BDA - Assignment 6

Anonymous

Contents


```
# To install aaltobda, see the General information in the assignment.
library(aaltobda)
library("ggplot2")
library(rstan)
```

```
## Loading required package: StanHeaders
```

```
## rstan (Version 2.21.2, GitRev: 2e1f913d3ca3)
```

```
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
```

Generalized linear model: Bioassay with Stan

Replicate the computations for the bioassay example of section 3.7 (BDA3) using Stan.

1.

The data for the assignment.

```
data("bioassay")

bioassay_data <- list(N = dim(bioassay)[1], x = bioassay$x,
n = bioassay$n, y = bioassay$y)
```

Write down the model for the bioassay data in Stan syntax.

```
### THE MODEL FOR BIOASSAY DATA USING STAN ###

# Declare the data required to fit the model here
data {
  int<lower=1> N; // number of data points
  int<lower=0> n[N]; // number of animals
  int<lower=0, upper=n[N]> y[N]; // number of deaths
  real x[N]; // dose (log g/ml)
}
```

```
# May be used to define new variables that can be computed based on the data.
# Or in other words, preprocess the data.
transformed data {
  vector[2] mu = [ 0, 10 ]';
  cov_matrix[2] Sigma = [ [4, 10] , [10, 100] ];
}

# Define the model parameters here, defines the sampling space
parameters {
  vector[2] theta; # two-element vector so that the first value denotes alpha and latter one beta
}

# The transformed parameters block allows users to define transforms
# of parameters within a mode.
# Allows for parameter processing before posterior is computed.

# Define the log probability function on the constrained parameter space here.
# The posterior.
model {
  // priors
  theta ~ multi_normal(mu, Sigma);

  // observation model / likelihood
  y[N] ~ binomial_logit(n[N], theta[1] + theta[2]*x[N]);

}
```

Fit the model.

```
bioassay_fit <- sampling(bioassay_model, bioassay_data, seed = 14102020)
```

```
##
## SAMPLING FOR MODEL 'a44bdbc150676cfd5bfaeba25a9dc675' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 1e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.1 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.015036 seconds (Warm-up)
## Chain 1:                0.017767 seconds (Sampling)
## Chain 1:                0.032803 seconds (Total)
```

```
## Chain 1:
##
## SAMPLING FOR MODEL 'a44bdbc150676cfd5bfaeba25a9dc675' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 5e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.05 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.016269 seconds (Warm-up)
## Chain 2:                0.014531 seconds (Sampling)
## Chain 2:                0.0308 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'a44bdbc150676cfd5bfaeba25a9dc675' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 5e-06 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.05 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.01535 seconds (Warm-up)
## Chain 3:                0.016215 seconds (Sampling)
## Chain 3:                0.031565 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'a44bdbc150676cfd5bfaeba25a9dc675' NOW (CHAIN 4).
## Chain 4:
```

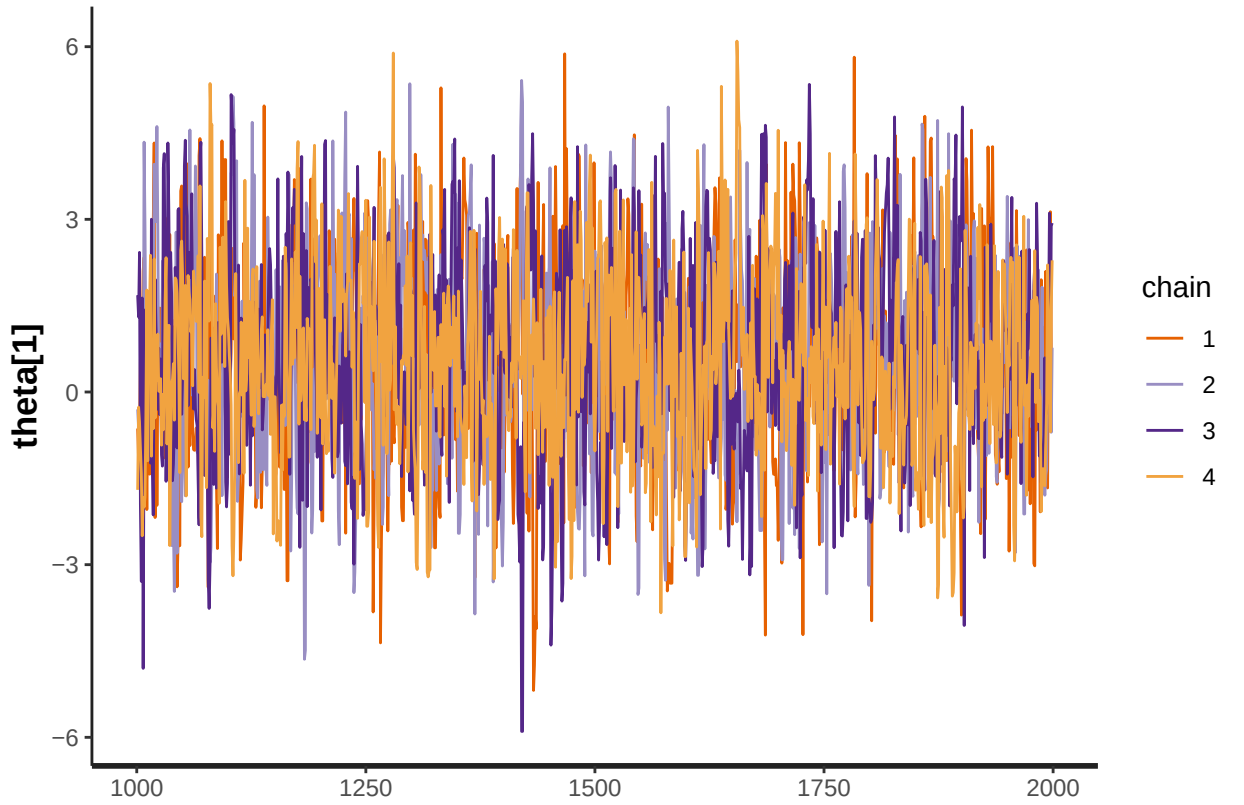
```
## Chain 4: Gradient evaluation took 5e-06 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.05 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.015499 seconds (Warm-up)
## Chain 4:                0.016182 seconds (Sampling)
## Chain 4:                0.031681 seconds (Total)
## Chain 4:
```

The initial points are set by Stan in a following manner: If there are user-supplied initial values for parameters, these are read using the same input mechanism and same file format as data reads. Any constraints declared on the parameters are validated for the initial values. If a variable's value violates its declared constraint, the program halts and a diagnostic message is printed. After being read, initial values are transformed to unconstrained values that will be used to initialize the sampler. (https://mc-stan.org/docs/2_19/reference-manual/initialization.html)

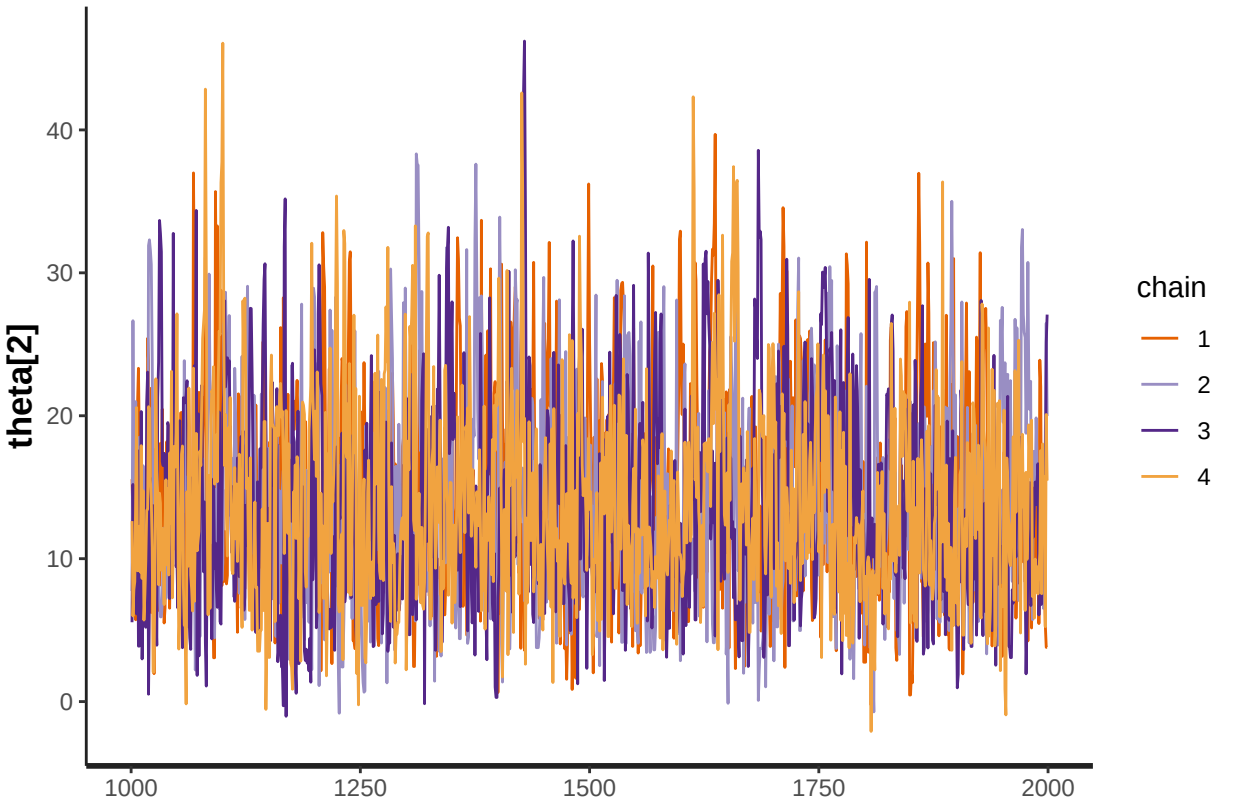
The default chain length / number of iterations that Stan uses is 2000. The default warmup length is half of that (1000). The default number of chains in Stan is 4. These default values are used in the model.

Plotting the chains to see if the model works. (Please note that "theta[1]"=α and "theta[2]"=β.)

```
# Convergence plot for alpha
traceplot(bioassay_fit, pars = "theta[1]", inc_warmup = FALSE)
```




```
# Convergence plot for beta
traceplot(bioassay_fit, pars = "theta[2]", inc_warmup = FALSE)
```



Model seems to be working nicely. Proceeding to analyze the convergence numerically.

2.

The basic idea of $\hat{R}$ is to serve as a convergence diagnostic that compares the between- and within-chain estimates for model parameters. With this estimand one can monitor convergence of the iterative simulation. If this potential scale reduction estimand gets high values, conducting further simulations can improve our inference about the target distribution.

Calling monitor(bioassay_fit) to obtain $\hat{R}$ for α and β.

```
monitor(bioassay_fit)
```

```
## Inference for the input samples (4 chains: each with iter = 2000; warmup = 0):
##
##            Q5  Q50  Q95 Mean  SD  Rhat Bulk_ESS Tail_ESS
## theta[1] -2.2  0.6  3.5  0.6 1.7     1     1617     2232
## theta[2]  3.9 13.1 27.2 14.0 7.2     1     1300     1876
## lp__     -2.7 -0.6  0.0 -0.9 0.9     1     1719     2064
##
## For each parameter, Bulk_ESS and Tail_ESS are crude measures of
## effective sample size for bulk and tail quantities respectively (an ESS > 100
## per chain is considered good), and Rhat is the potential scale reduction
## factor on rank normalized split chains (at convergence, Rhat <= 1.05).
```

As it says in R console output, at convergence $\hat{R}$ values for α and β should be smaller or equal than 1.05. In the data frame obtained from calling monitor(bioassay_fit), theta[1] equals to α and theta[2] to β. $\hat{R}$ values

for both of these equal to 1.

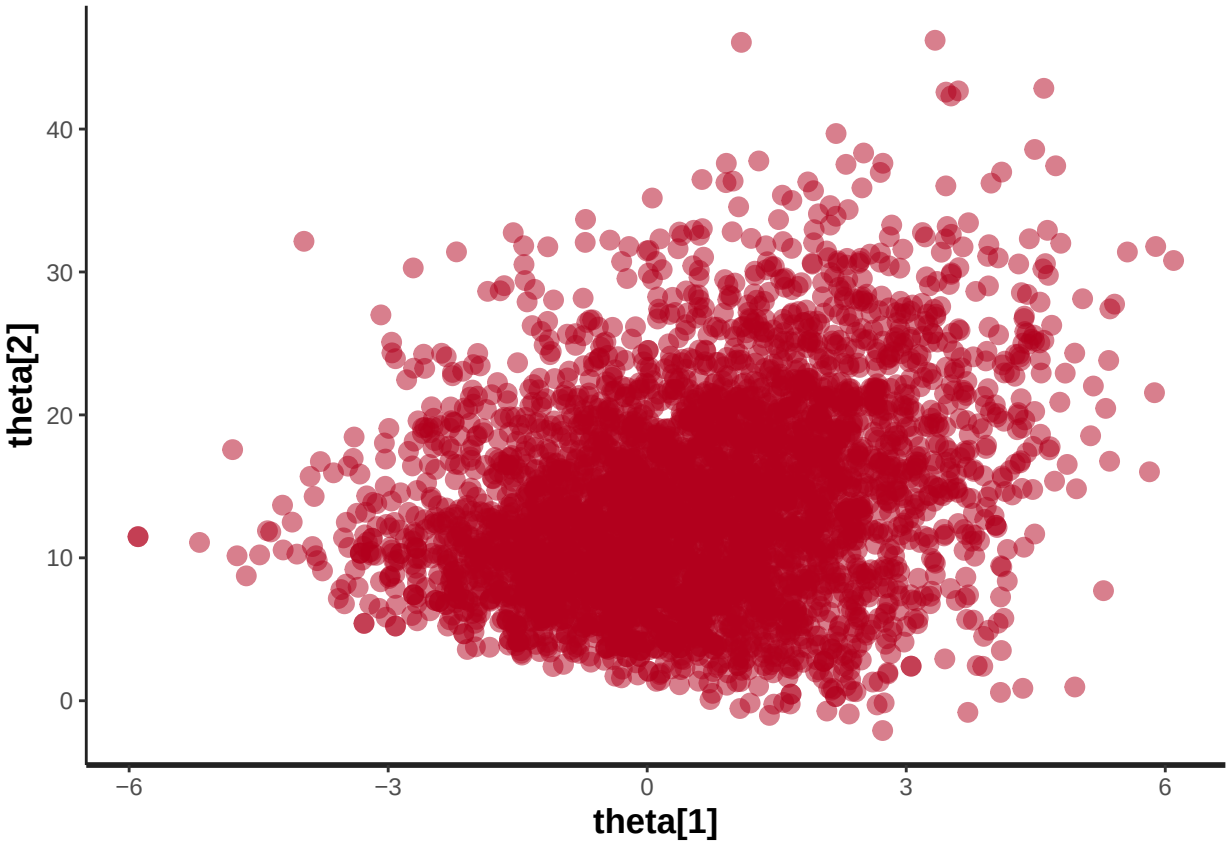
$$\hat{R}_{alpha} = 1, \hat{R}_{beta} = 1$$

This means that the chains have converged well and we do not need to proceed simulating further.

3.

Plotting the draws for α and β (scatter plot)and comparing to Figure 3.3b in BDA3.

```
stan_scat(bioassay_fit, pars = c("theta[1]", "theta[2]"))
```



The plot looks fairly similar to Figure 3.3b in BDA3.

4.

- Operating system: Linux.
- Programming environment: R.
- Interface used: RStan.

I didn't encounter any installation or compilation problems and didn't change to jupyter.cs.aalto.fi, as I did not encounter any problems when installing locally.