Introduction to Bayesian inference using Rstan: practical 2 solutions

The aim of this practical is to provide some practical experience in writing Stan programmes and using the `rstan` package for posterior inference. Some sections require more experience with statistics than others. These are marked with an asterisk.

First load the `rstan` and `jrRstan` packages:

```
library("rstan")
library("jrRstan")
```

If you have enough RAM, set options to allow parallel computation:

```
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
```

1 *Binomial regression*

In Section 5.2 we considered a generalised linear model in which we assumed a *Poisson* error distribution. In this section we will consider another regression problem, but this time we will use a *binomial* error distribution. This is called *binomial regression*. In the special case where a *logistic*[1] link function is utilised, the name *logistic regression* is often used. We will use a logistic link function in this example.

[1] Other possibilities include the *probit* link or the *complementary log-log* link.

As you might expect, the proportion of people who suffer side effects from a drug typically depends on the dose of that drug they are given. The `jrRstan` package contains a data set called `sideeffect` which can be loaded via:

```
data(sideeffect)
head(sideeffect)

##   dose   n effects
## 1  0.9  46      17
## 2  1.1  72      22
## 3  1.8 118      52
## 4  2.3  96      58
## 5  3.0  84      56
## 6  3.3  53      43
```

For each of a number of doses (`dose`), the data set contains the number (`n`) of patients given a particular drug to treat a medical condition, and the number of those patients suffering from a particular side effect (`effects`).

Suppose that the aim is to model the number Y_i of the n_i patients receiving dose i that suffer side effects in terms of the dose $\tilde{x}_i$ of the drug they are given. The model can be expressed as:

$$Y_i | n_i, p_i \sim \text{Bin}(n_i, p_i), \quad \text{independently for } i = 1, 2, \ldots, N,$$

where here $N = 7$. We will mean centre the covariate, taking $x_i = \tilde{x}_i - \sum_{i=1}^{N} \tilde{x}_i / N$, and use a logistic link function to connect the linear predictor:

$$\eta_i = \beta_1 + \beta_2 x_i$$

to the probability p_i that an individual receiving dose i suffers side effects. In other words:[2]

$$p_i = \frac{e^{\eta_i}}{1 + e^{\eta_i}}.$$

or equivalently:[3]

$$\eta_i = \log\left(\frac{p_i}{1 - p_i}\right).$$

[2] This is called the *logistic* transformation of η_i.

[3] This is called the *logit* (or *inverse logistic*) transformation of p_i.

Our model contains two parameters: β_1 and β_2. We will adopt the following prior:

$$\beta_1 \sim \mathrm{N}(-0.27, 0.68^2), \quad \text{and, independently,} \quad \beta_2 \sim \mathrm{N}(0.47, 0.31^2).$$

- Write a Stan model to represent the model and prior above, remembering to mean-centre the covariate.

```
// Binomial regression model, saved in sideeffect.stan
data {
  int<lower=1> N;           // Number doses
  int<lower=1> K;           /* Number columns in design matrix,
                               i.e. number covariates + 1 */
  matrix[N, K] Xtilde;      // Design matrix
  int<lower=0> y[N];        // Binomial response
  int<lower=0> n[N];        // Number patients (trials)
  real beta_mean[K];        // Means in beta priors
  real<lower=0> beta_sd[K]; // Standard deviations in beta priors
}
transformed data {
  matrix[N, K] X; /* Centred design matrix; we could have
                     avoided this block by performing the
                     transformation in R. */
  X[,1] = Xtilde[,1];
  X[,2] = Xtilde[,2] - mean(Xtilde[,2]);
}
parameters {
  vector[K] beta;
}
model {
  vector[N] eta = X * beta; // Fast matrix-vector calculation
  // Likelihood:
  y ~ binomial_logit(n, eta); /* Arithmetically stable form of
                                 fast, vectorized sampling
                                 statement */
  // Prior:
  beta ~ normal(beta_mean, beta_sd); /* Fast, vectorized sampling
                                        statement */
}
```

- Create a suitable data representation in R then compile and run the Stan programme.

```
## Set up data to pass to stan function in its "data" argument:
N = nrow(sideeffect)          # Number doses, i.e. 7
K = 2                         # Number columns in design matrix
Xtilde = matrix(1, N, K)      # Design matrix
Xtilde[,2] = sideeffect$dose
```

```
y = sideeffect$effects        # Binomial response
n = sideeffect$n              # Number patients (trials)
beta_mean = c(-0.27, 0.47)    # Means in beta priors
beta_sd = c(0.68, 0.31)       # Standard deviations in beta priors

sideeffect_data = list(N=N, K=K, y=y, n=n, Xtilde=Xtilde,
                       beta_mean=beta_mean, beta_sd=beta_sd)

## Compile and run Stan programme:
chains = 4
iter = 2000
thin = 1
sideeffect_stan = stan("sideeffect.stan", data=sideeffect_data,
                       chains=chains, iter=iter, warmup=iter/2,
                       thin=thin)
```

- Check both numerical and graphical diagnostics.

```
## Numerical diagnostics:
print(sideeffect_stan)

## Inference for Stan model: sideeffect.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##            mean se_mean   sd    2.5%     25%     50%     75%   97.5% n_eff
## beta[1]    0.29    0.00 0.10    0.11    0.23    0.29    0.36    0.49  2780
## beta[2]    0.75    0.00 0.10    0.55    0.68    0.75    0.82    0.96  2524
## lp__    -322.70    0.02 1.02 -325.49 -323.09 -322.38 -321.98 -321.72  1801
##         Rhat
## beta[1]    1
## beta[2]    1
## lp__       1
##
## Samples were drawn using NUTS(diag_e) at Fri Jan 17 17:46:47 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).

## Graphical diagnostics:
sideeffect_stan_arr = as.array(sideeffect_stan)
diagnostics(sideeffect_stan_arr)

## NULL

## Rhat values all close to 1, n_eff large, graphical
## diagnostics give no cause for concern.
```

2 *Random slope model for Oxboys data*

In Section 5.3 we examined the `Oxboys` data where $Y_{i,j}$ and $x_{i,j}$ denoted the j-th measurement of height and age on boy i, where $i = 1, \ldots, I$ and $j = 1, \ldots, J$. We can load the data in R through:

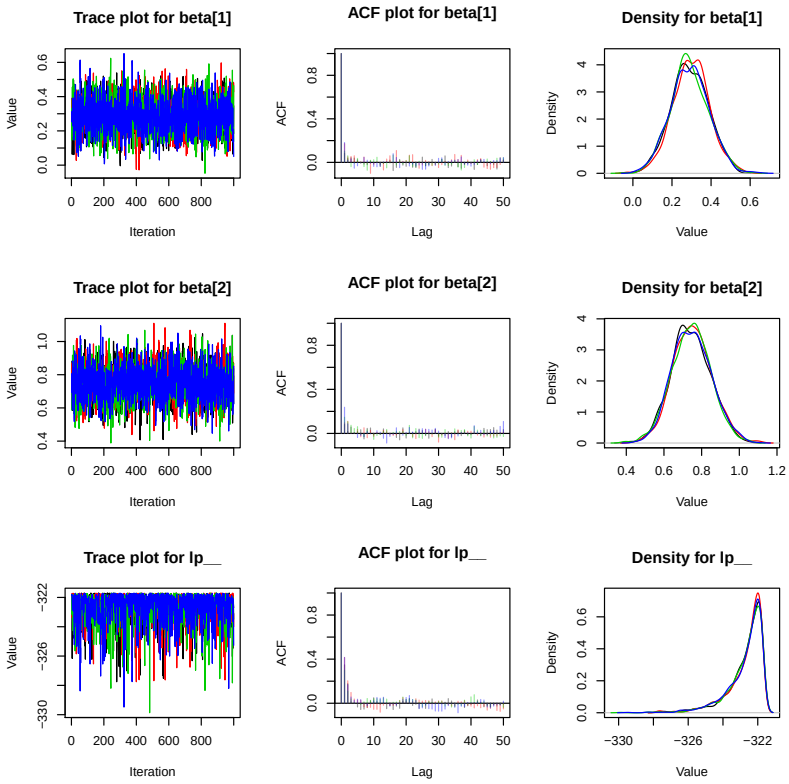

Figure 1: Graphical diagnostics for binomial regression.

```
library(nlme)
data(Oxboys)
head(Oxboys)

## Grouped Data: height ~ age | Subject
##   Subject     age height Occasion
## 1       1 -1.0000  140.5        1
## 2       1 -0.7479  143.4        2
## 3       1 -0.4630  144.8        3
## 4       1 -0.1643  147.1        4
## 5       1 -0.0027  147.7        5
## 6       1  0.2466  150.2        6
```

In lectures, we considered a *random intercept* model in which the intercept term in a regression of the height of a boy on his age could vary between individuals. In this section, we will consider an extension with both a *random intercept* and a *random slope* that can vary from one boy to the next. In other words we will fit a hierarchical model of the form:

$$Y_{i,j} = \alpha_0 + \alpha_{1,i} + (\beta_0 + \beta_{1,i})x_{i,j} + \epsilon_{i,j} \quad \text{where} \quad \epsilon_{i,j}|\sigma^2 \sim \mathrm{N}(0, \sigma^2) \quad (1)$$

independently for $i = 1, 2, \ldots, I$, $j = 1, 2, \ldots, J$ with:

$$\alpha_{1,i}|\sigma^2_{\alpha_1} \sim \mathrm{N}(0, \sigma^2_{\alpha_1}) \quad \text{and} \quad \beta_{1,i}|\sigma^2_{\beta_1} \sim \mathrm{N}(0, \sigma^2_{\beta_1}) \quad (2)$$

independently for $i = 1, 2, \ldots, I$. We now have two sets of random effects: the $\alpha_{1,i}$ and the $\beta_{1,i}$.

We complete our hierarchical model by specifying a prior for the parameters in the "top" level of the model (1):

$$\alpha_0 \sim \mathrm{N}(m_{\alpha_0}, s^2_{\alpha_0}), \quad \beta_0 \sim \mathrm{N}(m_{\beta_0}, s^2_{\beta_0}), \quad \sigma^2 \sim \mathrm{Gam}(a_{\sigma^2}, b_{\sigma^2}),$$

and a *hyperprior* for the parameters in the "bottom" level of the model (2):

$$\sigma^2_\alpha \sim \mathrm{Gam}(a_{\sigma^2_\alpha}, b_{\sigma^2_\alpha}), \quad \sigma^2_\beta \sim \mathrm{Gam}(a_{\sigma^2_\beta}, b_{\sigma^2_\beta}).$$

As in Chapter 5, when fitting the model we will take:

$$\begin{aligned} m_{\alpha_0} &= 140, \quad s_{\alpha_0} = 3, & m_{\beta_0} &= 15, \quad s_{\beta_0} = 7.5 \\ a_{\sigma^2} &= 1.1, \quad b_{\sigma^2} = 0.025, \\ a_{\sigma^2_\alpha} &= 1.1, \quad b_{\sigma^2_\alpha} = 0.05. \end{aligned}$$

For the constants $a_{\sigma^2_\beta}$ and $b_{\sigma^2_\beta}$ in the extra hyperprior for σ^2_β we will take:

$$a_{\sigma^2_\beta} = 1.1, \quad b_{\sigma^2_\beta} = 0.1.$$

- Extend the Stan programme for the random intercept model to additionally include a random slope.

```
// Random slope model, saved in oxboys_ranslope.stan
data {
  int<lower=1> N_subj;          // Number of subjects
  int<lower=1> N;               // Sample size
  int<lower=1> subj_label[N];   // Subject label
  vector[N] x;                  // Covariate (age)
  vector[N] y;                  // Response variable (height)
  real m_alpha0;                // Prior mean for alpha0
  real<lower=0> s_alpha0;       // Prior std. dev. for alpha0
  real m_beta0;                 // Prior mean for beta0
  real<lower=0> s_beta0;        // Prior std. dev. for beta0
  real<lower=0> a_sigsq;        // Prior shape for sigsq
  real<lower=0> b_sigsq;        // Prior rate for sigsq
  real<lower=0> a_sigsq_alpha1; // Prior shape for sigsq_alpha1
  real<lower=0> b_sigsq_alpha1; // Prior rate for sigsq_alpha1
  real<lower=0> a_sigsq_beta1;  // Prior shape for sigsq_beta1
  real<lower=0> b_sigsq_beta1;  // Prior rate for sigsq_beta1
}
parameters {
  real alpha0;
  vector[N_subj] alpha1;
  real beta0;
  vector[N_subj] beta1;
  real<lower=0> sigsq;
  real<lower=0> sigsq_alpha1;
  real<lower=0> sigsq_beta1;
}
model {
  // Likelihood:
  vector[N] alpha = alpha0 + alpha1[subj_label]; // Random intercept
  vector[N] beta = beta0 + beta1[subj_label];    // Random slope
```

```
  y ~ normal(alpha + beta .* x, sqrt(sigsq));
  // Prior:
  alpha0 ~ normal(m_alpha0, s_alpha0);
  beta0 ~ normal(m_beta0, s_beta0);
  sigsq ~ gamma(a_sigsq, b_sigsq);
  alpha1 ~ normal(0, sqrt(sigsq_alpha1));
  beta1 ~ normal(0, sqrt(sigsq_beta1));
  // "Hyper-prior":
  sigsq_alpha1 ~ gamma(a_sigsq_alpha1, b_sigsq_alpha1);
  sigsq_beta1 ~ gamma(a_sigsq_beta1, b_sigsq_beta1);
}
```

- Create a suitable data representation in R then compile and run the Stan programme.

```
## Set up data to pass to stan function in its "data" argument:
y = Oxboys$height                       # Response var. (height)
x = Oxboys$age                          # Covariate (age)
subj_label = as.numeric(Oxboys$Subject) # Subject label
N_subj = length(unique(subj_label))     # Number of subjects

## Data for random intercept model (from Chapter 5):
oxboys_ri_data = list(y=y, N=length(y), N_subj=N_subj,
             x=x, subj_label=subj_label,
             m_alpha0=140, s_alpha0=3, m_beta0=15, s_beta0=7.5,
             a_sigsq=1.1, b_sigsq=0.025,
             a_sigsq_alpha1=1.1, b_sigsq_alpha1=0.05)

## Data for random slope model:
oxboys_rs_data = oxboys_ri_data
oxboys_rs_data[["a_sigsq_beta1"]] = 1.1
oxboys_rs_data[["b_sigsq_beta1"]] = 0.1

## Compile and run Stan programme:
chains = 4
iter = 20000
thin = 10
oxboys_rs_stan = stan("oxboys_ranslope.stan", data=oxboys_rs_data,
                      chains=chains, iter=iter, warmup=iter/2,
                      thin=thin)
```

- Check both numerical and graphical diagnostics.

```
## Numerical diagnostics (here omitting the output for the
## random effects in alpha1 and beta1 to save space):
print(oxboys_rs_stan, pars=c("alpha1", "beta1"), include=FALSE)

## Inference for Stan model: oxboys_ranslope.
## 4 chains, each with iter=20000; warmup=10000; thin=10;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##                 mean se_mean    sd    2.5%     25%     50%     75%   97.5%
## alpha0        147.44    0.04  1.42  144.51  146.52  147.46  148.41  150.15
## beta0           6.56    0.01  0.36    5.89    6.32    6.56    6.80    7.29
## sigsq           0.44    0.00  0.05    0.36    0.41    0.44    0.47    0.54
## sigsq_alpha1   62.46    0.27 15.77   38.31   51.37   60.51   71.12   98.04
```

```
## sigsq_beta1     3.27    0.02  1.06    1.78    2.53    3.07    3.81    5.84
## lp__         -118.66    0.10  6.20 -132.18 -122.55 -118.16 -114.21 -108.02
##              n_eff Rhat
## alpha0        1262    1
## beta0         2125    1
## sigsq         4000    1
## sigsq_alpha1  3526    1
## sigsq_beta1   3700    1
## lp__          3991    1
##
## Samples were drawn using NUTS(diag_e) at Fri Jan 17 17:47:32 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).

# Graphical diagnostics:
oxboys_rs_stan_arr = as.array(oxboys_rs_stan)
diagnostics(oxboys_rs_stan_arr)

## NULL

## Rhat values all close to 1, n_eff large, graphical
## diagnostics give no cause for concern. However, note
## that this required a larger sample than the default
## of iter=2000.
```

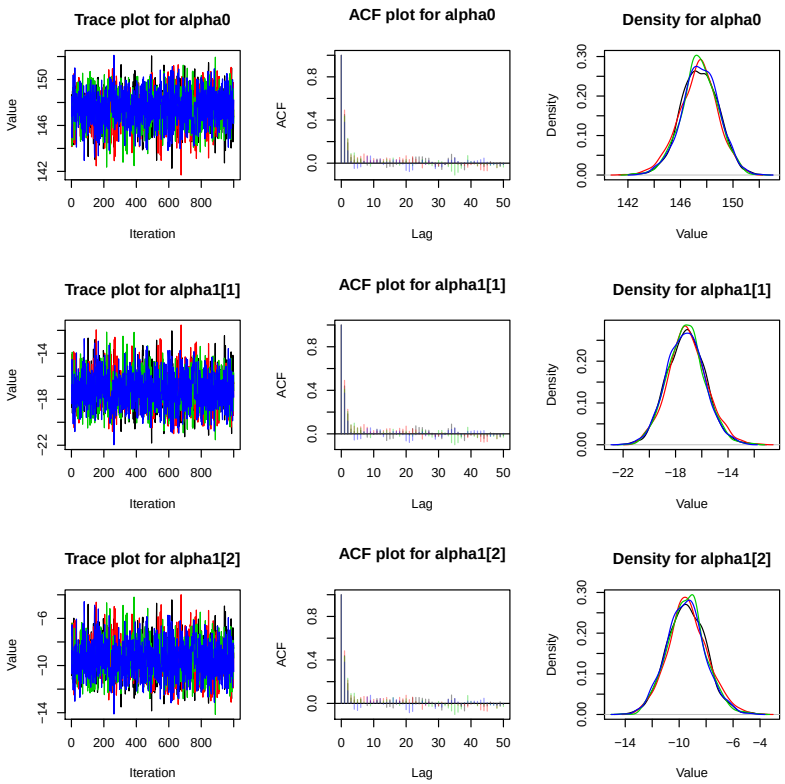

Figure 2: First page of graphical diagnostics for random slope model.

- **More difficult:** Modify the Stan programmes for the random intercept model and random slope model to include a `generated`

`quantities` block which computes the deviance of the model. (This will be similar to the code for the linear regression model in Chapter 3). Compute the DIC for each model and use it to decide which model is better according to this criterion.

```
// Random intercept model, saved in oxboys_ranintercept_dev.stan
functions {
  real deviance(vector y, vector x, int[] subj_label,
                real alpha0, vector alpha1, real beta0,
                real sigsq) {
    vector[num_elements(y)] alpha = alpha0 + alpha1[subj_label]; /*
                                            Random intercept */
    vector[num_elements(y)] eta = alpha + beta0 * x; /* Single
                                       matrix-vector calculation */
    real dev = (-2) * normal_lpdf(y | eta, sqrt(sigsq)); /*
                                  Vectorized form of the normal
                                  probability function */
    return dev;
  }
}
// ...
// Stan programme for random intercept model from Chapter 5
// ...
generated quantities {
  // Deviance:
  real dev = deviance(y, x, subj_label, alpha0, alpha1, beta0, sigsq);
}
```

```
// Random slope model, saved in oxboys_ranslope_dev.stan
functions {
  real deviance(vector y, vector x, int[] subj_label,
                real alpha0, vector alpha1, real beta0,
                vector beta1, real sigsq) {
    vector[num_elements(y)] alpha = alpha0 + alpha1[subj_label]; /*
                                            Random intercept */
    vector[num_elements(y)] beta = beta0 + beta1[subj_label]; /*
                                             Random slope */
    vector[num_elements(y)] eta = alpha + beta .* x; /* Single
                                    matrix-vector calculation */
    real dev = (-2) * normal_lpdf(y | eta, sqrt(sigsq)); /*
                                  Vectorized form of the normal
                                  probability function */
    return dev;
  }
}
// ...
// Stan programme for random slope model above
// ...
generated quantities {
  // Deviance:
  real dev = deviance(y, x, subj_label, alpha0, alpha1, beta0,
                      beta1, sigsq);
}
```

```
## Compile and run the random intercept programme:
oxboys_ri_stan = stan("oxboys_ranintercept_dev.stan", data=oxboys_ri_data,
```

```
                    chains=chains, iter=iter, warmup=iter/2,
                    thin=thin)
## Compile and run the random slope programme:
oxboys_rs_stan = stan("oxboys_ranslope_dev.stan", data=oxboys_rs_data,
                    chains=chains, iter=iter, warmup=iter/2,
                    thin=thin)
```

```
## Compute the DIC for the random intercept model:
output = as.array(oxboys_ri_stan)
(D_bar = mean(output[,,"dev"])) # Posterior expectation of deviance
```

```
## [1] 792.1522
```

```
alpha0_bar = mean(output[,,"alpha0"])
alpha1_bar = apply(output[,,paste("alpha1[",1:26,"]", sep="")],
                   3, mean)
beta0_bar = mean(output[,,"beta0"])
sigsq_bar = mean(output[,,"sigsq"]) # Posterior mean of params
expose_stan_functions(oxboys_ri_stan) # Export deviance function
D_theta_bar = deviance(oxboys_ri_data$y, oxboys_ri_data$x,
                       oxboys_ri_data$subj_label, alpha0_bar,
                       alpha1_bar, beta0_bar, sigsq_bar)
                       # Deviance at posterior mean
(pD = D_bar - D_theta_bar) # Effective number of parameters
```

```
## [1] 27.75813
```

```
(DIC = D_bar + pD) # DIC
```

```
## [1] 819.9104
```

```
## Compute the DIC for the random slope model:
output = as.array(oxboys_rs_stan)
(D_bar = mean(output[,,"dev"])) # Posterior expectation of deviance
```

```
## [1] 471.5893
```

```
alpha0_bar = mean(output[,,"alpha0"])
alpha1_bar = apply(output[,,paste("alpha1[",1:26,"]", sep="")],
                   3, mean)
beta0_bar = mean(output[,,"beta0"])
beta1_bar = apply(output[,,paste("beta1[",1:26,"]", sep="")],
                   3, mean)
sigsq_bar = mean(output[,,"sigsq"]) # Posterior mean of params
expose_stan_functions(oxboys_rs_stan) # Export deviance function
D_theta_bar = deviance(oxboys_rs_data$y, oxboys_rs_data$x,
                       oxboys_rs_data$subj_label, alpha0_bar,
                       alpha1_bar, beta0_bar, beta1_bar,
                       sigsq_bar)
                       # Deviance at posterior mean
(pD = D_bar - D_theta_bar) # Effective number of parameters
```

```
## [1] 51.90215
```

```
(DIC = D_bar + pD) # DIC
```

```
## [1] 523.4915
```

```
## The random slope model has the smaller DIC and so is
## to be preferred.
```

3 *Hierarchical model for rat tumour data*

In this section we will consider a well-known data set that is amenable to Bayesian hierarchical modelling. The data are available from the `jrRstan` package and concern the proportion of rats developing tumours in a number of laboratory studies. We can load the data via:

```
data(rats)
head(rats)

##   y  n
## 1 0 20
## 2 0 20
## 3 0 20
## 4 0 20
## 5 0 20
## 6 0 20
```

Our goal is to learn about the population probability of tumour amongst rats.

For study i, $i = 1, \ldots, N$, n_i denotes the total number of rats and Y_i denotes the number of rats who developed a tumour. To allow for the differences between studies in rats and experimental conditions, we let the probability of developing a tumour vary between studies and denote it by p_i for study i. We then assume the Y_i are independent binomial random variables given the study-specific probabilities p_i:

$$Y_i | n_i, p_i \sim \text{Bin}(n_i, p_i), \quad \text{independently for } i = 1, 2, \ldots, N,$$

where here $N = 71$. Suppose we initially thought the probabilities p_i might be around 0.1. If we were to learn that one probability p_i was larger than 0.1, this would cause us to revise upwards our "best guess" at the probabilities for other studies because we expect the p_i to be similar. To formalise this idea, we will assume the p_i are samples from some population distribution with common mean and variance. This is an example of a *Bayesian hierarchical model*. For convenience, we will reparameterise the binomial distributions so that instead of working with probabilities p_i we work with their logit transformations:

$$\theta_i = \log\left(\frac{p_i}{1 - p_i}\right)$$

and assume that

$$\theta_i | \mu, \sigma^2 \sim \text{N}(\mu, \sigma^2).$$

We allow the parameters μ and σ^2 to be unknown and give them prior distributions:

$$\mu \sim \text{N}(m_\mu, s_\mu^2) \quad \text{and} \quad \sigma^2 \sim \text{Gam}(a_{\sigma^2}, b_{\sigma^2}).$$

The mean μ can be thought of as the population value for the logit-probability of tumour amongst rats. The variance σ^2 quantifies the degree of variation in the population; the larger its value, the more heterogeneity we see in the probabilities between studies.

- Write a Stan model to represent the hierarchical model above.

```
// Hierarchical model for rat tumour data, saved in rats.stan
data {
  int<lower=0> N;        // No. studies (sample size)
  int<lower=0> y[N];     // No. of rats with tumours
  int<lower=0> n[N];     // Total number of rats
  real m_mu;             // Prior mean for mu
  real<lower=0> s_mu;    // Prior std. dev. for mu
  real<lower=0> a_sigsq; // Prior shape for sigsq
  real<lower=0> b_sigsq; // Prior rate for sigsq
}
parameters {
  vector[N] theta;
  real mu;
  real<lower=0> sigsq;
}
model {
  // Likelihood:
  y ~ binomial_logit(n, theta);
  // Prior:
  theta ~ normal(mu, sqrt(sigsq));
  // "Hyperprior":
  mu ~ normal(m_mu, s_mu);
  sigsq ~ gamma(a_sigsq, b_sigsq);
}
generated quantities {
  vector<lower=0, upper=1>[N] p = inv_logit(theta); /*
                    Study-specific prob. of tumour */
  real<lower=0, upper=1> p_pop = inv_logit(mu); /*
           "Population" probability of tumour */
}
```

- Take

$$m_\mu = 0, \quad s_\mu^2 = 2, \quad a_{\sigma^2} = 1.1, \quad b_{\sigma^2} = 1.3.$$

Create a suitable data representation in R then compile and run the Stan programme.

```
## Set up data to pass to stan function in its "data" argument:
rats_data = list(N=nrow(rats), y=rats$y, n=rats$n,
                 m_mu=0, s_mu=sqrt(2), a_sigsq=1.1, b_sigsq=1.3)

## Compile and run Stan programme:
chains = 4
iter = 2000
thin = 1
rats_stan = stan("rats.stan", data=rats_data,
                 chains=chains, iter=iter, warmup=iter/2,
                 thin=thin)

rats_reparam_stan = stan("rats_reparam.stan", data=rats_data,
                         chains=chains, iter=iter, warmup=iter/2,
                         thin=thin)
```

- Check both numerical and graphical diagnostics.

```
## Numerical diagnostics (here just printing a couple to save
## space):
print(rats_stan, pars=c("mu", "sigsq", "p_pop"))

## Inference for Stan model: rats.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##        mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## mu    -1.94    0.00 0.13 -2.20 -2.02 -1.93 -1.85 -1.71  1765 1.00
## sigsq  0.51    0.01 0.19  0.22  0.38  0.48  0.62  0.93   892 1.01
## p_pop  0.13    0.00 0.01  0.10  0.12  0.13  0.14  0.15  1786 1.00
##
## Samples were drawn using NUTS(diag_e) at Fri Jan 17 17:49:36 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).

## Graphical diagnostics
rats_stan_arr = as.array(rats_stan)
diagnostics(rats_stan_arr)

## NULL

## Rhat values all close to 1, n_eff large, graphical
## diagnostics give no cause for concern.
```

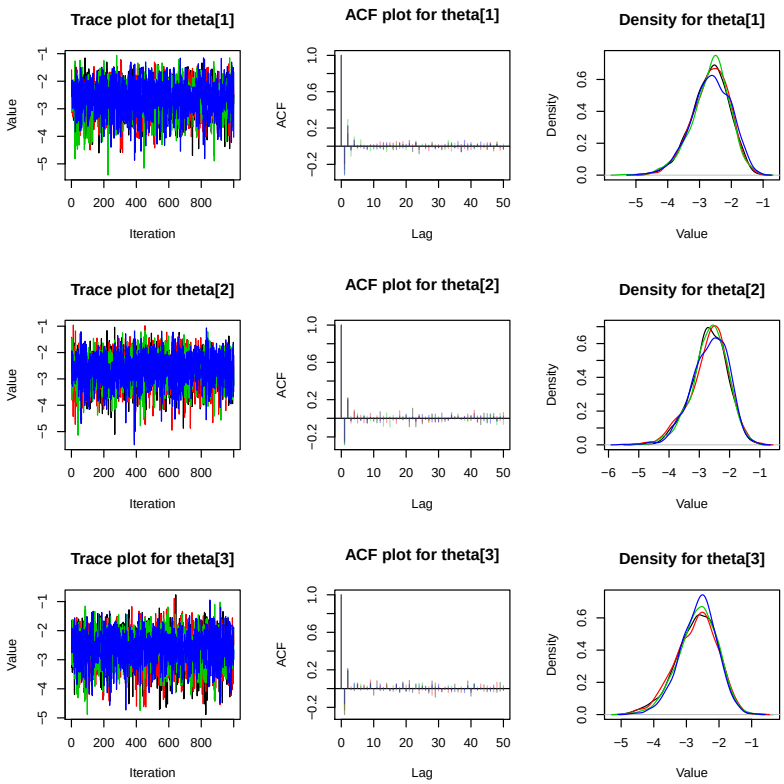

Figure 3: First page of the graphical diagnostics for the rat tumour example.

```
/* When using HMC to sample the posterior of hierarchical
   models, it can sometimes improve mixing if we adopt a
   different parameterisation that "decouples" the model
   and top level prior specifications. For the rats example,
   the Stan programme based on the decoupled parameterisation
   is shown below. See Section 15.5 in Gelman et al. (2013)
   for further details. */
data {
  int<lower=0> N;        // No. studies (sample size)
  int<lower=0> y[N];     // No. of rats with tumours
  int<lower=0> n[N];     // Total number of rats
  real m_mu;             // Prior mean for mu
  real<lower=0> s_mu;    // Prior std. dev. for mu
  real<lower=0> a_sigsq; // Prior shape for sigsq
  real<lower=0> b_sigsq; // Prior rate for sigsq
}
parameters {
  vector[N] phi; // New parameters
  real mu;
  real<lower=0> sigsq;
}
model {
  // Likelihood:
  y ~ binomial_logit(n, mu + sqrt(sigsq) * phi);
  // Prior:
  phi ~ normal(0, 1);
  // "Hyperprior":
  mu ~ normal(m_mu, s_mu);
  sigsq ~ gamma(a_sigsq, b_sigsq);
}
generated quantities {
  vector[N] theta = mu + sqrt(sigsq) * phi; /*
                        Original parameters */
  vector<lower=0, upper=1>[N] p = inv_logit(theta); /*
                     Study-specific prob. of tumour */
  real<lower=0, upper=1> p_pop = inv_logit(mu); /*
                 "Population" prob. of tumour */
}
```

```
## In this example, the decoupled parameterisation offers
## a marginal improvement in mixing ...
rats_reparam_stan = stan("rats_reparam.stan", data=rats_data,
                         chains=chains, iter=iter, warmup=iter/2,
                         thin=thin)
```

```
## ... as illustrated by, e.g.
print(rats_reparam_stan, pars=c("mu", "sigsq", "p_pop"))

## Inference for Stan model: rats_reparam.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##        mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## mu    -1.94    0.00 0.12 -2.20 -2.02 -1.93 -1.86 -1.72  2434 1.00
## sigsq  0.50    0.01 0.19  0.22  0.37  0.48  0.61  0.93   993 1.01
```

```
## p_pop  0.13    0.00 0.01  0.10  0.12  0.13  0.14  0.15  2493 1.00
##
## Samples were drawn using NUTS(diag_e) at Fri Jan 17 17:49:46 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).
```

4 *Three state mixture model**

In Section 5.4 we fitted a mixture model with $K = 2$ components to the `faithful` data on the times between successive eruptions of the Old Faithful geyser:

```
data(faithful)
head(faithful)

##   eruptions waiting
## 1     3.600      79
## 2     1.800      54
## 3     3.333      74
## 4     2.283      62
## 5     4.533      85
## 6     2.883      55
```

In the two-component case, we had component membership probabilities (π_1, π_2) which summed to one and so we reparameterised the model as $\pi_1 = \pi$ and $\pi_2 = 1 - \pi$ where $0 \leq \pi \leq 1$. We represented this in Stan as a constrained real:

```
real<lower=0,upper=1> pi;
```

In the more general case with K components we have vector $(\pi_1, \pi_2, \ldots, \pi_K)$ on the K-dimensional simplex. This can be represented in Stan as a `simplex` vector type:

```
simplex[K] pi_vec;
```

In the case of two-components, we assigned `pi` a symmetric Beta prior:

```
// Prior:
pi ~ beta(a_pi, a_pi);
```

The multivariate generalisation of this is a symmetric Dirichlet prior.

- Use the `lookup` function to find the Dirichlet distribution in the Stan manual:

```
lookup("Dirichlet") # Simply search for string rather than

##       StanFunction                     Arguments ReturnType Page
## 119 dirichlet_lpdf (vector theta | vector alpha)       real  537
## 120      dirichlet                             ~       real  537
## 121  dirichlet_rng                (vector alpha)     vector  537

                    # the name of a particular R function.
```

Try to generalise the Stan programme from Figure 5.7 so that it allows the number K of states to be a component of the `list` you pass to the `stan` function through its `data` argument.

```
/* General K component mixture of normals, saved in
   oldfaithful_gen.stan */
data {
  int<lower=0> N;          // Sample size
  int<lower=1> K;          // Number of components
  vector[N] y;             // Data
  real<lower=0> a_pi;      /* Repeated prior hyperparameter
                              for pi_vec */
  real m_mu;               // Prior mean for mu[k]
  real<lower=0> s_mu;      // Prior std. dev. for mu[k]
  real<lower=0> a_siqsq; // Prior shape for sigsq[k]
  real<lower=0> b_sigsq; // Prior rate for sigsq[k]
}
transformed data {
  vector<lower=0>[K] a_pi_vec;
  for(k in 1:K) a_pi_vec[k] = a_pi;
}
parameters {
  simplex[K] pi_vec;
  ordered[K] mu;
  vector<lower=0>[K] sigsq;
}
model {
  vector[K] tmp;
  // Likelihood:
  for(n in 1:N) {
    for(k in 1:K) {
      tmp[k] = log(pi_vec[k]) + normal_lpdf(y[n] | mu[k], sqrt(sigsq[k]));
    }
    target += log_sum_exp(tmp);
  }
  // Prior:
  pi_vec ~ dirichlet(a_pi_vec);
  mu ~ normal(m_mu, s_mu);
  sigsq ~ inv_gamma(a_siqsq, b_sigsq);
}
```

- Test the programme by compiling it then running it with $K = 2$ and $K = 3$ components.

```
## Set up data to pass to stan function in its "data" argument:
faithful_2_data = list(N=nrow(faithful), y=log(faithful$waiting),
                       K=2, a_pi=3, m_mu=4, s_mu=1,
                       a_siqsq=4, b_sigsq=0.04)

faithful_3_data = faithful_2_data
faithful_3_data[["K"]] = 3

faithful_3_data[["s_mu"]] = 0.1 # (Try with s_mu=1 and see what
                                #  happens ... )

## Compile and run Stan programme:
```

```
chains = 4
iter = 2000
thin = 1
faithful_2_stan = stan("oldfaithful_gen.stan", data=faithful_2_data,
                       chains=chains, iter=iter, warmup=iter/2,
                       thin=thin)

iter = 20000
thin = 10
faithful_3_stan = stan("oldfaithful_gen.stan", data=faithful_3_data,
                       chains=chains, iter=iter, warmup=iter/2,
                       thin=thin)
```

```
## Numerical diagnostics for two component model:
print(faithful_2_stan)

## Inference for Stan model: oldfaithful_gen.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##             mean se_mean   sd   2.5%    25%    50%    75%  97.5% n_eff
## pi_vec[1]   0.38    0.00 0.03   0.32   0.35   0.38   0.40   0.44  3301
## pi_vec[2]   0.62    0.00 0.03   0.56   0.60   0.62   0.65   0.68  3301
## mu[1]       4.00    0.00 0.01   3.98   3.99   4.00   4.01   4.03  2173
## mu[2]       4.38    0.00 0.01   4.37   4.38   4.38   4.39   4.40  4000
## sigsq[1]    0.01    0.00 0.00   0.01   0.01   0.01   0.01   0.02  2531
## sigsq[2]    0.01    0.00 0.00   0.00   0.00   0.01   0.01   0.01  2861
## lp__      140.47    0.04 1.63 136.43 139.61 140.81 141.68 142.65  1685
##           Rhat
## pi_vec[1]    1
## pi_vec[2]    1
## mu[1]        1
## mu[2]        1
## sigsq[1]     1
## sigsq[2]     1
## lp__         1
##
## Samples were drawn using NUTS(diag_e) at Fri Jan 17 17:49:54 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).

## Graphical diagnostics for two component model:
faithful_2_stan_arr = as.array(faithful_2_stan)
diagnostics(faithful_2_stan_arr)

## NULL

## Rhat values all close to 1, n_eff large, graphical
## diagnostics give no cause for concern.
```

```
## Numerical diagnostics for three component model:
print(faithful_3_stan)

## Inference for Stan model: oldfaithful_gen.
```

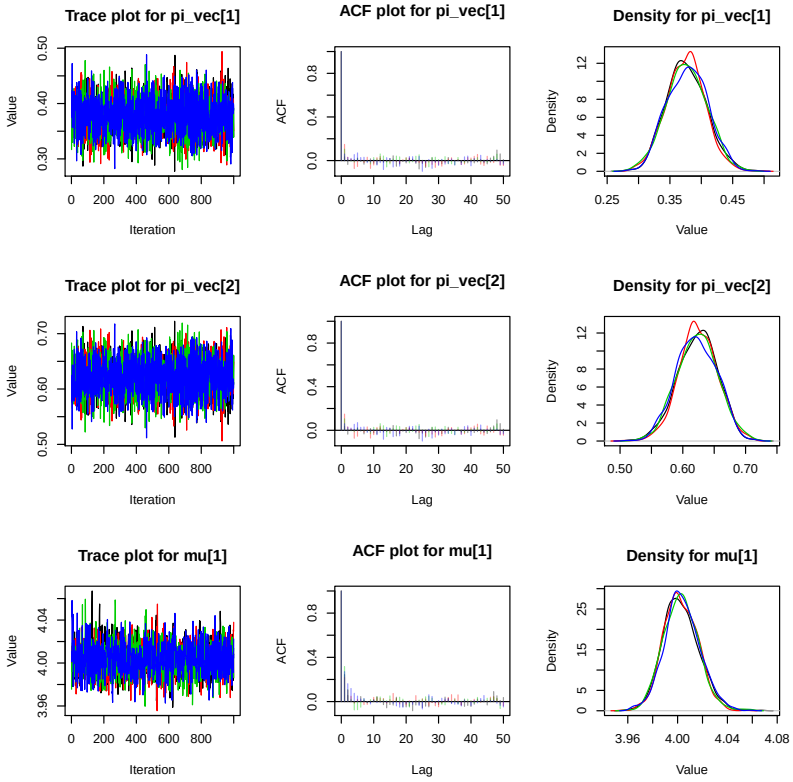

Figure 4: First page of graphical diagnostics for two component mixture.

```
## 4 chains, each with iter=20000; warmup=10000; thin=10;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##             mean se_mean   sd    2.5%     25%     50%     75%  97.5% n_eff
## pi_vec[1]   0.20    0.00 0.07    0.06    0.15    0.20    0.25    0.33  4000
## pi_vec[2]   0.19    0.00 0.07    0.06    0.14    0.19    0.24    0.33  4000
## pi_vec[3]   0.61    0.00 0.03    0.54    0.59    0.61    0.63    0.67  3942
## mu[1]       3.96    0.00 0.03    3.90    3.94    3.96    3.98    4.01  3955
## mu[2]       4.06    0.00 0.04    4.00    4.03    4.05    4.09    4.17  3568
## mu[3]       4.38    0.00 0.01    4.37    4.38    4.38    4.39    4.40  3884
## sigsq[1]    0.01    0.00 0.00    0.00    0.01    0.01    0.01    0.02  3760
## sigsq[2]    0.01    0.00 0.01    0.01    0.01    0.01    0.02    0.03  3535
## sigsq[3]    0.01    0.00 0.00    0.00    0.00    0.01    0.01    0.01  4000
## lp__      136.89    0.04 2.14 131.81 135.67 137.21 138.45 140.07  3681
##           Rhat
## pi_vec[1]    1
## pi_vec[2]    1
## pi_vec[3]    1
## mu[1]        1
## mu[2]        1
## mu[3]        1
## sigsq[1]     1
## sigsq[2]     1
## sigsq[3]     1
## lp__         1
##
## Samples were drawn using NUTS(diag_e) at Fri Jan 17 17:52:42 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).

## Graphical diagnostics for three component model:
faithful_3_stan_arr = as.array(faithful_3_stan)
diagnostics(faithful_3_stan_arr)

## NULL

## Numerical and graphical diagnostics do not look
## great, even with lots of iterations and thinning.
## Components 1 and 2 are very similar so I suspect
## there is some overfitting which is affecting the
## performance of the sampler.
```

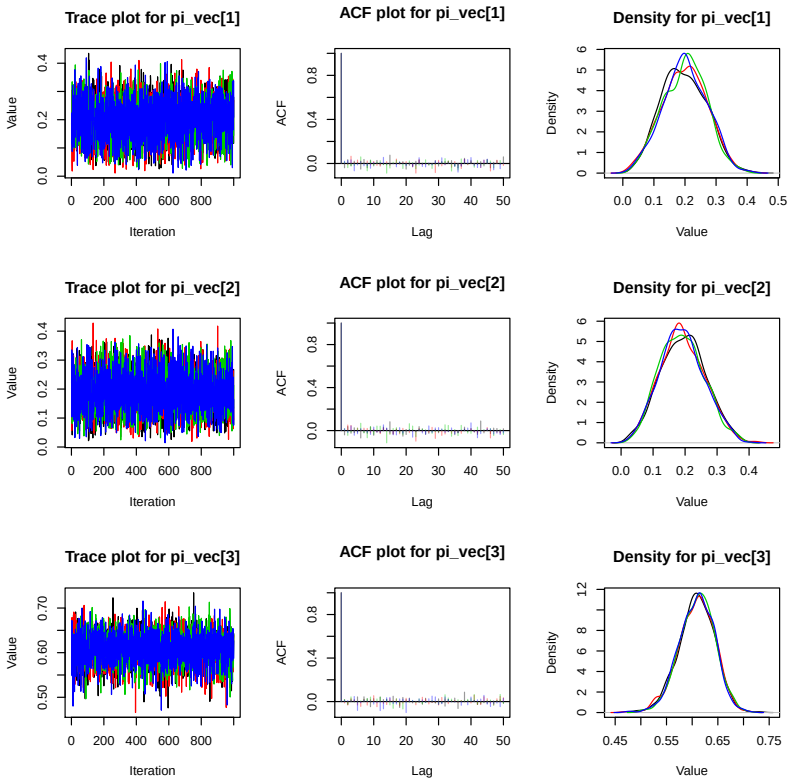

Figure 5: First page of graphical diagnostics for three component mixture.

5 *Survival with right censoring**

In this section we will consider another generalised linear model, this time assuming an *exponential* error distribution. The data we will consider concern the survival times T_n for $n = 1, \ldots, N$, in months, of $N = 148$ renal patients following kidney transplants. There is one covariate x_n for each patient, namely the total number of HLA-B or DR antigen mismatches between the donor and recipient; we might expect survival times to be shorter if there are more mismatches. For some patients, the month of death is observed. However, for

others the survival time is *right-censored* meaning we do not observe the time of death, only a time at which the patient was known still to be alive. This can happen for lots of reasons, for example, the study may have ended before the patient died or the patient may have been lost to follow-up. We introduce an indicator variable s_n representing the censoring status of the patient. We set $s_n = 1$ if the corresponding observation t_n on T_n represents the survival time of patient n. On the other hand, we set $s_n = 0$ if the observation t_n on T_n is a right-censored time, meaning all we know is that the survival time of patient n is greater than t_n, i.e. $T_n > t_n$.

The `renal` data set in the `jrRstan` package contains the survival data:

```
data(renal)
head(renal)

##       t status x
## 1 0.035      1 3
## 2 0.068      1 0
## 3 0.100      1 0
## 4 0.101      1 1
## 5 0.167      0 4
## 6 0.168      1 2
```

Our model for the survival times can be expressed as:

$$T_n|\lambda_n \sim \text{Exp}(\lambda_n), \quad \text{independently for } n = 1, 2, \ldots, N,$$

where $\lambda_n = \exp(\eta_n)$ is the reciprocal of the mean survival time for patient n. The corresponding *linear predictor* η_n takes the form:

$$\eta_n = \beta_1 + \beta_2 x_n.$$

We adopt the following prior for the two model parameters:

$$\beta_1 \sim \text{N}(0, 30^2), \quad \text{and, independently,} \quad \beta_2 \sim \text{N}(0, 30^2).$$

If all the survival times were observed, i.e. if $s_n = 1$ for all $n = 1, \ldots, N$, the likelihood would take the form:

$$\prod_{n=1}^{N} p(t_n|\lambda_n)$$

in which patient n contributes $p(t_n|\lambda_n)$ to the likelihood function, namely the density function of the $\text{Exp}(\lambda_n)$ distribution evaluated at the observed survival time $T_n = t_n$. However, if the time for patient n is right-censored our observation t_n does not represent the survival time T_n, and we only know that $T_n > t_n$. For such patients the contribution to the likelihood is therefore $\Pr(T_n > t_n|\lambda_n)$ which is the *survival function*, or *complementary cumulative distribution function*, of the $\text{Exp}(\lambda_n)$ distribution evaluated at the censored time t_n. Overall, therefore, the likelihood takes the form

$$\prod_{n:s_n=1} p(t_n|\lambda_n) \times \prod_{n:s_n=0} \Pr(T_n > t_n|\lambda_n)$$

where the first term is a product over the patients for whom we observe the survival times and the second term is a product over the patients whose survival times are right-censored. Because the Stan modelling language is very flexible, we can handle the non-standard second term in the likelihood by incrementing the log posterior density using the `target` keyword[4] and the log complementary cumulative distribution function of the exponential distribution, `exponential_lccdf`.

[4] As in the mixture model example.

- Write a Stan model to represent the model and prior above.

```
// Survival model with right-censoring, saved in renal_intout.stan
data {
  int<lower=0> N_obs;             /* No. patients with observed
                                     survival times */
  int<lower=0> N_cens;            /* No. patients with right-censored
                                     times */
  int<lower=1> J;                 // No. columns in design matrix
  vector<lower=0>[N_obs] t_obs;   // Observed survival times
  vector<lower=0>[N_cens] t_cens; // Right-censored times
  matrix[N_obs, J] X_obs;         /* Design matrix for observed
                                     patients */
  matrix[N_cens, J] X_cens;       /* Design matrix for
                                     right-censored patients */
  real beta_mean[J];              // Means in beta prior
  real<lower=0> beta_sd[J];       // Std. devs in beta prior
}
parameters {
  vector[J] beta;
}
model {
  // Likelihood for observed patients:
  t_obs ~ exponential(exp(X_obs * beta));
  // Likelihood for right-censored patients:
  target += exponential_lccdf(t_cens | exp(X_cens * beta));
  // Prior:
  beta ~ normal(beta_mean, beta_sd);
}
```

- Create a suitable data representation in R then compile and run the Stan programme.

```
## Set up data to pass to stan function in its "data" argument:
N_obs = nrow(renal[renal$status==1,])  # No. observed patients
N_cens = nrow(renal[renal$status==0,]) # No. right-censored patients
X_obs = matrix(1, N_obs, 2)            # Design matrix for observed
X_obs[,2] = renal$x[renal$status==1]   # patients
X_cens = matrix(1, N_cens, 2)          # Design matrix for
X_cens[,2] = renal$x[renal$status==0]  # right-censored patients

renal_data = list(N_obs=N_obs, N_cens=N_cens,
                  t_obs = renal$t[renal$status==1],
                  t_cens = renal$t[renal$status==0],
                  X_obs=X_obs, X_cens=X_cens, J=2,
                  beta_mean = c(0, 0), beta_sd = c(30, 30))
```

```
## Compile and run Stan programme:
chains = 4
iter = 10000
thin = 5
renal_stan = stan("renal_intout.stan", data=renal_data,
                  chains=chains, iter=iter, warmup=iter/2,
                  thin=thin)
```

- Check both numerical and graphical diagnostics.

```
## Numerical diagnostics:
print(renal_stan)

## Inference for Stan model: renal_intout.
## 4 chains, each with iter=10000; warmup=5000; thin=5;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##            mean se_mean   sd    2.5%     25%     50%     75%   97.5% n_eff
## beta[1]   -5.98    0.01 0.40   -6.80   -6.24   -5.95   -5.70   -5.25  3256
## beta[2]    0.98    0.00 0.17    0.65    0.87    0.98    1.09    1.31  3304
## lp__    -152.49    0.02 1.04 -155.32 -152.90 -152.18 -151.75 -151.48  3399
##         Rhat
## beta[1]    1
## beta[2]    1
## lp__       1
##
## Samples were drawn using NUTS(diag_e) at Fri Jan 17 17:52:46 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).

## Graphical diagnostics:
renal_stan_arr = as.array(renal_stan)
diagnostics(renal_stan_arr)

## NULL

## Rhat values all close to 1, n_eff large, graphical
## diagnostics give no cause for concern.
```

Solutions

Solutions are available as a vignette:

```
library("jrRstan")
vignette("solutions2", package="jrRstan")
```

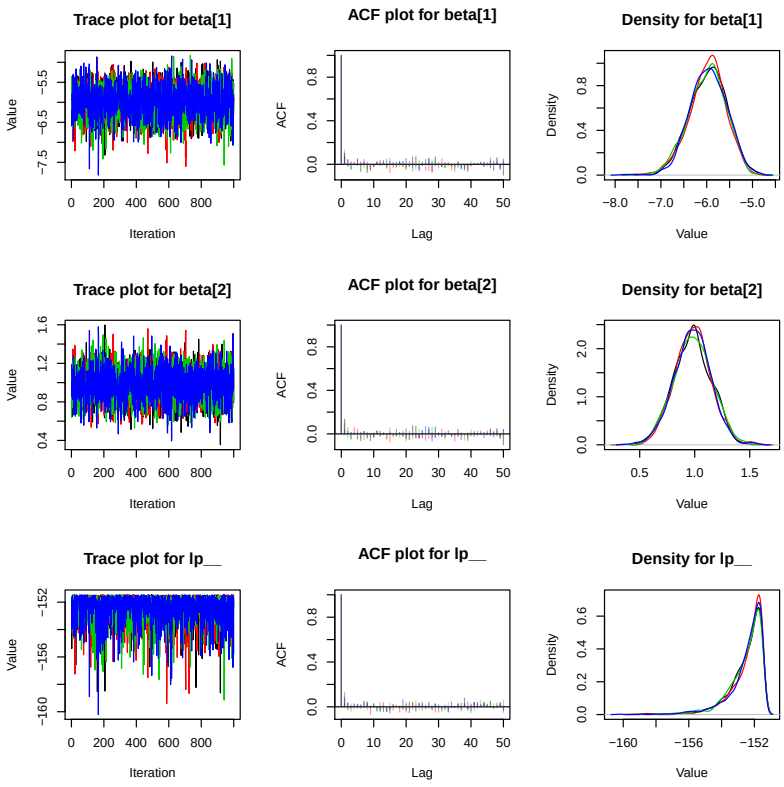

Figure 6: Graphical diagnostics for the survival model with right-censoring.