Ordinal Regression Model

James Totterdell

2021-10-25

Contents

1 Model

Let $Y_i \in \{1, ..., 8\}$ be the day 14 outcome for participant i where 1 is best and 8 is worst (death). Let $\tau_i \in \{1, 2, 3\}$ denote treatment arm and x_i the corresponding design encoding. For the (reduced) primary model we specify

$$\mathbb{P}[Y_i \leq k | \alpha, \beta; x_i] = \text{logit}^{-1}(\alpha_k + x_i^\mathsf{T}\beta), \quad i = 1, 2, ..., \ k = 1, ..., 7,$$

where $\{\alpha_k\}$ is increasing in k and $\beta = (\beta_1, \beta_2)^\mathsf{T}$ represent the cumulative log-odds ratio of treatments 2 and 3 relative to 1.

Define the outcome level probabilities in the control group by $\pi_k = \mathbb{P}[Y = k | \alpha]$, for $k = 1, ..., 8$ where

$$\pi_k = \begin{cases} 1 - \text{logit}^{-1}(\alpha_1) & \text{if } k = 1 \\ \text{logit}^{-1}(\alpha_{k-1}) - \text{logit}^{-1}(\alpha_k) & \text{if } k \in \{2, ..., 7\} \\ \text{logit}^{-1}(\alpha_7) & \text{if } k = 8 \end{cases}$$

We specify the priors on the level probabilities as Dirichlet on the simplex,

$$\pi \sim \text{Dirichlet}(\kappa)$$

for some $\kappa = (\kappa_1, ..., \kappa_8)$ where the sum $\sum \kappa_k$ gives the concentration. For uninformative we would usually choose something like $\sum \kappa_k = 1$.

For the coefficients we specify

$$\beta \sim \text{Normal}(0, \sigma_\beta)$$

for some chosen value σ_β.

2 Stan code

Given that we will only be simulating discrete covariates, we can collapse the model by covariate pattern. E.g. rather than model $Y_i \in \{1, ..., 8\}$ for $i = 1, ..., n$ we can collapse into J covariate patterns and model $Y_j \in \{1, ..., 8\}$ weighted by n_j with $\sum_j n_j = n$.

Note, in the following code `prior_counts` corresponds to κ, and `prior_sd` to σ_β in the above. Also `c` corresponds to α.

2.1 Per-participant implementation

```
writeLines(readLines("../stan/ordmod.stan"))
```

```
## functions {
##   vector make_cutpoints(vector p, real scale) {
##     int C = rows(p) - 1;
##     vector[C] cutpoints;
##     real running_sum = 0;
##     for(c in 1:C) {
##       running_sum += p[c];
##       cutpoints[c] = logit(running_sum);
##     }
##     return scale * cutpoints;
##   }
## }
##
## data {
##   int N; // number of records
##   int K; // number of response levels
##   int P; // number of covariates
##   int<lower=1,upper=K> y[N];
##   matrix[N, P] X; // design matrix
##   vector[K] prior_counts; // prior for Dirichlet on cuts
##   vector[P] prior_sd; // prior SD for beta coefficients
## }
##
## parameters {
##   simplex[K] pi;    // category probabilities
##   vector[P] beta_raw;   // covariate coefficients
## }
##
## transformed parameters {
##   vector[K-1] c; // cut-points on unconstrained scale
##   vector[P] beta = prior_sd .* beta_raw;
##   c = make_cutpoints(pi, 1);
## }
##
## model {
##   // Linear predictor
##   vector[N] eta = X * beta;
##
##   // Prior model
##   pi ~ dirichlet(prior_counts);
##   beta_raw ~ normal(0, 1);
##
```

```
##   // Observational model
##   y ~ ordered_logistic(eta, c);
## }
##
## generated quantities {
##   int<lower=1, upper=K> y_ppc[N];
##   // posterior predictive draws
##   for (n in 1:N)
##     y_ppc[n] = ordered_logistic_rng(X[n]*beta, c);
## }
```

2.2 Per-pattern implementation

```
writeLines(readLines("../stan/ordmodagg.stan"))
```

```
## // James Totterdell
## // Date: 2021-10-13
## //
## // if undertaking simulations we would generally not have
## // any continuous covariates, but instead just discrete treatment groups
## // Therefore, to save computation time we can aggregate over
## // the covariate patterns and weight by counts.
## // E.g. rather than analyse N = 2,100 participants 1:1:1 to 3 arms,
## // We can analyse the 3 covariate patterns weighted by sample size.
##
## functions {
##   // calculate the multinomial coefficient
##   // required if want to include constants in log likelihood evaluation
##   // - y: observed counts
##   real multinomial_coef(vector y) {
##     return lgamma(sum(y) + 1) - sum(lgamma(y + 1));
##   }
##
##   // make_cutpoints
##   // - p: outcome level probabilities
##   vector make_cutpoints(vector p, real scale) {
##     int C = rows(p) - 1;
##     vector[C] cutpoints;
##     real running_sum = 0;
##     for(c in 1:C) {
##       running_sum += p[c];
##       cutpoints[c] = logit(running_sum);
##     }
##     return scale * cutpoints;
##   }
##
##   // Pr(y == k) for k=1,...,K
##   // - c: cut-points for outcome levels
##   // - eta: linear predictor for each pattern
##   matrix Pr(vector c, vector eta) {
##     int N = num_elements(eta);
##     int K = num_elements(c) + 1;
##     matrix[N, K] out;
##     // for stability, work on log-scale
```

```
##     for (n in 1:N) {
##       out[n, 1] = log1m_exp(-log1p_exp(-(eta[n] - c[1]))); // ln(1 - inv_logit(eta[n] - c[1]))
##       out[n, K] = -log1p_exp(-(eta[n] - c[K-1]));          // ln(inv_logit(eta[n] - c[K-1]))
##       for (k in 2:(K - 1)) {
##         // ln(inv_logit(eta[n] - c[k-1]) - inv_logit(eta[n] - c[k]))
##         out[n, k] = log_diff_exp(-log1p_exp(-(eta[n] - c[k-1])), -log1p_exp(-(eta[n] - c[k])));
##       }
##     }
##     return exp(out);
##   }
##
##   // log-likelihood (multinomial)
##   // - p: level probabilities for each pattern
##   // - y: observed count for each level for each pattern
##   vector log_lik(matrix p, matrix y) {
##     int N = rows(y);
##     int K = cols(y);
##     vector[N] out;
##     for(n in 1:N) {
##       out[n] = 0.0;
##       for(k in 1:K) {
##         out[n] += y[n, k] * log(p[n, k]);
##       }
##     }
##     return out;
##   }
## }
##
## data {
##   int N; // number of records
##   int K; // number of response levels
##   int P; // number of covariates
##   matrix[N, K] y; // response record x level
##   matrix[N, P] X; // design matrix
##   vector[K] prior_counts; // prior for Dirichlet on cuts
##   vector[P] prior_sd; // prior SD for beta coefficients
## }
##
## transformed data {
##   vector[N] multinom_coef;
##   for(i in 1:N)
##     multinom_coef[i] = multinomial_coef(to_vector(y[i]));
## }
##
## parameters {
##   simplex[K] pi;      // outcome level probabilities for reference level
##   vector[P] beta_raw; // covariate coefficients
## }
##
## transformed parameters {
##   vector[K-1] alpha;    // outcome level cuts for reference pattern
##   matrix[N, K] p;   // matrix of level probabilities for covariate pattern and outcome level
##   vector[N] loglik; // store covariate pattern loglik contribution
##   vector[P] beta = prior_sd .* beta_raw;
```

```
##   vector[N] eta = X * beta;
##   alpha = make_cutpoints(pi, 1);
##   p = Pr(alpha, eta);
##   loglik = multinom_coef + log_lik(p, y);
## }
##
## model {
##   // Prior model
##   target += normal_lpdf(beta_raw | 0, 1);
##   target += dirichlet_lpdf(pi | prior_counts);
##   // Observational model
##   target += loglik;
## }
```

3 Example Use

3.1 Example 1

Per-pattern has much lower computational overhead than per-participant (3 likelihood evaluations vs 2,100).

If this is still too computationally demanding for sims, then can look at Laplace approximation for the posterior instead.

```
library(Hmisc)
library(data.table)
library(cmdstanr)
library(posterior)
library(bayesplot)

ordmod <- cmdstan_model("../stan/ordmod.stan")
ordmodagg <- cmdstan_model("../stan/ordmodagg.stan")

# Simulate some outcome data
N <- 2100
p <- rbind(
  rep(1/ 8, 8),
  pomodm(p = rep(1/ 8, 8), odds.ratio = 2), # Odds ratio for 2 for arm 2
  pomodm(p = rep(1/ 8, 8), odds.ratio = 0.5)) # Odds ratio of 0.5 for arm 3
x <- factor(1:3)
n <- rep(N/3, 3)
y <- matrix(0, 3, 8)

xx <- rep(x, times = N/3)
for(i in 1:N) {
  y[i] <- sample.int(8, 1, prob = p[xx[i], ])
}
D <- data.table(x = x, y = y)
Dagg <- D[, .N, keyby = .(x, y)]              # Per-participant uses long format
Dwide <- dcast(Dagg, x ~ y, value.var = 'N') # Aggregated model uses wide format

# Sanity check
mle_fit <- MASS::polr(ordered(y) ~ x, data = D)

X <- model.matrix( ~ x, data = D)[, -1]
ordmoddat <- list(
  y = D$y,
  N = nrow(D),
  K = 8,
  P = ncol(X),
  X = X,
  prior_counts = rep(1/8, 8),
  prior_sd = rep(1, 2)
)
ordmodfit <- ordmod$sample(
  data = ordmoddat,
  chains = 5,
  parallel_chains = 5,
  refresh = 0,
  iter_warmup = 500, iter_sampling = 2000)
```

```
## Running MCMC with 5 parallel chains...
##
## Chain 1 finished in 6.2 seconds.
## Chain 2 finished in 6.3 seconds.
## Chain 3 finished in 6.3 seconds.
## Chain 5 finished in 6.3 seconds.
## Chain 4 finished in 6.4 seconds.
##
## All 5 chains finished successfully.
## Mean chain execution time: 6.3 seconds.
## Total execution time: 6.5 seconds.
```

```
ordmodaggdat <- list(
  N = 3,
  K = 8,
  P = 2,
  y = as.matrix(Dwide[, -1]),
  X = model.matrix( ~ x)[, -1],
  prior_counts = rep(1/8, 8),
  prior_sd = rep(1, 2)
)
ordmodaggfit <- ordmodagg$sample(
  data = ordmodaggdat,
  chains = 5,
  parallel_chains = 5,
  refresh = 0,
  iter_warmup = 500, iter_sampling = 2000)
```

```
## Running MCMC with 5 parallel chains...
##
## Chain 1 finished in 0.2 seconds.
## Chain 2 finished in 0.2 seconds.
## Chain 3 finished in 0.2 seconds.
## Chain 4 finished in 0.2 seconds.
## Chain 5 finished in 0.2 seconds.
##
## All 5 chains finished successfully.
## Mean chain execution time: 0.2 seconds.
## Total execution time: 0.2 seconds.
```

```
# Check consistency of result
coef(mle_fit)
```

```
##         x2         x3
##  0.6840541 -0.5691340
```

```
as_draws_rvars(ordmodfit$draws("beta"))
```

```
## # A draws_rvars: 2000 iterations, 5 chains, and 1 variables
## $beta: rvar<2000,5>[2] mean ± sd:
## [1]  0.68 ± 0.094  -0.57 ± 0.094
```

```
as_draws_rvars(ordmodaggfit$draws("beta"))
```

```
## # A draws_rvars: 2000 iterations, 5 chains, and 1 variables
## $beta: rvar<2000,5>[2] mean ± sd:
## [1]  0.68 ± 0.095  -0.57 ± 0.094
```

```
as_draws_rvars(ordmodfit$draws("c"))
```

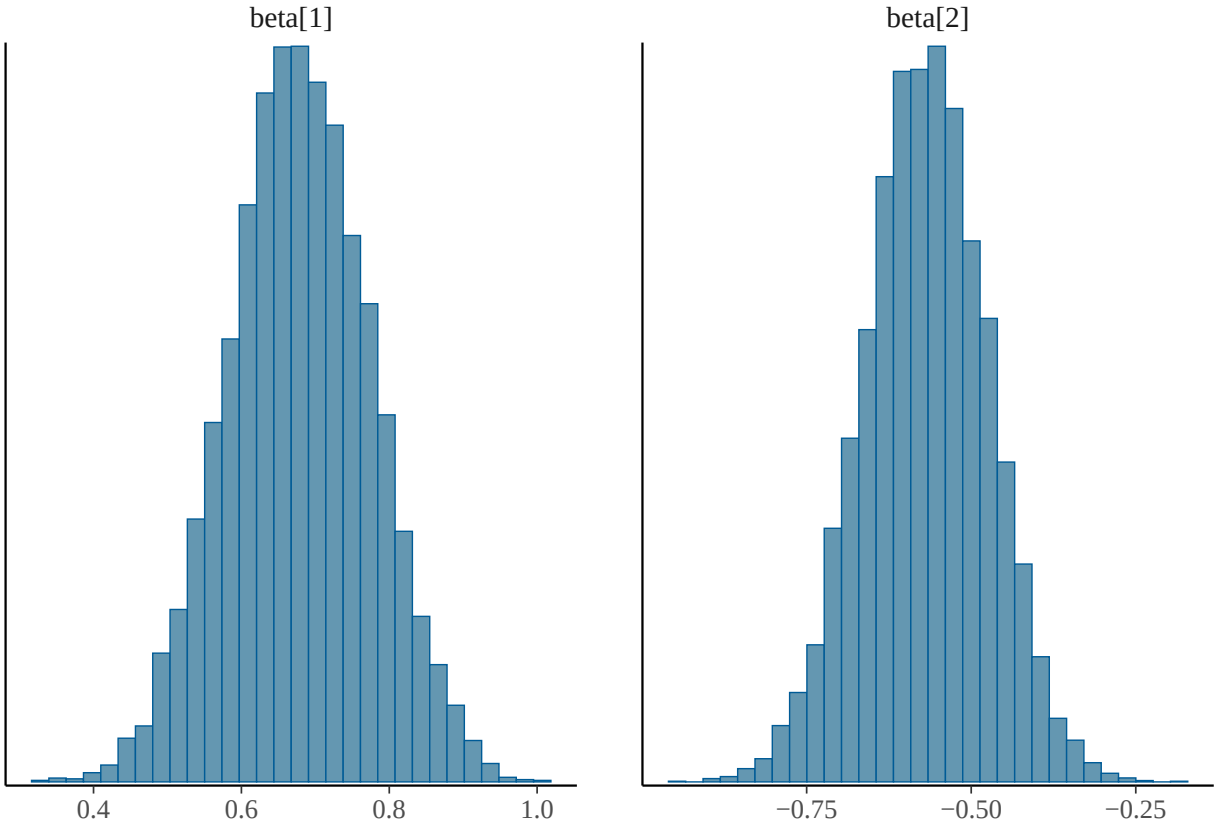
```
## # A draws_rvars: 2000 iterations, 5 chains, and 1 variables
## $c: rvar<2000,5>[7] mean ± sd:
## [1] -1.847 ± 0.083  -0.984 ± 0.073  -0.434 ± 0.070   0.058 ± 0.069
## [5]  0.598 ± 0.070   1.178 ± 0.074   1.980 ± 0.086
```

```
as_draws_rvars(ordmodaggfit$draws("alpha"))
```

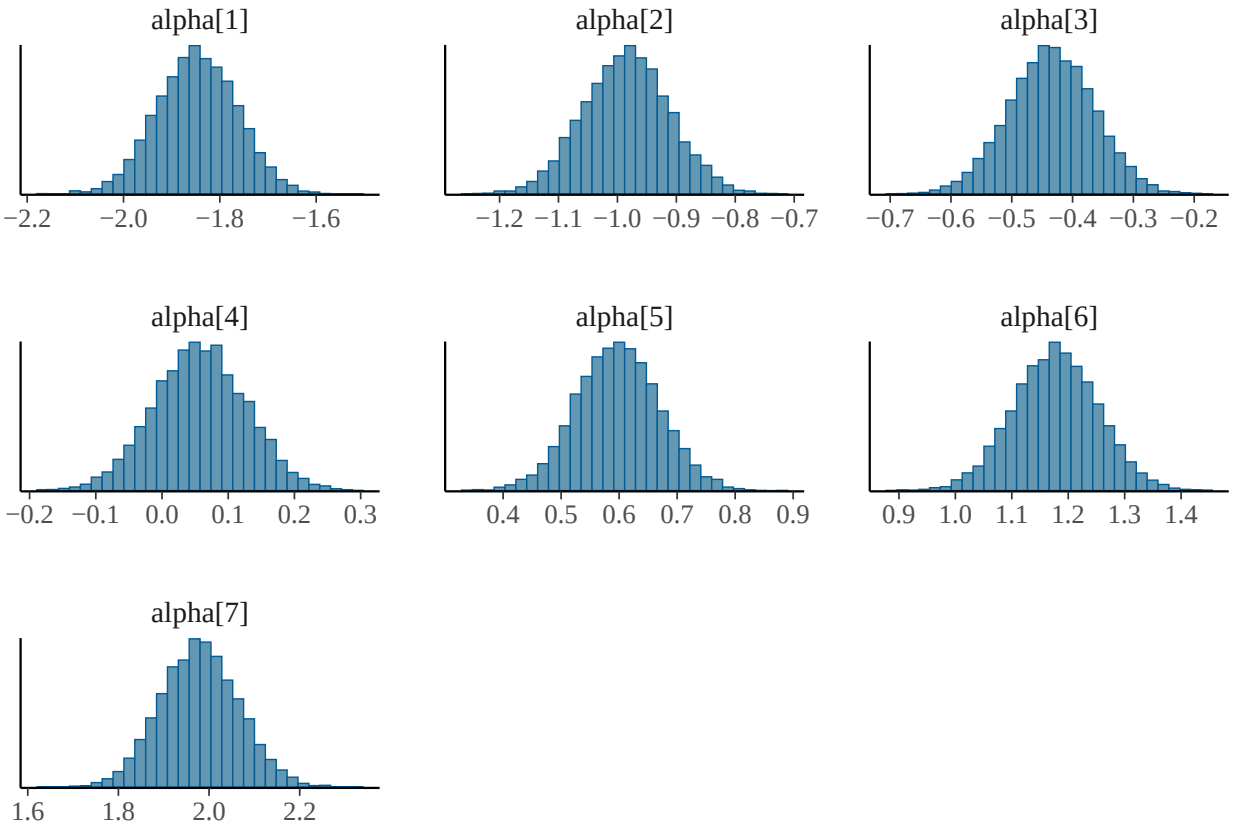
```
## # A draws_rvars: 2000 iterations, 5 chains, and 1 variables
## $alpha: rvar<2000,5>[7] mean ± sd:
## [1] -1.847 ± 0.083  -0.985 ± 0.073  -0.435 ± 0.070   0.057 ± 0.070
## [5]  0.597 ± 0.071   1.177 ± 0.075   1.979 ± 0.086
```

3.1.1 Posteriors

```
mcmc_hist(as_draws_matrix(ordmodaggfit$draws("beta")))
```




```
mcmc_hist(as_draws_matrix(ordmodaggfit$draws("alpha")))
```



Example 2

```
# Simulate some outcome data
N <- 600
p <- rbind(
  rep(1/ 8, 8),
  pomodm(p = rep(1/ 8, 8), odds.ratio = 2), # Odds ratio for 2 for arm 2
  pomodm(p = rep(1/ 8, 8), odds.ratio = 0.5)) # Odds ratio of 0.5 for arm 3
x <- factor(1:3)
n <- rep(N/3, 3)
y <- matrix(0, 3, 8)

xx <- rep(x, times = N/3)
for(i in 1:N) {
  y[i] <- sample.int(8, 1, prob = p[xx[i], ])
}
D <- data.table(x = x, y = y)
Dagg <- D[, .N, keyby = .(x, y)]              # Per-participant uses long format
Dwide <- dcast(Dagg, x ~ y, value.var = 'N') # Aggregated model uses wide format

# Sanity check
mle_fit <- MASS::polr(ordered(y) ~ x, data = D)

X <- model.matrix( ~ x, data = D)[, -1]
ordmoddat <- list(
  y = D$y,
  N = nrow(D),
  K = 8,
  P = ncol(X),
```

```
  X = X,
  prior_counts = rep(1/8, 8),
  prior_sd = rep(1, 2)
)
ordmodfit <- ordmod$sample(
  data = ordmoddat,
  chains = 5,
  parallel_chains = 5,
  refresh = 0,
  iter_warmup = 500, iter_sampling = 2000)
```

```
## Running MCMC with 5 parallel chains...
##
## Chain 1 finished in 1.8 seconds.
## Chain 2 finished in 1.8 seconds.
## Chain 3 finished in 1.8 seconds.
## Chain 4 finished in 1.8 seconds.
## Chain 5 finished in 1.8 seconds.
##
## All 5 chains finished successfully.
## Mean chain execution time: 1.8 seconds.
## Total execution time: 1.9 seconds.
```

```
ordmodaggdat <- list(
  N = 3,
  K = 8,
  P = 2,
  y = as.matrix(Dwide[, -1]),
  X = model.matrix( ~ x)[, -1],
  prior_counts = rep(1/8, 8),
  prior_sd = rep(1, 2)
)
ordmodaggfit <- ordmodagg$sample(
  data = ordmodaggdat,
  chains = 5,
  parallel_chains = 5,
  refresh = 0,
  iter_warmup = 500, iter_sampling = 2000)
```

```
## Running MCMC with 5 parallel chains...
##
## Chain 1 finished in 0.2 seconds.
## Chain 2 finished in 0.2 seconds.
## Chain 3 finished in 0.2 seconds.
## Chain 4 finished in 0.2 seconds.
## Chain 5 finished in 0.2 seconds.
##
## All 5 chains finished successfully.
## Mean chain execution time: 0.2 seconds.
## Total execution time: 0.3 seconds.
```

```
# Check consistency of result
coef(mle_fit)
```

```
##         x2         x3
```

```
##  0.7497065 -0.6731018
```

```
as_draws_rvars(ordmodfit$draws("beta"))
```

```
## # A draws_rvars: 2000 iterations, 5 chains, and 1 variables
## $beta: rvar<2000,5>[2] mean ± sd:
## [1]  0.74 ± 0.17  -0.66 ± 0.18
```

```
as_draws_rvars(ordmodaggfit$draws("beta"))
```

```
## # A draws_rvars: 2000 iterations, 5 chains, and 1 variables
## $beta: rvar<2000,5>[2] mean ± sd:
## [1]  0.74 ± 0.17  -0.66 ± 0.17
```

```
as_draws_rvars(ordmodfit$draws("c"))
```

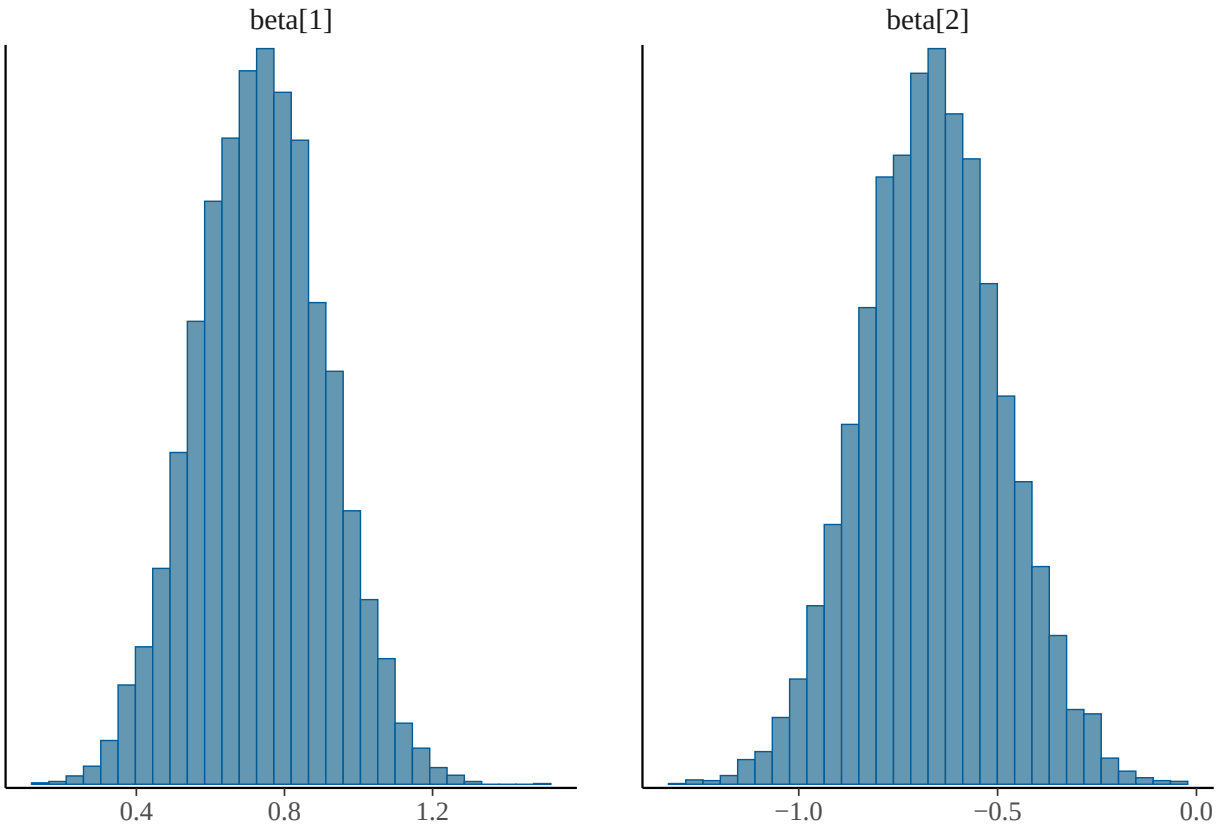
```
## # A draws_rvars: 2000 iterations, 5 chains, and 1 variables
## $c: rvar<2000,5>[7] mean ± sd:
## [1] -1.955 ± 0.16  -1.132 ± 0.14  -0.449 ± 0.13   0.075 ± 0.13   0.603 ± 0.13
## [6]  1.243 ± 0.14   1.930 ± 0.16
```

```
as_draws_rvars(ordmodaggfit$draws("alpha"))
```

```
## # A draws_rvars: 2000 iterations, 5 chains, and 1 variables
## $alpha: rvar<2000,5>[7] mean ± sd:
## [1] -1.955 ± 0.16  -1.132 ± 0.14  -0.448 ± 0.13   0.075 ± 0.13   0.604 ± 0.13
## [6]  1.244 ± 0.14   1.931 ± 0.16
```

3.1.2 Posteriors

```
mcmc_hist(as_draws_matrix(ordmodaggfit$draws("beta")))
```




```
mcmc_hist(as_draws_matrix(ordmodaggfit$draws("alpha")))
```