Generalized Linear Models with the rstanarm R Package

Ben Goodrich

April 20, 2020

Important Maximimum Entropy Distributions

- If Θ is some convex set, the maximum entropy distribution is the uniform distribution over Θ. For example, if $\Theta = [0, 1]$, it is the standard uniform distribution with PDF $f(\theta | \, a = 0, b = 1) = 1$
- If $\Theta = \mathbb{R}$, the maximum entropy distribution given an expectation and variance is the normal distribution. This extends to bivariate and multivariate distributions if you have given covariances.
- If $\Theta = \mathbb{R}_+$, then the maximum entropy distribution for a given expectation is the exponential distribution with expectation $\mu = \frac{1}{\lambda}$. You can utilize the fact that the median is $F^{-1}(0.5) = \mu \ln 2$ to go from the median to μ.
- The binomial and Poisson distributions are maximum entropy distributions given μ for their respective Ω
- Additional examples (often with weird constraints) are given at the bottom of https://en.wikipedia.org/wiki/Maximum_entropy_probability_distribution

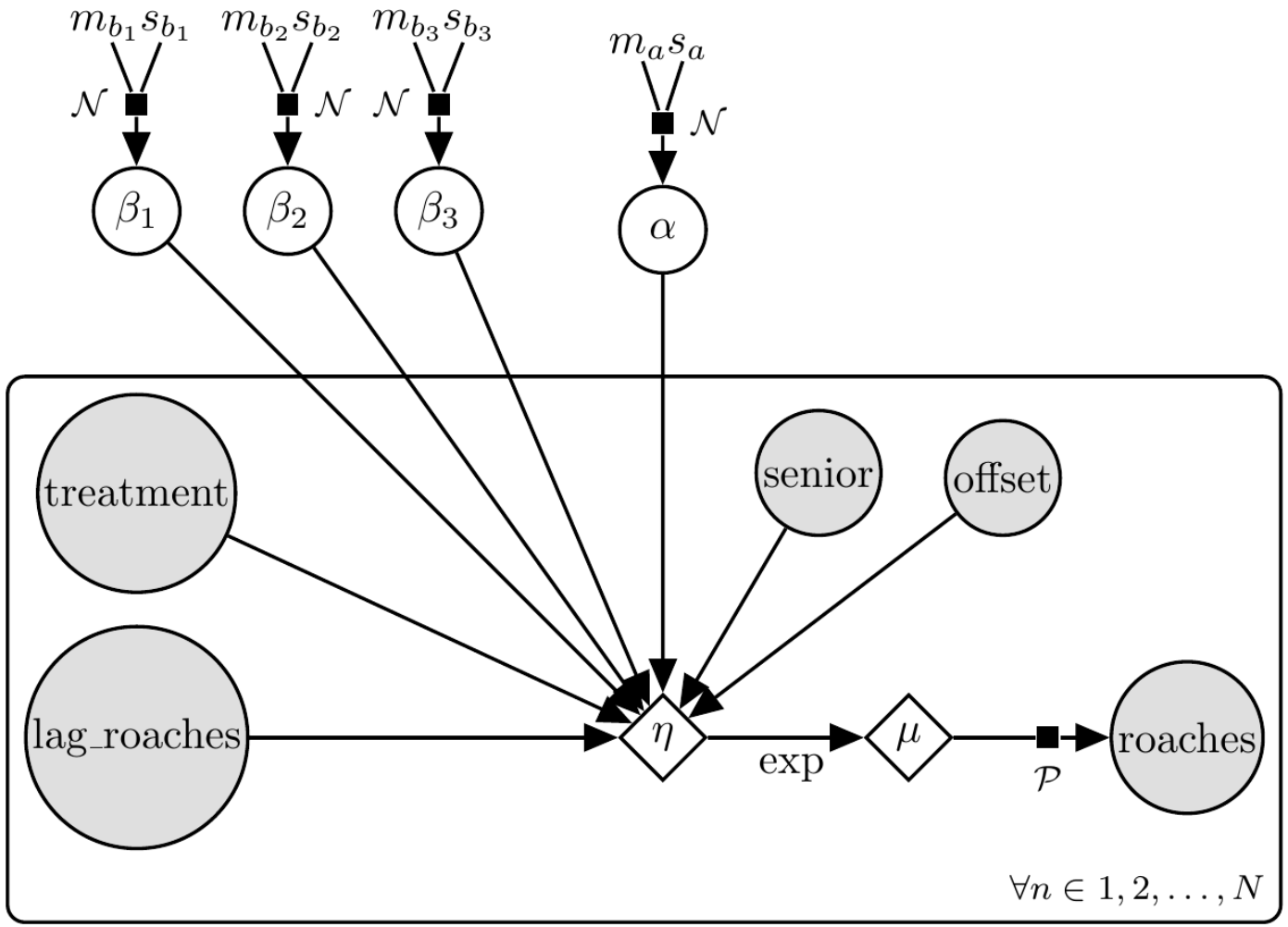
Do This Once on Each Computer You Use

- R comes with a terrible default coding for ordered factors in regressions known as “Helmert” contrasts
- Execute this once to change them to “treatment” contrasts, which is the conventional coding in the social sciences with dummy variables relative to a baseline category

```
cat('options(contrasts = c(unordered = "contr.treatment", ordered = "contr.treatment"))',
    file = "~/.Rprofile", sep = "\n", append = TRUE)
```

- Without this, you will get a weird rotation of the coefficients on dummy variables derived from ordered factors
- `"contr.sum"` is another reasonable (but rare) choice

Prior Predictive Distribution for Roach Study



Roach Model

Prior Predictive Distribution in Symbols

$$\alpha \sim \mathcal{N}\left(m_{\alpha}, s_{\alpha}\right)$$
$$\beta_1 \sim \mathcal{N}\left(m_{\beta_1}, s_{\beta_1}\right)$$
$$\beta_2 \sim \mathcal{N}\left(m_{\beta_2}, s_{\beta_2}\right)$$
$$\beta_3 \sim \mathcal{N}\left(m_{\beta_3}, s_{\beta_3}\right)$$
$$\forall n : \eta_n = \alpha + OFFSET_n + \beta_1 \times \log LAG_n + \beta_2 \times SENIOR_n + \beta_3 \times T_n$$
$$\forall n : \mu_n = e^{\eta_n}$$
$$\forall n : Y_n \sim \mathcal{P}oisson\left(\mu_n\right)$$

Breakout Rooms

Draw $S = 1000$ times (using `replicate`) from a prior predictive distribution

```
roaches <- roaches[roaches$roach1 > 0, ]; str(roaches)
```

```
## 'data.frame':    202 obs. of  5 variables:
##  $ y        : int  153 127 7 7 0 73 24 2 2 0 ...
##  $ roach1   : num  308 331.25 1.67 3 2 ...
##  $ treatment: int  1 1 1 1 1 1 1 0 0 0 ...
##  $ senior   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ exposure2: num  0.8 0.6 1 1 1.14 ...
```

- The average number of pre-treatment roaches (`roaches$roach1`) in a building was about 42 with a standard deviation of 75
- Use `log(roach1)` as the predictor
- The offset is `log(roaches$exposure2)`
- Remember to center the predictors, including the offset, when predicting

Posterior Distribution

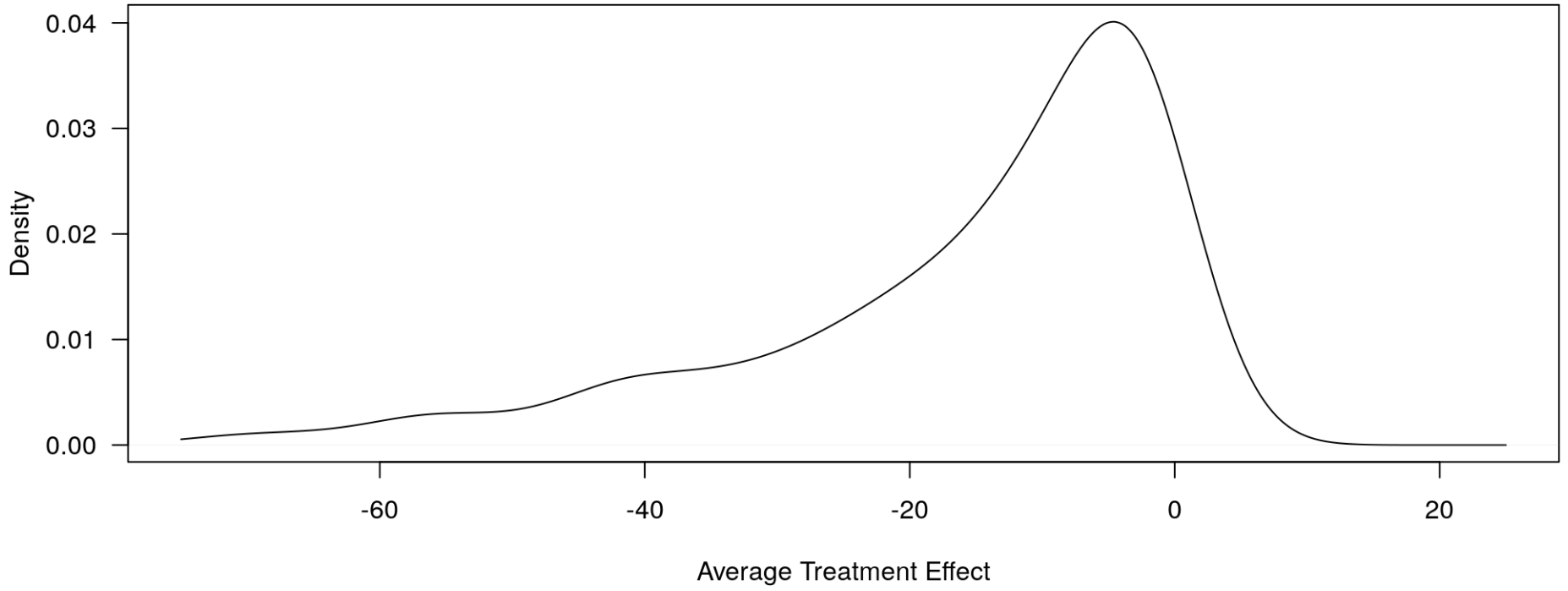
```
post <- stan_glm(y ~ senior + log(roach1) + treatment, data = roaches,
                 family = poisson, offset = log(exposure2), QR = TRUE,
                 prior_intercept = normal(location = log(42), scale = 4, autoscale = FALSE),
                 prior = normal(location = 0, scale = c(5, 5, 1), autoscale = FALSE))
```

```
print(post, digits = 2)
```

```
...
##             Median MAD_SD
## (Intercept)  1.57   0.05
## senior      -0.46   0.04
## log(roach1)  0.62   0.01
## treatment   -0.49   0.03
##
## Sample avg. posterior predictive distribution of y:
##          Median MAD_SD
## mean_PPD 31.54   0.55
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
...
```

Estimating Treatment Effects

```
df <- roaches; df$treatment <- 0
Y_0 <- posterior_linpred(post, newdata = df, offset = log(df$exposure2), transform = TRUE)
df$treatment <- 1
Y_1 <- posterior_linpred(post, newdata = df, offset = log(df$exposure2), transform = TRUE)
plot(density(colMeans(Y_1 - Y_0), from = -75, to = 25), xlab = "Average Treatment Effect", mai
```



Why NUTS Is Better than Other MCMC Samplers

- With Stan, it is almost always the case that things either go well or you get warning messages saying things did not go well
- Because Stan uses gradients, it scales well as models get more complex
- The first-order autocorrelation tends to be negative so you can get greater effective sample sizes (for mean estimates) than nominal sample size

```
round(bayesplot::neff_ratio(post), digits = 2)
```

```
## (Intercept)      senior log(roach1)   treatment
##        0.62        0.84        0.67        0.68
```

Divergent Transitions

- NUTS only uses first derivatives
- First order approximations to Hamiltonian physiscs are fine for if either the second derivatives are constant or the discrete step size is sufficiently small
- When the second derviatives are very not constant across Θ, Stan can (easily) mis-tune to a step size that is not sufficiently small and θ_k gets pushed to $\pm\infty$
- When this happens there will be a warning message, suggesting to increase `adapt_delta`
- When `adapt_delta` is closer to 1, Stan will tend to take smaller steps
- Unfortunately, even as `adapt_delta` $\lim 1$, there may be no sufficiently small step size and you need to try to reparameterize your model

Exceeding Maximum Treedepth

- When the step size is small, NUTS needs many (small) steps to cross the "typical" subset of Θ and hit the U-turn point
- Sometimes, NUTS has not U-turned when it reaches its limit of 10 steps (by default)
- When this happens there will be a warning message, suggesting to increase `max_treedepth`
- There is always a sufficiently high value of `max_treedepth` to allow NUTS to reach the U-turn point, but increasing `max_treedepth` by 1 approximately doubles the wall time to obtain S draws

Low Bayesian Fraction of Missing Information

- When the tails of the posterior PDF are very light, NUTS can have difficulty moving through Θ efficiently
- This will manifest itself in a low (and possibly unreliable) estimate of n_{eff}
- When this happens there will be a warning message, saying that the Bayesian Fraction of Missing Information (BFMI) is low
- In this situation, there is not much you can do except increase S or preferably reparameterize your model to make it easier for NUTS

Runtime Exceptions

- Sometimes you will get a “informational” (not error, not warning) message saying that some parameter that should be positive is zero or some parameter that should be finite is infinite
- This means that a 64bit computer could not represent the number accurately
- If it only happens a few times and only during the warmup phase, do not worry
- Otherwise, you might try to use functions that are more numerically stable, which is discussed throughout the Stan User Manual

Initial Values

- Sometimes a Markov Chain will not get started because the log-kernel evaluates to $-\infty$ or `NaN` at the initial values (which are randomly generated)
- You can provide you own initial values
- But it is usually easier to specify `init_r` to be some value between 0 and 2 (the default) which governs the range at which the initial values are drawn from on the unconstrained scale

Tail / Bulk ESS

- Sometimes you will get a message about the tail or bulk ESS being low
- Call `monitor` on an object produced by Stan to see the estimates
- I do not worry too much about it if ONLY the `lp__` margin is too low
- Otherwise, you can increase the number of iterations and / or chains
- But it does suggest that there is something in the parameterization of your model that would make it difficult for Stan to sample from the implied posterior distribution

ShinyStan

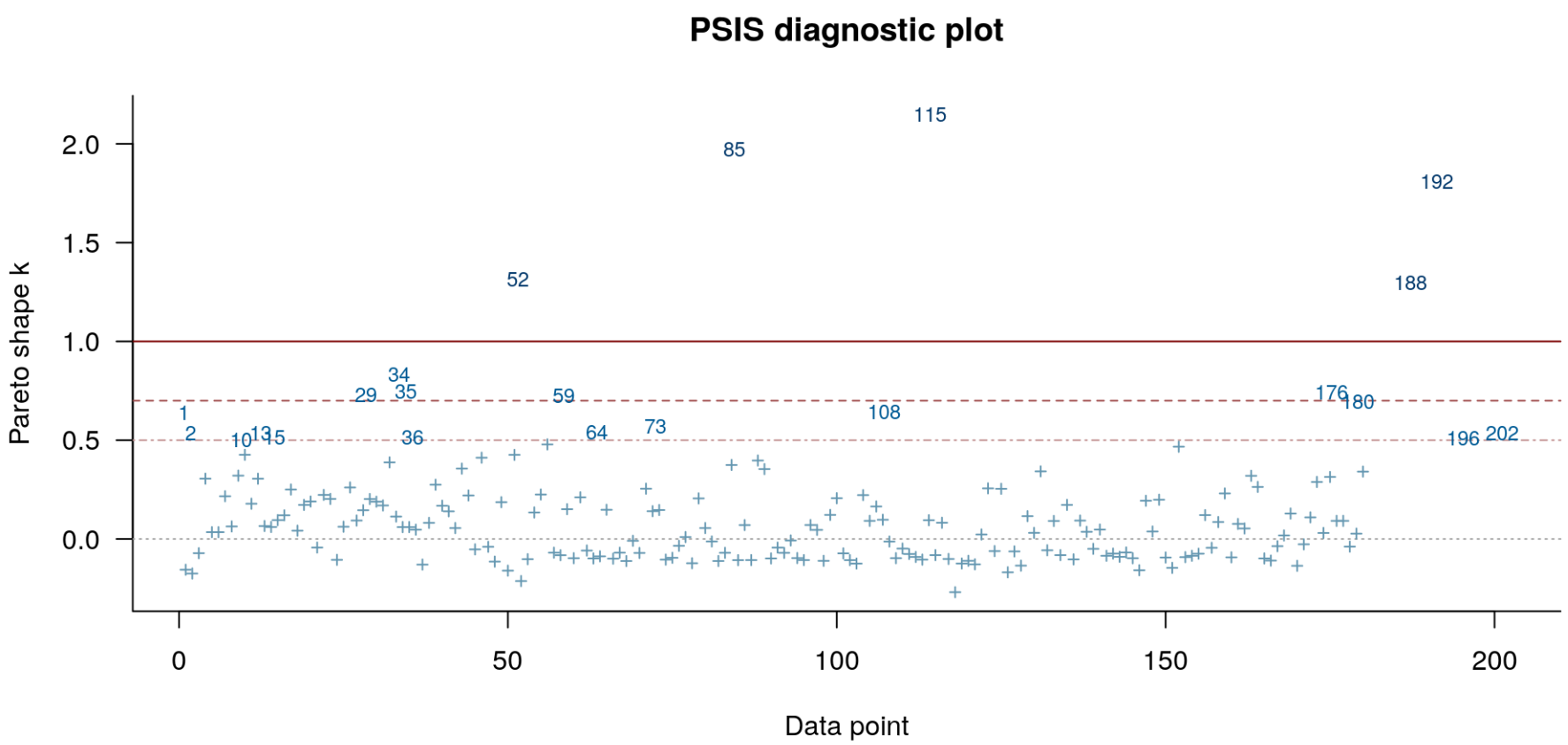
- ShinyStan can be launched on an object produced by rstanarm via

```
launch_shinystan(post)
```

- A webapp will open in your default web browser that helps you visualize the posterior distribution and diagnose problems

Sensitivity to Individual Observations

```
plot(loo(post), label_points = TRUE)
```



Numerical Assessment of Calibration

```
PPD <- posterior_predict(post); dim(PPD)
```

```
## [1] 4000  202
```

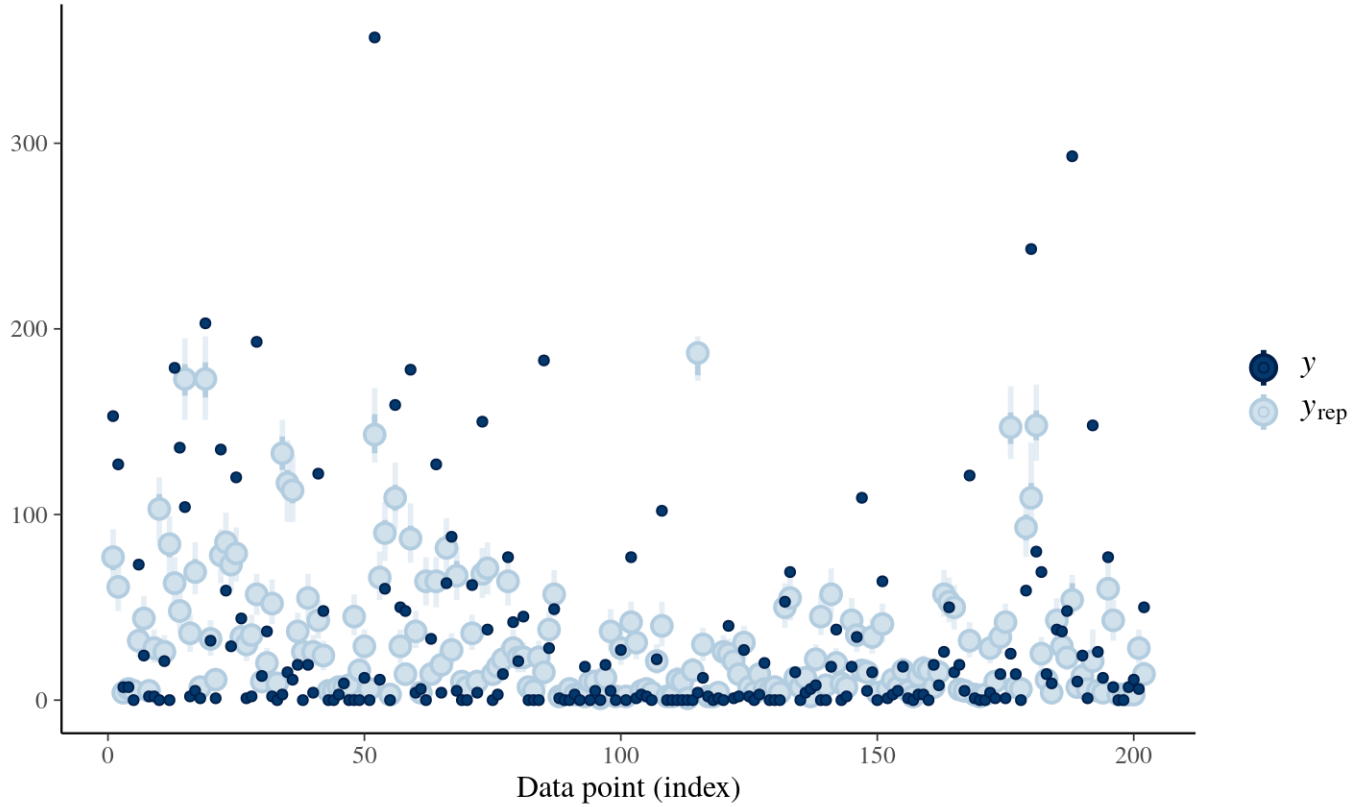
```
lower <- apply(PPD, MARGIN = 2, FUN = quantile, probs = 0.25)
upper <- apply(PPD, MARGIN = 2, FUN = quantile, probs = 0.75)
mean(roaches$y > lower & roaches$y < upper) # bad fit
```

```
## [1] 0.04950495
```

- Overal, the model is fitting the data poorly
- You will often overfit when you lazily use all predictors that are available in the dataset

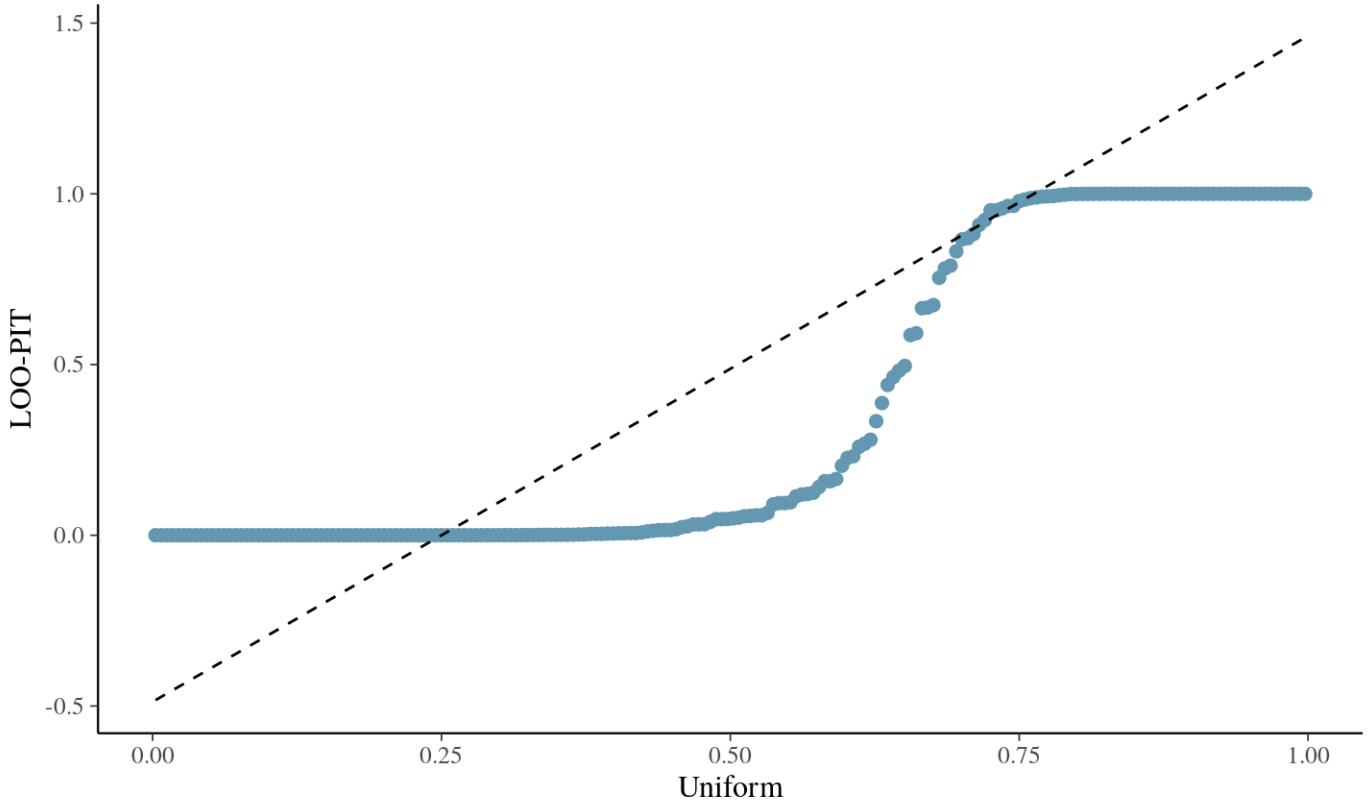
Leave-One-Out Based Intervals

```
library(bayesplot)
pp_check(post, plotfun = "loo_intervals")
```



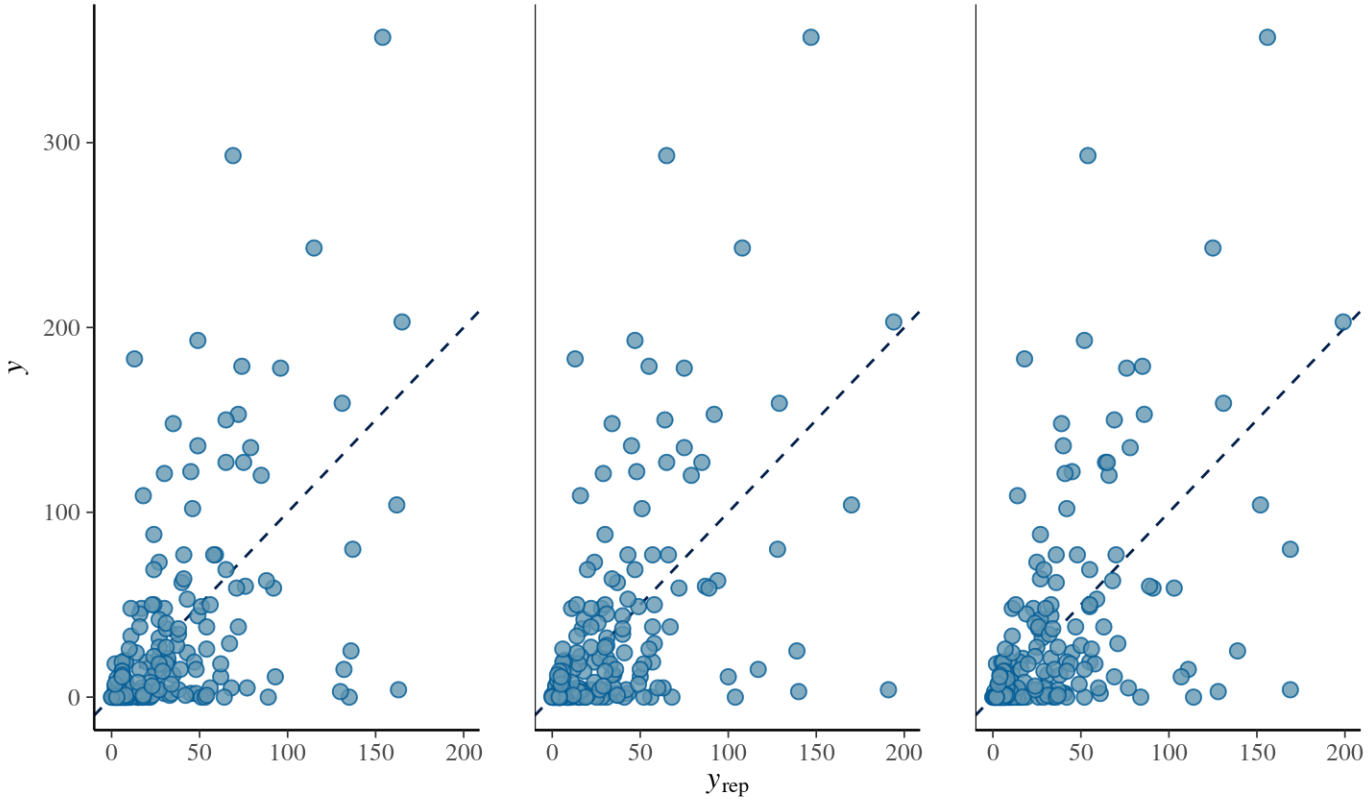
Leave-One-Out Probability Integral Transform

```
pp_check(post, plotfun = "loo_pit_qq")
```



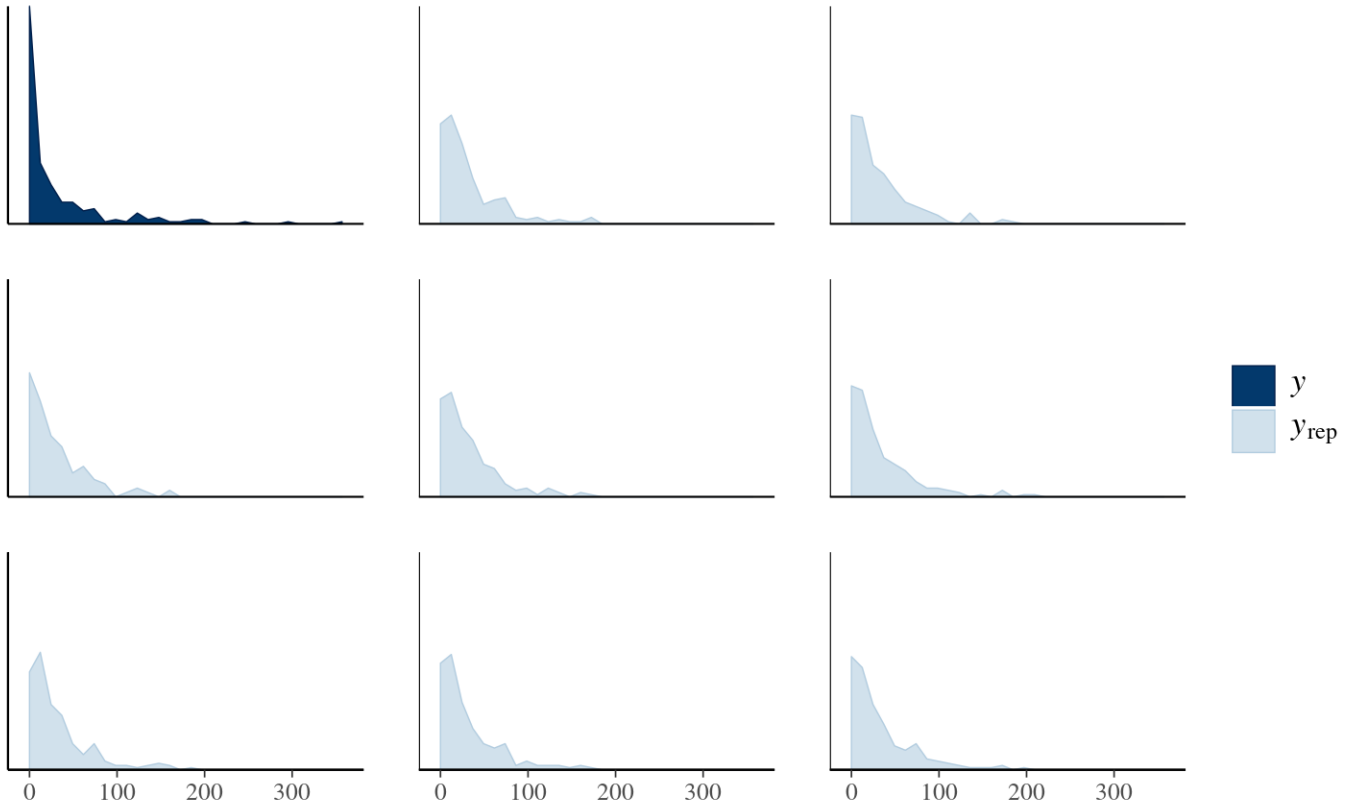
Scatterplots

```
pp_check(post, plotfun = "scatter")
```



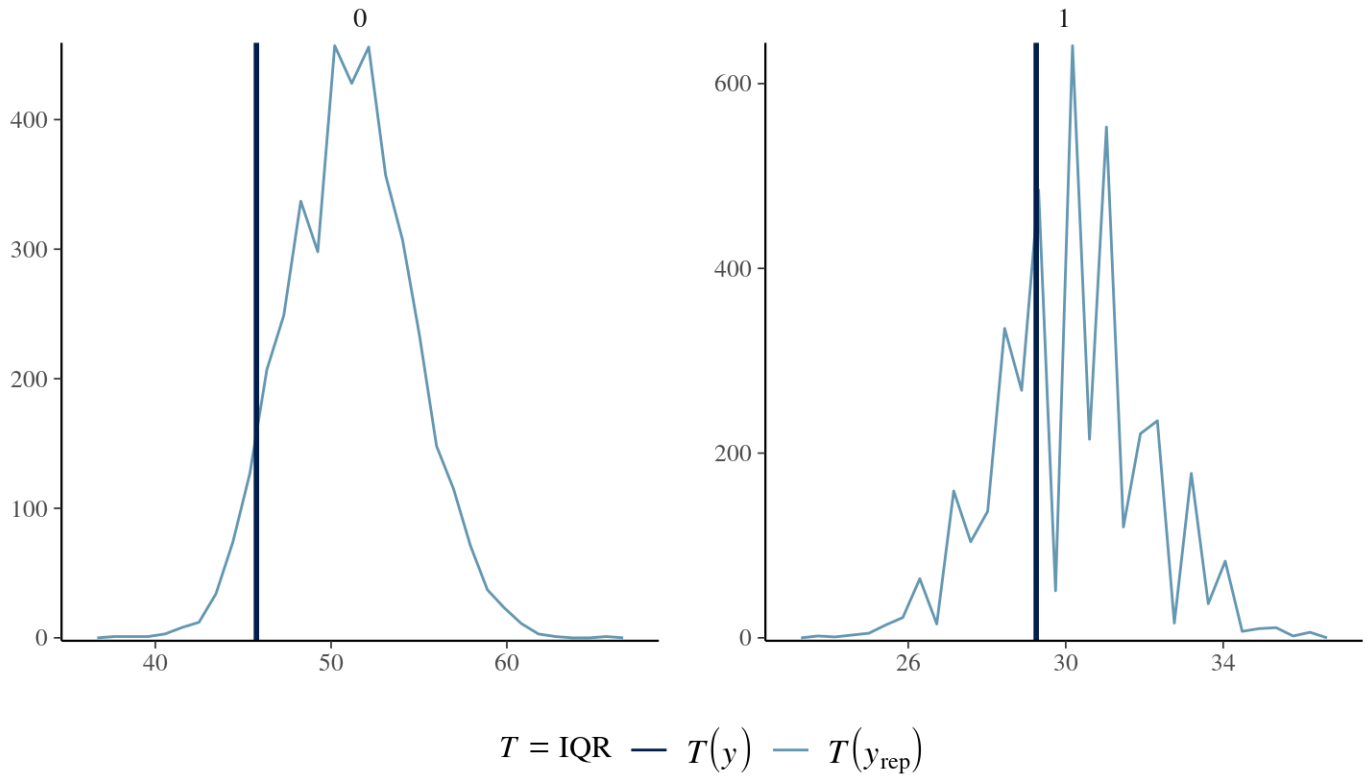
Frequency Polygons

```
pp_check(post, plotfun = "freqpoly")
```



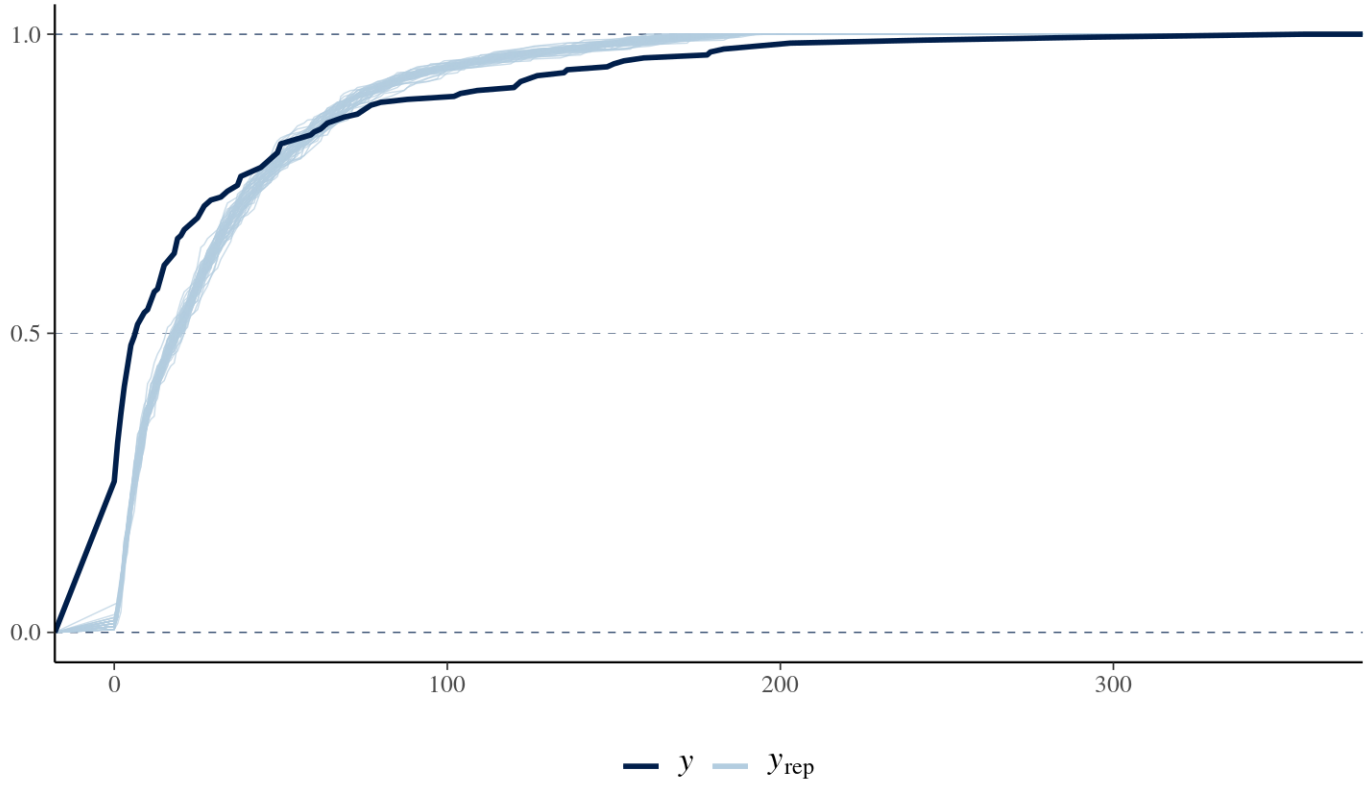
Frequency Polygons with Grouping

```
pp_check(post, plotfun = "stat_freqpoly_grouped", group = roaches$treatment, stat = "IQR") +
  legend_move("bottom")
```



Empirical CDF

```
pp_check(post, plotfun = "ecdf_overlay") + legend_move("bottom")
```



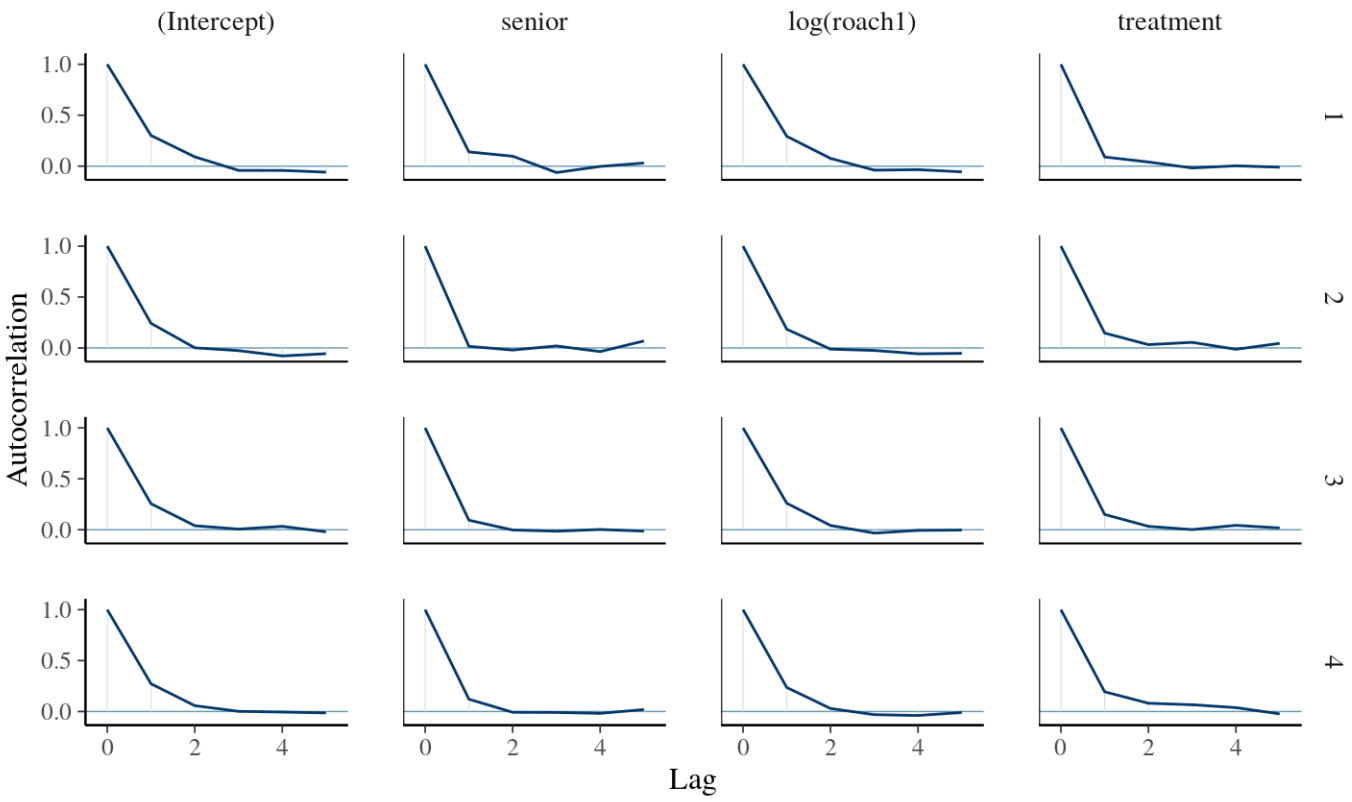
MCMC Performance Plots

```
format(available_mcmc(pattern = "data$", invert = TRUE))
```

```
##  [1] "mcmc_acf              " "mcmc_acf_bar          " "mcmc_areas            "
##  [4] "mcmc_areas_ridges     " "mcmc_combo            " "mcmc_dens             "
##  [7] "mcmc_dens_chains      " "mcmc_dens_overlay     " "mcmc_hex              "
## [10] "mcmc_hist             " "mcmc_hist_by_chain    " "mcmc_intervals        "
## [13] "mcmc_neff             " "mcmc_neff_hist        " "mcmc_nuts_acceptance  "
## [16] "mcmc_nuts_divergence  " "mcmc_nuts_energy       " "mcmc_nuts_stepsize    "
## [19] "mcmc_nuts_treedepth   " "mcmc_pairs             " "mcmc_parcoord         "
## [22] "mcmc_rank_hist        " "mcmc_rank_overlay     " "mcmc_recover_hist     "
## [25] "mcmc_recover_intervals" "mcmc_recover_scatter  " "mcmc_rhat             "
## [28] "mcmc_rhat_hist        " "mcmc_scatter          " "mcmc_trace            "
## [31] "mcmc_trace_highlight  " "mcmc_violin           "
```

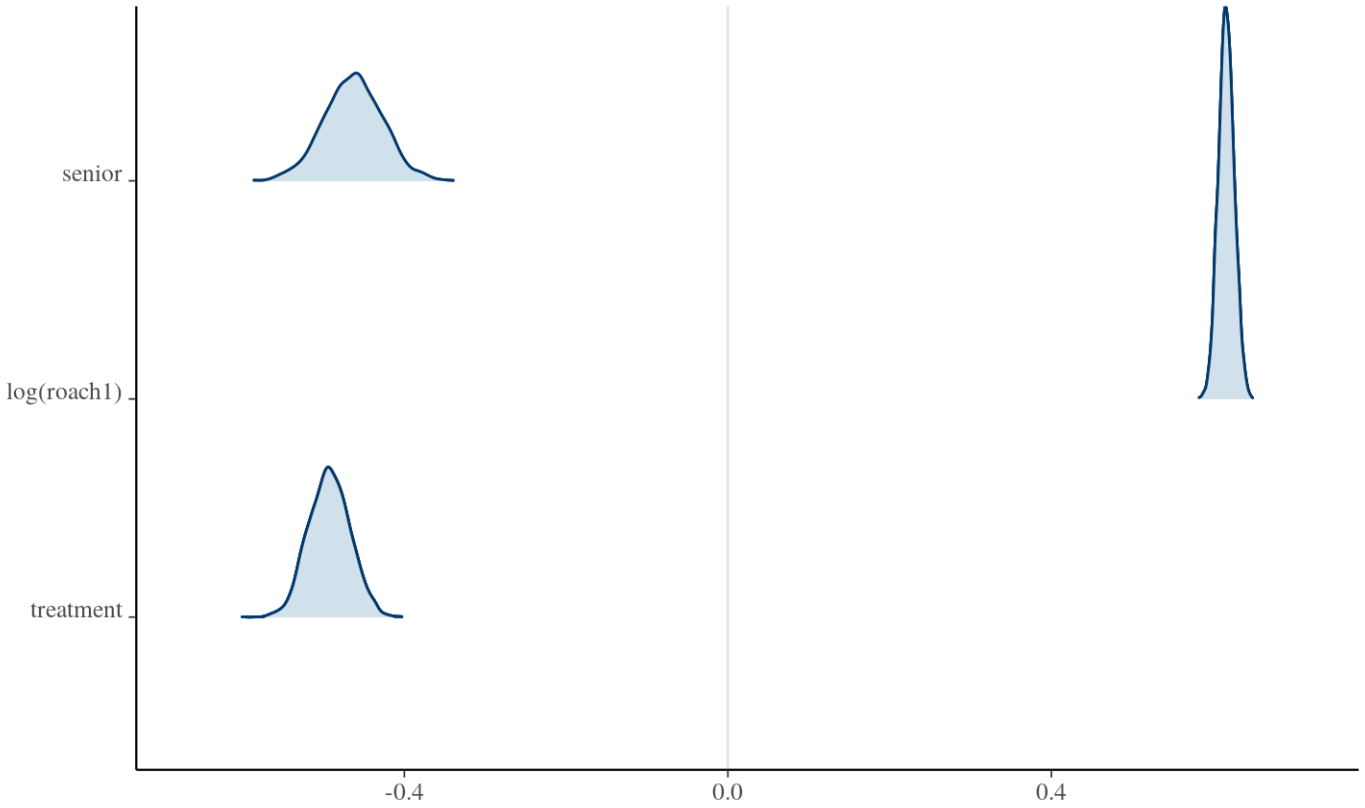
Autocorrelation Function

```
mcmc_acf(post, lags = 5)
```



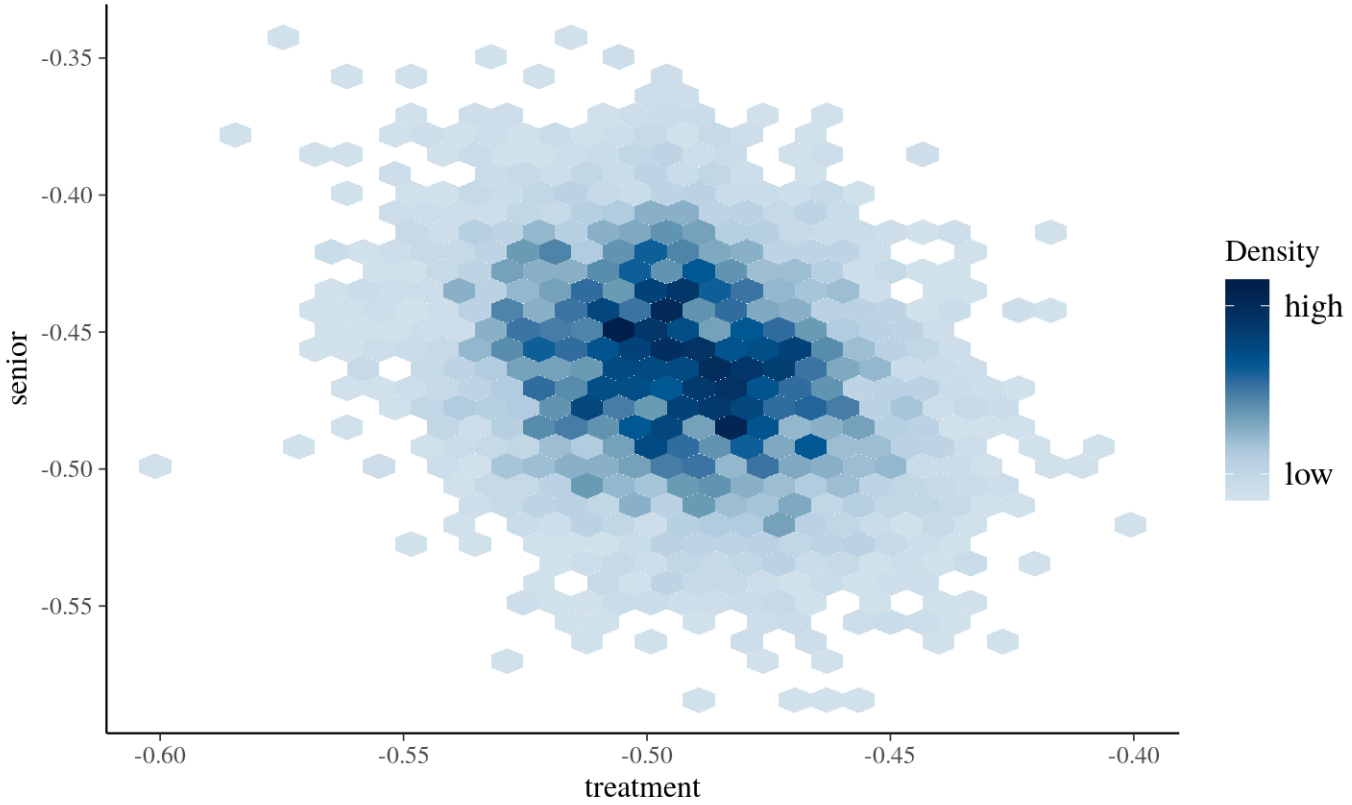
Ridges

```
mcmc_areas_ridges(post, regex_pars = "^[^(]") # exclude (Intercept)
```



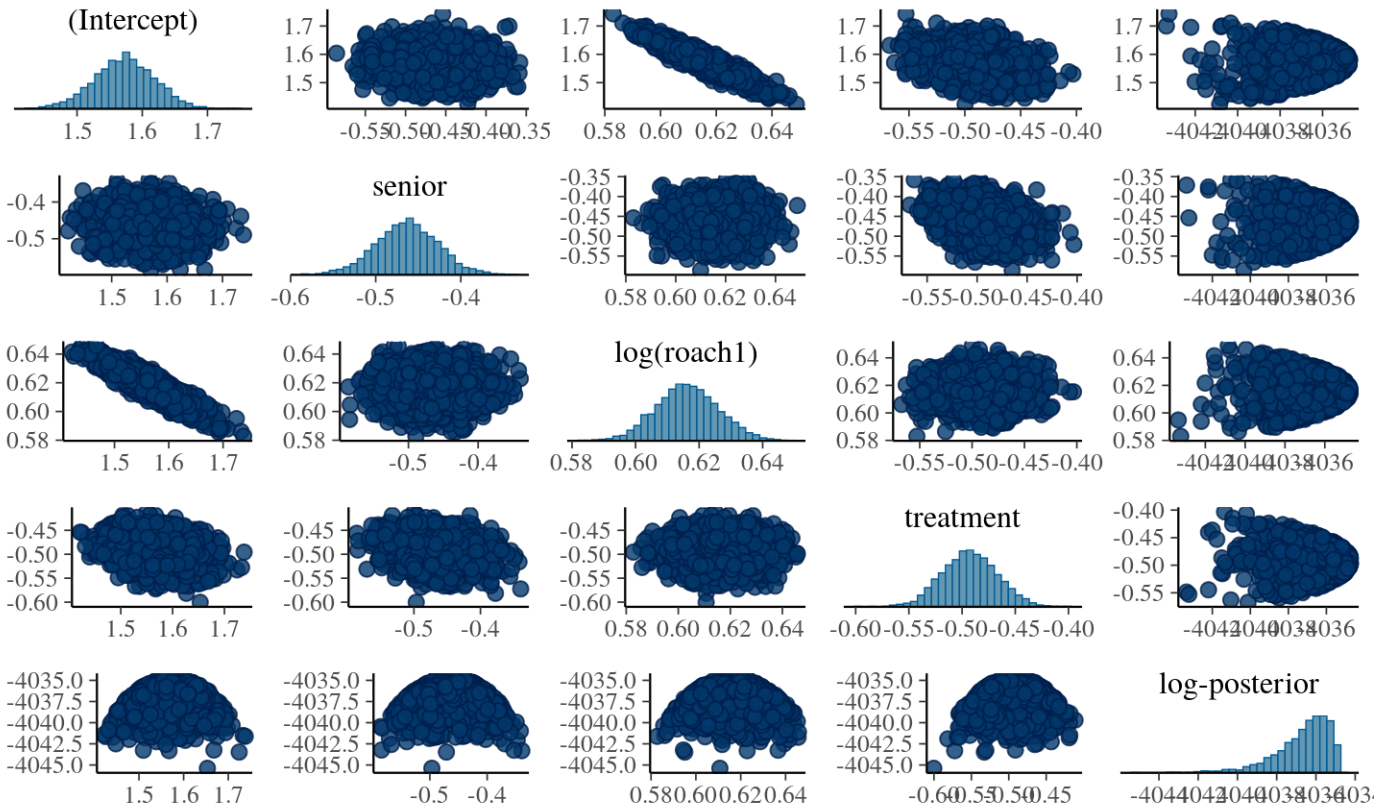
Hexograms

```
mcmc_hex(post, pars = c("treatment", "senior"))
```



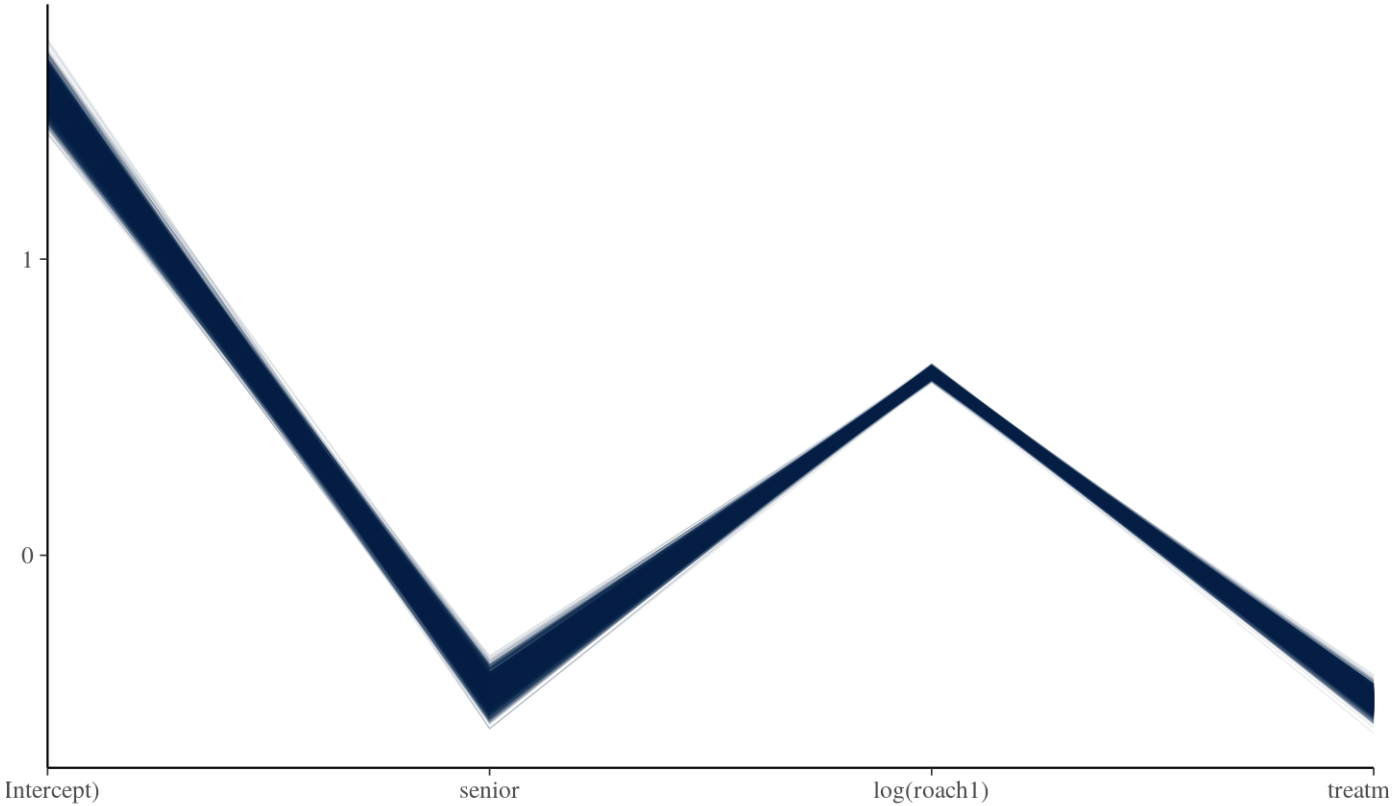
Pairs

```
pairs(post)
```

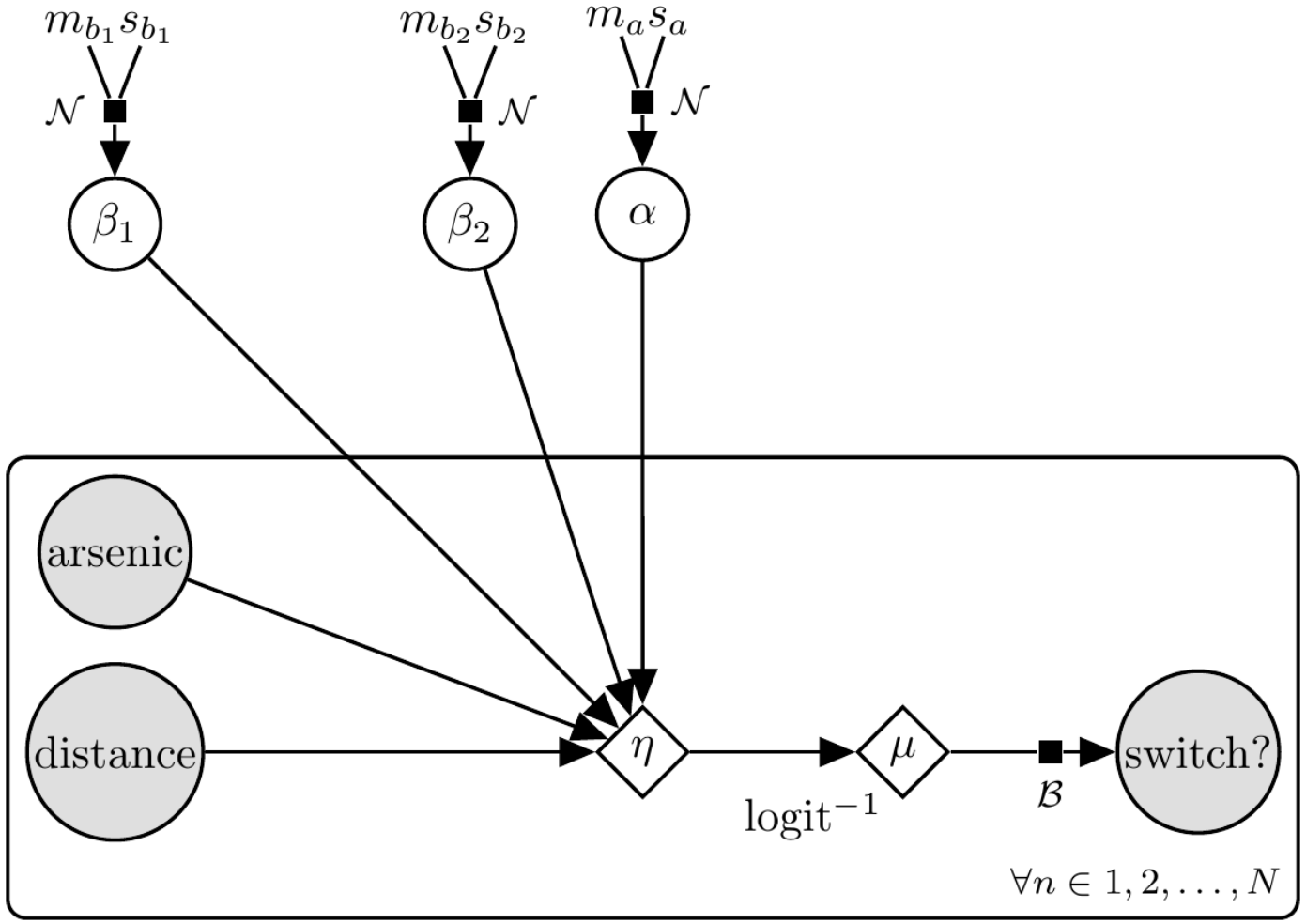


Parallel Coordinates

```
mcmc_parcoord(post, alpha = 0.1, np = nuts_params(post),
              np_style = parcoord_style_np(div_color = "red", div_size = 1, div_alpha = 1))
```



Prior Predictive Distribution for Well Switching

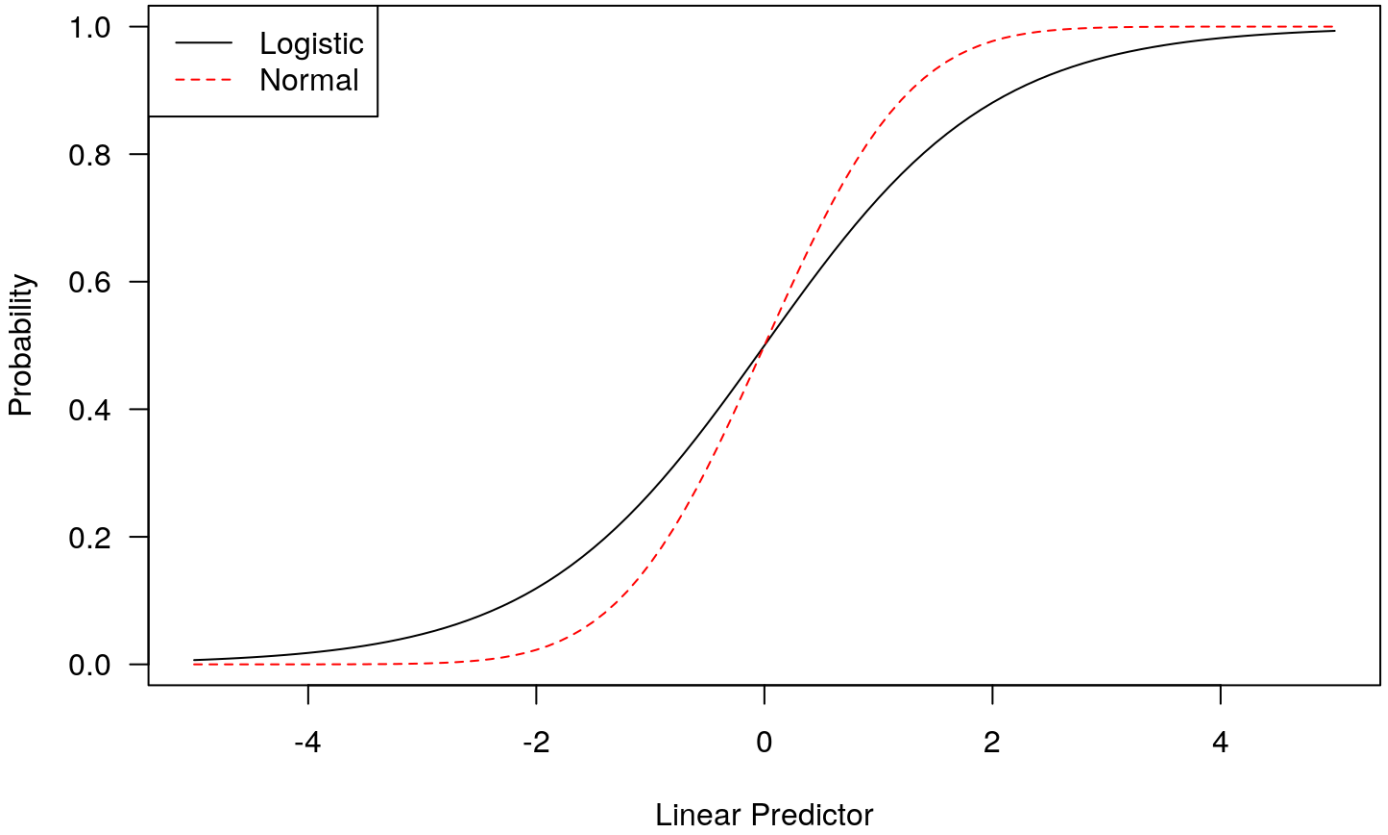


Well Switching Model

Prior Predictive Distribution in Symbols

$$\alpha \sim \mathcal{N}\left(m_{\alpha}, s_{\alpha}\right)$$
$$\beta_1 \sim \mathcal{N}\left(m_{\beta_1}, s_{\beta_1}\right)$$
$$\beta_2 \sim \mathcal{N}\left(m_{\beta_2}, s_{\beta_2}\right)$$
$$\forall n : \eta_n = \alpha + \beta_1 \times ARSENIC_n + \beta_2 \times DISTANCE_n$$
$$\forall n : \epsilon_n \sim \mathcal{L}ogistic\left(0, 1\right)$$
$$\forall n : u_n = \eta_n + \epsilon_n$$
$$\forall n : Y_n = u_n > 0$$

Inverse Link Functions



Breakout Rooms

Draw $S = 1000$ times (using `replicate`) from a prior predictive distribution

```
str(wells)
```

```
## 'data.frame':    3020 obs. of  5 variables:
##  $ switch : int  1 1 0 1 1 1 1 1 1 1 ...
##  $ arsenic: num  2.36 0.71 2.07 1.15 1.1 3.9 2.97 3.24 3.28 2.52 ...
##  $ dist   : num  16.8 47.3 21 21.5 40.9 ...
##  $ assoc  : int  0 0 0 0 1 1 1 0 1 1 ...
##  $ educ   : int  0 0 10 12 14 9 4 10 0 0 ...
```

- You do not have to use `assoc` or `educ`
- To draw from the logistic distribution use `rlogis`
- Remember to center the predictors when predicting

Posterior Distribution

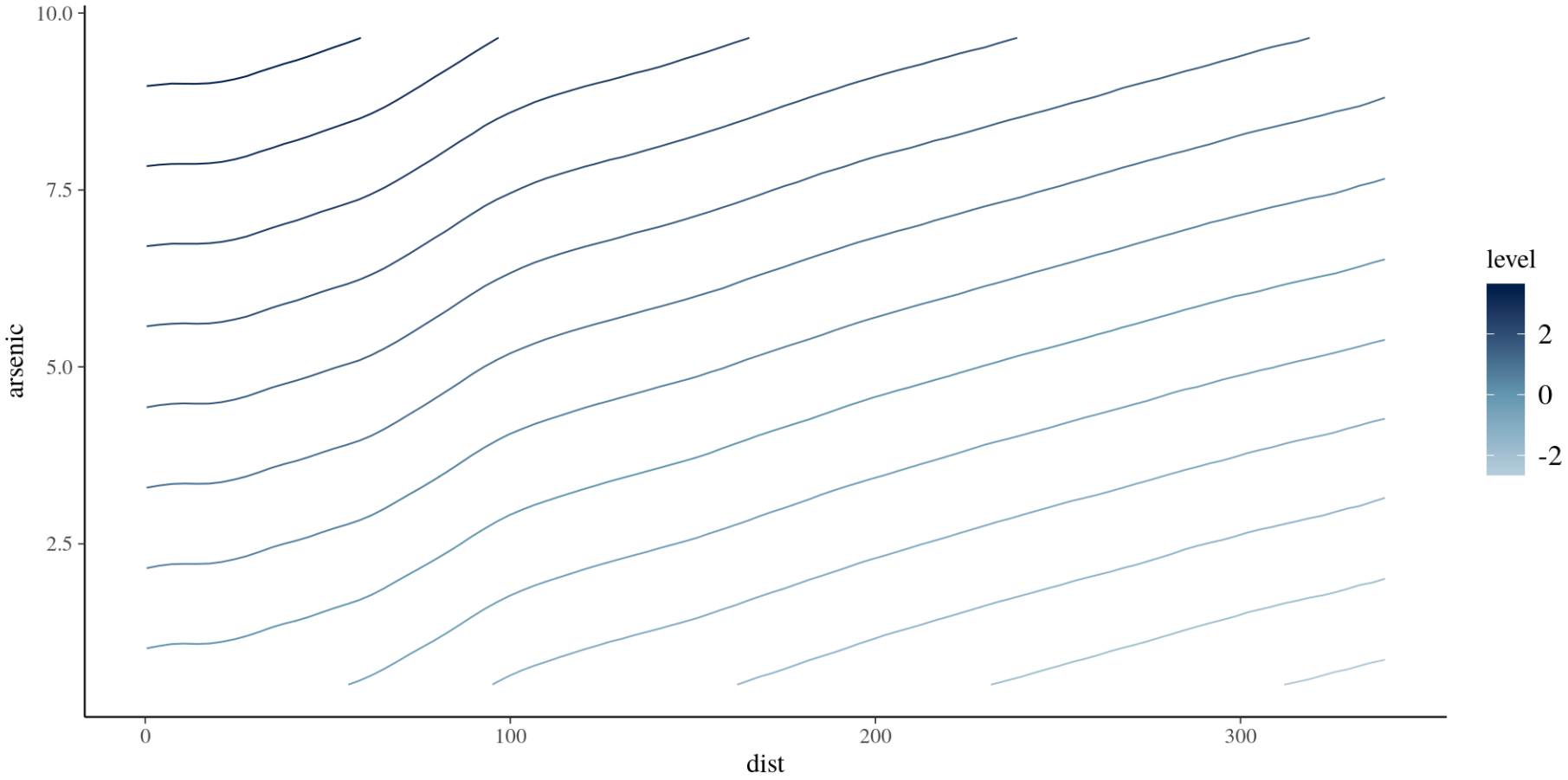
```
post <- stan_gamm4(switch ~ s(dist, arsenic), data = wells, family = binomial)
```

```
print(post, digits = 2)
```

```
...
## (Intercept)          0.33   0.04
## s(dist,arsenic).1   -0.03   0.56
## s(dist,arsenic).2    0.00   0.55
## s(dist,arsenic).3    0.00   0.55
## s(dist,arsenic).4    0.00   0.53
## s(dist,arsenic).5   -0.06   0.53
## s(dist,arsenic).6   -0.02   0.52
## s(dist,arsenic).7    0.00   0.52
## s(dist,arsenic).8   -0.04   0.55
## s(dist,arsenic).9   -0.08   0.55
## s(dist,arsenic).10 -0.04   0.53
## s(dist,arsenic).11  0.05   0.57
## s(dist,arsenic).12  0.09   0.55
## s(dist,arsenic).13 -0.31   0.62
## s(dist,arsenic).14 -0.23   0.59
## s(dist,arsenic).15  0.04   0.55
## s(dist,arsenic).16  0.02   0.55
## s(dist,arsenic).17 -0.03   0.53
## s(dist,arsenic).18 -0.11   0.54
## s(dist,arsenic).19  0.08   0.53
## s(dist,arsenic).20  0.02   0.45
## s(dist,arsenic).21 -0.04   0.47
## s(dist,arsenic).22 -0.02   0.53
## s(dist,arsenic).23 -0.64   0.50
## s(dist,arsenic).24 -0.20   0.41
## s(dist,arsenic).25 -0.17   0.56
## s(dist,arsenic).26  0.08   0.54
## s(dist,arsenic).27 -0.02   0.42
## s(dist,arsenic).28  7.97   1.06
## s(dist,arsenic).29  6.80   2.11
##
## Smoothing terms:
##                                 Median MAD_SD
## smooth_sd[s(dist,arsenic)1] 0.65   0.45
## smooth_sd[s(dist,arsenic)2] 4.62   1.19
##
## Sample avg. posterior predictive distribution
##          Median MAD_SD
## mean_PPD 0.58   0.01
##
```

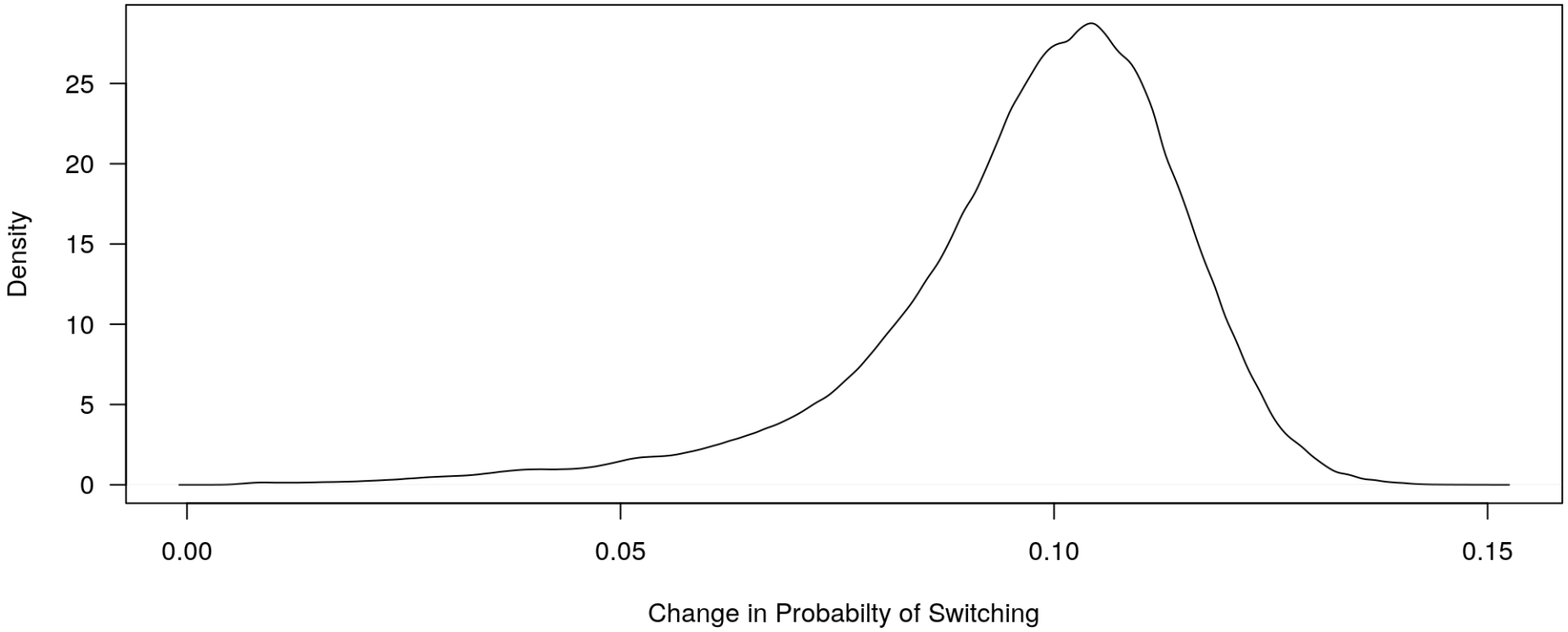
Nonlinear Plot

```
plot_nonlinear(post)
```



Plotting the Effect of an Increase in Arsenic

```
mu_0 <- posterior_linpred(post, transform = TRUE)
df <- wells; df$arsenic <- df$arsenic + 1
mu_1 <- posterior_linpred(post, newdata = df, transform = TRUE)
plot(density(mu_1 - mu_0), main = "", xlab = "Change in Probabilty of Switching")
```



Effective Number of Parameters

```
loo(post) # nominal number of estimated parameters is 32
```

```
##
## Computed from 4000 by 3020 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo  -1967.7 15.3
## p_loo         6.7  0.2
## looic      3935.4 30.6
## ------
## Monte Carlo SE of elpd_loo is 0.0.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     3018  99.9%   2220
##  (0.5, 0.7]   (ok)          2   0.1%   3759
##    (0.7, 1]   (bad)         0   0.0%   <NA>
##    (1, Inf)   (very bad)    0   0.0%   <NA>
##
## All Pareto k estimates are ok (k < 0.7).
## See help('pareto-k-diagnostic') for details.
```

A Binomial Model for Romney vs Obama in 2012

```
poll <- readRDS("GooglePoll.rds") # WantToWin is coded as 1 for Romney and 0 for Obama
library(dplyr)
collapsed <- filter(poll, !is.na(WantToWin)) %>%
             group_by(Region, Gender, Urban_Density, Age, Income) %>%
             summarize(Romney = sum(grepl("Romney", WantToWin)), Obama = n() - Romney) %>%
             na.omit
```

```
post <- stan_glm(cbind(Romney, Obama) ~ ., data = collapsed, family = binomial(link = "probit"), QR = TRUE)
```

```
## Warning in .local(object, ...): some chains had errors; consider specifying chains = 1 to
## debug
```

```
## here are whatever error messages were returned
```

```
print(post, digits = 2)
```

```
...
## $xbar
##  [1]  0.57  0.85  1.48  0.98  0.82  1.77  0.44  0.44  0.73 -1.04 -1.58 -0.99 -1.41  1.16
## [15] -1.02 -0.34
##
## $dense_X
## [1] TRUE
##
## $family
## [1] 5
##
## $link
## [1] 2
##
## $has_weights
## [1] FALSE
##
## $has_offset
## [1] FALSE
...
```

Same Linear Predictor, Logistic Link, R^2 Prior

```
post <- stan_polr(WantToWin ~ Region + Gender + Urban_Density + Age + Income,
                  data = poll, prior = R2(0.3, what = "mode"), init_r = 1)
```

```
print(post, digits = 2)
```

```
...
## (Intercept)            -0.52   0.14
## RegionNORTHEAST        -0.14   0.09
## RegionSOUTH             0.30   0.07
## RegionWEST             -0.14   0.08
## GenderMale              0.37   0.06
## Urban_DensitySuburban -0.20   0.09
## Urban_DensityUrban    -0.50   0.09
## Age25-34                0.10   0.10
## Age35-44                0.52   0.10
## Age45-54                0.82   0.09
## Age55-64                0.83   0.09
## Age65+                  1.30   0.11
## Income25,000-49,999    -0.11   0.08
## Income50,000-74,999    -0.07   0.09
## Income75,000-99,999    -0.09   0.14
## Income100,000-149,999   0.17   0.28
## Income150,000+          0.82   1.01
##
...
```

What Does `QR = TRUE` Do?

- Let the vector of linear predictions in a GLM be $\boldsymbol{\eta} = \mathbf{X}\boldsymbol{\beta}$
- If we apply the QR decomposition to the linear predictor,

$$\boldsymbol{\eta} = \overbrace{\mathbf{QR}}^{\mathbf{X}}\boldsymbol{\beta} = \mathbf{Q}\overbrace{\boldsymbol{\theta}}^{\mathbf{R}\boldsymbol{\beta}}$$

- When you specify `QR = TRUE` in `stan_glm` (or use `stan_lm` or `stan_polr`), **rstanarm** internally does a GLM using $\mathbf{Q}$ as the matrix of predictors instead of $\mathbf{X}$ to get the posterior distribution of $\boldsymbol{\theta}$ and then pre-multiplies each posterior draw of $\boldsymbol{\theta}$ by $\mathbf{R}^{-1}$ to get a posterior draw of $\boldsymbol{\beta} = \mathbf{R}^{-1}\boldsymbol{\theta}$
- Doing so makes it easier for NUTS to sample from the posterior distribution (of $\boldsymbol{\theta}$) efficiently because the columns of $\mathbf{Q}$ are orthogonal, whereas the columns of $\mathbf{X}$ are not