Pilot Analysis: Building a Linear Log Odds Model of Probability of Superiority

In this document, we build a two part mixture in brms which captures the proportion of responses that follow a linear log odds (LLO) pattern vs a constant response pattern.

The LLO model follows from related work (https://www.frontiersin.org/articles/10.3389/fnins.2012.00001/full) suggesting that the human perception of probability is encoded on a log odds scale. On this scale, the slope of a linear model represents the shape and severity of the function describing bias in probability perception. The greater the deviation of from a slope of 1 (i.e., ideal performance), the more biased the judgments of probability. Slopes less than one correspond to the kind of bias predicted by excessive attention to the mean. On the same log odds scale, the intercept is a crossover-point which should be proportional to the number of categories of possible outcomes among which probability is divided. In our case, the intercept should be about 0.5 since workers are judging the probability of a new machine vs an old machine producing more (defective) widgets.

Although we start by building up the LLO model on its own, there are a large number of responses near 50% (the middle of the probability scale) which cannot be accounted for by the LLO model. Thus, we add a mixture component for a random constant response per subject and model the proportion of responses which match this pattern depending on the visualization condition.

Load and Prepare Data

We load worker responses from our pilot and do some preprocessing.

```
# read in data
full_df <- read_csv("pilot-anonymous.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   workerId = col_character(),
##   batch = col_integer(),
##   condition = col_character(),
##   start_gain_frame = col_character(),
##   numeracy = col_integer(),
##   gender = col_character(),
##   age = col_character(),
##   education = col_character(),
##   chart_use = col_character(),
##   intervene = col_integer(),
##   outcome = col_character(),
##   pSup = col_integer(),
##   trial = col_character(),
##   trialIdx = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

```
# preprocessing
responses_df <- full_df %>%
  rename( # rename to convert away from camel case
    worker_id = workerId,
    account_value = accountValue,
    ground_truth = groundTruth,
    p_award_with = pAwardWith,
    p_award_without = pAwardWithout,
    p_superiority = pSup,
    start_time = startTime,
    resp_time = respTime,
    trial_dur = trialDur,
    trial_idx = trialIdx
  ) %>%
  filter(trial_idx != "practice", trial_idx != "mock") %>% # remove practice and mock tr
ials from responses dataframe, leave in full version
  mutate(
    # flip probability of superiority responses below 50% for the loss frame
    p_superiority = if_else(ground_truth < 0.5,
                            100 - p_superiority,
                            as.numeric(p_superiority)), # hack to avoid type error
    # mutate to jitter probability of superiority away from boundaries
    p_superiority = ifelse(p_superiority == 0, 0.25, p_superiority),            # avoid
 responses equal to zero
    p_superiority = ifelse(p_superiority == 100, 99.75, p_superiority)         # avoid r
esponses equal to one-hundred
  )

head(responses_df)
```

```
## # A tibble: 6 x 29
##   worker_id batch condition baseline award_value exchange start_gain_frame
##   <chr>     <int> <chr>        <dbl>       <dbl>    <dbl> <chr>
## 1 bf797261      4 means_on…      0.5        2.25      0.5 True
## 2 bf797261      4 means_on…      0.5        2.25      0.5 True
## 3 bf797261      4 means_on…      0.5        2.25      0.5 True
## 4 bf797261      4 means_on…      0.5        2.25      0.5 True
## 5 bf797261      4 means_on…      0.5        2.25      0.5 True
## 6 bf797261      4 means_on…      0.5        2.25      0.5 True
## # … with 22 more variables: total_bonus <dbl>, duration <dbl>,
## #   numeracy <int>, gender <chr>, age <chr>, education <chr>,
## #   chart_use <chr>, account_value <dbl>, ground_truth <dbl>,
## #   intervene <int>, outcome <chr>, pAwardCurrent <dbl>, pAwardNew <dbl>,
## #   p_award_with <dbl>, p_award_without <dbl>, p_superiority <dbl>,
## #   payoff <dbl>, resp_time <dbl>, start_time <dbl>, trial <chr>,
## #   trial_dur <dbl>, trial_idx <chr>
```

We need the data in a format where it is prepared for modeling. This means converting both probability of superiority judgments and the ground truth to a logit scale. It also means that we want problem framing as a factor.

```
# create data frame for model
model_df_llo <- responses_df %>%
  mutate( # apply logit function to p_sup judgments and ground truth
    lo_p_sup=qlogis(p_superiority / 100),
    lo_ground_truth=qlogis(ground_truth),
    frame = as.factor(if_else(ground_truth > 0.5, "gain", "loss"))
  )
```

Distribution of Probability of Superiority Judgments

We start as simply as possible by just modeling the distribution of probability of superiority judgements on the log odds scale.

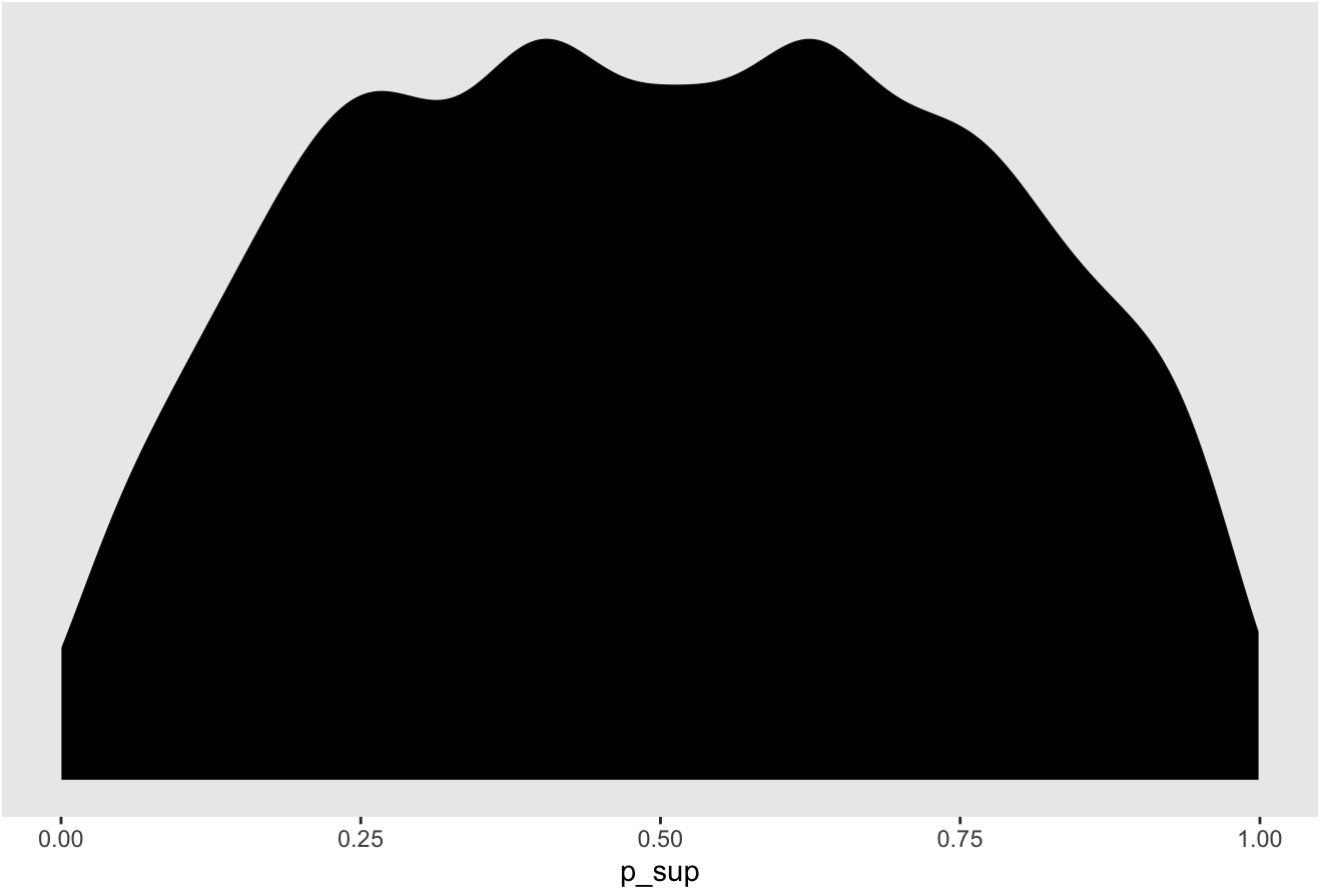
Let's check that our priors seem reasonable.

```
# prior predictive check
n <- 1e4
tibble(sample_mu    = rnorm(n, mean = 0, sd = 1),
       sample_sigma = rnorm(n, mean = 0, sd = 1)) %>%
  mutate(lo_p_sup = rnorm(n, mean = sample_mu, sd = sample_sigma),
         p_sup = plogis(lo_p_sup)) %>%
  ggplot(aes(x = p_sup)) +
  geom_density(fill = "black", size = 0) +
  scale_y_continuous(NULL, breaks = NULL) +
  labs(subtitle = expression(paste("Prior predictive distribution for ", italic(h[i])))) +
  theme(panel.grid = element_blank())
```

```
## Warning in rnorm(n, mean = sample_mu, sd = sample_sigma): NAs produced
```

```
## Warning: Removed 4938 rows containing non-finite values (stat_density).
```

Prior predictive distribution for h_i



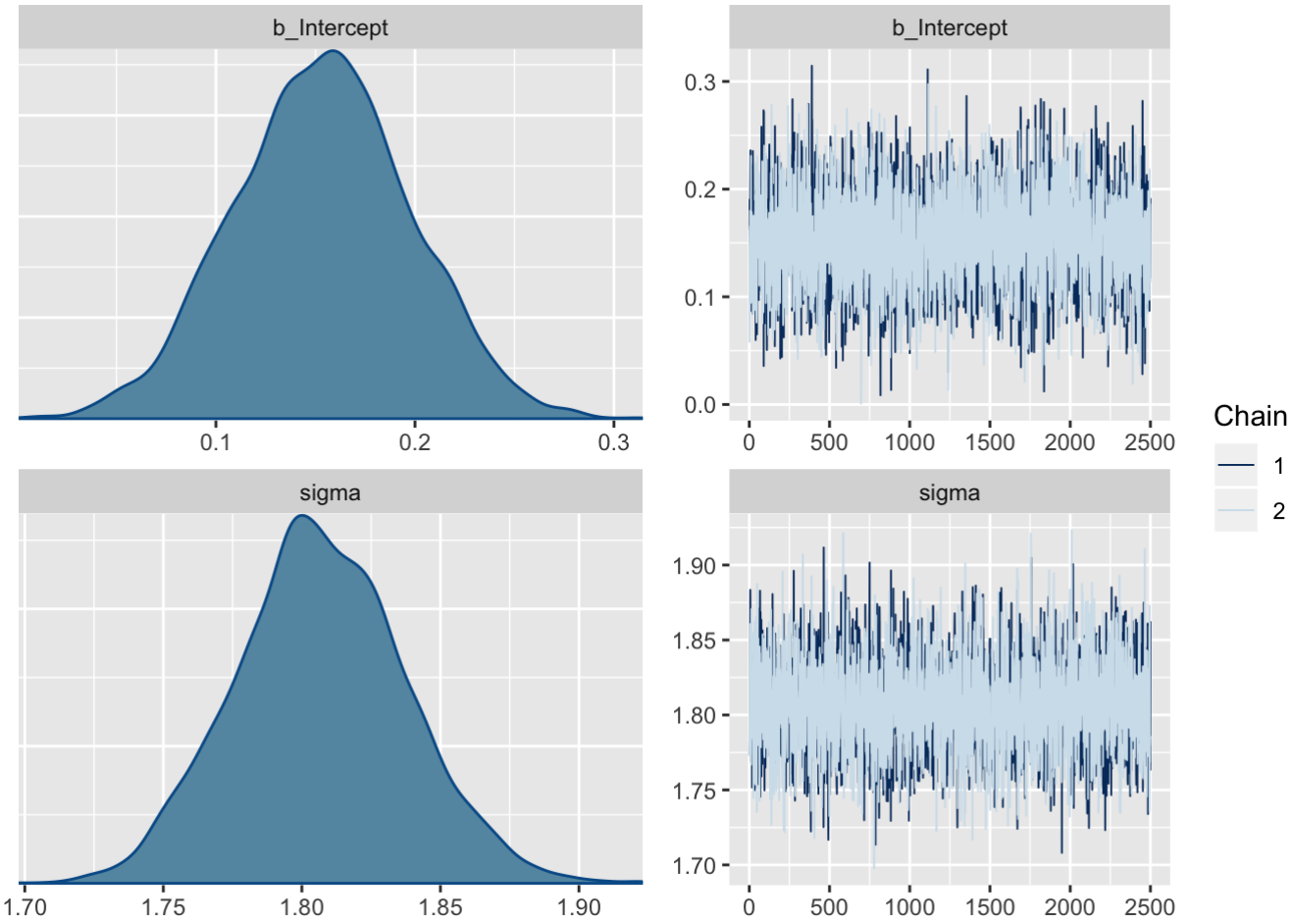
Fit an intercept model.

```
# starting as simple as possible: learn the distribution of lo_p_sup
m.lo_p_sup <- brm(data = model_df_llo, family = gaussian,
              lo_p_sup ~ 1,
              prior = c(prior(normal(0, 1), class = Intercept),
                        prior(normal(0, 1), class = sigma)),
              iter = 3000, warmup = 500, chains = 2, cores = 2,
              file = "model-fits/lo_mdl")
```

Check diagnostics:

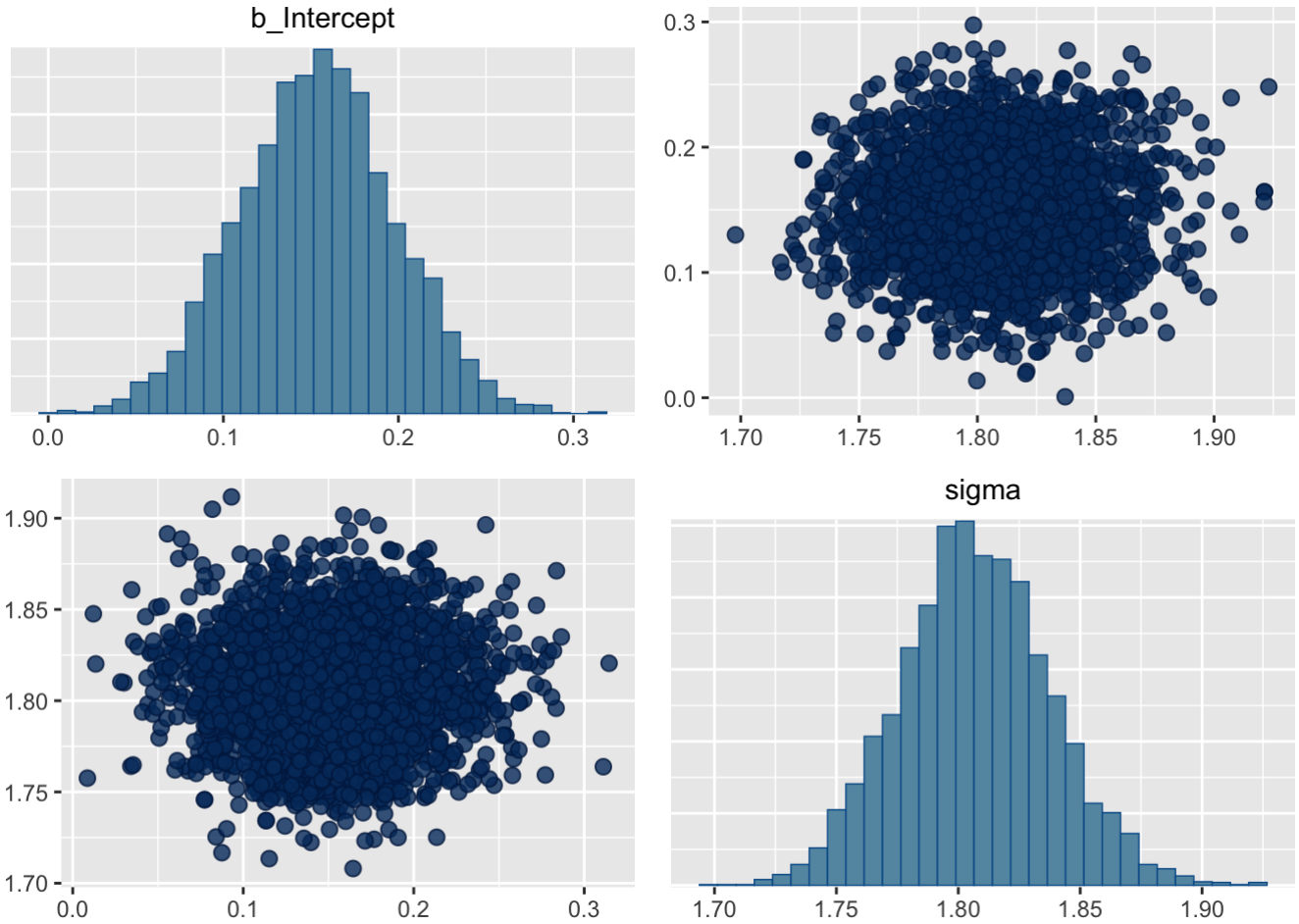
- Trace plots

```
# trace plots
plot(m.lo_p_sup)
```



- Pairs plot

```
# pairs plot
pairs(m.lo_p_sup)
```



- Summary

```
# model summary
print(m.lo_p_sup)
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: lo_p_sup ~ 1
##    Data: model_df_llo (Number of observations: 1568)
## Samples: 2 chains, each with iter = 3000; warmup = 500; thin = 1;
##          total post-warmup samples = 5000
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## Intercept     0.15      0.04     0.07     0.24       4612 1.00
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## sigma     1.81      0.03     1.75     1.87       3379 1.00
##
## Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
## is a crude measure of effective sample size, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

Let's get a sense of the distribution of posterior draws.

```
# get posterior draws
post <- posterior_samples(m.lo_p_sup)

# get summary stats on distribution of each parameter
post %>%
  select(-lp__) %>% # drop log probability of posterior draws
  gather(parameter) %>%
  group_by(parameter) %>%
  mean_qi(value)
```

```
## Warning: Unquoting language objects with `!!!` is deprecated as of rlang 0.4.0.
## Please use `!!` instead.
##
##   # Bad:
##   dplyr::select(data, !!!enquo(x))
##
##   # Good:
##   dplyr::select(data, !!enquo(x))    # Unquote single quosure
##   dplyr::select(data, !!!enquos(x))  # Splice list of quosures
##
## This warning is displayed once per session.
```

```
## # A tibble: 2 x 7
##   parameter   value .lower .upper .width .point .interval
##   <chr>       <dbl>  <dbl>  <dbl>  <dbl> <chr>  <chr>
## 1 b_Intercept 0.154 0.0669  0.239   0.95 mean   qi
## 2 sigma       1.81  1.75    1.87    0.95 mean   qi
```

Linear Log Odds Model of Probability of Superiority

Now well add in a slope parameter.

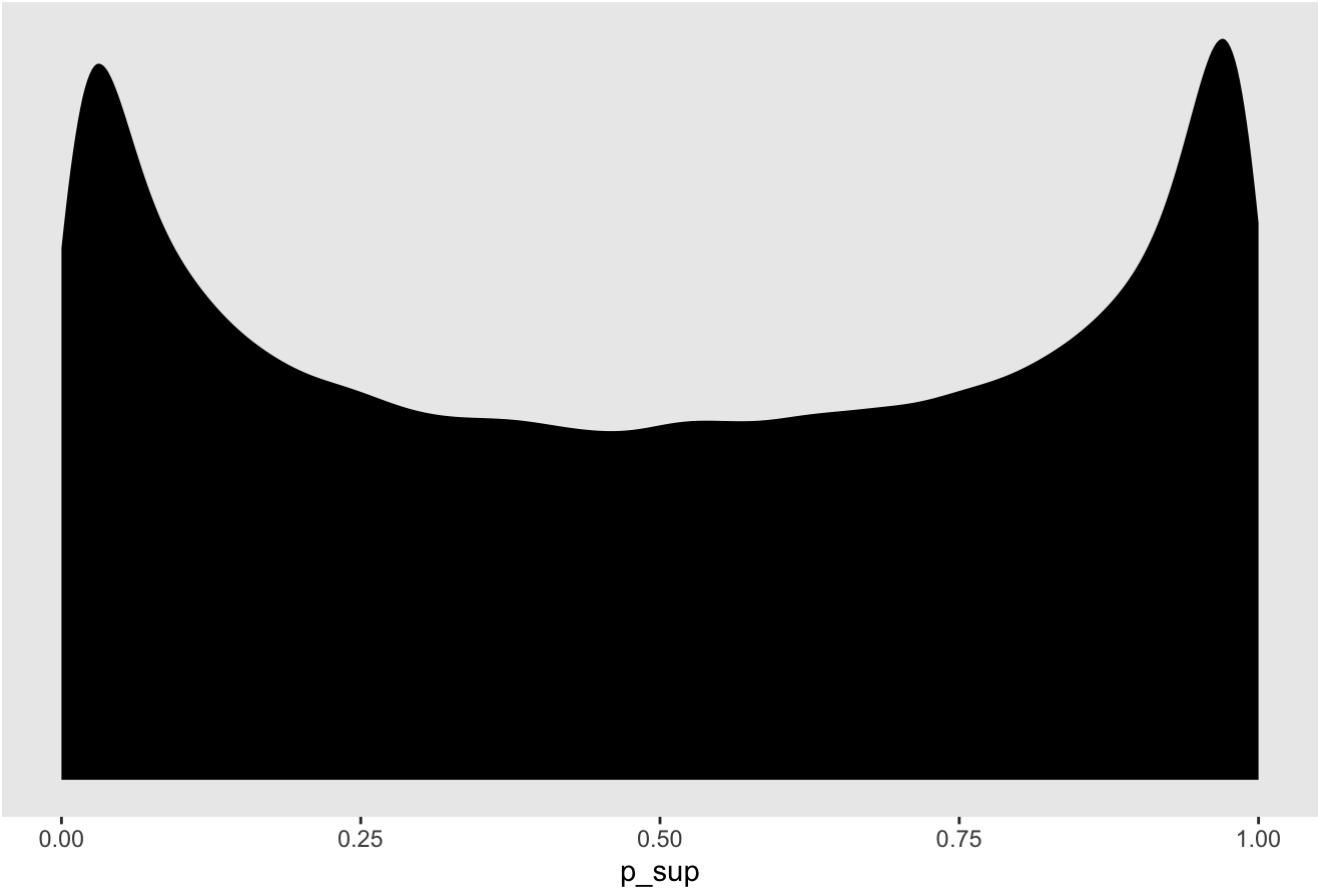
Let's check that our priors seem reasonable.

```
# prior predictive check
n <- 1e4
tibble(intercept    = rnorm(n, mean = 0, sd = 1),
       slope        = rnorm(n, mean = 0, sd = 1),
       sample_sigma = rnorm(n, mean = 0, sd = 1),
       lo_ground_truth = list(qlogis(unique(responses_df$ground_truth)))) %>%
  unnest() %>%
  mutate(lo_p_sup = rnorm(n * length(unique(responses_df$ground_truth)), mean = intercep
t + slope * lo_ground_truth, sd = sample_sigma),
         p_sup = plogis(lo_p_sup)) %>%
  ggplot(aes(x = p_sup)) +
  geom_density(fill = "black", size = 0) +
  scale_y_continuous(NULL, breaks = NULL) +
  labs(subtitle = expression(paste("Prior predictive distribution for ", italic(h[i])))
+
  theme(panel.grid = element_blank())
```

```
## Warning in rnorm(n * length(unique(responses_df$ground_truth)), mean =
## intercept + : NAs produced
```

```
## Warning: Removed 141176 rows containing non-finite values (stat_density).
```

Prior predictive distribution for h_i



What we get using generic weakly informative priors seems reasonable given that we are sampling more at extreme levels of ground truth than values in the middle of the scale.

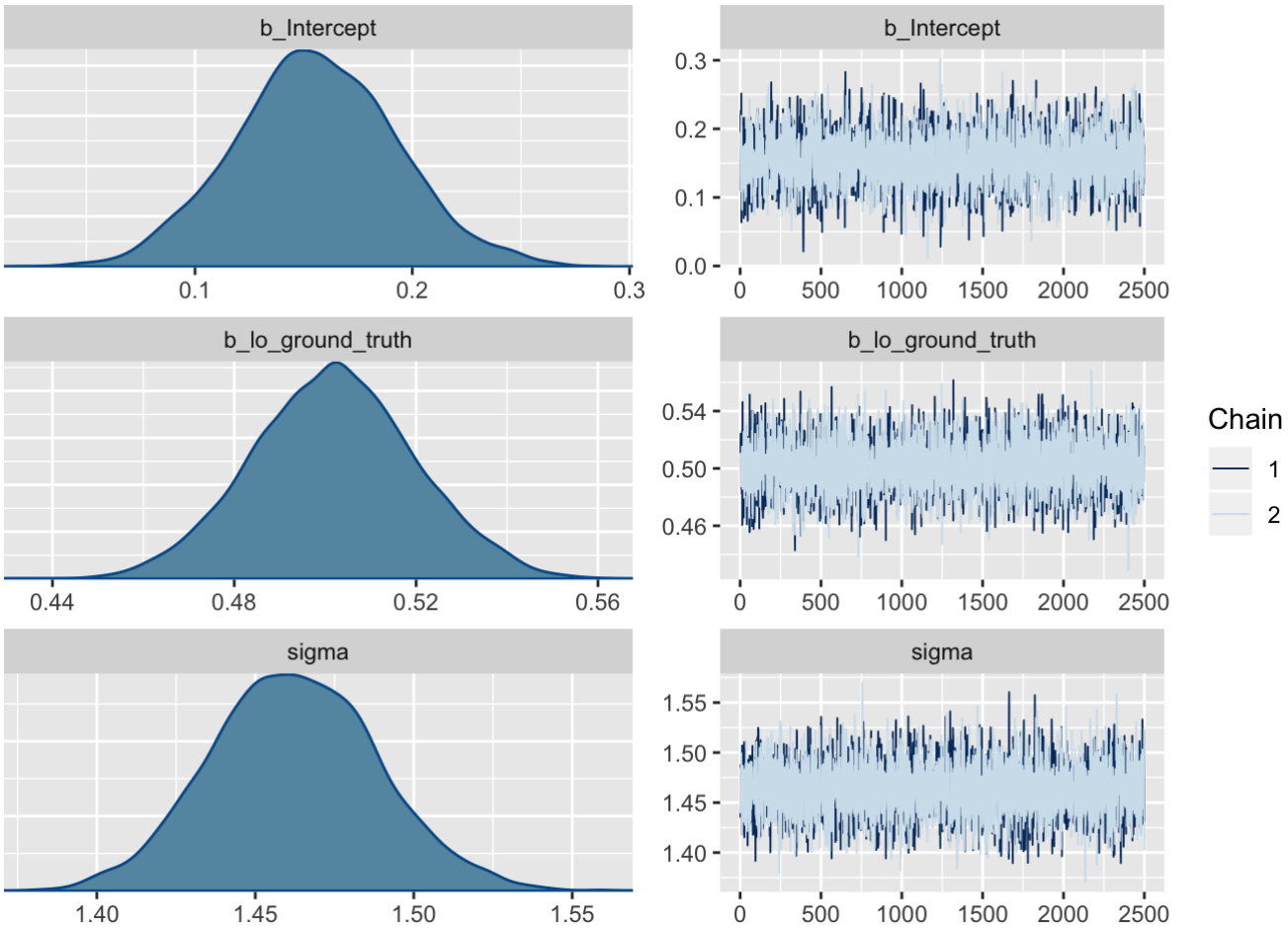
Now, let's fit the linear log odds (LLO) model.

```
# linear log odds model: lo_p_sup ~ 1 + lo_ground_truth
m.llo_p_sup <- brm(data = model_df_llo, family = gaussian,
              lo_p_sup ~ 1 + lo_ground_truth,
              prior = c(prior(normal(0, 1), class = Intercept),
                        prior(normal(0, 1), class = b),
                        prior(normal(0, 1), class = sigma)),
              iter = 3000, warmup = 500, chains = 2, cores = 2,
              file = "model-fits/llo_mdl")
```

Check diagnostics:

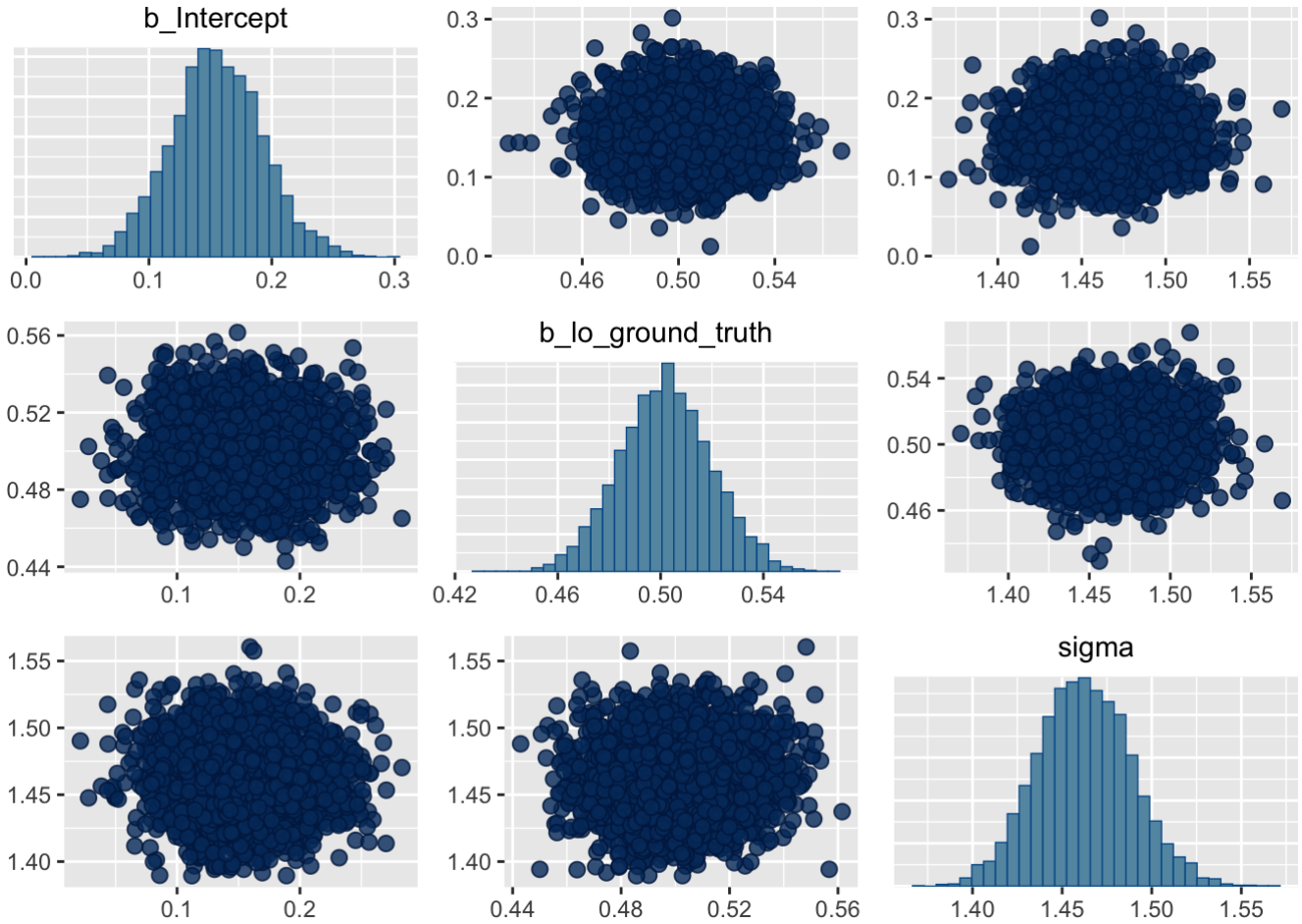
- Trace plots

```
# trace plots
plot(m.llo_p_sup)
```



- Pairs plot. It looks like we might have an issue with multicolinearity because b_Intercept and b_lo_ground_truth are so highly correlated.

```
# pairs plot
pairs(m.llo_p_sup)
```



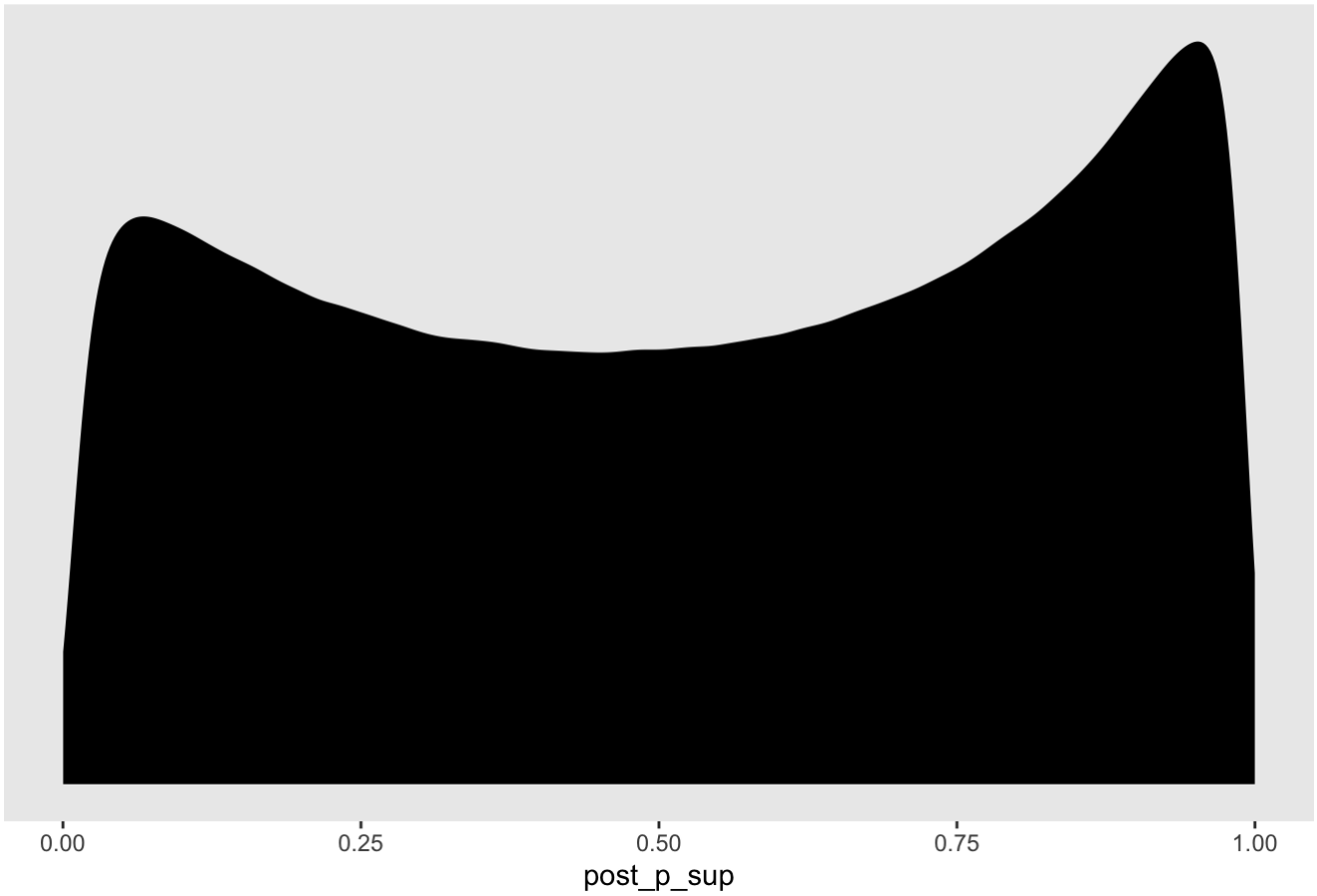
- Summary

```
# model summary
print(m.llo_p_sup)
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: lo_p_sup ~ 1 + lo_ground_truth
##    Data: model_df_llo (Number of observations: 1568)
## Samples: 2 chains, each with iter = 3000; warmup = 500; thin = 1;
##          total post-warmup samples = 5000
##
## Population-Level Effects:
##                Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## Intercept          0.16      0.04     0.08     0.23       4057 1.00
## lo_ground_truth    0.50      0.02     0.47     0.54       5261 1.00
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## sigma     1.46      0.03     1.41     1.52       5243 1.00
##
## Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
## is a crude measure of effective sample size, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

Let's check out a posterior predictive distribution for probability of superiority.

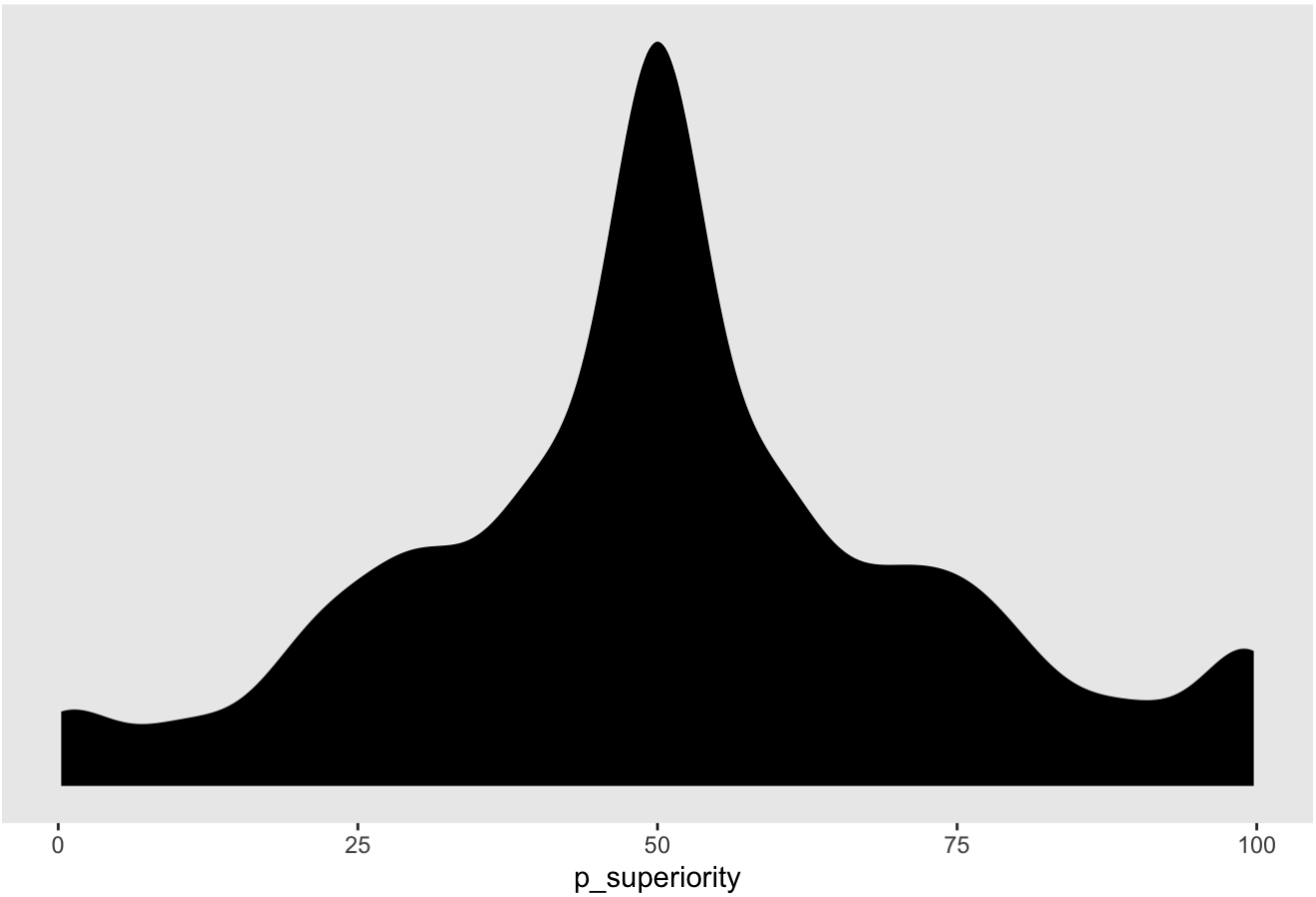
```
# posterior predictive check
model_df_llo %>%
  select(lo_ground_truth) %>%
  add_predicted_draws(m.llo_p_sup, prediction = "lo_p_sup", seed = 1234) %>%
  mutate(post_p_sup = plogis(lo_p_sup)) %>%
  ggplot(aes(x = post_p_sup)) +
  geom_density(fill = "black", size = 0) +
  scale_y_continuous(NULL, breaks = NULL) +
  labs(subtitle = "Posterior predictive distribution for probability of superiority",
       post_p_sup = NULL) +
  theme(panel.grid = element_blank())
```



The posterior predictive distributions seems too wide and a little skewed. How do the posterior predictions compare to the observed data?

```
# data density
model_df_llo %>%
  ggplot(aes(x = p_superiority)) +
  geom_density(fill = "black", size = 0) +
  scale_y_continuous(NULL, breaks = NULL) +
  labs(subtitle = "Data distribution for probability of superiority") +
  theme(panel.grid = element_blank())
```

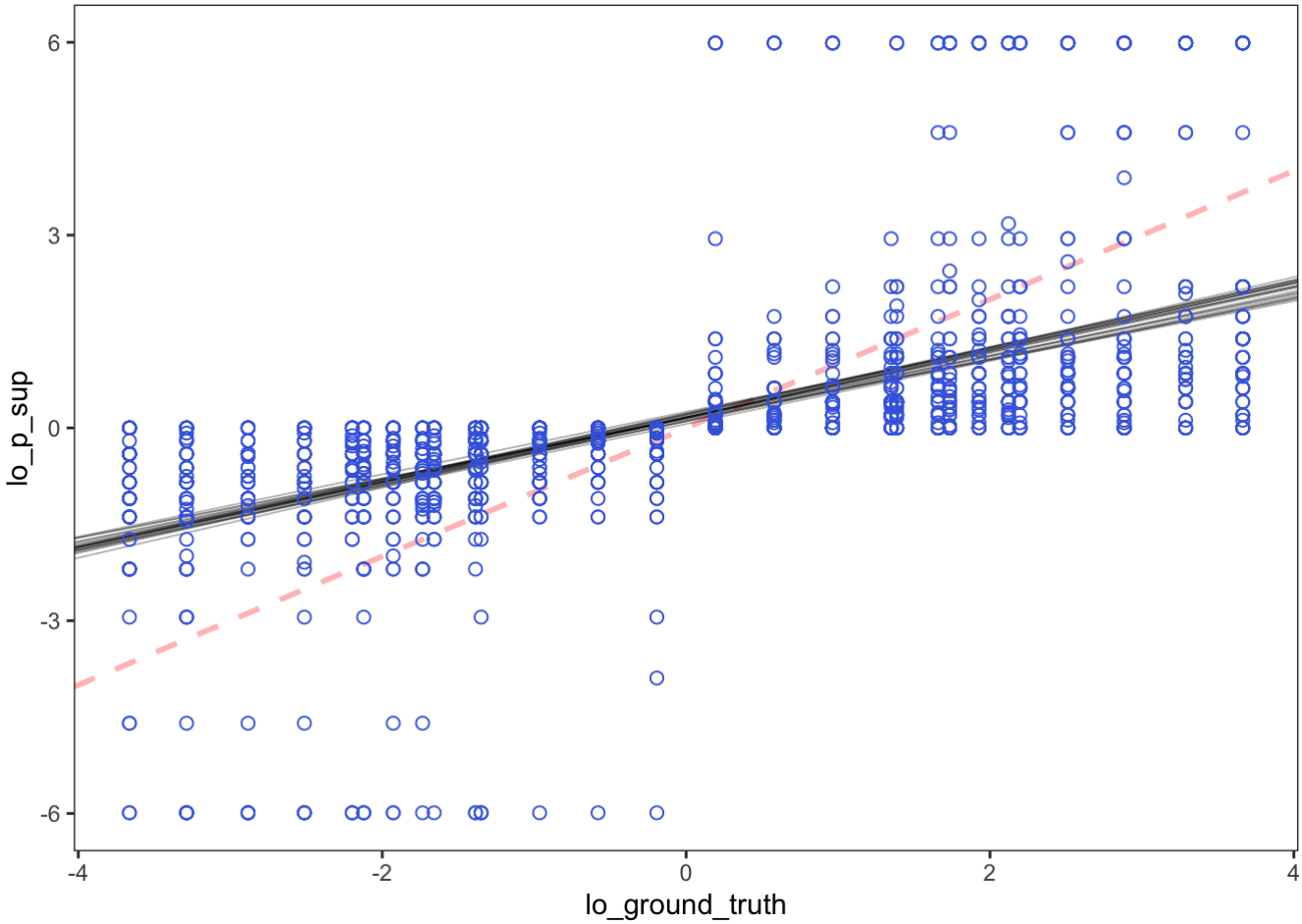
Data distribution for probability of superiority



The peak in the center of the probability scale seems to suggest that the data generating process is a 50% inflated mixture. Also, note that we see the same skew in the observed data as in the posterior predictions, suggesting that the model has learned this from the data.

Let's take a look at some of the estimated linear models.

```
# get posterior samples
post  <- posterior_samples(m.llo_p_sup)

# plot estimated linear models against observed data
model_df_llo %>%
ggplot(aes(x = lo_ground_truth, y = lo_p_sup)) +
  geom_abline(intercept = 0, slope = 1, size = 1, alpha = .3, color = "red", linetype =
"dashed") + # ground truth
  geom_abline(intercept = post[1:20, 1],
              slope     = post[1:20, 2],
              size = 1/3, alpha = .3) +
  geom_point(shape = 1, size = 2, color = "royalblue") +
  coord_cartesian(xlim = quantile(model_df_llo$lo_ground_truth, c(0, 1)),
                  ylim = quantile(model_df_llo$lo_p_sup, c(0, 1))) +
  theme_bw() +
  theme(panel.grid = element_blank())
```



Add Different Linear Models per Visualization Condition

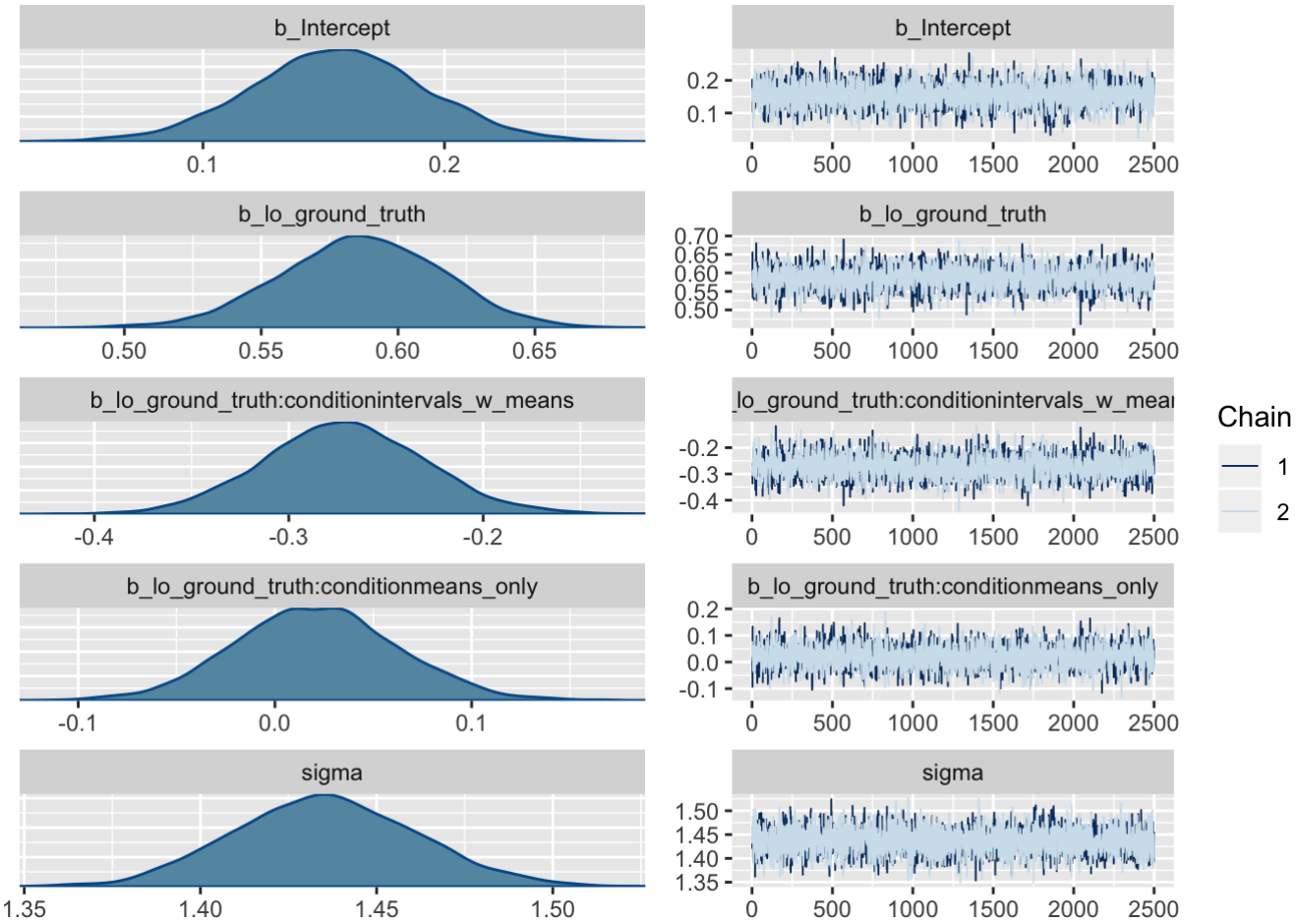
In the LLO framework, what we really want to know about is the impact of visualization condition on the slopes of linear models in log odds space. Do some visualizations lead to more extreme patterns of bias than others? To test this, we'll add an interaction between visualization condition and the ground truth.

```
# update the llo model of p_sup responses to include an interaction
m.vis.llo_p_sup <- update(m.llo_p_sup,
                          formula = lo_p_sup ~ 1 + lo_ground_truth + lo_ground_truth:cond
ition,
                          newdata = model_df_llo,
                          file = "model-fits/llo_mdl_vis")
```

Check diagnostics:

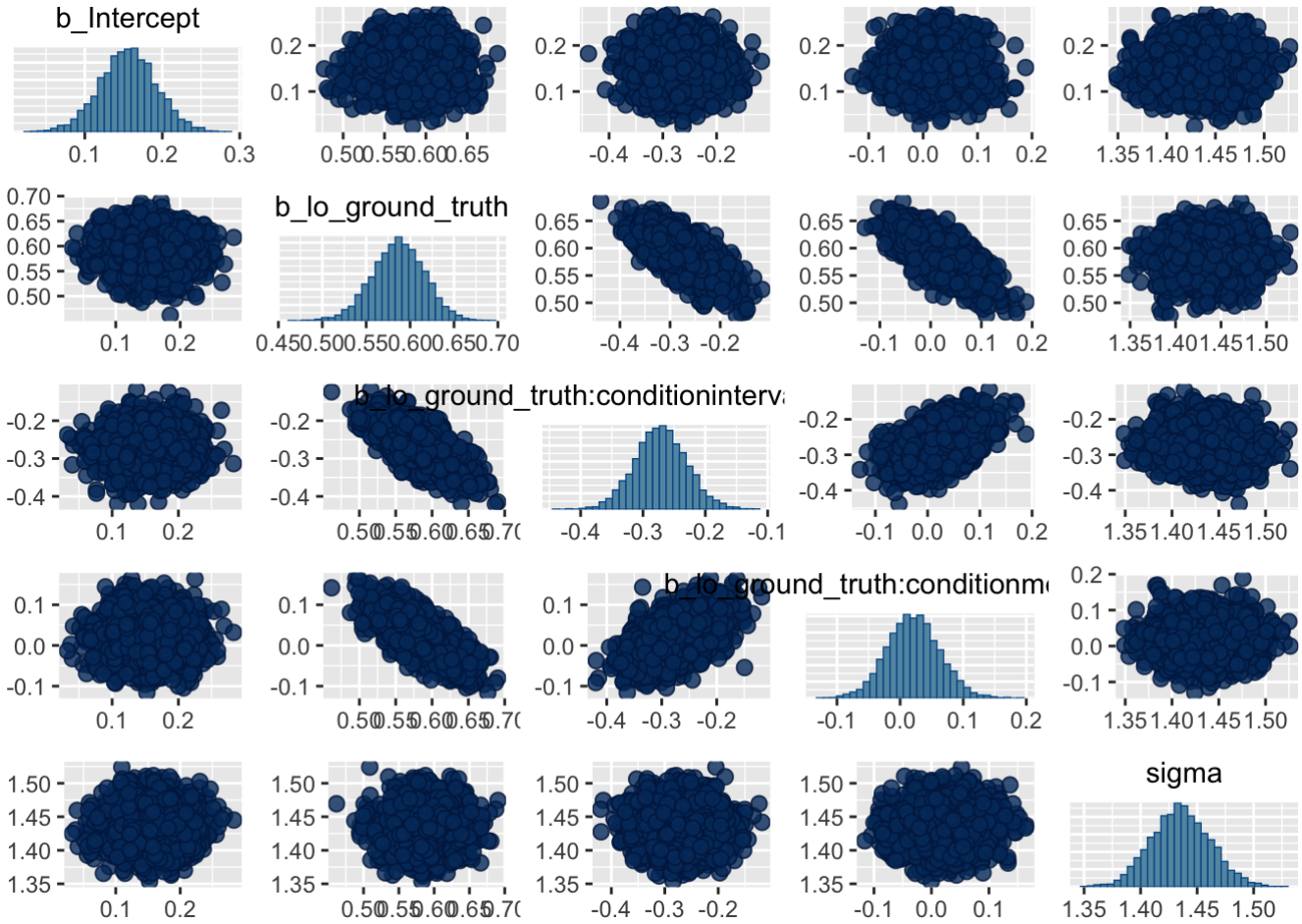
- Trace plots

```
# trace plots
plot(m.vis.llo_p_sup)
```



- Pairs plot

```
# pairs plot
pairs(m.vis.llo_p_sup)
```



Theses slopes are looking pretty correlated.

- Summary

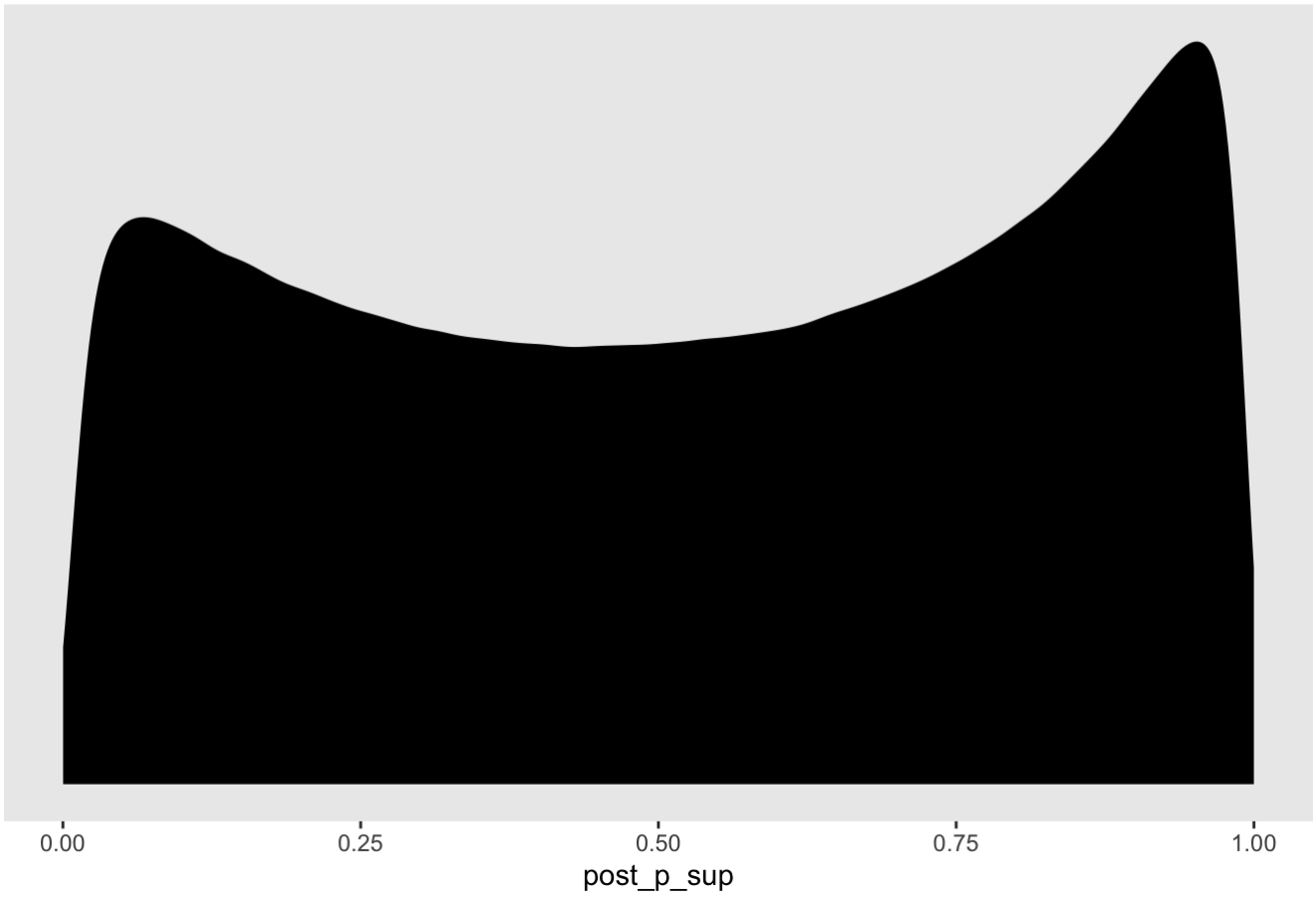
```
# model summary
print(m.vis.llo_p_sup)
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: lo_p_sup ~ lo_ground_truth + lo_ground_truth:condition
##    Data: model_df_llo (Number of observations: 1568)
## Samples: 2 chains, each with iter = 3000; warmup = 500; thin = 1;
##          total post-warmup samples = 5000
##
## Population-Level Effects:
##                                          Estimate Est.Error l-95% CI
## Intercept                                    0.16      0.04     0.08
## lo_ground_truth                              0.59      0.03     0.53
## lo_ground_truth:conditionintervals_w_means   -0.27      0.04    -0.35
## lo_ground_truth:conditionmeans_only           0.02      0.04    -0.06
##                                          u-95% CI Eff.Sample Rhat
## Intercept                                    0.23       4732 1.00
## lo_ground_truth                              0.64       2625 1.00
## lo_ground_truth:conditionintervals_w_means   -0.19       2957 1.00
## lo_ground_truth:conditionmeans_only           0.10       3115 1.00
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## sigma     1.44      0.03     1.39     1.49       4245 1.00
##
## Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
## is a crude measure of effective sample size, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

Let's check out a posterior predictive distribution for probability of superiority. This should look more like a mixture with different slopes, but it still looks really wide.

```
# posterior predictive check
model_df_llo %>%
  select(lo_ground_truth, condition) %>%
  add_predicted_draws(m.vis.llo_p_sup, prediction = "lo_p_sup", seed = 1234) %>%
  mutate(post_p_sup = plogis(lo_p_sup)) %>%
  ggplot(aes(x = post_p_sup)) +
  geom_density(fill = "black", size = 0) +
  scale_y_continuous(NULL, breaks = NULL) +
  labs(subtitle = "Posterior predictive distribution for probability of superiority") +
  theme(panel.grid = element_blank())
```

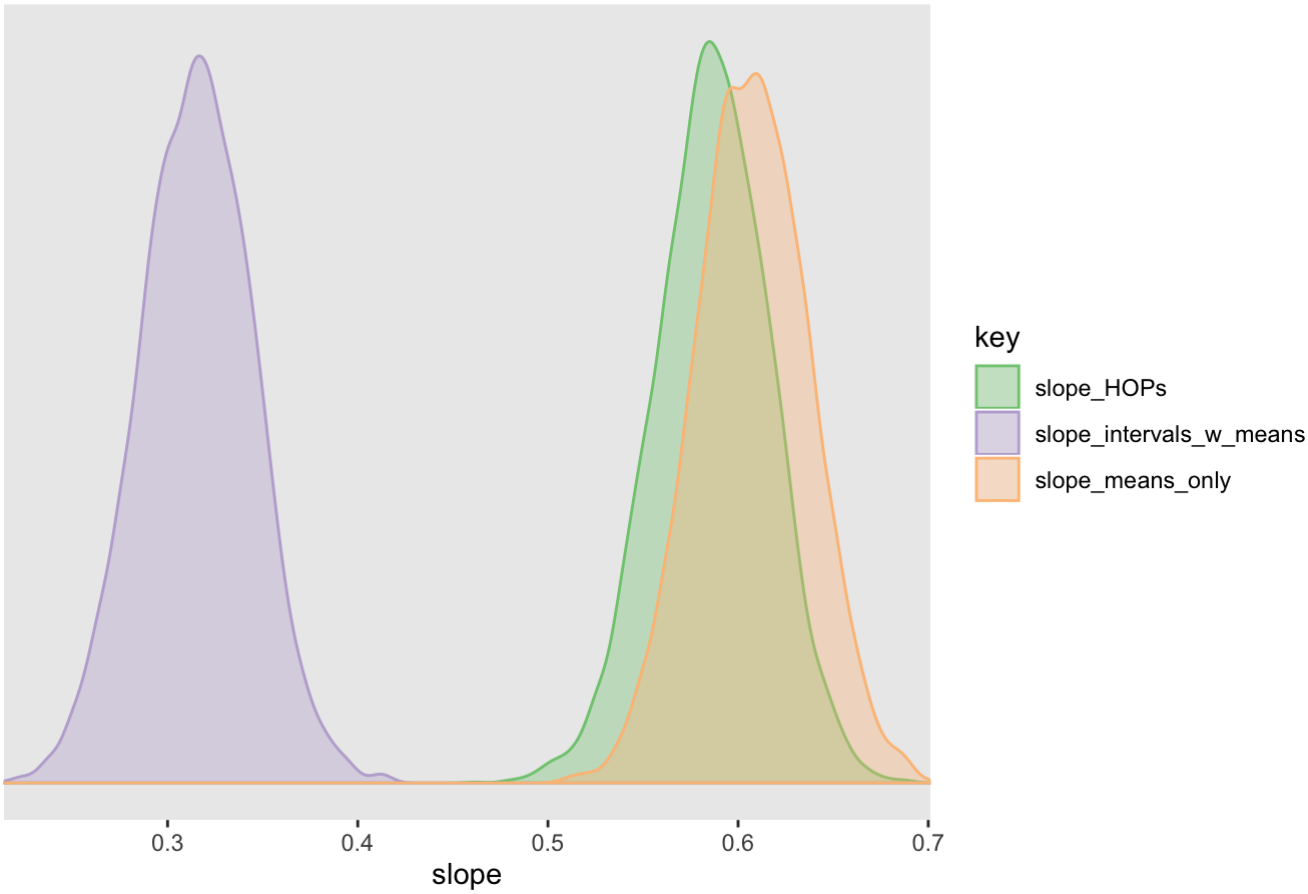
Posterior predictive distribution for probability of superiority



What does the posterior for the slope in each visualization condition look like?

```
# use posterior samples to define distributions for the slope in each visualization cond
ition
posterior_samples(m.vis.llo_p_sup) %>%
  transmute(slope_HOPs = b_lo_ground_truth,
            slope_intervals_w_means = b_lo_ground_truth + `b_lo_ground_truth:conditionin
tervals_w_means`,
            slope_means_only = b_lo_ground_truth + `b_lo_ground_truth:conditionmeans_onl
y`) %>%
  gather(key, value) %>%
  ggplot(aes(x = value, group = key, color = key, fill = key)) +
  geom_density(alpha = 0.35) +
  scale_fill_brewer(type = "qual", palette = 1) +
  scale_color_brewer(type = "qual", palette = 1) +
  scale_x_continuous(expression(slope), expand = c(0, 0)) +
  scale_y_continuous(NULL, breaks = NULL) +
  labs(subtitle = "Posterior for slopes by visualization condition") +
  theme(panel.grid = element_blank())
```

Posterior for slopes by visualization condition

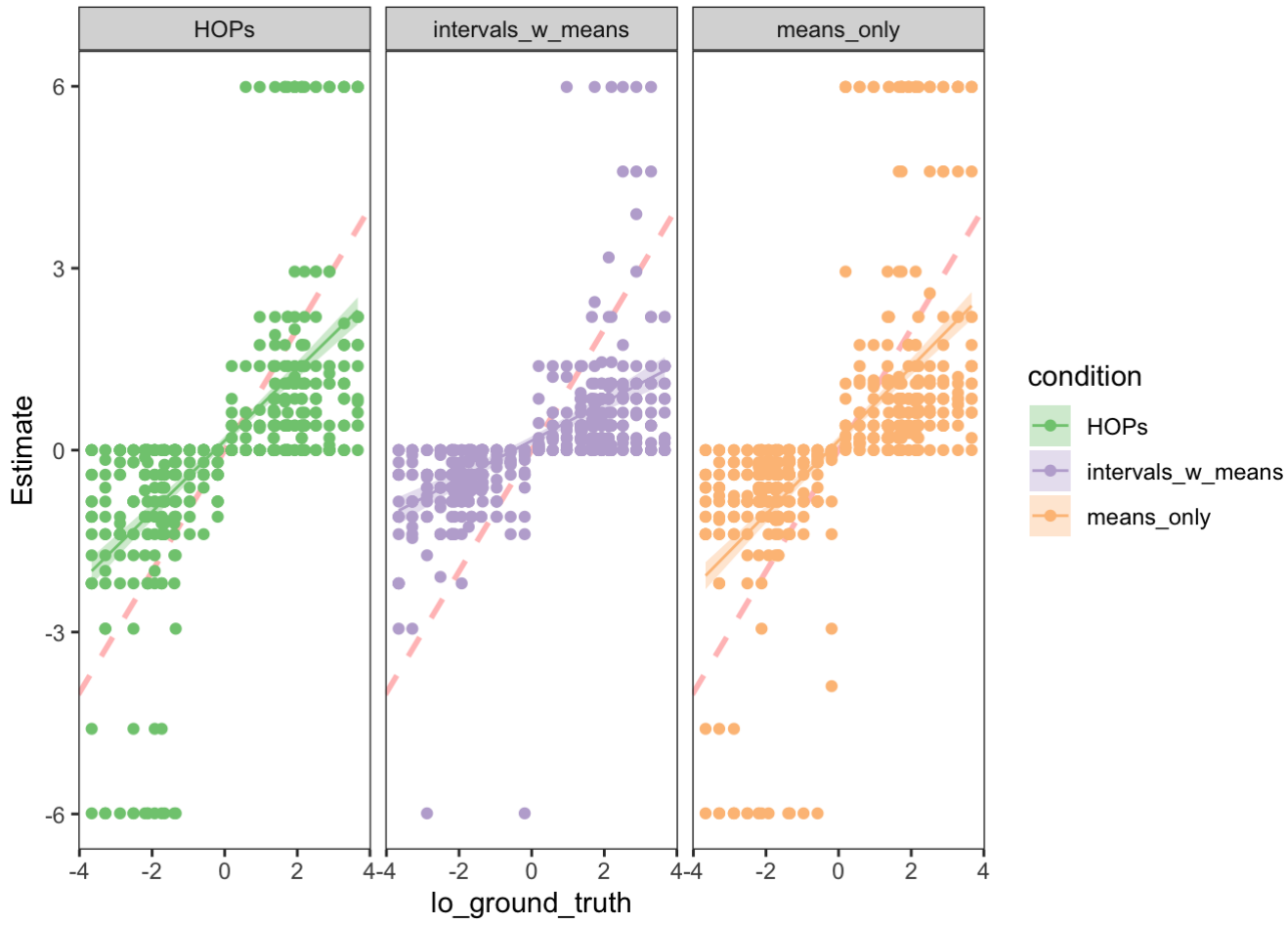


Recall that a slope of 1 reflects zero bias. This suggests that users are biased toward responses of 50% in all conditions.

Let's take a look at some of the estimated linear models per visualization condition.

```
# set up new dataframe for prediction
nd <- tibble(lo_ground_truth = seq(from = quantile(model_df_llo$lo_ground_truth, 0), to
 = quantile(model_df_llo$lo_ground_truth, 1), length.out = 30) %>%
         rep(., times = 3),
         condition = rep(unique(model_df_llo$condition), each = 30))

# pass our new predictive dataframe through our fitted model
fitd_m.vis.llo_p_sup <- fitted(m.vis.llo_p_sup, newdata = nd) %>%
  as_tibble() %>%
  bind_cols(nd)
```

```
# plot estimated linear models against observed data
model_df_llo %>%
  ggplot(aes(x = lo_ground_truth)) +
  geom_abline(intercept = 0, slope = 1, size = 1, alpha = .3, color = "red", linetype =
"dashed") + # ground truth
  geom_ribbon(data = fitd_m.vis.llo_p_sup,
              aes(ymin  = Q2.5,
                  ymax  = Q97.5,
                  fill  = condition,
                  group = condition),
              alpha = .35) +
  geom_line(data = fitd_m.vis.llo_p_sup,
              aes(y     = Estimate,
                  color = condition,
                  group = condition)) +
  geom_point(aes(y = lo_p_sup, color = condition)) +
    scale_fill_brewer(type = "qual", palette = 1) +
  scale_color_brewer(type = "qual", palette = 1) +
  coord_cartesian(xlim = quantile(model_df_llo$lo_ground_truth, c(0, 1)),
                  ylim = quantile(model_df_llo$lo_p_sup, c(0, 1))) +
  theme_bw() +
  theme(panel.grid = element_blank()) +
  facet_grid(. ~ condition)
```



This looks pretty promising, but let's see if we can account for some of the variability by separating it into error vs individual variability.

Add Hierarchy for Slopes and Intercepts

The models we've created thus far fail to account for much of the noise in the data. Here, we attempt to parse some heterogeniety in responses by modeling a random effect of worker on slopes and intercepts. This introduces a hierarchical component to our model in order to account for individual differences in the best fitting linear model for each worker's data.

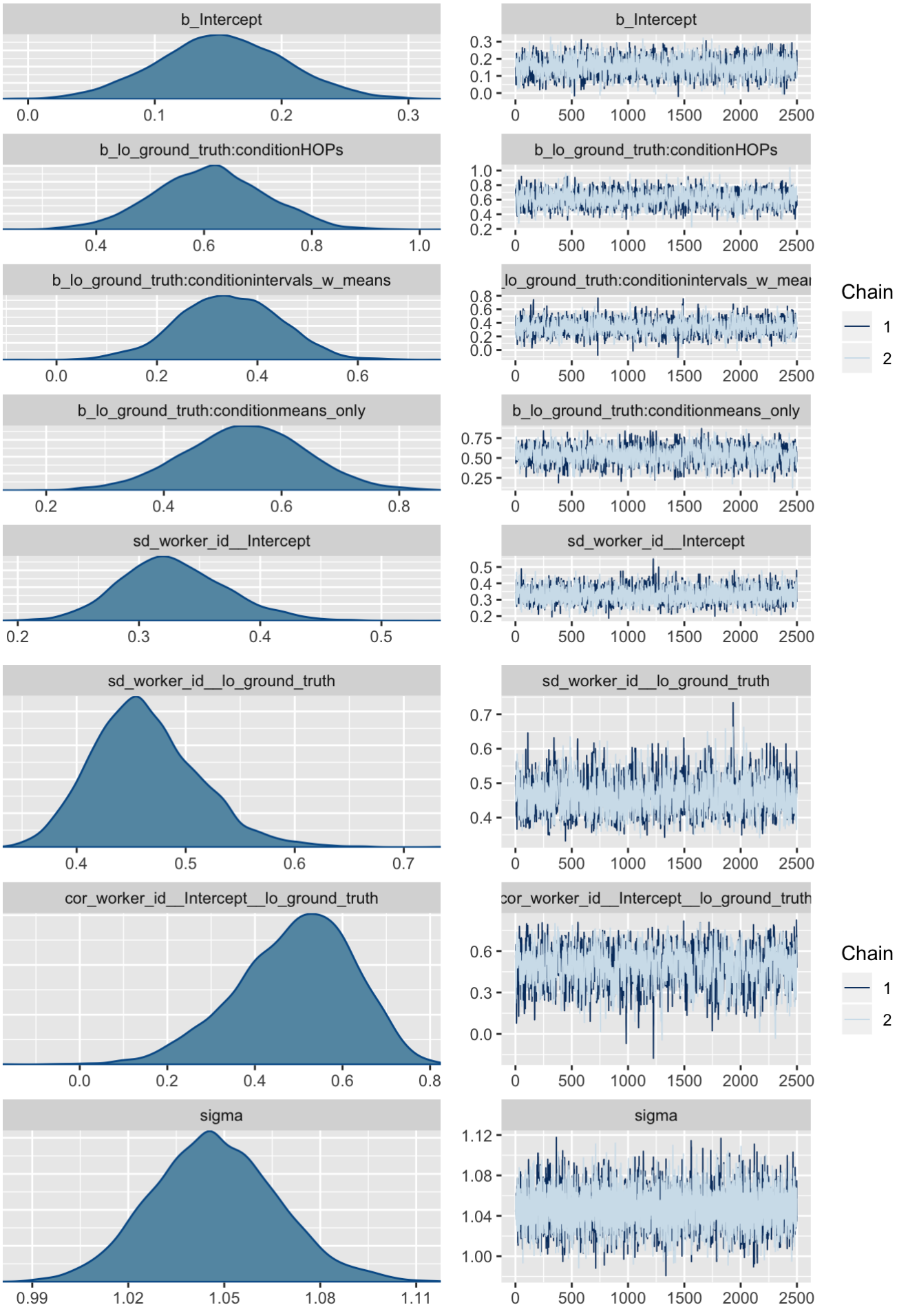
```
# update the llo model of p_sup responses to include an interaction
m.vis.wrkr.llo_p_sup <- update(m.llo_p_sup,
                         formula = lo_p_sup ~ (1 + lo_ground_truth|worker_id) + lo_groun
d_truth:condition,
                         newdata = model_df_llo,
                         file = "model-fits/llo_mdl_vis_wrkr")
```

```
## The desired updates require recompiling the model
```

Check diagnostics:

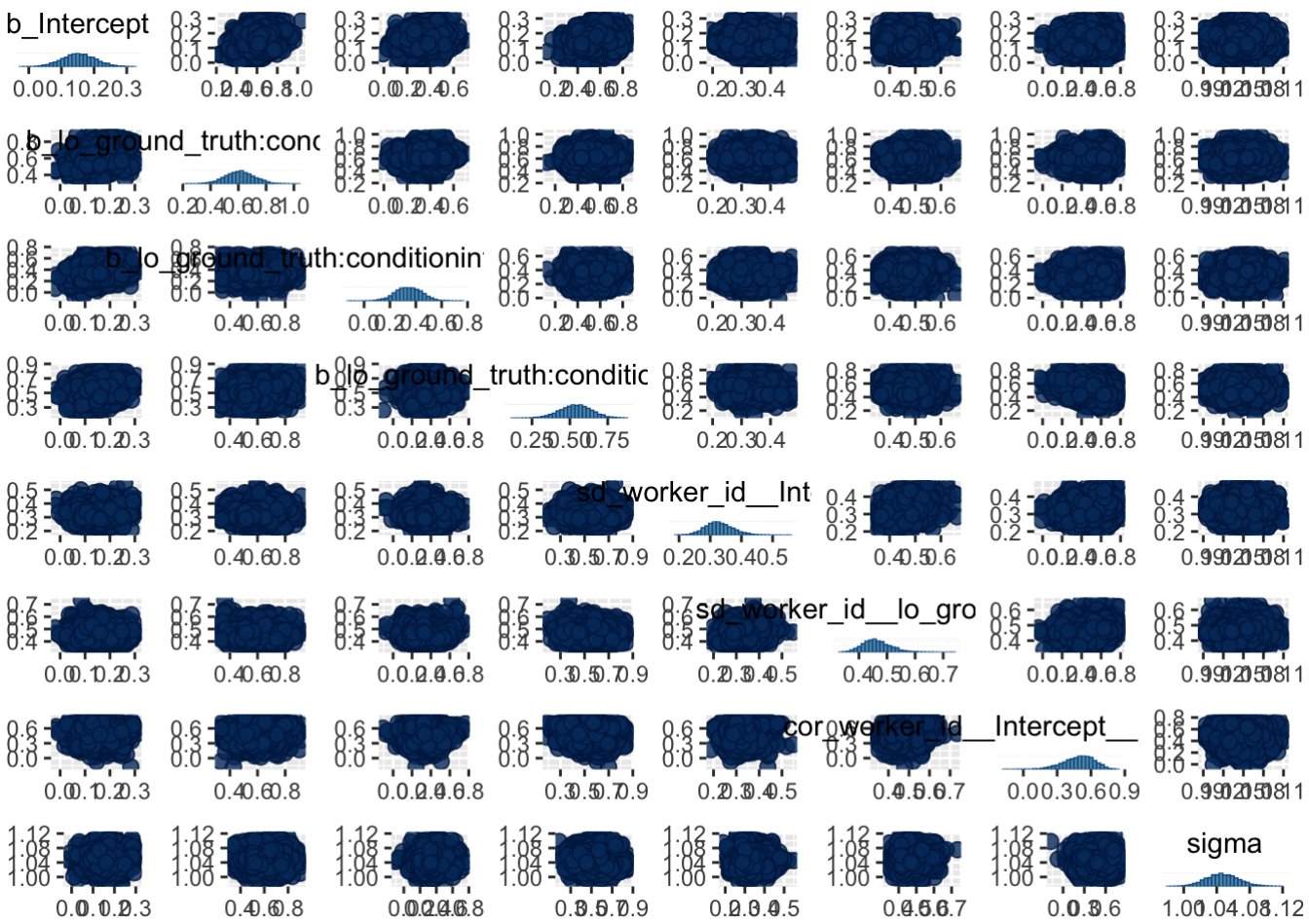
- Trace plots

```
# trace plots
plot(m.vis.wrkr.llo_p_sup)
```


b_Intercept
b_lo_ground_truth:conditionHOPs
b_lo_ground_truth:conditionintervals_w_means
b_lo_ground_truth:conditionmeans_only
sd_worker_id__Intercept
sd_worker_id__lo_ground_truth
cor_worker_id__Intercept__lo_ground_truth
sigma
Chain
1
2

- Pairs plot

```
# pairs plot
pairs(m.vis.wrkr.llo_p_sup)
```



- Summary

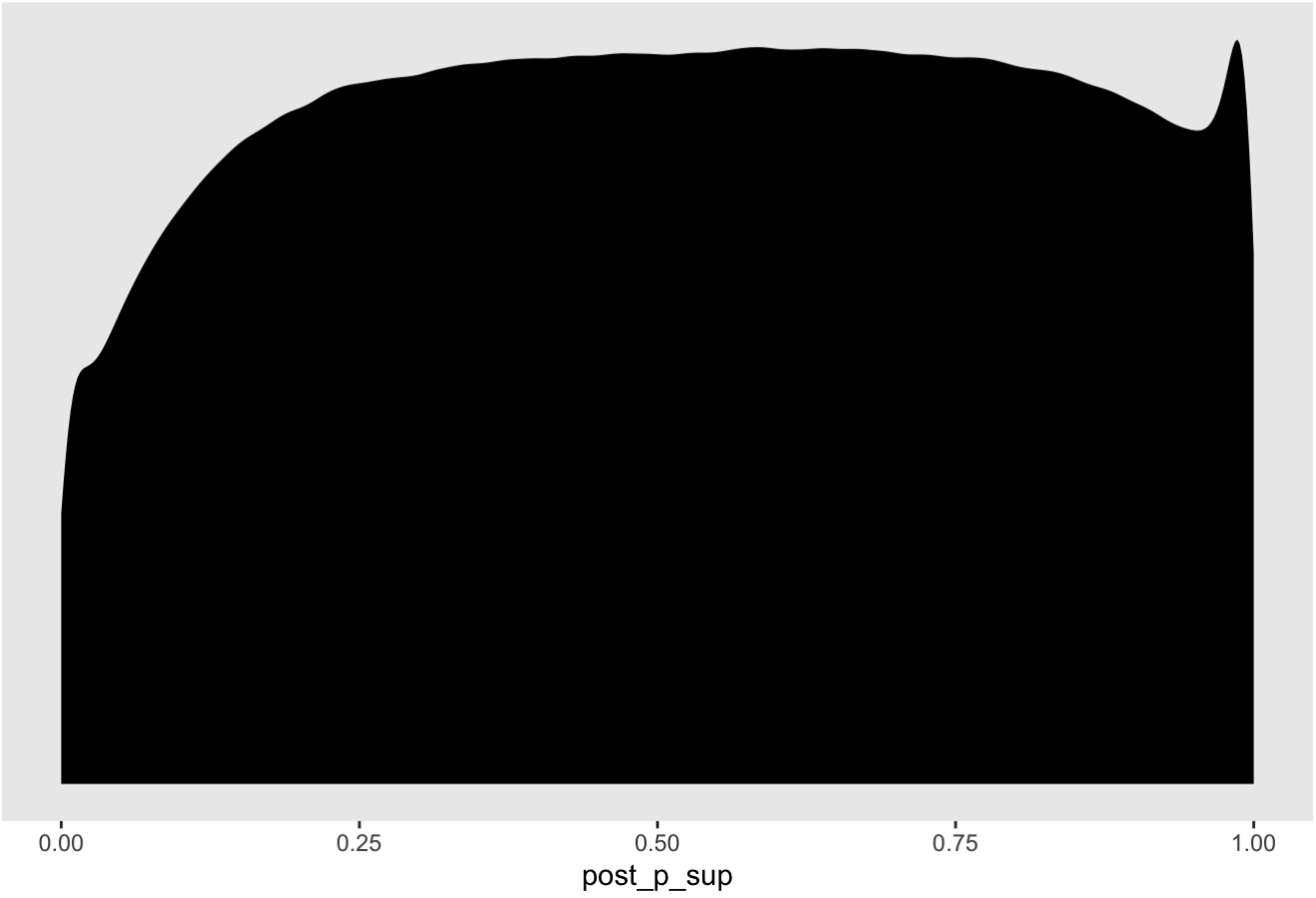
```
# model summary
print(m.vis.wrkr.llo_p_sup)
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: lo_p_sup ~ (1 + lo_ground_truth | worker_id) + lo_ground_truth:condition
##    Data: model_df_llo (Number of observations: 1568)
## Samples: 2 chains, each with iter = 3000; warmup = 500; thin = 1;
##          total post-warmup samples = 5000
##
## Group-Level Effects:
## ~worker_id (Number of levels: 56)
##                               Estimate Est.Error l-95% CI u-95% CI
## sd(Intercept)                     0.33      0.04     0.25     0.42
## sd(lo_ground_truth)               0.46      0.05     0.38     0.57
## cor(Intercept,lo_ground_truth)    0.49      0.13     0.21     0.72
##                               Eff.Sample Rhat
## sd(Intercept)                       2350 1.00
## sd(lo_ground_truth)                 1686 1.00
## cor(Intercept,lo_ground_truth)      1214 1.00
##
## Population-Level Effects:
##                                         Estimate Est.Error l-95% CI
## Intercept                                   0.15      0.05     0.05
## lo_ground_truth:conditionHOPs               0.61      0.10     0.41
## lo_ground_truth:conditionintervals_w_means  0.34      0.10     0.14
## lo_ground_truth:conditionmeans_only         0.54      0.10     0.33
##                                         u-95% CI Eff.Sample Rhat
## Intercept                                   0.25       2455 1.00
## lo_ground_truth:conditionHOPs               0.81       1755 1.00
## lo_ground_truth:conditionintervals_w_means  0.54       1850 1.00
## lo_ground_truth:conditionmeans_only         0.74       1809 1.00
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## sigma     1.05      0.02     1.01     1.09       7443 1.00
##
## Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
## is a crude measure of effective sample size, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

Let's check out a posterior predictive distribution for probability of superiority. This distribution is looking less spread toward the tails than in previous iterations of the model, which suggests that individual variation in slopes accounts for some (but not all) of the 50% responses.

```
# posterior predictive check
model_df_llo %>%
  select(lo_ground_truth, condition, worker_id) %>%
  add_predicted_draws(m.vis.wrkr.llo_p_sup, prediction = "lo_p_sup", seed = 1234) %>%
  mutate(post_p_sup = plogis(lo_p_sup)) %>%
  ggplot(aes(x = post_p_sup)) +
  geom_density(fill = "black", size = 0) +
  scale_y_continuous(NULL, breaks = NULL) +
  labs(subtitle = "Posterior predictive distribution for probability of superiority") +
  theme(panel.grid = element_blank())
```

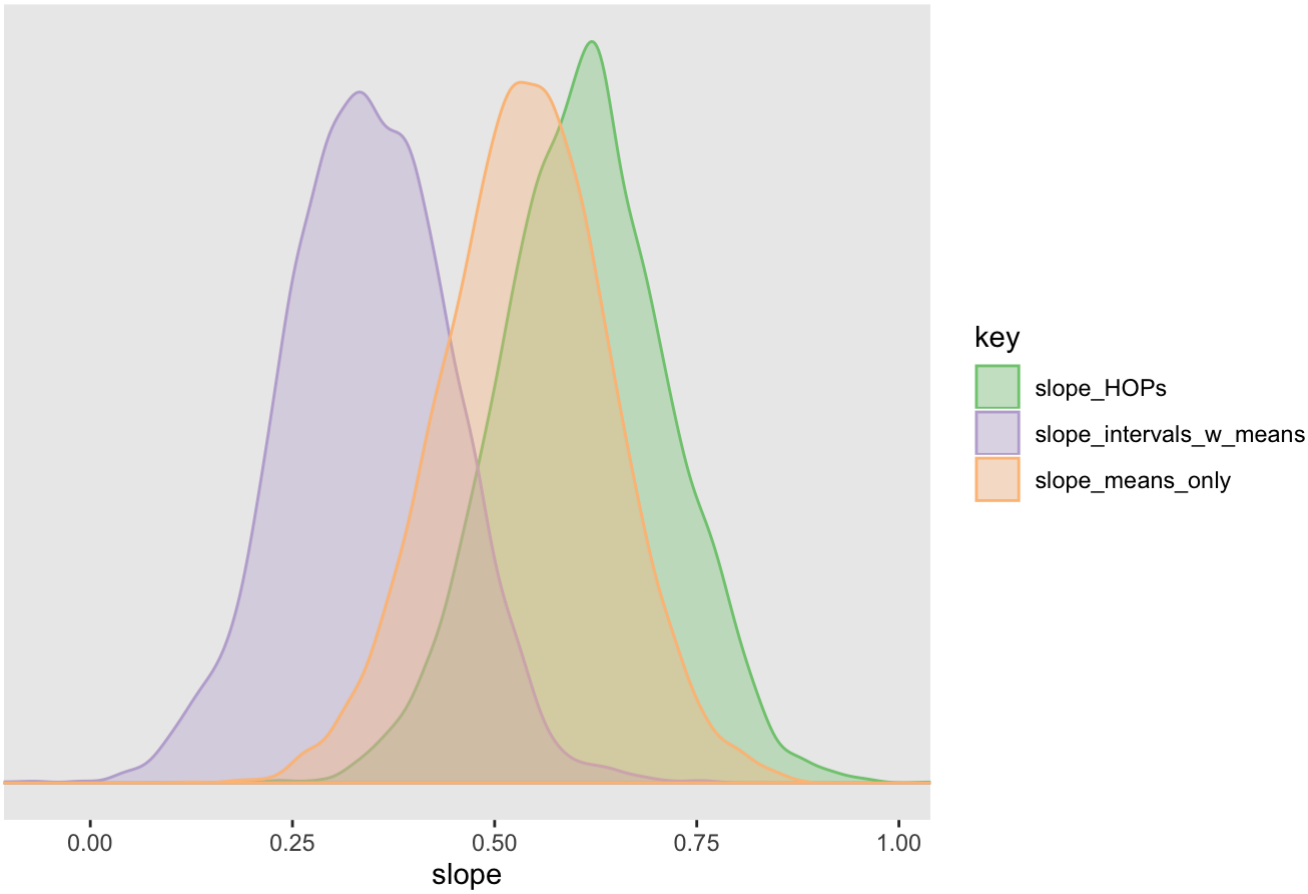
Posterior predictive distribution for probability of superiority



What do the posterior for the effect of each visualization condition look like?

```
# use posterior samples to define distributions for the slope in each visualization cond
ition
posterior_samples(m.vis.wrkr.llo_p_sup) %>%
  # transmute(slope_HOPs = `b_conditionHOPs:lo_ground_truth`,
  #           slope_intervals_w_means = `b_conditionintervals_w_means:lo_ground_truth`,
  #           slope_means_only = `b_conditionmeans_only:lo_ground_truth`) %>%
  transmute(slope_HOPs = `b_lo_ground_truth:conditionHOPs`,
            slope_intervals_w_means = `b_lo_ground_truth:conditionintervals_w_means`,
            slope_means_only = `b_lo_ground_truth:conditionmeans_only`) %>%
  gather(key, value) %>%
  ggplot(aes(x = value, group = key, color = key, fill = key)) +
  geom_density(alpha = 0.35) +
  scale_fill_brewer(type = "qual", palette = 1) +
  scale_color_brewer(type = "qual", palette = 1) +
  scale_x_continuous(expression(slope), expand = c(0, 0)) +
  scale_y_continuous(NULL, breaks = NULL) +
  labs(subtitle = "Posterior for slopes by visualization condition") +
  theme(panel.grid = element_blank())
```

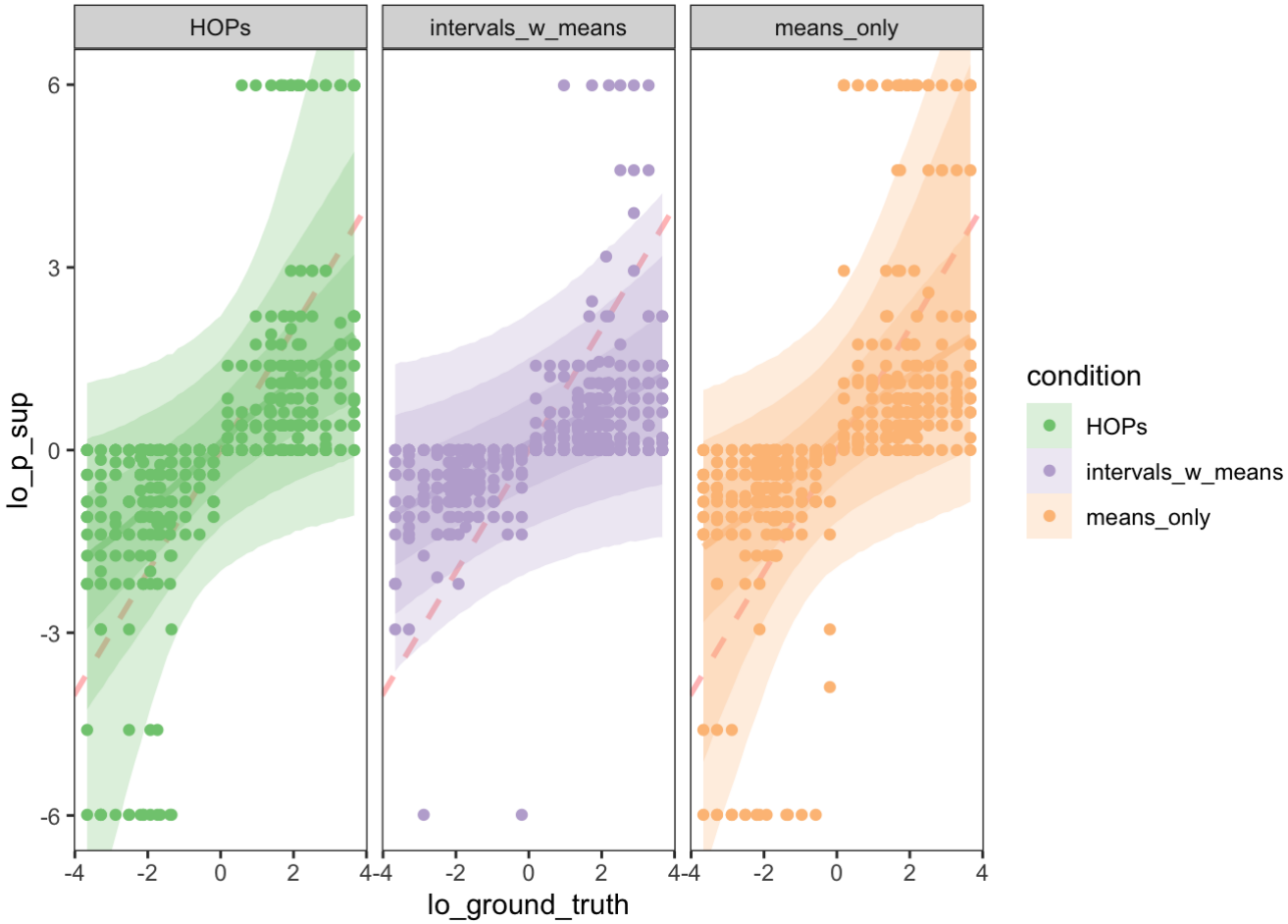
Posterior for slopes by visualization condition



The effect we saw earlier is still present but with more uncertainty now.

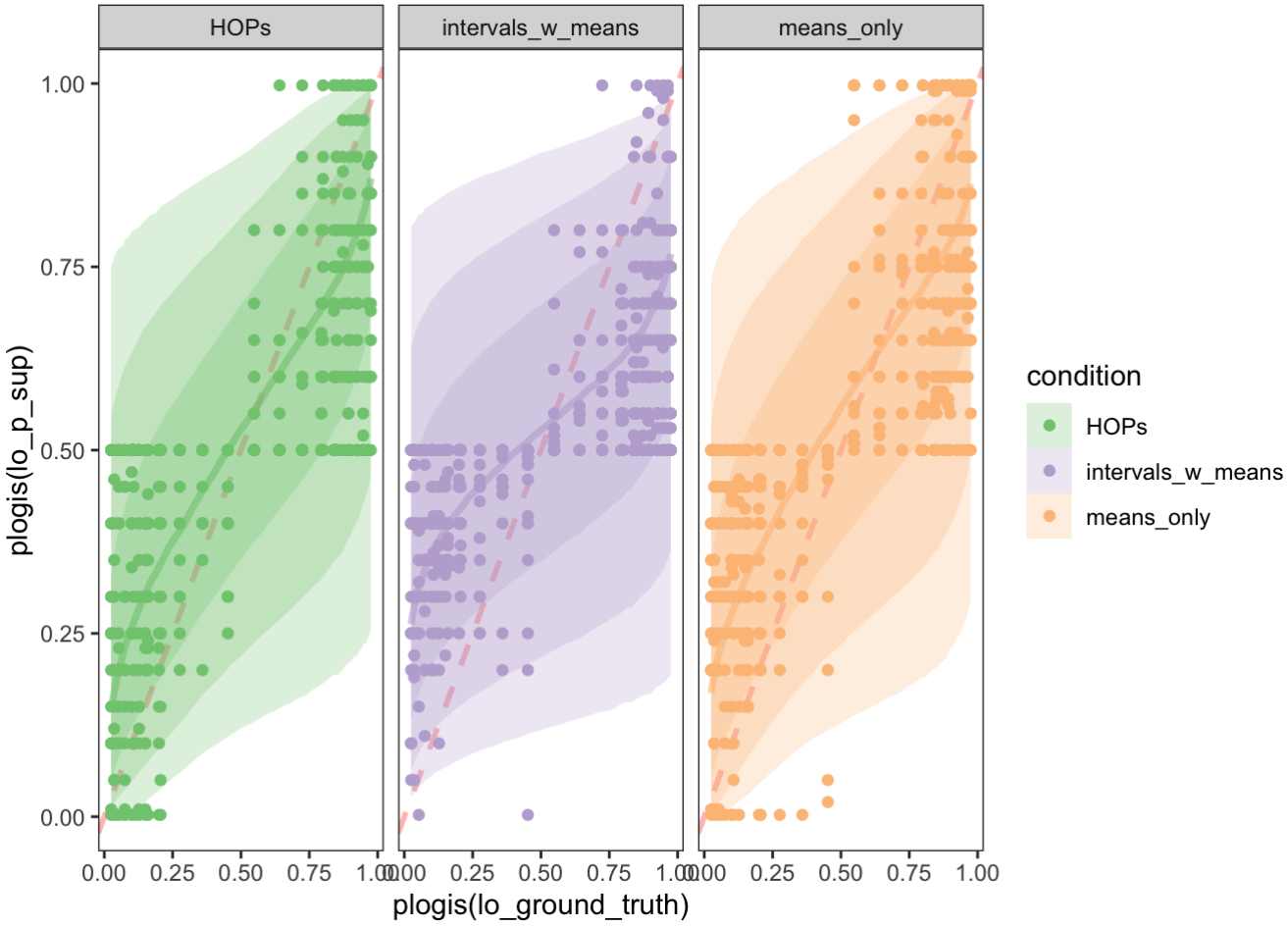
Let's take a look at some of the estimated linear models per visualization condition.

```
# this time we'll adopt functions from the tidybayes package to make plotting posterior
 predictions easier
model_df_llo %>%
  group_by(condition, worker_id) %>%
  data_grid(lo_ground_truth = seq_range(lo_ground_truth, n = 51)) %>%
  add_predicted_draws(m.vis.wrkr.llo_p_sup) %>%
  ggplot(aes(x = lo_ground_truth, y = lo_p_sup, color = condition, fill = condition)) +
  geom_abline(intercept = 0, slope = 1, size = 1, alpha = .3, color = "red", linetype =
"dashed") + # ground truth
  stat_lineribbon(aes(y = .prediction), .width = c(.95, .80, .50), alpha = .25) +
  geom_point(data = model_df_llo) +
  scale_fill_brewer(type = "qual", palette = 1) +
  scale_color_brewer(type = "qual", palette = 1) +
  coord_cartesian(xlim = quantile(model_df_llo$lo_ground_truth, c(0, 1)),
                  ylim = quantile(model_df_llo$lo_p_sup, c(0, 1))) +
  theme_bw() +
  theme(panel.grid = element_blank()) +
  facet_grid(. ~ condition)
```



What does this look like in probability units?

```
# this time we'll adopt functions from the tidybayes package to make plotting posterior
 predictions easier
model_df_llo %>%
  group_by(condition, worker_id) %>%
  data_grid(lo_ground_truth = seq_range(lo_ground_truth, n = 51)) %>%
  add_predicted_draws(m.vis.wrkr.llo_p_sup) %>%
  ggplot(aes(x = plogis(lo_ground_truth), y = plogis(lo_p_sup), color = condition, fill
 = condition)) +
  geom_abline(intercept = 0, slope = 1, size = 1, alpha = .3, color = "red", linetype =
"dashed") + # ground truth
  stat_lineribbon(aes(y = plogis(.prediction)), .width = c(.95, .80, .50), alpha = .25)
 +
  geom_point(data = model_df_llo) +
  scale_fill_brewer(type = "qual", palette = 1) +
  scale_color_brewer(type = "qual", palette = 1) +
  coord_cartesian(xlim = quantile(plogis(model_df_llo$lo_ground_truth), c(0, 1)),
                  ylim = quantile(plogis(model_df_llo$lo_p_sup), c(0, 1))) +
  theme_bw() +
  theme(panel.grid = element_blank()) +
  facet_grid(. ~ condition)
```



Again, this is promising, but it is still very noisy. Also, there are areas of high posterior density where there are no observations. Part of the reason for this that we restricted the response scale to 50-100 for this iteration of the pilot.

Add a Slope Interaction for Problem Framing

Similar to how we model the effects of visualization conditions on the slope of the LLO model as interactions with the ground truth, we add an interaction for gain/loss framing. This gives us a three way interaction.

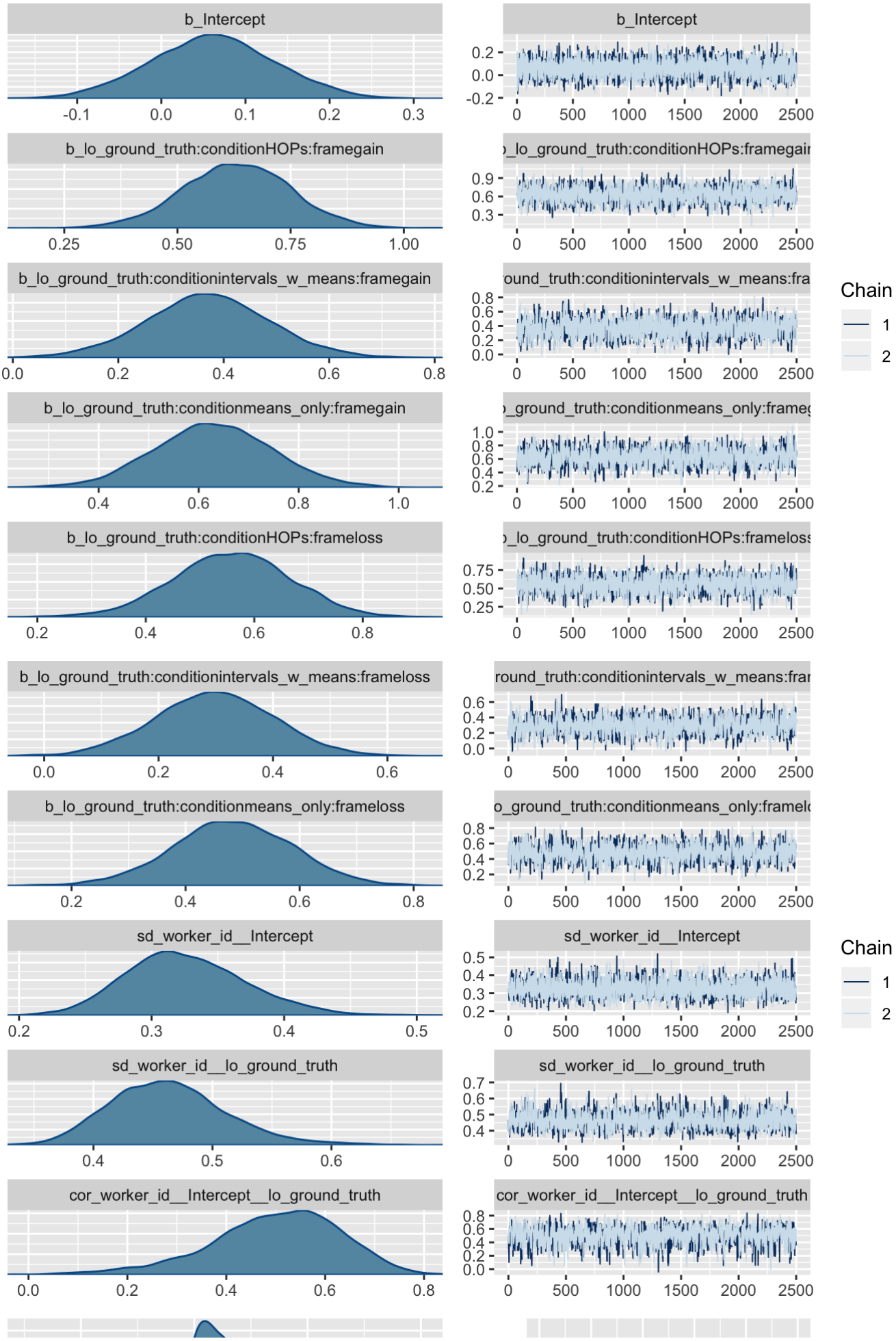
```
# update the llo model of p_sup responses to include an interaction
m.vis.frame.wrkr.llo_p_sup <- update(m.llo_p_sup,
                         formula = lo_p_sup ~ (1 + lo_ground_truth|worker_id) + lo_groun
d_truth:condition:frame,
                         newdata = model_df_llo,
                         file = "model-fits/llo_mdl_vis_frame_wrkr")
```

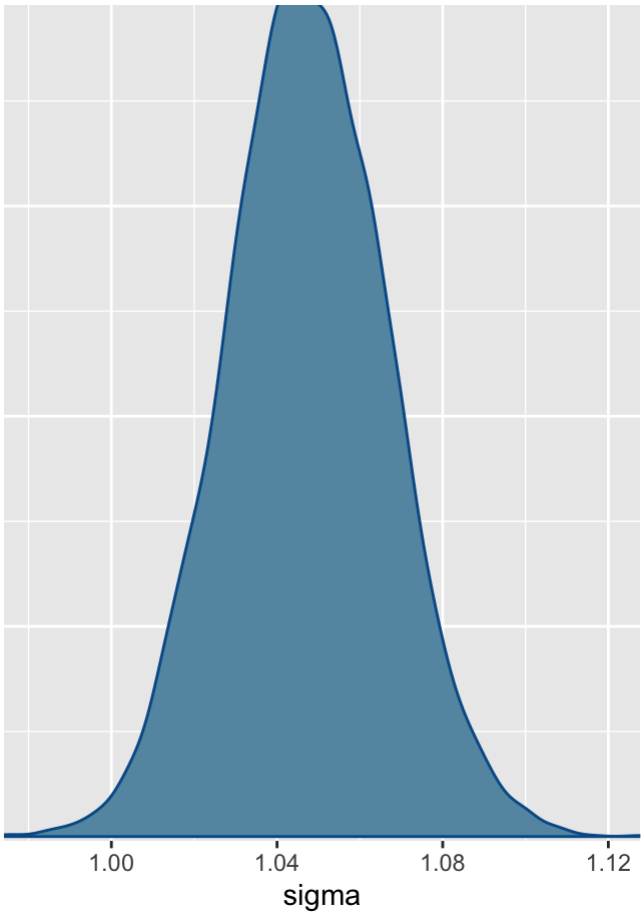
```
## The desired updates require recompiling the model
```

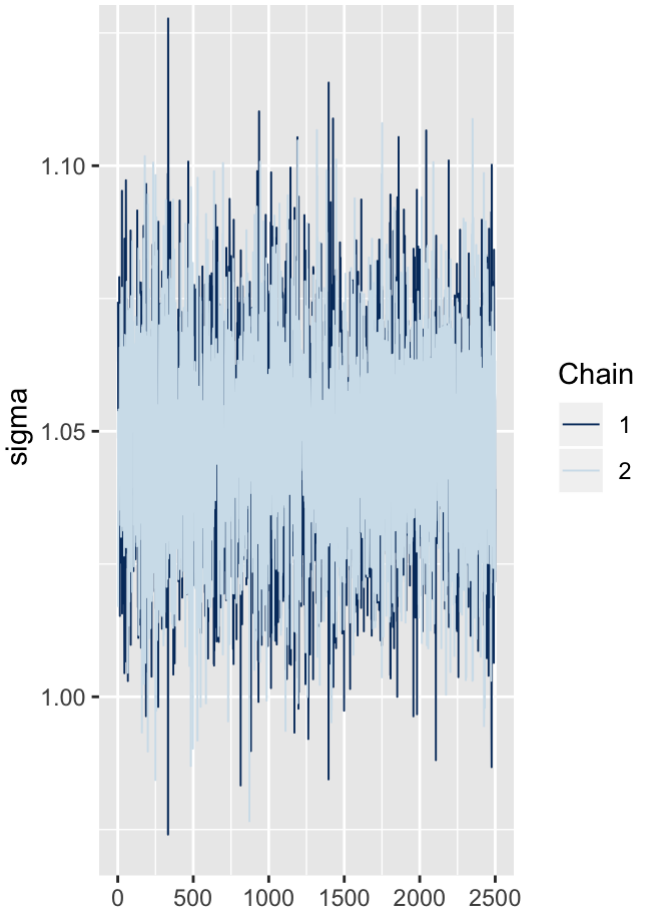
Check diagnostics:

- Trace plots

```
# trace plots
plot(m.vis.frame.wrkr.llo_p_sup)
```

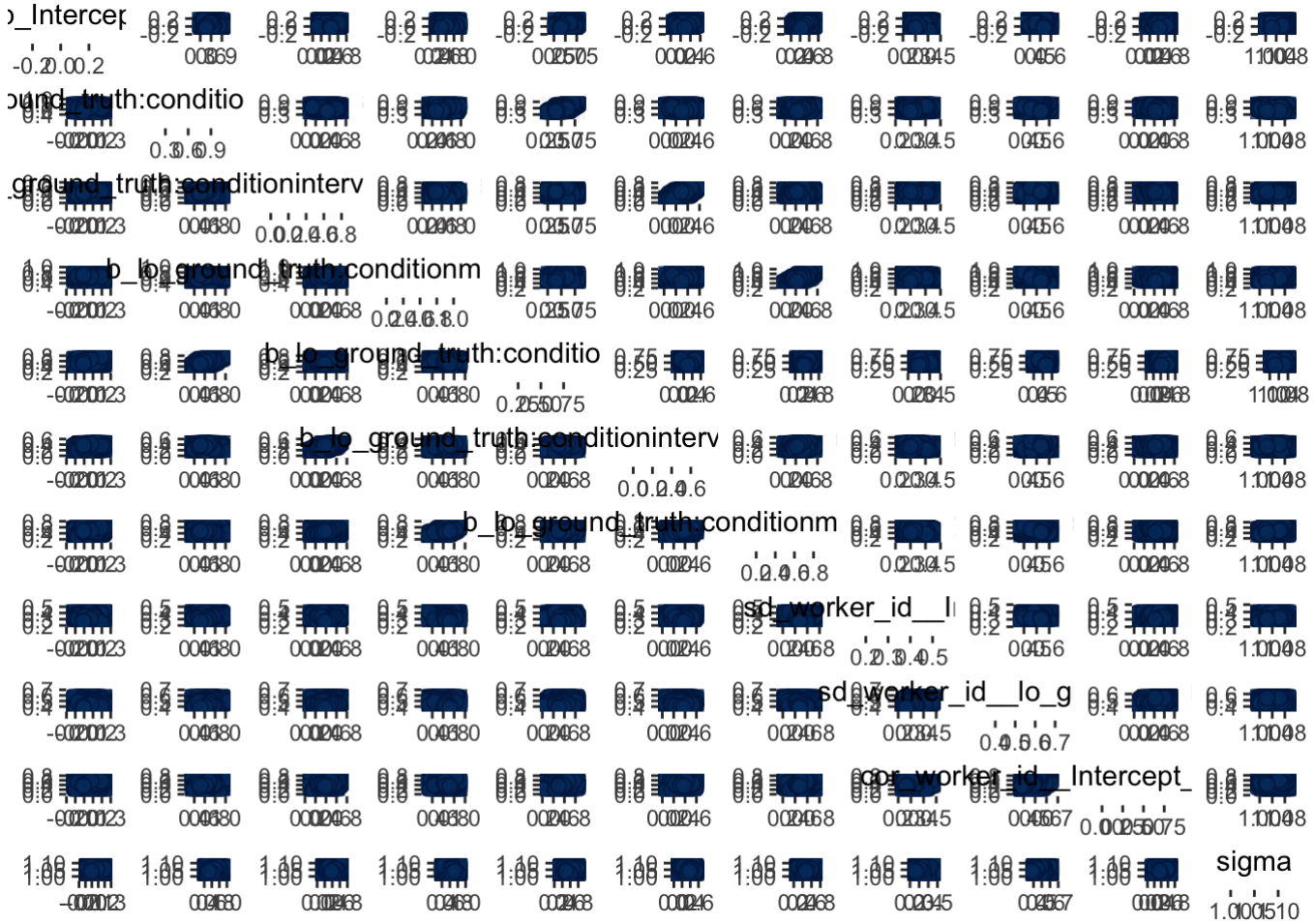

b_Intercept
b_lo_ground_truth:conditionHOPs:framegain
b_lo_ground_truth:conditionintervals_w_means:framegain
b_lo_ground_truth:conditionmeans_only:framegain
b_lo_ground_truth:conditionHOPs:frameloss
b_lo_ground_truth:conditionintervals_w_means:frameloss
b_lo_ground_truth:conditionmeans_only:frameloss
sd_worker_id__Intercept
sd_worker_id__lo_ground_truth
cor_worker_id__Intercept__lo_ground_truth
Chain
1
2





- Pairs plot

```
# pairs plot
pairs(m.vis.frame.wrkr.llo_p_sup)
```



- Summary

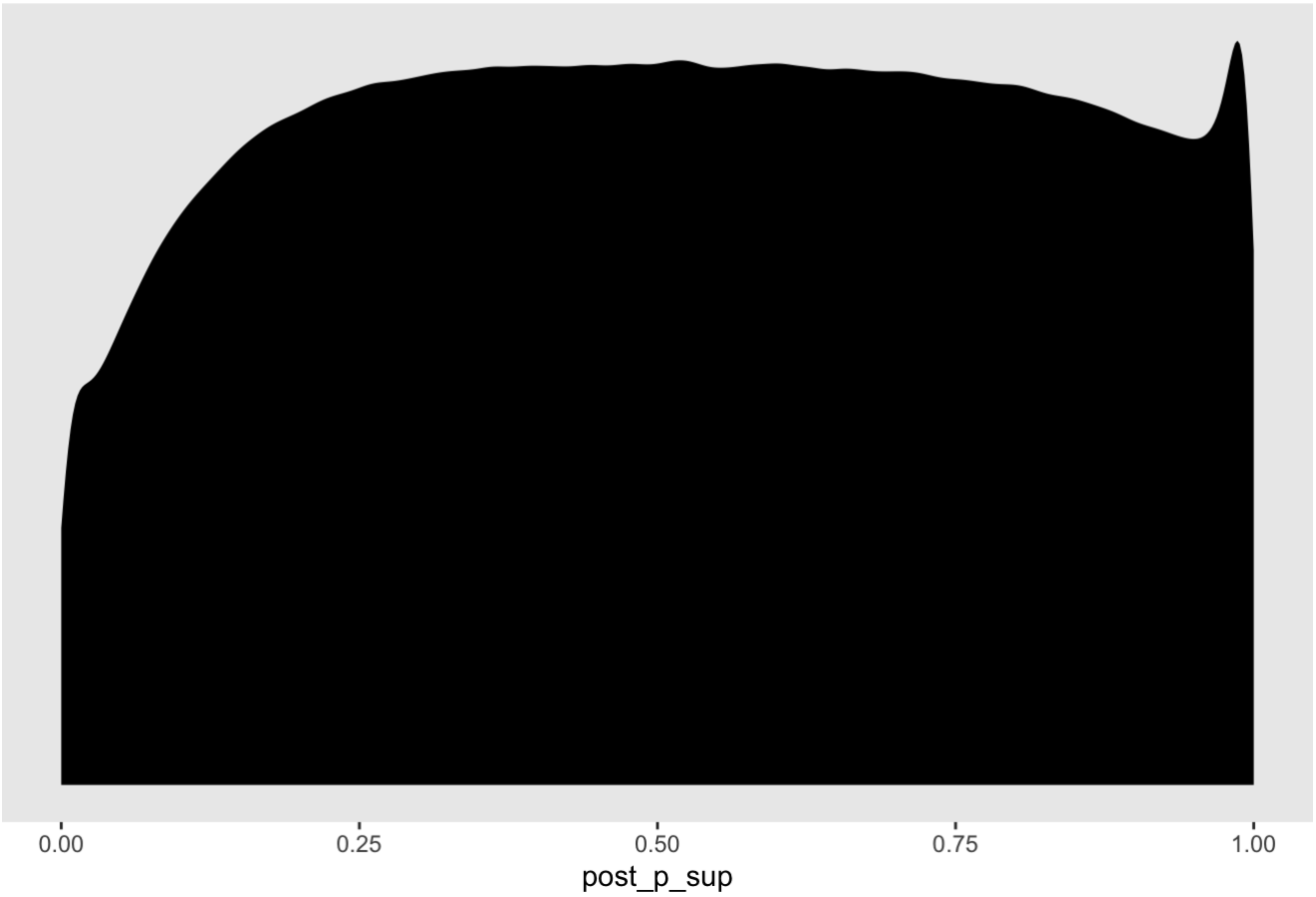
```
# model summary
print(m.vis.frame.wrkr.llo_p_sup)
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: lo_p_sup ~ (1 + lo_ground_truth | worker_id) + lo_ground_truth:condition:fra
me
##    Data: model_df_llo (Number of observations: 1568)
## Samples: 2 chains, each with iter = 3000; warmup = 500; thin = 1;
##          total post-warmup samples = 5000
##
## Group-Level Effects:
## ~worker_id (Number of levels: 56)
##                               Estimate Est.Error l-95% CI u-95% CI
## sd(Intercept)                     0.33      0.04     0.25     0.42
## sd(lo_ground_truth)               0.46      0.05     0.38     0.57
## cor(Intercept,lo_ground_truth)    0.50      0.13     0.20     0.72
##                               Eff.Sample Rhat
## sd(Intercept)                       2760 1.00
## sd(lo_ground_truth)                 1450 1.00
## cor(Intercept,lo_ground_truth)       980 1.00
##
## Population-Level Effects:
##                                                    Estimate Est.Error
## Intercept                                              0.06      0.07
## lo_ground_truth:conditionHOPs:framegain                0.63      0.12
## lo_ground_truth:conditionintervals_w_means:framegain    0.37      0.12
## lo_ground_truth:conditionmeans_only:framegain           0.63      0.12
## lo_ground_truth:conditionHOPs:frameloss                 0.56      0.11
## lo_ground_truth:conditionintervals_w_means:frameloss    0.30      0.10
## lo_ground_truth:conditionmeans_only:frameloss           0.48      0.11
##                                                    l-95% CI u-95% CI
## Intercept                                             -0.08     0.20
## lo_ground_truth:conditionHOPs:framegain                0.39     0.86
## lo_ground_truth:conditionintervals_w_means:framegain    0.14     0.60
## lo_ground_truth:conditionmeans_only:framegain           0.40     0.86
## lo_ground_truth:conditionHOPs:frameloss                 0.34     0.77
## lo_ground_truth:conditionintervals_w_means:frameloss    0.09     0.50
## lo_ground_truth:conditionmeans_only:frameloss           0.27     0.69
##                                                    Eff.Sample Rhat
## Intercept                                                2976 1.00
## lo_ground_truth:conditionHOPs:framegain                  1938 1.00
## lo_ground_truth:conditionintervals_w_means:framegain      1327 1.00
## lo_ground_truth:conditionmeans_only:framegain            1450 1.00
## lo_ground_truth:conditionHOPs:frameloss                  1744 1.00
## lo_ground_truth:conditionintervals_w_means:frameloss      1222 1.00
## lo_ground_truth:conditionmeans_only:frameloss            1286 1.00
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## sigma     1.05      0.02     1.01     1.09       8987 1.00
##
## Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
## is a crude measure of effective sample size, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

Let's check out a posterior predictive distribution for probability of superiority. This still looks too wide considering the number of responses near 50% that we see in the data.

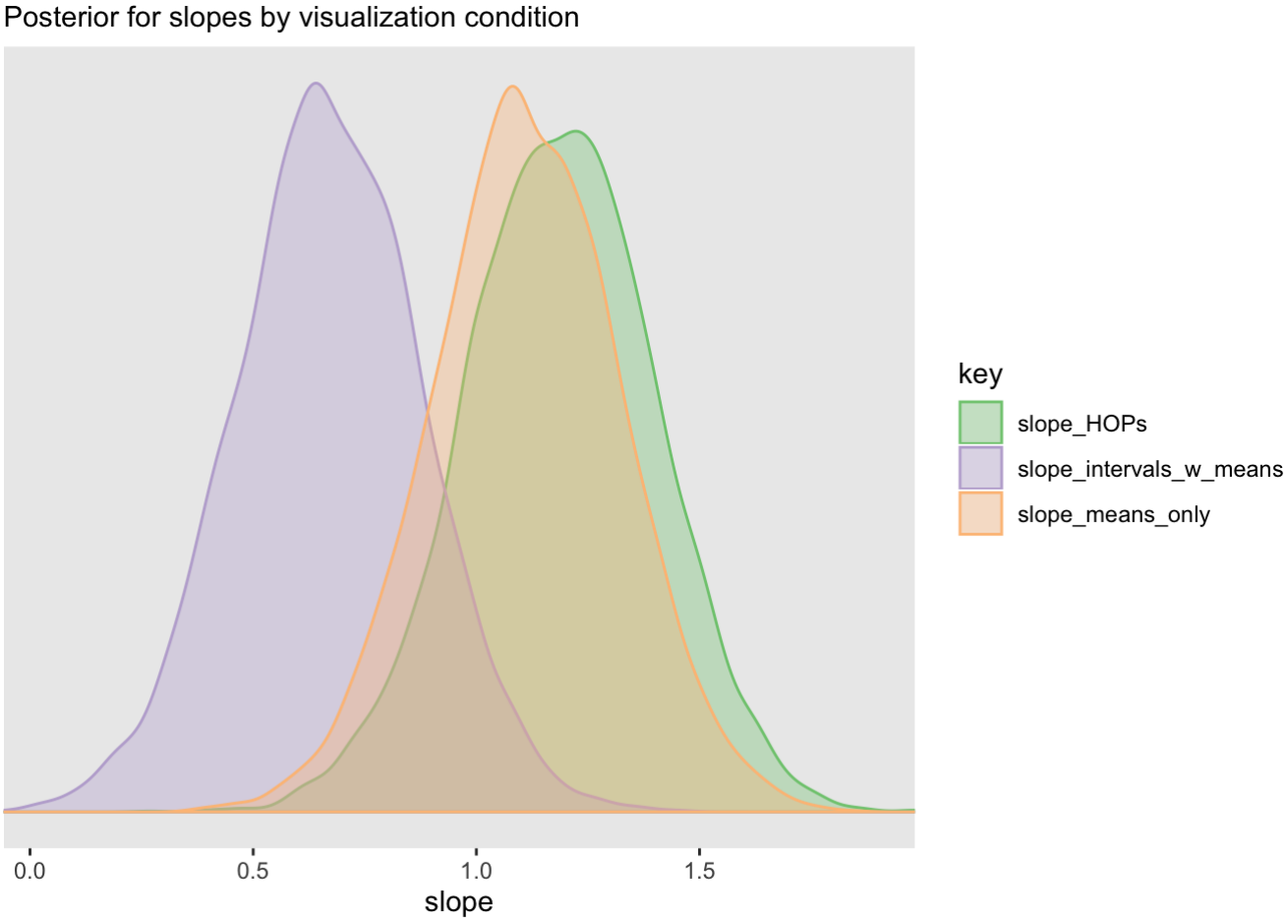
```
# posterior predictive check
model_df_llo %>%
  select(lo_ground_truth, condition, frame, worker_id) %>%
  add_predicted_draws(m.vis.frame.wrkr.llo_p_sup, prediction = "lo_p_sup", seed = 1234)
 %>%
  mutate(post_p_sup = plogis(lo_p_sup)) %>%
  ggplot(aes(x = post_p_sup)) +
  geom_density(fill = "black", size = 0) +
  scale_y_continuous(NULL, breaks = NULL) +
  labs(subtitle = "Posterior predictive distribution for probability of superiority") +
  theme(panel.grid = element_blank())
```

Posterior predictive distribution for probability of superiority



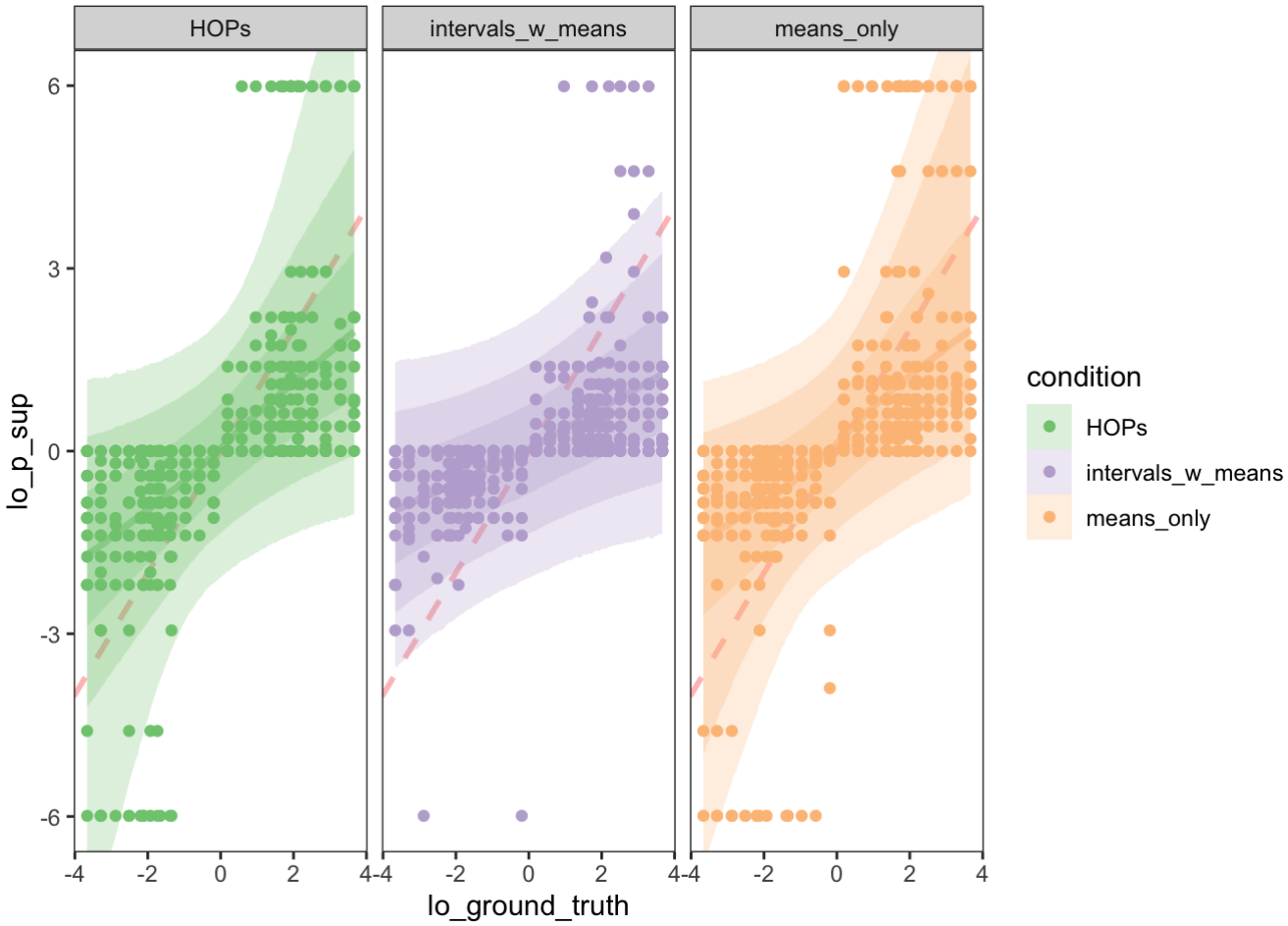
What does the posterior for the effect of each visualization condition look like?

```
# use posterior samples to define distributions for the slope in each visualization cond
ition
posterior_samples(m.vis.frame.wrkr.llo_p_sup) %>%
  transmute(slope_HOPs = `b_lo_ground_truth:conditionHOPs:framegain` + `b_lo_ground_trut
h:conditionHOPs:frameloss`,
            slope_intervals_w_means = `b_lo_ground_truth:conditionintervals_w_means:fram
egain` + `b_lo_ground_truth:conditionintervals_w_means:frameloss`,
            slope_means_only = `b_lo_ground_truth:conditionmeans_only:framegain` + `b_lo
_ground_truth:conditionmeans_only:frameloss`) %>%
  gather(key, value) %>%
  ggplot(aes(x = value, group = key, color = key, fill = key)) +
  geom_density(alpha = 0.35) +
  scale_fill_brewer(type = "qual", palette = 1) +
  scale_color_brewer(type = "qual", palette = 1) +
  scale_x_continuous(expression(slope), expand = c(0, 0)) +
  scale_y_continuous(NULL, breaks = NULL) +
  labs(subtitle = "Posterior for slopes by visualization condition") +
  theme(panel.grid = element_blank())
```



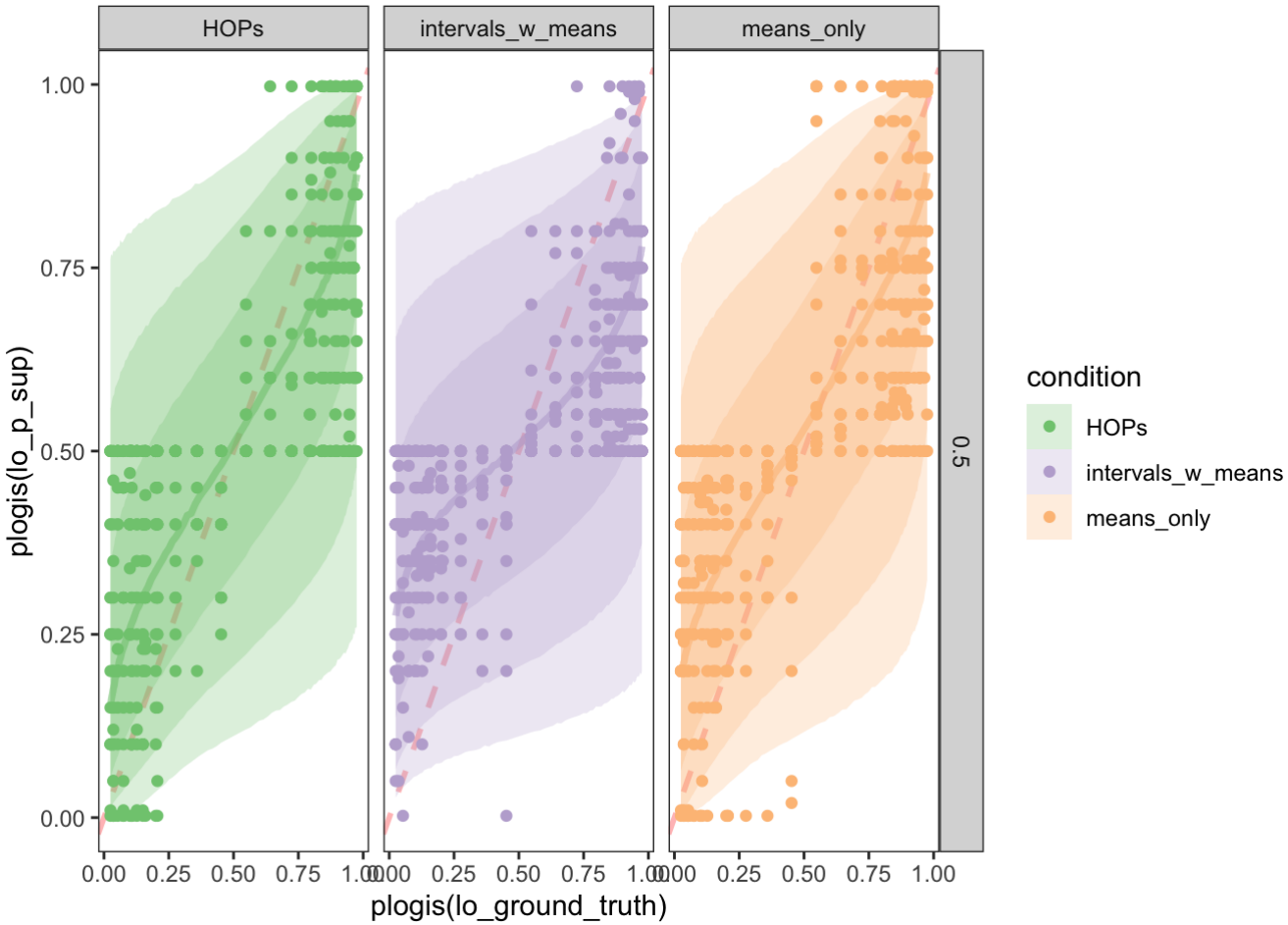
Let's take a look at some of the estimated linear models per visualization condition. Adding different slopes per framing condition should introduce some asymmetry on either side of 50%.

```
# this time we'll adopt functions from the tidybayes package to make plotting posterior
 predictions easier
model_df_llo %>%
  group_by(condition, frame, worker_id) %>%
  data_grid(lo_ground_truth = seq_range(lo_ground_truth, n = 51)) %>%
  add_predicted_draws(m.vis.frame.wrkr.llo_p_sup) %>%
  ggplot(aes(x = lo_ground_truth, y = lo_p_sup, color = condition, fill = condition)) +
  geom_abline(intercept = 0, slope = 1, size = 1, alpha = .3, color = "red", linetype =
"dashed") + # ground truth
  stat_lineribbon(aes(y = .prediction), .width = c(.95, .80, .50), alpha = .25) +
  geom_point(data = model_df_llo) +
  scale_fill_brewer(type = "qual", palette = 1) +
  scale_color_brewer(type = "qual", palette = 1) +
  coord_cartesian(xlim = quantile(model_df_llo$lo_ground_truth, c(0, 1)),
                  ylim = quantile(model_df_llo$lo_p_sup, c(0, 1))) +
  theme_bw() +
  theme(panel.grid = element_blank()) +
  facet_grid(. ~ condition)
```



What does this look like in probability units?

```
model_df_llo %>%
  group_by(condition, frame, worker_id) %>%
  data_grid(lo_ground_truth = seq_range(lo_ground_truth, n = 51)) %>%
  add_predicted_draws(m.vis.frame.wrkr.llo_p_sup) %>%
  ggplot(aes(x = plogis(lo_ground_truth), y = plogis(lo_p_sup), color = condition, fill
 = condition)) +
  geom_abline(intercept = 0, slope = 1, size = 1, alpha = .3, color = "red", linetype =
"dashed") + # ground truth
  stat_lineribbon(aes(y = plogis(.prediction)), .width = c(.95, .80, .50), alpha = 1/4)
 +
  geom_point(data = model_df_llo) +
  scale_fill_brewer(type = "qual", palette = 1) +
  scale_color_brewer(type = "qual", palette = 1) +
  coord_cartesian(xlim = quantile(plogis(model_df_llo$lo_ground_truth), c(0, 1)),
                  ylim = quantile(plogis(model_df_llo$lo_p_sup), c(0, 1))) +
  theme_bw() +
  theme(panel.grid = element_blank()) +
  facet_grid(baseline ~ condition)
```



Let's check whether adding parameters to account for problem framing is worth it insofar as the added parameters contribute more to predictive validity than they contribute to overfitting. We'll determine this by comparing the models with and without parameters for baseline according to the widely applicable information criterion (WAIC). Lower values of WAIC indicate a better fitting model.

```
waic(m.vis.wrkr.llo_p_sup, m.vis.frame.wrkr.llo_p_sup)
```

```
##                                                    WAIC     SE
## m.vis.wrkr.llo_p_sup                             4702.79 132.18
## m.vis.frame.wrkr.llo_p_sup                       4704.10 132.12
## m.vis.wrkr.llo_p_sup - m.vis.frame.wrkr.llo_p_sup   -1.31   4.76
```

The fact that WAIC values for the two models are approximately equal suggests that we don't get much improvement in model fit from adding the parameters for problem framing to this model. This makes sense as the problem framing is more likely to impact decisions than probability judgments.

A Mixture Model of Fake Probability of Superiority Responses

In order to help our model accommodate the fact that some workers choose the same response regardless of the ground truth, we will need a mixture between two submodels: a LLO process vs a constant response process. To get this working, we'll start by building a fake data set with a known data-generating process.

Fake Data

This simulated data is a 20/80 mixture of the ground truth vs a constant response of 99.75%, both with standard normal noise added. Let's prep the data for the model.

```
# parameters to recover
p_process <- c(0.2, 0.8) # population probabilities of each process
Sigma <- matrix(c(1, .1, .1, 1), 2, 2) # covariance matrix
sigma_err <- 1 # residual error in log odds units

# number of trials per participant (should be a multiple of 24 data conditions)
n_trials_per_worker <- 200
n_trial_scalar <- n_trials_per_worker / 24

# softmax function
softmax <- function(x) {
  return(exp(x) / sum(exp(x)))
}

# predictions on each trial per process: llo vs guess_one
process_df <- responses_df %>% rowwise() %>%
  mutate(
    llo = ground_truth * 100, # our simulated llo model has zero bias
    guess_one = 99.75
  ) %>%
  gather(process, process_est, llo, guess_one) %>% # reshape
  select(worker_id, ground_truth, process, process_est)

# user same p_process for each worker (no hierarchy)
fake_worker_params_df <- model_df_llo %>%
  select(worker_id) %>%
  distinct(worker_id) %>%
  rowwise() %>%
  mutate(
    p_process_worker = list(p_process)) # same value for each worker (no hierarchy)

# fake data
fake_df_llo_mix <- model_df_llo %>%
  left_join(fake_worker_params_df, by="worker_id") %>%
  slice(rep(1:n(), each = n_trial_scalar)) %>%
  rowwise() %>%
  mutate(process = sample(x = c("llo", "guess_one"), size = n(), replace = TRUE, prob =
 p_process_worker)) %>% # sample process to use on each trial
  left_join(process_df, by = c("worker_id", "ground_truth", "process")) %>% # add proces
s estimates to model_df
  select(-p_superiority) %>% # remove actual responses
  mutate(
    lo_p_sup = qlogis(process_est / 100) + rnorm(n(), 0, sigma_err), # likelihood functi
on
    p_sup = plogis(lo_p_sup) * 100, # simulated responses from known data generating pro
cess
    lo_ground_truth = qlogis(ground_truth)
  )
```

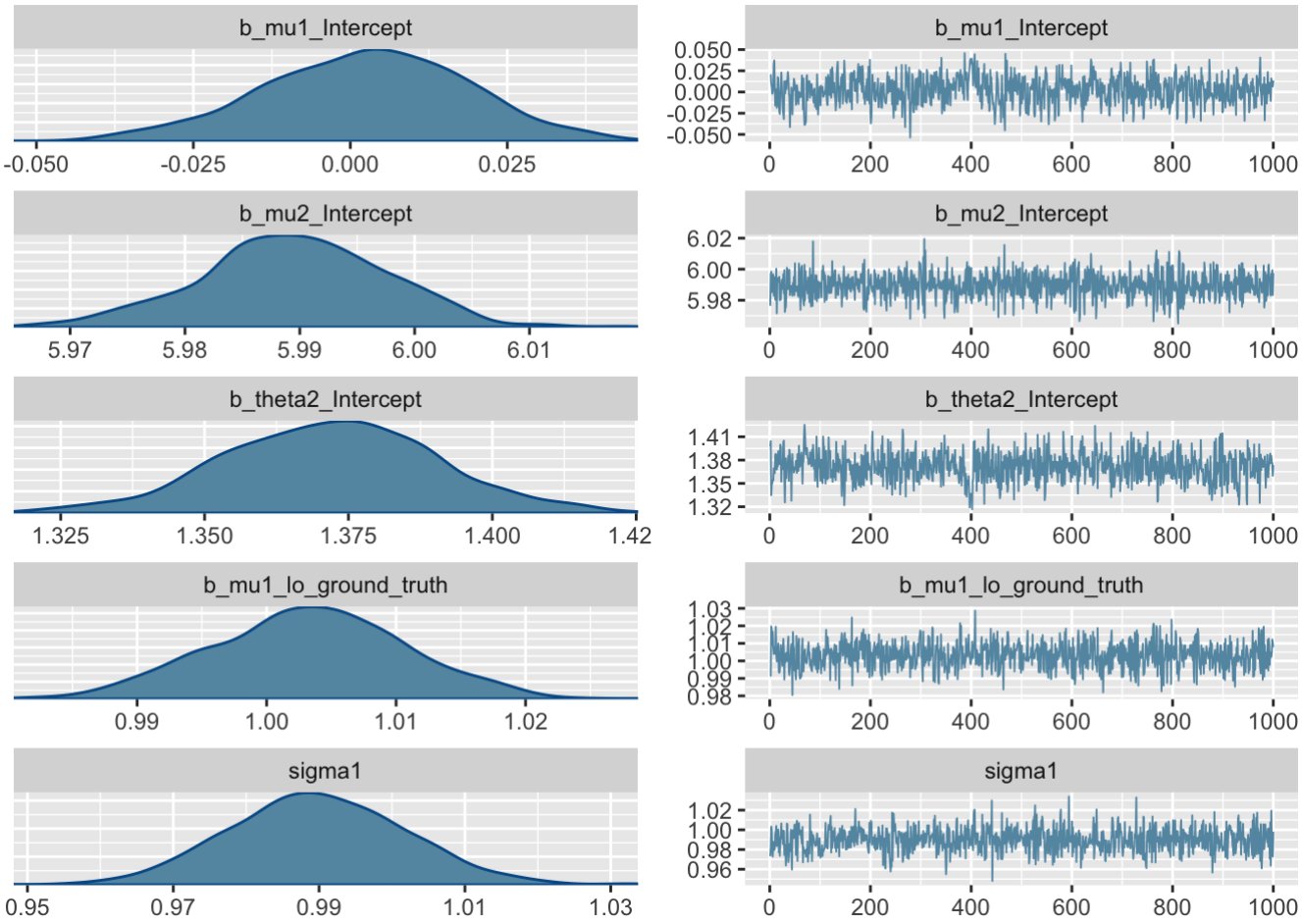
We fit a model that matches the data-generating process, and importantly, we estimate the log odds of the constant response strategy theta2.

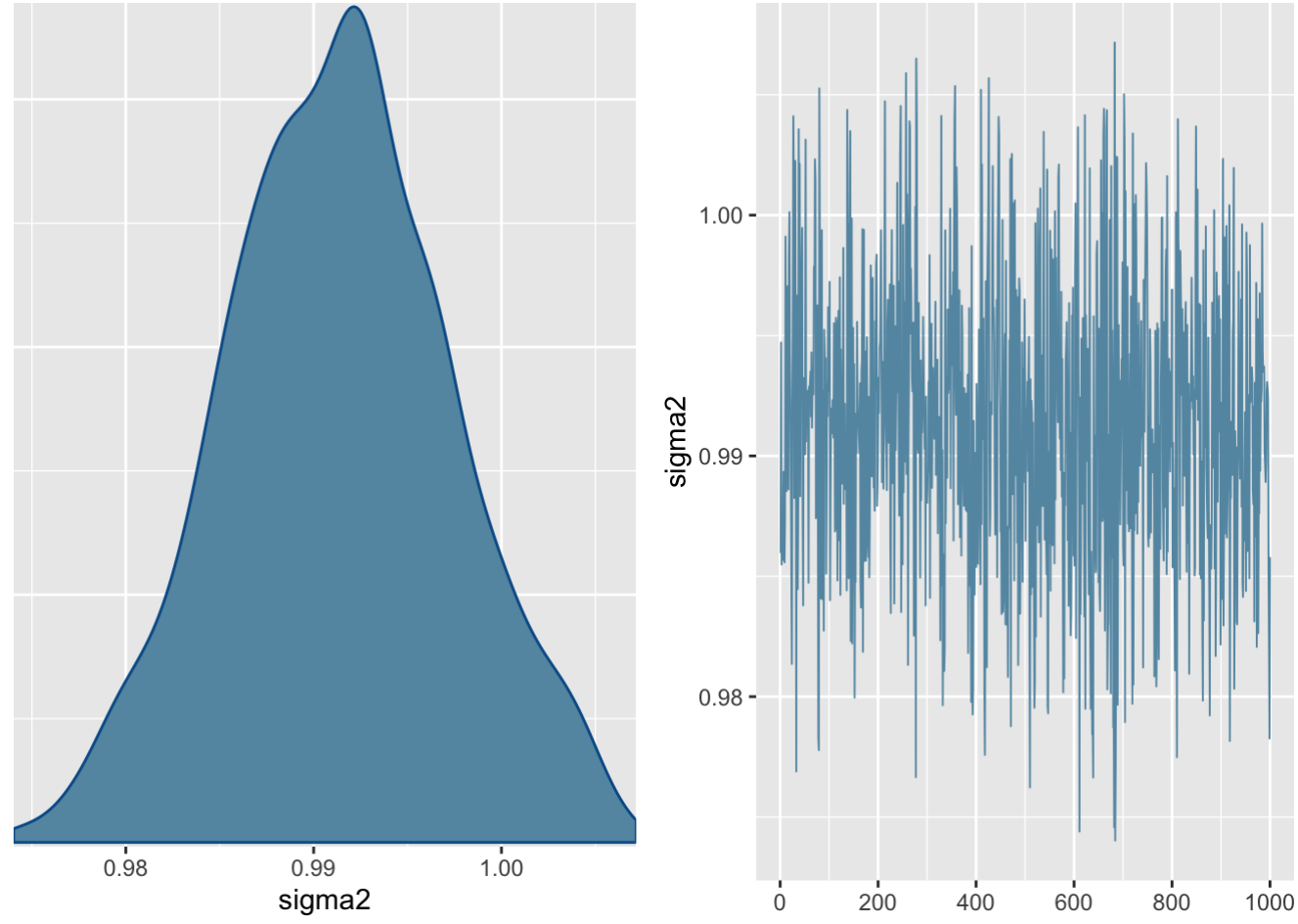
```
# get_prior(
#   bf(lo_p_sup ~ 1, mu1 ~ lo_ground_truth, mu2 ~ 1, theta2 ~ 1),
#     data = fake_df_llo_mix,
#     family = mixture(gaussian, gaussian)
# )

# fit the model
m.fake.llo_mix <- brm(
  bf(lo_p_sup ~ 1, mu1 ~ lo_ground_truth, mu2 ~ 1, theta2 ~ 1),
  data = fake_df_llo_mix,
  family = mixture(gaussian, gaussian, order = 'mu'),
  prior = c(
    prior(normal(0, 1), class = b, dpar = mu1),
    prior(normal(0, 1), class = Intercept, dpar = mu1),
    prior(normal(0, 1), class = Intercept, dpar = mu2)
  ),
  inits = 1, chains = 1, cores = 2,
  control = list(adapt_delta = 0.999, max_treedepth=12),
  file = "model-fits/llo_mix_mdl_fake"
)
```

Check diagnostics:

- Trace plots

```
# trace plots
plot(m.fake.llo_mix)
```

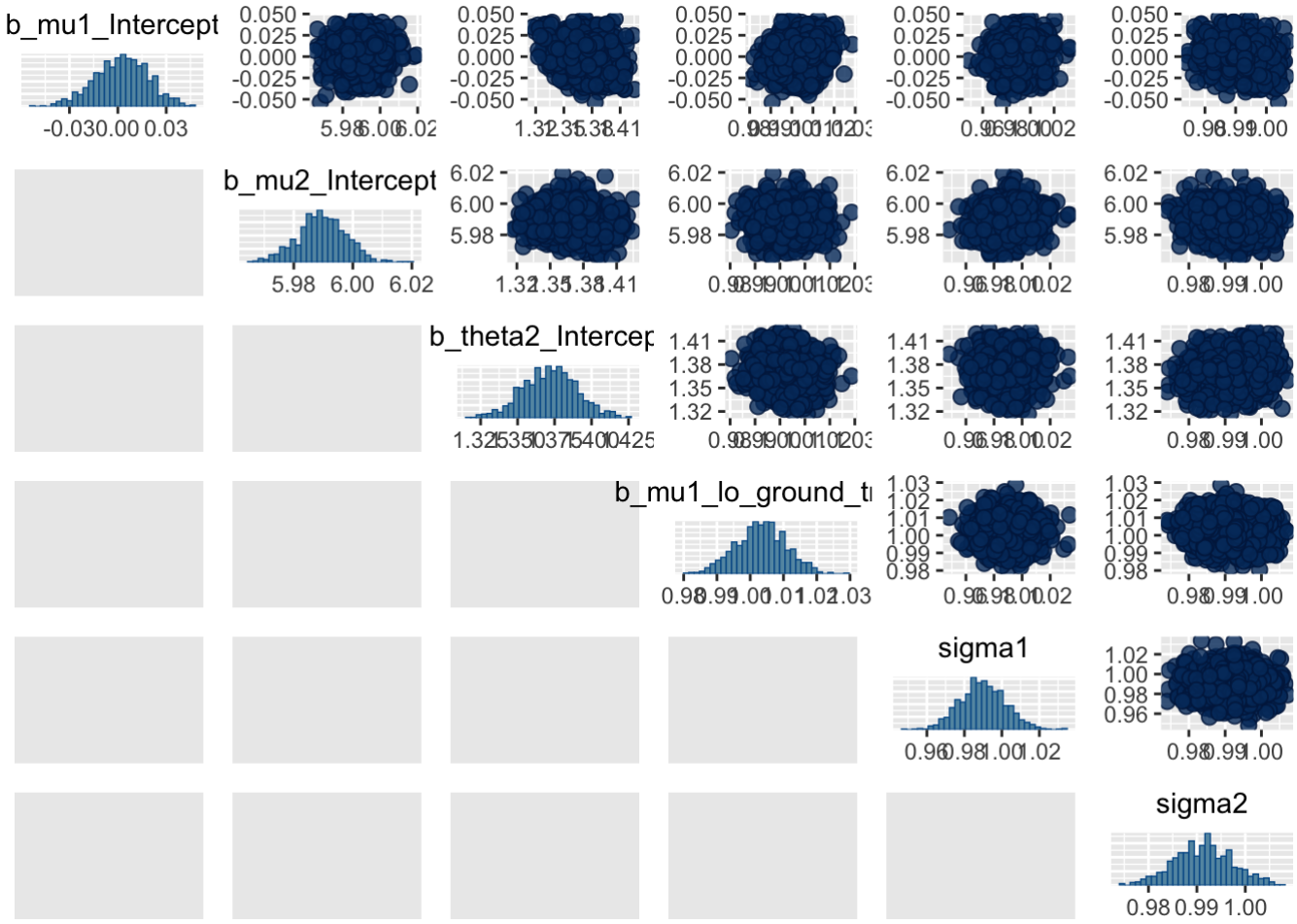

b_mu1_Intercept
-0.050
-0.025
0.000
0.025
b_mu1_Intercept
0.050
0.025
0.000
-0.025
-0.050
0
200
400
600
800
1000
b_mu2_Intercept
5.97
5.98
5.99
6.00
6.01
b_mu2_Intercept
6.02
6.00
5.98
b_theta2_Intercept
1.325
1.350
1.375
1.400
1.42
b_theta2_Intercept
1.41
1.38
1.35
1.32
b_mu1_lo_ground_truth
0.99
1.00
1.01
1.02
b_mu1_lo_ground_truth
1.03
1.02
1.01
1.00
0.99
0.98
sigma1
0.95
0.97
0.99
1.01
1.03
sigma1
1.02
1.00
0.98
0.96


0.98
0.99
1.00
sigma2
sigma2
1.00
0.99
0.98
0
200
400
600
800
1000

- Pairs plot

```
# pairs plot
pairs(m.fake.llo_mix)
```

```
## Warning in bayesplot::mcmc_pairs(samples, ...): Only one chain in 'x'. This
## plot is more useful with multiple chains.
```



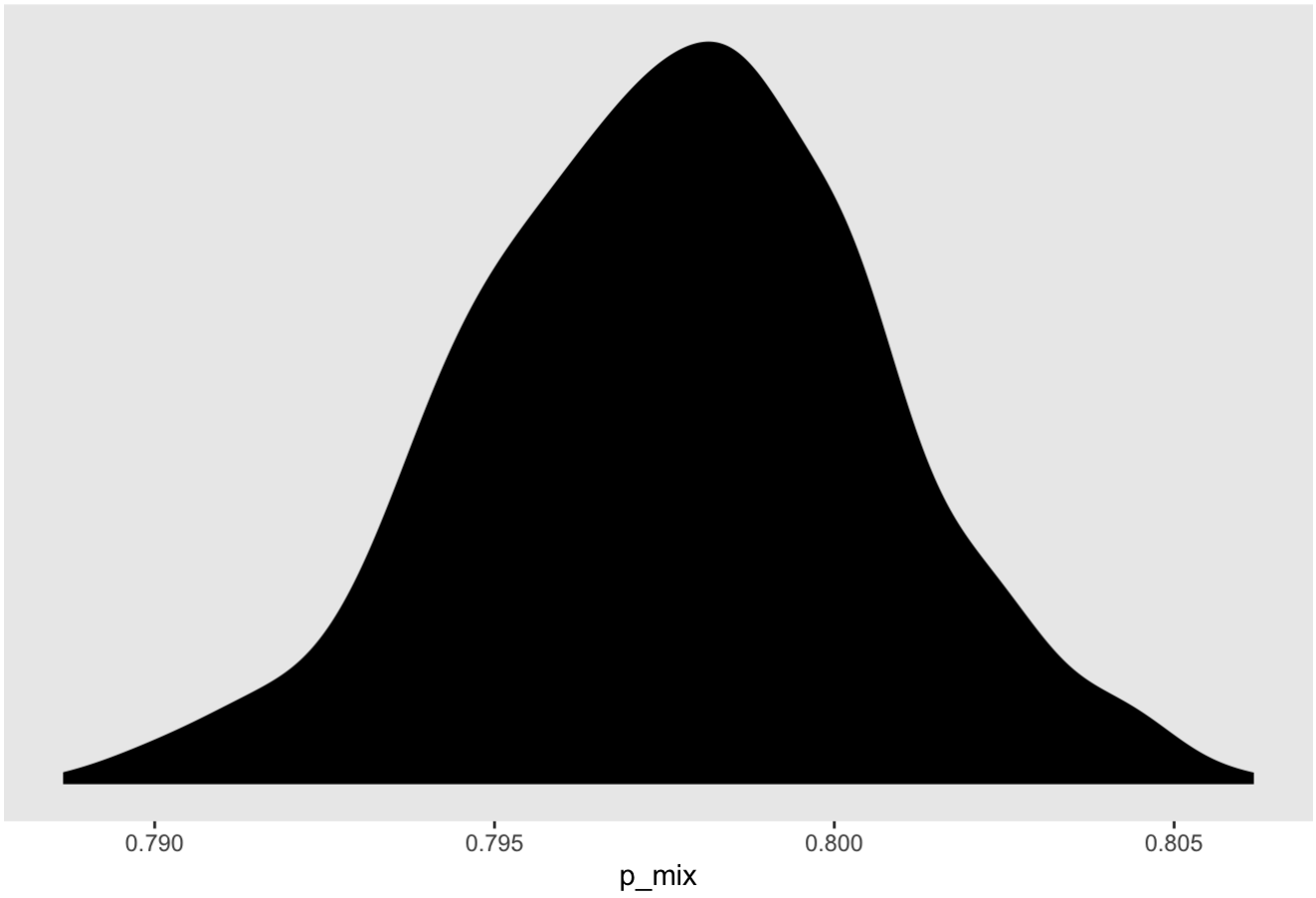
- Summary

```
# model summary
print(m.fake.llo_mix)
```

```
##  Family: mixture(gaussian, gaussian)
##   Links: mu1 = identity; sigma1 = identity; mu2 = identity; sigma2 = identity; theta1
= identity; theta2 = identity
## Formula: lo_p_sup ~ 1
##          mu1 ~ lo_ground_truth
##          mu2 ~ 1
##          theta2 ~ 1
##    Data: fake_df_llo_mix (Number of observations: 20928)
## Samples: 1 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 1000
##
## Population-Level Effects:
##                     Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## mu1_Intercept           0.00      0.02    -0.03     0.03        490 1.00
## mu2_Intercept           5.99      0.01     5.97     6.00       1147 1.00
## theta2_Intercept        1.37      0.02     1.33     1.41        575 1.00
## mu1_lo_ground_truth     1.00      0.01     0.99     1.02        892 1.00
##
## Family Specific Parameters:
##        Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## sigma1     0.99      0.01     0.97     1.01        751 1.00
## sigma2     0.99      0.01     0.98     1.00        720 1.01
##
## Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
## is a crude measure of effective sample size, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

So, did we accurately recover the mixture proportions? Let's find out by plotting the posterior for theta. Because theta is in log odds units we'll transform it into probability units.

```
# posteriors of mixture proportion
posterior_samples(m.fake.llo_mix) %>%
  mutate(p_mix = plogis(b_theta2_Intercept)) %>%
  ggplot(aes(x = p_mix)) +
  geom_density(fill = "black", size = 0) +
  scale_y_continuous(NULL, breaks = NULL) +
  labs(subtitle = "Posterior distribution for mixture proportion") +
  theme(panel.grid = element_blank())
```

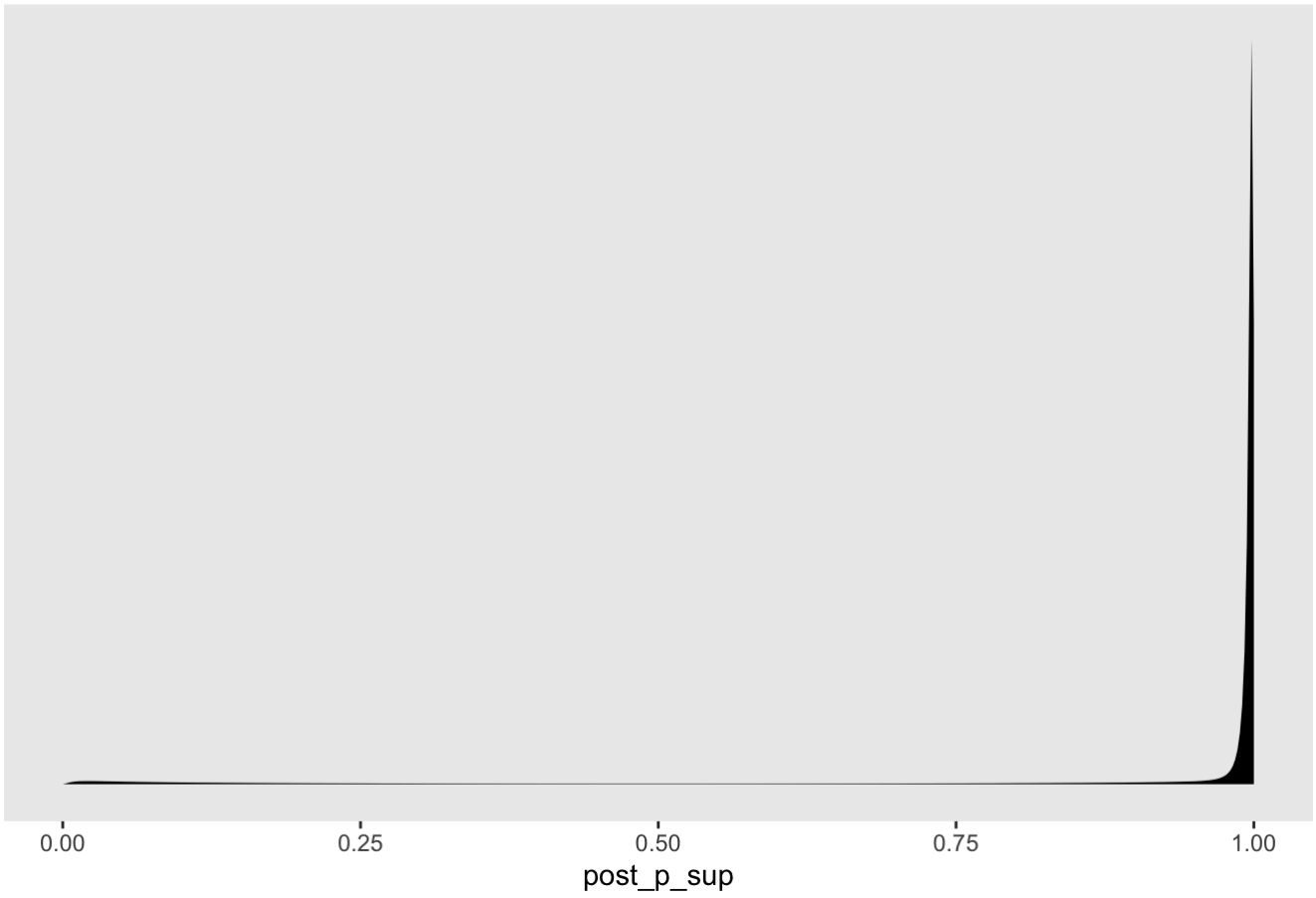


We succeeded in estimating the proportion of each alternative within a simulated mixture. This approach should help us to dramatically improve our model of real data.

Let's also check out a posterior predictive distribution for the fake probability of superiority responses.

```
# posterior predictive check
fake_df_llo_mix %>%
  select(lo_ground_truth, worker_id) %>%
  add_predicted_draws(m.fake.llo_mix, prediction = "lo_p_sup", seed = 1234) %>%
  mutate(post_p_sup = plogis(lo_p_sup)) %>%
  ggplot(aes(x = post_p_sup)) +
  geom_density(fill = "black", size = 0) +
  scale_y_continuous(NULL, breaks = NULL) +
  labs(subtitle = "Posterior predictive distribution for probability of superiority") +
  theme(panel.grid = element_blank())
```

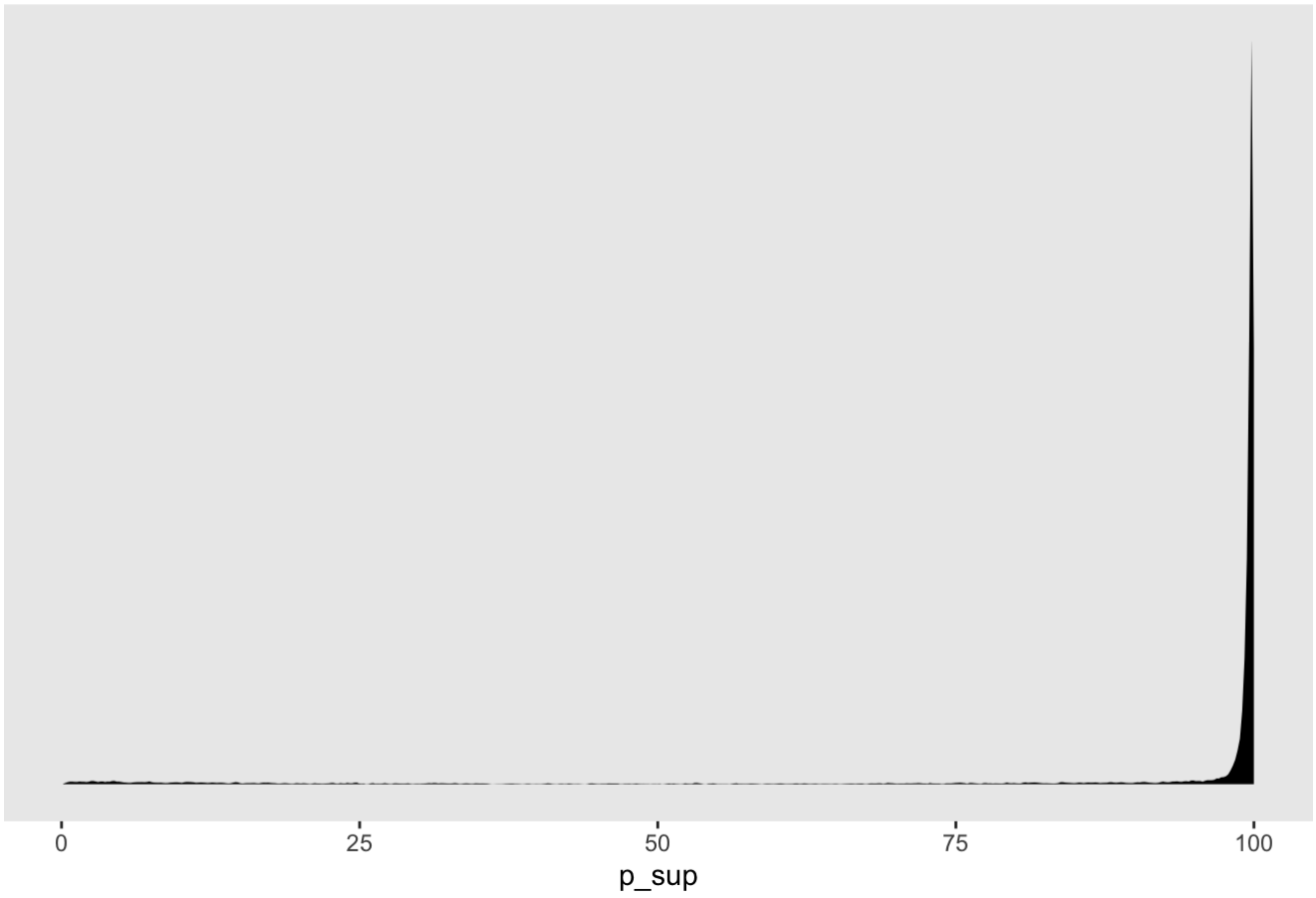
Posterior predictive distribution for probability of superiority



And lets's compare this to the data distribution.

```
# posterior predictive check
fake_df_llo_mix %>%
  ggplot(aes(x = p_sup)) +
  geom_density(fill = "black", size = 0) +
  scale_y_continuous(NULL, breaks = NULL) +
  labs(subtitle = "Distribution of fake probability of superiority responses") +
  theme(panel.grid = element_blank())
```

Distribution of fake probability of superiority responses



Mixture of the LLO Model and a Random Constant Response

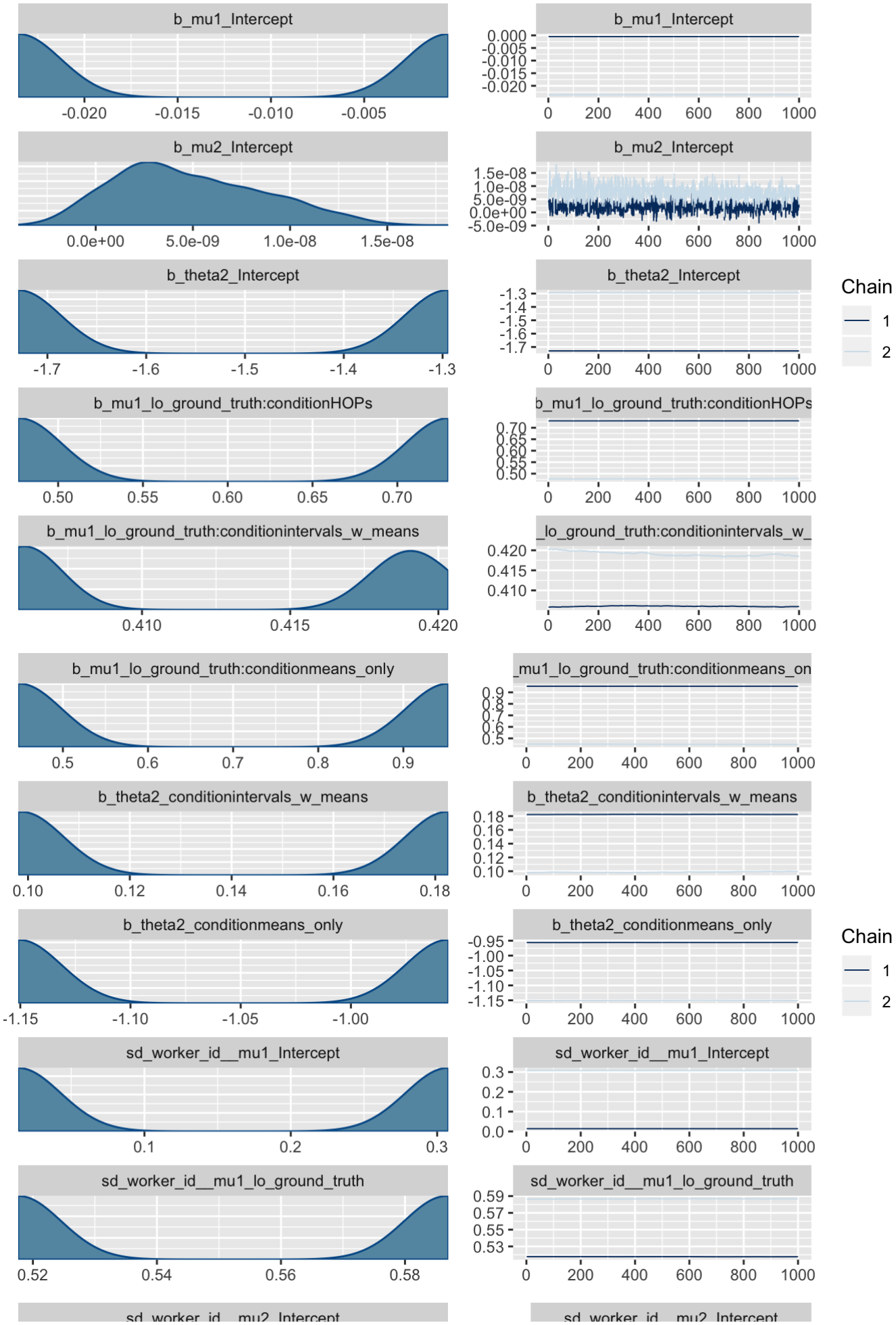
Let's adopt this mixture model for our actual probability of superiority responses by adding a constant response distribution to the iteration of the LLO model with different slopes per visualization condition and random slopes and intercepts per worker.

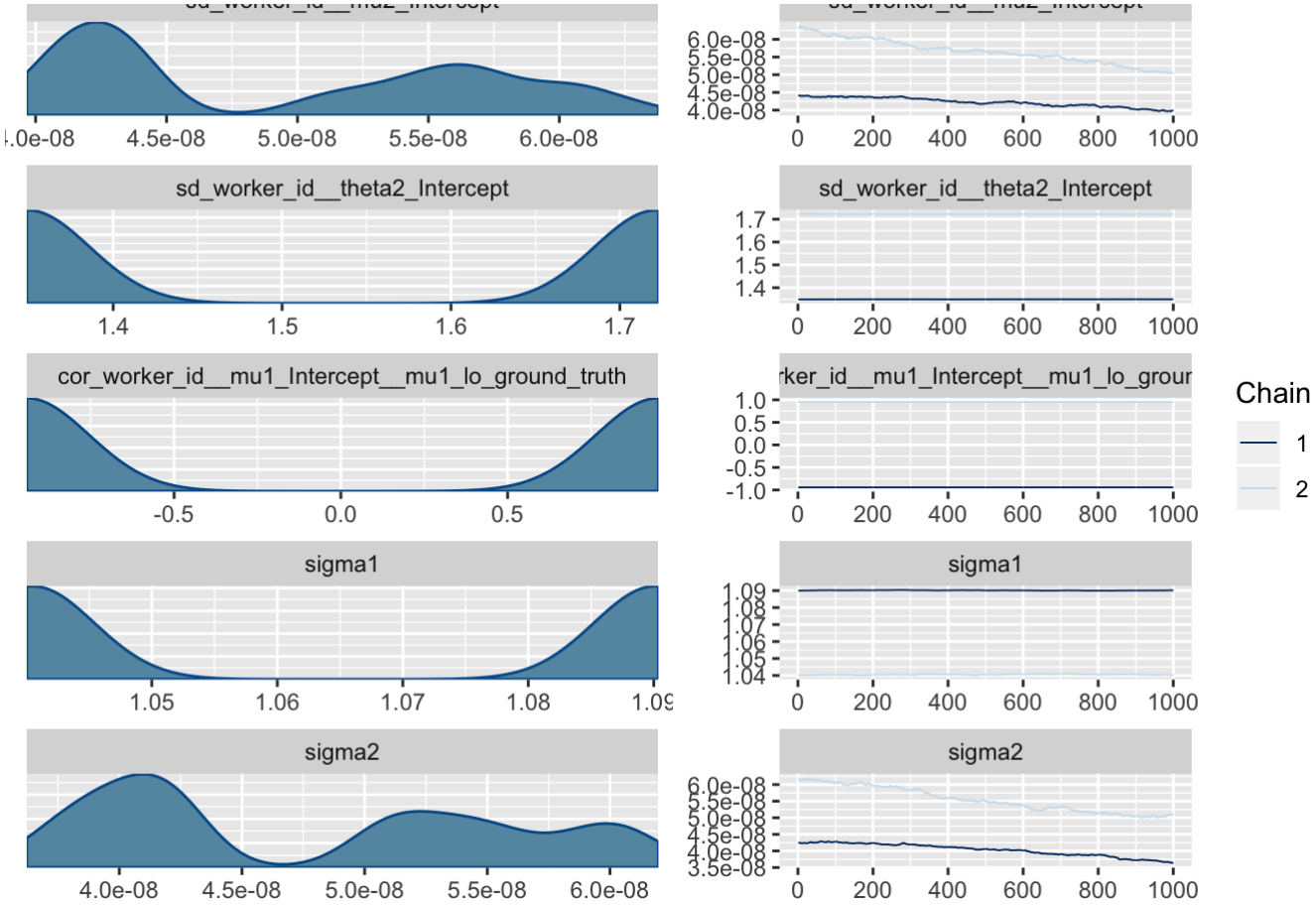
```
m.vis.wrkr.llo_mix <- brm(
  bf(lo_p_sup ~ 1,
    mu1 ~ (1 + lo_ground_truth|worker_id) + lo_ground_truth:condition, # llo model
    mu2 ~ (1|worker_id), # random constant response per worker (to account for people wh
o always answer the same, often but not always 50%)
    theta2 ~ (1|worker_id) + condition # the proportion of responses that are constant
  ),
  data = model_df_llo,
  family = mixture(gaussian, gaussian, order = 'mu'),
  prior = c(
    prior(normal(0, 1), class = Intercept, dpar = mu1),
    prior(normal(0, 1), class = Intercept, dpar = mu2)
  ),
  inits = 1, chains = 2, cores = 2,
  control = list(adapt_delta = 0.999, max_treedepth=15),
  file = "model-fits/llo_mix_mdl_vis_wrkr"
)
```

Check diagnostics:

- Trace plots

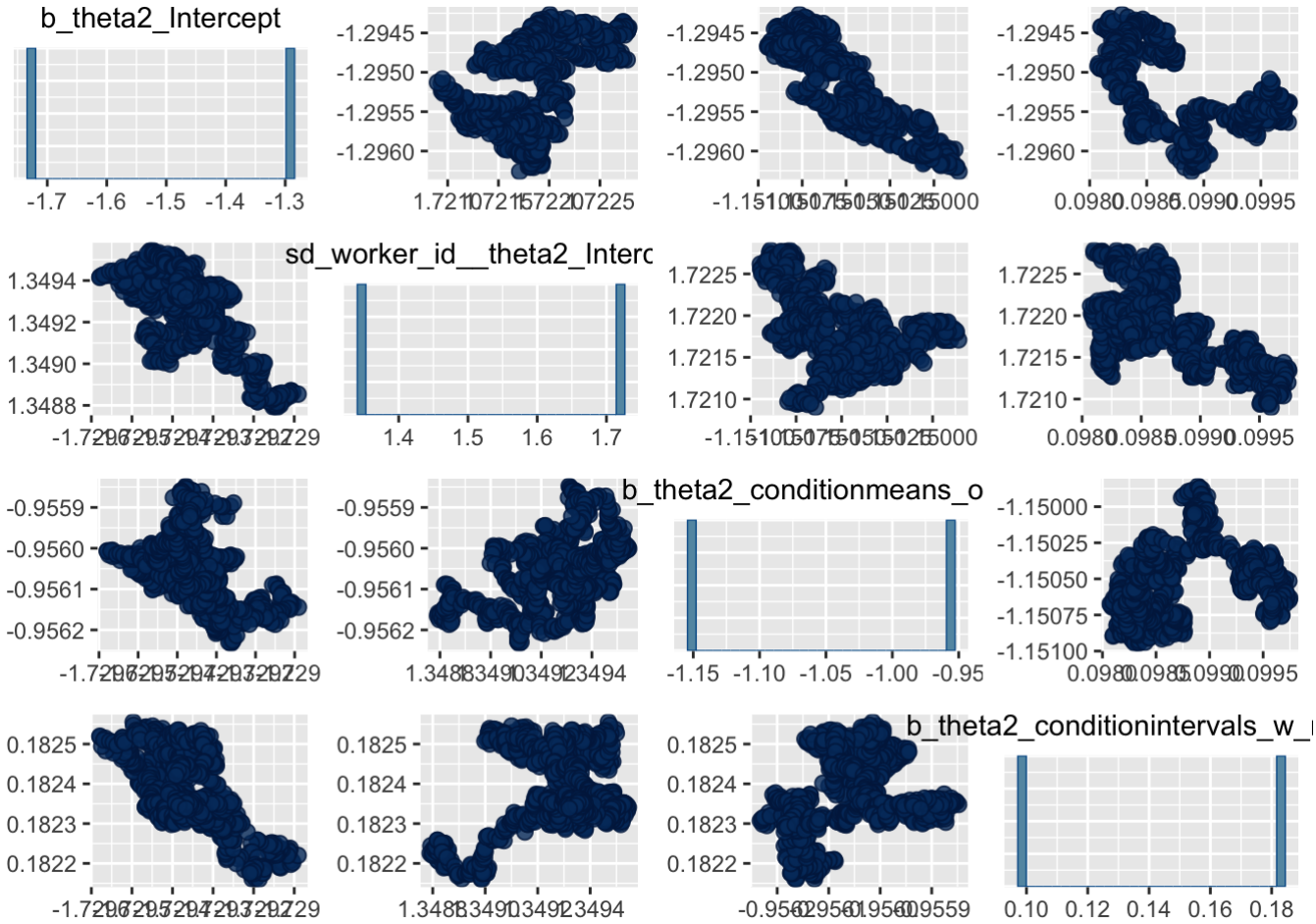
```
# trace plots
plot(m.vis.wrkr.llo_mix)
```


b_mu1_Intercept
b_mu2_Intercept
b_theta2_Intercept
b_mu1_lo_ground_truth:conditionHOPs
b_mu1_lo_ground_truth:conditionintervals_w_means
b_mu1_lo_ground_truth:conditionmeans_only
b_theta2_conditionintervals_w_means
b_theta2_conditionmeans_only
sd_worker_id__mu1_Intercept
sd_worker_id__mu1_lo_ground_truth
Chain
1
2

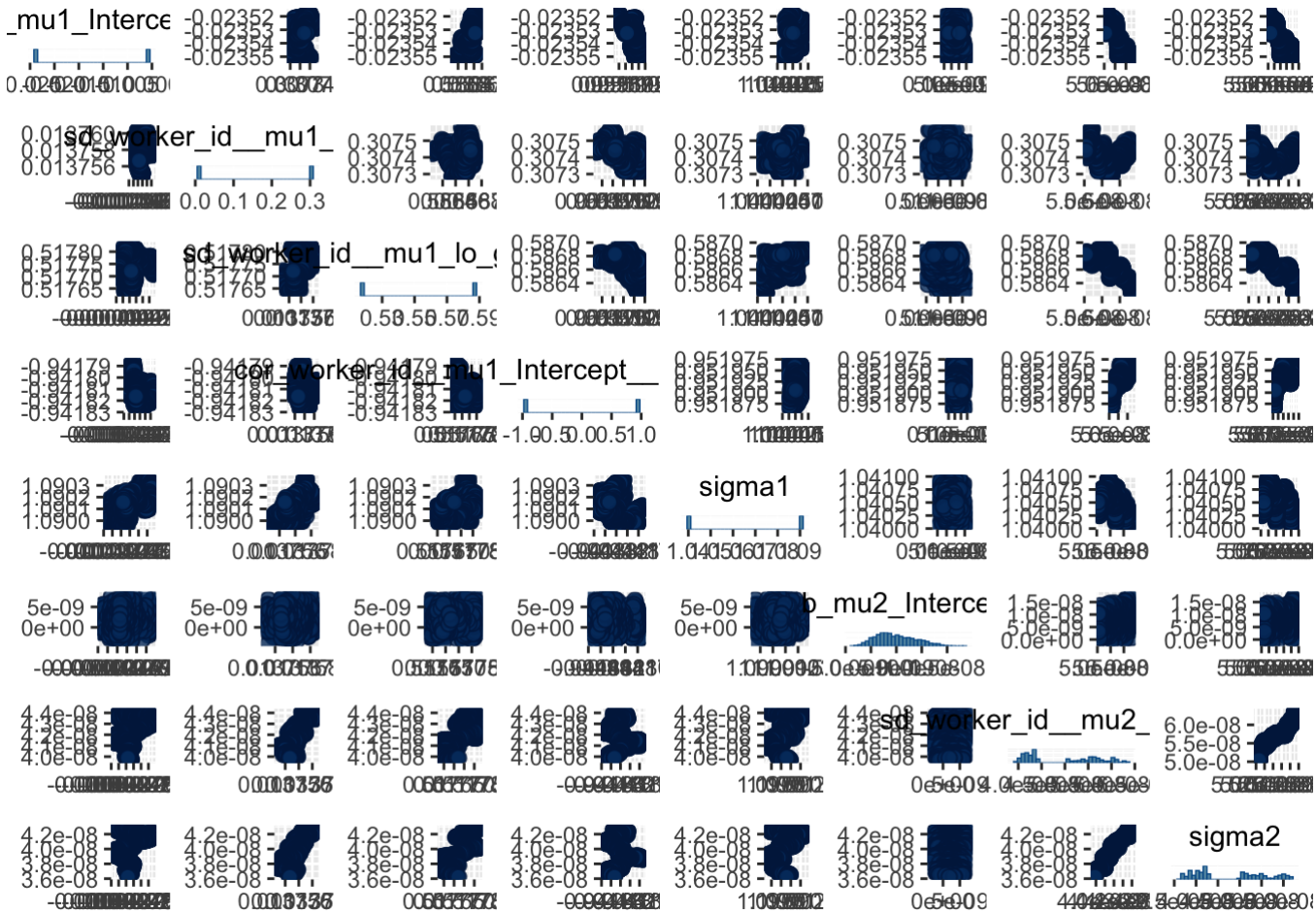


- Pairs plot

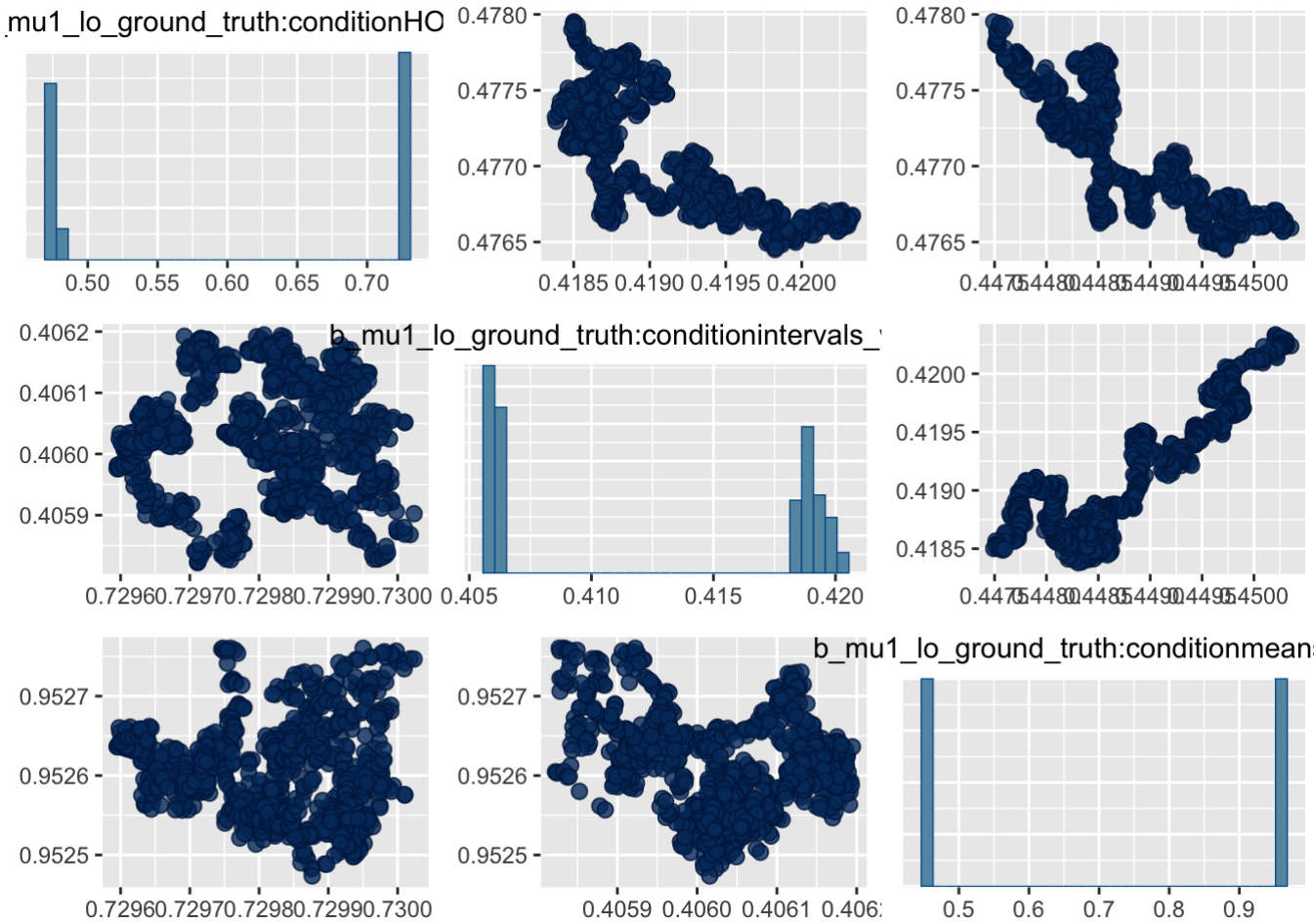
```
# pairs plot (too many things to view at once, so we've grouped them)
# mixture proportions
pairs(m.vis.wrkr.llo_mix, pars = c("b_theta2_Intercept",
                                   "sd_worker_id__theta2_Intercept",
                                   "b_theta2_conditionHOPs", # same as Intercept for this mo
del because of dummy coding
                                   "b_theta2_conditionmeans_only",
                                   "b_theta2_conditionintervals_w_means"))
```




```
# pairs plot (too many things to view at once, so we've grouped them)
# hyperparameters
pairs(m.vis.wrkr.llo_mix, pars = c("b_mu1_Intercept",
                                   "sd_worker_id__mu1_Intercept",
                                   "sd_worker_id__mu1_lo_ground_truth",
                                   "cor_worker_id__mu1_Intercept__mu1_lo_ground_truth",
                                   "sigma1",
                                   "b_mu2_Intercept",
                                   "sd_worker_id__mu2_Intercept",
                                   "sigma2"))
```




```
# pairs plot (too many things to view at once, so we've grouped them)
# slope effects
pairs(m.vis.wrkr.llo_mix, pars = c("b_mu1_lo_ground_truth:conditionHOPs",
                                   "b_mu1_lo_ground_truth:conditionintervals_w_means",
                                   "b_mu1_lo_ground_truth:conditionmeans_only"))
```



- Summary

```
# model summary
print(m.vis.wrkr.llo_mix)
```

```
## Warning: The model has not converged (some Rhats are > 1.1). Do not analyse the resul
ts!
## We recommend running more iterations and/or setting stronger priors.
```

```
##  Family: mixture(gaussian, gaussian)
##   Links: mu1 = identity; sigma1 = identity; mu2 = identity; sigma2 = identity; theta1
= identity; theta2 = identity
## Formula: lo_p_sup ~ 1
##          mu1 ~ (1 + lo_ground_truth | worker_id) + lo_ground_truth:condition
##          mu2 ~ (1 | worker_id)
##          theta2 ~ (1 | worker_id) + condition
##    Data: model_df_llo (Number of observations: 1568)
## Samples: 2 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 2000
##
## Group-Level Effects:
## ~worker_id (Number of levels: 56)
##                                     Estimate Est.Error l-95% CI
## sd(mu1_Intercept)                       0.16      0.15     0.01
## sd(mu1_lo_ground_truth)                 0.55      0.03     0.52
## sd(mu2_Intercept)                       0.00      0.00     0.00
## sd(theta2_Intercept)                    1.54      0.19     1.35
## cor(mu1_Intercept,mu1_lo_ground_truth)  0.01      0.95    -0.94
##                                     u-95% CI Eff.Sample     Rhat
## sd(mu1_Intercept)                       0.31          1  4754.17
## sd(mu1_lo_ground_truth)                 0.59          1   540.89
## sd(mu2_Intercept)                       0.00          1     5.85
## sd(theta2_Intercept)                    1.72          1   883.06
## cor(mu1_Intercept,mu1_lo_ground_truth)  0.95          1 80113.43
##
## Population-Level Effects:
##                                              Estimate Est.Error l-95% CI
## mu1_Intercept                                   -0.01      0.01    -0.02
## mu2_Intercept                                    0.00      0.00    -0.00
## theta2_Intercept                                -1.51      0.22    -1.73
## mu1_lo_ground_truth:conditionHOPs                 0.60      0.13     0.48
## mu1_lo_ground_truth:conditionintervals_w_means    0.41      0.01     0.41
## mu1_lo_ground_truth:conditionmeans_only           0.70      0.25     0.45
## theta2_conditionintervals_w_means                 0.14      0.04     0.10
## theta2_conditionmeans_only                       -1.05      0.10    -1.15
##                                              u-95% CI Eff.Sample    Rhat
## mu1_Intercept                                   -0.00          1 4117.25
## mu2_Intercept                                    0.00          2    1.58
## theta2_Intercept                                -1.29          1 1238.10
## mu1_lo_ground_truth:conditionHOPs                 0.73          1 1028.05
## mu1_lo_ground_truth:conditionintervals_w_means    0.42          1   31.50
## mu1_lo_ground_truth:conditionmeans_only           0.95          1  978.04
## theta2_conditionintervals_w_means                 0.18          1  222.35
## theta2_conditionmeans_only                       -0.96          1  810.87
##
## Family Specific Parameters:
##        Estimate Est.Error l-95% CI u-95% CI Eff.Sample   Rhat
## sigma1     1.07      0.02     1.04     1.09          1 214.46
## sigma2     0.00      0.00     0.00     0.00          1   6.30
##
## Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
```

```
## is a crude measure of effective sample size, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

Something clearly went horribly wrong when fitting this model. In past pilots, it has helped to reparameterize the model in order to represent the correlation between the three parameters representing rates of the constant response strategy in each visualization condition (e.g., `b_theta2_conditionmeans_only`). We do this by adding a multivariate normal prior.

Revised Mixture of the LLO Model and a Random Constant Response

This is an adaptation of the previous model which attempts to remedy the issues we saw in the pairs plots. What we do here is add a multivariate normal prior for the rate of random constant responses in each visualization condition. This allows the model to learn both the mean rate for each visualization condition `mu_theta2` (in log odds units) and the shared covariance matrix for the rates in each condition `Sigma_theta2`.

```
# define stanvars for multi_normal prior on condition effects
stanvars <- stanvar(rep(1, 3), "mu_theta2", scode = "  vector[3] mu_theta2;") +
  stanvar(diag(3), "Sigma_theta2", scode = "  matrix[3, 3] Sigma_theta2;")

# fit the model
m.vis.wrkr.llo_mix2 <- brm(
   bf(lo_p_sup ~ 1,
    mu1 ~ (1 + lo_ground_truth|worker_id) + lo_ground_truth:condition, # our most recent
llo model
    mu2 ~ (1|worker_id), # random constant response per worker (to account for people wh
o always answer the same, often but not always 50%)
    theta2 ~ (1|worker_id) + 0 + condition # the proportion of responses that are consta
nt
  ),
  data = model_df_llo,
  family = mixture(gaussian, gaussian, order = 'mu'),
  prior = c(
    prior(normal(0, 1), class = Intercept, dpar = mu1),
    prior(normal(0, 1), class = Intercept, dpar = mu2),
    prior("multi_normal(mu_theta2, Sigma_theta2)", class = b, dpar = theta2)
  ),
  stanvars = stanvars,
  inits = 1, chains = 2, cores = 2,
  control = list(adapt_delta = 0.999, max_treedepth=15),
  file = "model-fits/llo_mix_mdl_vis_wrkr2"
)
```

This also takes forever to run, which suggests to me that it probably has the same problems. I think it that constraining the response scale was the problem here.