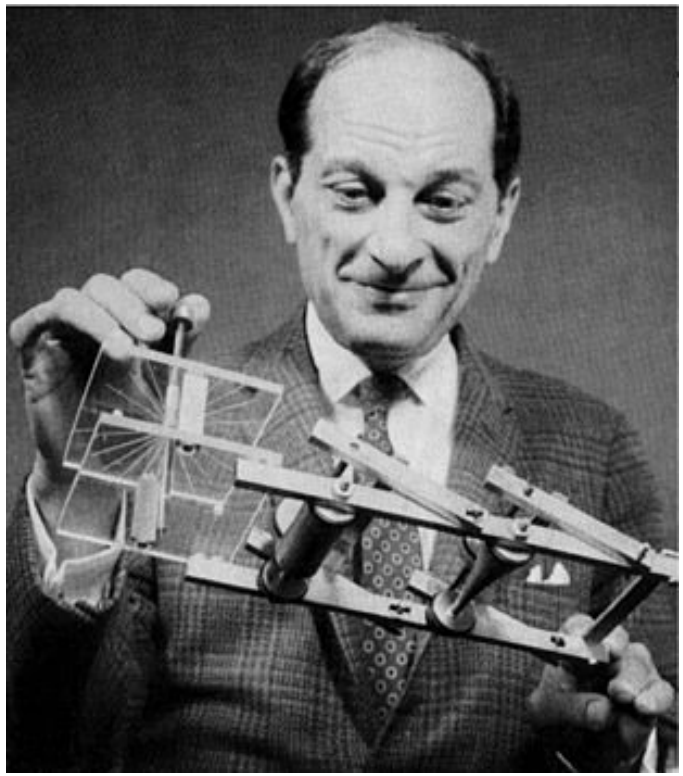
A Short Introduction in Bayesian Modelling Using Stan

Lab 2: Examples of Bayesian Models in Stan

L. Egidi - DEAMS, Units (legidi@units.it)

Thursday 18 March 2021

Origins



Stanislaw Ulam (1909-1984): Manhattan project, H-Bomb experiments in Los Alamos, MCMC father jointly with John von Neumann.

Indice

What is Stan?

- Probabilistic programming language and inference algorithms.
- Stan program
  - declares data and (constrained) parameter variables
  - defines log posterior (or penalized likelihood)
- Stan inference
  - MCMC for full Bayes
  - Variational Bayes for approximate Bayes
  - Optimization for (penalized) MLE
- Stan ecosystem
  - lang, math library (C++)
  - interfaces and tools (R, Python, Julia, many more)
  - documentation (example model repo, ▸ user guide & reference manual, ▸ case studies, R package vignettes)
  - online community (▸ Stan Forums on Discourse)

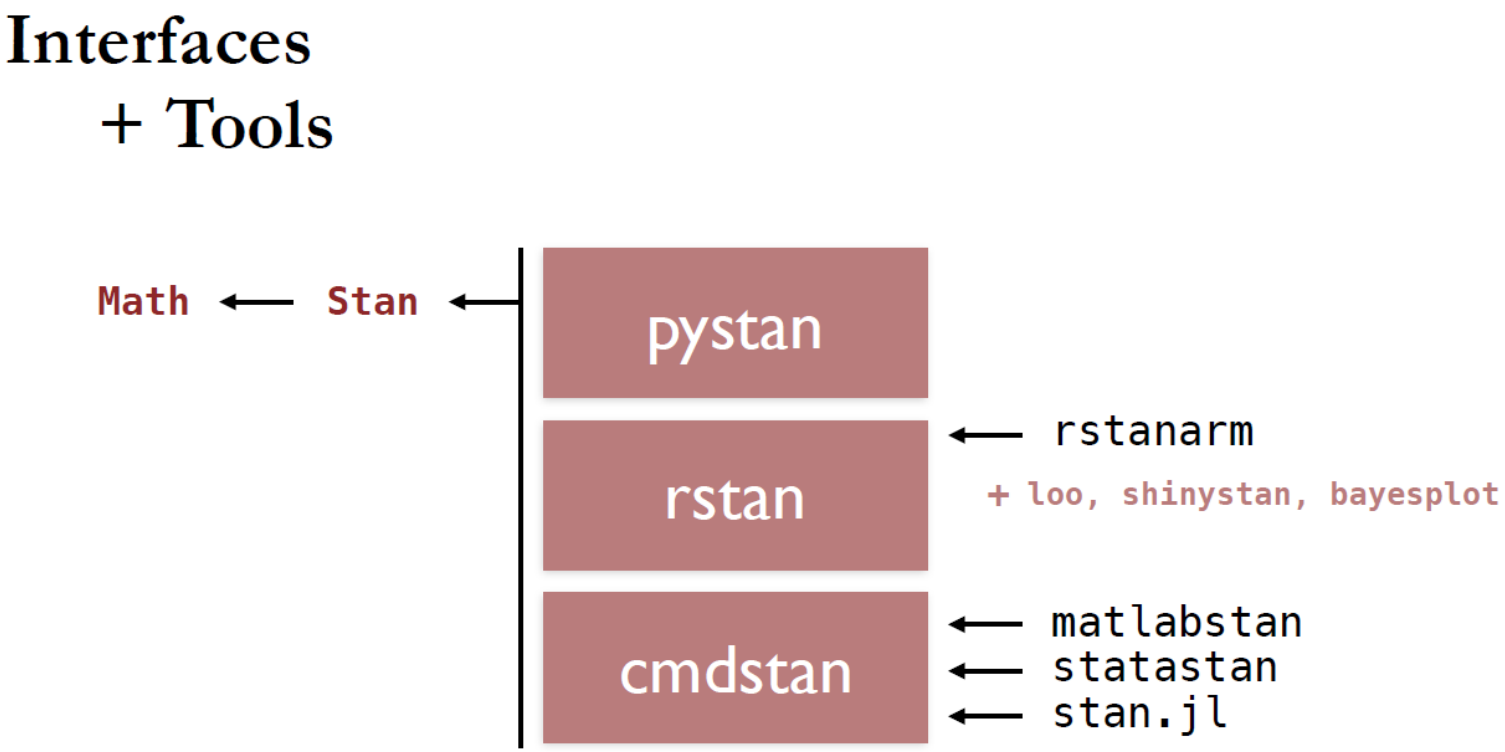
Why Stan?

- Fit rich Bayesian statistical models. Close to the *big data* philosophy.
- Efficiency
  - Hamiltonian Monte Carlo + NUTS
  - Compiled to C++
- Flexible domain specific language
- "Freedom-respecting, open-source"
  - doc & written materials
  - interacting community
  - continuous development
- Interaction with some other R packages designed to explore the Stan output.

Who is using Stan?

- Biological & physical sciences: clinical trials, epidemiology, genomics, population ecology, entomology, ophthalmology, neurology, agriculture, fisheries, cancer biology, astrophysics & cosmology, molecular biology, oceanography, climatology.
- Social sciences: population dynamics. psycholinguistics, social networks, political science, human development, economics.
- Many more: sports analytics, public health, publishing, finance, pharma, actuarial, recommender systems, educational testing, materials engineering.

Interfaces


Interfaces
+ Tools
Math
Stan
pystan
rstan
cmdstan
rstanarm
+ loo, shinystan, bayesplot
matlabstan
statastan
stan.jl

Improving MCMC performance

With Stan, we aim to provide an MCMC implementation that works robustly for as many target distributions as possible

- Gibbs, RW Metropoilis can be very inefficient, hard to diagnose.
- To explore complicated high-dimensional spaces we need to leverage what we know about the geometry of the typical set.
- For such a reason, Stan enjoys **Hamiltonian Monte Carlo**.

The Stan users may use, analyze and interpret HMC outputs as they were *standard* MCMC outputs.

Before starting

What is a Bayesian model?

- Building a Bayesian model forces us to build a model for how the data is generated
- We often think of this as specifying a prior and a likelihood, as if these are two separate things
- They are not!

Generative models

The philosophy behind Stan is to think generatively.

The model is expressed as a joint probability distribution of observed and unobserved variables, which may be decomposed as follows:

$$p(y, \theta) = p(y|\theta)\pi(\theta) \tag{1}$$

The posterior of interest is then proportional to the joint distribution (1):

$$p(\theta|y) \propto p(y|\theta)\pi(\theta) \tag{2}$$

Generative models

A Bayesian modeller commits to to an a priori joint distribution:

$$p(y,\theta) = \underbrace{p(y|\theta)\pi(\theta)}_{Likelihood \times Prior} = \underbrace{\pi(\theta|y)p(y)}_{Posterior \times Marginal \;\; Likelihood} \tag{3}$$

Generative models and vague priors

What is the problem with *vague/diffuse* priors?

- If we use an improper prior, then we do not specify a joint model for our data and parameters.
- More importantly, we do not specify a data generating mechanism $p(y)$.
- By construction, these priors do not regularize inferences, which is quite often a bad idea
- Proper but diffuse is better than .improper but is still often problematic.

Generative models

- If we disallow improper priors, then Bayesian modeling is generative.
- In particular, we have a simple way to simulate from $p(y)$:

$$\theta^\star \sim \pi(\theta) \quad \Longleftrightarrow \quad y^\star \sim p(y)$$
$$\downarrow$$
$$y^\star \sim p(y|\theta^\star)$$

Stan computations

Stan works in logarithmic terms: all the computations are actually done on log-scale. So, for the posterior we have.

$$\log(\pi(\theta|y)) = \log(\pi(\theta)) + \log(p(y|\theta)) + \text{constant} \tag{4}$$

Products become sums of logs:

$$p(y|\theta) = \prod_{i=1}^{n} p(y_i|\theta) \rightarrow \log(p(y|\theta)) = \sum_{i=1}^{n} \log(p(y_i|\theta)).$$

Starting point

We are now going to write a Stan program together:

- Open a new empty file in RStudio
- Save it as `linear_regression.stan`

Blocks strategy

Stan programs are organized into blocks:

- data block: declare data types, sizes, and constraints. Read from data source and constraints validated. Evaluated: once.
- parameters block: declare parameter types, sizes, and constraints. Evaluated: every log prob evaluation.
- transformed parameters block: declare those parameters transformed from the original ones declared in the parameters block. Evaluated: every log prob evaluation.
- model block: statements defining the posterior density in log scale. Evaluated: every log prob evaluation.
- generated quantities: declare and define derived variables. (P)RNGs, predictions, event probabilities, decision making. Constraints validated. Evaluated: once per draw.

Data block

```
data {
  // Dimensions
  int<lower=1> N;
  int<lower=1> K;

  // Variables
  matrix[N, K] X;
  vector[N] y;
}
```

```
// single line comment
/* multiple lines of
comments */
```

Parameters' block

```
parameters {
  real alpha;
  vector[K] beta;
  real<lower=0> sigma;
}
```

constraints *required* in **parameters** block

Model block

```
model {
  // priors  (flat, uniform, if omitted)
  sigma ~ exponential(1);
  alpha ~ normal(0, 10);
  for (k in 1:K) beta[k] ~ normal(0, 5);


  for (n in 1:N) {
    y[n] ~ normal(X[n, ] * beta + alpha, sigma);
  }
}
```

Why is the default automatically uniform?

- $\pi(\theta) \propto 1$ (0 on log scale)
- Nothing added to log prob

Generated quantities block

```
generated quantities {
   vector[N] y_rep;
   for (n in 1:N) {
     real y_hat = X[n,] * beta + alpha;  // local/temp
     y_rep[n] = normal_rng(y_hat, sigma);
   }
}
```

Complete Stan model

```
data {
  int<lower=1>  N;
  int<lower=1>  K;
  matrix[N, K] X;
  vector[N] y;
}
parameters {
  real alpha;
  vector[K] beta;
  real<lower=0> sigma;
}
model {
  sigma ~ exponential(1);
  alpha ~ normal(0, 10);
  for (k in 1:K) beta[k] ~ normal(0, 5);

  for (n in 1:N)
    y[n] ~ normal(alpha + X[n, ] * beta, sigma);
}
generated quantities {
  vector[N] y_rep;
  for (n in 1:N)
    y_rep[n] = normal_rng(alpha + X[n,] * beta, sigma);
}
```

Observed variables

Unobserved variables

$$\log \pi(\theta)$$
$$+$$
$$\log \mathrm{p}(\mathrm{y} \mid \theta)$$

Simulate from generative model

Launching the Stan model from R

Now we may launch the Stan program directly in R:

```
library(rstan)

# passing the data (already stored)
data <- list(N=N, K=K, X=X, y=y)

# fitting the model
fit1 <- stan(
  file = 'linear_regression.stan',
  data = data,
  iter = 2000,
  chains = 4)

# extracting the estimates
sims <- extract(fit1)
```

First example: 8 schools

This example studied coaching effects from eight schools.
We denote with y_{ij} the result of the i-th test in the j-th school. We assume the following model:

$$y_{ij} \sim \mathcal{N}(\theta_j, \sigma_y^2)$$
$$\theta_j \sim \mathcal{N}(\mu, \tau^2)$$

Do some schools perform better/worse according to these coaching effects?
Here is the data, already aggregated by schools:

```
schools_dat <- list(J = 8,
                    y = c(28,  8, -3,  7, -1,  1, 18, 12),
                    sigma = c(15, 10, 16, 11,  9, 11, 10, 18))
```

First Stan model: 8 schools

```
// saved as 8schools.stan
data {
  int<lower=0> J;          // number of schools
  real y[J];               // estimated treatment effects
  real<lower=0> sigma[J]; // standard error of effect estimates
}
parameters {
  real mu;                // population treatment effect
  real<lower=0> tau;      // standard deviation in treatment effects
  vector[J] eta;          // unscaled deviation from mu by school
}
transformed parameters {
  vector[J] theta = mu + tau * eta;    // school treatment effects
}
model {
  eta ~ normal(0,1);           // prior
  y ~ normal(theta, sigma);    //likelihood
}
```

First example: 8 schools

To fit the model and visualize the estimates, it is sufficient to type in R the following commands (with 2000 iterations and 4 chains as a default):

```
fit_8schools <- stan(file = '8schools.stan', data = schools_dat)
print(fit_8schools, pars=c("mu", "tau", "theta"))
```

```
          mean  sd    2.5%  25%   50%   75% 97.5% n_eff Rhat
mu        7.89 5.04   -2.31 4.74  7.92 11.05 18.05  2352  1
tau       6.70 5.71    0.24 2.61  5.43  9.19 21.16  1480  1
theta[1] 11.36 8.23   -2.25 6.18 10.29 15.46 31.15  3161  1
theta[2]  7.89 6.21   -4.43 3.96  7.83 11.78 20.47  4923  1
theta[3]  6.05 7.59 -10.81 1.92  6.56 10.81 20.25  4057  1
theta[4]  7.60 6.44   -5.36 3.74  7.63 11.73 20.57  5055  1
theta[5]  5.13 6.23   -8.45 1.35  5.60  9.26 16.37  4346  1
theta[6]  5.95 6.68   -8.21 1.99  6.30 10.21 18.21  4313  1
theta[7] 10.62 6.93   -1.58 6.12 10.14 14.53 25.63  3381  1
theta[8]  8.40 7.77   -7.18 3.84  8.26 12.63 25.78  3854  1
```

Indice

Posterior graphical analysis with bayesplot

Once we fit a model, it is to vital check it via graphical inspection. The bayesplot package (for any help, see the ▸ vignette) is designed to this task.

The package allows to display:

- Posterior uncertainty intervals
- Univariate marginal posterior distributions
- Bivariate plots
- Trace plots
- Posterior predictive plots

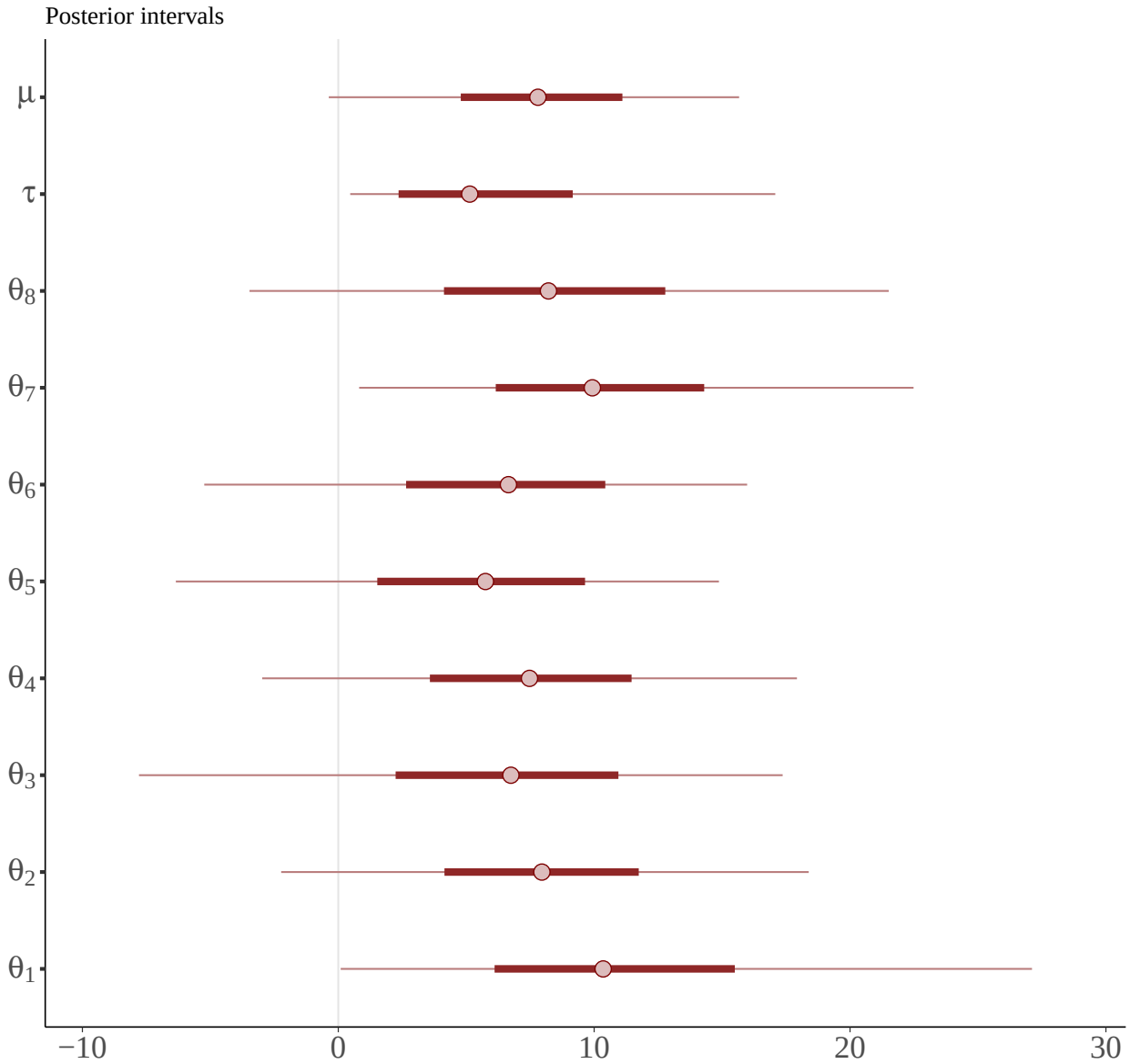
Posterior graphical analysis with bayesplot

The first step is to save the posterior. Then you have many choices:

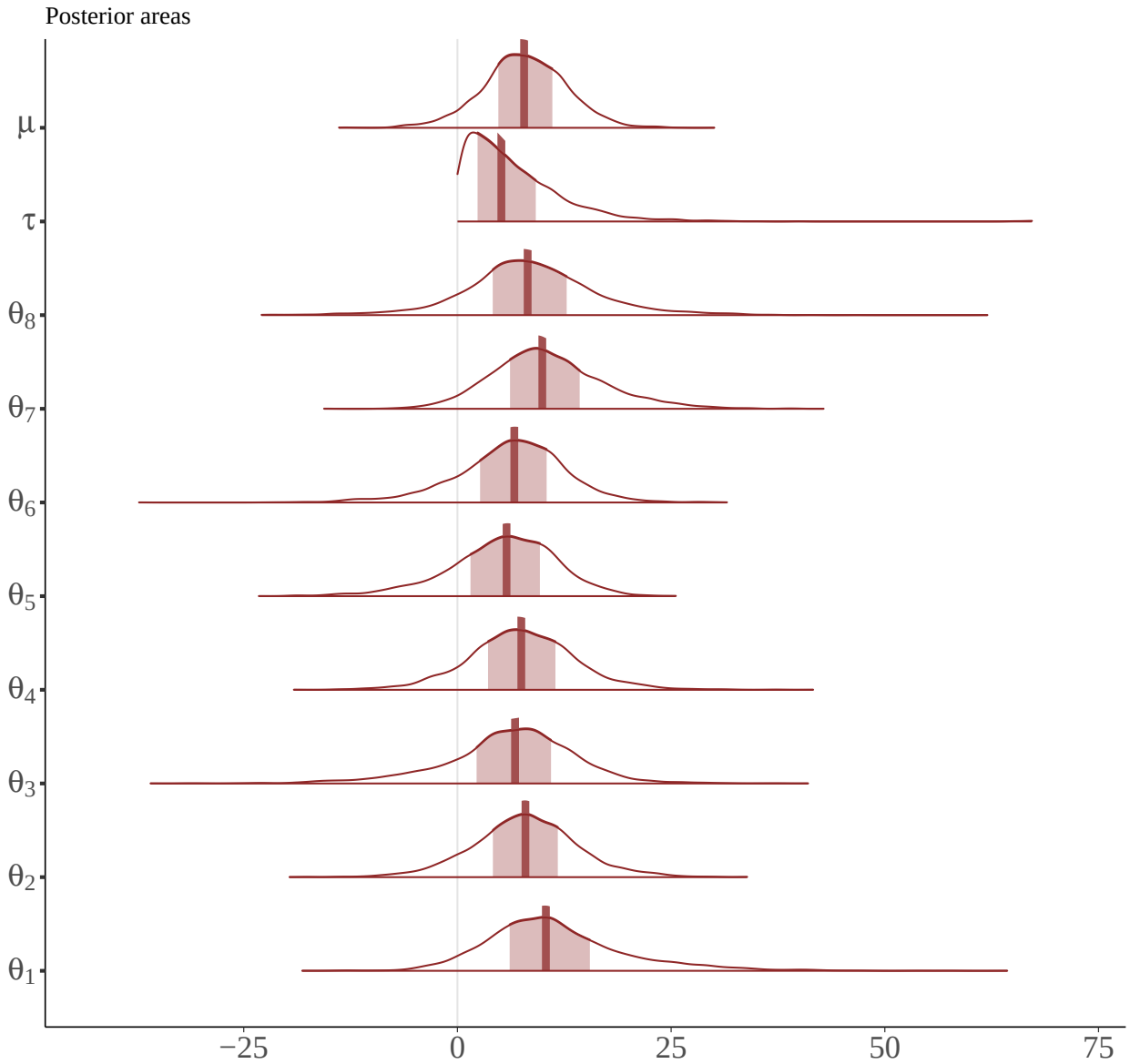
```
library(bayesplot)
posterior <- as.array(fit_8schools)

mcmc_intervals(posterior)      # posterior intervals
mcmc_areas(posterior)          # posterior areas
mcmc_dens(posterior)           # marginal posteriors
mcmc_pairs(posterior)          # bivariate plots
mcmc_trace(posterior)          # trace plots
```

With the arguments `pars` or `regex_pars` you may select the desired parameters.

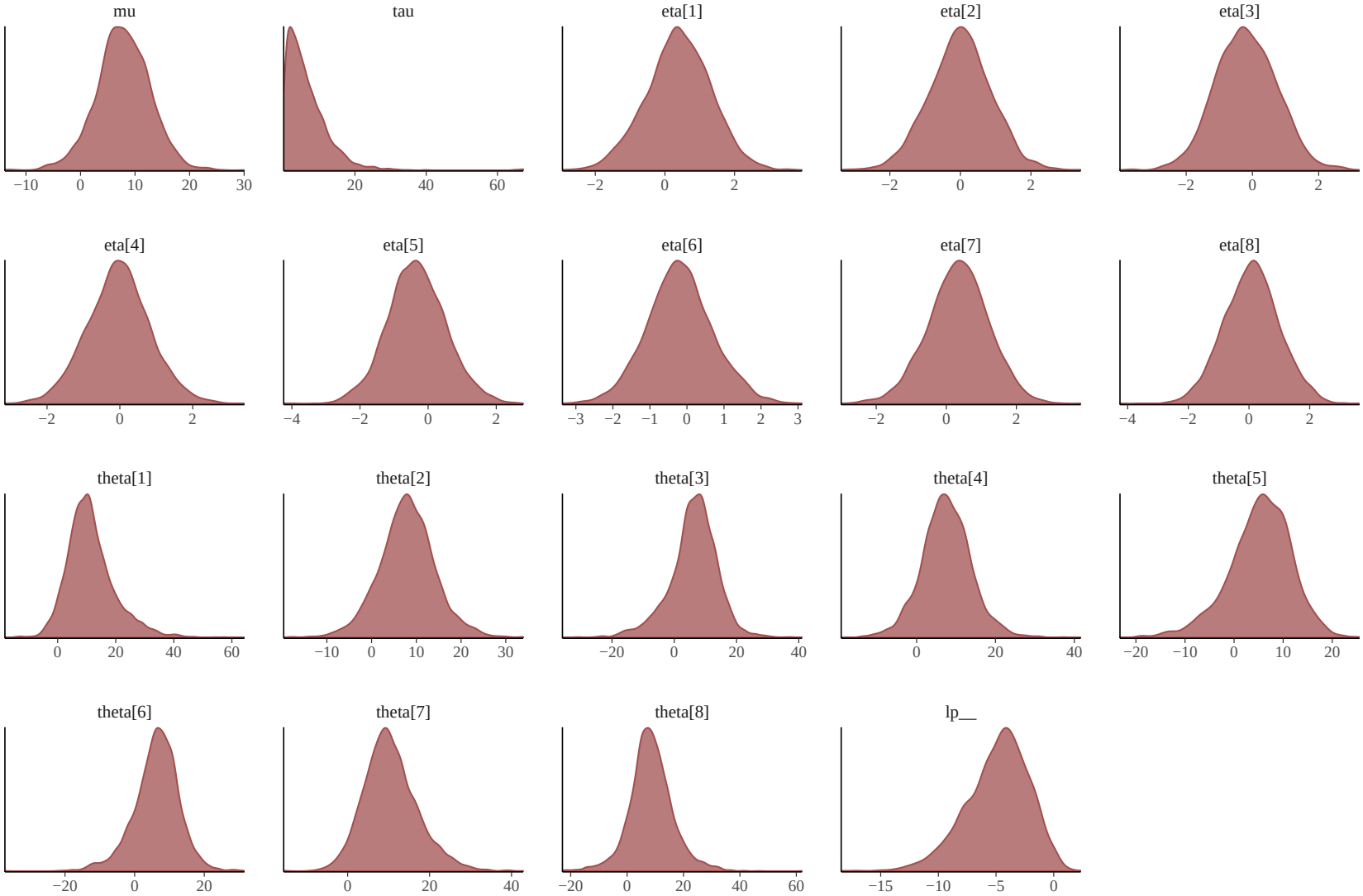
Posterior uncertainty intervals



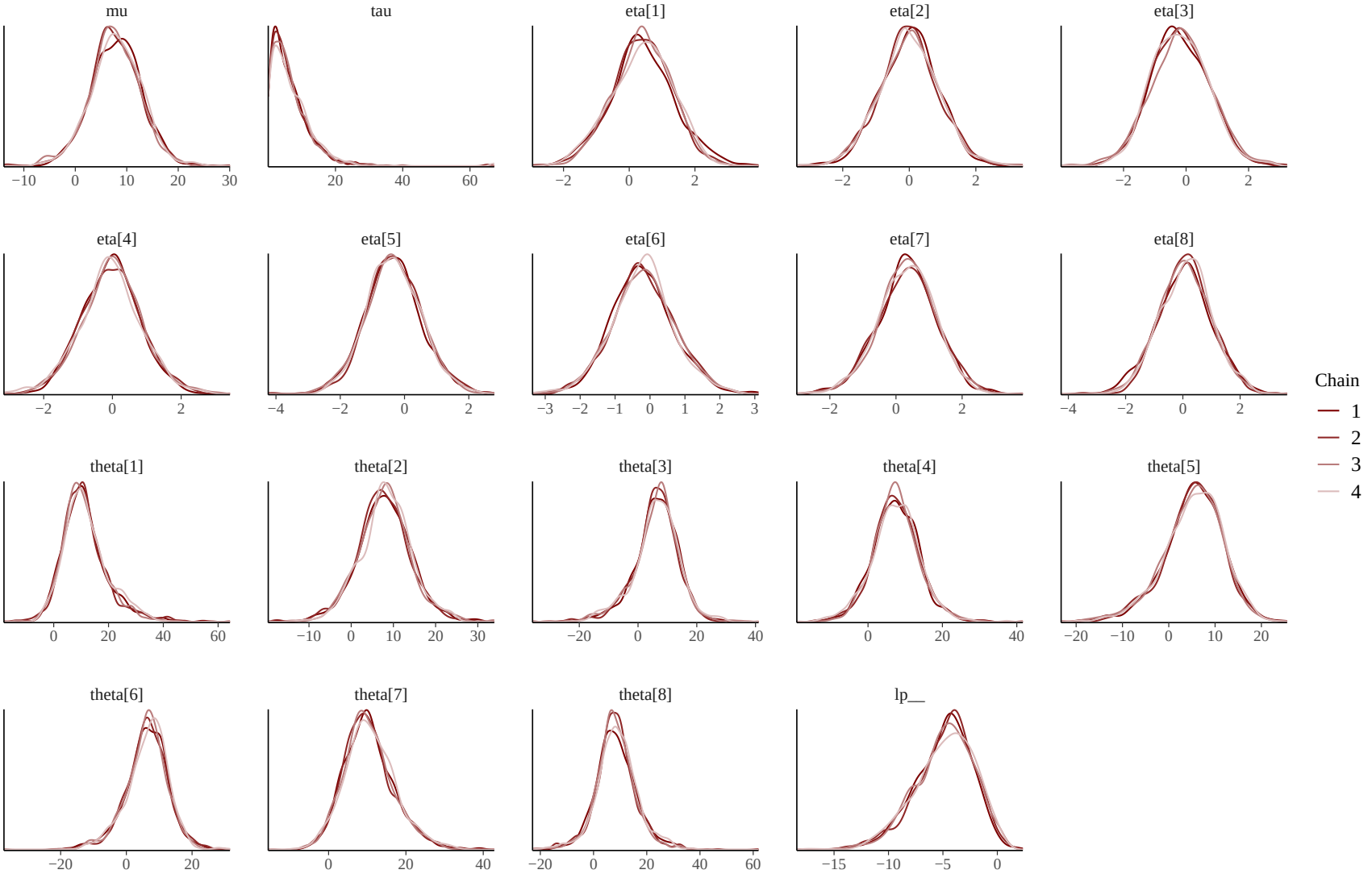
Posterior uncertainty areas



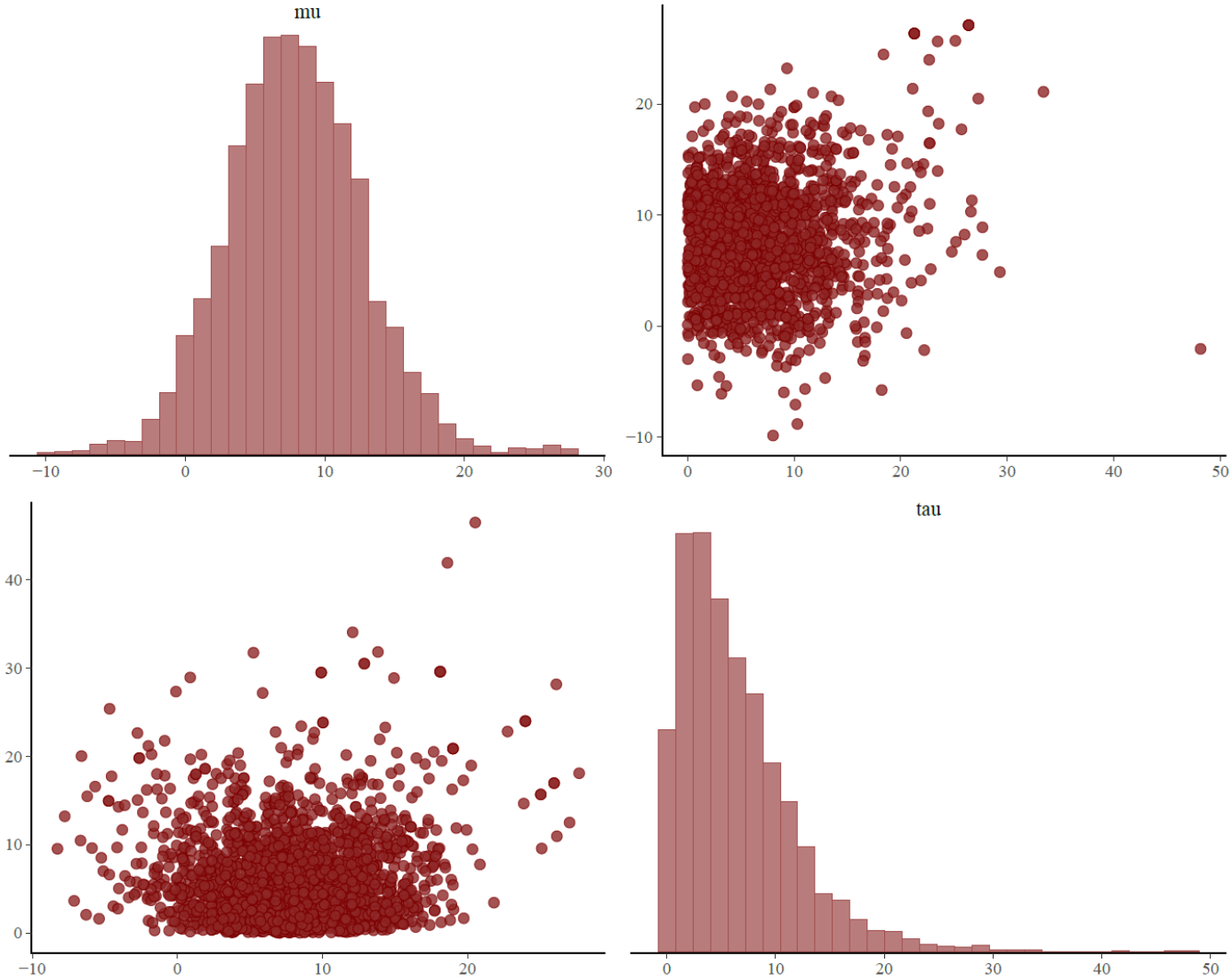
Marginal posteriors



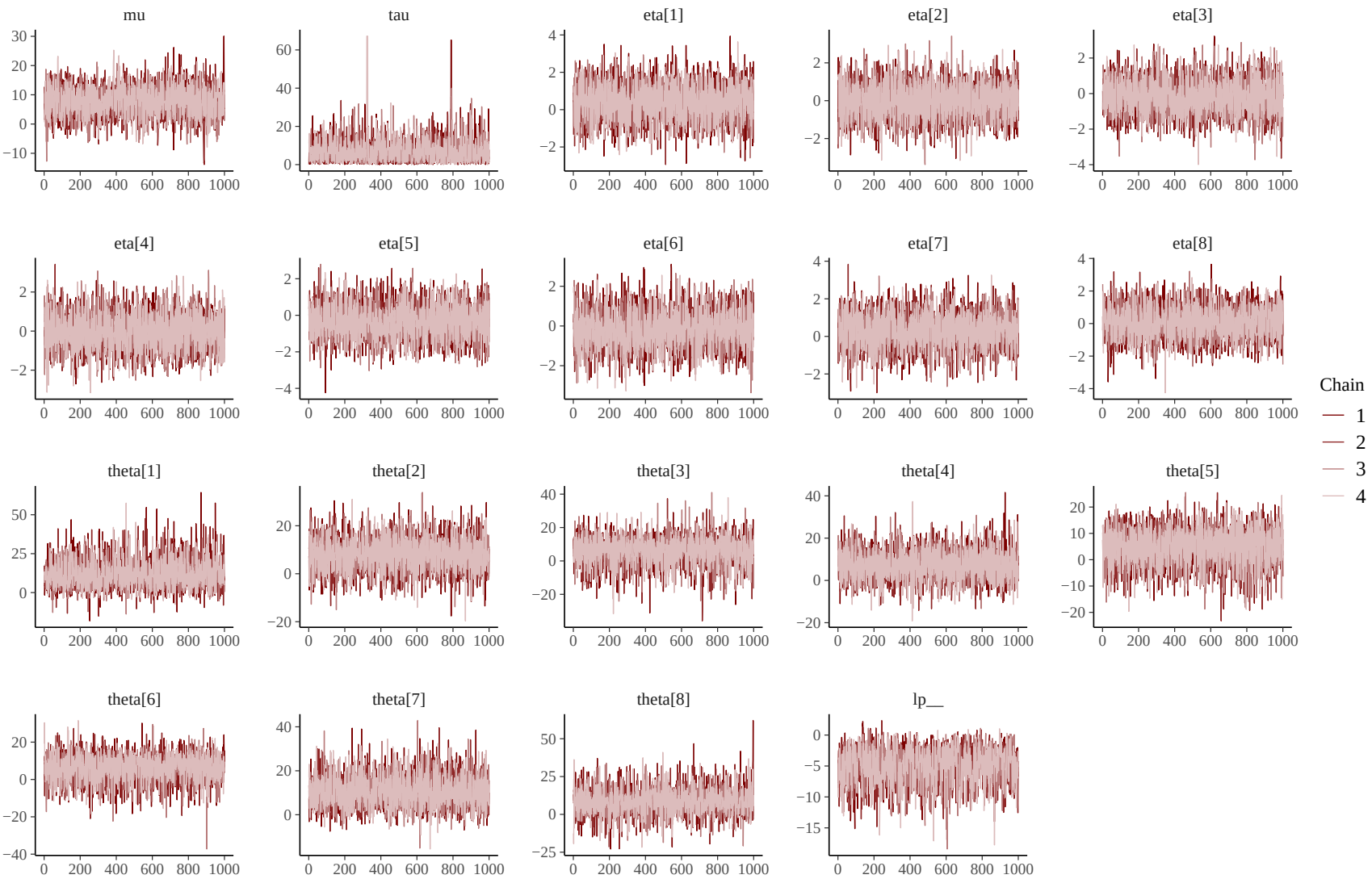
Marginal posteriors separated for each chain



Bivariate posterior plots



Trace plots for the Markov chains



Our challenge with Stan

Main steps in Stan:

- write simple and more complex model in Stan (lm, glm, hierarchical models,...)
- analyze the posterior summaries
- criticize the model and, eventually, change/reparametrize it

Further reading

Further reading:

- Carpenter, B, and Gelman, A, Hoffman, M.D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., Riddell, A. (2017). Stan: A Probabilistic Programming Language, *Journal of statistical software 76(1)*. Here the ▸ pdf

Further optional reading about Hamiltonian Monte Carlo:

- Betancourt, M. (2017) A conceptual introduction to Hamiltonian Monte Carlo. Here the ▸ pdf

Indice

Motivations

Once we have accomplished the first two steps of a Bayesian analysis—constructing a probability model and computing the posterior distribution of all estimands—we should not ignore the relatively easy step to assessing the fit of the model to the data and to our substantive knowledge.

It is worth to remind that we use the term *model* to encompass the sampling distribution, the prior distribution, any hierarchical structure, and issues such as which explanatory variables have been included in a regression.

Motivations

It is not correct to ask 'Is our model true or false?', since probability models in most data analysis will not be perfectly true.

The more relevant question is 'Do the model's deficiencies have a noticeable effect on the substantive inferences?'. Remember the George E.P. Box quote:

All models are wrong, but some are useful.

How to judge when assumptions of convenience can be made safely is a central task of Bayesian sensitivity analysis. Failures in the model lead to practical problems by creating false inferences about estimands of interest.

The external validation paradigm

We can check a model by external validation using the model to make predictions about future/hypothetical data, and then collecting those data and comparing to their predictions.

Bayesian analysis use *posterior predictive checking* to check the joint posterior predictive distribution of future data given the data at hand, $p(\tilde{y}|y)$.

The idea is the following: if the model fits, then replicated data under the model should look similar to observed data. To put in another way, the observed data should look *plausible* under the posterior predictive distribution.

Posterior predictive checking

The basic technique for checking the fit of a model is to draw simulated values from the joint posterior predictive distribution of replicated data and compare these samples to the observed data. Any systematic differences between the simulation and the data indicate potential failings of the model.

We define y^{rep} as the replicated data that *could have been observed*. We distinguish between y^{rep} and $\tilde{y}$:

- $\tilde{y}$ is any future observable value or vector of observable quantities (out-of-sample replication)
- y^{rep} is specifically a replication just like y (in-sample replication)

Posterior predictive checking

The posterior predictive distribution of y^{rep} given the current state of knowledge is:

$$p(y^{\text{rep}}|y) = \int p(y^{\text{rep}}|\theta)\pi(\theta|y)d\theta. \tag{5}$$

We measure the *discrepancy* between model and data by defining some test quantities $T(y, \theta)$, the aspects of the data we wish to check. T is a scalar summary of parameters and data that is used to compare data to predictive simulations.

Test (or discrepancy) quantities play the role in Bayesian model checking that test statistics play in classical testing.

Eight schools: model checking

Consider again the eight schools example about the effects of special coaching programs on test scores in each of eight high-schools:

$$y_{ij} \sim \mathcal{N}(\theta_j, \sigma_y^2)$$
$$\theta_j \sim \mathcal{N}(\mu, \tau^2)$$

The example is based on many assumptions:

1. normality of the estimates y_j given θ_j and σ_j, where the σ_j are assumed known;
2. exchangeability of the prior distribution of the θ_j's;
3. normality of the prior distribution of each θ_j given μ and τ.

The exchangeability assumption means that we will let the data tell us about the relative ordering and similarity of effects in the eight schools.

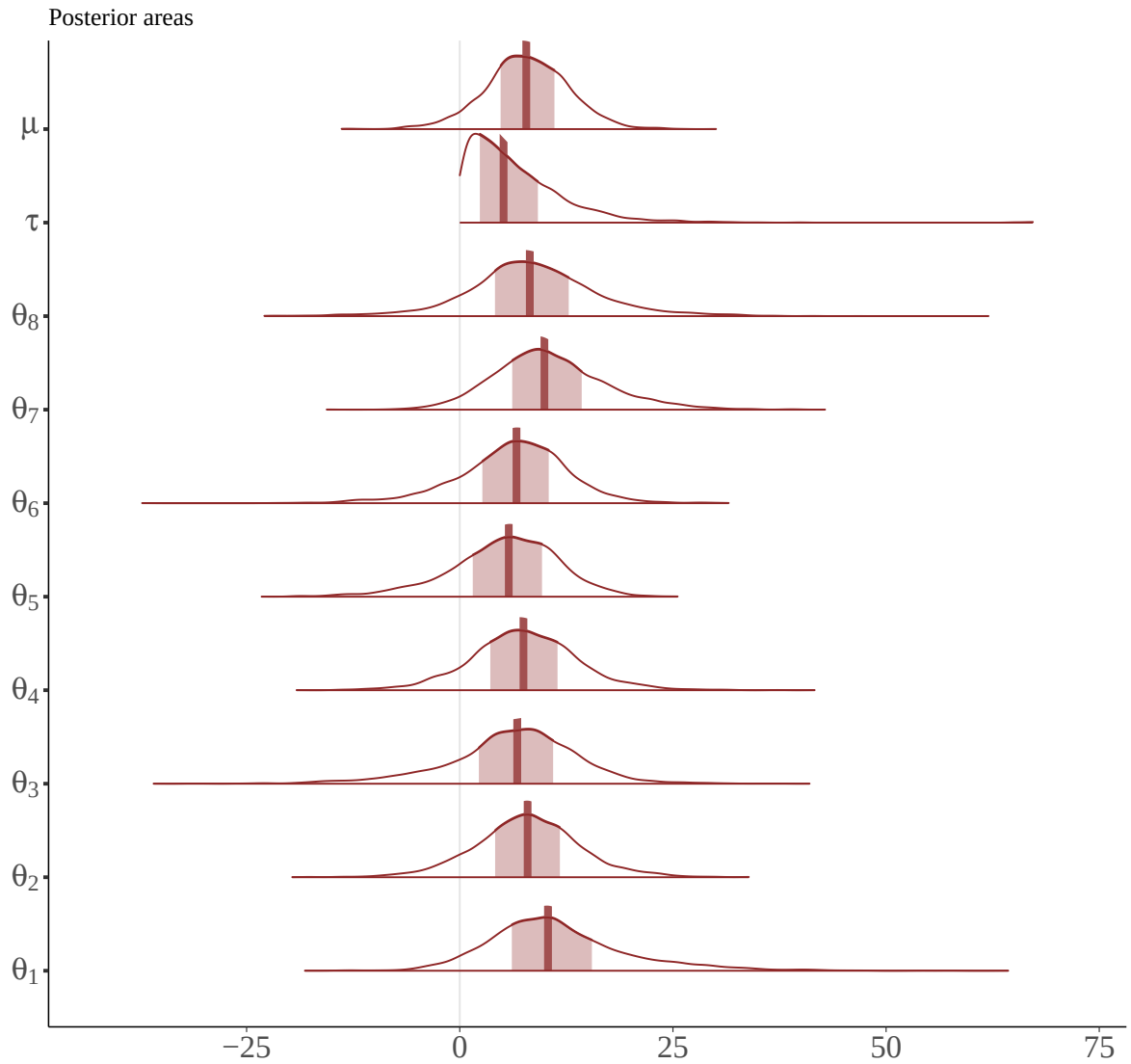
Eight schools: Stan model

```
data {
  int<lower=0> J; // number of schools
  real y[J]; // estimated treatment effects
  real<lower=0> sigma[J]; // s.e. of effect estimates
}
parameters {
  real mu;
  real<lower=0> tau;
  real eta[J];
}
transformed parameters {
  real theta[J];
  for (j in 1:J)
    theta[j] = mu + tau * eta[j];
}
```

Eight schools: Stan model (cont.)

```
model {
  target += normal_lpdf(eta | 0, 1);
  target += normal_lpdf(y | theta, sigma);
}
generated quantities {
  real y_rep[J];
  for (j in 1:J)
     y_rep[j] = normal_rng(theta[j], sigma[j]);
}
```

Eight schools: estimation



Replications

We simulate the ppd of a hypothetical replication of the experiment. In Stan, we do this by the cycled instruction:

```
y_rep[j] = normal_rng(theta[j], sigma[j]);
```

We have now S draws for the replicated vector $y^{\text{rep}} = (y_1^{\text{rep}}, \ldots, y_8^{\text{rep}})$. We should now visualize this distribution over the S draws and detect eventual deficiencies of the model.

We will perform many kinds of pp checks. The main tool here is visualization. All the plots are obtained with the `bayesplot` package.

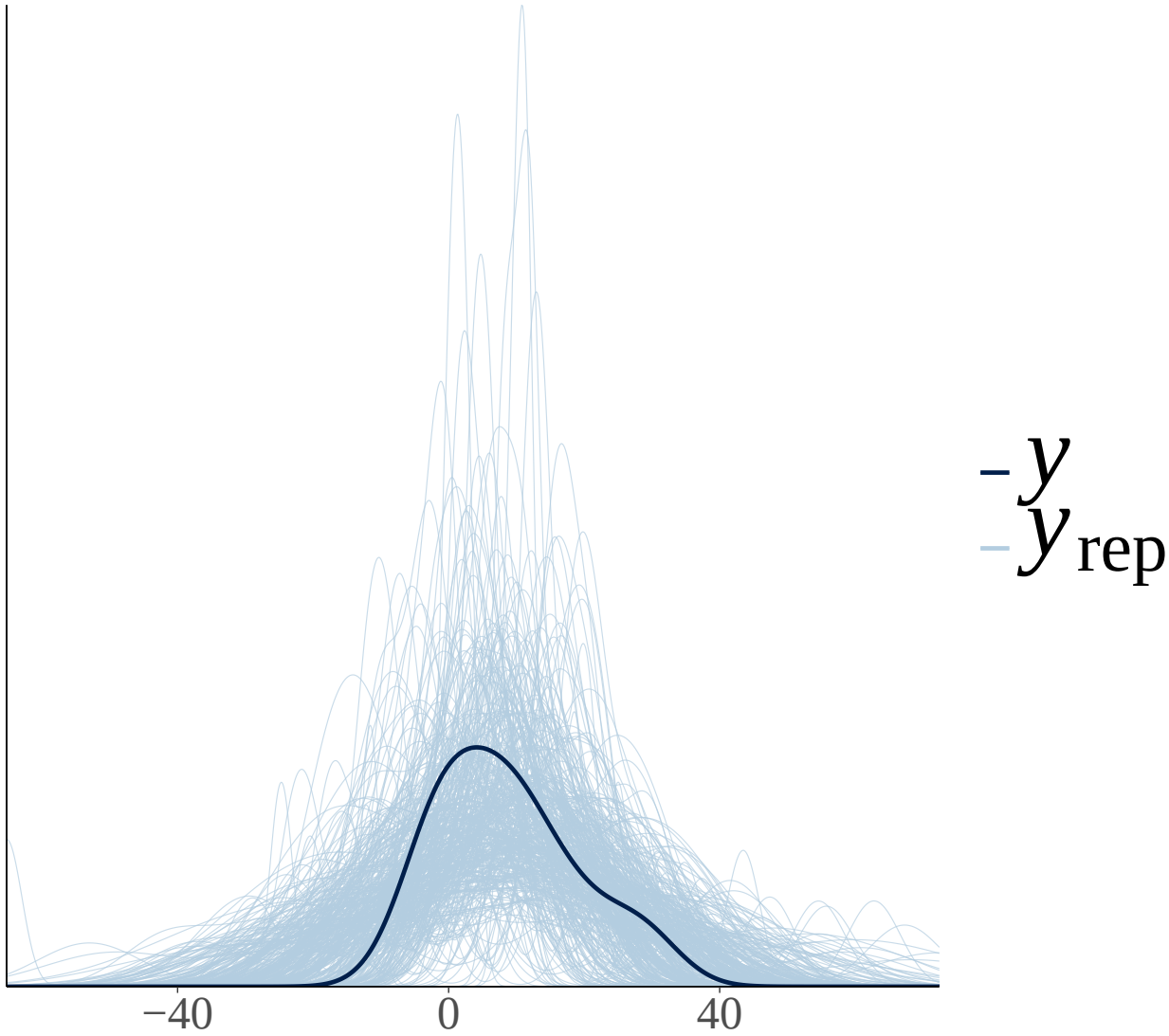
Graphical posterior predictive checks

The basic idea of graphical model checking is to *display the data alongside simulated data* from the fitted model, and to look for systematic discrepancies between real and simulated data. Essentially, we may recognize three kinds of graphical display:

- direct display of all the data
- display of data summaries or parameter inferences
- graphs of residuals or other measures of discrepancy between model and data.

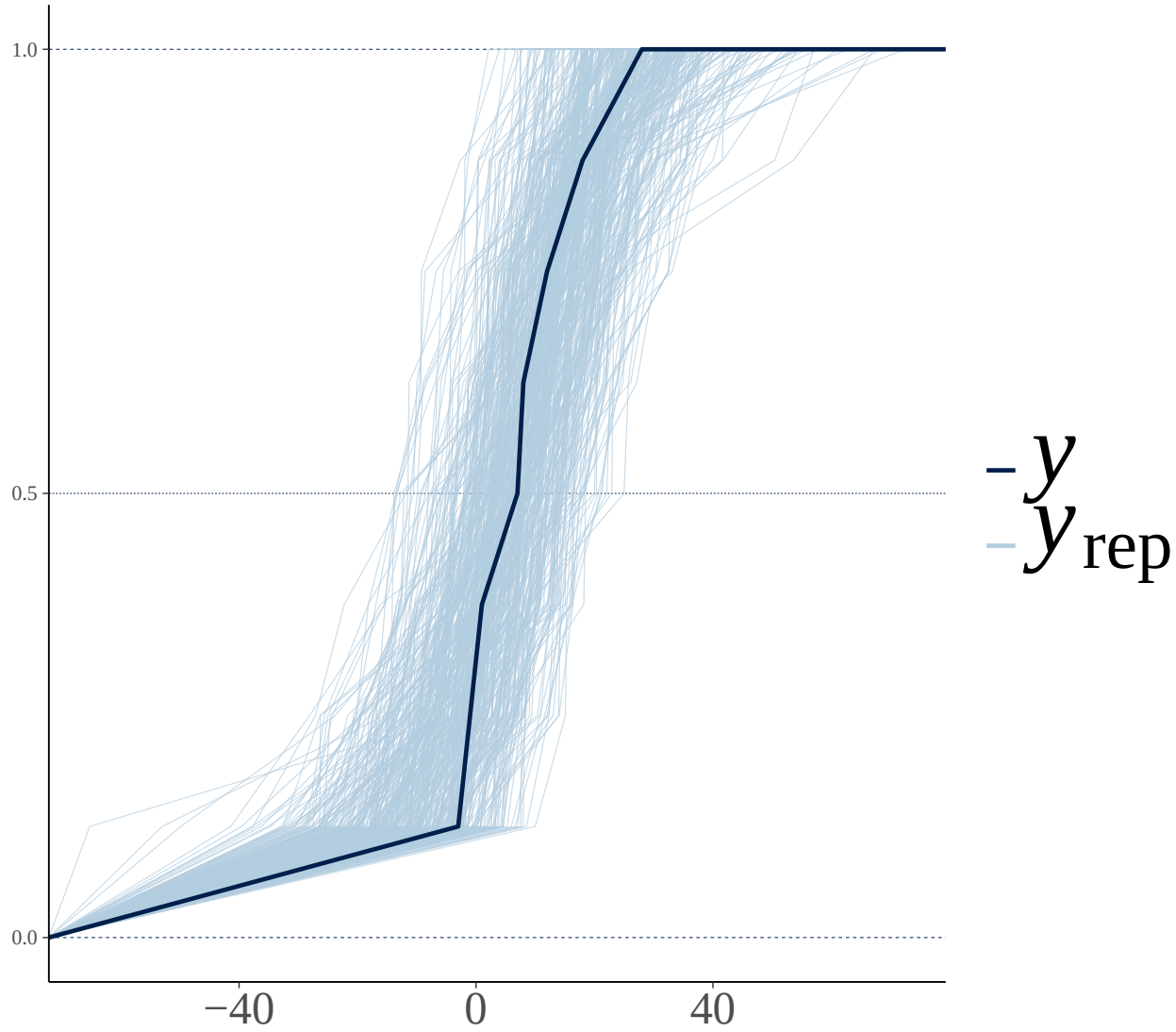
Check 1: distribution of replicated data vs real data

```
ppc_dens_overlay(y,y_rep)
```



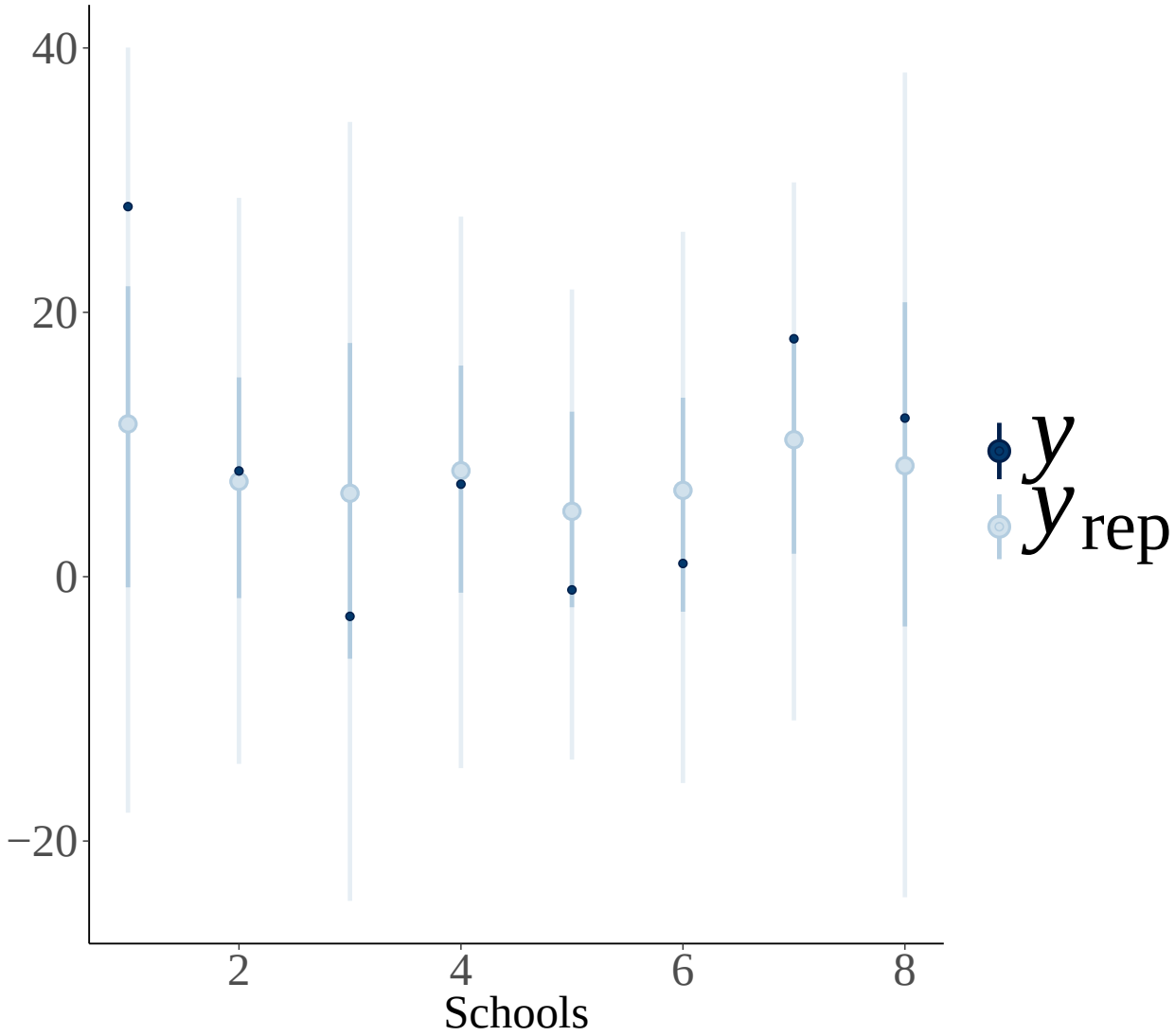
Check 2: empirical distribution function

```
ppc_ecdf_overlay(y,y_rep)
```



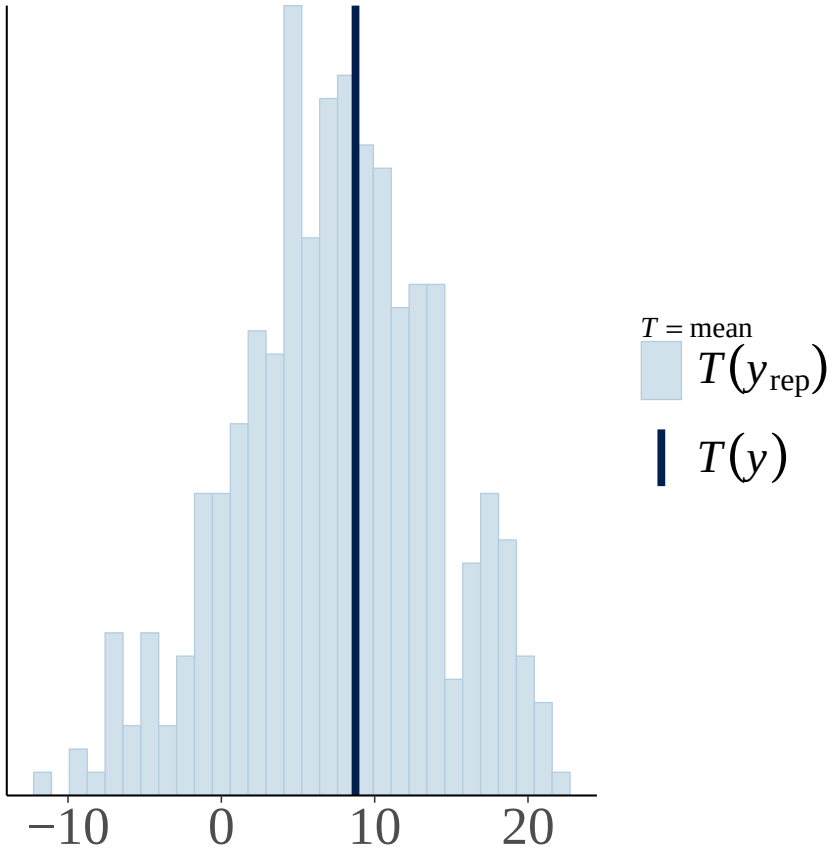
Check 3: predictive intervals vs observed values

```
ppc_intervals(y,y_rep)
```

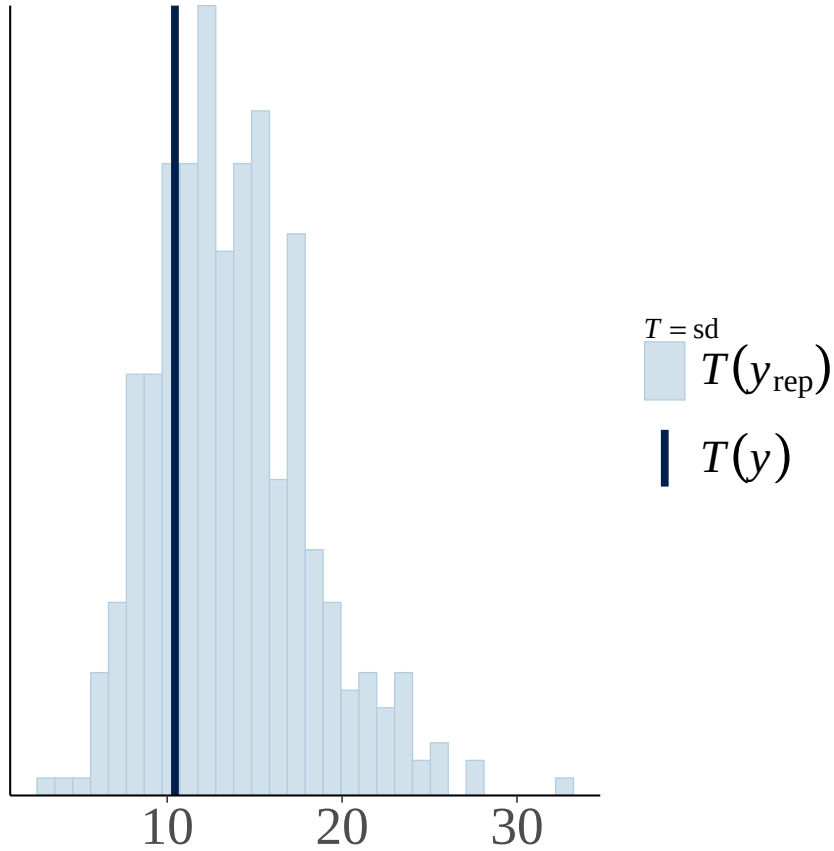


Check 4: statistics

```
ppc_stat(y,y_rep)
```



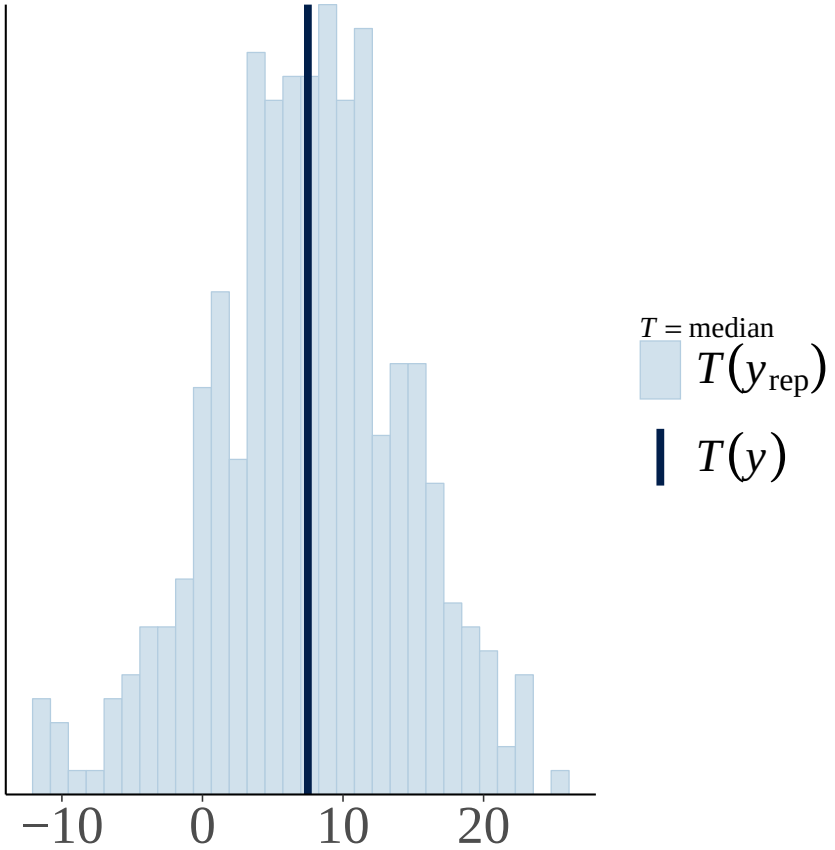
(a) $T(y) = \bar{y}$



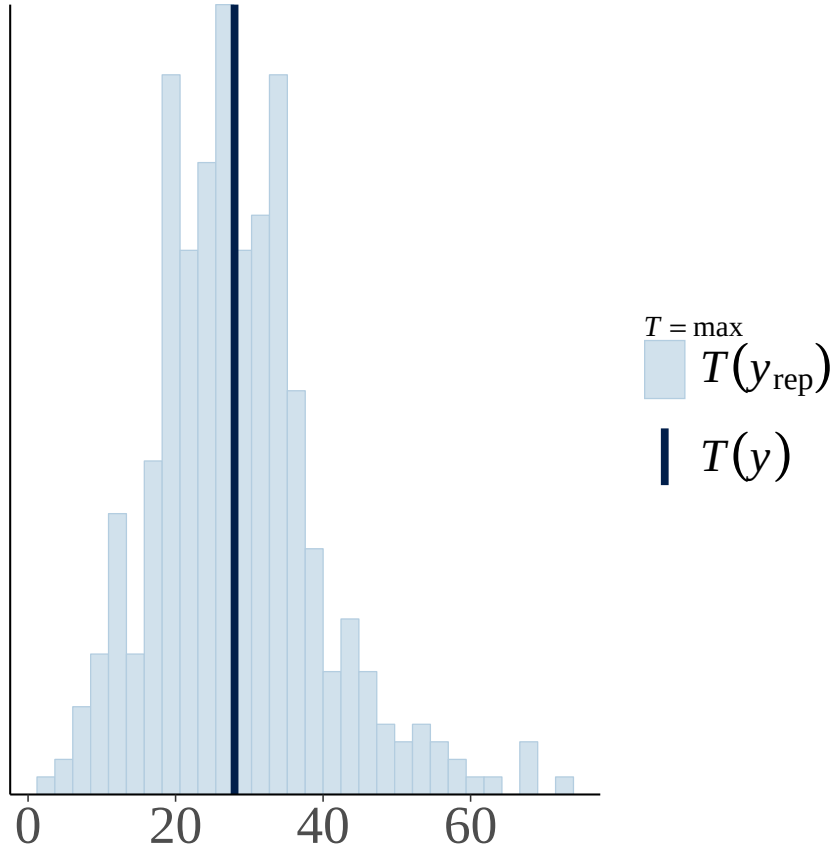
(b) $T(y) = \mathrm{sd}(y)$

Check 4: statistics

```
ppc_stat(y,y_rep)
```



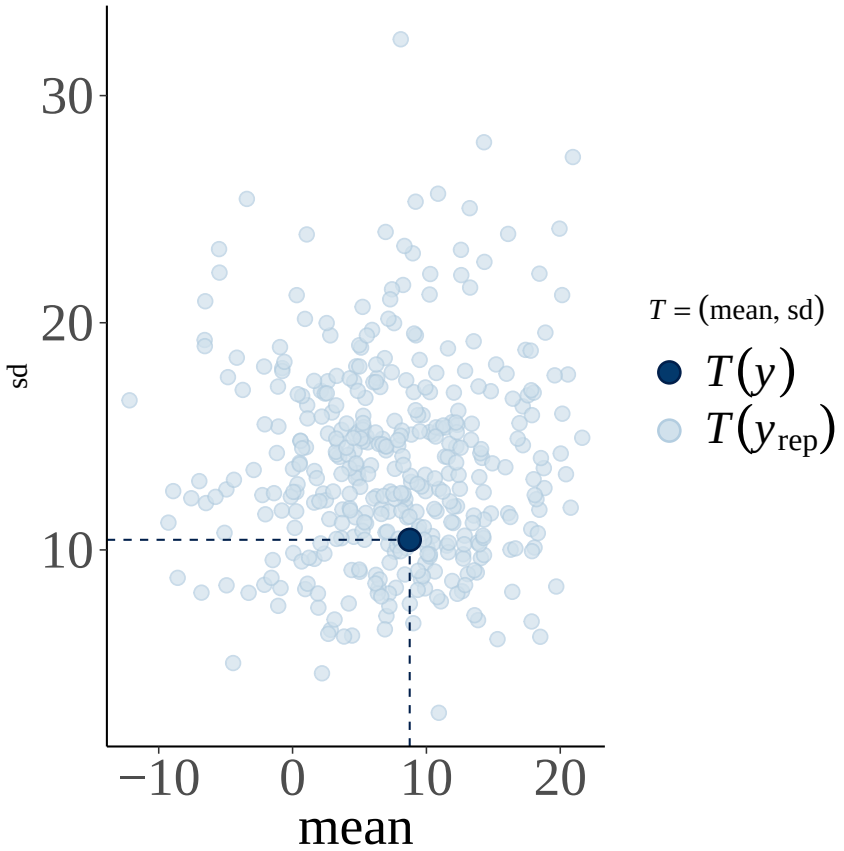
(c) $T(y) = \text{Me}(y)$



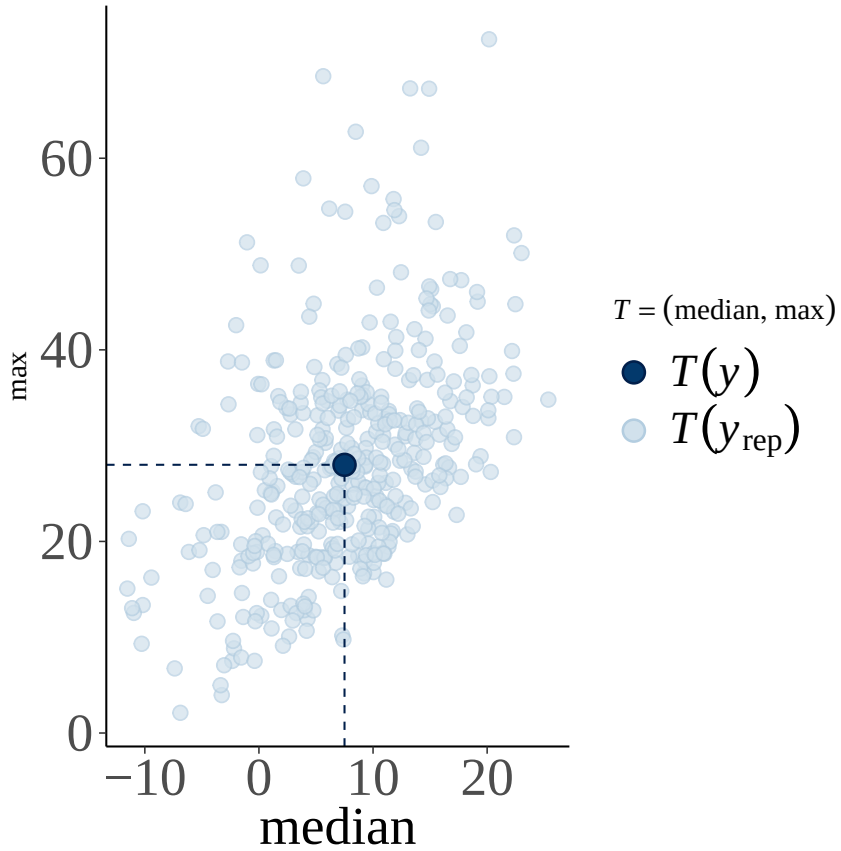
(d) $T(y) = \max(y)$

Check 5: bivariate statistics

```
ppc_stat_2d(y,y_rep)
```



(e) Mean and sd



(f) Median and max

Comments for the pp checks

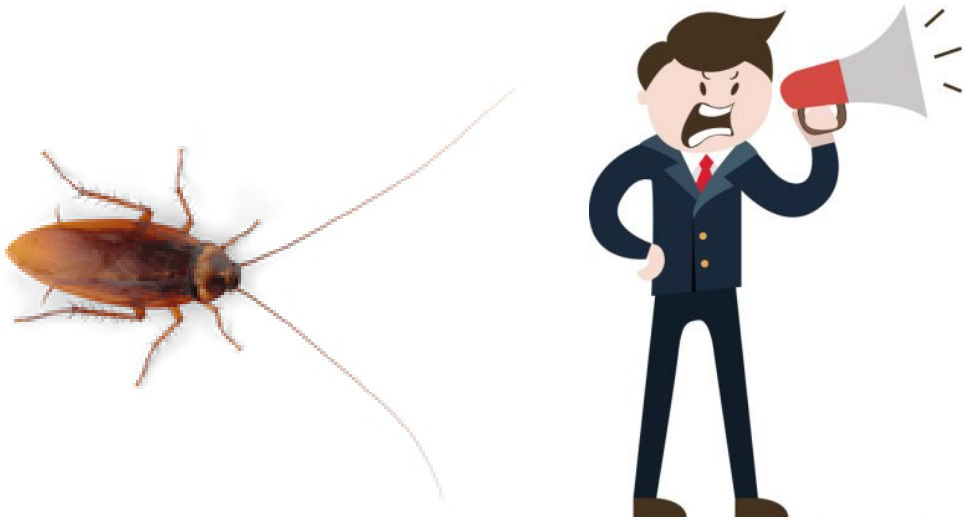
- The graphical summaries suggest that the model generates predicted results similar to the observed data in the study. Observed test statistics fall within their replicated distributions (Check 4), and the distribution of the data is coherent with the replicated ones (Check 1 and 2).
- As a further measure of discrepancy, we may compute the estimated Bayesian p-value from check 4: in each of the four considered statistics, $p_B \approx 0.5$. Remember that a model is suspect if p_B is close to 0 or 1. If a p-value is close to 0 or 1, it is not so important exactly how extreme it is!

Indice

Discrete data regression: cockroaches data

Cockroaches data

A company that owns many residential buildings throughout New York City tells that they are concerned about the number of cockroach complaints that they receive from their 10 buildings. They provide you some data collected in an entire year for each of the buildings and ask you to build a model for predicting the number of complaints over the next months.



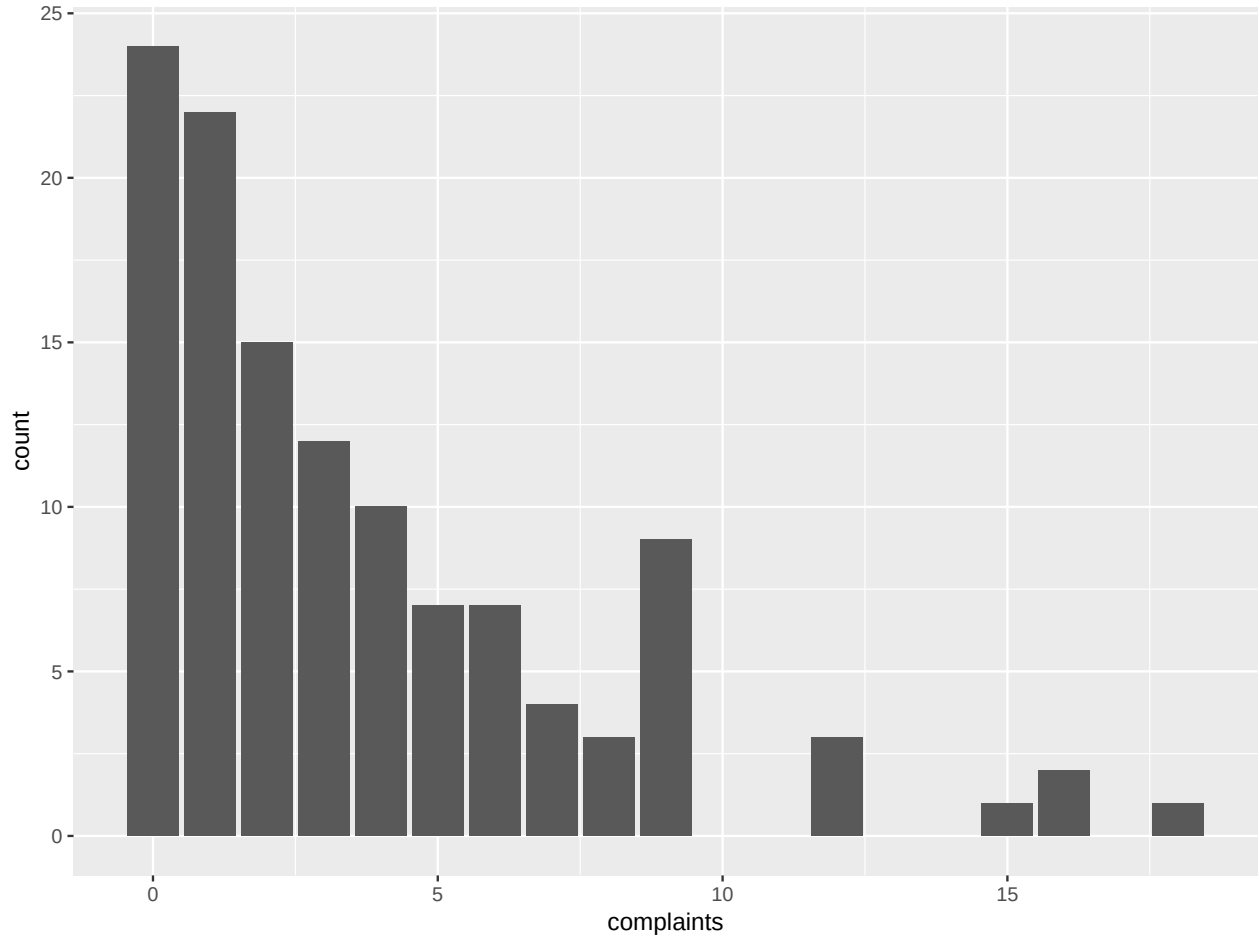
Discrete data regression: cockroaches data

We have access to the following fields (`pest_data.RDS`):

- `complaints`: Number of complaints per building in the current month
- `traps`: The number of traps used per month per building
- `live_in_super`: An indicator for whether the building has a live-in super
- `age_of_building`: The age of the building
- `total_sq_foot`: The total square footage of the building
- `average_tenant_age`: The average age of the tenants per building
- `monthly_average_rent`: The average monthly rent per building
- `floors`: The number of floors per building

Discrete data regression: cockroaches data

Let's make some plots of the raw data, such as the distribution of the complaints:



Poisson regression: cockroaches data

A common way of modeling this sort of skewed, single bounded count data is as a Poisson random variable. For simplicity, we will start assuming:

- ungrouped data, with no building distinction
- no time-trend structures

We use the number bait stations placed in the building, denoted below as `traps`, as explanatory variable. This model assumes that the mean and variance of the outcome variable `complaints` (number of complaints) is the same. For the i-th complaint, $i = 1, \ldots, n$, we have

$$\begin{aligned} \text{complaints}_i &\sim \text{Poisson}(\lambda_i) \\ \lambda_i &= \exp(\eta_i) \\ \eta_i &= \alpha + \beta \, \text{traps}_i \end{aligned}$$

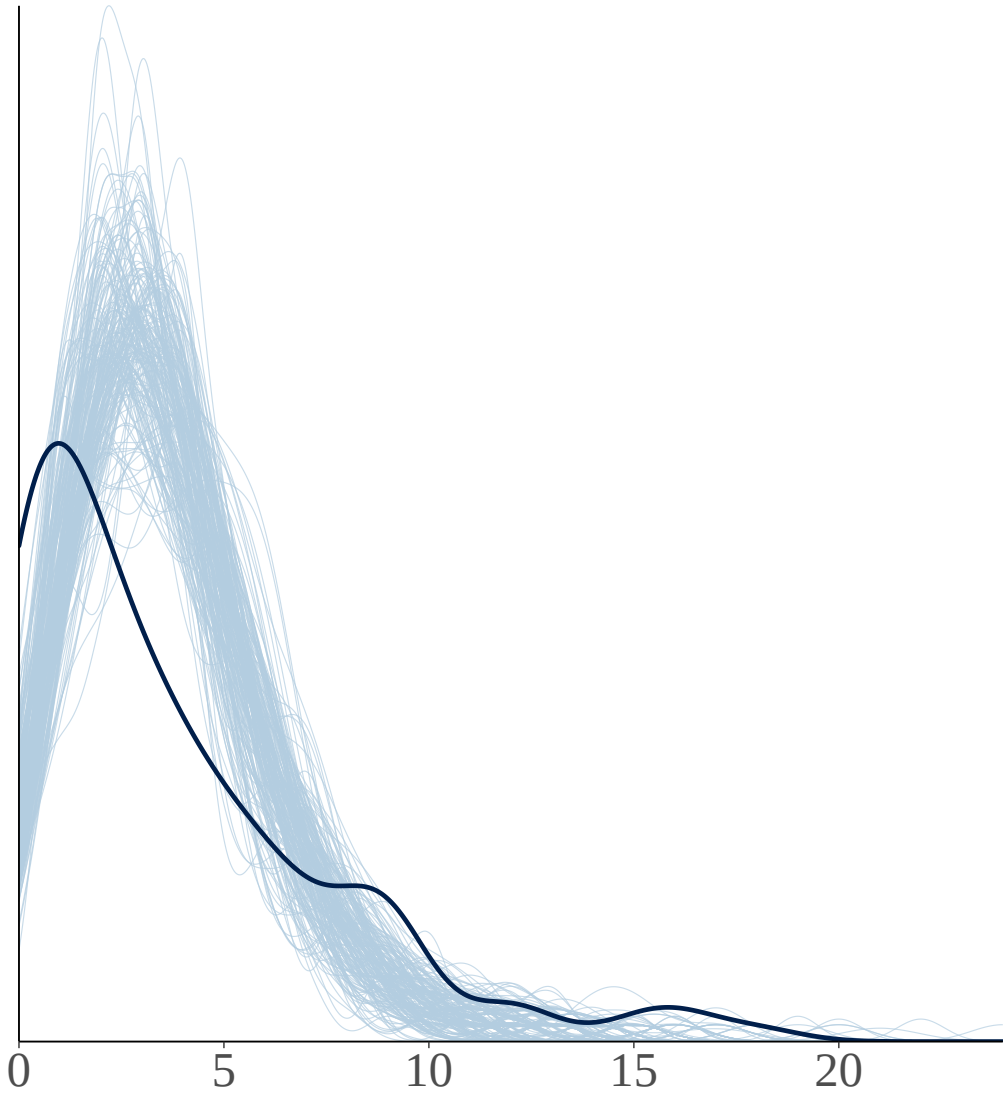
Pest control example

We fit the model in Stan and we obtain the following posterior estimates (R output):

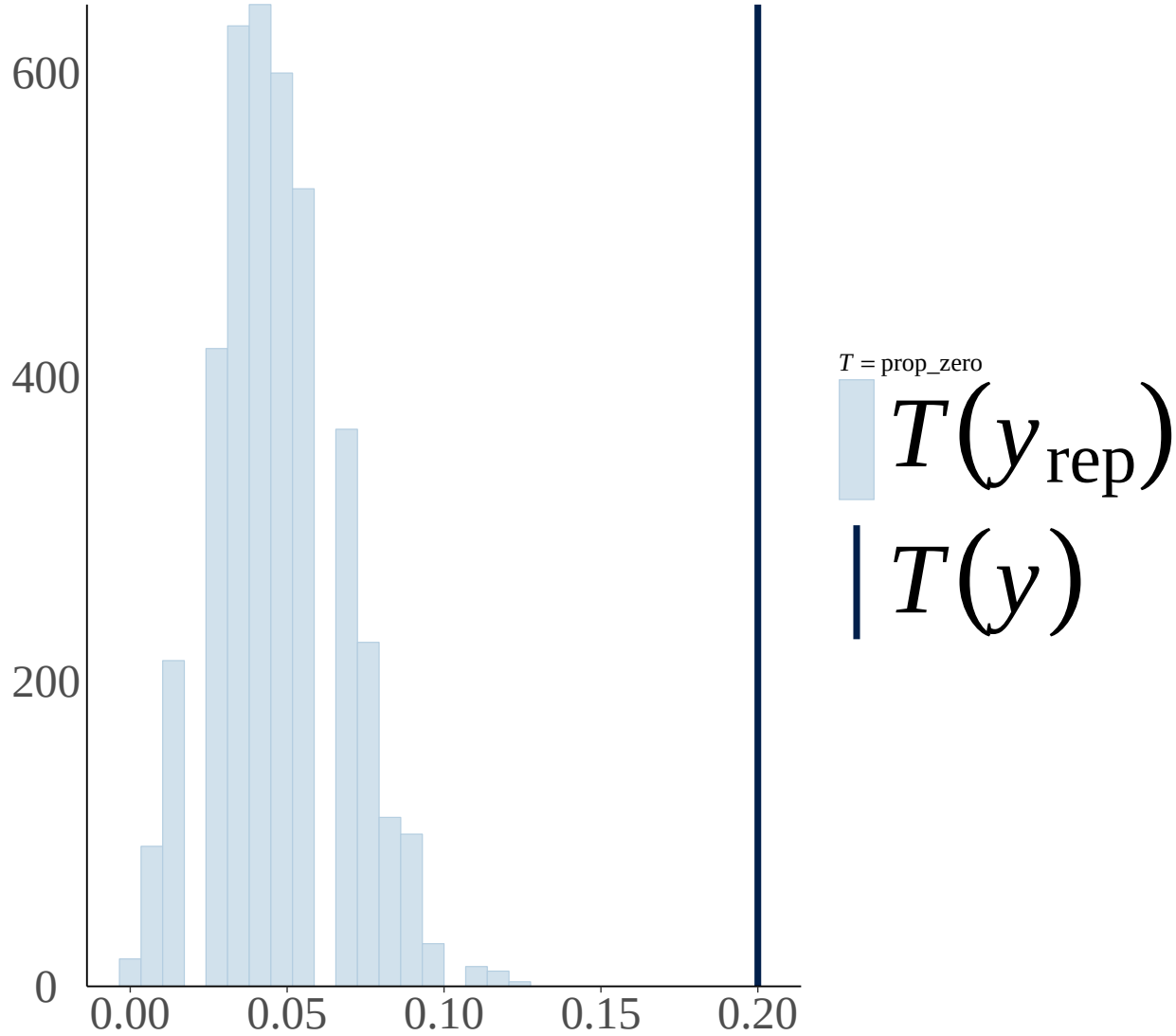
	mean	sd	2.5%	25%	50%	75%	97.5%	n_eff	Rhat
alpha	2.58	0.15	2.28	2.48	2.58	2.69	2.88	979	1
beta	-0.19	0.02	-0.24	-0.21	-0.19	-0.18	-0.15	997	1

Pest control: pp check. Densities

We check the model via some simulated data:



Pest control: pp check. Proportion of zeros

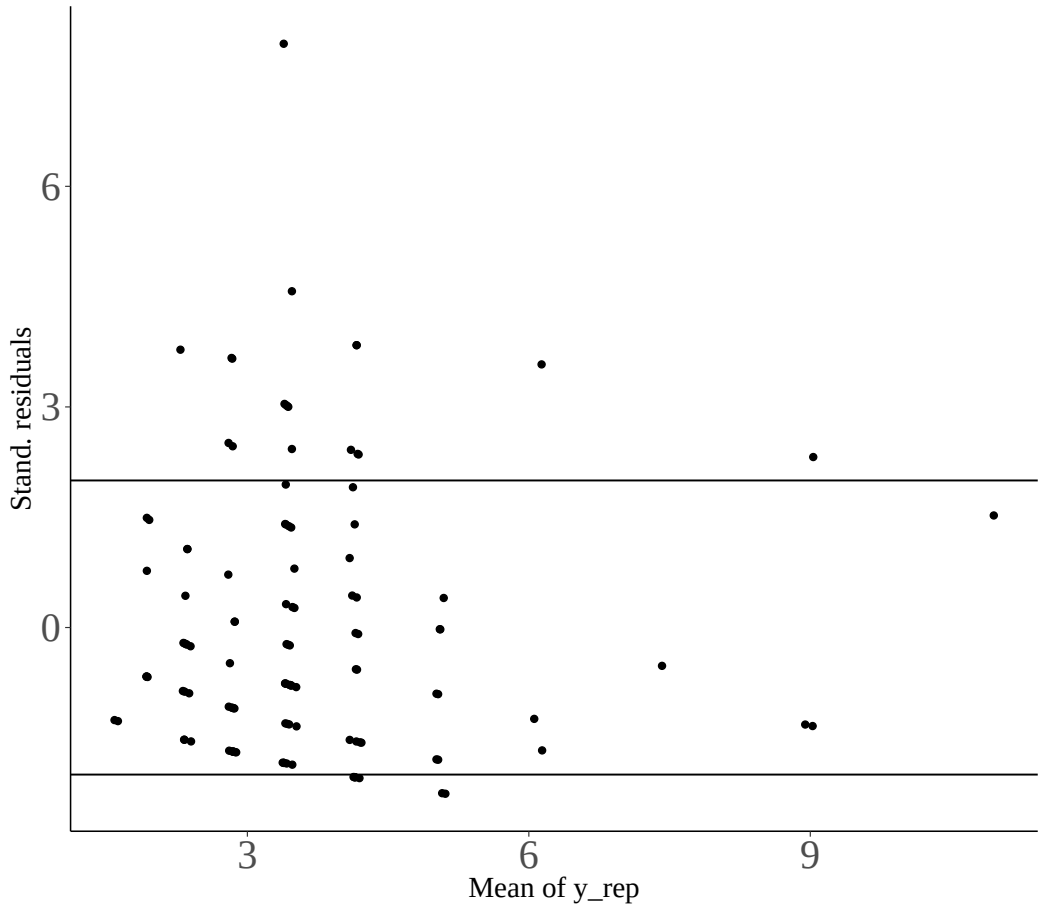


Pest control: pp check.

Comments:

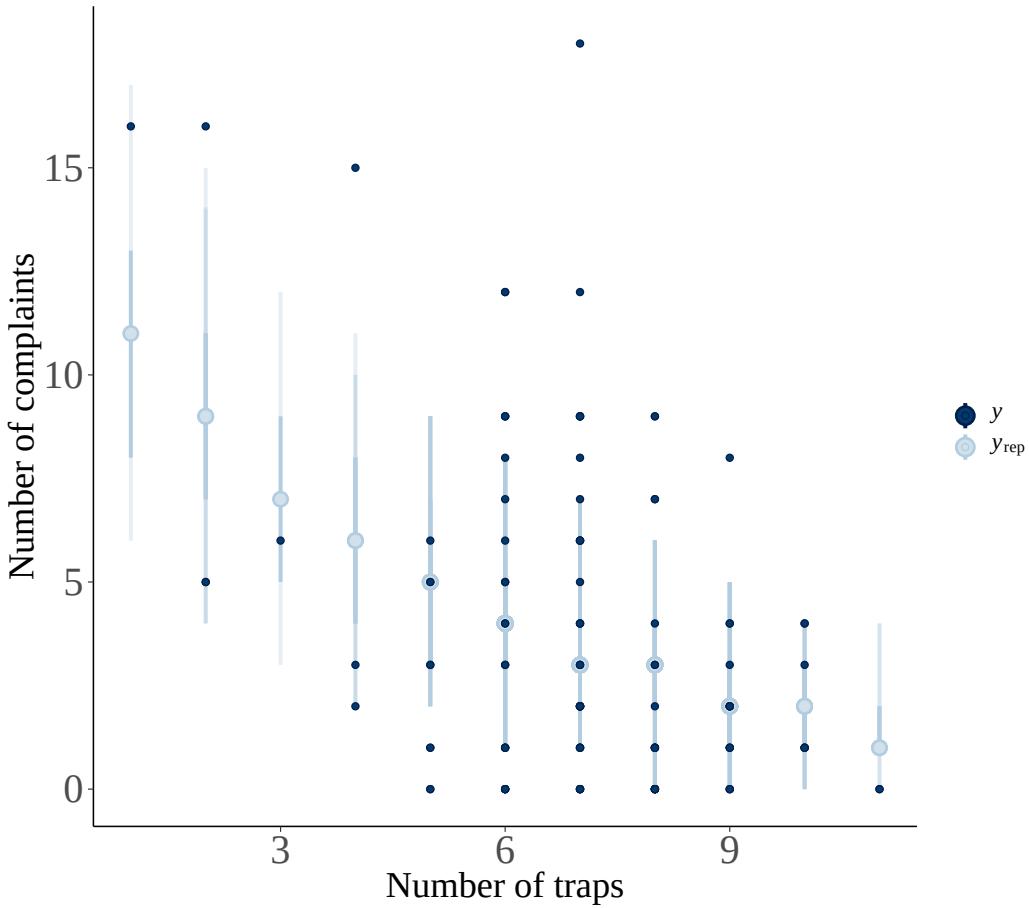
- We immediately realize that replicated distributions are far from the observed data distribution, and that the proportion of zero assumed by the Poisson model is quite underestimated...It is clear that the model does not capture this feature of the data well at all.
- Maybe the Poisson distribution distribution is not suited in this case...let's still explore the standardised residuals of the observed vs predicted number of complaints.
- We can also view how the predicted number of complaints varies with the number of traps.

Pest control: pp check. Residuals



It looks as though we have more positive residuals than negative $\Rightarrow$ the model tends to underestimate the number of complaints.

Pest control: pp check. Predictive intervals



We can see that the model does not seem to fully capture the data.

Strategies when a pp check fails

What to do if a pp check fails? There is not a unique answer. However, some tips may be the following ones:

- extend the model: augment the predictors, include eventual hierarchies
- change the sampling distribution
- change the priors
- transform your data, for instance using logarithm scale.

In what follows, we do not include further predictors, but we will work on the choice of the sampling distribution and, finally, we will consider further hierarchies.

Pest example. Negative binomial model

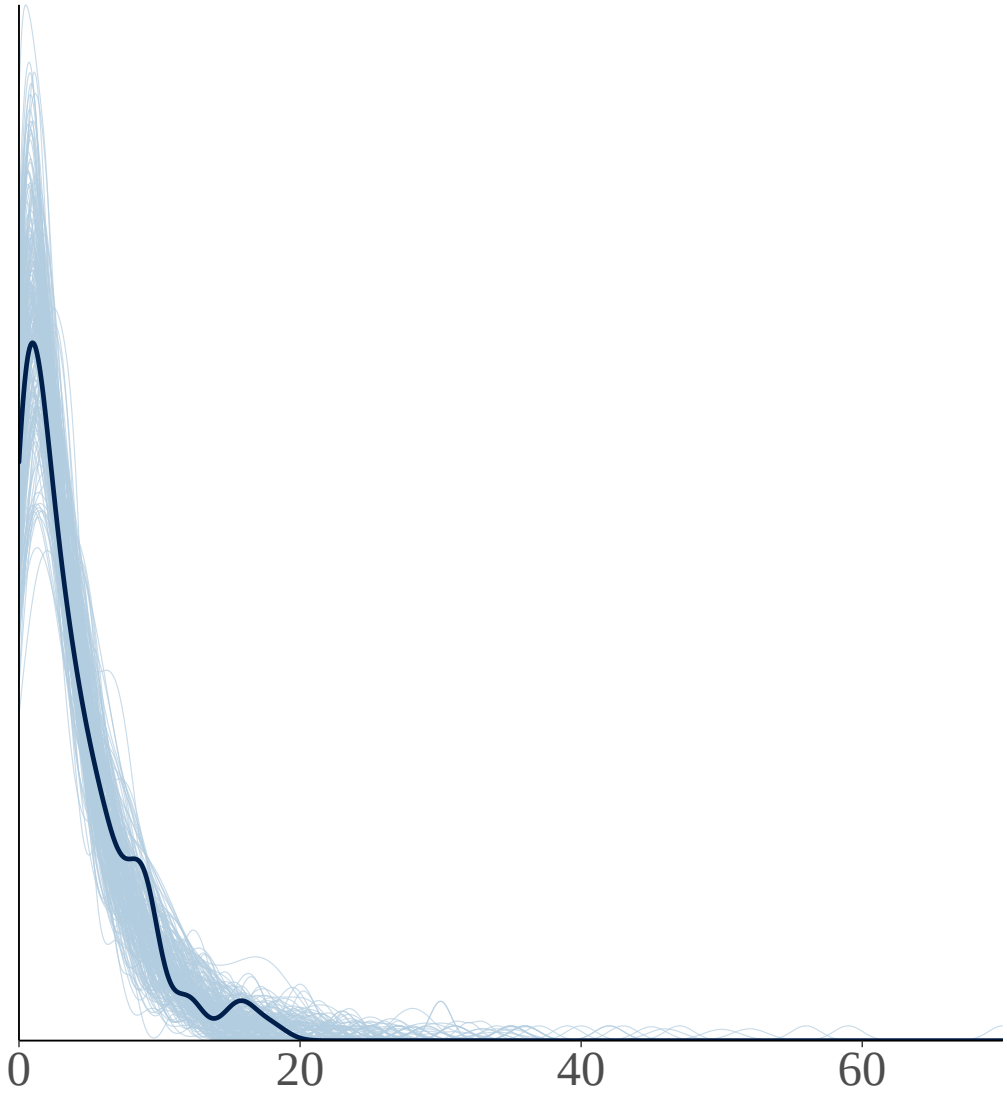
As already seen, negative binomial distribution may capture the *overdispersion* in the data with the parameter ϕ:

$$\begin{aligned} \text{complaints}_i &\sim \text{Neg-Binomial}(\lambda_i, \phi) \\ \lambda_i &= \exp(\eta_i) \\ \eta_i &= \alpha + \beta \, \text{traps}_i \end{aligned}$$

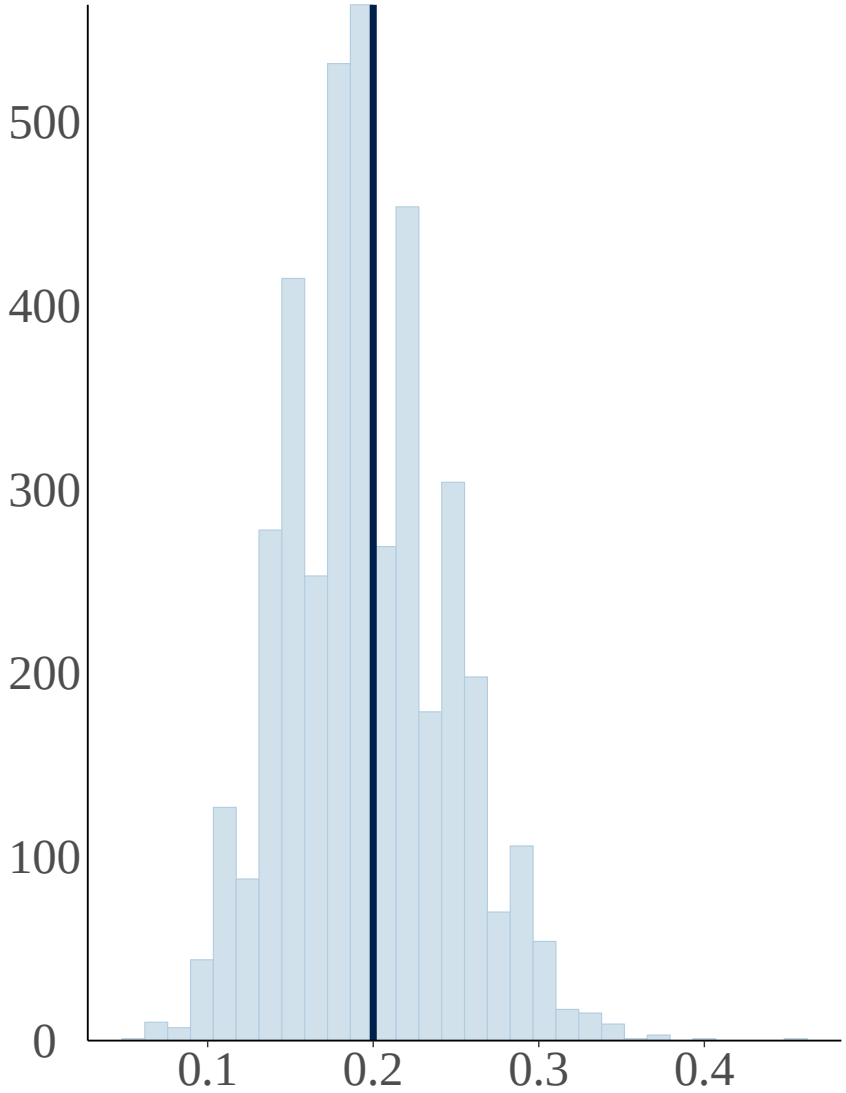
We fit also the negative-binomial model in Stan:

	mean	sd	2.5%	25%	50%	75%	97.5%	n_eff	Rhat
alpha	2.49	0.34	1.81	2.26	2.49	2.73	3.16	1177	1
beta	-0.18	0.05	-0.27	-0.21	-0.18	-0.15	-0.09	1167	1

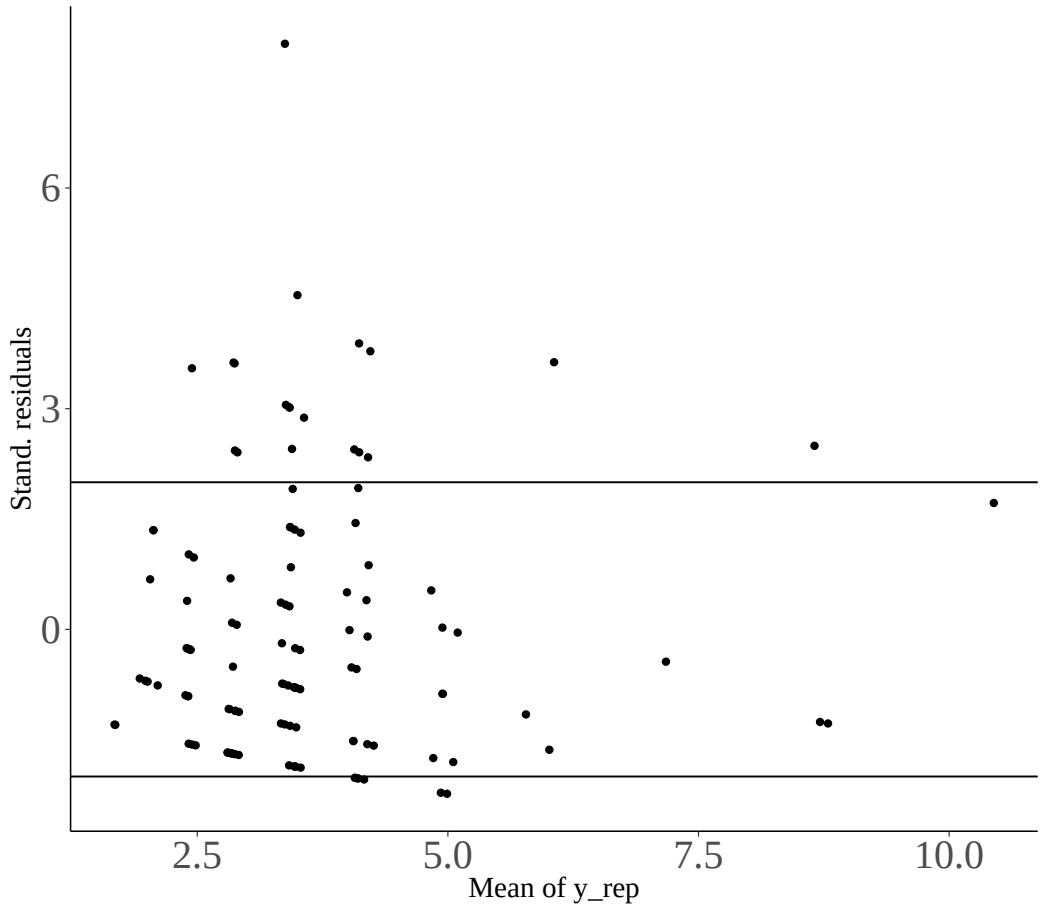
Pest control, NB model. PP check: densities



Pest control, NB model: pp check. Proportion of zeros



Pest control, NB model: pp check. Residuals



It looks as though we have more positive residuals than negative $\Rightarrow$ the model tends to underestimate the number of complaints.

Comments for pp check, NB model

- It appears that our model now captures both the number of small counts better as well as the tails. The negative binomial model does a better job in capturing the number of zeros.
- However, we still have some very large standardized residuals. This might be because we are currently ignoring that the data are clustered by buildings, and that the probability of roach issue may vary substantially across buildings.

Pest control: Hierarchical modelling

Let's add a hierarchical intercept parameter, α_b at the building level to our model. Thus, for the i-th complaint in the b-th building we have:

$$
\begin{aligned}
\text{complaints}_{ib} &\sim \text{Neg-Binomial}(\lambda_{ib}, \phi) \\
\lambda_{ib} &= \exp\left(\eta_{ib}\right) \\
\eta_{ib} &= \alpha_{b(i)} + \beta\, \text{traps}_i + \beta_{\text{super}}\, \text{super}_i + \text{log\_sq\_foot}_i \\
\alpha_b &\sim \mathcal{N}(\mu, \sigma_\alpha^2)
\end{aligned}
$$

One of our predictors varies only by building, so we can rewrite the above model more efficiently like so:

$$
\begin{aligned}
\eta_{ib} &= \alpha_{b(i)} + \beta\, \text{traps}_i + \text{log\_sq\_foot}_i \\
\alpha_b &\sim \mathcal{N}(\mu + \beta_{\text{super}}\, \text{super}_i, \sigma_\alpha^2)
\end{aligned}
$$

Pest control: Hierarchical modelling

We have more information at the building level as well, like the average age of the residents, the average age of the buildings, and the average per-apartment monthly rent so we can add that data into a matrix called `building_data`, which will have one row per building and four columns:

- `live_in_super`: An indicator for whether the building has a live-in super
- `age_of_building`: The age of the building
- `average_tenant_age`: The average age of the tenants per building
- `monthly_average_rent`: The average monthly rent per building

We'll write the Stan model like:

$$\begin{aligned} \eta_{ib} &= \alpha_{b(i)} + \beta \, \text{traps}_i + \text{log\_sq\_foot}_i \\ \alpha_b &\sim \mathcal{N}(\mu + \texttt{building\_data}\,\zeta, \sigma_\alpha^2) \end{aligned} \tag{6}$$

Model fit in Stan

We fit the model in Stan, at the end we obtain these warnings:

```
Warning messages:
1: There were 915 divergent transitions after warmup.
Increasing adapt_delta above 0.8 may help.
```

What happened? We get a bunch of warnings from Stan about divergent transitions, which is an indication that there may be regions of the posterior that have not been explored by the Markov chains. We will return to this issue later...

In this example we will see that we have divergent transitions because we need to reparametrize our model - i.e., we will retain the overall structure of the model, but transform some of the parameters so that it is easier for Stan to sample from the parameter space.

Model fit in Stan

	mean	sd	2.5%	25%	50%	75%	97.5%	n_eff	Rhat
sigma_alpha	0.25	0.17	0.05	0.13	0.22	0.34	0.69	182	1.03
beta	-0.23	0.06	-0.35	-0.27	-0.22	-0.19	-0.11	715	1.00
mu	1.25	0.42	0.43	0.98	1.22	1.53	2.12	849	1.00
phi	1.54	0.36	0.99	1.29	1.49	1.75	2.38	302	1.01
alpha[1]	1.28	0.54	0.21	0.95	1.24	1.62	2.37	1007	1.00
alpha[2]	1.23	0.52	0.21	0.91	1.20	1.56	2.31	914	1.00
alpha[3]	1.39	0.49	0.51	1.05	1.38	1.71	2.41	397	1.01
alpha[4]	1.43	0.48	0.53	1.09	1.39	1.75	2.42	561	1.00
alpha[5]	1.07	0.42	0.25	0.76	1.08	1.33	1.94	880	1.01
alpha[6]	1.16	0.48	0.22	0.86	1.16	1.45	2.16	914	1.00
alpha[7]	1.43	0.52	0.49	1.07	1.39	1.77	2.51	434	1.01
alpha[8]	1.27	0.42	0.45	1.00	1.29	1.52	2.12	1156	1.00
alpha[9]	1.40	0.55	0.29	1.05	1.41	1.74	2.51	1077	1.00
alpha[10]	0.86	0.37	0.17	0.60	0.85	1.11	1.62	644	1.01

Model fit in Stan

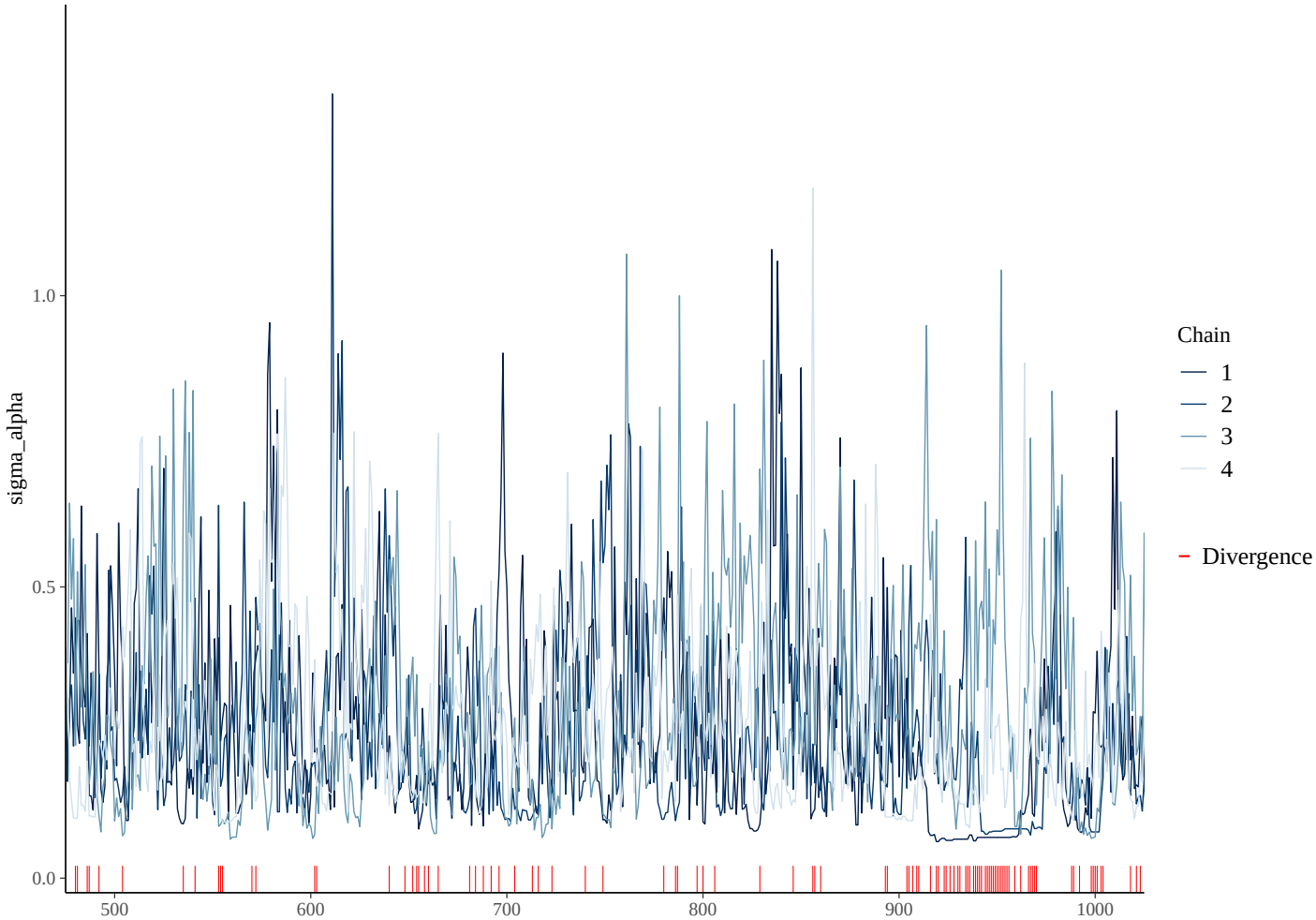
Before we go through exactly how to do this reparameterization, we will first go through what indicates that this is something that reparameterization will resolve. We will go through:

1. Examining the fitted parameter values, including the effective sample size
2. Traceplots and scatterplots that reveal particular patterns in locations of the divergences.

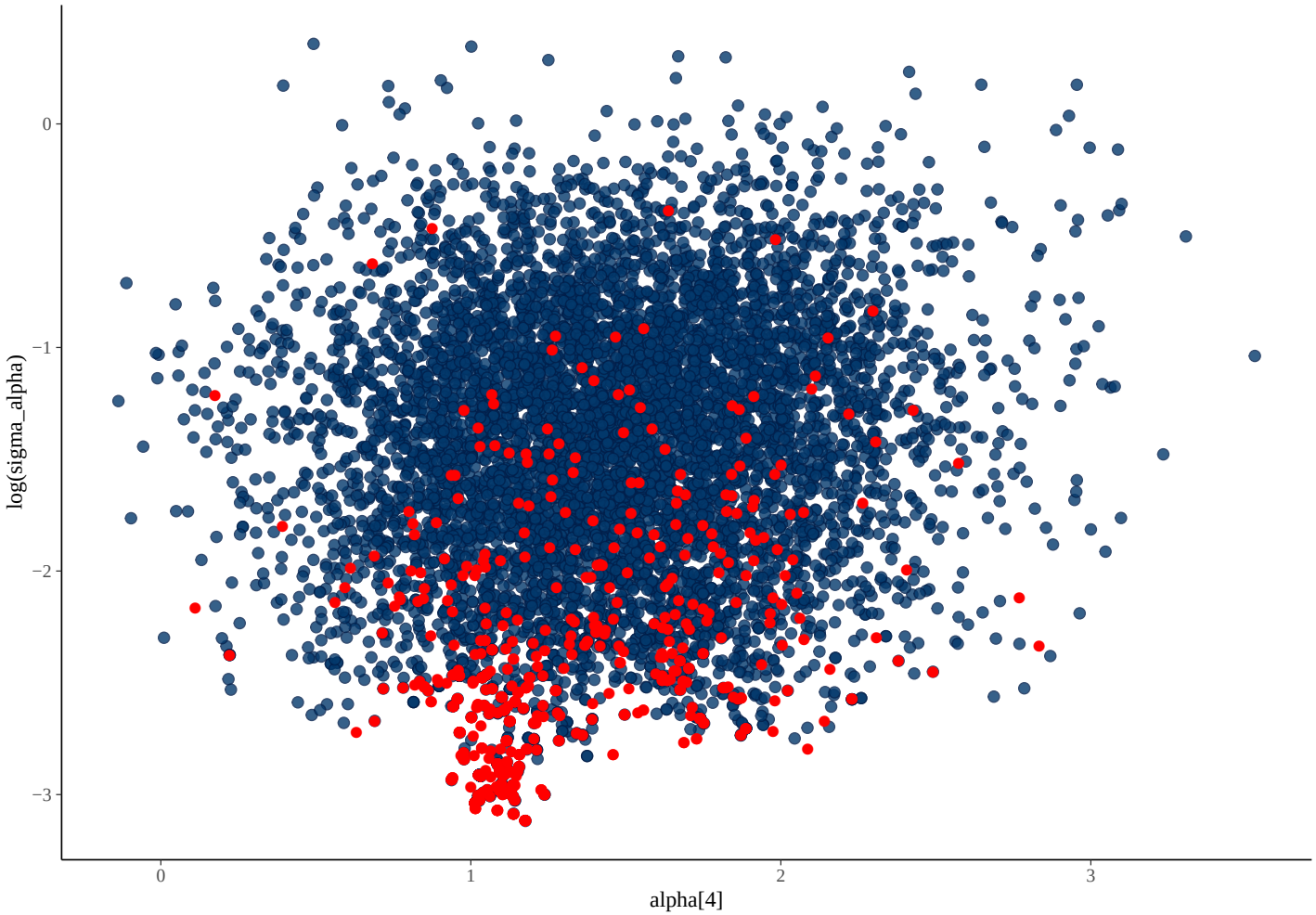
The effective samples are quite low for many of the parameters relative to the total number of samples. This alone isn't indicative of the need to reparameterize, but it indicates that we should look further at the trace plots and pairs plots.

Model fit in Stan

First let's look at the traceplots to see if the divergent transitions form a pattern.

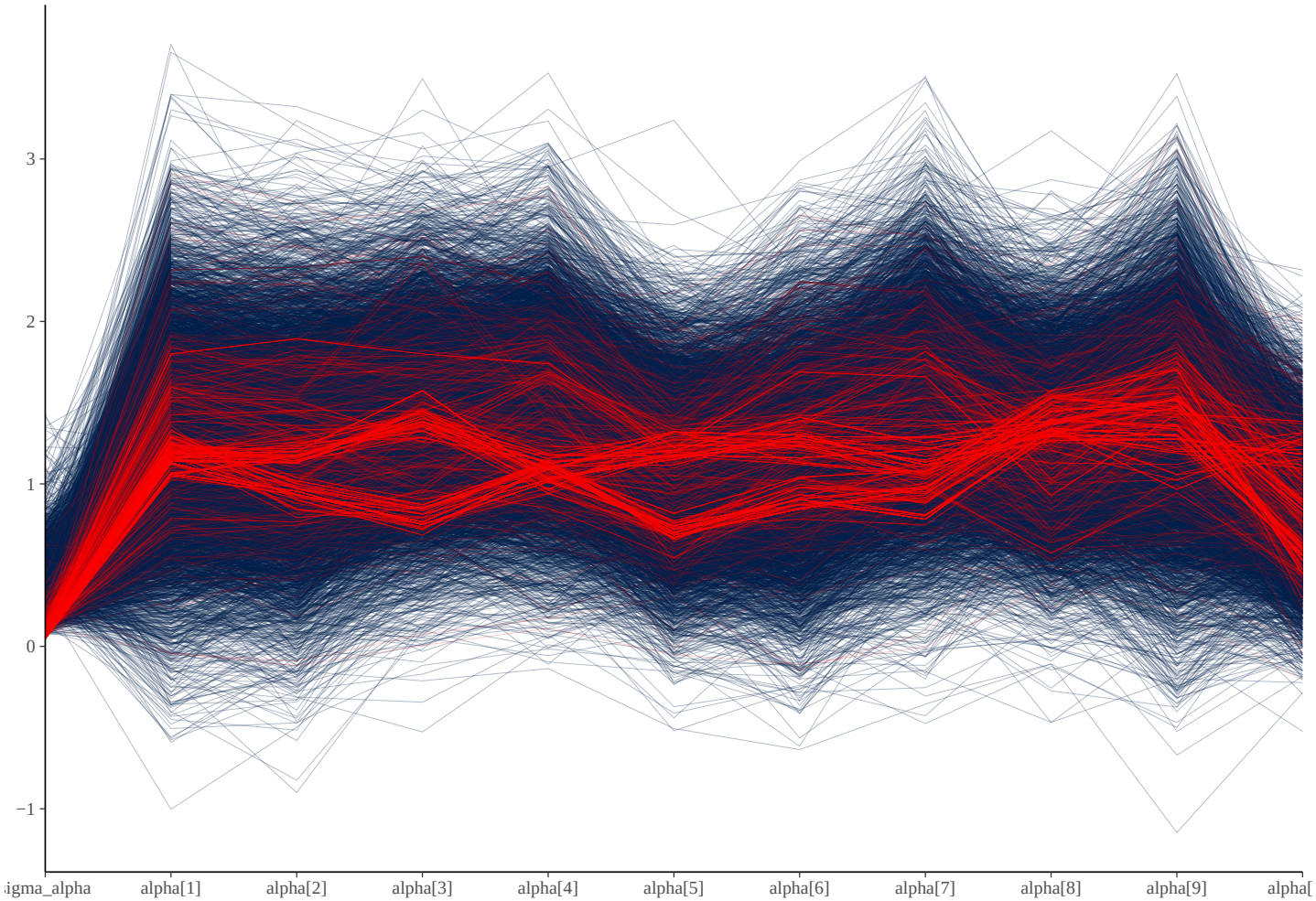


Model fit in Stan



Model fit in Stan

Another way to look at the divergences is via a parallel coordinates plot:



Model fit in Stan

Comments:

- Looks as if the divergent parameters, the little red bars underneath the traceplots correspond to samples where the sampler gets stuck at one parameter value for σ_α.
- What we have in the scatterplot, is a cloud-like shape, with most of the divergences clustering towards the bottom. We'll see a bit later that we actually want this to look more like a funnel than a cloud, but the divergences are indicating that the sampler can't explore the narrowing neck of the funnel.
- From the parallel plot, again, we see evidence that our problems concentrate when σ_α is small.

Model fit in Stan: non-centered parametrization

CENTERED

$$\eta_{ib} = \alpha_{b(i)} + \beta\, x_i$$
$$\alpha_b \sim \mathcal{N}(\mu + \zeta\, z_b, \sigma_\alpha^2)$$

NON-CENTERED

$$\eta_{ib} = \alpha_{b(i)} + \beta\, x_i$$
$$\alpha_b = \mu + \zeta\, z_b + \sigma_\alpha \tilde{\alpha}_b$$
$$\tilde{\alpha}_b \sim \mathcal{N}(0, 1)$$

We should use the non-centered parameterization for α_b. We define a vector of auxiliary variables in the parameters block, `alpha_raw` that is given a $\mathcal{N}(0, 1)$ prior in the model block. We then make `alpha` a transformed parameter. We can reparameterize the random intercept α_b, which is distributed:

$$\alpha_b \sim \mathcal{N}(\mu + \texttt{building\_data}\,\zeta, \sigma_\alpha^2)$$

Model fit in Stan: non-centered parametrization

In the `transformed parameters` block we define now:

```
transformed parameters {
  vector[J] alpha;
  alpha = mu + building_data * zeta + sigma_alpha * alpha_raw;
}
```

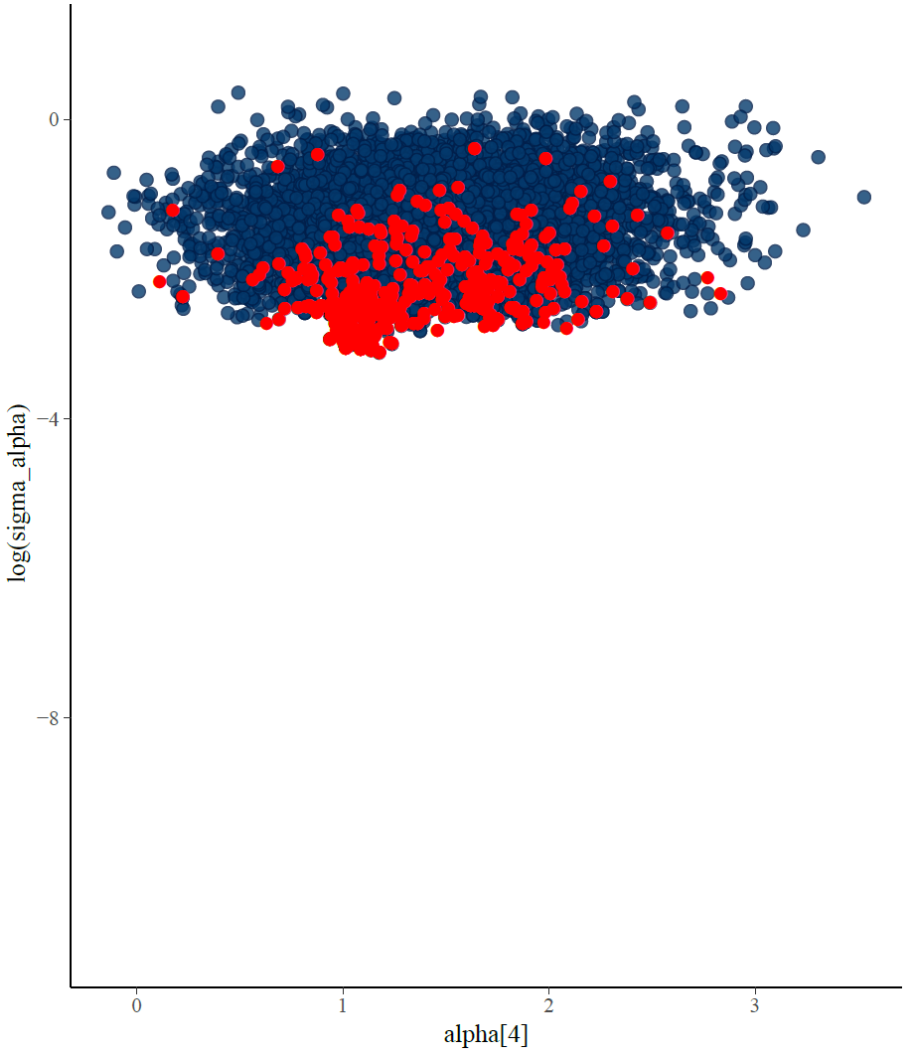
This gives `alpha` a $\mathcal{N}(\mu + \texttt{building\_data}\,\zeta, \sigma_\alpha^2)$ distribution, but it decouples the dependence of the density of each element of `alpha` from `sigma_alpha` (σ_α).

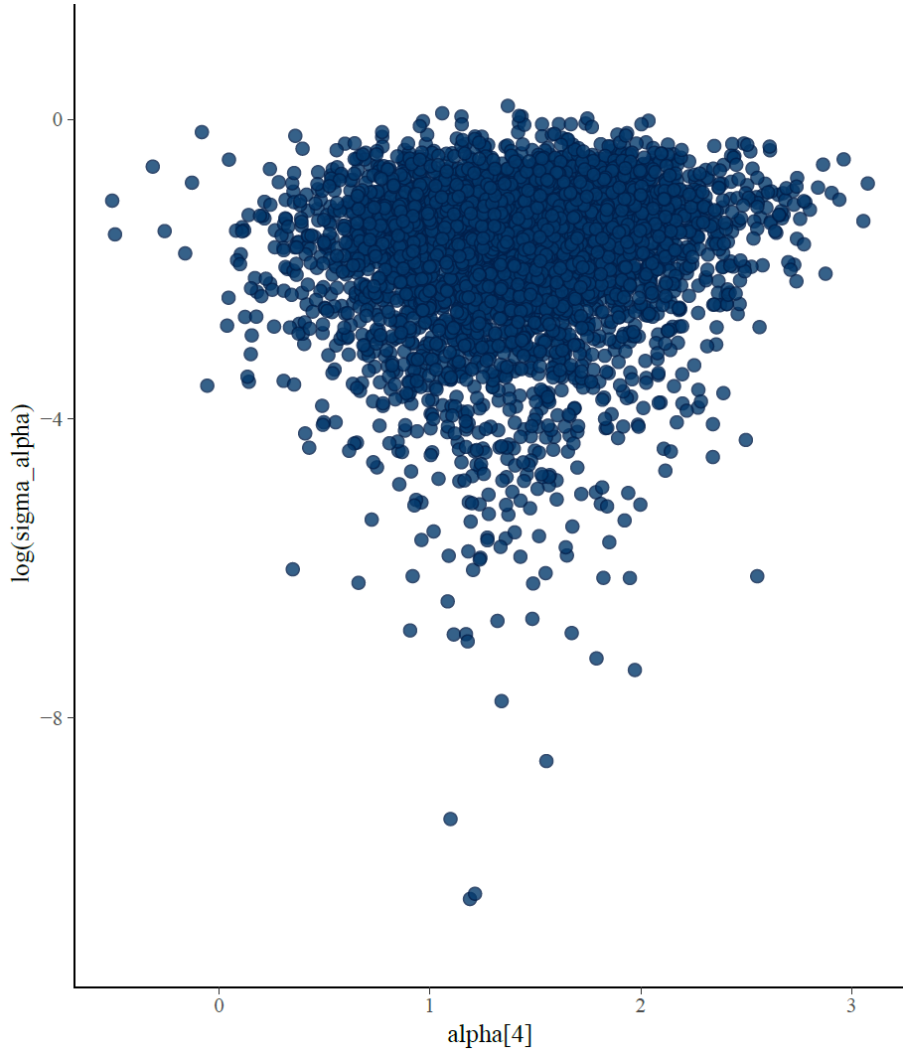
We fit this new model version in Stan. We will examine the effective sample size of the fitted model to see whether we've fixed the problem with our reparameterization.

Model fit in Stan: non-centered parametrization

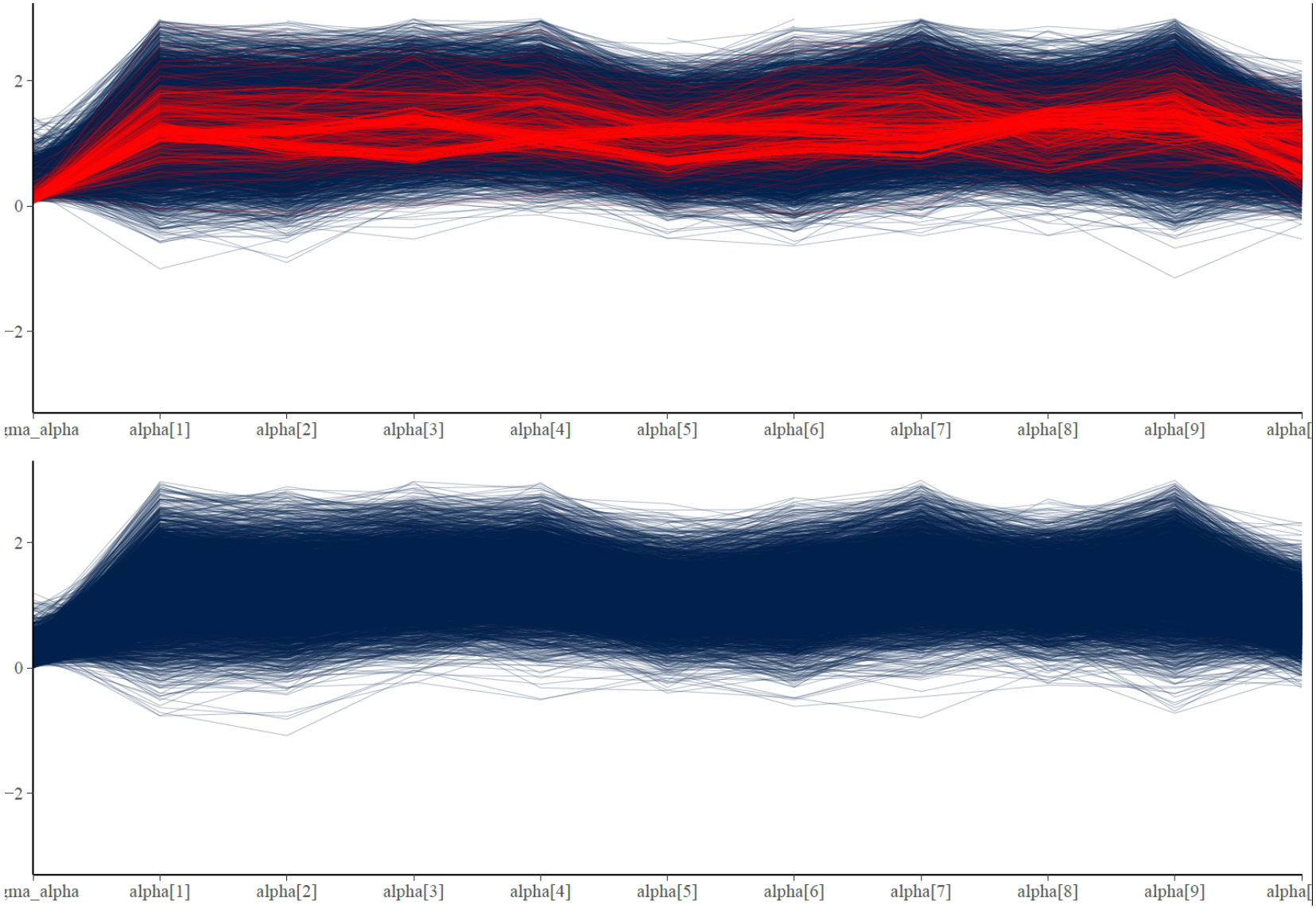
	mean	sd	2.5%	25%	50%	75%	97.5%	n_eff	Rhat
sigma_alpha	0.23	0.17	0.01	0.10	0.20	0.32	0.63	1447	1
beta	-0.23	0.06	-0.35	-0.27	-0.23	-0.19	-0.11	2649	1
mu	1.25	0.44	0.40	0.95	1.24	1.54	2.12	2555	1
phi	1.58	0.34	1.03	1.34	1.54	1.77	2.35	4256	1
alpha[1]	1.27	0.56	0.15	0.90	1.27	1.64	2.37	2566	1
alpha[2]	1.21	0.53	0.19	0.86	1.21	1.56	2.28	2551	1
alpha[3]	1.38	0.49	0.42	1.05	1.38	1.71	2.38	2672	1
alpha[4]	1.42	0.49	0.46	1.08	1.42	1.74	2.39	2783	1
alpha[5]	1.08	0.42	0.26	0.81	1.07	1.34	1.92	3162	1
alpha[6]	1.17	0.49	0.22	0.85	1.17	1.49	2.12	2502	1
alpha[7]	1.45	0.52	0.42	1.10	1.44	1.79	2.49	2996	1
alpha[8]	1.23	0.43	0.40	0.94	1.23	1.52	2.10	3481	1
alpha[9]	1.41	0.58	0.25	1.03	1.42	1.80	2.51	2780	1
alpha[10]	0.86	0.37	0.17	0.61	0.85	1.11	1.60	3417	1

Model fit in Stan: centered vs non-centered parametrization





Model fit in Stan: centered vs non-centered parametrization

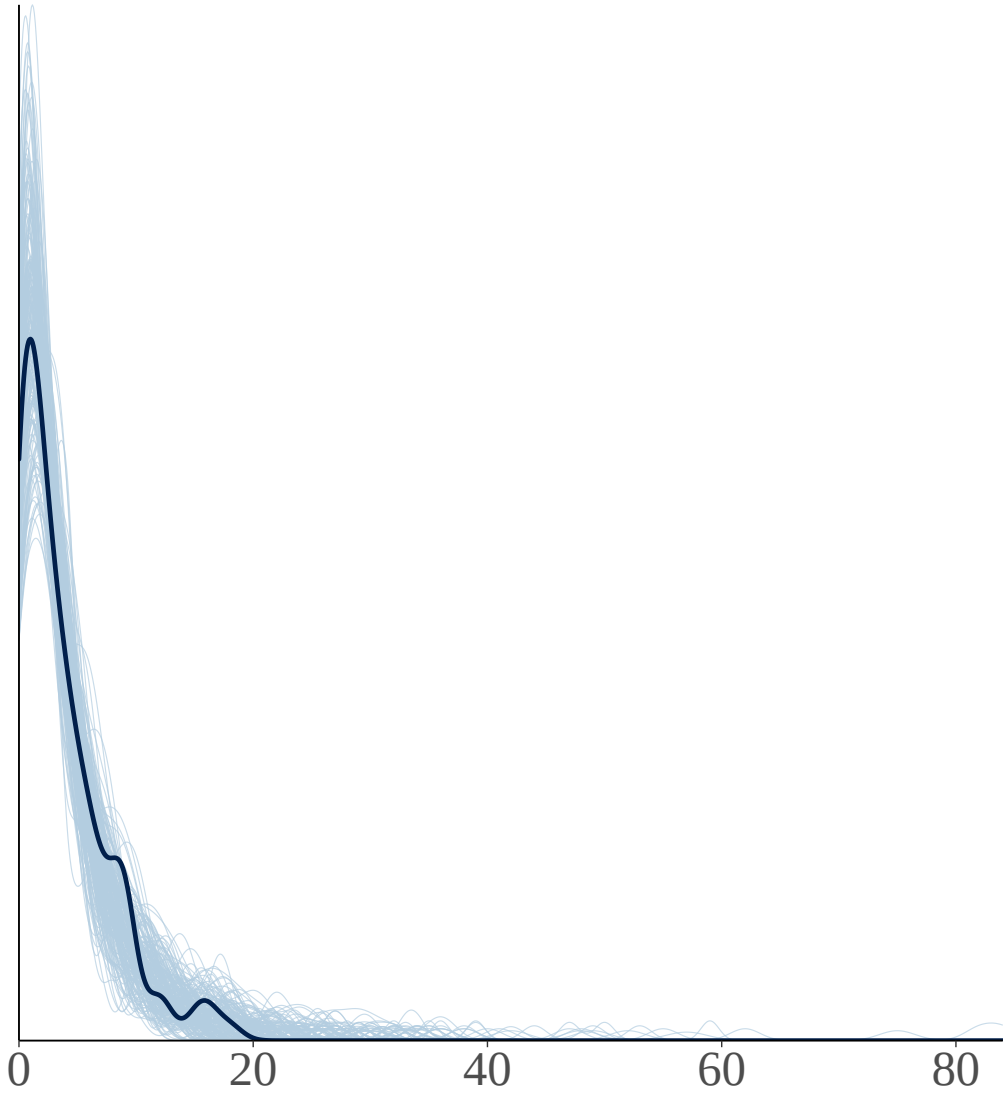


Model fit in Stan: centered vs non-centered parametrization

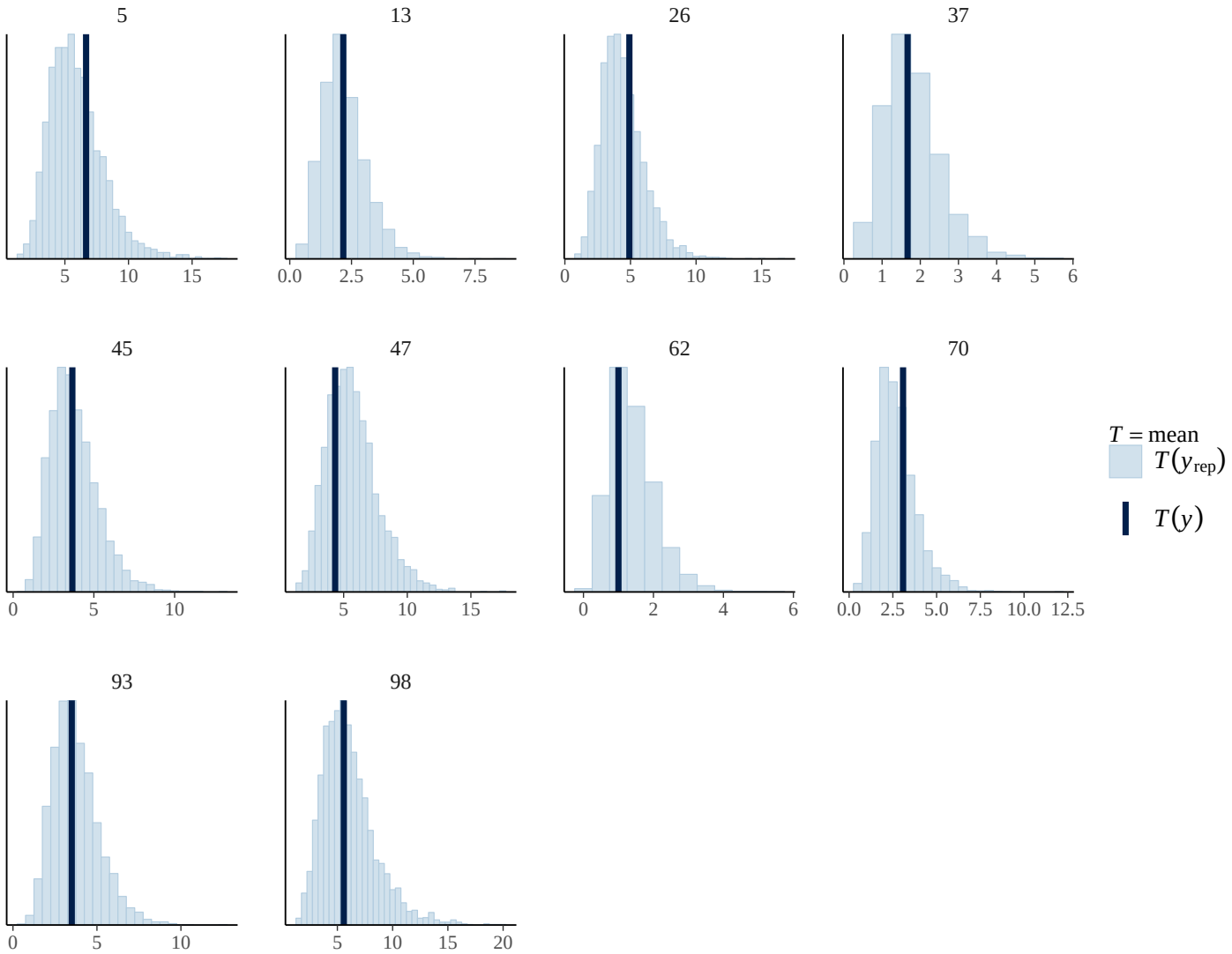
Comments:

- This has improved the effective sample sizes of α.
- No more divergent transitions!

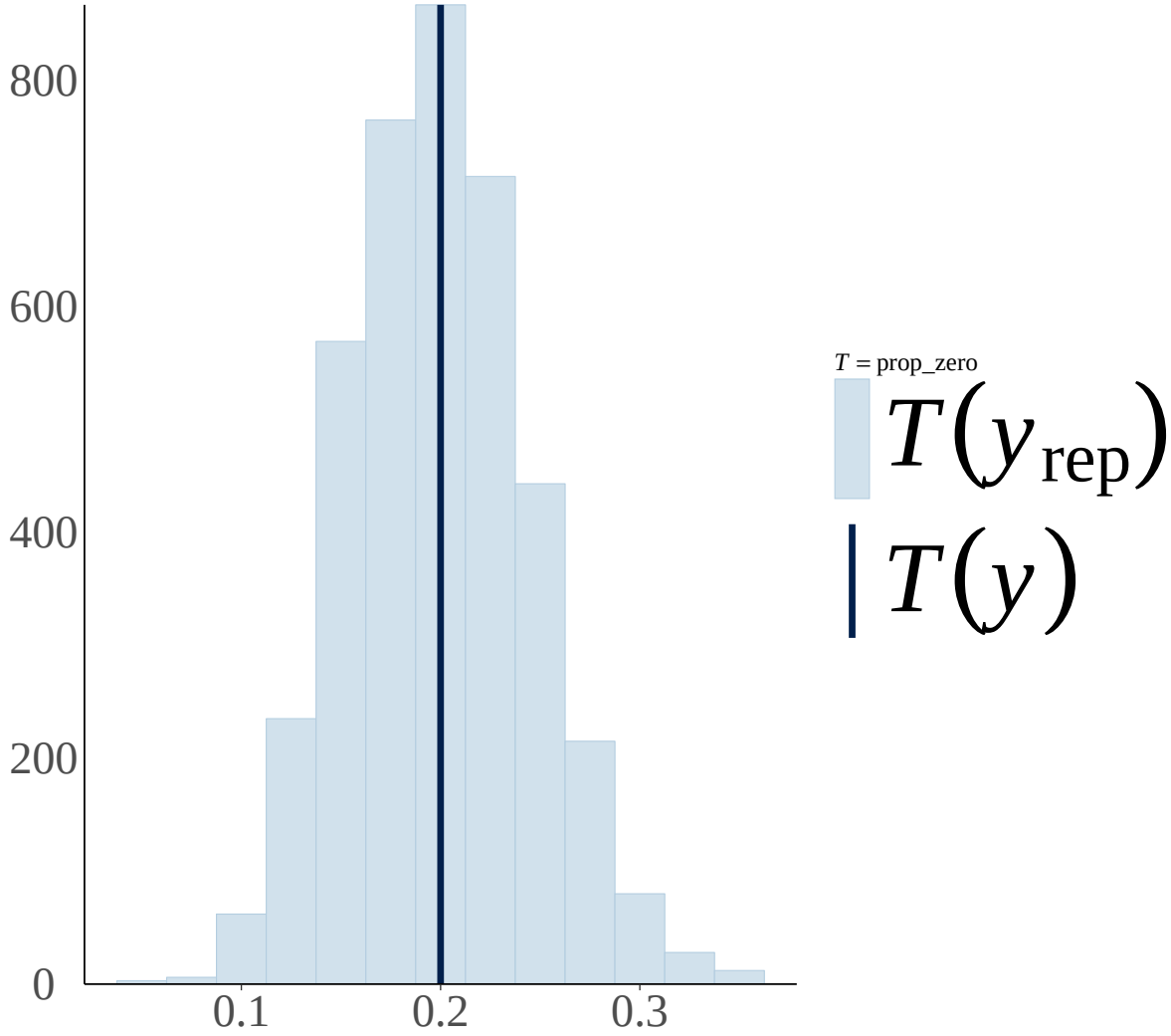
Pest model, hierarchical NB ncp model. PP check: densities



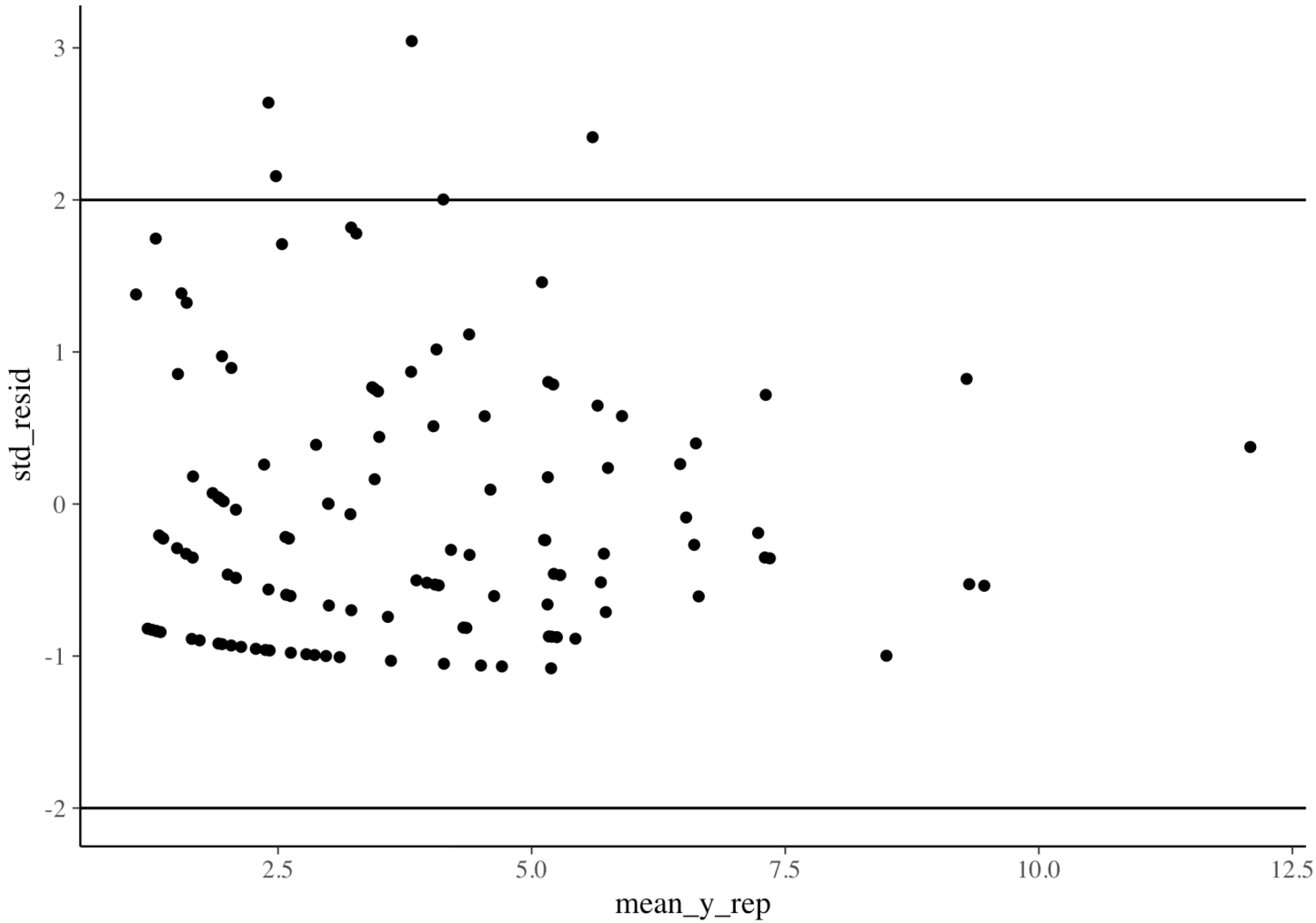
Pest model, hierarchical NB ncp model. PP check: statistics



Pest model, hier NB ncp model. PP check: prop. of zeros



Pest control, hier NB ncp model: pp check. Residuals



Better!

Further readings

Further reading:

- Chapter 6 from *BDA*, A. Gelman et al. (model checking)
- Chapter 20 from the Stan Users Guide (reparametrization)

Indice

Motivations

Bayesian models can be evaluated and compared in several ways. Most simply, any model or set of models can be taken as an exhaustive set, in which case all inference is summarized by the posterior distribution.

The fit of model to data can be assessed using posterior predictive checks, prior predictive checks or, more generally, mixed checks for hierarchical models.

However, we may need a pure comparison set of tools: when several candidate models are available, they can be compared and averaged using:

- Bayes factors (which is equivalent to embedding them in a larger discrete model)
- Predictive information criteria (AIC, DIC, BIC, WAIC,...)

Indice

Evaluate predictive accuracy

One way to evaluate a model is through the accuracy of its predictions. Sometimes we care about this accuracy for its own sake, as when evaluating a forecast. In other settings, predictive accuracy is valued not for its own sake but rather for comparing different models. We are interested in prediction accuracy for two reasons:

- to measure the performance of a model that we are using;
- second, to compare models.

If we consider data $y_1, \ldots, y_n$ modelled as independent given parameters θ, thus $p(y|\theta) = \prod_{i=1}^{n} p(y_i|\theta)$. A general summary of predictive fit is the log predictive density, $\log(p(y|\theta))$. When comparing models of differing size, it is important to make some adjustment for the natural ability of a larger model to fit data better, even if only by chance.

Evaluate predictive accuracy

The general form for the predictive information criteria that we will encounter is the following:

$$\text{crit} = -2\text{lpd} + \text{penalty}$$

- lpd is a measure of the log predictive density of the fitted model.
- penalty is a penalization accounting for the effective number of parameters of the fitted model.

The interpretation is the following: the lower is a particular value for an information criteria, and the better is the model fit. Moreover, if two competing models share the same value for the log predictive density, the model with less parameters is favored.

This is the Occam's Razor occurring in statistics:

Frustra fit per plura quod potest fieri per pauciora

Evaluate predictive accuracy

- Difficulty All the proposed measures are attempting to perform what is, in general, an impossible task: to obtain an unbiased and accurate measure of out-of-sample prediction error that will be valid over a general class of models and that requires minimal computation.

Watanabe-Akaike Information Criteria (WAIC)

We define the log pointwise predictive density for a single value y_i:

$$\text{lppd} = \sum_{i=1}^{n} \log(p(y_i|y)) = \sum_{i=1}^{n} \log \int p(y_i|\theta)\pi(\theta|y)d\theta. \tag{7}$$

To compute the lppd in practice, we can evaluate the expectation using draws from $\pi(\theta|y)$, the usual posterior simulations, which we label $\theta^{(s)}$, $s = 1, \ldots, S$, defining the computed log pointwise predictive density:

$$\widehat{\text{lppd}} = \sum_{i=1}^{n} \log \left(\frac{1}{S} \sum_{s=1}^{S} p(y_i|\theta^{(s)}) \right) \tag{8}$$

WAIC

Then, we define the WAIC as follows:

$$\text{WAIC} = -2\text{lppd} + 2p_{\text{WAIC}}, \tag{9}$$

where the quantity p_{WAIC} is defined as:

$$p_{\text{WAIC}} = \sum_{i=1}^{n} \text{Var}_{\theta|y}(\log(p(y_i|\theta)),$$

which computes the variance separately for each data point. We can practically compute this quantity by using:

$$\sum_{i=1}^{n} \text{Var}_{s=1}^{S}(\log(p(y_i|\theta^{(s)})),$$

where $\text{Var}_{s=1}^{S}$ represents the sample variance,
$\text{Var}_{s=1}^{S} a_s = \frac{1}{S-1}\sum_{s=1}^{S}(a_s - \bar{a})^2$.

WAIC

- Compared to AIC and DIC, WAIC has the desirable property of averaging over the posterior distribution rather than conditioning on a point estimate.
- This is especially relevant in a predictive context, as WAIC is evaluating the predictions that are actually being used for new data in a Bayesian context. AIC and DIC estimate the performance of the plugin predictive density, but Bayesian users of these measures would still use the posterior predictive density for predictions.
- WAIC works also with singular models and thus is particularly helpful for models with hierarchical and mixture structures in which the number of parameters increases with sample size and where point estimates often do not make sense.

Leave-one-out cross-validation

In Bayesian cross-validation, the data are repeatedly partitioned into a training set y_{train} and a holdout set y_{holdout}, and then the model is fit to y_{train} (thus yielding a posterior distribution $\pi(\theta|y_{\text{train}})$), with this fit evaluated using an estimate of the log predictive density of the holdout data, $\log(p_{\text{train}}(y_{\text{holdout}})) = \log \int p_{\text{pred}}(y_{\text{holdout}}|\theta)\pi_{\text{train}}(\theta)d\theta$. The Bayesian leave-one-out cross-validation (LOO-CV) estimate of out-of-sample predictive fit is:

$$\text{lppd}_{\text{loo-cv}} = \sum_{i=1}^{n} \log(p(y_i|y_{-i})) = \sum_{i=1}^{n} \log \int p(y_i|\theta)\pi(\theta|y_{-i})d\theta, \qquad (10)$$

where y_{-i} represents the data without the i-th data point. This quantity is usually calculated as:

$$\sum_{i=1}^{n} \log \left(\frac{1}{S} \sum_{i=1}^{n} p(y_i|\theta^{(s)}) \right)$$

Leave-one-out cross-validation

- Each prediction is conditioned on $n-1$ data points, which causes underestimation of the predictive fit. For large n the difference is negligible, but for small n (or when using K-fold cross-validation) we can use a first order bias correction.
- Cross-validation is like WAIC in that it requires data to be divided into disjoint, ideally conditionally independent, pieces. This represents a limitation of the approach when applied to structured models.
- In addition, cross-validation can be computationally expensive except in settings where shortcuts are available to approximate the distributions $p(y_i|y_{-i})$.
- The purpose of using LOO or WAIC is to estimate the accuracy of the predictive distribution $p(\tilde{y}_i|y)$.

Importance sampling LOO (IS-LOO)

If the n points are conditionally independent in the data model we can then evaluate $p(y_i|y_{-i})$ with draws $\theta^{(s)}$ from the full posterior $\pi(\theta|y)$ using importance ratios:

$$r_i^{(s)} = \frac{1}{p(y_i|\theta^{(s)})} \propto \frac{\pi(\theta^{(s)}|y_{-i})}{\pi(\theta^{(s)}|y)}$$

to get the importance sampling leave-one-out (IS-LOO) predictive distribution,

$$p(\tilde{y}_i|y_{-i}) \approx \frac{\sum_{s=1}^{S} r_i^{(s)} p(\tilde{y}_i|\theta^{(s)})}{\sum_{s=1}^{S} r_i^{(s)}} \tag{11}$$

Evaluating this LOO log predictive density at the held-out data point y_i, we get

$$p(y_i|y_{-i}) \approx \frac{1}{\frac{1}{S}\sum_{s=1}^{S} \frac{1}{p(y_i|\theta^{(s)})}}$$

PSIS-LOO

A direct use of r_i induces instability because the importance ratios can have high or infinite variance. We can improve the LOO estimate using Pareto smoothed importance sampling (PSIS), which applies a smoothing procedure to the importance weights. Here the main steps:

1. Since the distribution of the importance weights used in LOO may have a long right tail, we fit a generalized Pareto distribution to the tail (20% largest importance ratios $r^{(s)}$). The computation is done separately for each held-out data point i.
2. Stabilize the importance ratios by replacing the largest ratios by the expected values of the order statistics of the fitted generalized Pareto distribution. Label these values as $\tilde{\omega}_i^{(s)}$.
3. To guarantee finite variance of the estimate, truncate each vector of weights at $S^{3/4}\bar{w}_i$, where $\bar{w}_i$ is the average of the S smoothed weights corresponding to the distribution holding out data point i . Finally, label these truncated weights as $\omega_i^{(s)}$.

PSIS-LOO

The PSIS estimate of the LOO expected log pointwise predictive density (PSIS-LOO) is the same as in (11), but with the new weights ω_i in place of r_i. The LOOIC criteria is then defined as:

$$\text{LOOIC} = -2 \sum_{i=1}^{n} \log \left(\frac{\sum_{s=1}^{S} \omega_i^{(s)} p(\tilde{y}_i | \theta^{(s)})}{\sum_{s=1}^{S} \omega_i^{(s)}} \right) \tag{12}$$

The estimated shape parameter $\hat{k}$ of the generalized Pareto distribution can be used to assess the reliability of the estimate:

- $k < 1/2$: the variance of the raw importance ratios is finite, the central limit theorem holds, and the estimate converges quickly.
- $k > 1/2$: the variance of the PSIS estimate is finite but may be large.

Indice

Implementation in Stan

We illustrate how to write Stan code that computes and stores the pointwise log-likelihood using the eight schools example. The model is unchanged, we only need to store the pointwise log-likelihood (the `log_lik` object) in the `generated quantities` block:

```
...
generated quantities {
 vector[J] log_lik;
  for (j in 1:J){
     log_lik[j] = normal_lpdf(y[j]| theta[j], sigma[j]);
  }
}
```

The loo package

The loo R package provides the functions loo() and waic() for efficiently computing PSIS-LOO and WAIC for fitted Bayesian models using the methods described before.

These functions take as their argument an $S \times n$ loglikelihood matrix, where S is the size of the posterior sample (the number of retained draws) and n is the number of data points.

The loo() function returns PSIS-LOOIC and p_{LOO}. The waic() function computes the analogous functions for WAIC.

Using the loo package

```
y <- c(28,8,-3,7,-1,1,18,12)
sigma <- c(15,10,16,11,9,11,10,18)
J <- 8
data <- list(y = y, sigma=sigma, J = J)
fit_1 <- stan("8schools.stan",
              data = data, iter=200,
              cores = 4, chains =4)
#computing psis-looic
log_lik_1 <- extract_log_lik(fit_1)
loo_1 <- loo(log_lik_1)
print(loo_1)

          Estimate  SE
elpd_loo    -30.8 0.9
p_loo         1.3 0.3
looic        61.7 1.8
```

Eight schools example: model comparison

Model:

$$y_j \sim \mathcal{N}(\theta_j, \sigma_y^2)$$
$$\theta_j \sim \mathcal{N}(\mu, \tau^2)$$

Three possible priors (then, three models):

1. $\tau \propto 1$
2. $\tau^2 \sim \text{InvGamma}(0.001, 0.001)$
3. $\tau \sim \text{HalfCauchy}(0, 2.5)$

Let's compare the model through LOOIC and WAIC.

Eight schools example: model comparison. LOOIC

```
loo_diff <- compare(loo_1, loo_2, loo_3)
loo_diff
      elpd_diff se_diff elpd_loo p_loo looic
loo_2   0.0       0.0   -30.6      0.8  61.1
loo_3  -0.1       0.0   -30.6      0.8  61.2
loo_1  -0.4       0.3   -31.0      1.4  61.9
```

Model 2 (inverse gamma) is slightly favorite in terms of lower LOOIC. Model (1) reports the lowest LOOIC. Anyway, differences between models are quite negligible.

Eight schools example: model comparison. WAIC

```
waic_1 <- waic(log_lik_1)
waic_2 <- waic(log_lik_2)
waic_3 <- waic(log_lik_3)

waic_diff <- compare(waic_1, waic_2, waic_3)
waic_diff
```

```
       elpd_diff se_diff elpd_waic p_waic waic
waic_2   0.0       0.0   -30.6       0.8   61.1
waic_3   0.0       0.0   -30.6       0.8   61.2
waic_1  -0.3       0.3   -30.9       1.3   61.8
```

Model 2 (inverse gamma) is slightly favorite in terms of lower WAIC. The number of effective parameters, p_{WAIC}, is 0.8 for model 2 and 3, and 1.3 for model 1 (uniform prior).

Some considerations

Formulas such as AIC, DIC, and WAIC fail in various examples: AIC does not work in settings with strong prior information, DIC gives nonsensical results when the posterior distribution is not well summarized by its mean, and WAIC relies on a data partition that would cause difficulties with structured models such as for spatial or network data. Cross-validation is appealing but can be computationally expensive and also is not always well defined in dependent data settings.

But there are times when it can be useful to compare highly dissimilar models, and, for that purpose, predictive comparisons can make sense. In addition, measures of effective numbers of parameters are appealing tools for understanding statistical procedures

Further readings

Further reading:

- Gelman, A., Hwang, J., and Vehtari, A. (2014). Understanding predictive information criteria for Bayesian models. *Statistics and Computing*, 24(6), 997–1016. Here the pdf
- Vehtari, A., Gelman, A., and Gabry, J. (2017). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*, 27(5),1413–1432. Here the pdf