homework14

1. Review the following case study, focusing on the first CAR model:

- Note there are two model they the tutorial used. In this homework I will focus on the second one which is a more compliated and more reasonable model.

```
library(rstan)
```

```
## Loading required package: ggplot2
```

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

```
## rstan (Version 2.18.1, 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)
```

```
options(mc.cores = parallel::detectCores())
rstan_options(auto_write = TRUE)
print_file <- function(file) {
  cat(paste(readLines(file), "\n", sep=""), sep="")
}
```

Load in the Scotland data.

```
# Define MCMC parameters
niter <- 1E4   # definitely overkill, but good for comparison
nchains <- 4
W <- A # adjacency matrix
scaled_x <- c(scale(x))
X <- model.matrix(~scaled_x)
full_d <- list(n = nrow(X),         # number of observations
               p = ncol(X),         # number of coefficients
               X = X,               # design matrix
               y = O,               # observed number of cases
               log_offset = log(E), # log(expected) num. cases
               W = W)               # adjacency matrix
```

a. Simulate fake data and check that the model recovers the parameters. Feel free to simplify the model as necessary.

```
library(MASS)
set.seed(123)
# simulate the fake data
## seting the parameters
tau <- rgamma(n=1,2,2)
beta <- rnorm(n=2,0,1)
alpha <- runif(n=1,0,1)
true_parameters <- list('tau'=tau,'beta'=beta,'alpha'=alpha)
true_parameters
```

```
## $tau
## [1] 0.4460468
##
## $beta
## [1]  1.190207 -1.689556
##
## $alpha
## [1] 0.892419
```

```
### simulate the fake data
set.seed(1234)
mu <- rep(0,full_d$n)
D <- diag(apply(full_d$W,1,sum))
prec <- true_parameters$tau * ( D - true_parameters$alpha * full_d$W)
sig <- solve(prec)
phi <- mvrnorm(n=1,mu = mu,Sigma = sig)
lambda <- c()
for (j in 1:full_d$n){
    lambda[j] <- exp(sum(full_d$X[j,] * beta)  + phi[j] + full_d$log_offset[j])
}
fake_y <- sapply(lambda,rpois,n=1)
```

First: try the CAR model

```
print_file('homework14.stan')
```

```
## Warning in readLines(file): incomplete final line found on
## 'homework14.stan'
```

```
## data {
##   int<lower = 1> n;
##   int<lower = 1> p;
##   matrix[n, p] X;
##   int<lower = 0> y[n];
##   vector[n] log_offset;
##   matrix<lower = 0, upper = 1>[n, n] W;
## }
## transformed data{
##   vector[n] zeros;
##   matrix<lower = 0>[n, n] D;
##   {
##     vector[n] W_rowsums;
##     for (i in 1:n) {
##       W_rowsums[i] = sum(W[i, ]);
##     }
##     D = diag_matrix(W_rowsums);
##   }
##   zeros = rep_vector(0, n);
## }
## parameters {
##   vector[p] beta;
##   vector[n] phi;
##   real<lower = 0> tau;
##   real<lower = 0, upper = 1> alpha;
## }
## model {
##   phi ~ multi_normal_prec(zeros, tau * (D - alpha * W));
##   beta ~ normal(0, 1);
##   tau ~ gamma(2, 2);
##   y ~ poisson_log(X * beta + phi + log_offset);
## }
## generated quantities{
##   int<lower = 0> y_rep[n];
##   y_rep = poisson_log_rng(X * beta + phi + log_offset);
## }
```

```
comp_model <- stan_model('homework14.stan')
```

```
## Warning in readLines(file, warn = TRUE): incomplete final line found on '/
## Users/yi/Desktop/study/subjects/bayesian-data-analysis/homework/homework
## 14/homework14.stan'
```

```
fake_data <- list(n = nrow(X),          # number of observations
                  p = ncol(X),          # number of coefficients
                  X = X,                # design matrix
                  y = fake_y,                # observed number of cases
                  log_offset = log(E), # log(expected) num. cases
                  W = W)
fit_model_fake <- sampling(comp_model, data = fake_data, seed = 123)
```

```
## Warning: There were 13 transitions after warmup that exceeded the maximum treedepth.
Increase max_treedepth above 10. See
## http://mc-stan.org/misc/warnings.html#maximum-treedepth-exceeded
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
posterior_parameters <- as.matrix(fit_model_fake,pars = c('tau','beta','alpha'))
library(bayesplot)
```

```
## This is bayesplot version 1.6.0
```

```
## - Online documentation and vignettes at mc-stan.org/bayesplot
```

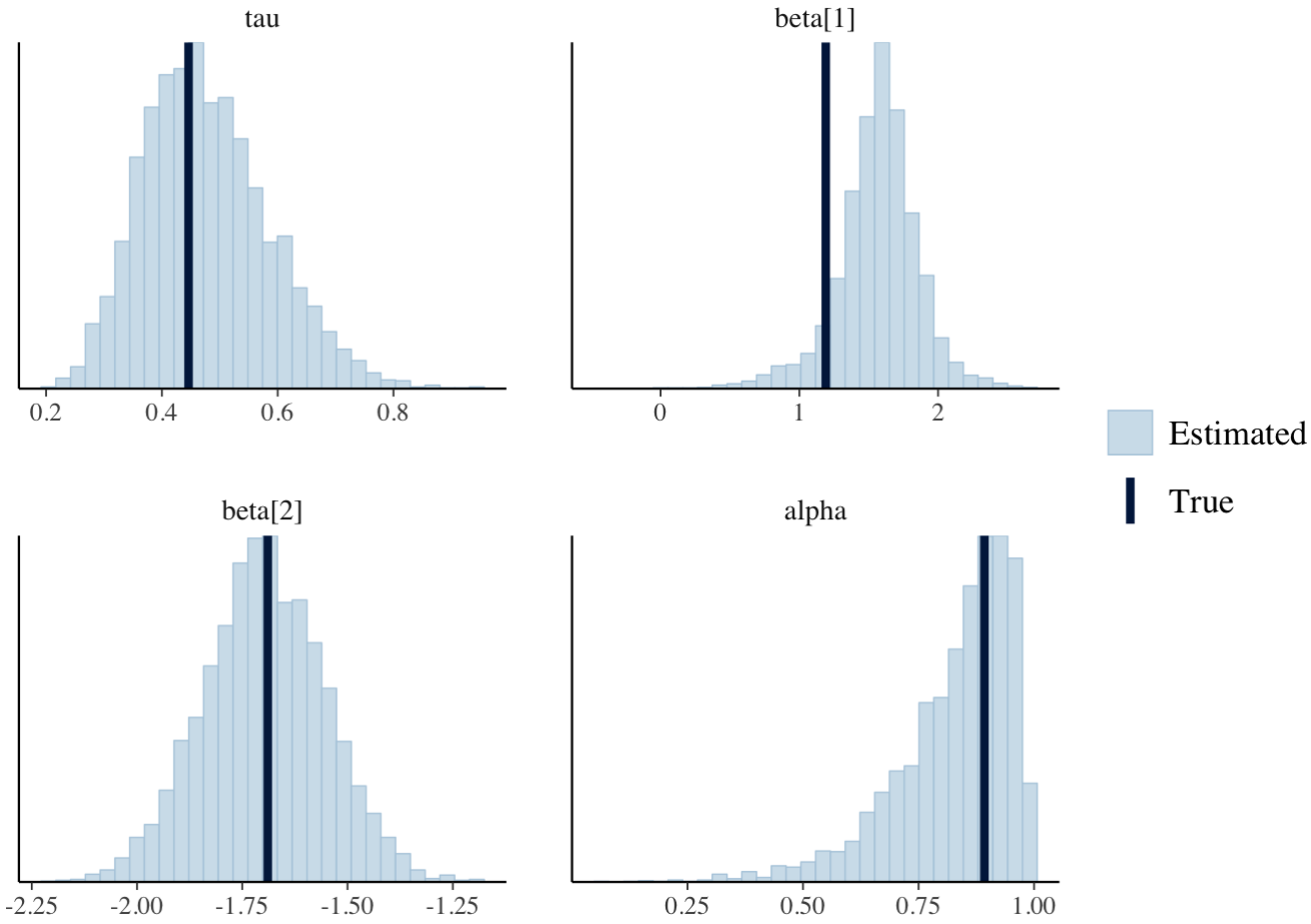
```
## - bayesplot theme set to bayesplot::theme_default()
```

```
##    * Does _not_ affect other ggplot2 plots
```

```
##    * See ?bayesplot_theme_set for details on theme setting
```

```
mcmc_recover_hist(posterior_parameters,true = c(true_parameters$tau,true_parameters$bet
a,true_parameters$alpha))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



As we can see the true parameters are well within the parameter space.

Second try the model with sparse representation

```
print_file('car_sparse.stan')
```

```
## Warning in readLines(file): incomplete final line found on
## 'car_sparse.stan'
```

```
## functions {
##   /**
##   * Return the log probability of a proper conditional autoregressive (CAR) prior
##   * with a sparse representation for the adjacency matrix
##   *
##   * @param phi Vector containing the parameters with a CAR prior
##   * @param tau Precision parameter for the CAR prior (real)
##   * @param alpha Dependence (usually spatial) parameter for the CAR prior (real)
##   * @param W_sparse Sparse representation of adjacency matrix (int array)
##   * @param n Length of phi (int)
##   * @param W_n Number of adjacent pairs (int)
##   * @param D_sparse Number of neighbors for each location (vector)
##   * @param lambda Eigenvalues of D^{-1/2}*W*D^{-1/2} (vector)
##   *
##   * @return Log probability density of CAR prior up to additive constant
##   */
##   real sparse_car_lpdf(vector phi, real tau, real alpha,
##     int[,] W_sparse, vector D_sparse, vector lambda, int n, int W_n) {
##       row_vector[n] phit_D; // phi' * D
##       row_vector[n] phit_W; // phi' * W
##       vector[n] ldet_terms;
##
##       phit_D = (phi .* D_sparse)';
##       phit_W = rep_row_vector(0, n);
##       for (i in 1:W_n) {
##         phit_W[W_sparse[i, 1]] = phit_W[W_sparse[i, 1]] + phi[W_sparse[i, 2]];
##         phit_W[W_sparse[i, 2]] = phit_W[W_sparse[i, 2]] + phi[W_sparse[i, 1]];
##       }
##
##       for (i in 1:n) ldet_terms[i] = log1m(alpha * lambda[i]);
##       return 0.5 * (n * log(tau)
##                     + sum(ldet_terms)
##                     - tau * (phit_D * phi - alpha * (phit_W * phi)));
##   }
## }
## data {
##   int<lower = 1> n;
##   int<lower = 1> p;
##   matrix[n, p] X;
##   int<lower = 0> y[n];
##   vector[n] log_offset;
##   matrix<lower = 0, upper = 1>[n, n] W; // adjacency matrix
##   int W_n;                // number of adjacent region pairs
## }
## transformed data {
##   int W_sparse[W_n, 2];   // adjacency pairs
##   vector[n] D_sparse;     // diagonal of D (number of neigbors for each site)
##   vector[n] lambda;       // eigenvalues of invsqrtD * W * invsqrtD
##
##   { // generate sparse representation for W
##   int counter;
##   counter = 1;
##   // loop over upper triangular part of W to identify neighbor pairs
```

```
##     for (i in 1:(n - 1)) {
##       for (j in (i + 1):n) {
##         if (W[i, j] == 1) {
##           W_sparse[counter, 1] = i;
##           W_sparse[counter, 2] = j;
##           counter = counter + 1;
##         }
##       }
##     }
##   }
##   for (i in 1:n) D_sparse[i] = sum(W[i]);
##   {
##     vector[n] invsqrtD;
##     for (i in 1:n) {
##       invsqrtD[i] = 1 / sqrt(D_sparse[i]);
##     }
##     lambda = eigenvalues_sym(quad_form(W, diag_matrix(invsqrtD)));
##   }
## }
## parameters {
##   vector[p] beta;
##   vector[n] phi;
##   real<lower = 0> tau;
##   real<lower = 0, upper = 1> alpha;
## }
## model {
##   phi ~ sparse_car(tau, alpha, W_sparse, D_sparse, lambda, n, W_n);
##   beta ~ normal(0, 1);
##   tau ~ gamma(2, 2);
##   y ~ poisson_log(X * beta + phi + log_offset);
## }
## generated quantities{
##   int<lower=0> y_rep[n];
##   y_rep = poisson_log_rng(X * beta + phi + log_offset);
## }
```

```
fake_data_sp <- list(n = nrow(X),         # number of observations
             p = ncol(X),         # number of coefficients
             X = X,               # design matrix
             y = fake_y,               # observed number of cases
             log_offset = log(E), # log(expected) num. cases
             W_n = sum(W) / 2,    # number of neighbor pairs
             W = W)               # adjacency matrix

sp_comp_model <- stan_model('car_sparse.stan')
```

```
## Warning in readLines(file, warn = TRUE): incomplete final line found on '/
## Users/yi/Desktop/study/subjects/bayesian-data-analysis/homework/homework
## 14/car_sparse.stan'
```

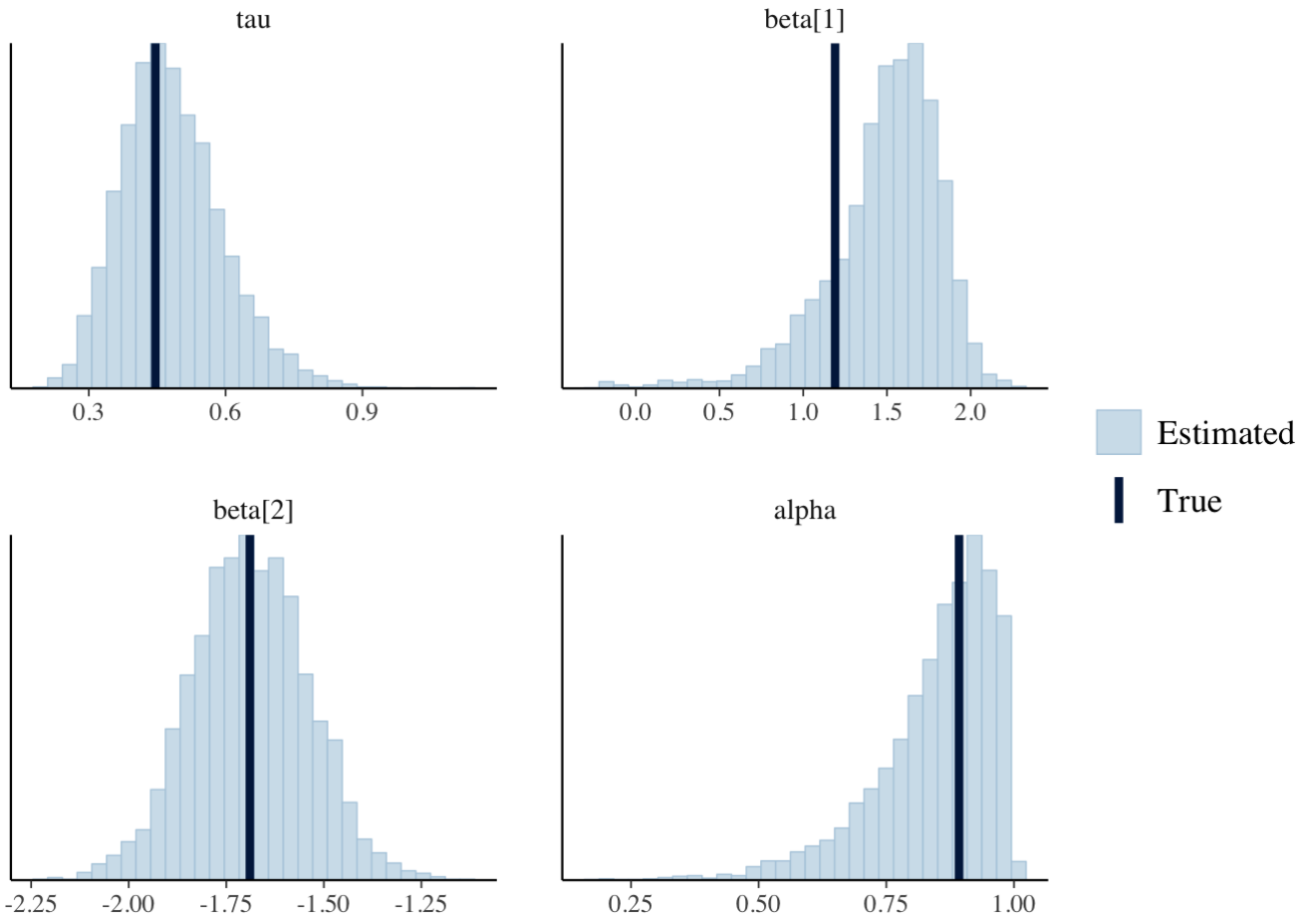
```
sp_model_fake <- sampling(sp_comp_model, data = fake_data_sp, seed = 123)
```

```
## Warning: There were 7 transitions after warmup that exceeded the maximum treedepth. I
ncrease max_treedepth above 10. See
## http://mc-stan.org/misc/warnings.html#maximum-treedepth-exceeded
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
sp_posterior_parameters <- as.matrix(sp_model_fake,pars = c('tau','beta','alpha'))
library(bayesplot)
mcmc_recover_hist(sp_posterior_parameters,true = c(true_parameters$tau,true_parameters$b
eta,true_parameters$alpha))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Again we can see the true parameters are well within the parameter space.

b. In two or three sentences, discuss the strengths and weaknesses of the model. How might the model be expanded?

The strength of this model:

1. clearly the CAR model is good prior distribution which make use of the spatial information. This make sense that the local enviornment and the interation of the population make the place near to each other have certain influence which should be considered.

The weekness of this model:

1. the count data use the likelihood of poisson distribution. The poisson distribution have the mean and variance paramter as the same called lambda. This may not true for the data. As we can see variance is much bigger than mean in the real model. I know the distribution of y is condition on this local parameters but here the difference is just seems to be too big. I will try to use the negative binormal regression instead of poisson.

```
mean(full_d$y);var(full_d$y)
```

```
## [1] 9.571429
```

```
## [1] 62.54026
```

2. I could try to use more robust prior like t distribution with df equal to cauchy distribution.

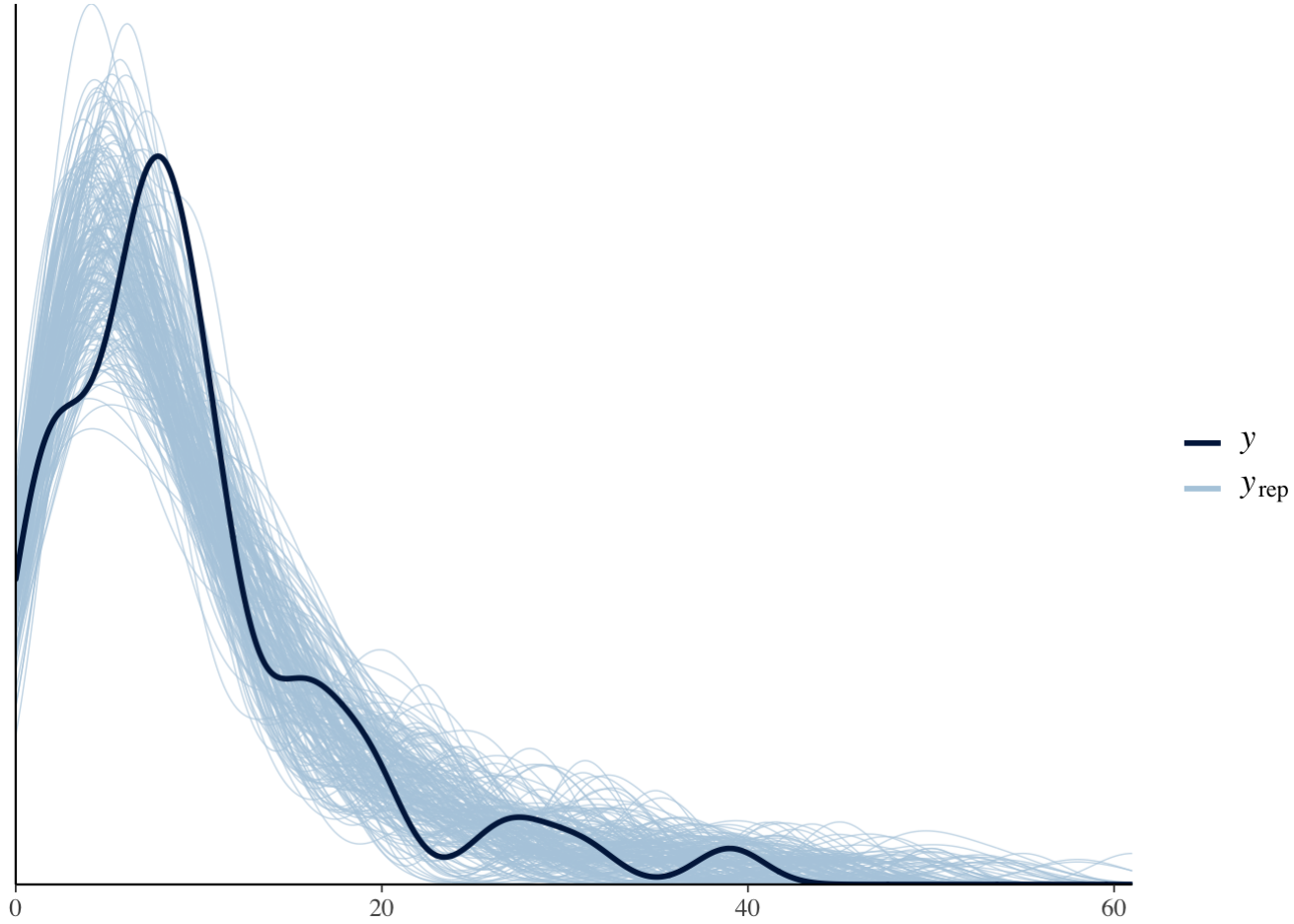
2. model fit and extend the model

a. Fit the model to the real data and perform model checking and/or validation (Chapters 6 and 7 of BDA).

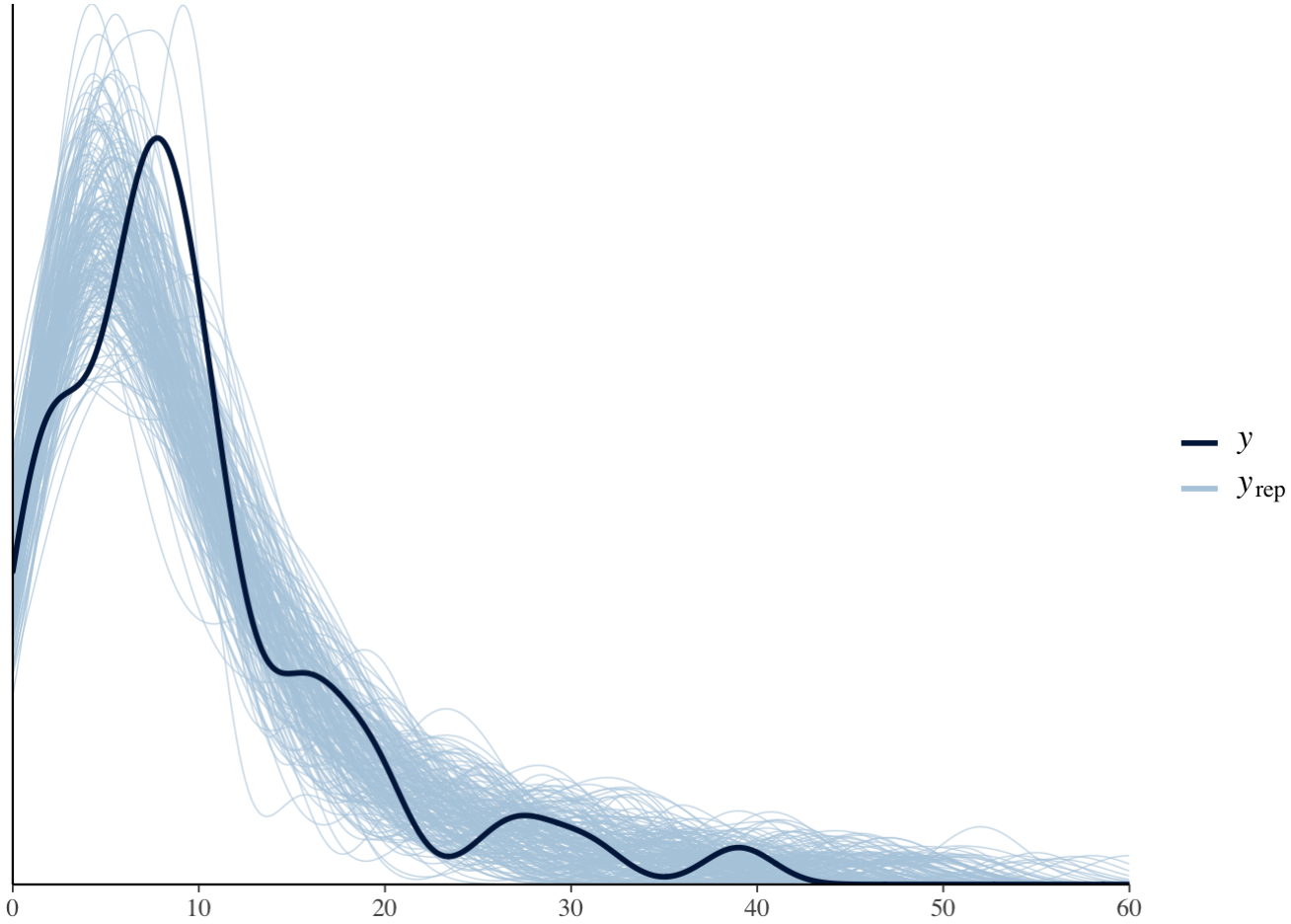
```
#fit the model with real data
full_real <- list(n = nrow(X),          # number of observations
                  p = ncol(X),         # number of coefficients
                  X = X,               # design matrix
                  y = O,               # observed number of cases
                  log_offset = log(E), # log(expected) num. cases
                  W = W)               # adjacency matrix
sp_real <- list(n = nrow(X),           # number of observations
                p = ncol(X),          # number of coefficients
                X = X,                # design matrix
                y = O,                # observed number of cases
                log_offset = log(E), # log(expected) num. cases
                W_n = sum(W) / 2,     # number of neighbor pairs
                W = W)                # adjacency matrix
real_fit <- stan('homework14.stan',data=full_real)
real_fit_sp <- stan('car_sparse.stan',data=sp_real)
```

Let do the posterior predictive test.

```
sims <- extract(real_fit)
sims_sp <- extract(real_fit_sp)
## for the CAR model
y_rep <- sims$y_rep
ppc_dens_overlay(full_real$y, y_rep[1:200,])
```



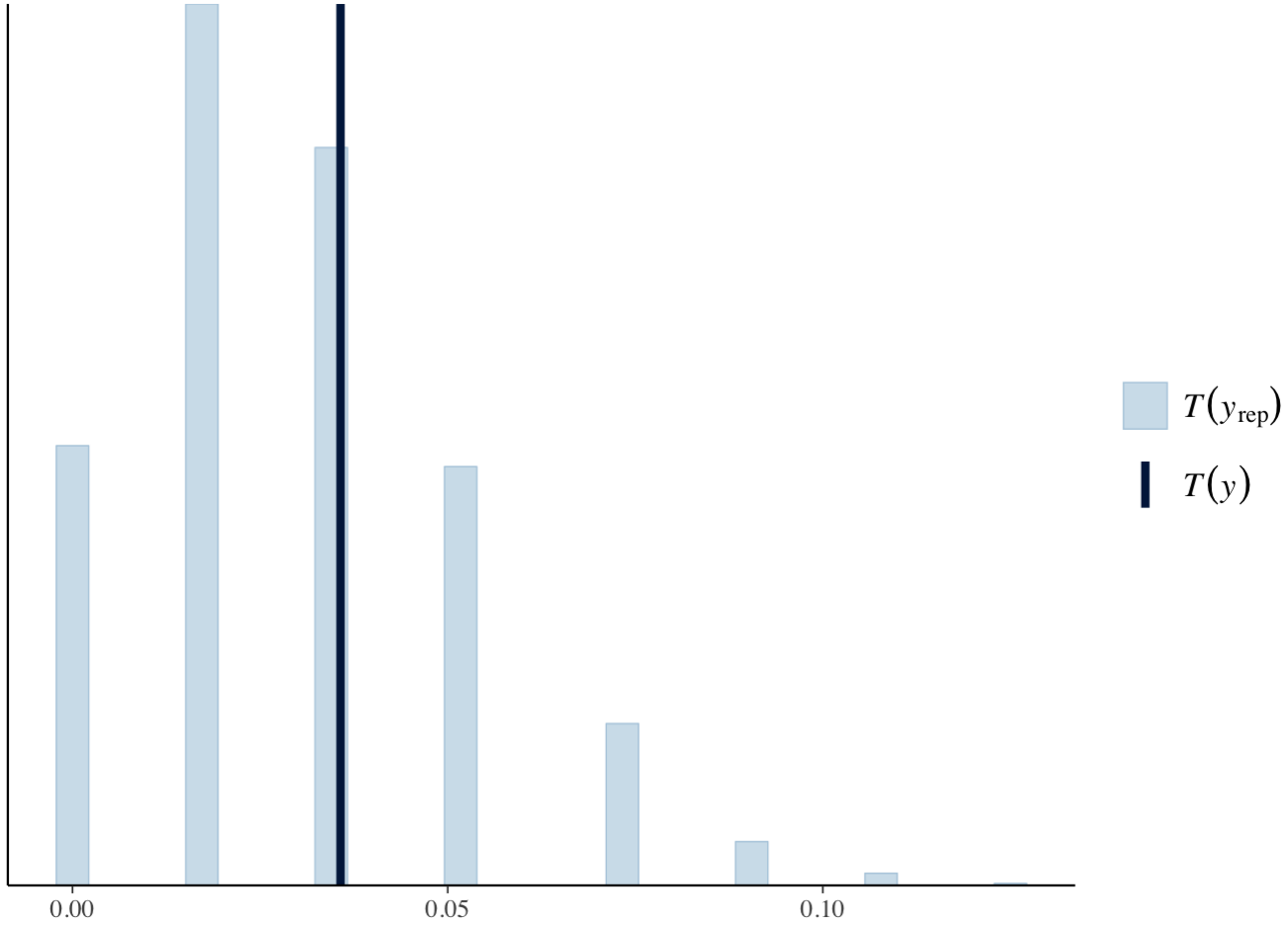

```
## for the spear model
y_rep_sp <- sims_sp$y_rep
ppc_dens_overlay(sp_real$y, y_rep_sp[1:200,])
```



check the prop_zero

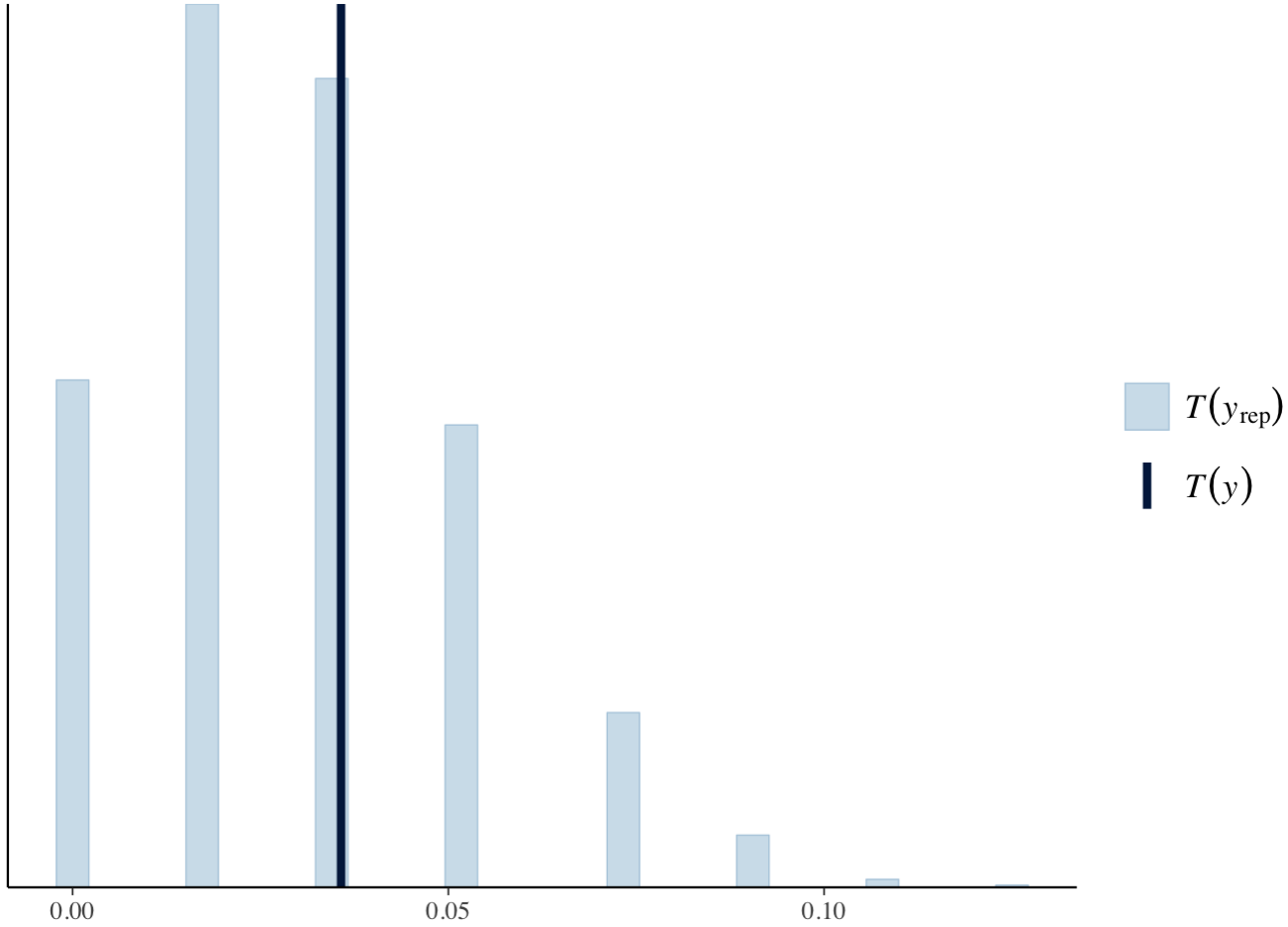
```
ppc_stat(y = full_real$y, yrep = y_rep, stat = function(x){mean(x==0)})
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
ppc_stat(y = sp_real$y, yrep = y_rep_sp, stat = function(x){mean(x==0)})
```

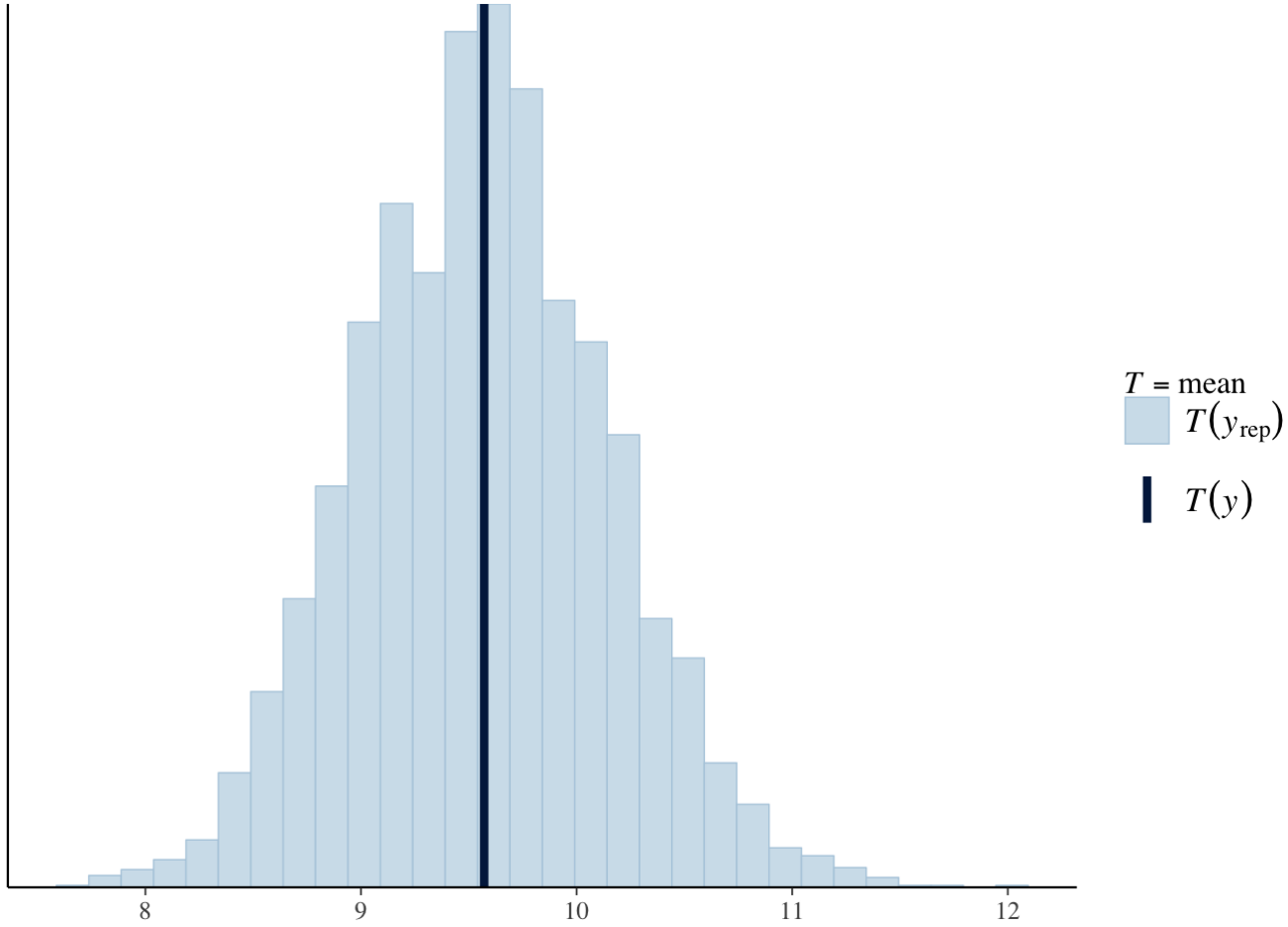
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



check the mean

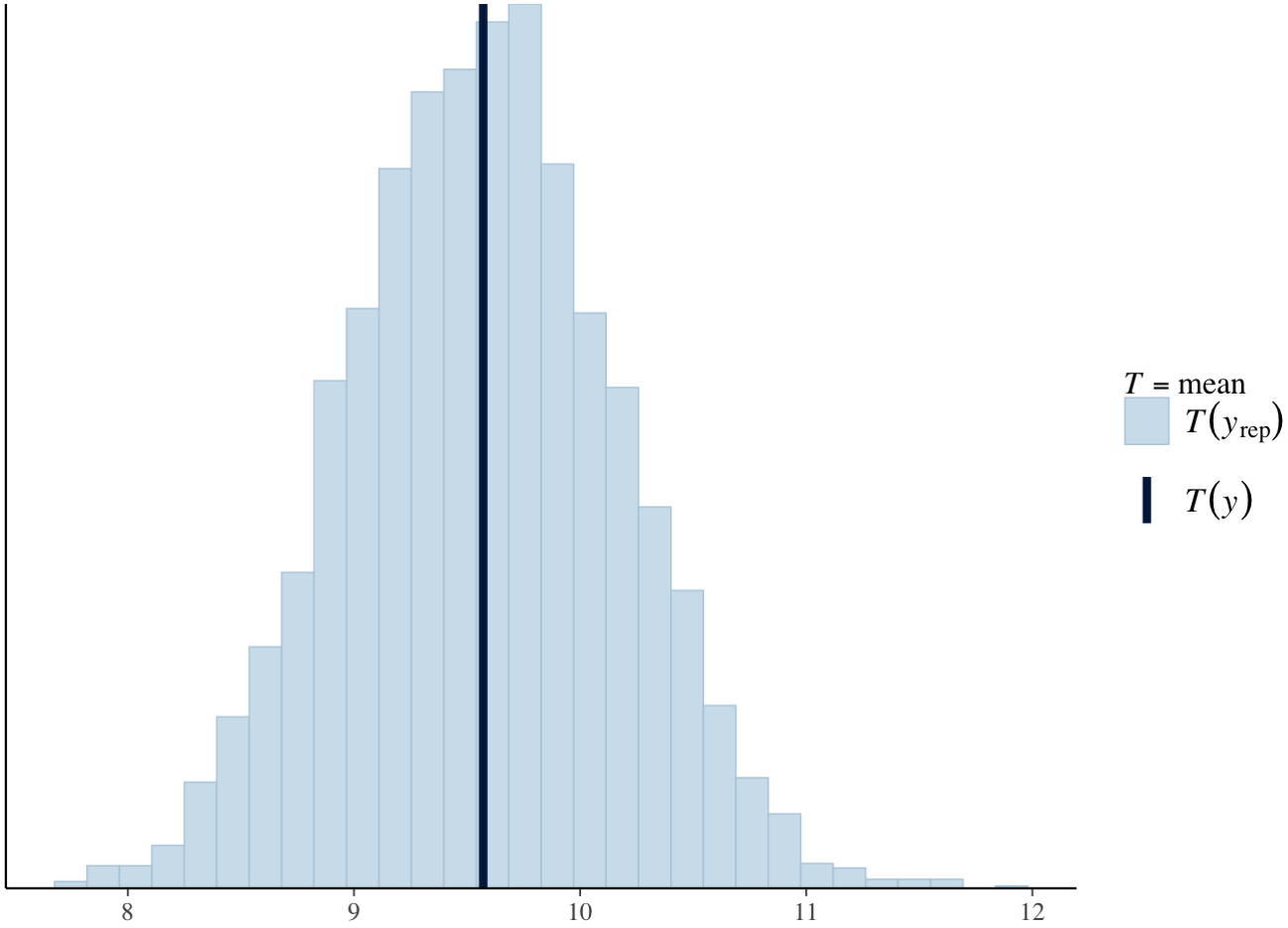
```
ppc_stat(y = full_real$y, yrep = y_rep, stat = mean)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
ppc_stat(y = sp_real$y, yrep = y_rep_sp, stat =mean)
```

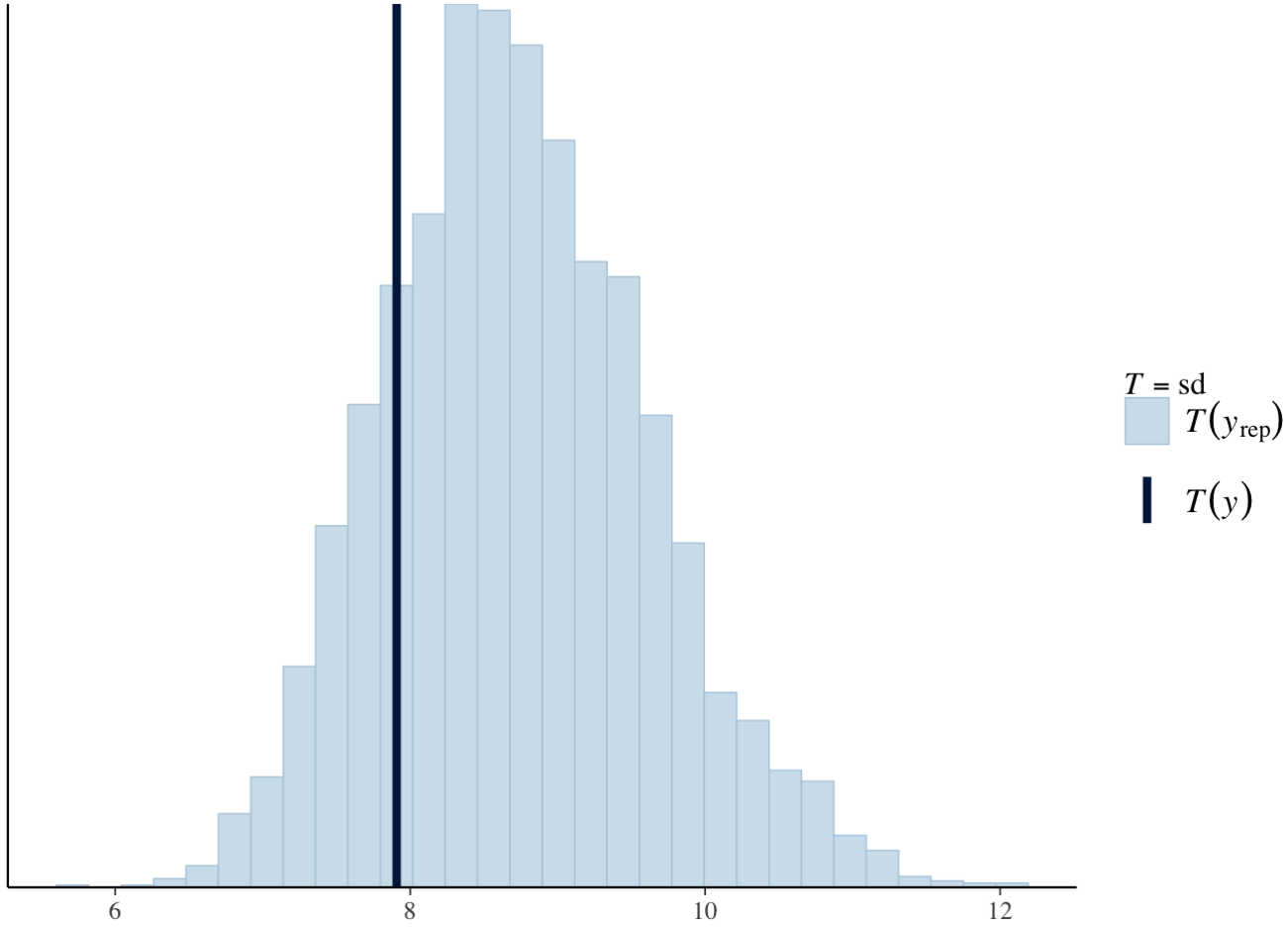
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



check the standard deviation

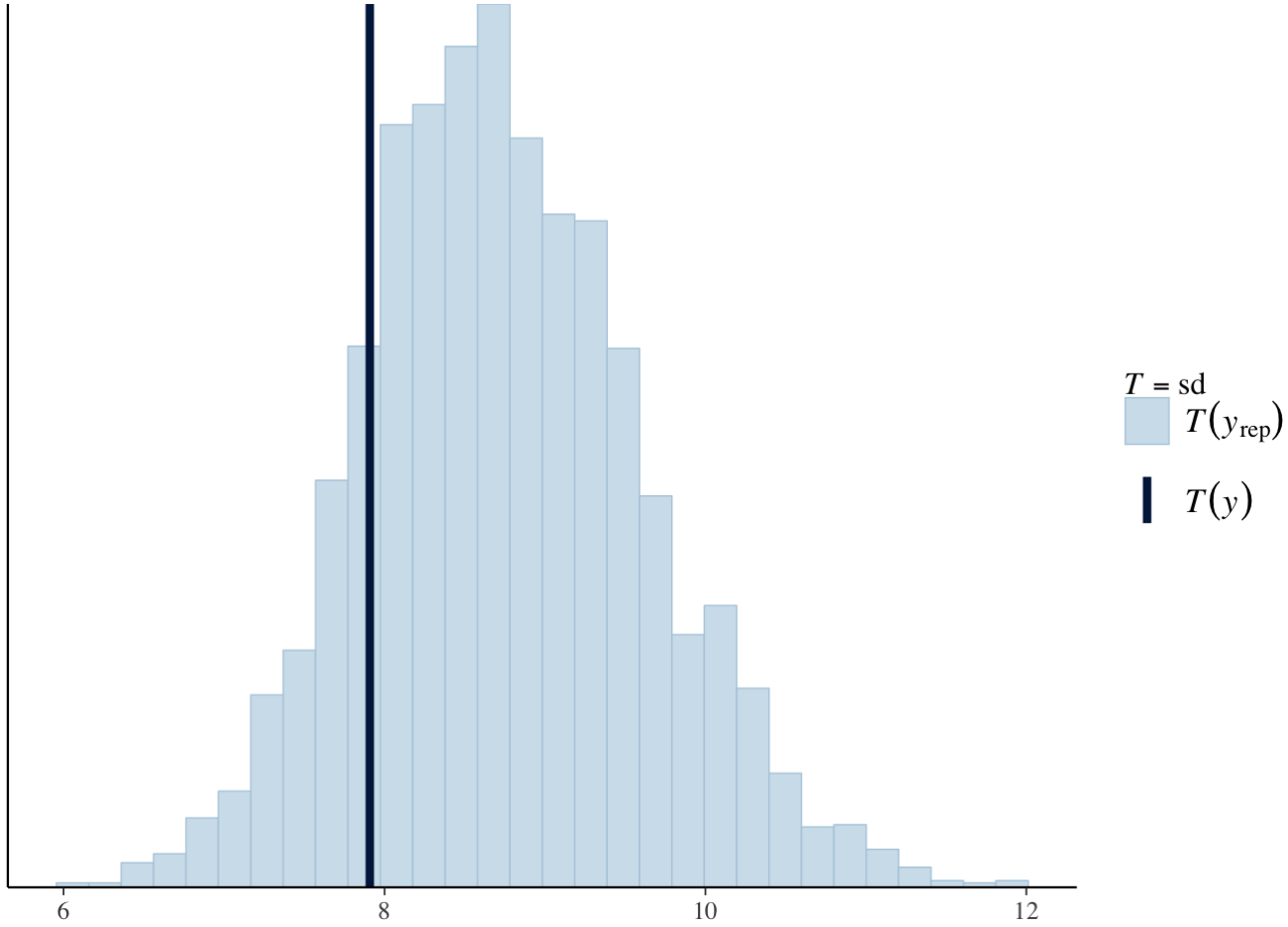
```
ppc_stat(y = full_real$y, yrep = y_rep, stat = sd)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



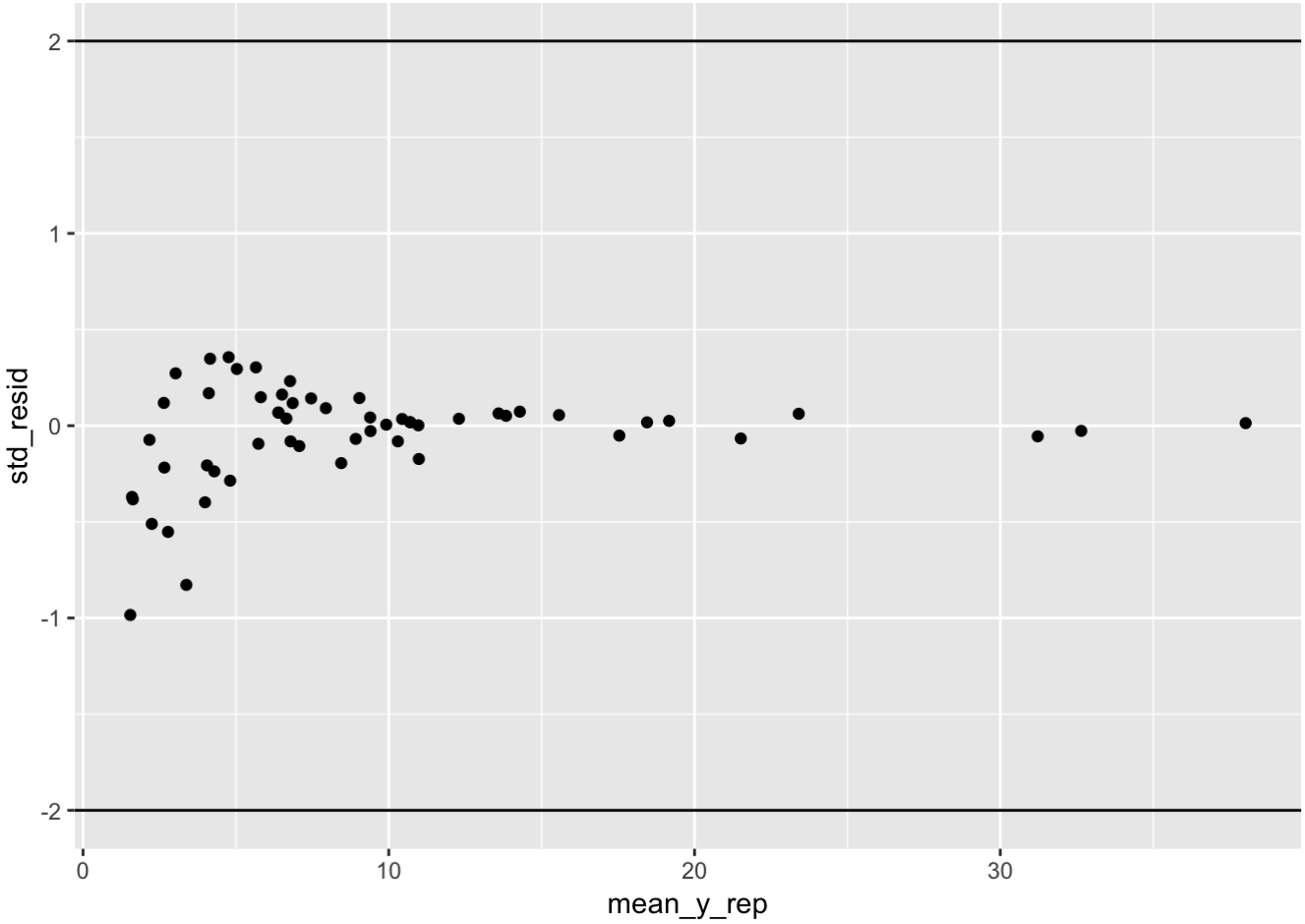

```
ppc_stat(y = sp_real$y, yrep = y_rep_sp, stat =sd)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

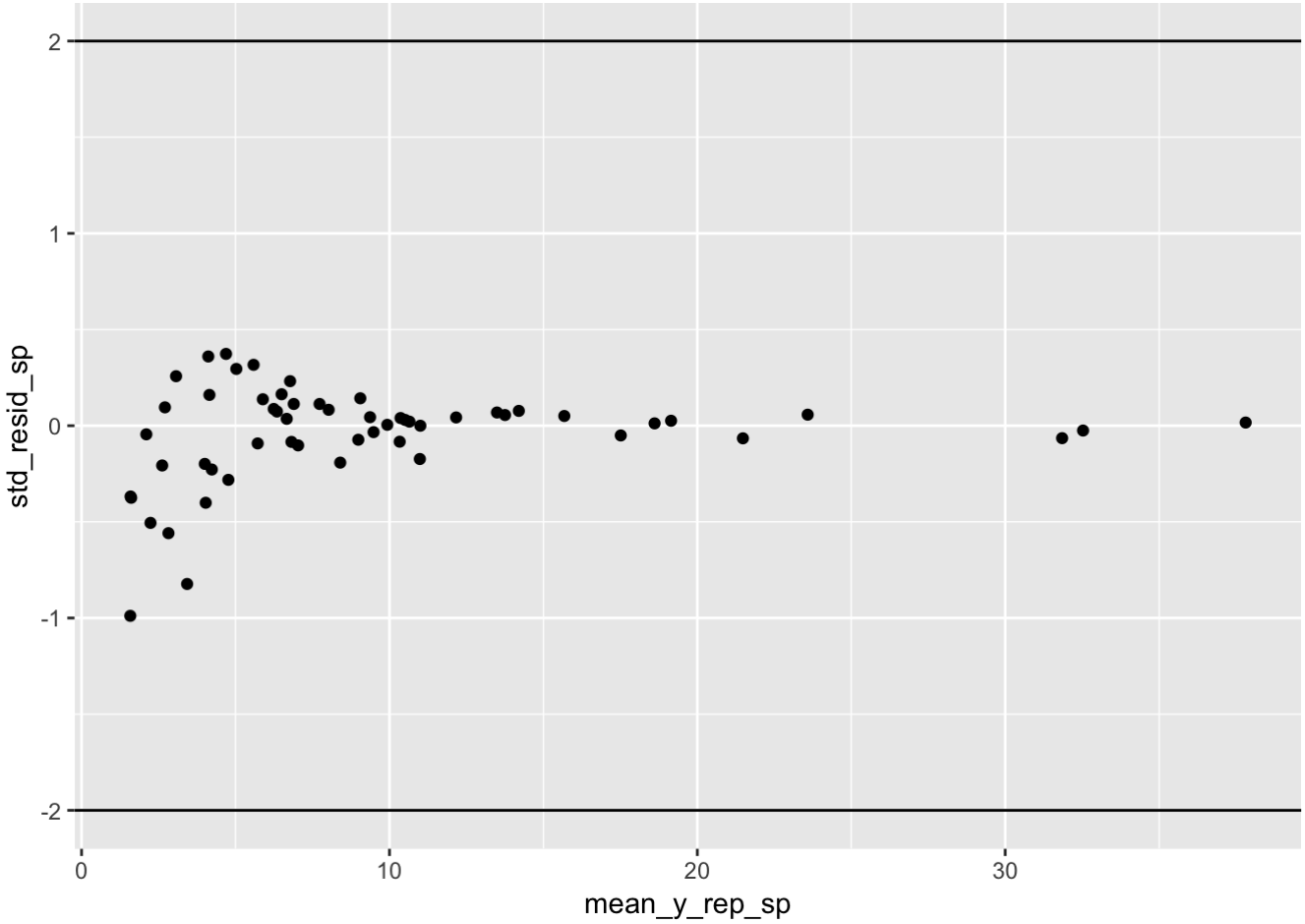


check the standardized residual plot

```
# FOR THE CRA MODEL
lambda <- as.matrix(full_real$X) %*% as.matrix(apply(sims$beta,2,mean)) + apply(sims$ph
i,2,mean) + full_real$log_offset
mean_y_rep <- colMeans(y_rep)
std_resid <- (full_real$y - mean_y_rep) / sqrt(mean_y_rep + mean_y_rep^2*lambda)
qplot(mean_y_rep, std_resid) + hline_at(2) + hline_at(-2)
```

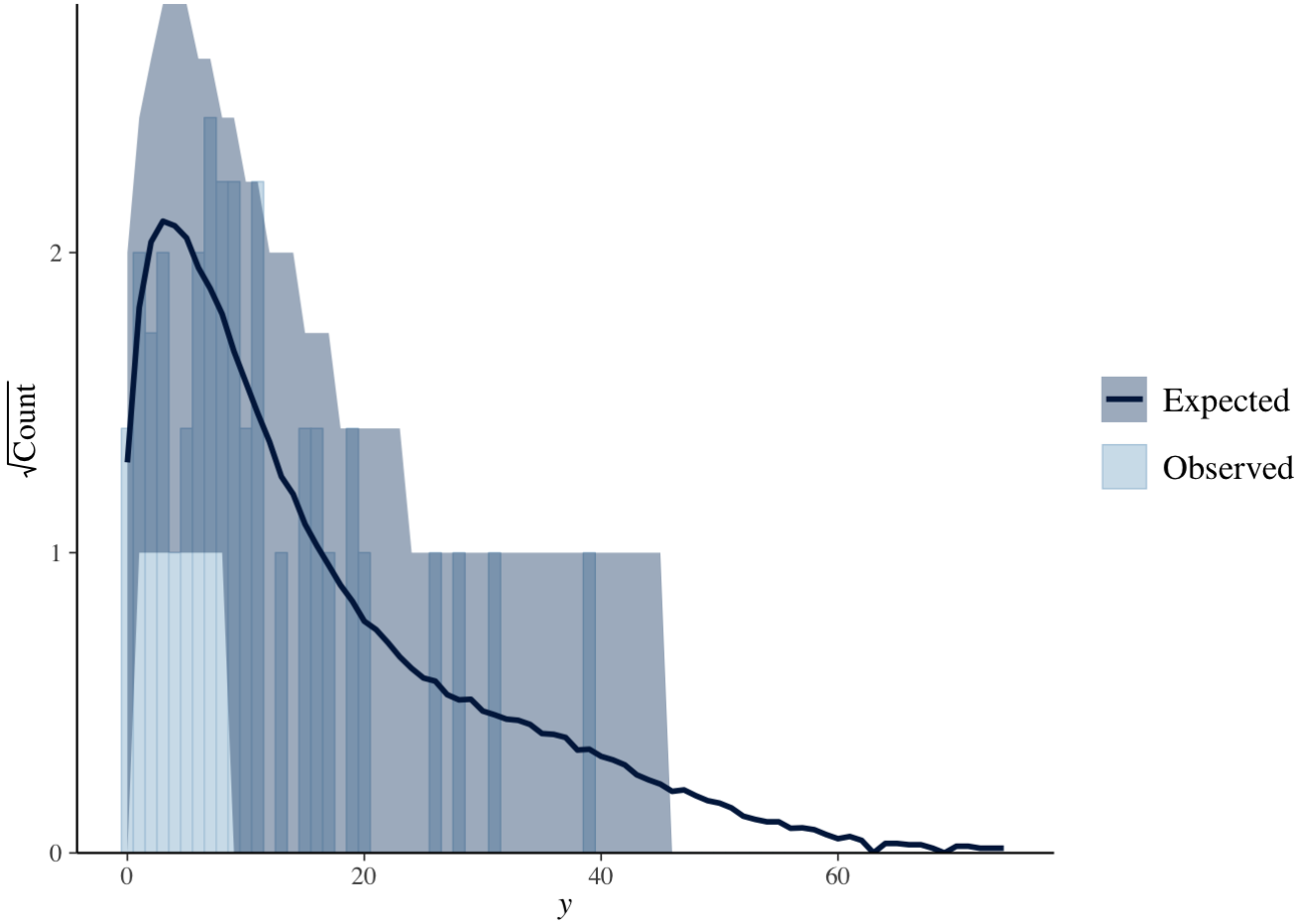



```
# For the spear model
lambda_sp <- as.matrix(sp_real$X) %*% as.matrix(apply(sims_sp$beta,2,mean)) + apply(sims
_sp$phi,2,mean) + sp_real$log_offset
mean_y_rep_sp <- colMeans(y_rep_sp)
std_resid_sp <- (sp_real$y - mean_y_rep_sp) / sqrt(mean_y_rep_sp + mean_y_rep_sp^2*lambd
a_sp)
qplot(mean_y_rep_sp, std_resid_sp) + hline_at(2) + hline_at(-2)
```

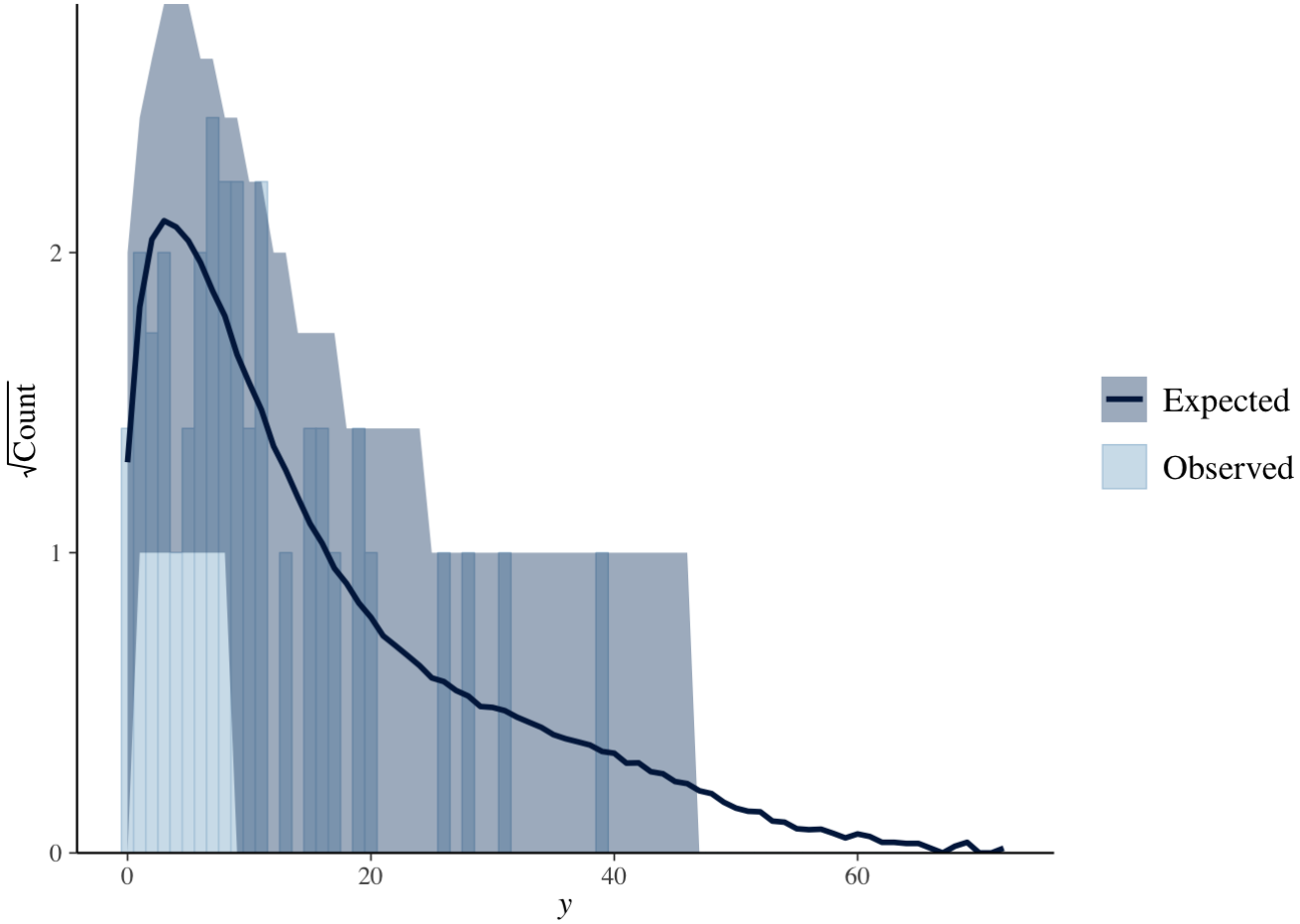


As we can see that

```
ppc_rootogram(full_real$y,yrep=y_rep)
```




```
ppc_rootogram(sp_real$y,yrep = y_rep_sp)
```



In summarize, all of the test shows that both model did really good job. And we can even do more like calcuelate waic or do cross validation. But, here I think the plot is good enough to show that the model is reasonable. And they are not so different.

b. Expand the model as discussed in 1.b./class and interpret the results.

Two main expendation I did:

1. $y_i Neg-Binomial(exp(X_i+ _i + log(offset_i)),_i) $
2. $_i cauchy(0) $

```
print_file('my.stan')
```

```
## functions {
##   /**
##   * Return the log probability of a proper conditional autoregressive (CAR) prior
##   * with a sparse representation for the adjacency matrix
##   *
##   * @param phi Vector containing the parameters with a CAR prior
##   * @param tau Precision parameter for the CAR prior (real)
##   * @param alpha Dependence (usually spatial) parameter for the CAR prior (real)
##   * @param W_sparse Sparse representation of adjacency matrix (int array)
##   * @param n Length of phi (int)
##   * @param W_n Number of adjacent pairs (int)
##   * @param D_sparse Number of neighbors for each location (vector)
##   * @param lambda Eigenvalues of D^{-1/2}*W*D^{-1/2} (vector)
##   *
##   * @return Log probability density of CAR prior up to additive constant
##   */
##   real sparse_car_lpdf(vector phi, real tau, real alpha,
##     int[,] W_sparse, vector D_sparse, vector lambda, int n, int W_n) {
##       row_vector[n] phit_D; // phi' * D
##       row_vector[n] phit_W; // phi' * W
##       vector[n] ldet_terms;
##
##       phit_D = (phi .* D_sparse)';
##       phit_W = rep_row_vector(0, n);
##       for (i in 1:W_n) {
##         phit_W[W_sparse[i, 1]] = phit_W[W_sparse[i, 1]] + phi[W_sparse[i, 2]];
##         phit_W[W_sparse[i, 2]] = phit_W[W_sparse[i, 2]] + phi[W_sparse[i, 1]];
##       }
##
##       for (i in 1:n) ldet_terms[i] = log1m(alpha * lambda[i]);
##       return 0.5 * (n * log(tau)
##                     + sum(ldet_terms)
##                     - tau * (phit_D * phi - alpha * (phit_W * phi)));
##   }
## }
## data {
##   int<lower = 1> n;
##   int<lower = 1> p;
##   matrix[n, p] X;
##   int<lower = 0> y[n];
##   vector[n] log_offset;
##   matrix<lower = 0, upper = 1>[n, n] W; // adjacency matrix
##   int W_n;                // number of adjacent region pairs
## }
## transformed data {
##   int W_sparse[W_n, 2];   // adjacency pairs
##   vector[n] D_sparse;     // diagonal of D (number of neigbors for each site)
##   vector[n] lambda;       // eigenvalues of invsqrtD * W * invsqrtD
##
##   { // generate sparse representation for W
##   int counter;
##   counter = 1;
##   // loop over upper triangular part of W to identify neighbor pairs
```

```
##     for (i in 1:(n - 1)) {
##       for (j in (i + 1):n) {
##         if (W[i, j] == 1) {
##           W_sparse[counter, 1] = i;
##           W_sparse[counter, 2] = j;
##           counter = counter + 1;
##         }
##       }
##     }
##   }
##   for (i in 1:n) D_sparse[i] = sum(W[i]);
##   {
##     vector[n] invsqrtD;
##     for (i in 1:n) {
##       invsqrtD[i] = 1 / sqrt(D_sparse[i]);
##     }
##     lambda = eigenvalues_sym(quad_form(W, diag_matrix(invsqrtD)));
##   }
## }
## parameters {
##   vector[p] beta;
##   vector[n] phi;
##   real<lower = 0> tau;
##   real<lower = 0, upper = 1> alpha;
##   real<lower = 0> sigma; // hyperparameter of variance
## }
## model {
##   phi ~ sparse_car(tau, alpha, W_sparse, D_sparse, lambda, n, W_n);
##   beta ~ cauchy(0,1);
##   tau ~ gamma(2, 2);
##   sigma ~ cauchy(0,1);
##   y ~ neg_binomial_2_log(X * beta + phi + log_offset,sigma);
## }
## generated quantities{
##   int<lower=0> y_rep[n];
##   y_rep = neg_binomial_2_log_rng(X * beta + phi + log_offset,sigma);
## }
```

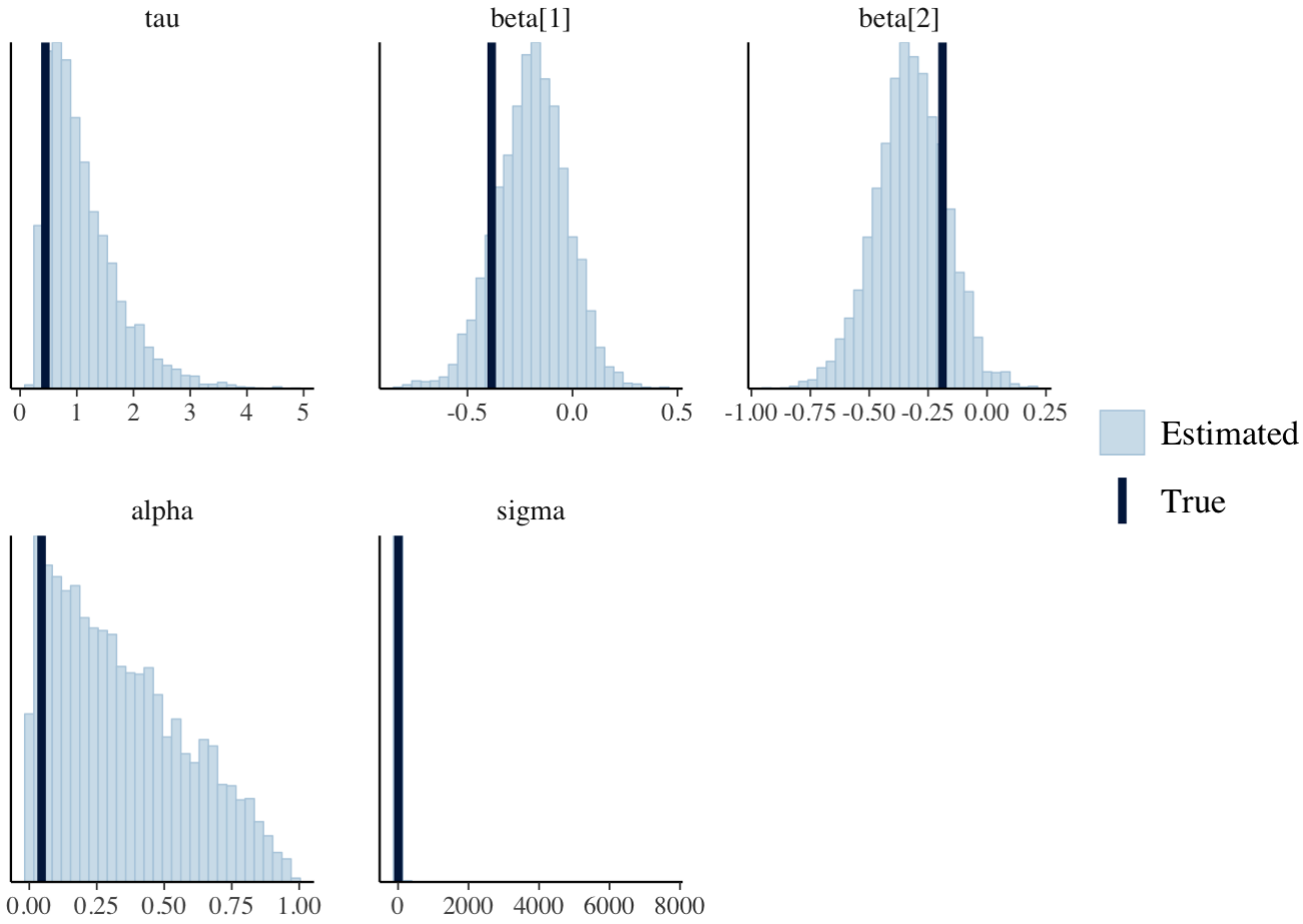
```
library(MASS)
set.seed(123)
# simulate the fake data
## seting the parameters
tau <- rgamma(n=1,2,2)
beta <- rcauchy(n=2,0)
alpha <- runif(n=1,0,1)
sigma <- abs(rcauchy(n=1,1))
true_parameters <- list('tau'=tau,'beta'=beta,'alpha'=alpha,'sigma'=sigma)
true_parameters
```

```
## $tau
## [1] 0.4460468
##
## $beta
## [1] -0.3850032 -0.1892392
##
## $alpha
## [1] 0.0455565
##
## $sigma
## [1] 10.29609
```

```
### simulate the fake data
set.seed(1234)
mu <- rep(0,full_d$n)
D <- diag(apply(full_d$W,1,sum))
prec <- true_parameters$tau * ( D - true_parameters$alpha * full_d$W)
sig <- solve(prec)
phi <- mvrnorm(n=1,mu = mu,Sigma = sig)
lambda <- c()
for (j in 1:full_d$n){
    lambda[j] <- exp(sum(full_d$X[j,] * beta)  + phi[j] + full_d$log_offset[j])
}
fake_y <- sapply(lambda,rnegbin,n=1,theta=sigma)
```

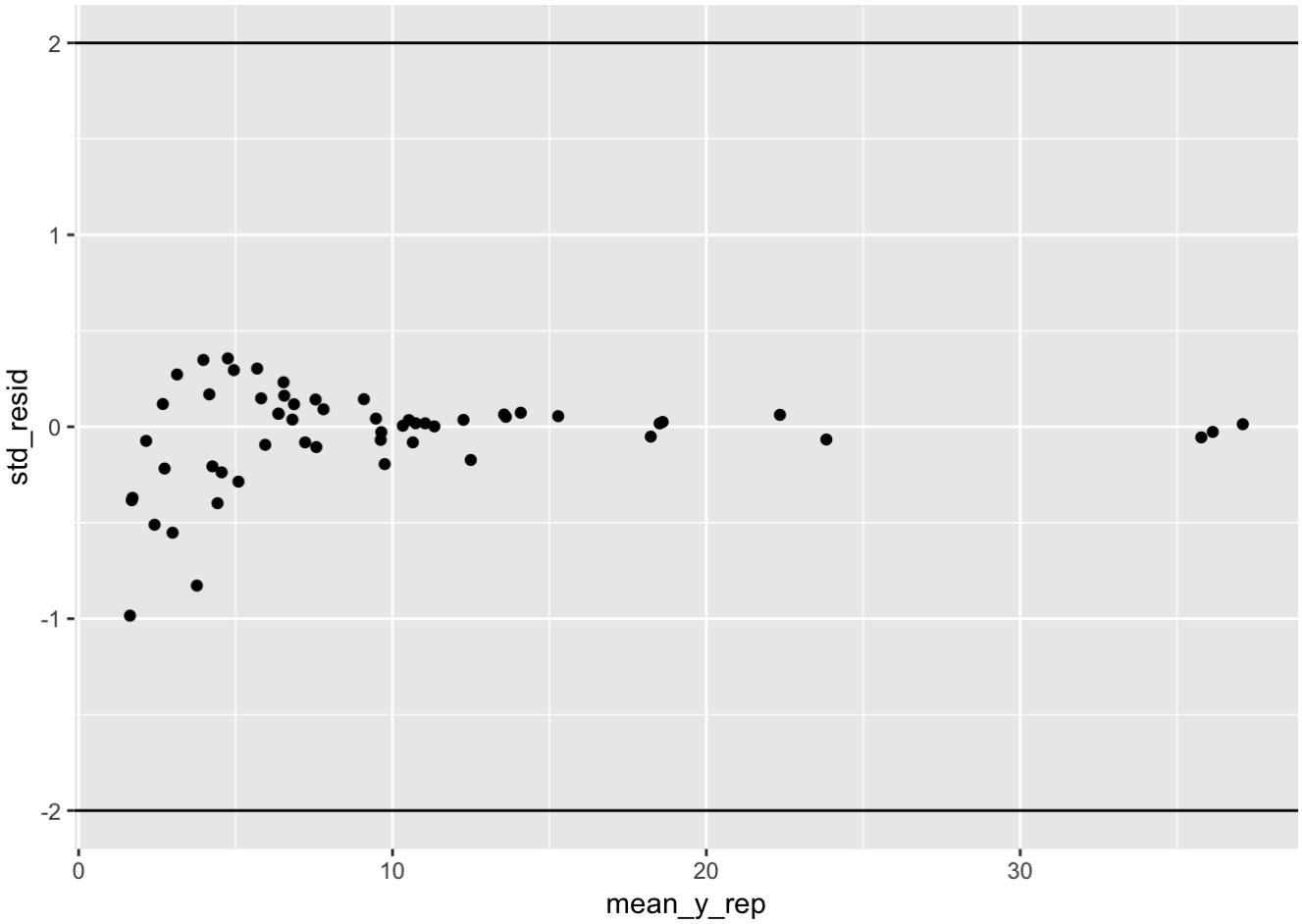
```
comp_model <- stan_model('my.stan')
fake_data <- list(n = nrow(X),          # number of observations
             p = ncol(X),         # number of coefficients
             X = X,               # design matrix
             y = fake_y,               # observed number of cases
             log_offset = log(E), # log(expected) num. cases
             W_n = sum(W) / 2,    # number of neighbor pairs
             W = W)
fit_model_fake <- sampling(comp_model, data = fake_data, seed = 123)
posterior_parameters <- as.matrix(fit_model_fake,pars = c('tau','beta','alpha','sigma'))
mcmc_recover_hist(posterior_parameters,true = c(true_parameters$tau,true_parameters$bet
a,true_parameters$alpha,true_parameters$sigma))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

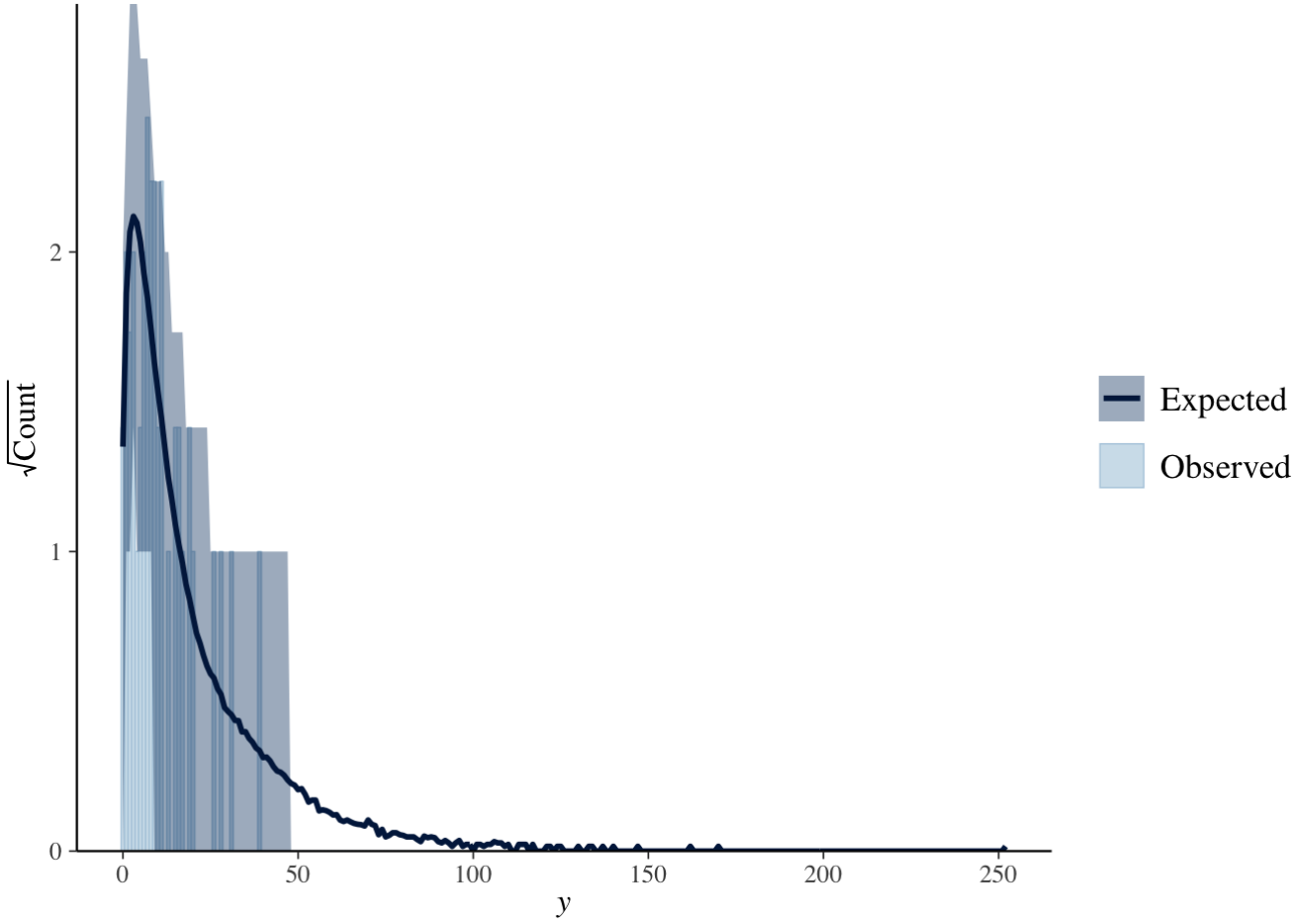



```
real_fit <- stan('my.stan',data=sp_real)
```

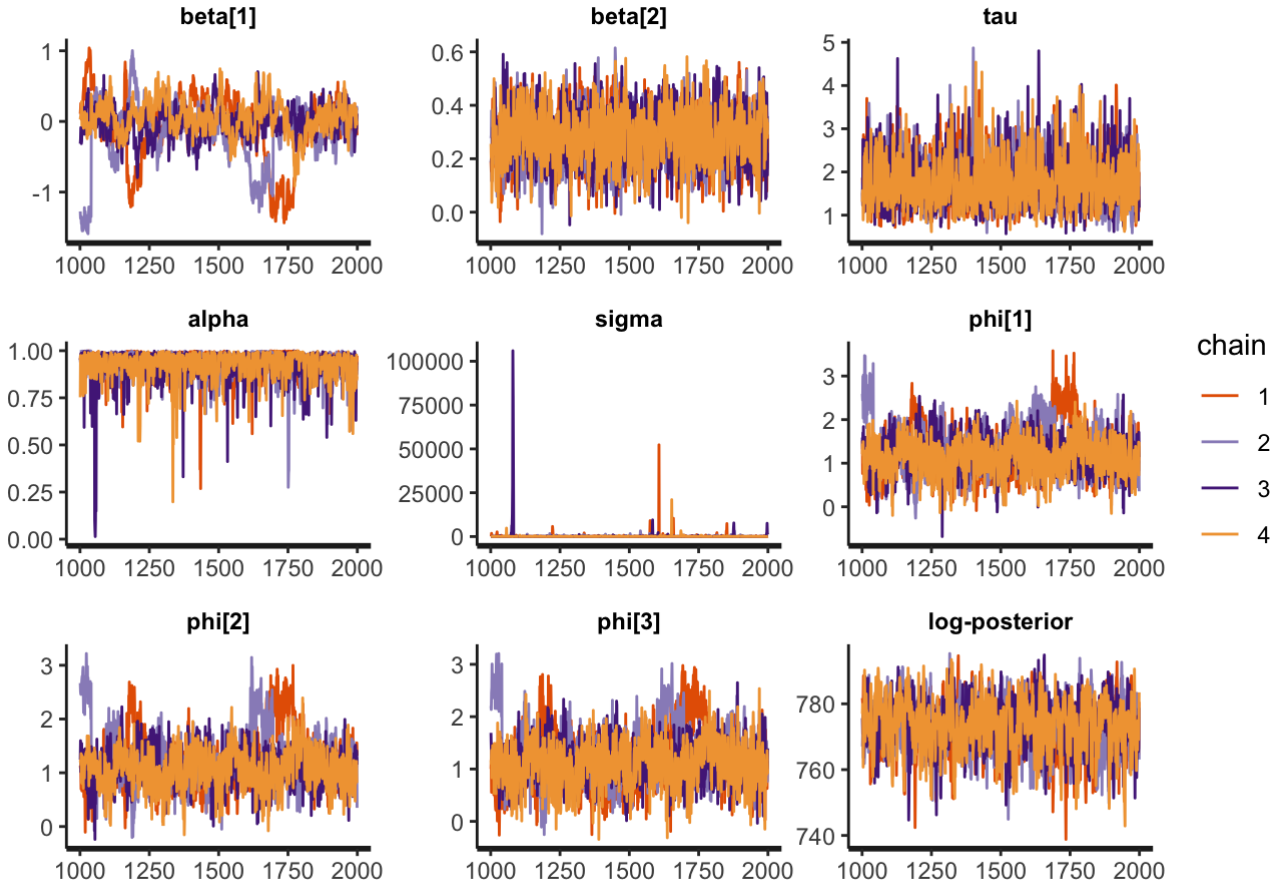
```
sims <- extract(real_fit)
y_rep <- sims$y_rep
lambda <- as.matrix(sp_real$X) %*% as.matrix(apply(sims$beta,2,mean)) + apply(sims$phi,2
,mean) + sp_real$log_offset
mean_y_rep <- colMeans(y_rep)
std_resid_sp <- (sp_real$y - mean_y_rep) / sqrt(mean_y_rep + mean_y_rep^2*lambda)
qplot(mean_y_rep, std_resid) + hline_at(2) + hline_at(-2)
```




```
ppc_rootogram(sp_real$y,yrep = y_rep)
```




```
to_plot <- c('beta', 'tau', 'alpha','sigma', 'phi[1]', 'phi[2]', 'phi[3]', 'lp__')
traceplot(real_fit, pars = to_plot)
```




```
print(real_fit,pars = c('beta', 'tau', 'alpha','sigma', 'phi[1]', 'phi[2]', 'phi[3]', 'l
p__'))
```

```
## Inference for Stan model: my.
## 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]  -0.06    0.05    0.37  -1.10  -0.18   0.00   0.15   0.54    62
## beta[2]   0.28    0.00    0.10   0.08   0.21   0.28   0.34   0.47  1235
## tau       1.76    0.01    0.57   0.90   1.35   1.68   2.08   3.09  1622
## alpha     0.93    0.00    0.08   0.75   0.91   0.95   0.98   1.00   554
## sigma   193.19   72.94 2462.32   5.79  14.36  27.49  61.58 680.70  1140
## phi[1]    1.24    0.05    0.51   0.32   0.91   1.21   1.52   2.38   121
## phi[2]    1.18    0.05    0.46   0.36   0.88   1.13   1.41   2.36    88
## phi[3]    1.19    0.05    0.50   0.25   0.87   1.15   1.47   2.36   106
## lp__    773.36    0.25    7.66 756.64 768.63 773.81 778.68 787.01   958
##         Rhat
## beta[1] 1.05
## beta[2] 1.00
## tau     1.00
## alpha   1.01
## sigma   1.00
## phi[1]  1.03
## phi[2]  1.04
## phi[3]  1.03
## lp__    1.00
##
## Samples were drawn using NUTS(diag_e) at Fri Nov  2 18:11:24 2018.
## 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).
```

As we can see the model fit well. But to be honest, not so good for posterior predective test which I tried but not put here.