weib_my

Data download

```
library(dplyr)
library(ggplot2)
library("rstan")
library(loo)
library(bayesplot)
```

Data preparation

```
data("cancer", package = "survival")
data = cancer %>% na.omit()
```

Censoring status will be transformed to the column with 0-censored, 1-observed. From binary variable "sex" dummies will be made and the continuous variables will be centered.

```
X=data
X$status[X$status==1] = 0
X$status[X$status==2] = 1
#X$male = ifelse(X$sex==1,1,0)
#X$female = ifelse(X$sex==2,1,0)
X$age=X$age-mean(X$age)
X$meal.cal=X$meal.cal-mean(X$meal.cal)
X$wt.loss=X$wt.loss-mean(X$wt.loss)

Xcens=X[X$status==0,]
Xcens = Xcens[-c(1,2,3)]
ycens=X$time[X$status==0]

Xobs=X[X$status==1,]
Xobs = as.matrix(Xobs[-c(1,2,3)])
yobs=X$time[X$status==1]
```

Weibull model with censored

```
weibull_model = "
data {
  int<lower=0> N;    // number of object
  int<lower=0> M;    // number of features
  int<lower=0> Ncen;    // number of object
  vector<lower=0>[N] yobs; // target-survival time
  matrix[N, M] Xobs;    // covariates
  vector<lower=0>[Ncen] ycen; // target-survival time
  matrix[Ncen, M] Xcen;    // covariates
}

parameters {
  vector[M] beta;       // regressors
  real<lower=0> alpha;  // shape parameter
}

transformed parameters {
  // Log inverse link function
  vector<lower=0>[N] sigma = exp(-Xobs*beta / alpha);

}

model {
  // priors
  beta ~ normal(0, 10);
  alpha ~ gamma(1, 1);

  // fitting model
  yobs ~ weibull(alpha, sigma);

  // Increment log-density with Survival Function
  target += weibull_lccdf(ycen | alpha, exp(-Xcen*beta / alpha));
}

generated quantities {
  // compute predictive distribution for survival time
  real ypred[N] = weibull_rng(alpha, sigma);

  // log-likelihood
  vector[N+Ncen] log_lik;
  for (i in 1:N) {
    log_lik[i] = weibull_lpdf(yobs[i] | alpha, sigma[i]);
  }
  // Survival function
  for (j in 1:Ncen){
    log_lik[N+j] = weibull_lccdf(ycen[j] | alpha, exp(-Xcen[j,]*beta / alpha));
  }
}
"
```

```
data_model = list(
  yobs = yobs,
  Xobs = Xobs,
  N = nrow(Xobs),
  M = ncol(Xobs),
  ycen = ycens,
  Xcen = Xcens,
  Ncen = nrow(Xcens)
)
```

```
sampling = stan(model_code=weibull_model,
                data=data_model,
                iter=5000
)
```

Checking convergence

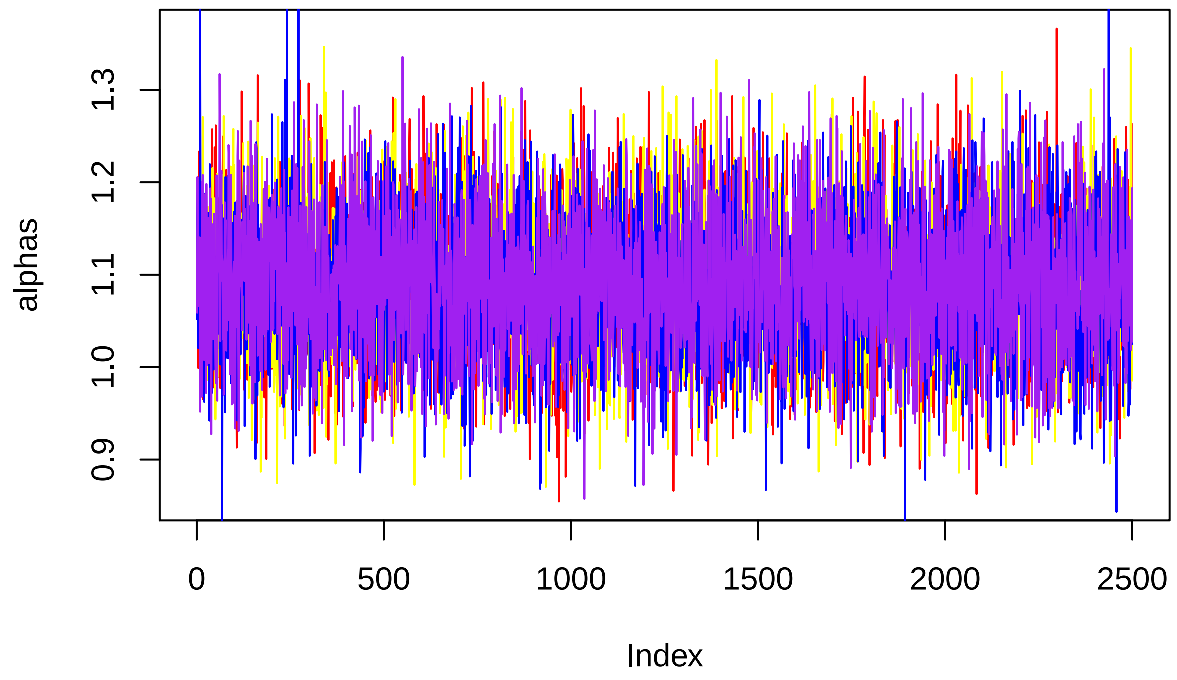
Rhat values are lower than 1.05. It means that the number of iterations is enough and samples converged.

```
sampling_monit=monitor(sampling)
```

```
betas=extract(sampling)$beta
alphas=extract(sampling)$alpha
```

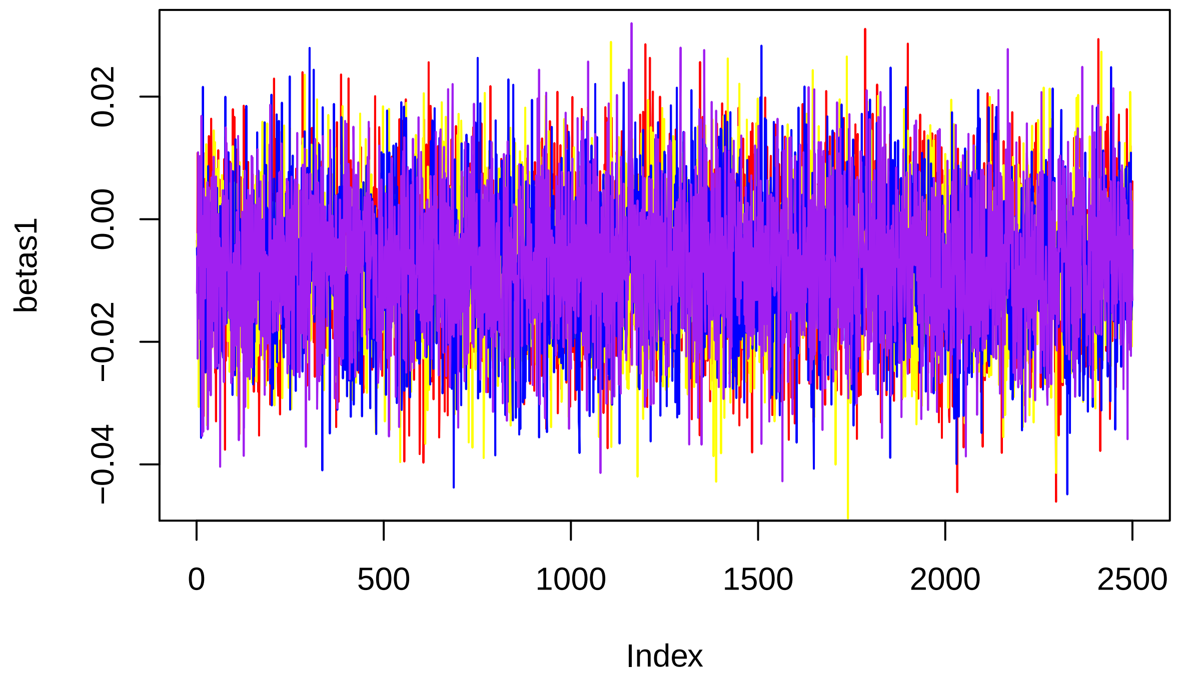
All chains for α in a single line-plot (does not include warm-up).

```
plot(alphas[1:2500],type="l",col="red",ylab="alphas")
lines(alphas[2501:5000],col="yellow")
lines(alphas[5001:7500],col="blue")
lines(alphas[7501:10000],col="purple")
```



All chains for, for instance, β_1 in a single line-plot (does not include warm-up).

```
plot(betas[,1][1:2500],type="l",col="red", ylab="betas1")
lines(betas[,1][2501:5000],col="yellow")
lines(betas[,1][5001:7500],col="blue")
lines(betas[,1][7501:10000],col="purple")
```

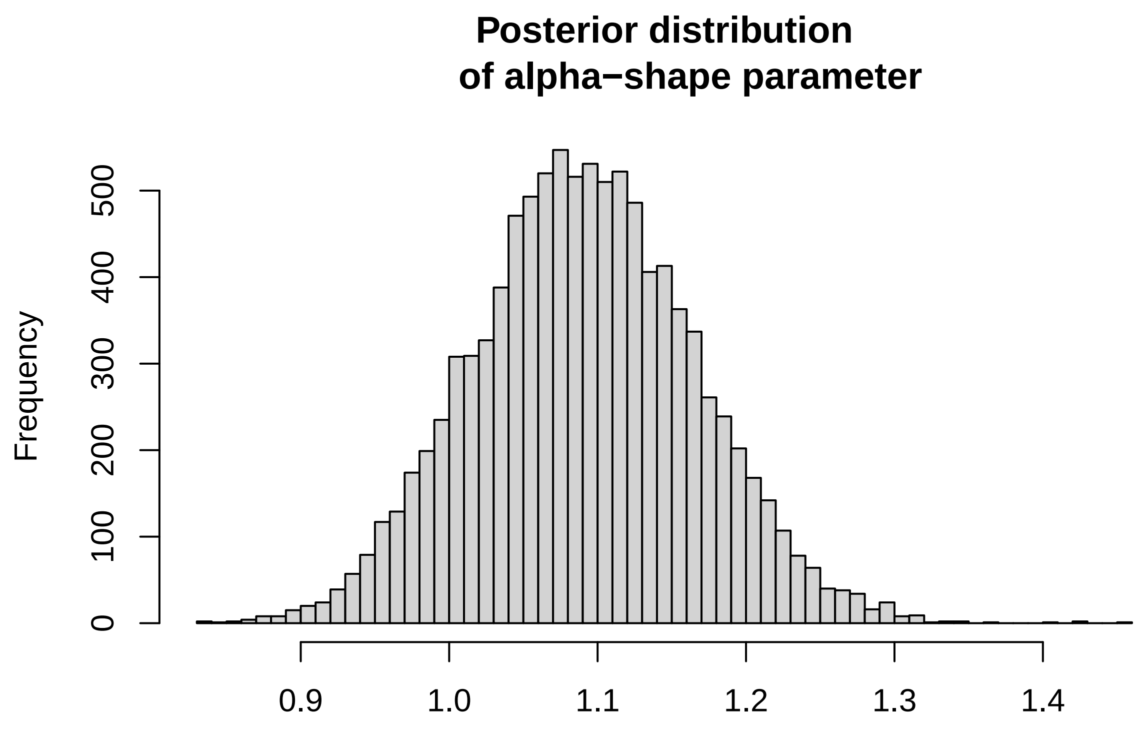


We can see that the sequences are mixed (they are not far away from each other), they are not separated from each other, and two variance components (the variance within each sequence and the variance between sequences) don't differ much. The chains have converged.

Model distributions and characteristics

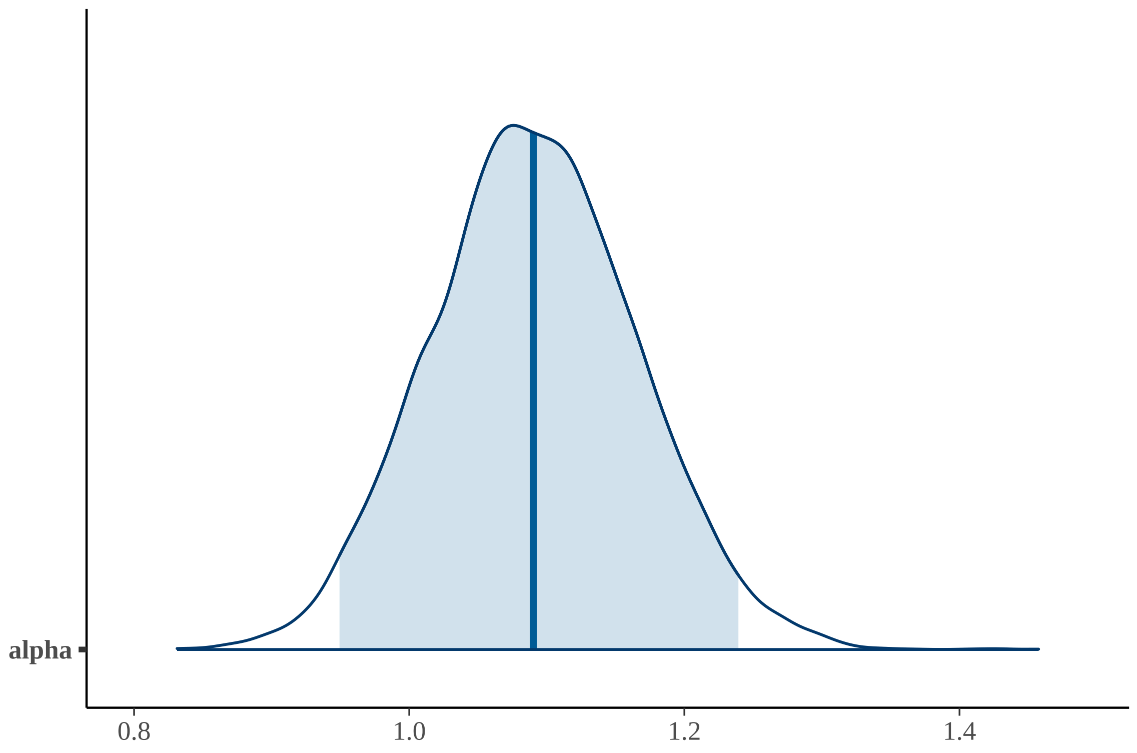
Posterior distribution of alpha-shape parameter

```r
hist(extract(sampling)$alpha, main="Posterior distribution
     of alpha-shape parameter", xlab="",breaks = 50)
```



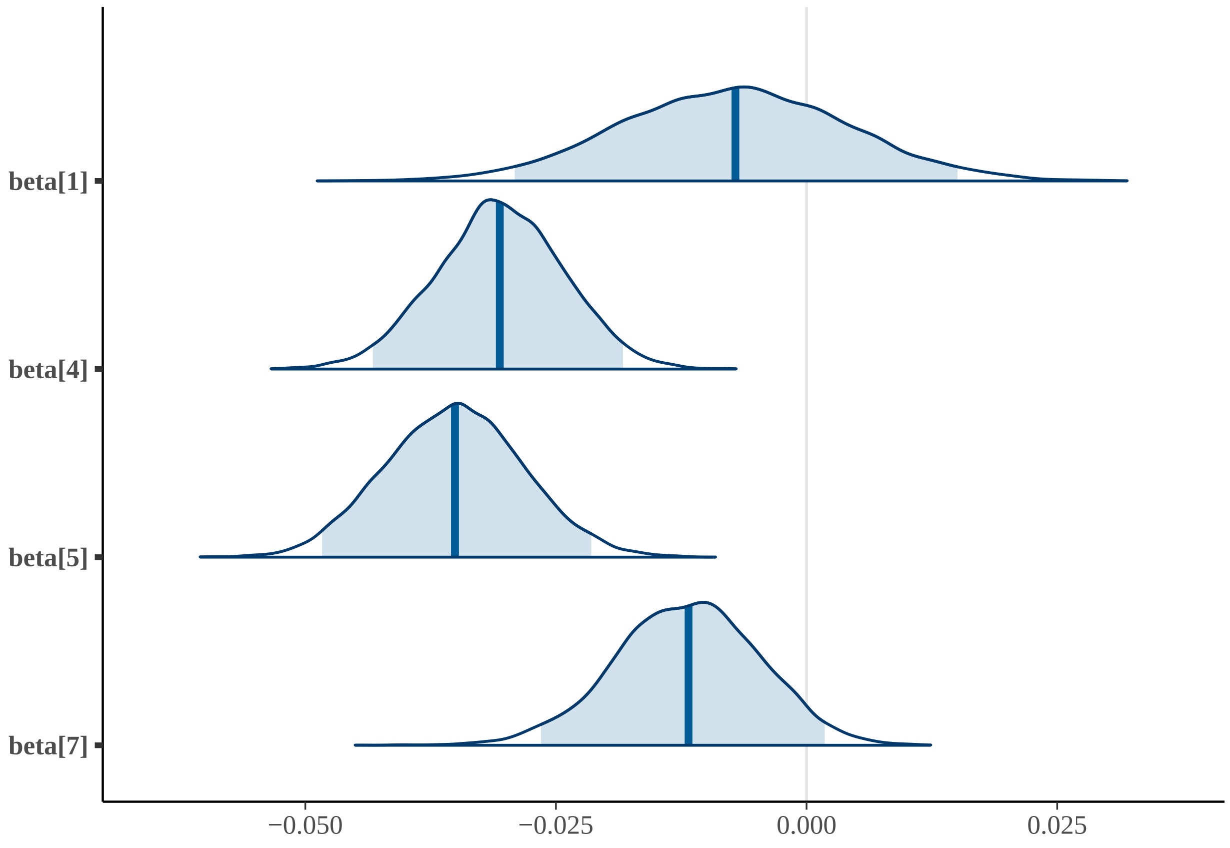
95% credible intervals of sampled alpha.

```
bayesplot::mcmc_areas(as.matrix(sampling),
                      pars = c("alpha"), prob = 0.95)
```



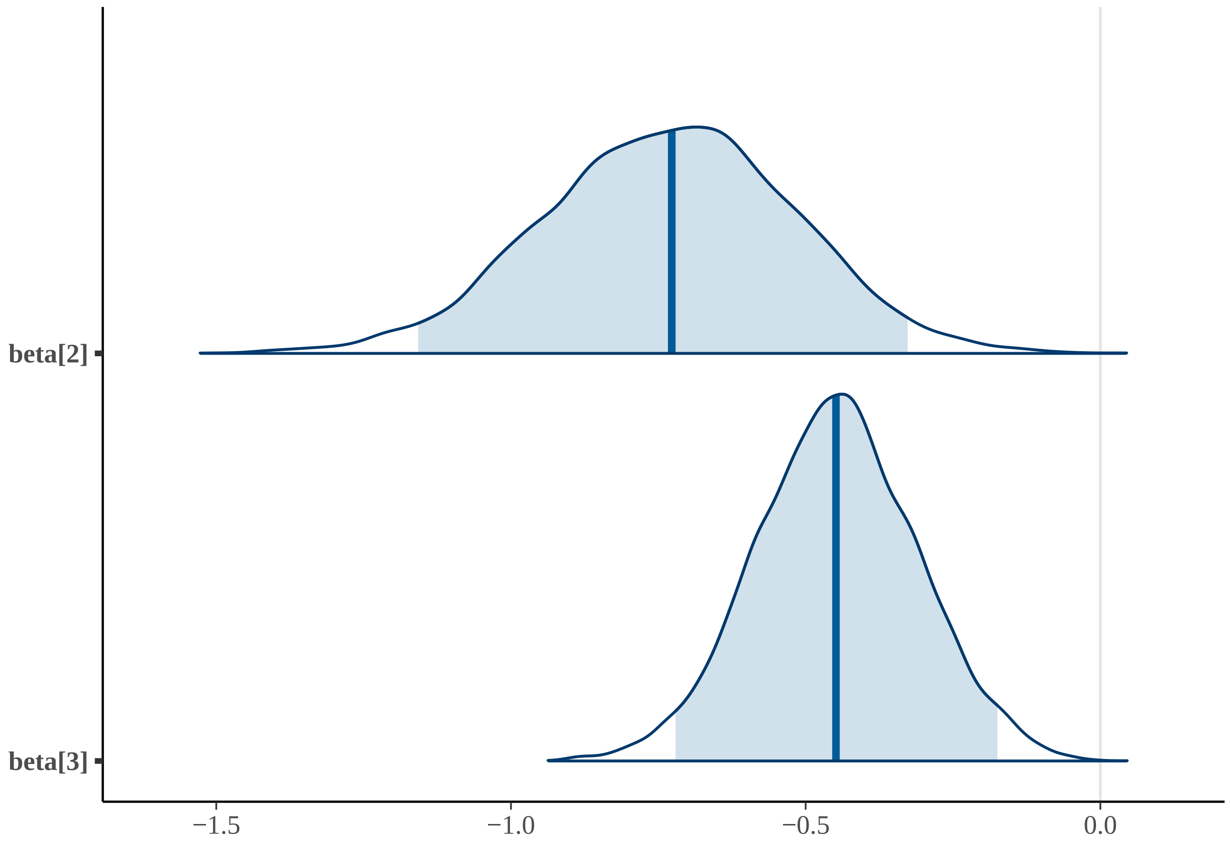
95% credible intervals of sampled betas.

```
bayesplot::mcmc_areas(as.matrix(sampling),
                      pars = c("beta[1]","beta[4]","beta[5]","beta[7]"), prob = 0.95)
```

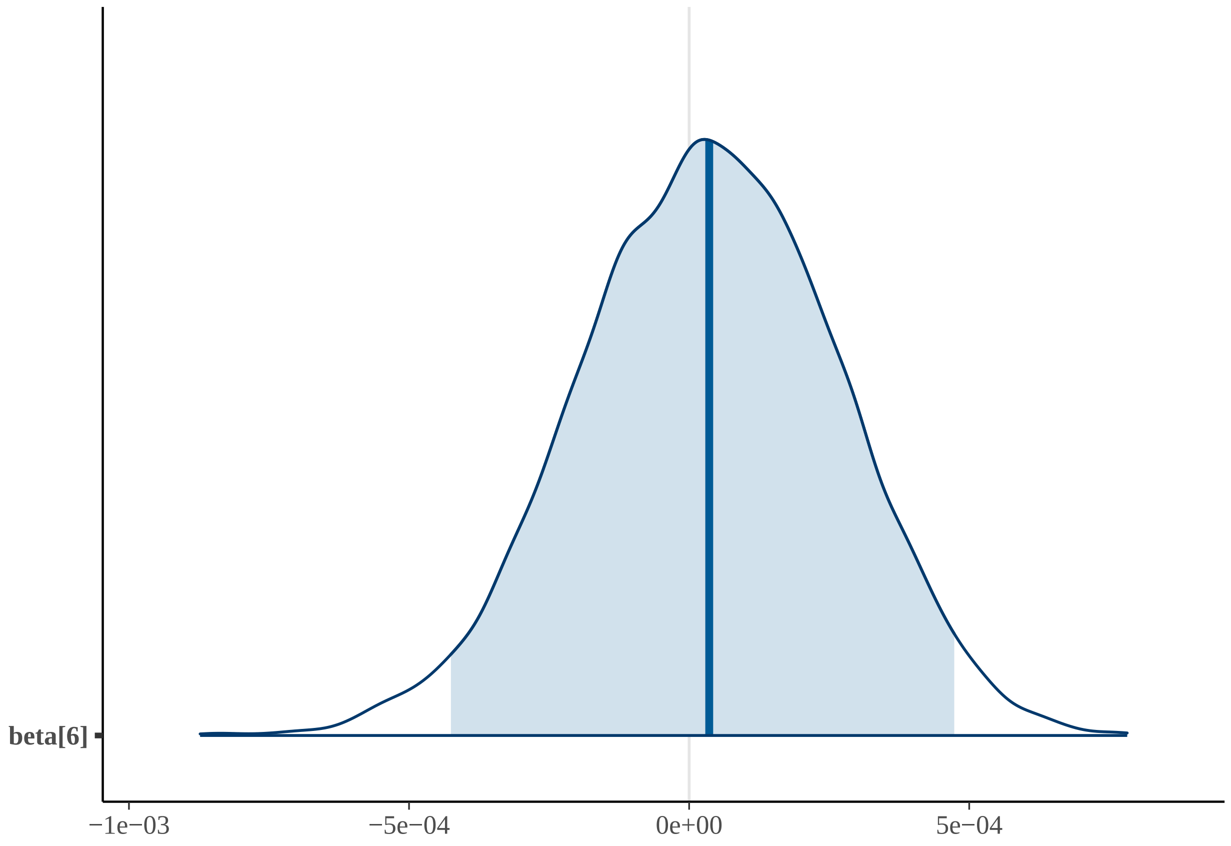


95% credible intervals of sampled betas.

```
par(mfrow=c(2,2))
bayesplot::mcmc_areas(as.matrix(sampling),
                      pars = c("beta[2]","beta[3]"), prob = 0.95)
```



```
bayesplot::mcmc_areas(as.matrix(sampling),
                      pars = c("beta[6]"), prob = 0.95)
```



Computation of the PSIS-LOO elpd values and the k-values

```
log_lik_sep = extract_log_lik(sampling, merge_chains = FALSE)
r_eff_sep = relative_eff(exp(log_lik_sep), cores = 2)
loo_sep = loo(log_lik_sep, r_eff = r_eff_sep, cores = 2)
```

```
## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

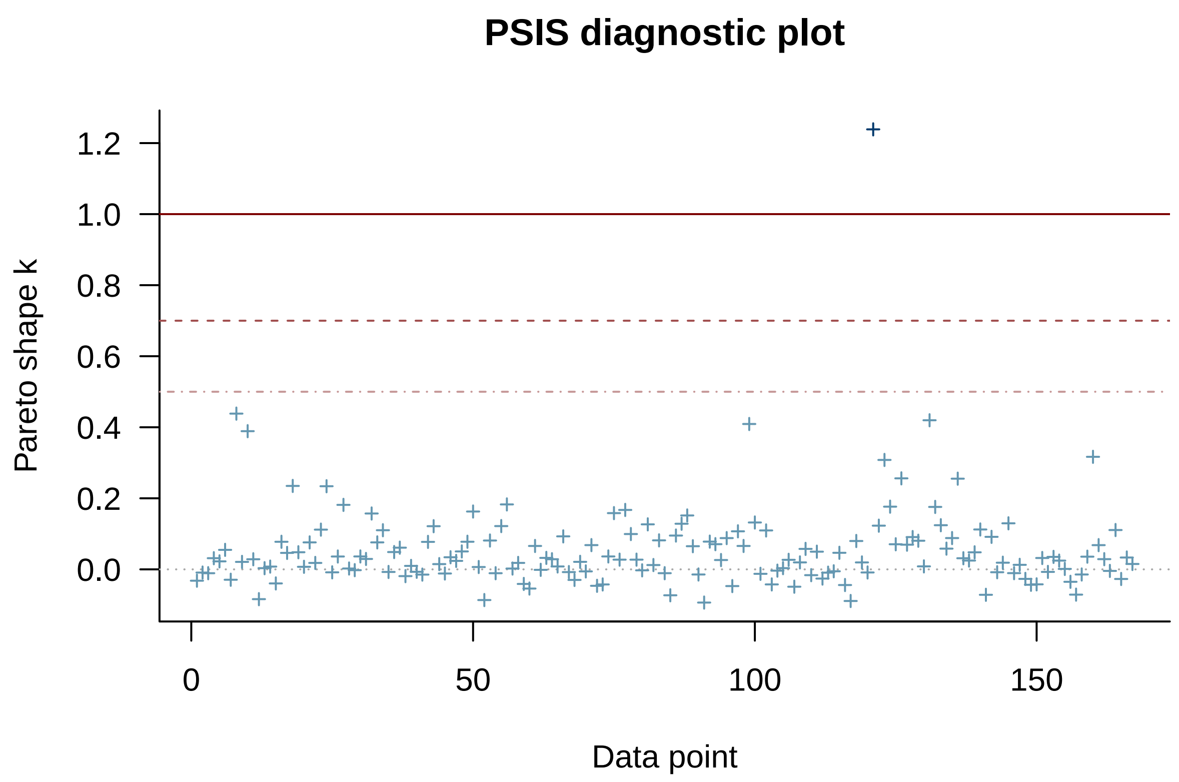
```
print(loo_sep)
```

```
##
## Computed from 10000 by 167 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -864.3 37.1
## p_loo        13.0  5.4
## looic      1728.5 74.3
## ------
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     166   99.4%   1899
##  (0.5, 0.7]   (ok)         0    0.0%   <NA>
##    (0.7, 1]   (bad)        0    0.0%   <NA>
##    (1, Inf)   (very bad)   1    0.6%   5
## See help('pareto-k-diagnostic') for details.
```

```
psis_sep = psis(-log_lik_sep, r_eff = r_eff_sep)
```

```
## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
plot(psis_sep)
```


PSIS diagnostic plot
Pareto shape k
Data point