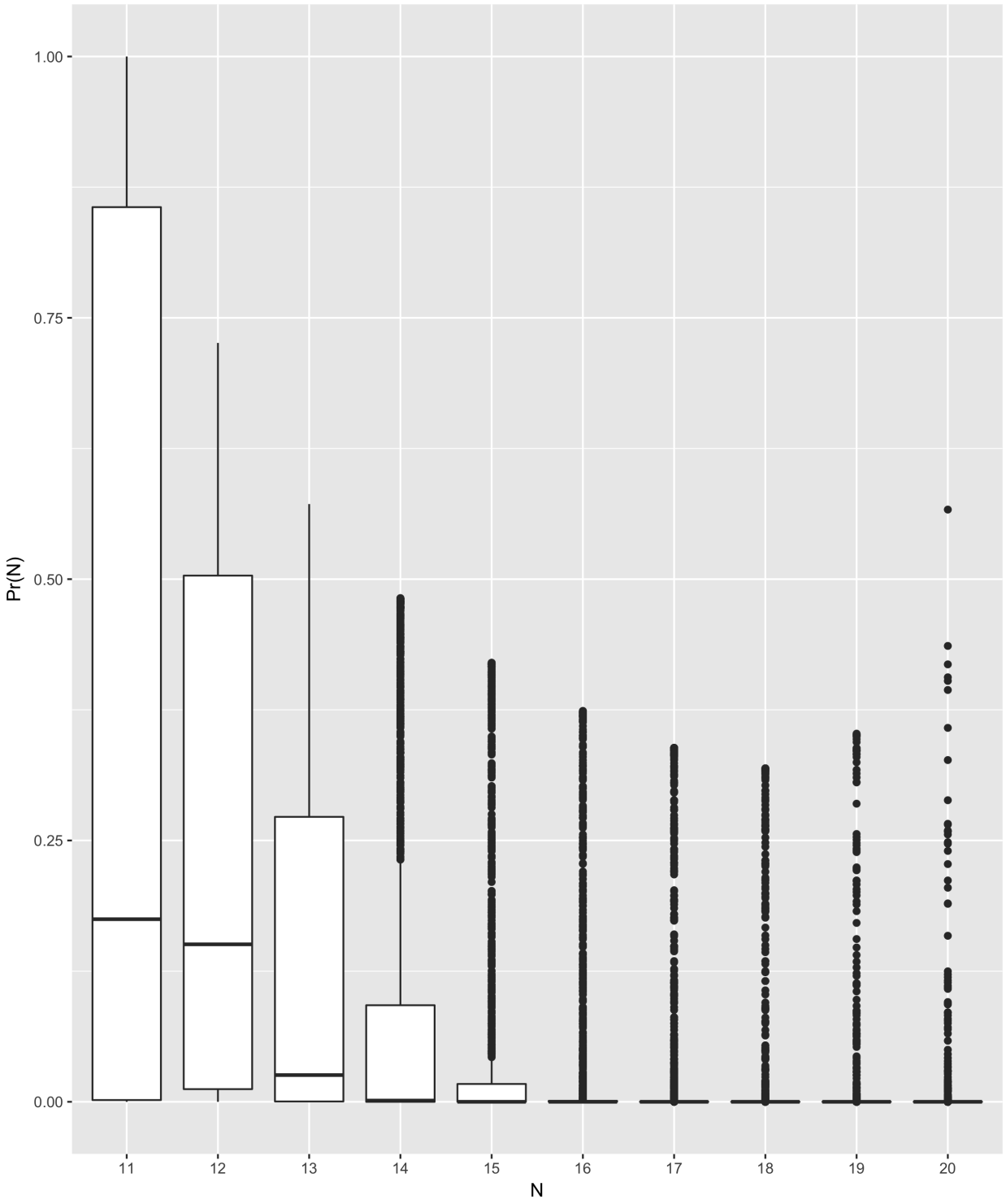

Figure 16.2: The estimated probabilities for each N in the coin flipping example.

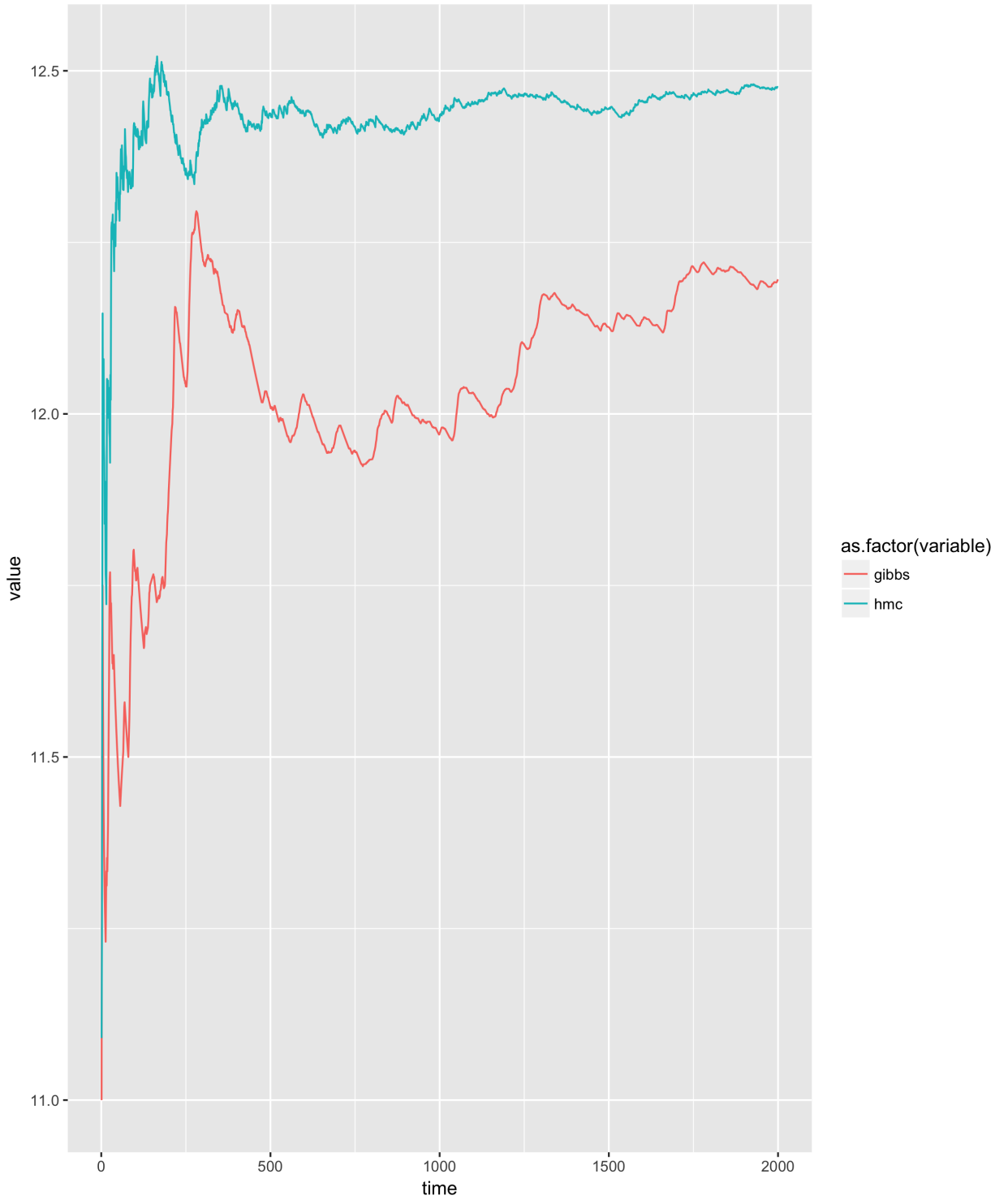

Figure 16.3: The rate of convergence of the mean N for the coin flipping example by HMC and Gibbs.

Bibliography

[1] *The World Almanac and Book of Facts.* 1975.