Chapter 17

Hierarchical models

17.1 A meta-analysis of beta blocker trials

Table 17.1 shows the results of some of the 22 trials included in a meta-analysis of clinical trial data on the effect of beta-blockers on reducing the risk of myocardial infarction [2]. The file `hierarchical_betaBlocker.csv` contains the full dataset.

The aim of this meta-analysis is to determine a robust estimate of the effect of beta-blockers by pooling information from a range of previous studies (this problem has been adapted from [3]).

Table 17.1: The data from the original study.

	Mortality: deaths/total	
Study	**Treated**	**Control**
1	3/38	3/39
2	7/114	14/116
3	5/69	11/93
4	102/1533	127/1520
...		
20	32/209	40/218
21	27/391	43/364
22	22/680	39/647

Problem 17.1.1. Start by assuming that the numbers of deaths in the control (r_i^c) and treated (r_i^t) groups for each trial are given by binomial distributions of the form:

$$r_i^c \sim \mathcal{B}(p_i^c, n_i^c) \tag{17.1}$$
$$r_i^t \sim \mathcal{B}(p_i^t, n_i^t) \tag{17.2}$$

where (n_i^t, n_i^c) are the numbers of individuals in the treatment and control datasets respectively. Further assume that the probabilities of mortality in the treatment and control datasets are given by:

$$logit(p_i^c) = \mu_i \tag{17.3}$$
$$logit(p_i^t) = \mu_i + \delta_i \tag{17.4}$$
$$\tag{17.5}$$

where $logit(x) = log\left(\frac{x}{1-x}\right)$, and we expect $\delta_i < 0$ if the beta-blockers have the desired effect. We assume the following diffuse priors for the parameters

$$\mu_i \sim \mathcal{N}(0, 10) \tag{17.6}$$
$$\delta_i \sim \mathcal{N}(0, 10) \tag{17.7}$$
$$\tag{17.8}$$

Estimate the posteriors for δ_i for the above model using Stan, or otherwise. Note: that for this model there is no inter-dependence between the studies. (Hint: use the Stan function `binomial_logit`.)

The code to estimate this model is given by:

```
data {
  int<lower=0> N;
  int<lower=0> nt[N];
  int<lower=0> rt[N];
  int<lower=0> nc[N];
  int<lower=0> rc[N];
}

parameters {
  vector[N] mu;
  vector[N] delta;
}

model {
  rt ~ binomial_logit(nt, mu + delta);
  rc ~ binomial_logit(nc, mu);
  delta  ~ normal(0, 10);
  mu ~ normal(0, 10);
}
```

The posteriors for δ_i in this model are fairly wide and contain non-zero densities at zero (Figure 17.1).

Problem 17.1.2. An alternative framework is a hierarchical model where we assume there to be a common over-arching distribution, across trials such that $\delta_i \sim \mathcal{N}(d, \sigma)$. By assuming the following

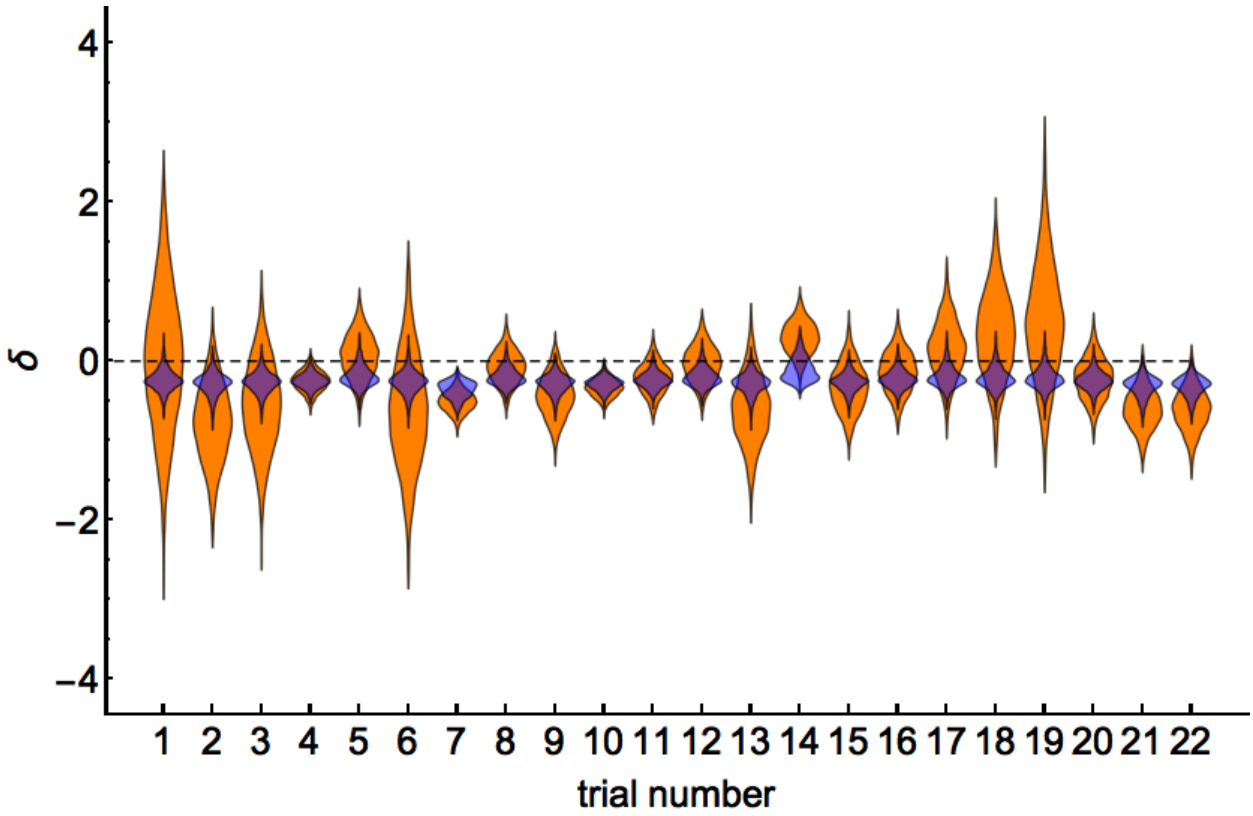

Figure 17.1: The posterior estimates of δ_i for the fully-heterogeneous (orange) and hierarchical (blue) models.

priors on these parameters estimate this model:

$$d \sim \mathcal{N}(0, 10) \tag{17.9}$$

$$\sigma \sim Cauchy(0, 2.5), \text{ for } \sigma \geq 0 \tag{17.10}$$

Estimate the posteriors for δ_i using Stan. How do these estimates compare to the non-hierarchical model?

The code for this problem is shown below:

```
data {
  int<lower=0> N;
  int<lower=0> nt[N];
  int<lower=0> rt[N];
  int<lower=0> nc[N];
  int<lower=0> rc[N];
}

parameters {
  real d;
  real<lower=0> sigma;
```

```
  vector[N] mu;
  vector[N] delta;
}

model {
  rt ~ binomial_logit(nt, mu + delta);
  rc ~ binomial_logit(nc, mu);
  delta  ~ normal(d, sigma);
  mu ~ normal(0, 10);
  d ~ normal(0, 10);
  sigma ~ cauchy(0, 2.5);
}
```

When I used the above code I ran into a few divergent iterations - this should **not** be ignored, and is best handled by increasing adapt_delta=0.95 when calling Stan. This should help the sampler avoid taking too large steps, and diverging in areas of high posterior curvature. The problematic regions of parameter space here are due to the correlation in estimates between d and the various δ_i; this is unsurprising, each trial only has a relatively small data sample. As such, it is difficult to disentangle the specific individual study effects from the overall effect d.

The hierarchical estimates of the effect of the drug are much more concentrated (Figure 17.1) - by pooling data across all studies we are better able to precisely estimate the effect of the drug. These indicate that the beta-blockers appear to act as desired - decreasing the probability of mortality.

Problem 17.1.3. Using the hierarchical model estimate the cross-study effect of the beta-blockers. (Hint: use the `generated quantities` code block.)

The code for to sample an overall δ is given below:

```
generated quantities {
  real delta_overall;
  delta_new = normal_rng(d, sigma);
}
```

Overall we estimate a negative value for δ (Figure 17.2). Whilst the posterior does overlap zero, we are fairly confident in concluding that $\delta < 0$.

Problem 17.1.4. For an out of sample trial suppose we know that $\mu_i = -2.5$. Using the cross-study estimates for δ estimate the reduction in probability for a patient taking the beta-blockers.

This is done by using the inverse-logit transformation (the logistic sigmoid). Essentially you want to evaluate logistic-sigmoid(-2.5) and compare it with logistic-sigmoid(-2.5-delta), across all the samples in your model. This results in a posterior distribution that is peaked at about 0.015 (Figure 17.3); indicating about a 1.5% reduction in mortality risk for those patients taking beta-blockers.

Problem 17.1.5. Estimate a model with a single, constant value of δ and μ across all trials. Graph the posterior for δ, and compare it with the cross-study hierarchical model estimate.

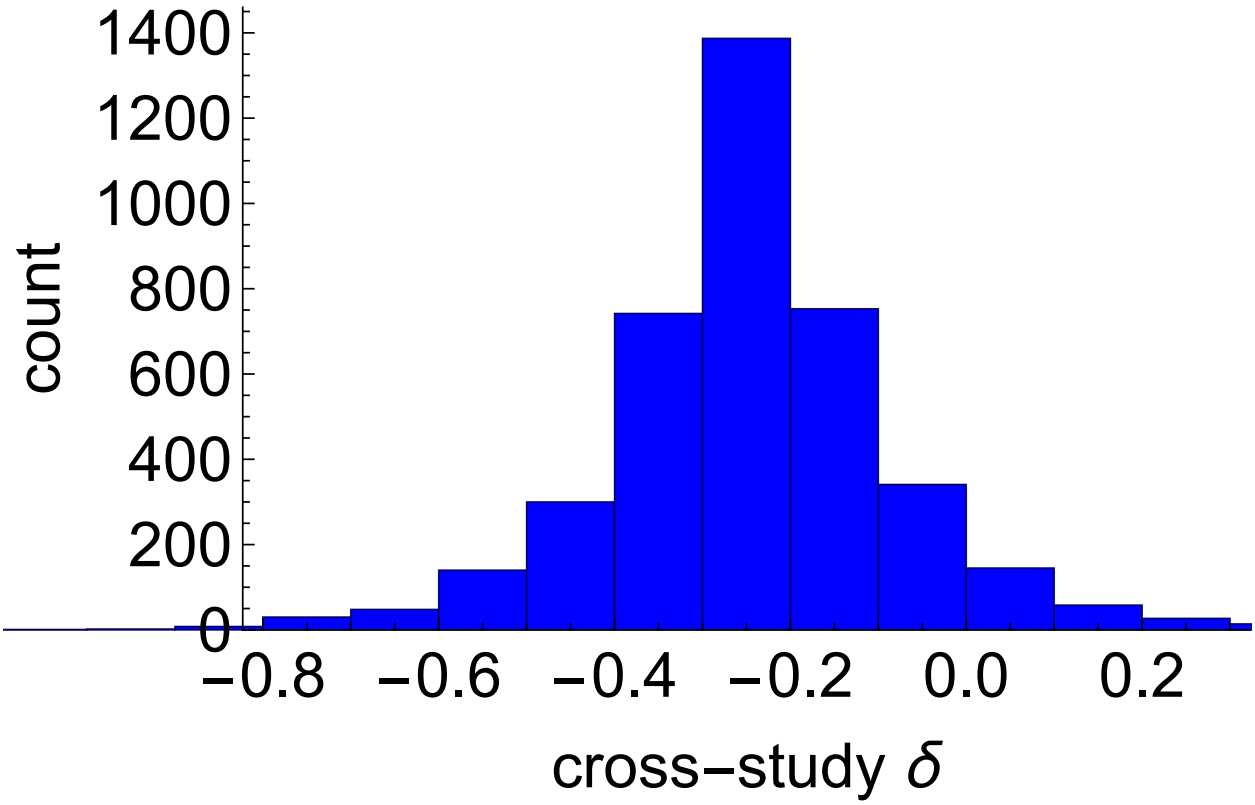

Figure 17.2: The posterior estimate of δ across all trials for the hierarchical model.

The non-hierarchical model gives us a false confidence in our estimates of δ, by assuming that the data from the individual studies are equivalent (exchangeable). This means that the estimate of δ obtained is more concentrated than for the hierarchical model (Figure 17.4).

Problem 17.1.6. Carry out appropriate posterior predictive checks on the homogeneous and hierarchical models, and hence conclude the preferred modelling choice.

One check to do here is to generate posterior predictive data sets of the same shape as the real data, and for each trial record whether the simulated value is greater than the actual. This can be done fairly easily using the `generated quantities` block (shown here for the homogeneous model):

```
generated quantities {
  int<lower=0> simTreatMort[N];
  int<lower=0> simContrMort[N];
  int indicatorTreat[N];
  int indicatorContr[N];
  for (i in 1:N) {
    simTreatMort[i] = binomial_rng(nt[i], inv_logit(mu + delta));
    simContrMort[i] = binomial_rng(nc[i], inv_logit(mu));
    indicatorTreat[i] = (simTreatMort[i] > rt[i]);
    indicatorContr[i] = (simContrMort[i] > rc[i]);
```

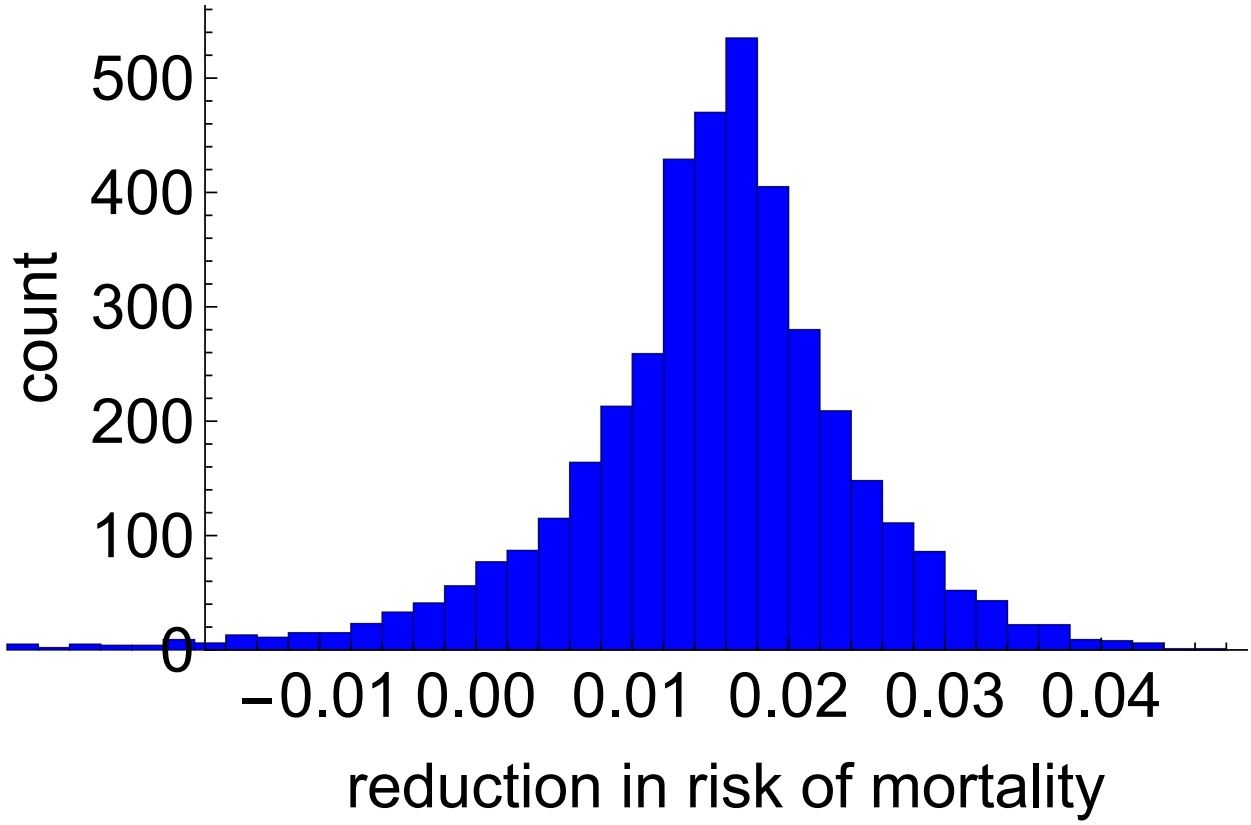

Figure 17.3: The posterior estimates of the reduction in mortality risk associated with taking a beta-blocker when $\mu = -2.5$.

```
  }
}
```

Doing this for both models we find that there are a range of Bayesian p values near 0 or 1 for the homogeneous model, whereas this is not the case for the hierarchical model (Figure 17.5.) Intuitively - by assuming that there was no difference between the data from each study - the homogeneous coefficient model is unable to replicate the degree of variation we see in the real data. We therefore prefer the hierarchical model.

17.2 I can't get no sleep

These data are from a study described in Belenky et al. (2003) [1] that measured the effect of sleep deprivation on cognitive performance. There were 18 subjects chosen from a population of interest (lorry drivers) who were restricted to 3 hours of sleep during the trial. On each day of the experiment their reaction time to a visual stimulus was measured. The data for this example are contained within `evaluation_sleepstudy.csv`, and contains three variables: Reaction, Days and Subject ID which measure the reaction time of a given subject on a particular day.

A simple model that explains the variation in reaction times is a linear regression model of the form:

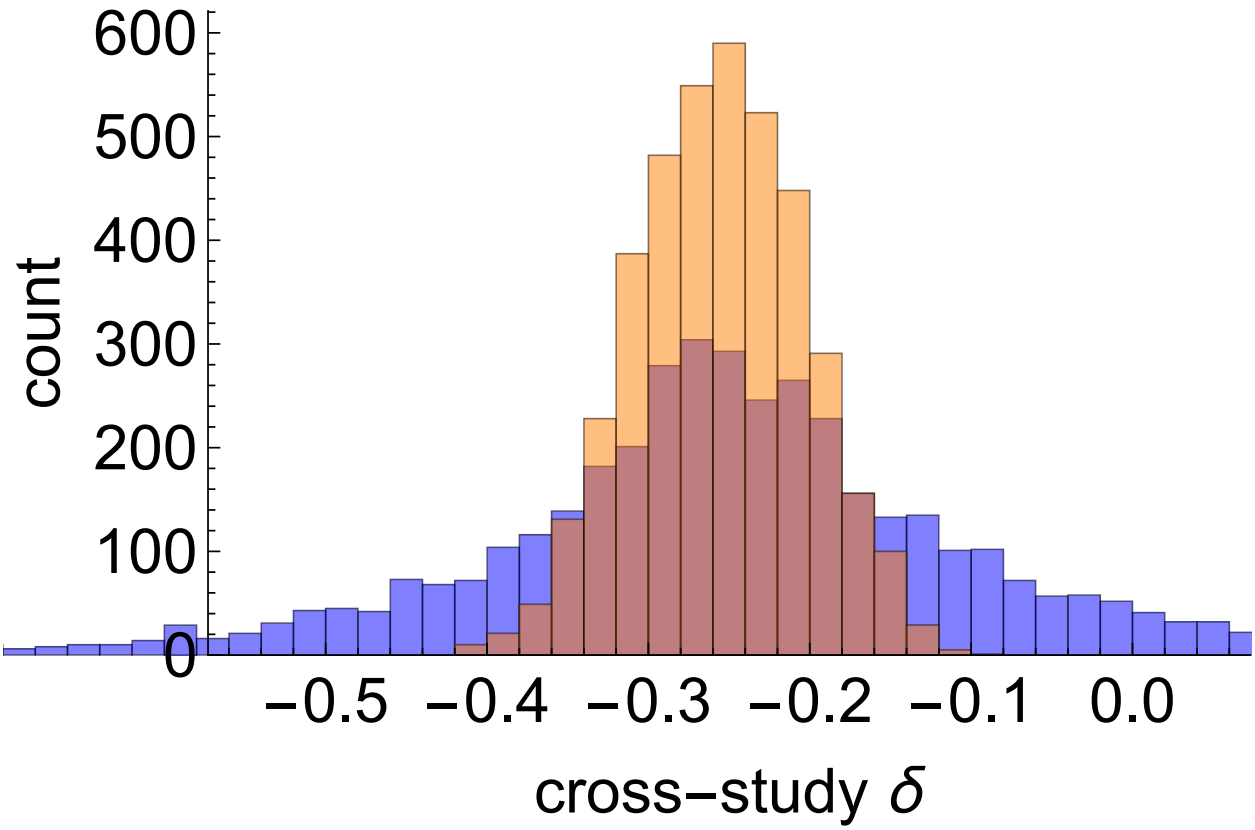

Figure 17.4: The posterior estimates of cross-study δ for the hierarchical (blue) and homogeneous (orange) models.

$$R(t) \sim \mathcal{N}(\alpha + \beta t, \sigma) \tag{17.11}$$

where $R(t)$ is the reaction time on day t of the experiment across all observations.

Problem 17.2.1. Assuming $\mathcal{N}(0, 250)$ priors on both α and β code up the above model in Stan. Use it to generate 1000 samples per chain, across 4 chains. Has the sampling algorithm converged?

```
data {
  int N; // number of observations
  matrix[N,2] X; // ones + days of sleep deprivation
  vector[N] R; // reaction times
}

parameters {
  vector[2] gamma;
  real<lower=0> sigma;
}
```

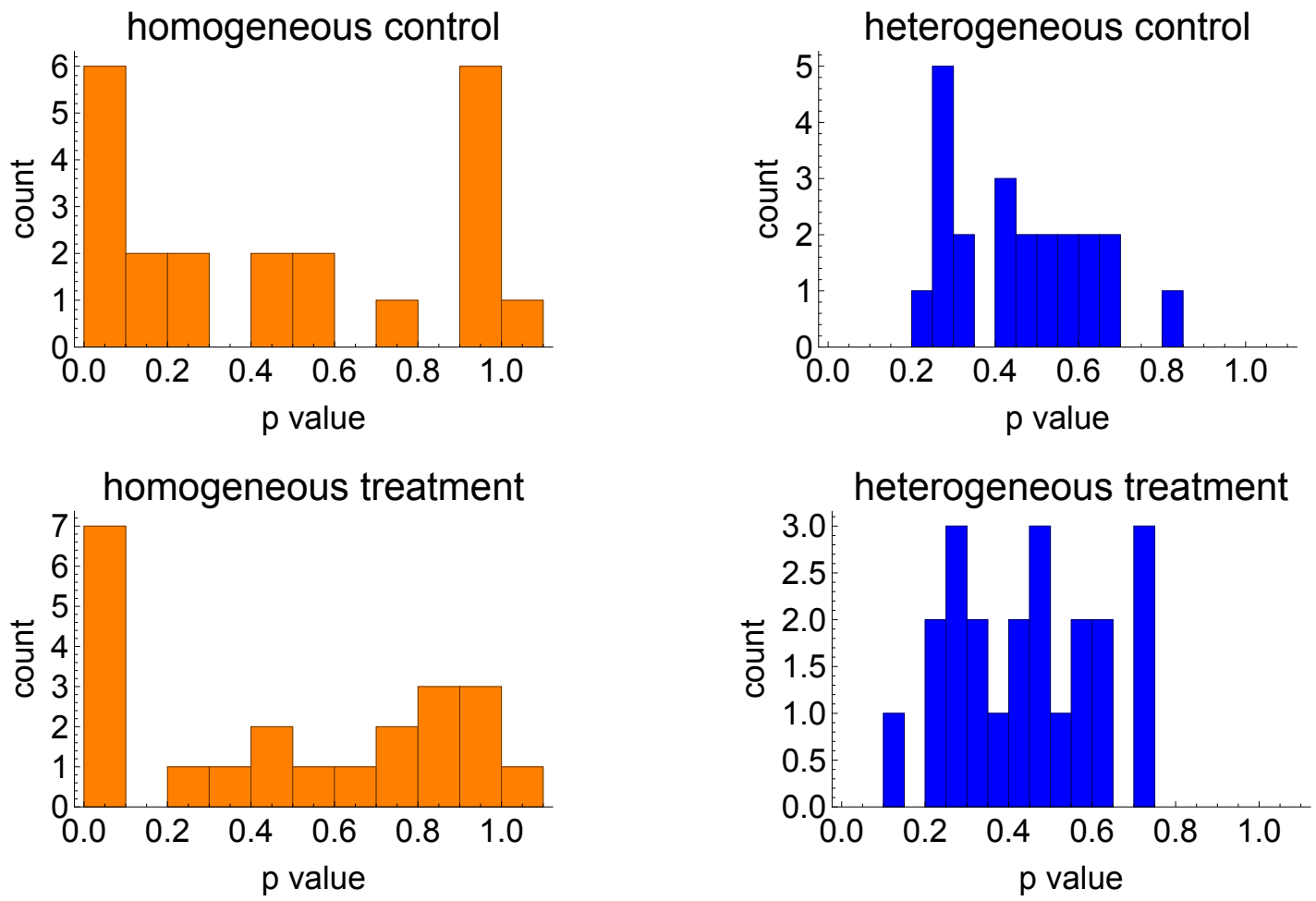

Figure 17.5: The distribution of Bayesian p values measuring whether the posterior predictive data exceeds the actual across each of the 22 trials for the homogeneous (orange) and hierarchical (blue) models.

```
model {
  R ~ normal(X * gamma, sigma);
 gamma ~ normal(0, 250);
}
```

After the requisite number of iterates the value for $\hat{R} < 1.1$, meaning it appears that the sampling distribution has converged.

Problem 17.2.2. Plot the posterior samples for α and β. What is the relationship between the two variables, and why?

There is a strong negative correlation between the estimates of these two variables. This is because to generate a line going through the centre of the dataset, if the intercept increases, the gradient must decrease.

Problem 17.2.3. By using the `generated quantities` code block or otherwise generate samples from the posterior predictive distribution. By overlaying the real time series for each individual on a graph of the posterior predictive comment on the fit of the model to data.

The posterior predictive distribution - whilst being a reasonable fit to the data for the anonymous data - is not able to fit well the data at the individual level (Figure 17.6).

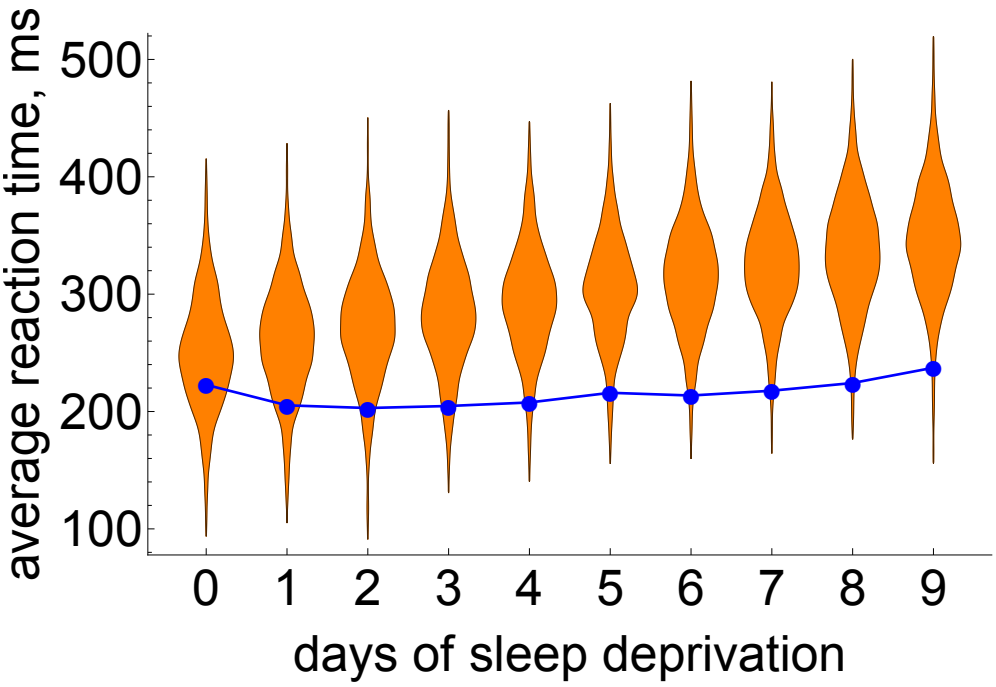

Figure 17.6: The posterior predictive distribution (orange) and the data for one individual in the sleep study (blue) using the "homogeneous coefficients" model.

Problem 17.2.4. Fit a model with separate (α, β) for each individual in the dataset. Use separate and independent $\mathcal{N}(0, 250)$ priors for the parameters. Again use 1000 samples per chain over 4 chains.

This is best done by creating an index of 1 to 18; corresponding to individual subject ID. The model can then be coded up as below:

```
data {
  int N; // number of observations
  vector[N] t; // days of sleep deprivation
  vector[N] R; // reaction times of individuals in the study
  int subject[N]; // subject ID
}

parameters {
  real alpha[18];
  real beta[18];
```

```
  real<lower=0> sigma;
}

model {
  for (i in 1:N)
    R[i] ~ normal(alpha[subject[i]] + beta[subject[i]] * t[i], sigma);

  alpha ~ normal(0, 250);
  beta ~ normal(0, 250);
  sigma ~ normal(0, 50);
}
```

Problem 17.2.5. Compute the posterior mean estimates of the β parameters for the new "heterogeneous-parameters" model. How do these compare to the single β estimate obtained for the homogeneous model?

The homogeneous estimate is about 10.4, with the heterogeneous estimates ranging from -2.8 (for subject 9) to 21.9 (for subject 1). Overall the heterogeneous estimates should have a mean that is roughly similar to the single estimate (it's not exactly so).

Problem 17.2.6. Using the `generated quantities` code block, or otherwise, generate samples from the posterior predictive distribution. By comparing individual subject data to the posterior predictive samples, comment on the fit of the new model.

The posterior predictive distribution can be generated in similar fashion to previously, but using the individual subject IDs as an array index.

```
generated quantities {
  vector[N] R_simulated; // store post-pred samples
  for (i in 1:N)
    R_simulated[i] = normal_rng(alpha[subject[i]] + beta[subject[i]] * t[i],
                                sigma);
}
```

The heterogeneous coefficients model is able to fit the data much more effectively at the individual data (Figure 17.7). This is unsurprising - essentially we may be guilty of overfitting the model to the data.

Problem 17.2.7. Partition the data into two subsets: a training set (of subjects 1-17) and a testing set (of subject 18 only). By fitting both models - the heterogeneous and homogeneous coefficients models - on the training sets, compare the performance of each model on predicting the test set data.

To do this I create two new data variables that hold only the data for the 18th subject. I then change the original data so that it only holds data for subjects 1-17. Of course these manipulations could be handled directly in the Stan code itself, but I prefer to do this outside. The code for the heterogeneous model is shown below:

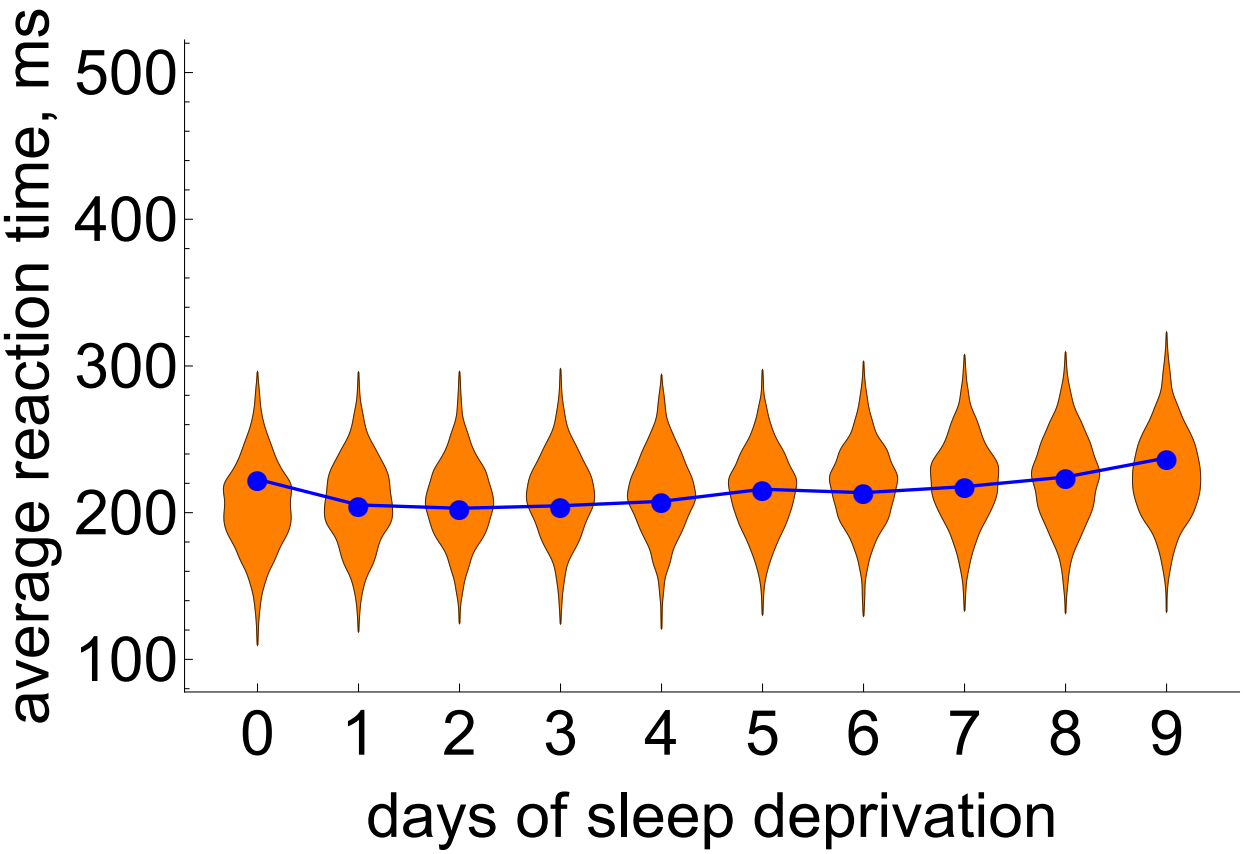

Figure 17.7: The posterior predictive distribution (orange) and the data for one individual in the sleep study (blue) using the "heterogeneous coefficients" model.

```
data {
  int N; // number of observations in training set
  vector[N] t; // days of sleep deprivation in training set
  vector[N] R; // reaction times of individuals in the training set
  int subject[N];
  int N2; // number of data points in the test set
  vector[N2] t2; // time obs in test set
}

parameters {
  real alpha;
  real beta;
  real<lower=0> sigma;
}

model {
  for (i in 1:N)
    R[i] ~ normal(alpha[subject[i]] + beta[subject[i]] * t[i], sigma);
```

```
  alpha ~ normal(0, 250);
  beta ~ normal(0, 250);
  sigma ~ normal(0, 50);
}

generated quantities {
  vector[N2] R_simulated; // store post-pred samples
  real aAlpha;
  real aBeta;

  aAlpha = normal_rng(0, 250);
  aBeta = normal_rng(0, 250);
  for (i in 1:N2)
    R_simulated[i] = normal_rng(aAlpha + aBeta * t2[i], sigma);
}
```

For the heterogeneous model there is really only one way to generate predictions for the test set - sample a value of the parameters from the priors, and using these parameter values to generate predictive datasets. Because the priors are wide this actually produces very poor predictions (Figure 17.8).

The homogeneous coefficients model however performs much better as is much more generalisable to new datasets. Intuitively the heterogeneous coefficients model is overfit to the data.

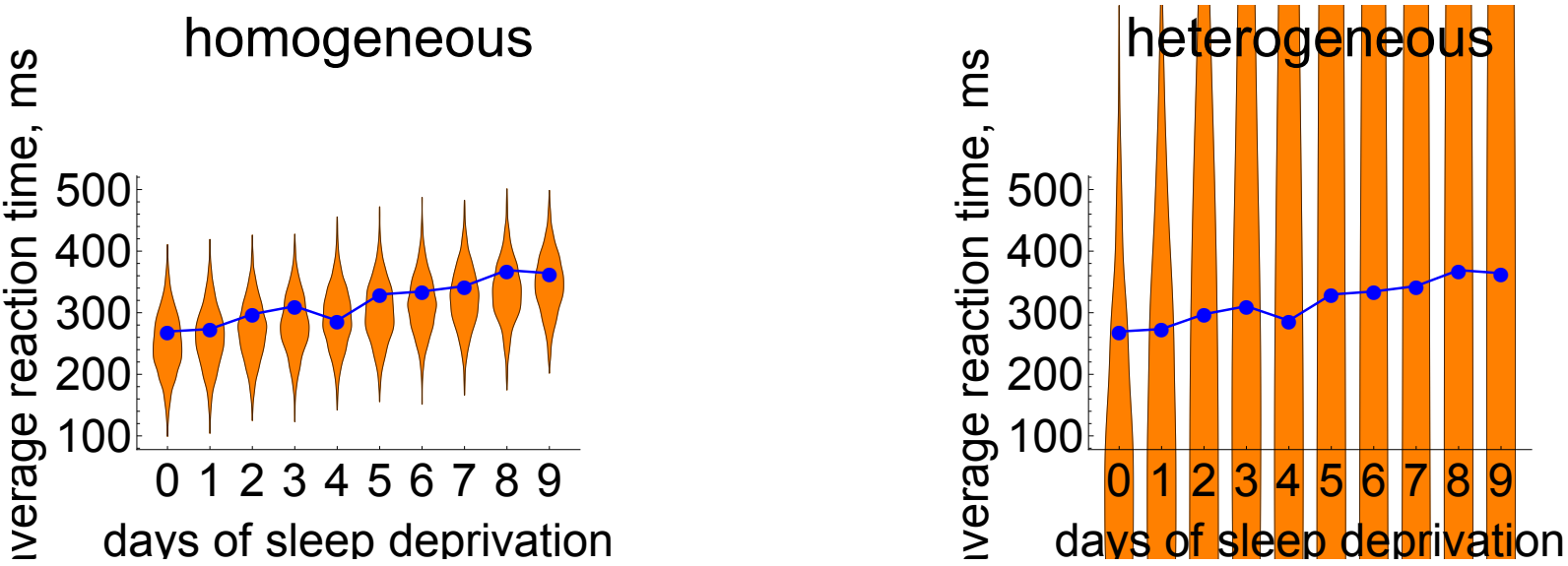

Figure 17.8: The posterior predictive distribution (orange) and the data for subject 18 for a model fitted to the other 17 subjects' data, across the homogeneous coefficients (left), and heterogeneous coefficients (right) models.

Problem 17.2.8. Alternatively we can fit a hierarchical model to the data which (hopefully) captures some of the best elements of each of the aforementioned models. Here we assume that the

individual (α, β) for each subject are allowed to vary, but there is some overarching "population-level" distribution from which they are drawn. Assume that the coefficients have the following relationships:

$$\alpha_i \sim \mathcal{N}(a, b) \tag{17.12}$$
$$\beta_i \sim \mathcal{N}(c, d) \tag{17.13}$$
$$a \sim \mathcal{N}(100, 100) \tag{17.14}$$
$$b \sim Cauchy(0, 5) \tag{17.15}$$
$$c \sim \mathcal{N}(10, 5) \tag{17.16}$$
$$d \sim Cauchy(0, 1) \tag{17.17}$$

Code up the above model and compare the posterior distribution for β for the hierarchical model, with those from the heterogeneous ones.

The posterior distribution for the parameters exhibits shrinkage towards the grand mean (Figure 17.9). In general those parameter estimates with a. the highest uncertainty, and b. lie furthest away from the mean, are shrunk the most in hierarchical models.

Problem 17.2.9. Graph the posterior distribution for β for another individual (not in the original dataset). How does this distribution compare to the value of β obtained from the homogeneous coefficient model? (Hint: use the `generated quantities` block to generate samples of β from the top-level parameters c and d.)

The code to sample a value of β for an out-of-sample individual is given below:

```
generated quantities {
  real aBeta;
  aBeta = normal_rng(c, d);
}
```

The posterior distribution for β has a mean of 10.2 (about the same as the original homogeneous estimate), but is wider (Figure 17.10).

17.3 Hierarchical ODEs: Bacterial cell population growth

The file `hierarchical_ode.csv` contains data for 5 replicates of an experiment in which bacterial cell population numbers were measured over time. The following model for bacterial population size is proposed to explain the data:

$$\frac{\mathrm{d}N}{\mathrm{d}t} = \alpha N(1 - \beta N). \tag{17.18}$$

However measurement of bacterial cell numbers is subject to random, uncorrelated measurement error:

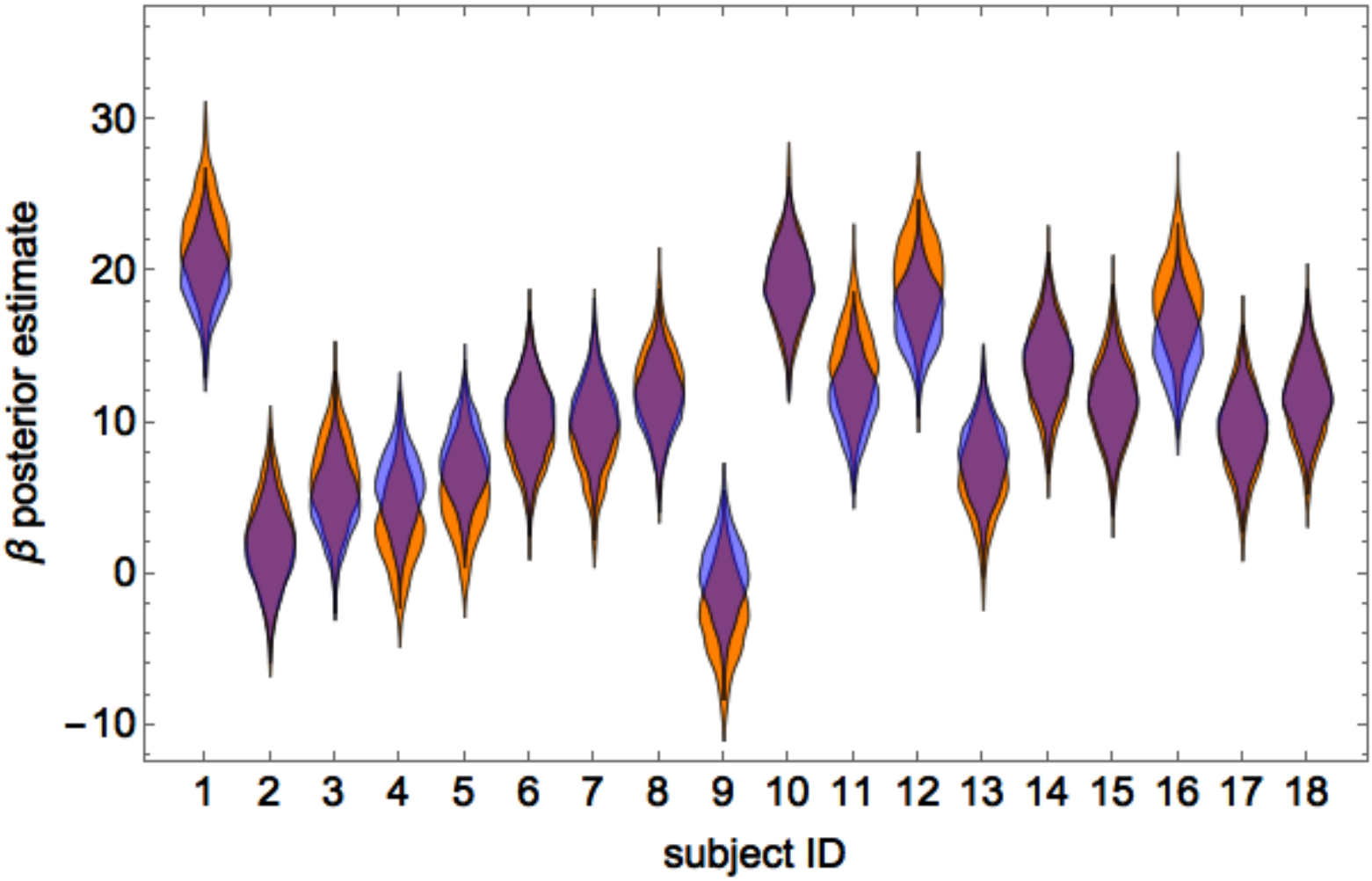

Figure 17.9: The posterior estimates of β for the heterogeneous estimates (orange) versus the hierarchical estimates (blue).

$$N^*(t) \sim \mathcal{N}(N(t), \sigma), \tag{17.19}$$

where $N^*(t)$ is the measured number of cells, and $N(t)$ is the true population size. Finally we suppose that the initial number of bacteria cells is unknown, and hence must be estimated.

Further we assume the following priors:

$$\begin{aligned} \alpha &\sim \mathcal{N}(0, 2) \\ \beta &\sim \mathcal{N}(0, 2) \\ \sigma &\sim \mathcal{N}(0, 1) \\ N(0) &\sim \mathcal{N}(5, 2) \end{aligned}$$

where all parameters have a lower value of zero.

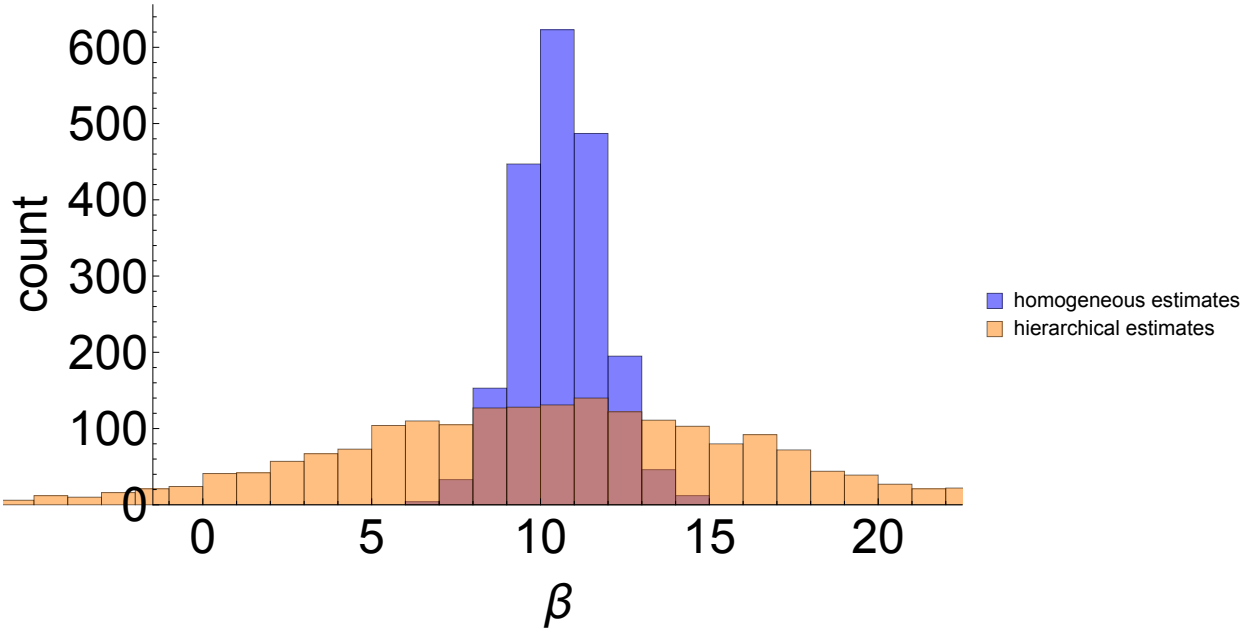

Figure 17.10: The posterior estimates of β for an out-of-sample individual for the homogeneous coefficients and hierarchical models.

Problem 17.3.1. Write a Stan function that returns $\frac{\mathrm{d}N}{\mathrm{d}t}$. (Hint 1: this will need to be done within the "functions" block at the top of the Stan file. Hint 2: the function must have a structure:

```
real[] bacteria_deriv(real t, real[] y, real[] theta, real[] x_r, int[] x_i)
```

where the variables x_i and x_r are not used here, but nonetheless need to be defined:

```
transformed data {
real x_r[0];
int x_i[0];
}
```

)

See answer to problem 17.3.2 for code.

Problem 17.3.2. Estimate a model where the parameters (α, β) are assumed to be the same across all experimental replicates.

```
functions {
  real[] bacteria_deriv(real t, real[] y, real[] theta, real[] x_r, int[] x_i) {
    real dydt[1];

    dydt[1] = theta[1] * y[1] * (1 - theta[2] * y[1]);
    return dydt;
  }
```

```
}

data {
  int<lower=1> T;
  int<lower=0> N;
  real t0;
  real ts[T];
  matrix[T,N] y;
}

transformed data {
  real x_r[0];
  int x_i[0];
}

parameters {
  real<lower=0, upper=2> theta[2]; // contains parameters (alpha,beta)
  real<lower=0> sigma;
  real<lower=0, upper=10> y0[1];
}

model {
  real y_hat[T, 1];
  sigma ~ cauchy(0, 1);
  theta ~ normal(0, 2);
  y0 ~ normal(5, 2);
  y_hat = integrate_ode(bacteria_deriv, y0, t0, ts, theta, x_r, x_i);
  for (i in 1:N)
    for (t in 1:T)
      y[t, i] ~ normal(y_hat[t, 1], sigma);
}

// use this to capture the log-likelihood for later parts of the question
generated quantities {
  vector[N * T] logLikelihood;
  int k;
  real y_hat[T, 1];

  k = 1;
  y_hat = integrate_ode(bacteria_deriv, y0, t0, ts, theta, x_r, x_i);

  for (i in 1:N){
    for (t in 1:T){
      logLikelihood[k] = normal_log(y[t, i], y_hat[t, 1], sigma);
      k = k + 1;
    }
  }
```

```
}
```

The posteriors for (α, β) are shown in Figure 17.11.

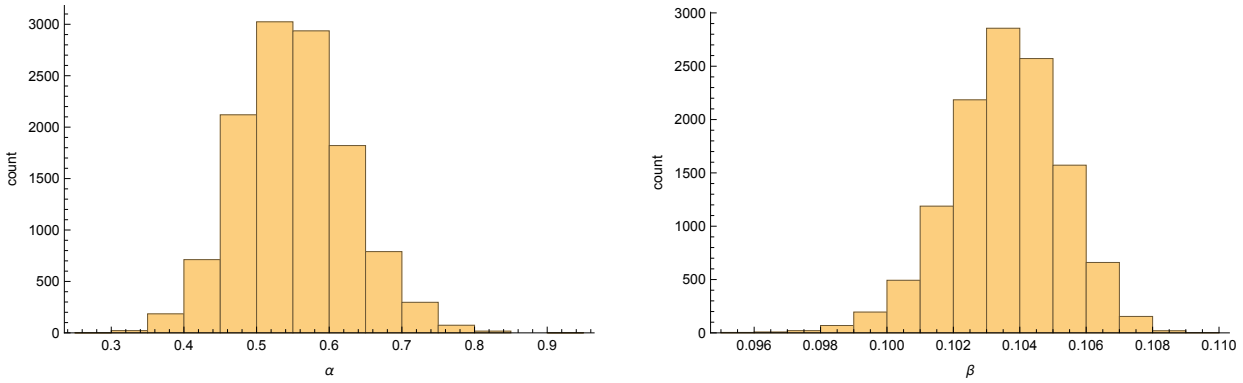

Figure 17.11: Homogeneous model estimates of (α, β).

Problem 17.3.3. By graphing the data, or otherwise, comment on the assumption of a common (α, β) across all replicates.

There is quite a clear variability between the replicates across different experiments (Figure 17.12). This makes the assumption of common parameter values across all replicates look quite weak. An alternative here would be to do some posterior predictive checks, but this isn't really needed here to be honest since the raw data plots are illuminating.

Problem 17.3.4. Now estimate a model that estimates separate values for (α, β) across all replicates. Graph the posterior distribution for each parameter.

```
parameters {
  real<lower=0, upper=2> theta[N, 2];
  real<lower=0> sigma;
  real<lower=0, upper=10> y0[1];
}

model {
  real y_hat[T, 1];
  sigma ~ cauchy(0, 1);
  y0 ~ normal(5, 2);

  for (i in 1:N){
    theta[i] ~ normal(0, 2);
    y_hat = integrate_ode(bacteria_deriv, y0, t0, ts, theta[i], x_r, x_i);
    for (t in 1:T)
      y[t, i] ~ normal(y_hat[t, 1], sigma);
  }
```

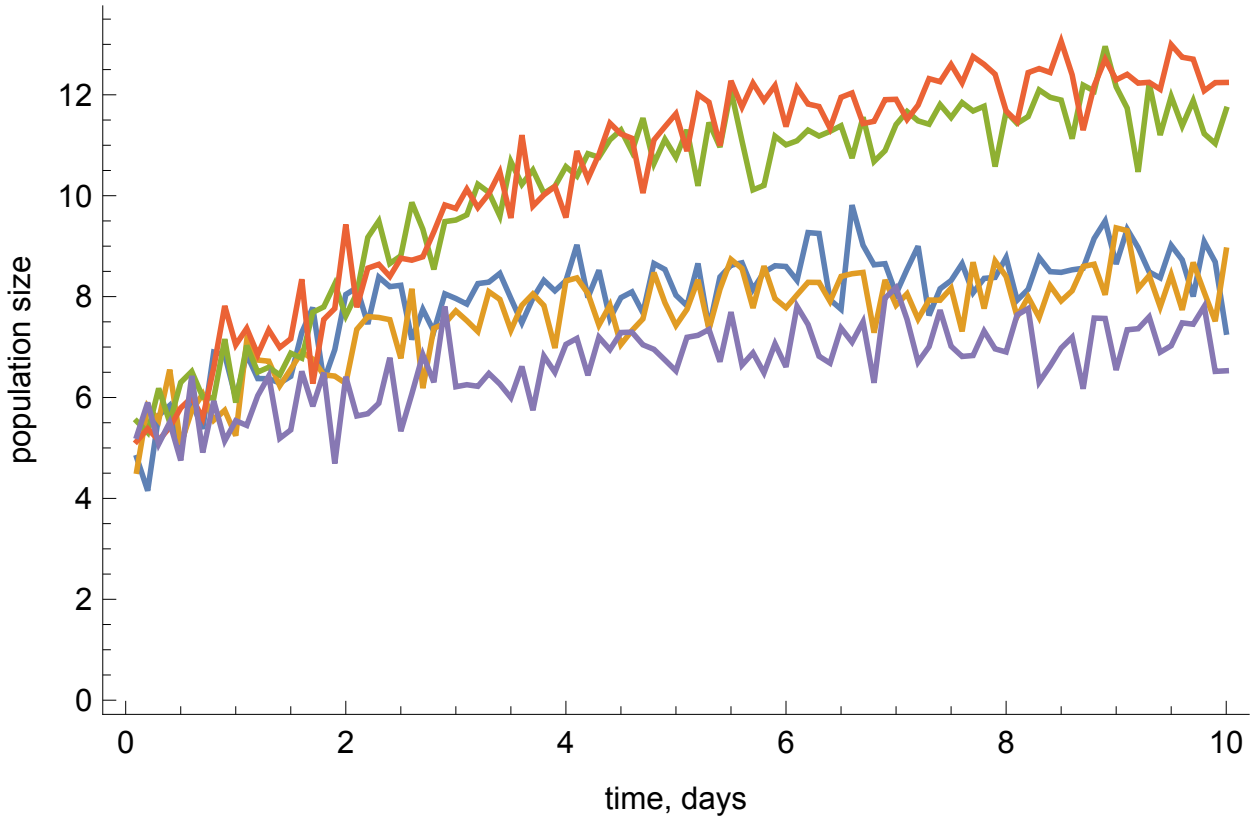

Figure 17.12: Time series plot of 5 experimental replicates.

```
}

generated quantities {
  vector[N * T] logLikelihood;
  int k;
  real y_hat[T, 1];

  k = 1;
  for (i in 1:N){
    y_hat = integrate_ode(bacteria_deriv, y0, t0, ts, theta[i], x_r, x_i);
    for (t in 1:T){
      logLikelihood[k] = normal_log(y[t, i],y_hat[t, 1], sigma);
      k = k + 1;
    }
  }
}
```

There is considerable heterogeneity in posterior estimates of (α, β) (Figure 17.13)

Problem 17.3.5. Estimate a hierarchical model assuming the following priors:

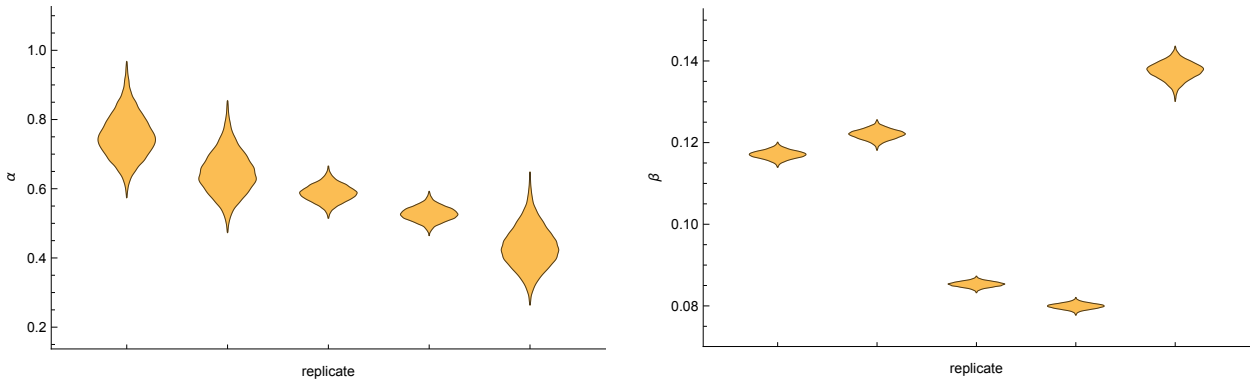

Figure 17.13: Posterior parameter estimates for heterogeneous model.

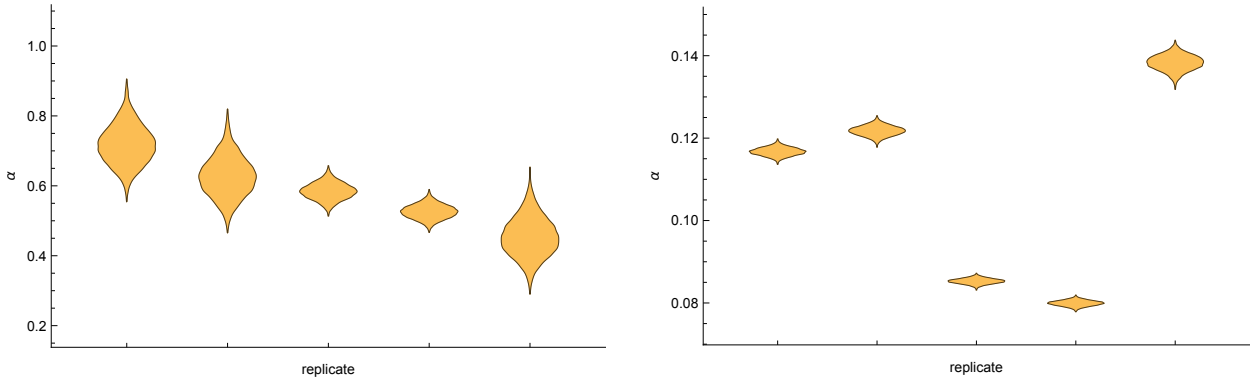

Figure 17.14: Posterior parameter estimates for hierarchical model.

$$
\begin{aligned}
\alpha &\sim \Gamma(a, b) \\
\beta &\sim \Gamma(c, d) \\
a &\sim \mathcal{N}(20, 5) \\
b &\sim \mathcal{N}(40, 5) \\
c &\sim \mathcal{N}(10, 3) \\
d &\sim \mathcal{N}(100, 5)
\end{aligned}
$$

Compare your estimates of (α, β) with those from the completely heterogeneous model.

```
functions {
  real[] bacteria_deriv(real t, real[] y, real[] theta, real[] x_r, int[] x_i) {
    real dydt[1];

    dydt[1] = theta[1] * y[1] * (1 - theta[2] * y[1]);
    return dydt;
```

```
  }
}

data {
  int<lower=1> T;
  int<lower=0> N;
  real t0;
  real ts[T];
  matrix[T, N] y;
}

transformed data {
  real x_r[0];
  int x_i[0];
}

parameters {
  real<lower=0> a1[2];
  real<lower=0> a2[2];
  real<lower=0, upper=2> theta[N, 2];
  real<lower=0> sigma;
  real<lower=0, upper=10> y0[1];
}

model {
  real y_hat[T, 1];
  a1[1] ~ normal(20, 5);
  a1[2] ~ normal(40, 5);
  a2[1] ~ normal(10, 3);
  a2[2] ~ normal(100, 5);
  sigma ~ cauchy(0, 1);
  y0 ~ normal(5, 2);

  for (i in 1:N){
    theta[i, 1] ~ gamma(a1[1], a1[2]);
    theta[i, 2] ~ gamma(a2[1], a2[2]);
    y_hat = integrate_ode(bacteria_deriv, y0, t0, ts, theta[i], x_r, x_i);
    for (t in 1:T)
      y[t, i] ~ normal(y_hat[t, 1], sigma);
  }
}

generated quantities {
  vector[N * T] logLikelihood;
  int k;
  real y_hat[T, 1];
  real aTheta[2];
```

```
  real y_hat_overall[T, 1];

  aTheta[1] = gamma_rng(a1[1], a1[2]);
  aTheta[2] = gamma_rng(a2[1], a2[2]);
  y_hat_overall = integrate_ode(bacteria_deriv, y0, t0, ts, aTheta, x_r, x_i);
  k = 1;
  for (i in 1:N){
    y_hat = integrate_ode(bacteria_deriv, y0, t0, ts, theta[i], x_r, x_i);
    for (t in 1:T){
      logLikelihood[k] = normal_log(y[t, i], y_hat[t, 1], sigma);
      k = k + 1;
    }
  }
}
```

There is very limited shrinkage versus the purely heterogeneous model (Figure 17.13 versus Figure 17.14.) This is because there is quite a lot of data for each replicate.

Problem 17.3.6. Estimate the overall (α, β) for the hierarchical model. How do these compare to the pooled model estimates?

The estimates reflect greater uncertainty compared to the pooled model (Figure 17.15 versus Figure 17.11). This is desirable since the pooled model understates uncertainty.

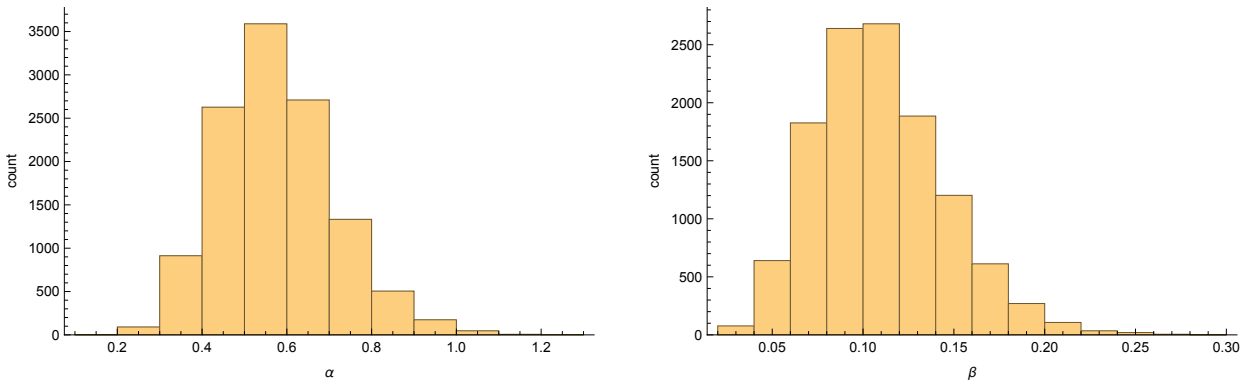

Figure 17.15: Overall parameter estimates for the posterior parameters in the hierarchical model.

Problem 17.3.7. By holding out one of your datasets, compare the predictive performance of each model.

This will favour the hierarchical model (the pooled model performs considerably worse using naive estimates of model performance.)

17.4 Bowel cancer model selection

The file `hierarchical_cancer.csv` contains (fictitious) data on the population size of a given county (N) and the number of bowel cancer cases in that county (X). In this question we aim to build a model to estimate the underlying rate of cancer occurrence λ.

Problem 17.4.1. A simple model is to assume that cancer occurrence is an independent event, and hence we use the following model,

$$X_i \sim Poisson(N_i\lambda) \tag{17.20}$$

where N_i is the population in county i, and X_i is the number of cases of bowel cancer in the same county. In Stan write a model to estimate the underlying rate of bowel cancer occurrence (λ), where we assume a prior of the form $\lambda \sim \mathcal{N}(0.5, 0.5)$.

```
data{
  int K;
  vector[K] N;
  int X[K];
}

parameters{
  real<lower=0> lambda;
}
model{
  X ~ poisson(lambda * N);
  lambda ~ normal(0.5, 0.5);
}
```

which should estimate $\lambda \approx 0.1$.

Problem 17.4.2. Using the `generated quantities` block record the estimated log-likelihood of each data point, for each posterior sample of λ.

```
generated quantities{
  vector[K] lLoglikelihood;
  for(i in 1:K)
    lLoglikelihood[i] = poisson_lpmf(X[i] | N[i] * lambda);
}
```

Problem 17.4.3. By using Stan's `optimizing` function to obtain the MAP estimate of λ, estimate the expected log pointwise predictive density (elpd) via a DIC method,

$$\widehat{\text{elpd}} = log\ p(X|\hat{\theta}_{Bayes}) - \underbrace{2V_{s=1}^{S} log\ p(X|\theta_s)}_{DIC} \tag{17.21}$$

where $V_{s=1}^{S} log\ p(X|\theta_s)$ is the variance in log-likelihood for all data points across S posterior draws. Hint: the latter part of the formula requires that we estimate the model by sampling.

The MAP estimates of the model log-likelihood can be determined using,

```
bFit <- optimizing(aModel, data=list(X=X, N=N, K=length(N)))
likelihoodBayes <- sum(bFit$par[2:1001])
```

Then estimating the model by sampling we can then obtain the p_{DIC} term,

```
fit <- sampling(aModel, data=list(X=X, N=N, K=length(N)), iter=200, chains=4)
lLoglikelihood <- extract_log_lik(fit, 'lLoglikelihood')
aLogLikelihood <- rowSums(lLoglikelihood)
pDIC <- 2 * var(aLogLikelihood)
```

which we then use to estimate $\widehat{\text{elpd}}$,

```
aDIC <- likelihoodBayes - pDIC
```

which should be $\approx$ -3996.

Problem 17.4.4. Estimate elpd using the AIC method. Hint: use Stan's `optimizing` function where the Stan file has had the prior commented out, to achieve the maximum likelihood estimate of the log-likelihood.

The AIC method penalises the estimated log-likelihood by one since there is only a single parameter in the model. Hence this estimate can be obtained by using,

```
bModel <- stan_model('poissonCancerML.stan')
bFit <- optimizing(bModel, data=list(X=X, N=N, K=length(N)))
likelihoodML <- sum(bFit$par[2:1001])
aAIC <- likelihoodML - 1
```

which should yield $\approx$ -3996 (it's slightly higher than that obtained from the DIC method).

Problem 17.4.5. Either manually or using the "loo" package in R estimate the elpd by a WAIC method. If you choose the manual method, this can be done with the following formula,

$$\widehat{\text{elpd}} = \underbrace{\sum_{i=1}^{N} log\left(\frac{1}{S}\sum_{s=1}^{S} p(X_i|\theta_s)\right)}_{\text{log pointwise predictive density}} - p_{WAIC} \tag{17.22}$$

where $p_{WAIC} = \sum_{i=1}^{N} V_{s=1}^{S} var_{post}\left[log\ p(X_i|\theta_s)\right]$.

Using the loo package this is quite straightforward,

```
library(loo)
lLoglikelihood <- extract_log_lik(fit, 'lLoglikelihood')
aWAIC <- waic(lLoglikelihood)
```

which should yield a value of ≈ -3999.

Alternatively, doing this manually,

```
library(matrixStats)
bWAIC_1 <- sum(sapply(seq(1, 1000, 1),
               function(i) logSumExp(lLoglikelihood[, i]) - log(400)))
bWAIC_p <- sum(sapply(seq(1, 1000, 1),
               function(i) var(lLoglikelihood[, i])))
bWAIC <- bWAIC_1 - bWAIC_p
```

where I have used `logSumExp` because it is more numerically stable than doing the piecewise application of the exponential. This should yield a value identical to that obtained via loo ≈ -3999.

Problem 17.4.6. By partitioning the data into 10 folds of training and testing sets (where one data point occurs in each testing set once only), estimate the out-of-sample predictive capability of the model. Hint 1: in R use the "Caret" package's `createFolds` to create 10 non-overlapping folds. Hint 2: adjust your Stan program to calculate the log-likelihood on the test set.

In R the folds can be created by,

```
lFolds <- createFolds(X)
```

The Stan program can be changed to the below which allows the estimation of out-of-sample predictive capability using,

```
data{
  int KTrain;
  vector[KTrain] NTrain;
  int XTrain[KTrain];

  // hold out set
  int KTest;
  vector[KTest] NTest;
  int XTest[KTest];
}

parameters{
  real<lower=0> lambda;
}

model{
  XTrain ~ poisson(lambda * NTrain);
  lambda ~ normal(0.5, 0.5);
```

```
}

generated quantities{
  vector[KTest] lLoglikelihood;
  for(i in 1:KTest)
    lLoglikelihood[i] = poisson_lpmf(XTest[i] | NTest[i] * lambda);
}
```

Then we create a loop in R that stores the log-likelihood for each fold,

```
vLogLikelihood <- vector(length=10, mode='list')
for(i in 1:10){
  print(i)
  aFold <- lFolds[[i]]
  XTrain <- X[-aFold]
  NTrain <- N[-aFold]
  XTest <- X[aFold]
  NTest <- N[aFold]
  aFit <- sampling(aModel, data=list(XTrain=XTrain,
                                     NTrain=NTrain,
                                     KTrain=length(NTrain),
                                     XTest=XTest,
                                     NTest=NTest,
                                     KTest=length(NTest)),
                                     iter=200, chains=4)
  vLogLikelihood[[i]] <- extract_log_lik(aFit, 'lLoglikelihood')
}
```

From which we can estimate the elpd by,

```
aLogTotal <- 0
for(i in 1:10){
  aLogLikeTemp <- vLogLikelihood[[i]]
  aLogTotal <- aLogTotal + sum(colMeans(aLogLikeTemp))
}
```

which should yield $\approx$ -3997. So in this case all measures look reasonably close to the value obtained by cross-validation.

Problem 17.4.7. A colleague suggests fitting a negative binomial sampling model to the data, in case over-dispersion exists. Using a prior $\kappa \sim log - \mathcal{N}(0, 0.5)$ on the dispersion parameter, change your Stan model to use this distribution, and estimate the out-of-sample predictive density using any of the previous methods. Which model do you prefer? Hint: use Stan's `neg_binomial_2` function to increment the log-probability.

The new Stan program should look something like,

```
data{
  int KTrain;
  vector[KTrain] NTrain;
  int XTrain[KTrain];

  // hold out set
  int KTest;
  vector[KTest] NTest;
  int XTest[KTest];
}

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

model{
  XTrain ~ neg_binomial_2(lambda * NTrain, kappa);
  lambda ~ normal(0.5, 0.5);
  kappa ~ lognormal(0, 0.5);
}

generated quantities{
  vector[KTest] lLoglikelihood;
  for(i in 1:KTest)
    lLoglikelihood[i] = neg_binomial_2_lpmf(XTest[i] | NTest[i] * lambda,
                                            kappa);
}
```

And via manual cross-validation (the best method available here) I obtain an estimated elpd $\approx$ -2731. So the negative binomial model is a significantly better fit to the data on the face of it. However, to do this comparison correctly it is necessary to do pairwise comparison, which takes the variability in log likelihood into account, then compares a z score with a standard normal,

```
z <- sum(lLoglikelihood1 - lLoglikelihood2) / (sqrt(length(lLoglikelihood1)) *
         sd(lLoglikelihood1 - lLoglikelihood2))
p <- 1 - pnorm(z)
```

which will be tiny here.

Problem 17.4.8. A straightforward way to estimate the marginal likelihood is to use,

$$p(X) \approx \frac{1}{S} \sum_{s=1}^{S} p(X|\theta_s) \tag{17.23}$$

where $\theta_s \sim p(\theta)$. Either using Stan's `generated quantities` block or otherwise estimate the

marginal likelihood of the Poisson model. (Hint: if you use Stan then you need to use `log_sum_exp` to marginalise the sampled log probabilities.)

In Stan this can be done using the following block,

```
generated quantities{
  real loglikelihood;
  real<lower=0> lambda1;
  lambda1 = normal_rng(0.5, 0.5);
  while(lambda1 <= 0)
    lambda1 = normal_rng(0.5, 0.5);
  loglikelihood = 0;
  for(i in 1:K)
    loglikelihood = loglikelihood + poisson_lpmf(X[i] | lambda1 * N[i]);
}
```

Note in the above we are using samples from the prior **not** the posterior. The above is just using Stan like a random number generator. We can then estimate the marginal likelihood by doing the following in R,

```
library(matrixStats)
lMarginalLog <- extract(fit, 'loglikelihood')[[1]]
logSumExp(lMarginalLog)
```

which should yield a value $\approx$ -3991. However there is a large variance in this estimator for more complex models.

Problem 17.4.9. Estimate the marginal likelihood of the negative binomial model, and hence estimate the log Bayes Factor. Which model do you prefer?

This is best computed on the log scale (using the output from logSumExp), and should yield something like $\log BF \approx 1200$ in favour of the negative binomial model.

Bibliography

[1] Gregory Belenky, Nancy J Wesensten, David R Thorne, Maria L Thomas, Helen C Sing, Daniel P Redmond, Michael B Russo, and Thomas J Balkin. Patterns of performance degradation and restoration during sleep restriction and subsequent recovery: A sleep dose-response study. *Journal of sleep research*, 12(1):1–12, 2003.

[2] John B Carlin. Meta-analysis for 2× 2 tables: A bayesian approach. *Statistics in medicine*, 11(2):141–158, 1992.

[3] David J Spiegelhalter, Nicola G Best, Bradley P Carlin, and Angelika Van Der Linde. Bayesian measures of model complexity and fit. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 64(4):583–639, 2002.