

# 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,
    company_value = companyValue,
    ground_truth = groundTruth,
    p_contract_new = pContractNew,
    p_contract_old = pContractOld,
    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
trials from responses dataframe, leave in full version
  mutate( # mutate to jitter probability of superiority away from boundaries
    p_superiority = ifelse(p_superiority == 0, 0.25, p_superiority), # avoid
    id responses equal to zero
    p_superiority = ifelse(p_superiority == 100, 99.75, p_superiority) # avoid
    d responses equal to one-hundred
  ) #>%
  # mutate( # mutate to rows where intervene == -1 for some reason
  #   intervene = if_else(intervene == -1,
  #     # repair
  #     if_else((payoff == (contract_value - 1) | payoff == (-contr
  act_value - 1) | payoff == -1),
  #       1, # payed for intervention
  #       0), # didn't pay for intervention
  #     # don't repair
  #     as.numeric(intervene) # hack to avoid type error
  #   )
  # )

head(responses_df)
```

```
## # A tibble: 6 x 27
##   worker_id batch condition baseline contract_value exchange
##   <chr>      <int> <chr>        <dbl>        <dbl>      <dbl>
## 1 a3ee04d1     13 means_on...     0.5         2.25     0.0480
## 2 a3ee04d1     13 means_on...     0.5         2.25     0.0480
## 3 a3ee04d1     13 means_on...     0.5         2.25     0.0480
## 4 a3ee04d1     13 means_on...     0.5         2.25     0.0480
## 5 a3ee04d1     13 means_on...     0.5         2.25     0.0480
## 6 a3ee04d1     13 means_on...     0.5         2.25     0.0480
## # ... with 21 more variables: start_gain_frame <chr>, total_bonus <dbl>,
## #   duration <dbl>, numeracy <int>, gender <chr>, age <chr>,
## #   education <chr>, chart_use <chr>, company_value <dbl>,
## #   ground_truth <dbl>, intervene <int>, outcome <chr>,
## #   p_contract_new <dbl>, p_contract_old <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 baseline as a factor rather than a numeric value.

```
# create data frame for model
model_df_ll0 <- 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),
    baseline=as.factor(baseline)
  )
```

# 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.

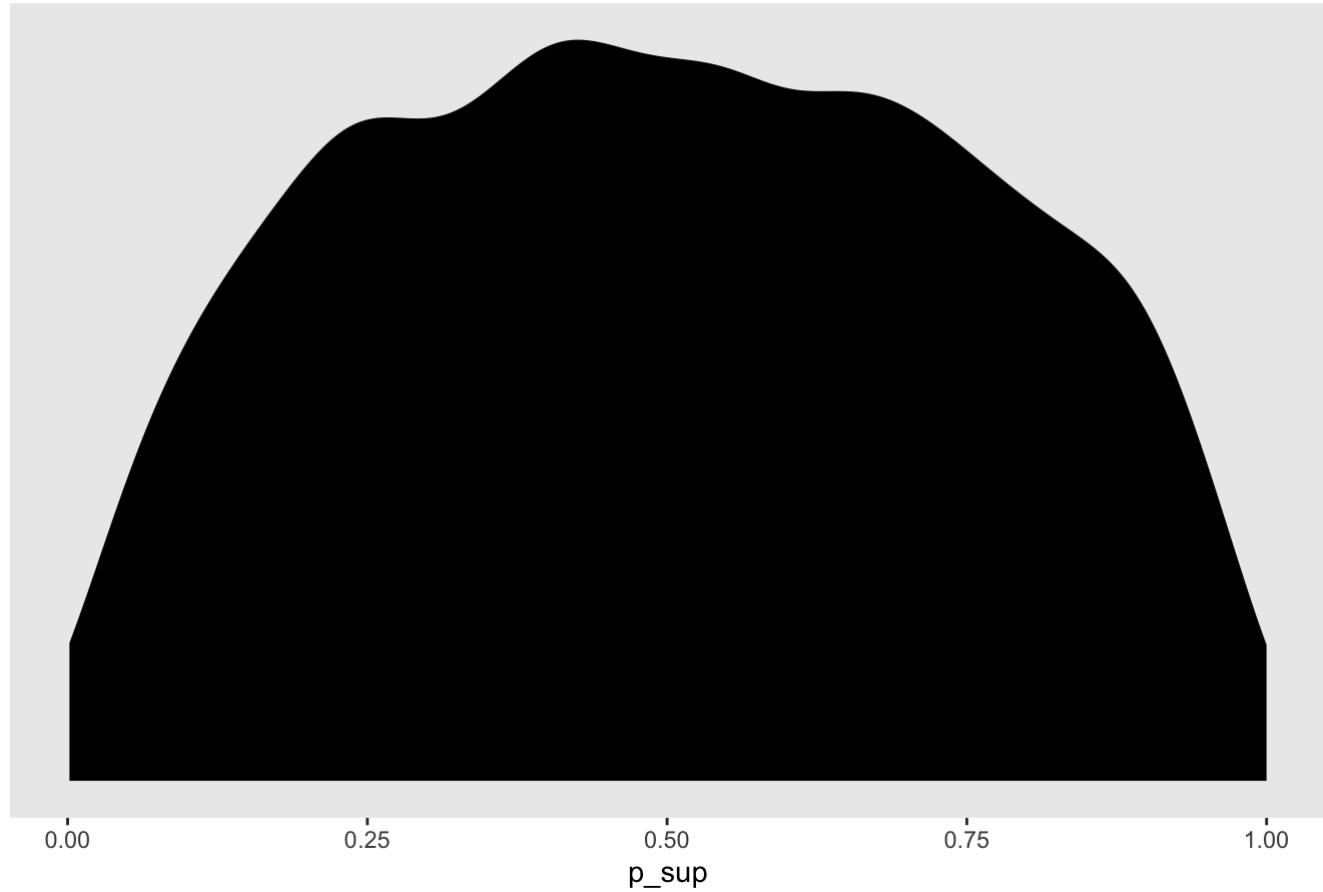
Let's check that our priors seem reasonable.

```
# 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 4983 rows containing non-finite values (stat_density).
```

### Prior predictive distribution for $h_i$



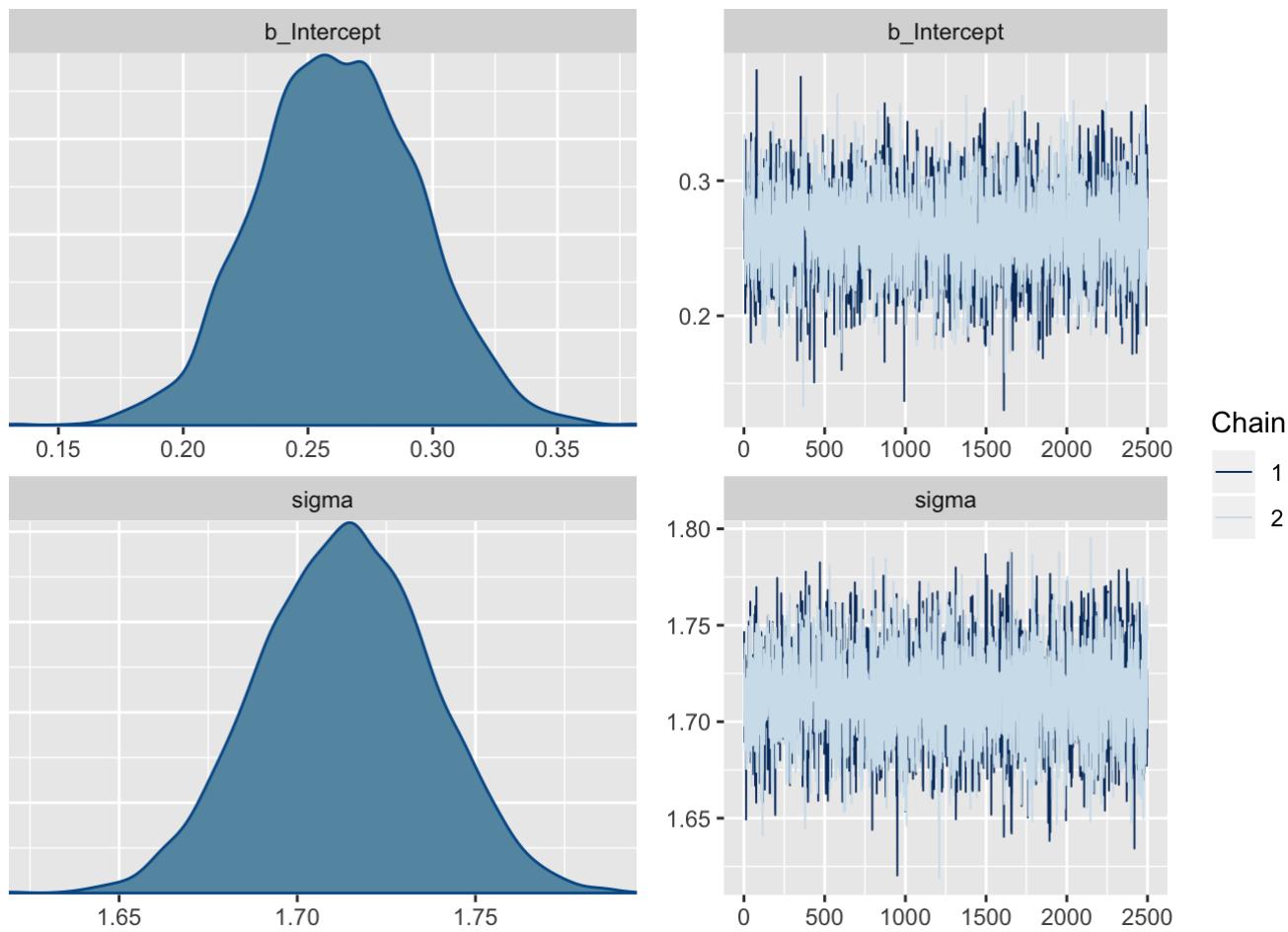
Fit an intercept model.

```
# starting as simple as possible: learn the distribution of lo_p_sup
m.lo_p_sup <- brm(data = model_df_lllo, 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:

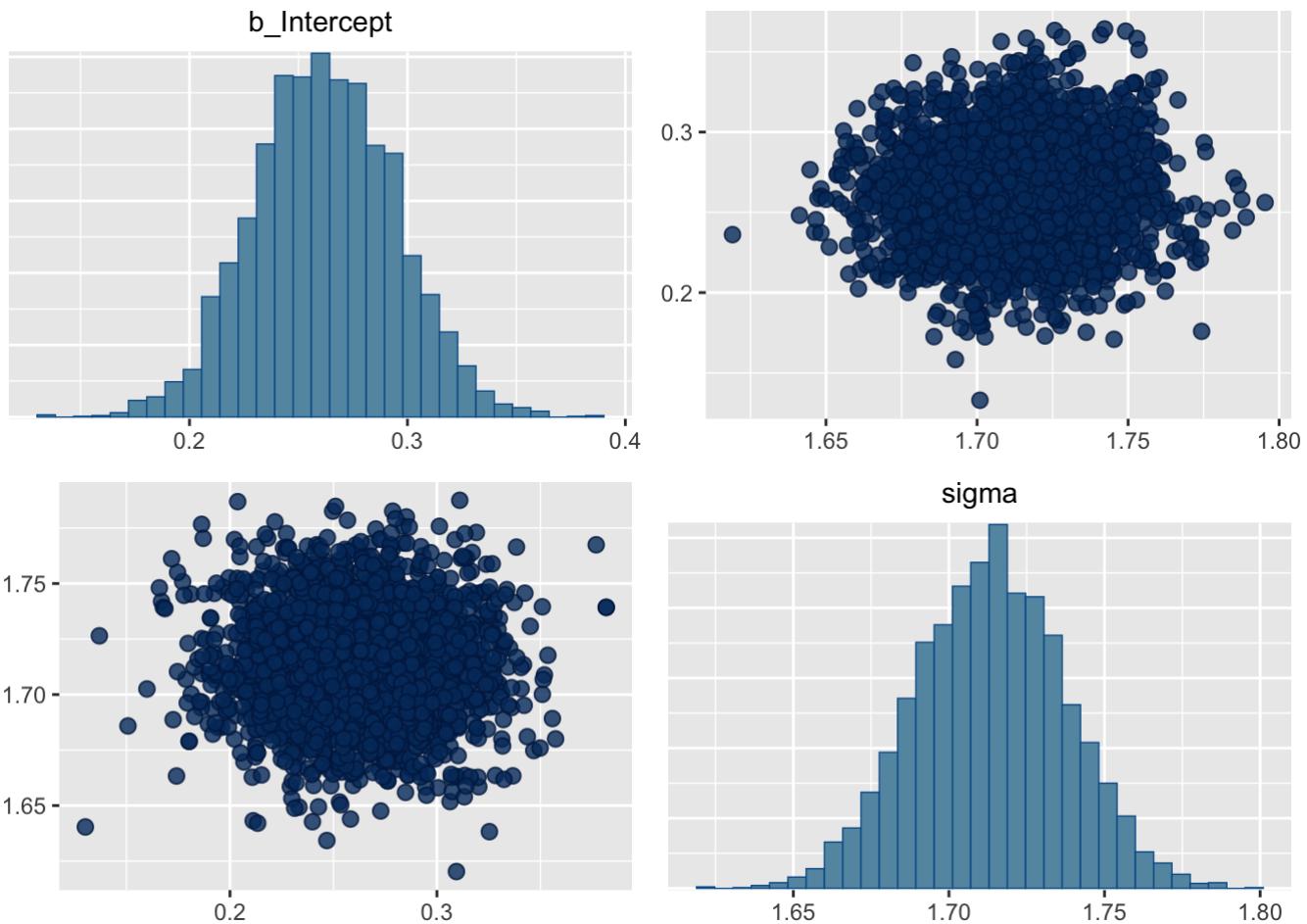
- Trace plots. The value of sigma seems pretty large.

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



- Pairs plot. These look great.

```
# 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: 2616)
## 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     0.26      0.03     0.20     0.33       4166  1.00
##
## Family Specific Parameters:
##             Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## sigma        1.71      0.02     1.67     1.76       4663  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 0.262  0.199  0.325  0.95  mean   qi
## 2 sigma        1.71   1.67   1.76   0.95  mean   qi

```

## Linear Log Odds Model of Probability of Superiority

Now we'll add in a slope parameter. Hopefully, this will reduce the residual variance.

Let's check that our priors seem reasonable.

```

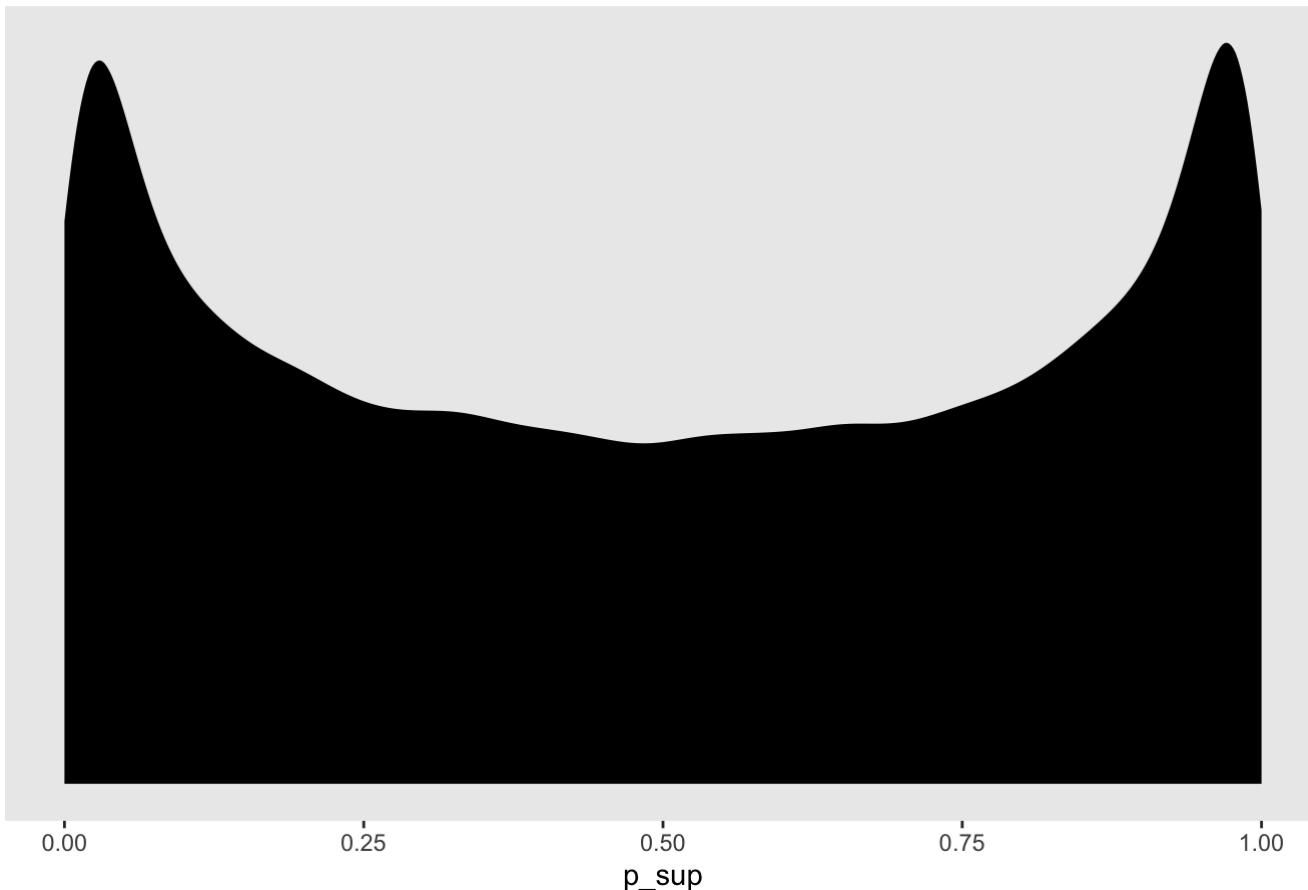
# prior predictive check
n <- 1e4
tibble(intercept      = rnorm(n, mean = 0, sd = 1),
       slope         = rnorm(n, mean = 0, sd = 1),
       sample_sigma = rnorm(n, mean = 0, sd = 1),
       lo_ground_truth = list(qlogis(unique(responses_df$ground_truth)))) %>%
  unnest() %>%
  mutate(lo_p_sup = rnorm(n * length(unique(responses_df$ground_truth))), mean = intercept + 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 119712 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.

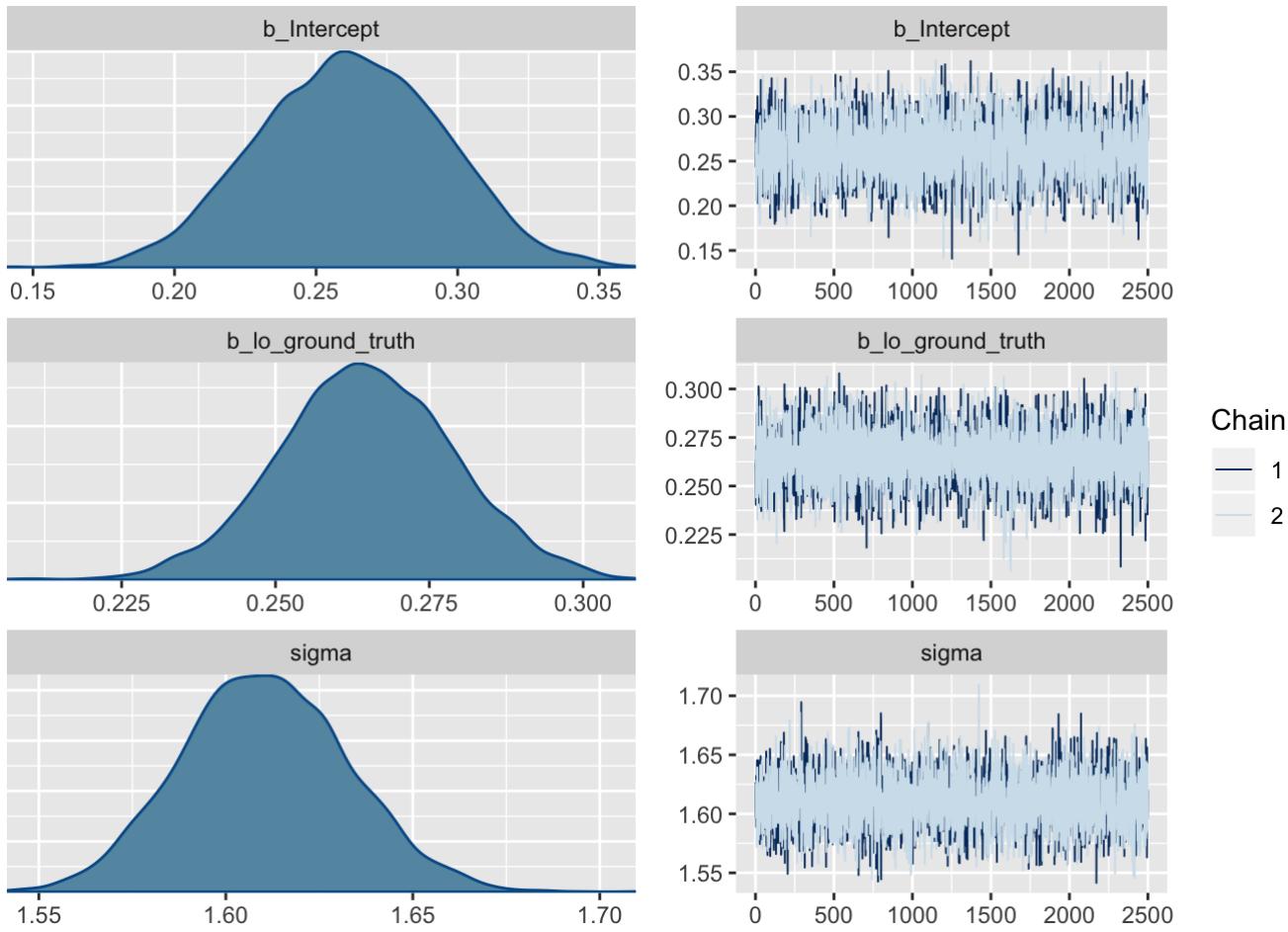
Now, let's fit the linear log odds (LLO) model.

```
# 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:

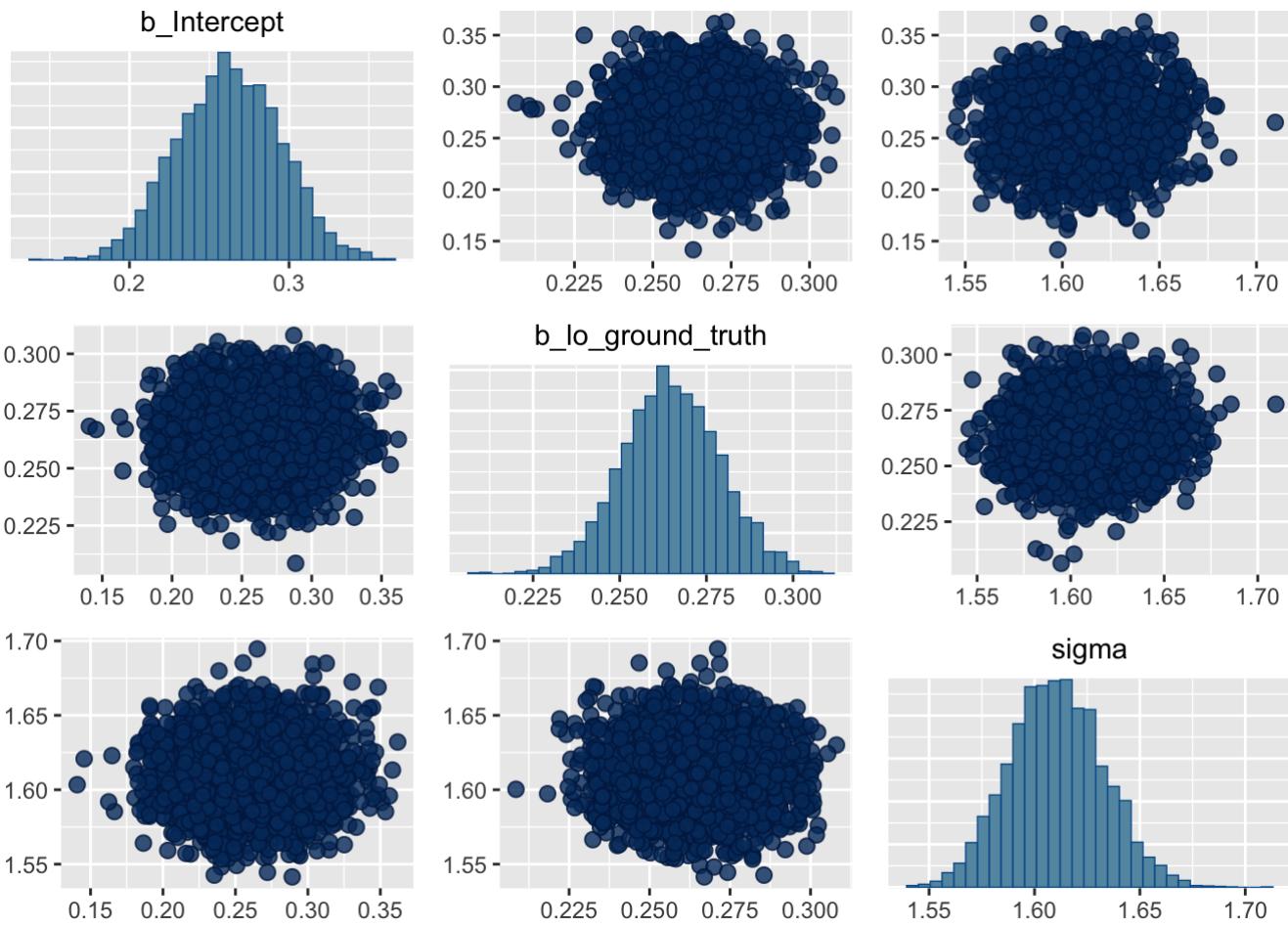
- Trace plots

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



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

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



- Summary

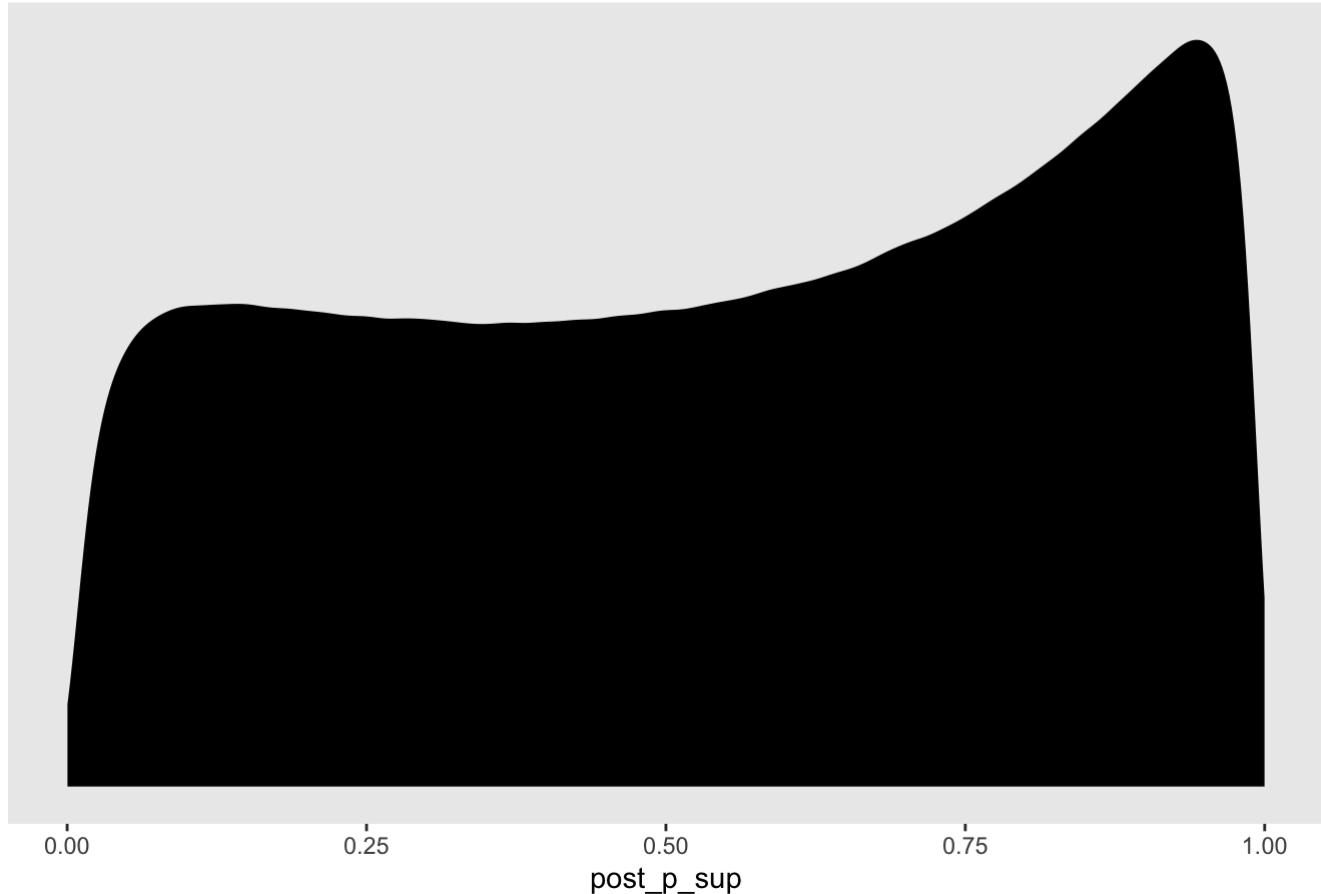
```
# model summary
print(m.llo_p_sup)
```

```
## Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: lo_p_sup ~ 1 + lo_ground_truth
## Data: model_df_llo (Number of observations: 2616)
## Samples: 2 chains, each with iter = 3000; warmup = 500; thin = 1;
##           total post-warmup samples = 5000
##
## Population-Level Effects:
##             Estimate Est.Error 1-95% CI u-95% CI Eff.Sample Rhat
## Intercept      0.26      0.03    0.20    0.32       4210  1.00
## lo_ground_truth 0.26      0.01    0.24    0.29       5457  1.00
##
## Family Specific Parameters:
##             Estimate Est.Error 1-95% CI u-95% CI Eff.Sample Rhat
## sigma        1.61      0.02    1.57    1.66       5075  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_lllo %>%
  select(lo_ground_truth) %>%
  add_predicted_draws(m.lllo_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())
```

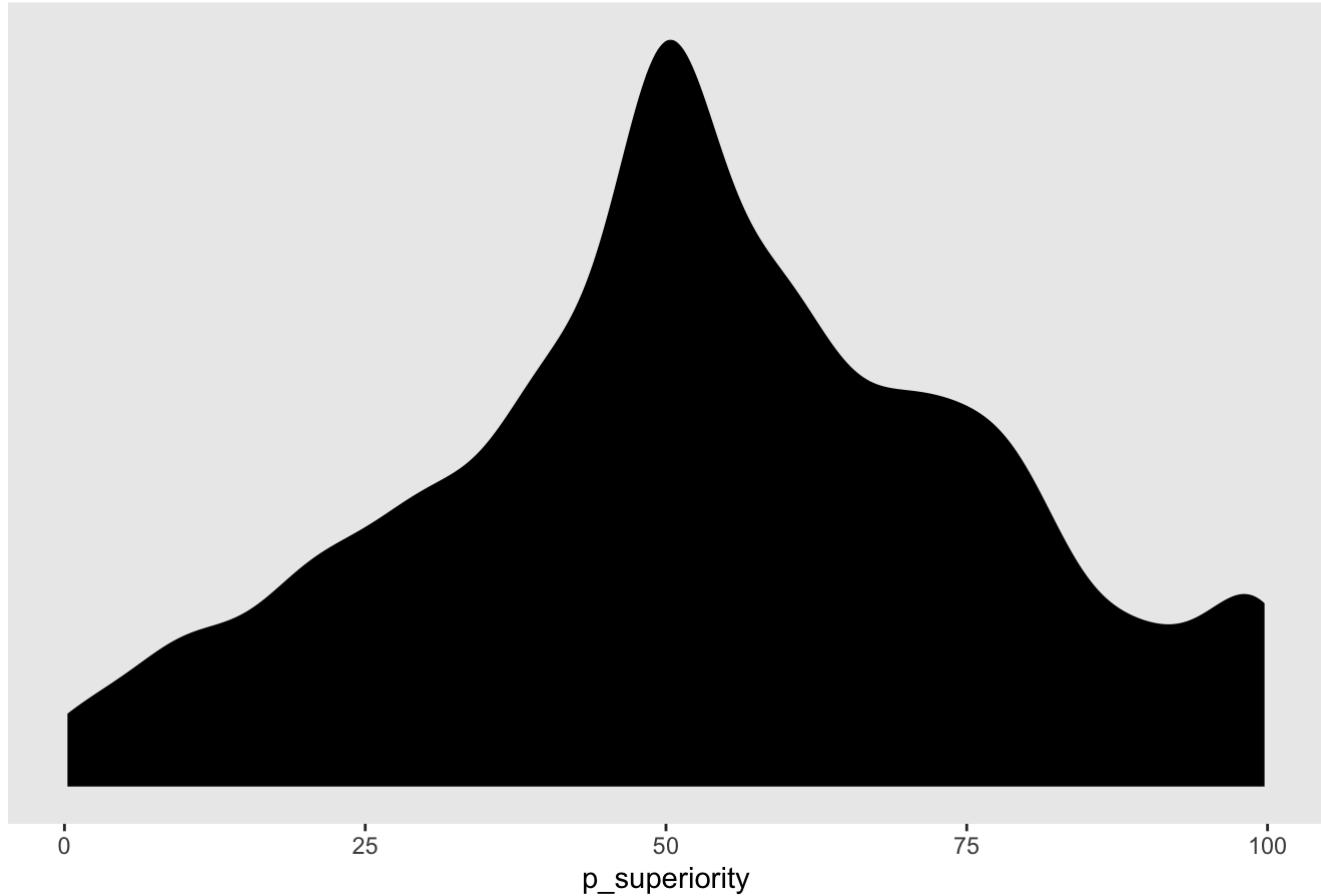
Posterior predictive distribution for probability of superiority



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_lllo %>%
  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

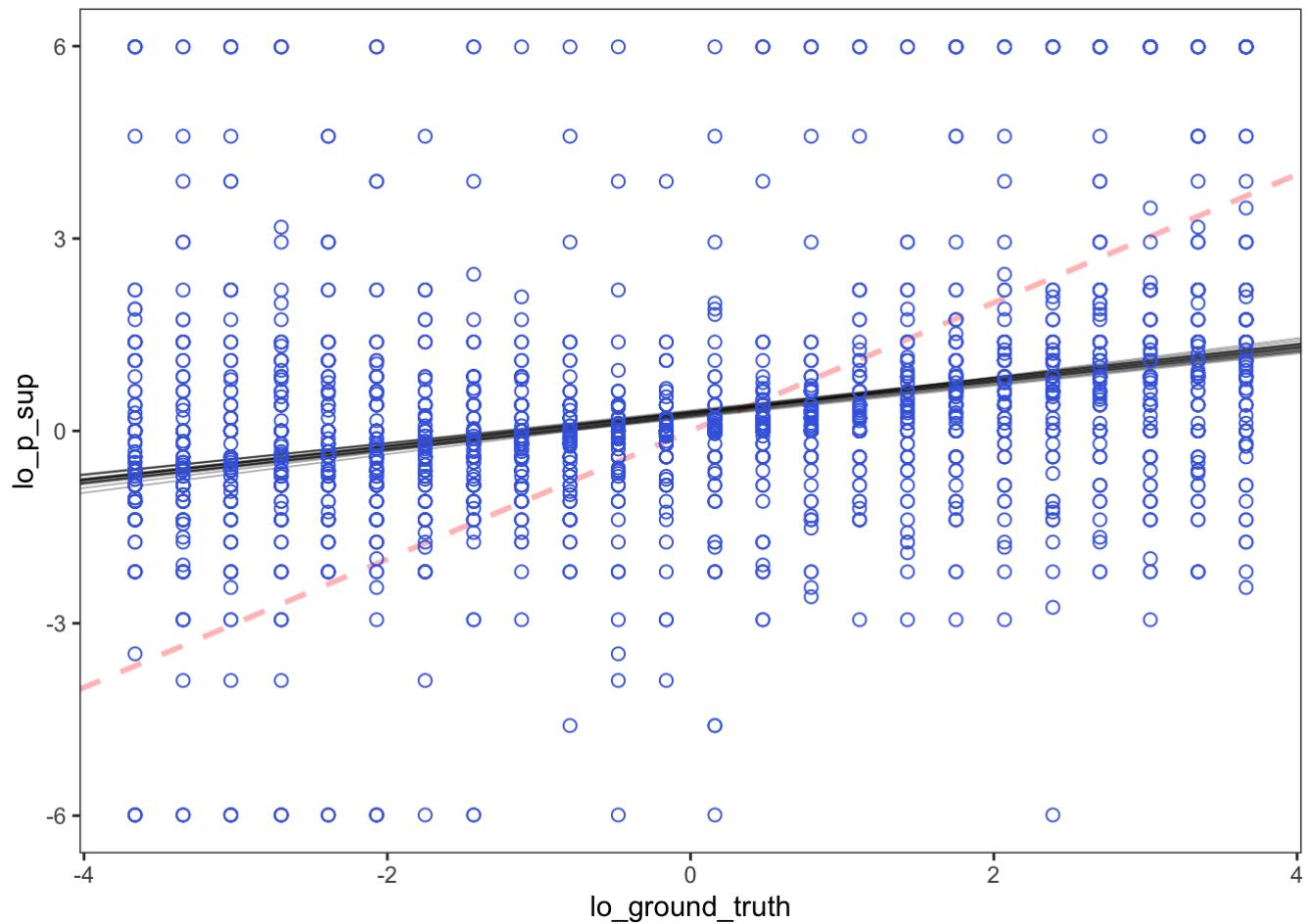


The peak in the center of the probability scale seems to suggest that the data generating process is a 50% inflated mixture. Also, note that we see the same skew in the observed data as in the posterior predictions, suggesting that the model has learned this from the data.

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

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

# plot estimated linear models against observed data
model_df_lllo %>%
  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_lllo$lo_ground_truth, c(0, 1)),
                  ylim = quantile(model_df_lllo$lo_p_sup, c(0, 1))) +
  theme_bw() +
  theme(panel.grid = element_blank())
```



## Add Different Linear Models per Visualization Condition

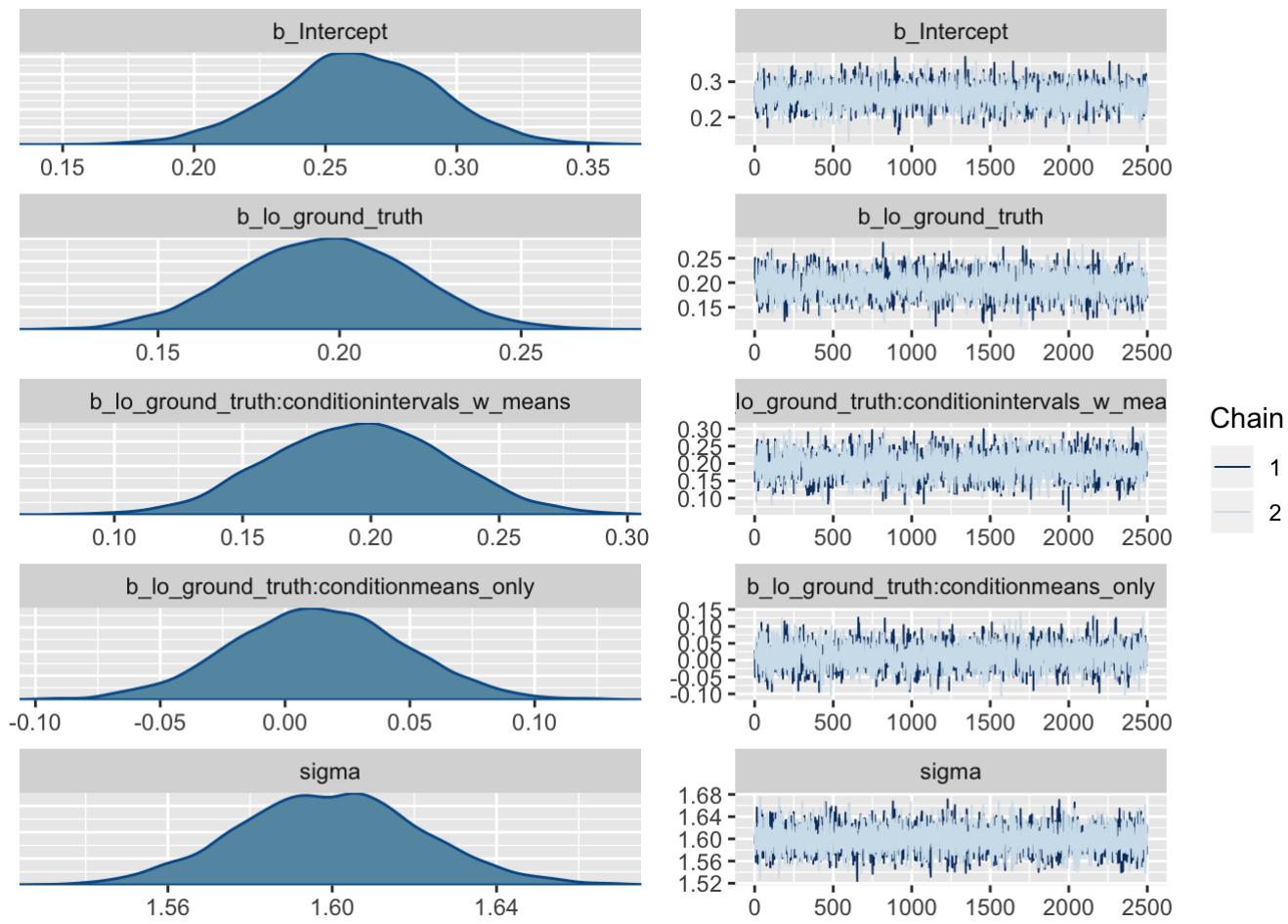
In the LLO framework, what we really want to know about is the impact of visualization condition on the slopes of linear models in log odds space. Do some visualizations lead to more extreme patterns of bias than others? To test this, we'll add an interaction between visualization condition and the ground 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:c
                           ondition,
                           newdata = model_df_llo,
                           file = "model-fits/llo_mdl_vis")
```

Check diagnostics:

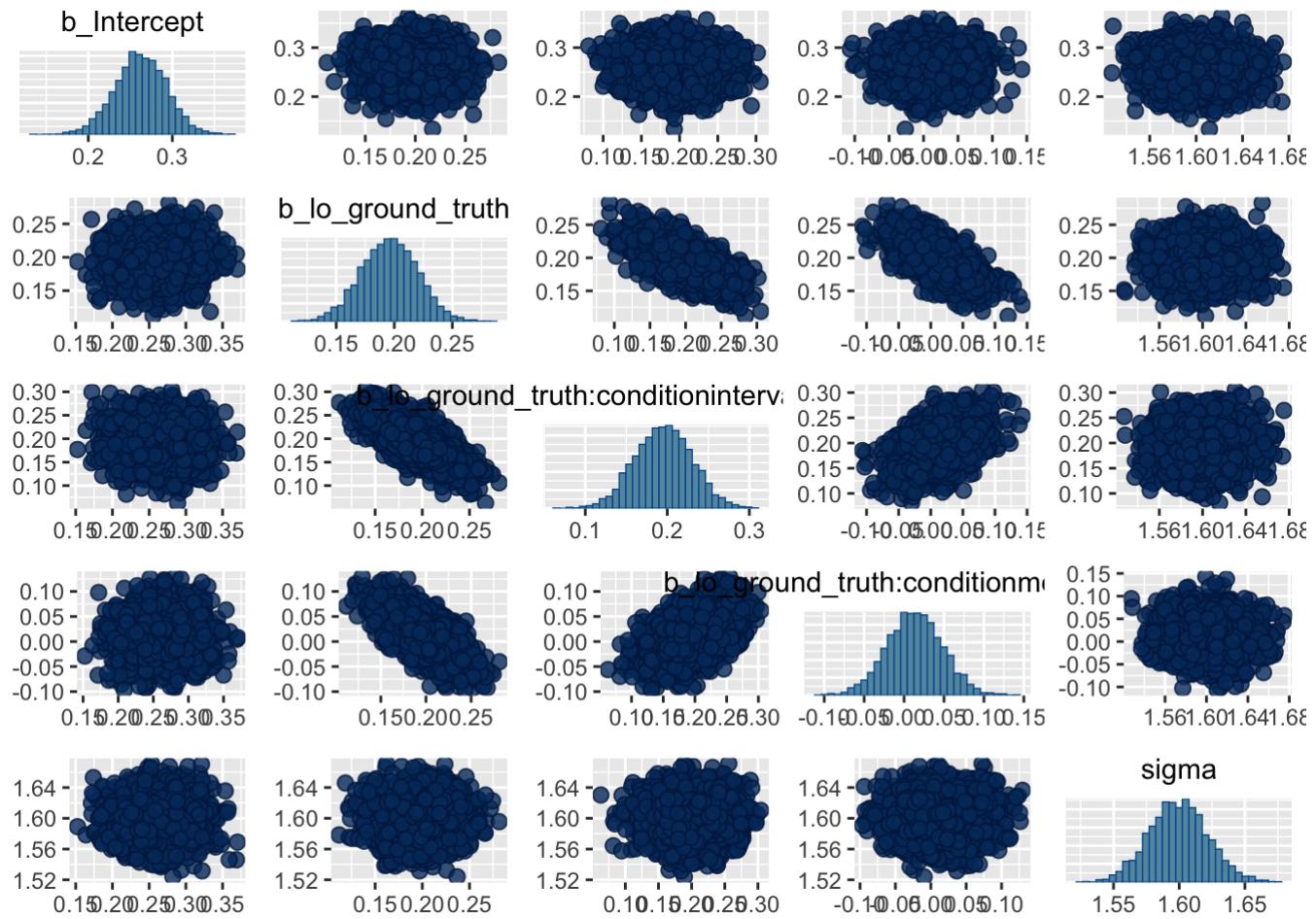
- Trace plots

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



- Pairs plot

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



- Summary

```
# model summary
print(m.vis.lllo_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: 2616)
## 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                  0.26      0.03    0.20      0.32      5360  1.00
## lo_ground_truth             0.20      0.02    0.15
## lo_ground_truth:conditionintervals_w_means 0.20      0.03    0.13
## lo_ground_truth:conditionmeans_only       0.01      0.03   -0.06
##                               u-95% CI Eff.Sample Rhat
## Intercept                  0.32      5360  1.00
## lo_ground_truth             0.24      2482  1.00
## lo_ground_truth:conditionintervals_w_means 0.26      2911  1.00
## lo_ground_truth:conditionmeans_only       0.08      2816  1.00
##
## Family Specific Parameters:
##           Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## sigma     1.60      0.02    1.56      1.64      5440  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).

```

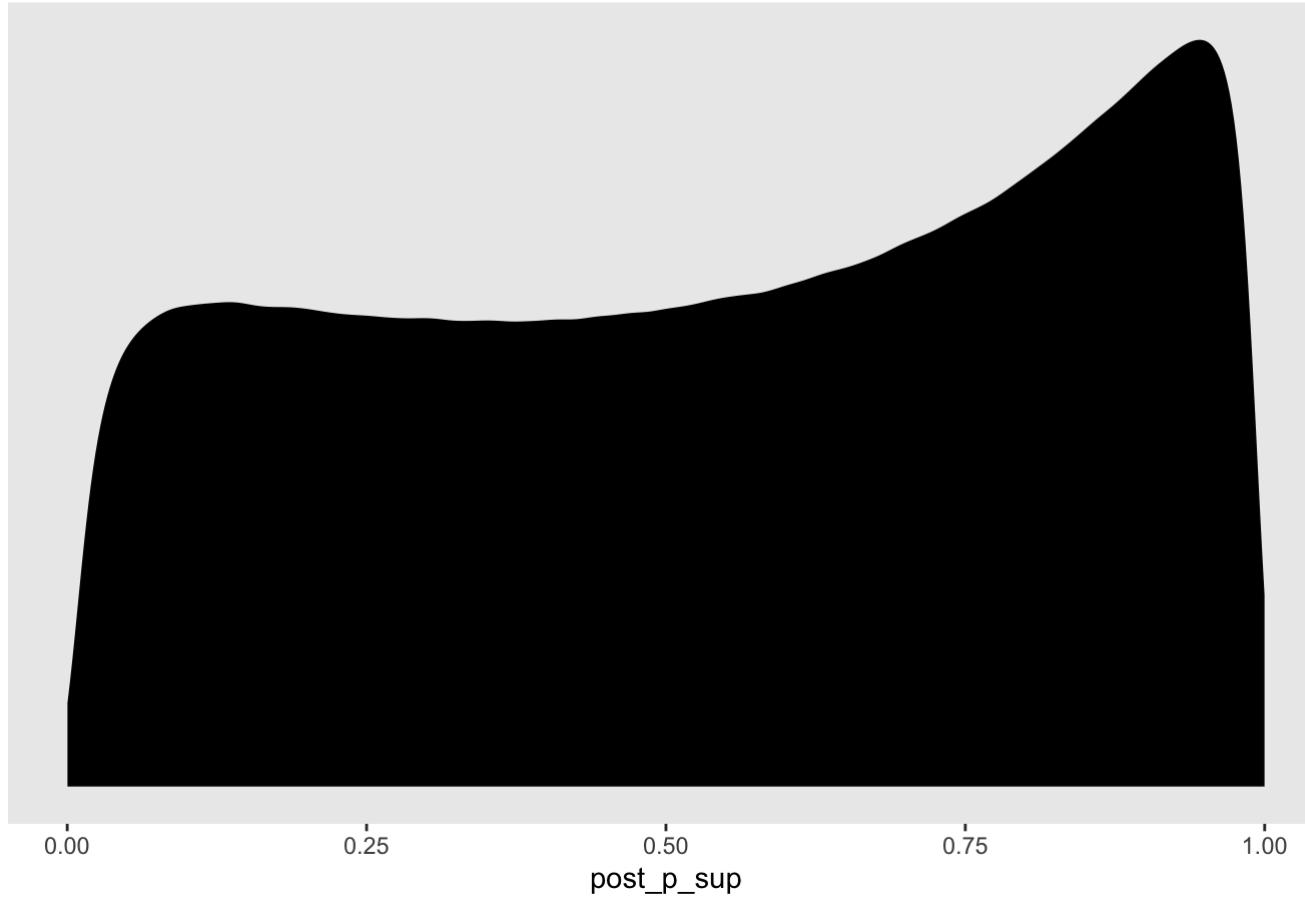
Let's check out a posterior predictive distribution for probability of superiority. This should look more like a mixture with different slopes, but it still looks really wide.

```

# 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())

```

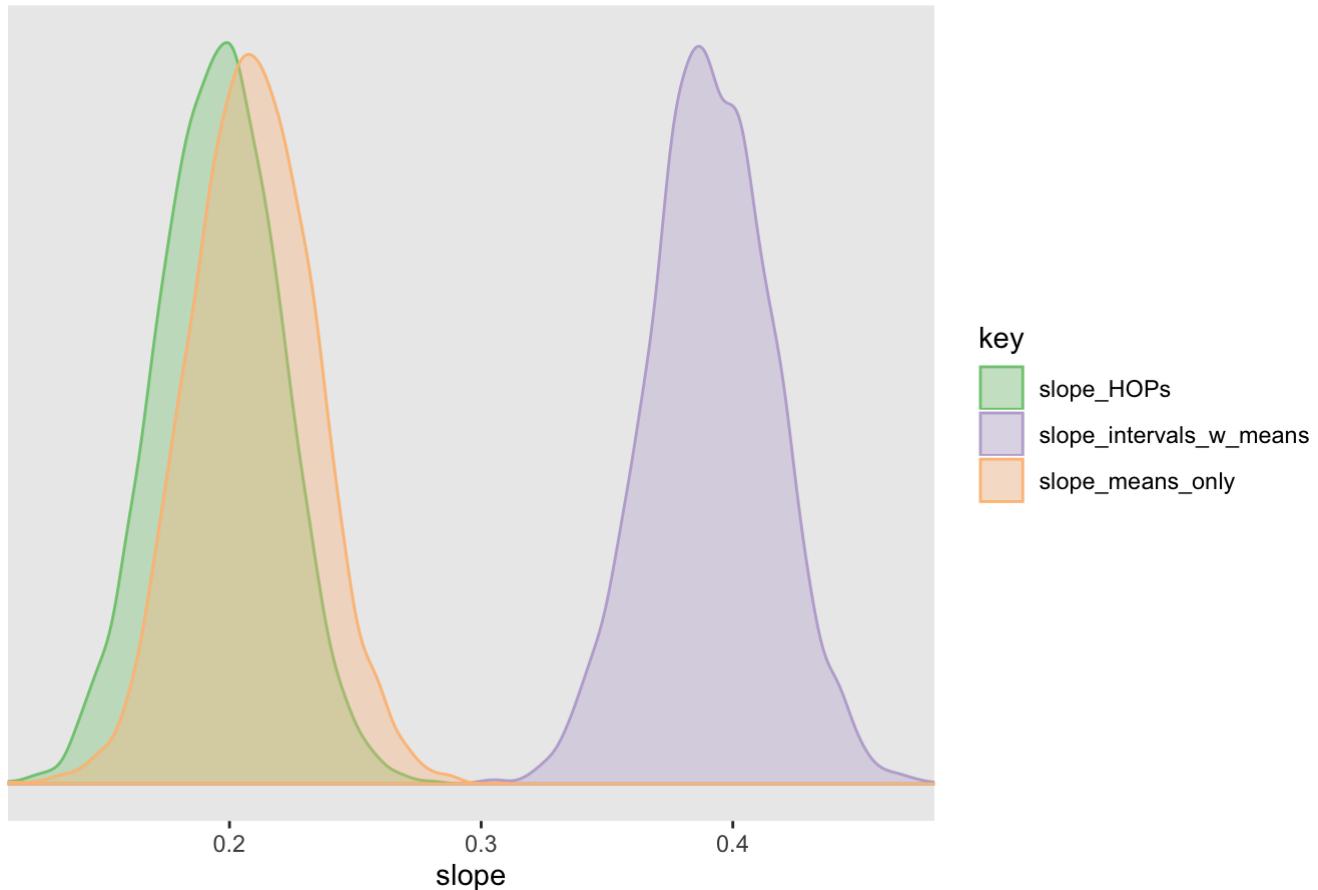
### Posterior predictive distribution for probability of superiority



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 condition
posterior_samples(m.vis.lllo_p_sup) %>%
  transmute(slope_HOPs = b_lo_ground_truth,
            slope_intervals_w_means = b_lo_ground_truth + `b_lo_ground_truth:condition`_nintervals_w_means`,
            slope_means_only = b_lo_ground_truth + `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



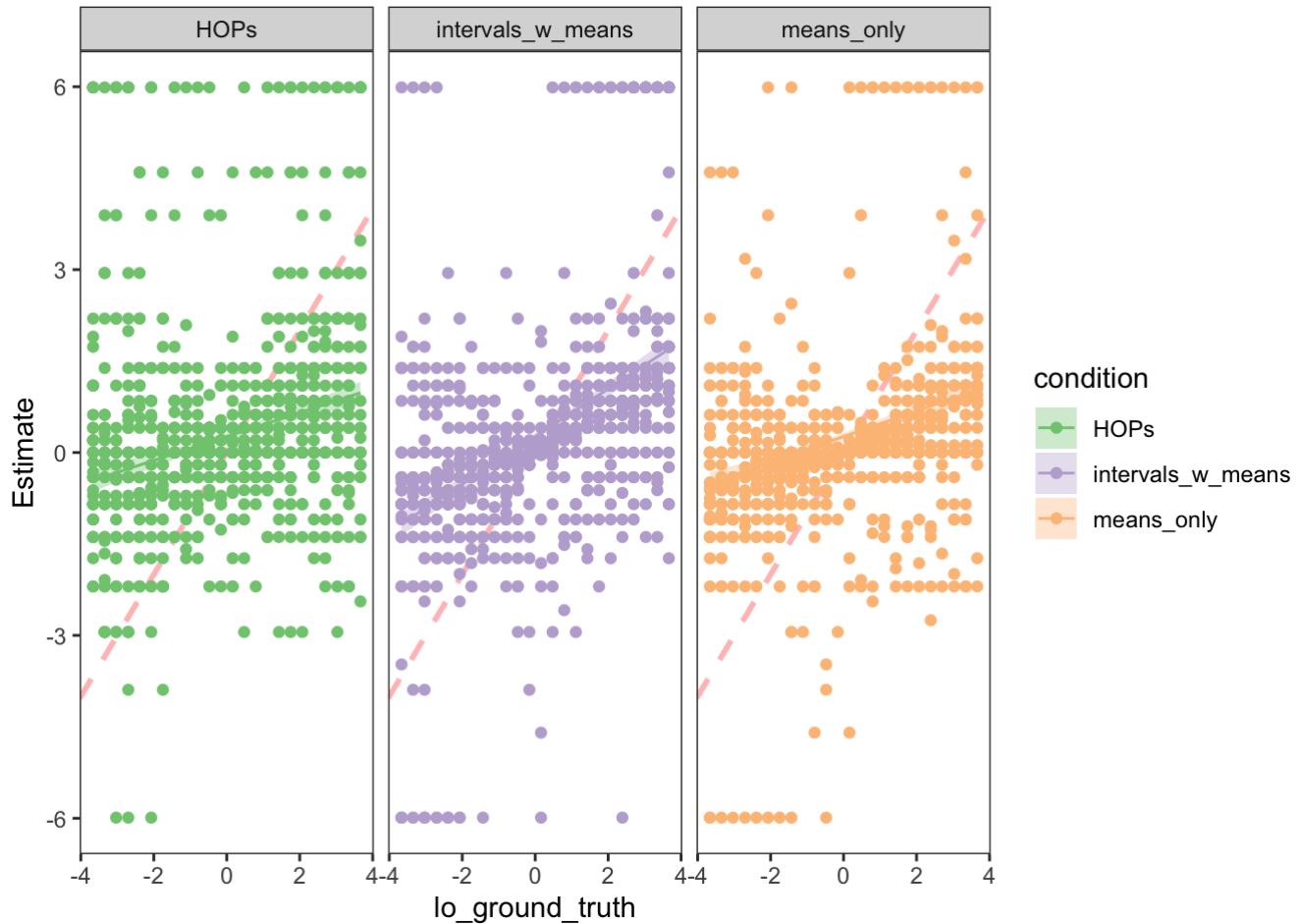
Recall that a slope of 1 reflects zero bias. This suggests that users are biased toward responses of 50% in all conditions.

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_lllo %>%
  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.lllo_p_sup,
    aes(ymin = Q2.5,
        ymax = Q97.5,
        fill = condition,
        group = condition),
    alpha = .35) +
  geom_line(data = fitd_m.vis.lllo_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_lllo$lo_ground_truth, c(0, 1)),
                  ylim = quantile(model_df_lllo$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 heterogeneity 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.

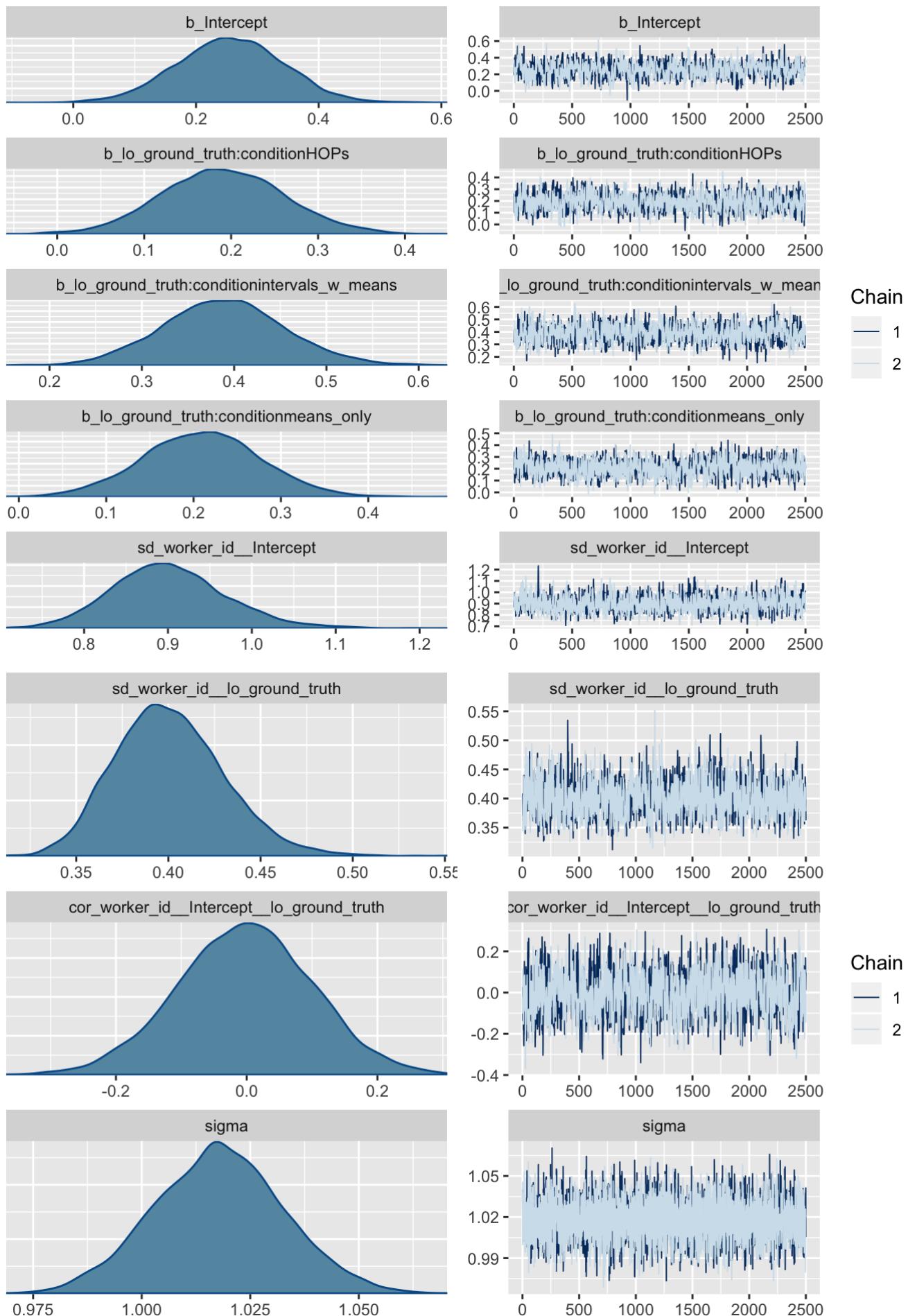
```
# update the llo model of p_sup responses to include an interaction
m.vis.wrkr.llo_p_sup <- update(m.llo_p_sup,
                                formula = lo_p_sup ~ (1 + lo_ground_truth|worker_id) + lo_gr
ound_truth:condition,
                                newdata = model_df_llo,
             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

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

Check diagnostics:

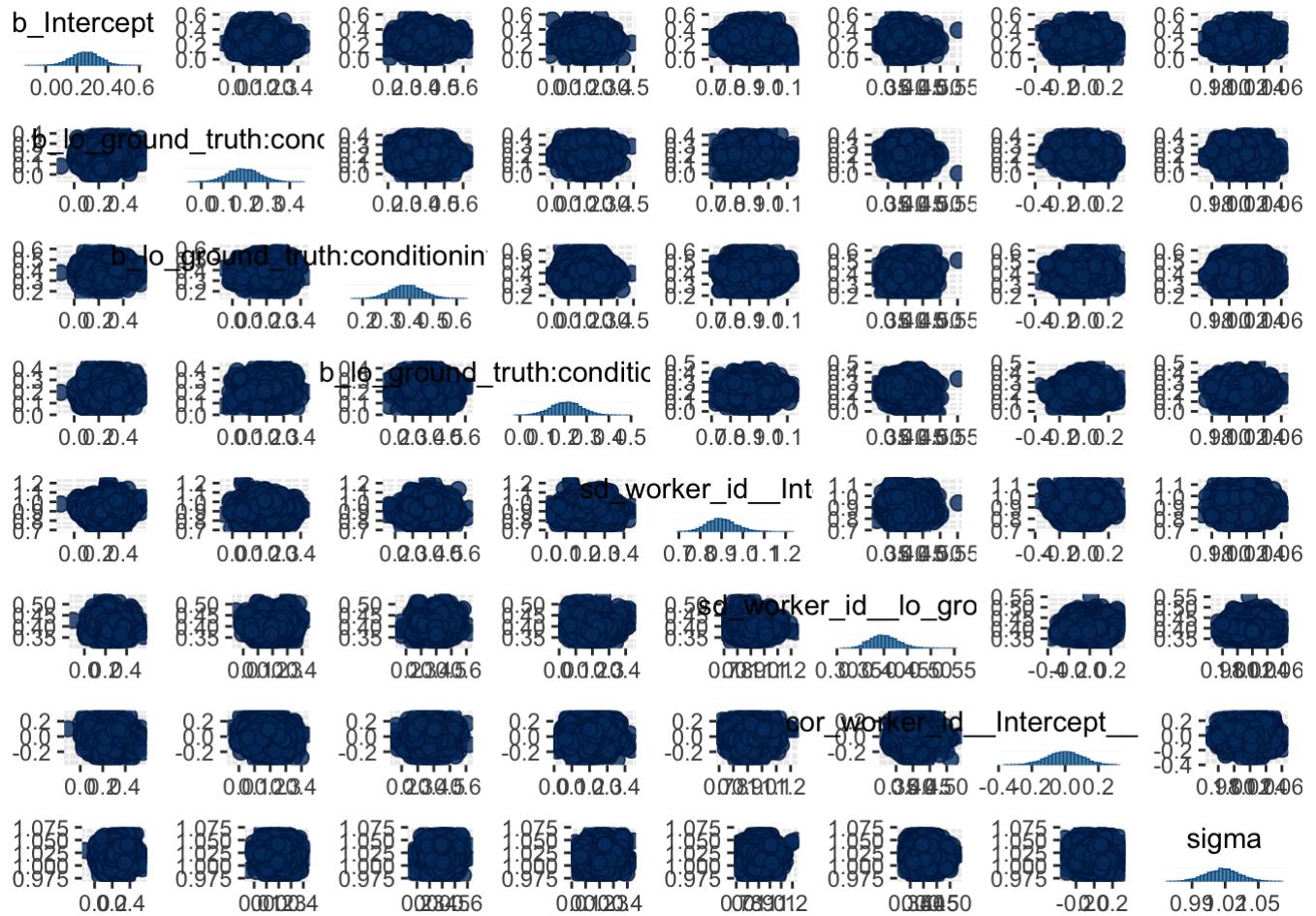
- Trace plots

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



- Pairs plot

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



- Summary

```
# model summary
print(m.vis.wrkr.lllo_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: 2616)
## Samples: 2 chains, each with iter = 3000; warmup = 500; thin = 1;
##          total post-warmup samples = 5000
##
## Group-Level Effects:
## ~worker_id (Number of levels: 109)
##                               Estimate Est.Error l-95% CI u-95% CI
## sd(Intercept)              0.90     0.07    0.78    1.04
## sd(lo_ground_truth)        0.40     0.03    0.35    0.46
## cor(Intercept,lo_ground_truth) -0.00    0.10   -0.20    0.20
##                               Eff.Sample Rhat
## sd(Intercept)             1008 1.00
## sd(lo_ground_truth)       1205 1.00
## cor(Intercept,lo_ground_truth) 1011 1.00
##
## Population-Level Effects:
##                               Estimate Est.Error l-95% CI
## Intercept                  0.26     0.09    0.09
## lo_ground_truth:conditionHOPs 0.19     0.07    0.06
## lo_ground_truth:conditionintervals_w_means 0.39     0.07    0.26
## lo_ground_truth:conditionmeans_only 0.21     0.07    0.08
##                               u-95% CI Eff.Sample Rhat
## Intercept                  0.43      676 1.00
## lo_ground_truth:conditionHOPs 0.33      846 1.00
## lo_ground_truth:conditionintervals_w_means 0.52      984 1.00
## lo_ground_truth:conditionmeans_only 0.34     1192 1.00
##
## Family Specific Parameters:
##           Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## sigma     1.02     0.01    0.99    1.05      6476 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).

```

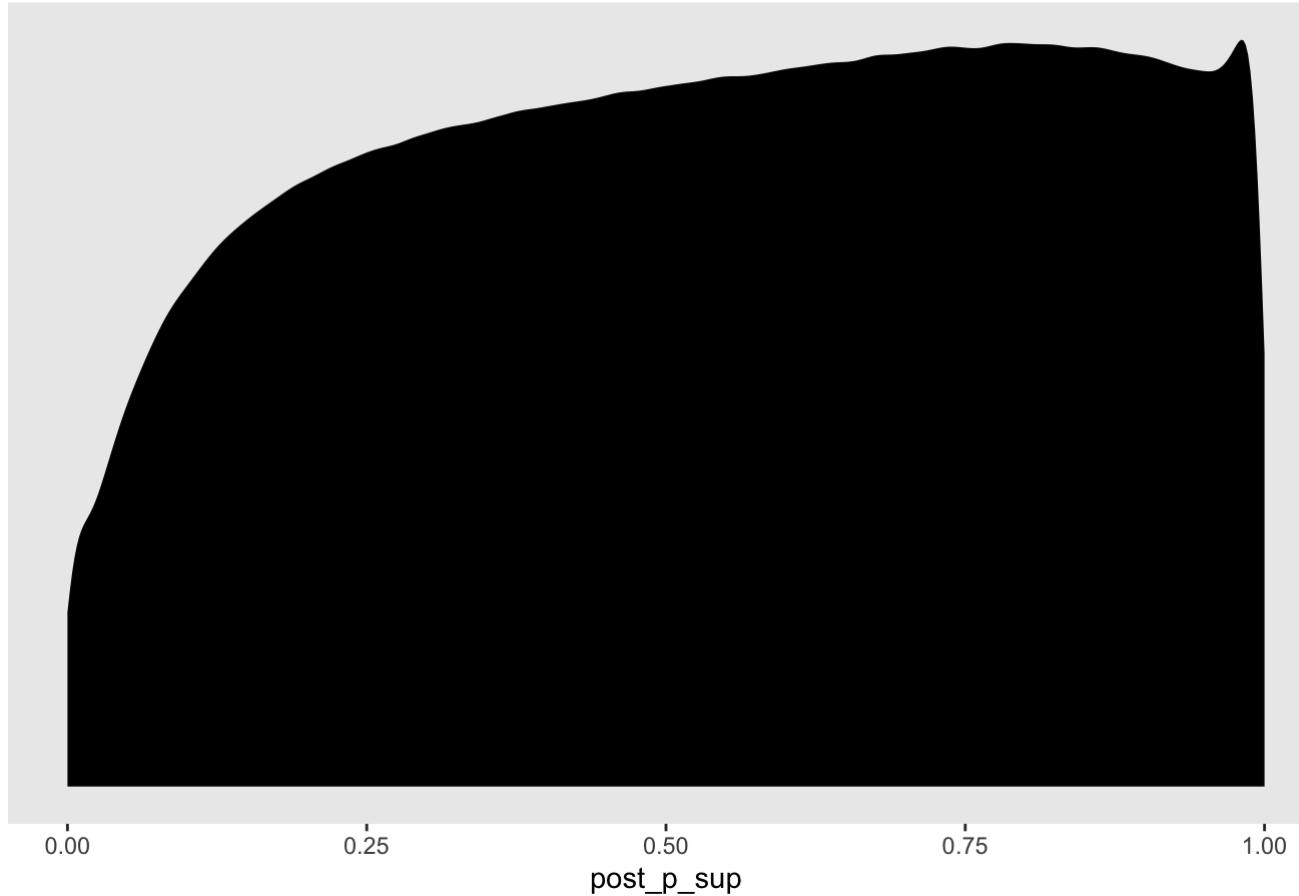
Let's check out a posterior predictive distribution for probability of superiority. This distribution is looking less spread toward the tails than in previous iterations of the model, which suggests that individual variation in slopes accounts for some (but not all) of the 50% responses.

```

# posterior predictive check
model_df_llo %>%
  select(lo_ground_truth, condition, worker_id) %>%
  add_predicted_draws(m.viz.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())

```

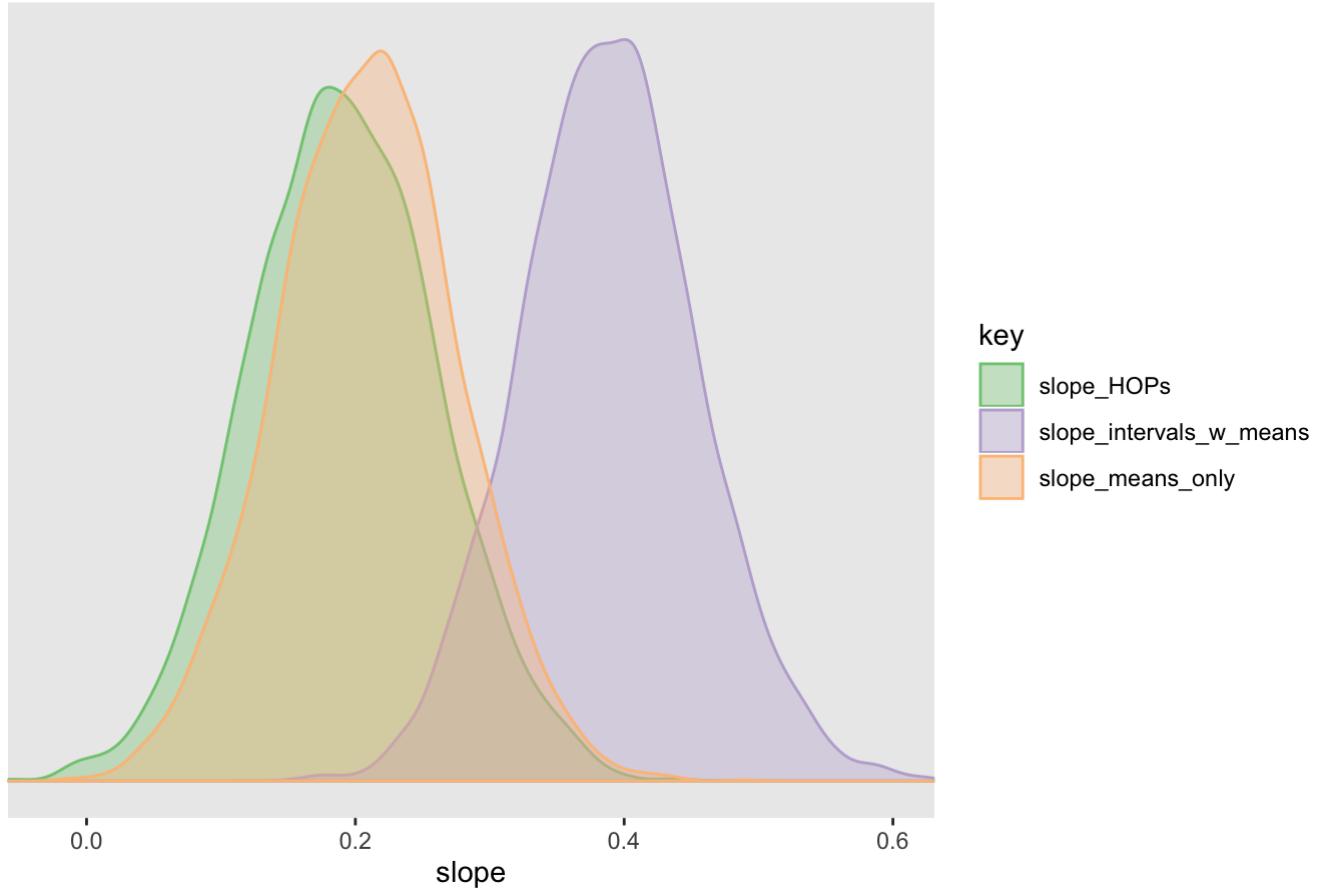
### Posterior predictive distribution for probability of superiority



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 condition
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())
```

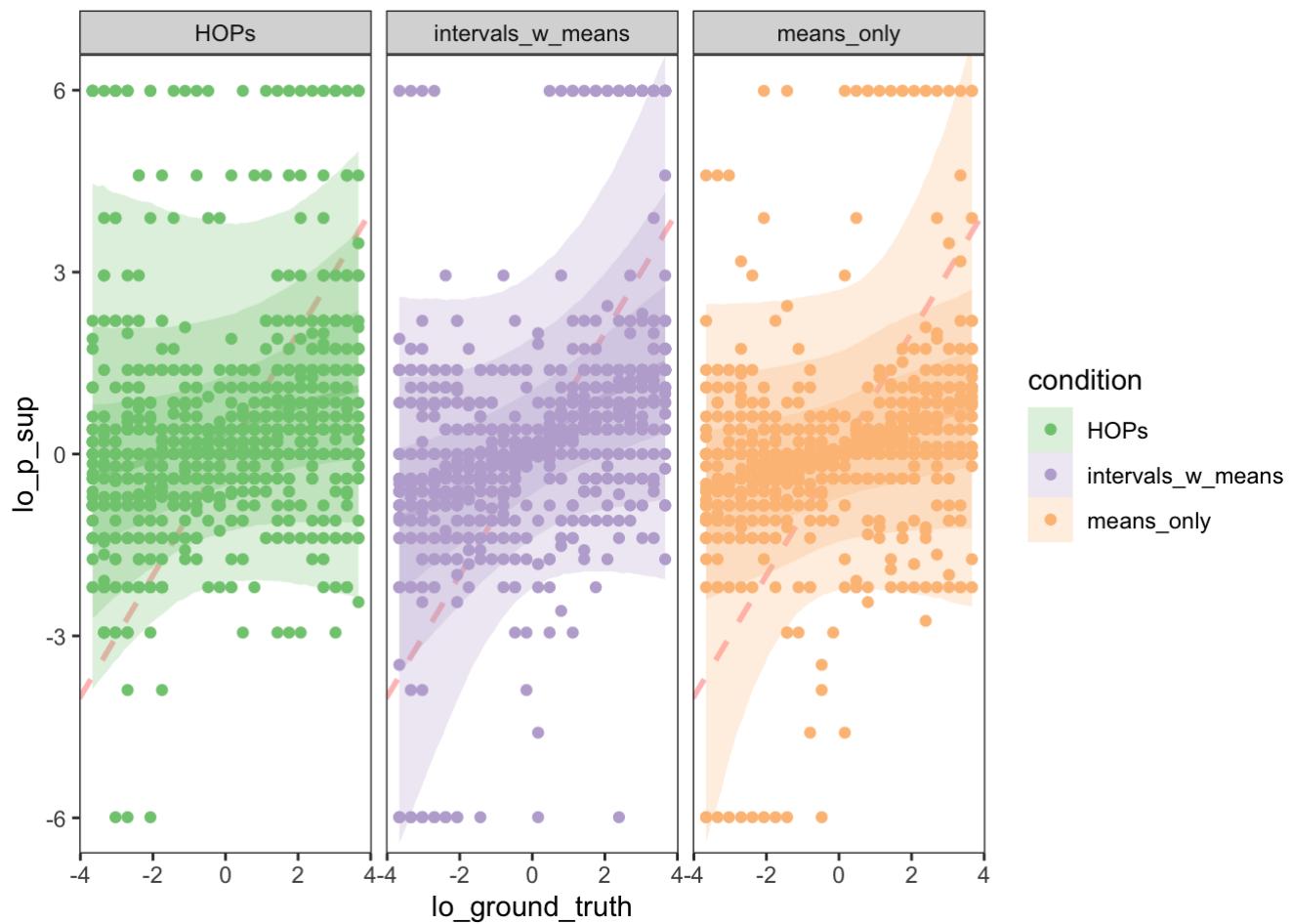
### Posterior for slopes by visualization condition



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

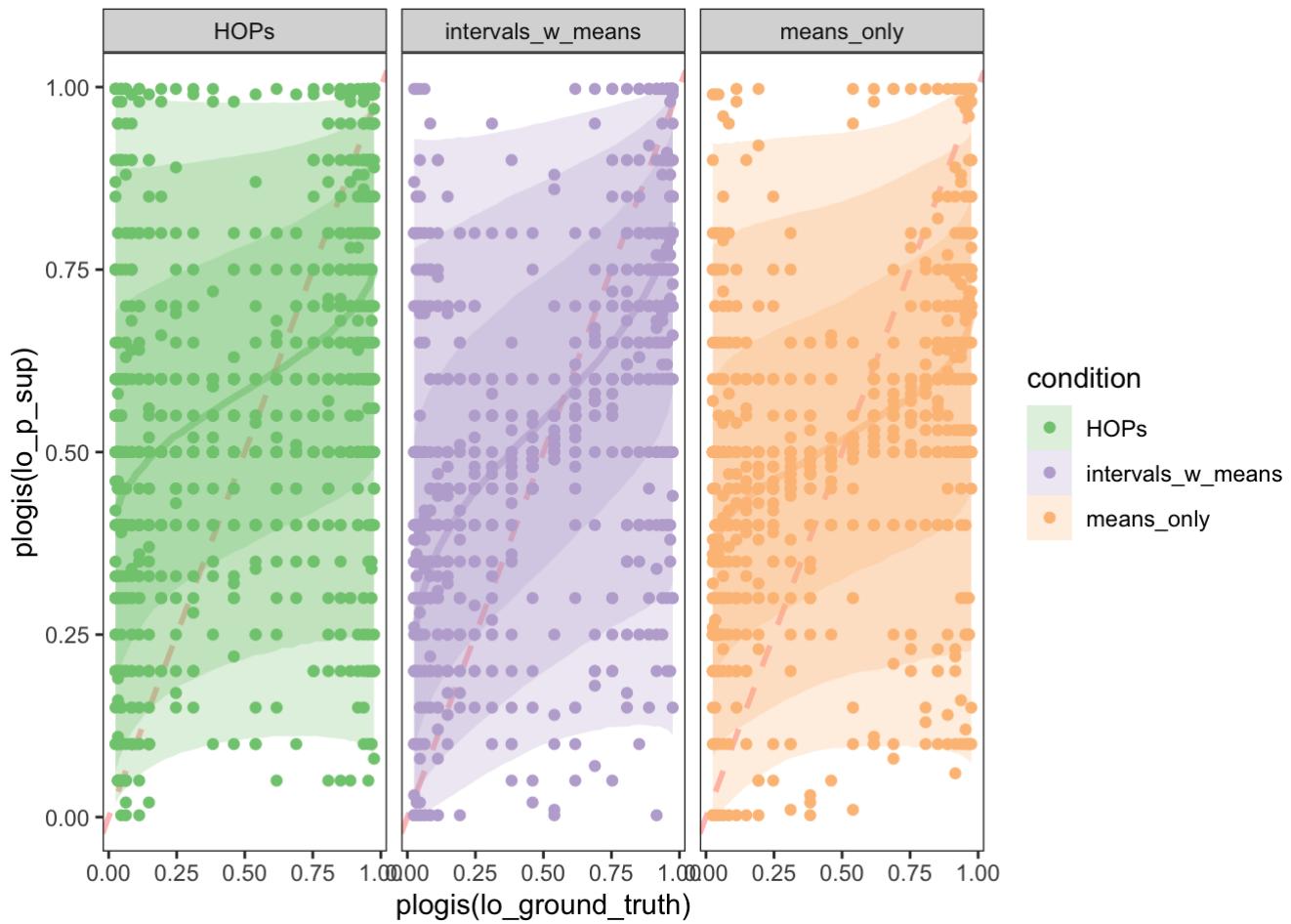
Let's take a look at some of the estimated linear models per visualization condition.

```
# this time we'll adopt functions from the tidybayes package to make plotting posterior
or predictions easier
model_df_lllo %>%
  group_by(condition, worker_id) %>%
  data_grid(lo_ground_truth = seq_range(lo_ground_truth, n = 51)) %>%
  add_predicted_draws(m.vis.wrkr.lllo_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_lllo) +
  scale_fill_brewer(type = "qual", palette = 1) +
  scale_color_brewer(type = "qual", palette = 1) +
  coord_cartesian(xlim = quantile(model_df_lllo$lo_ground_truth, c(0, 1)),
                  ylim = quantile(model_df_lllo$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_ll0 %>%
  group_by(condition, worker_id) %>%
  data_grid(lo_ground_truth = seq_range(lo_ground_truth, n = 51)) %>%
  add_predicted_draws(m.vis.wrkr.ll0_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_ll0) +
  scale_fill_brewer(type = "qual", palette = 1) +
  scale_color_brewer(type = "qual", palette = 1) +
  coord_cartesian(xlim = quantile(plogis(model_df_ll0$lo_ground_truth), c(0, 1)),
                  ylim = quantile(plogis(model_df_ll0$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 in the `intervals_w_means` and `means_only` conditions where there are no observations. Maybe adding the baseline condition to the model will improve the fit?

## Add a Slope Interaction for Baseline Condition

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 baseline conditions. This gives us a three way interaction.

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

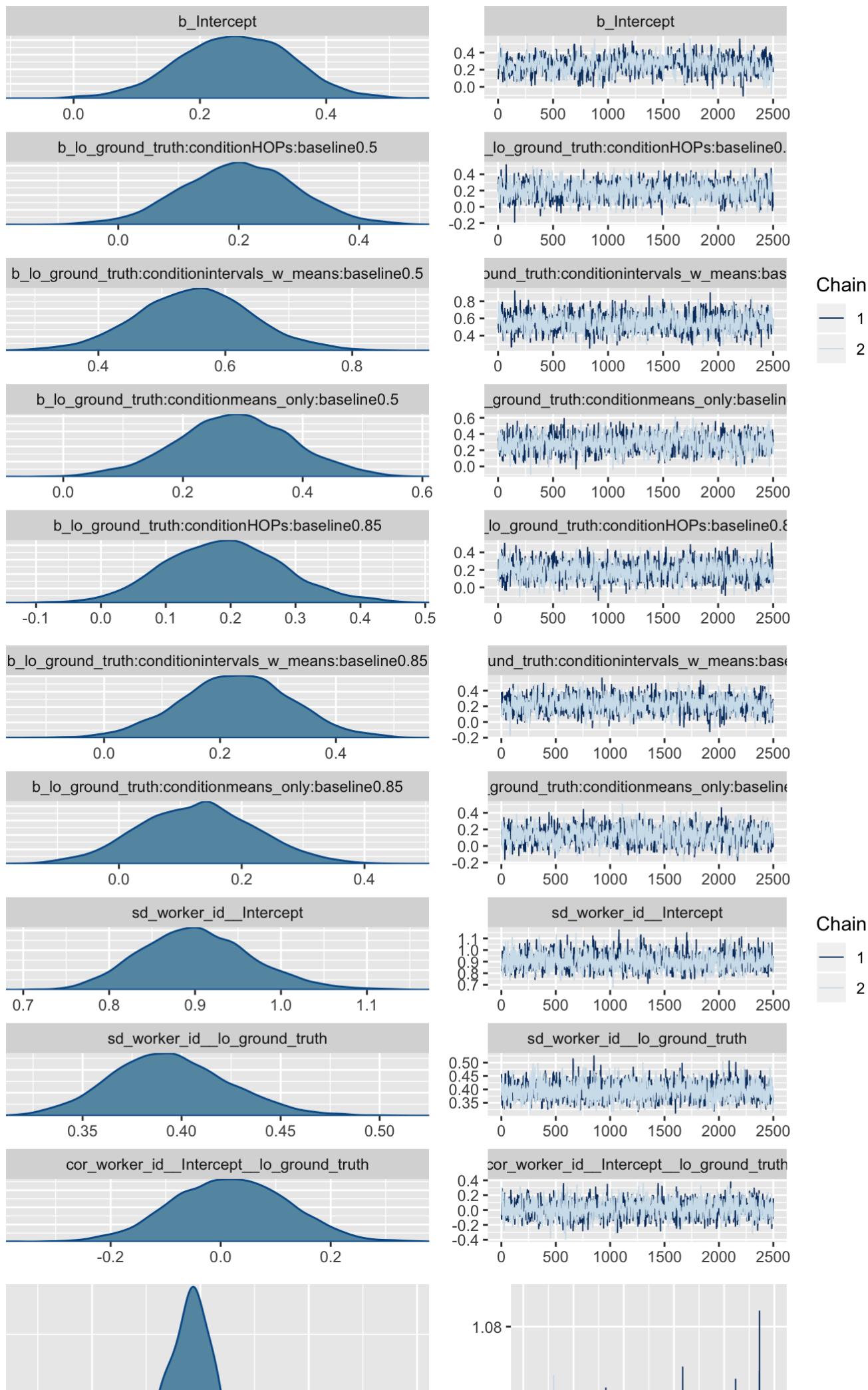
```
## The desired updates require recompiling the model
```

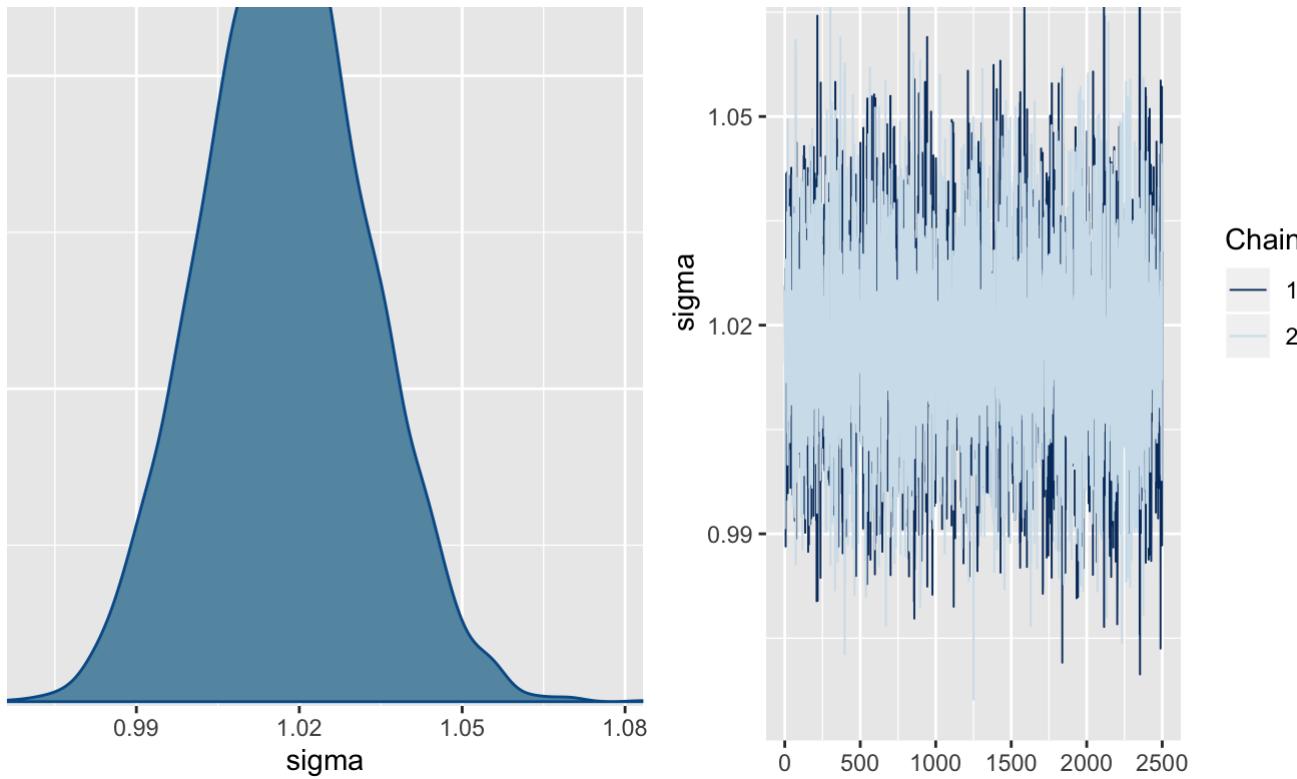
Check diagnostics:

- Trace plots

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

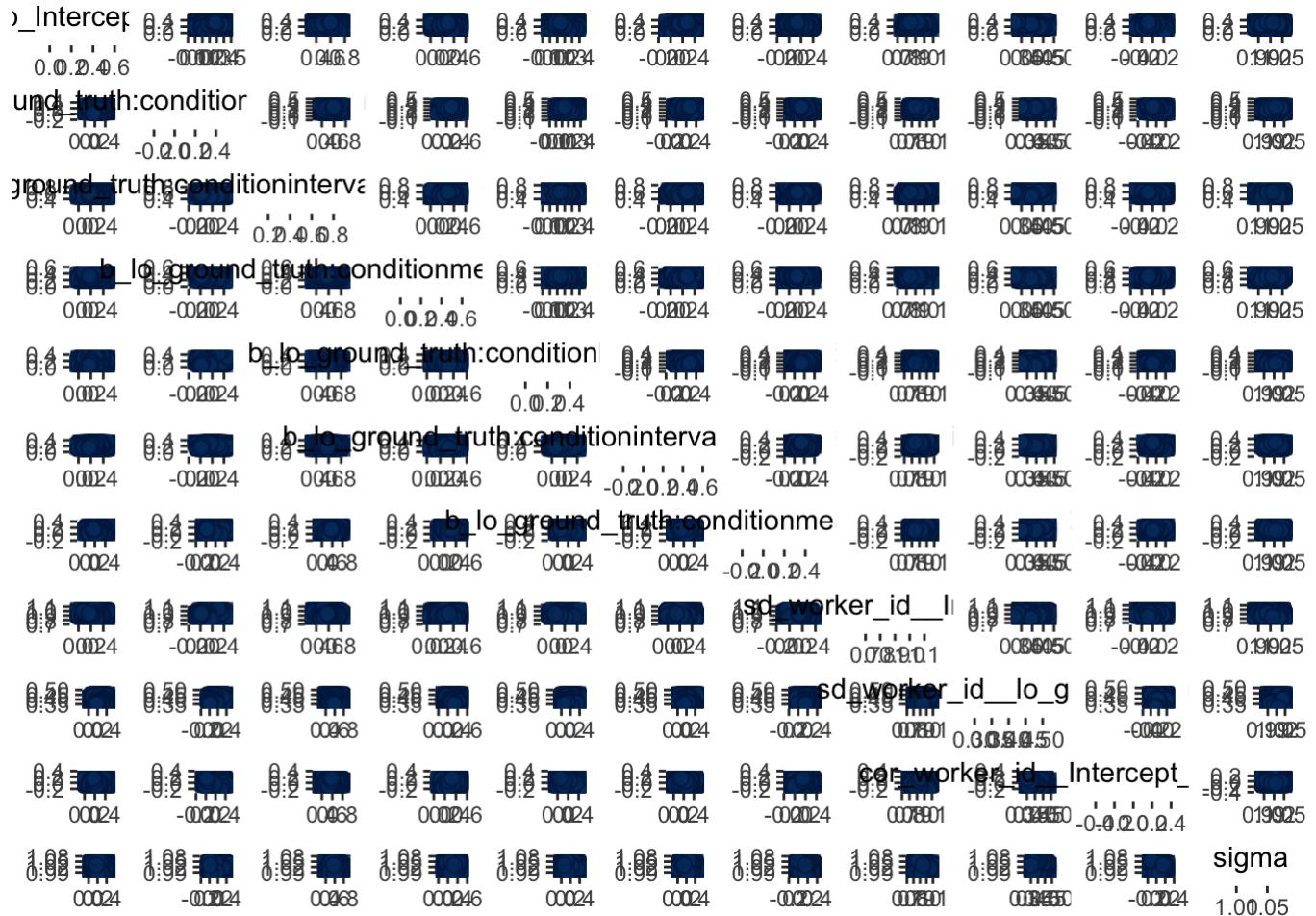






- Pairs plot

```
# pairs plot
pairs(m.vis.base.wrkr.lllo_p_sup)
```



- Summary

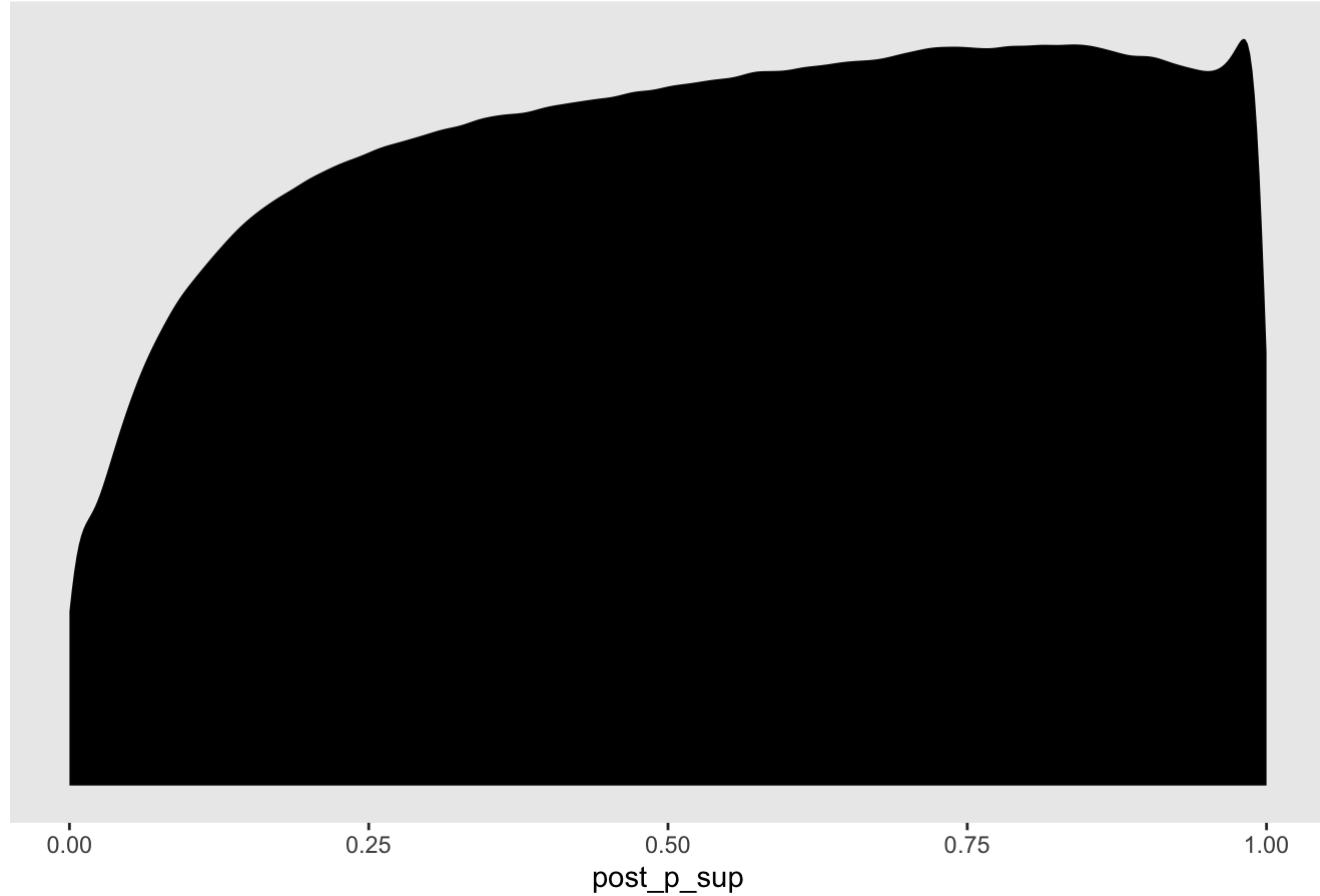
```
# model summary
print(m.vis.base.wrkr.lllo_p_sup)
```

```
## Family: gaussian
## Links: mu = identity; sigma = identity
## Formula: lo_p_sup ~ (1 + lo_ground_truth | worker_id) + lo_ground_truth:condition:baseline
## Data: model_df_lllo (Number of observations: 2616)
## Samples: 2 chains, each with iter = 3000; warmup = 500; thin = 1;
##          total post-warmup samples = 5000
##
## Group-Level Effects:
## ~worker_id (Number of levels: 109)
##             Estimate Est.Error 1-95% CI u-95% CI
## sd(Intercept)      0.90     0.06    0.79    1.04
## sd(lo_ground_truth) 0.39     0.03    0.34    0.46
## cor(Intercept,lo_ground_truth) 0.02     0.10   -0.18    0.22
##                 Eff.Sample Rhat
## sd(Intercept)        1203 1.00
## sd(lo_ground_truth) 1436 1.00
## cor(Intercept,lo_ground_truth) 1221 1.00
##
## Population-Level Effects:
##             Estimate Est.Error
## Intercept                  0.26     0.09
## lo_ground_truth:conditionHOPs:baseline0.5      0.20     0.09
## lo_ground_truth:conditionintervals_w_means:baseline0.5 0.55     0.09
## lo_ground_truth:conditionmeans_only:baseline0.5      0.29     0.10
## lo_ground_truth:conditionHOPs:baseline0.85       0.19     0.09
## lo_ground_truth:conditionintervals_w_means:baseline0.85 0.23     0.10
## lo_ground_truth:conditionmeans_only:baseline0.85      0.13     0.09
##                 1-95% CI u-95% CI
## Intercept                  0.08     0.43
## lo_ground_truth:conditionHOPs:baseline0.5      0.02     0.38
## lo_ground_truth:conditionintervals_w_means:baseline0.5 0.37     0.73
## lo_ground_truth:conditionmeans_only:baseline0.5      0.09     0.48
## lo_ground_truth:conditionHOPs:baseline0.85       0.02     0.38
## lo_ground_truth:conditionintervals_w_means:baseline0.85 0.04     0.41
## lo_ground_truth:conditionmeans_only:baseline0.85      -0.06    0.32
##                 Eff.Sample Rhat
## Intercept                  555 1.00
## lo_ground_truth:conditionHOPs:baseline0.5      1183 1.00
## lo_ground_truth:conditionintervals_w_means:baseline0.5 1340 1.00
## lo_ground_truth:conditionmeans_only:baseline0.5      985 1.00
## lo_ground_truth:conditionHOPs:baseline0.85       1203 1.00
## lo_ground_truth:conditionintervals_w_means:baseline0.85 975 1.00
## lo_ground_truth:conditionmeans_only:baseline0.85      1047 1.00
##
## Family Specific Parameters:
##             Estimate Est.Error 1-95% CI u-95% CI Eff.Sample Rhat
## sigma      1.02     0.02    0.99    1.05        9120 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_lllo %>%
  select(lo_ground_truth, condition, baseline, worker_id) %>%
  add_predicted_draws(m.vis.base.wrkr.lllo_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())
```

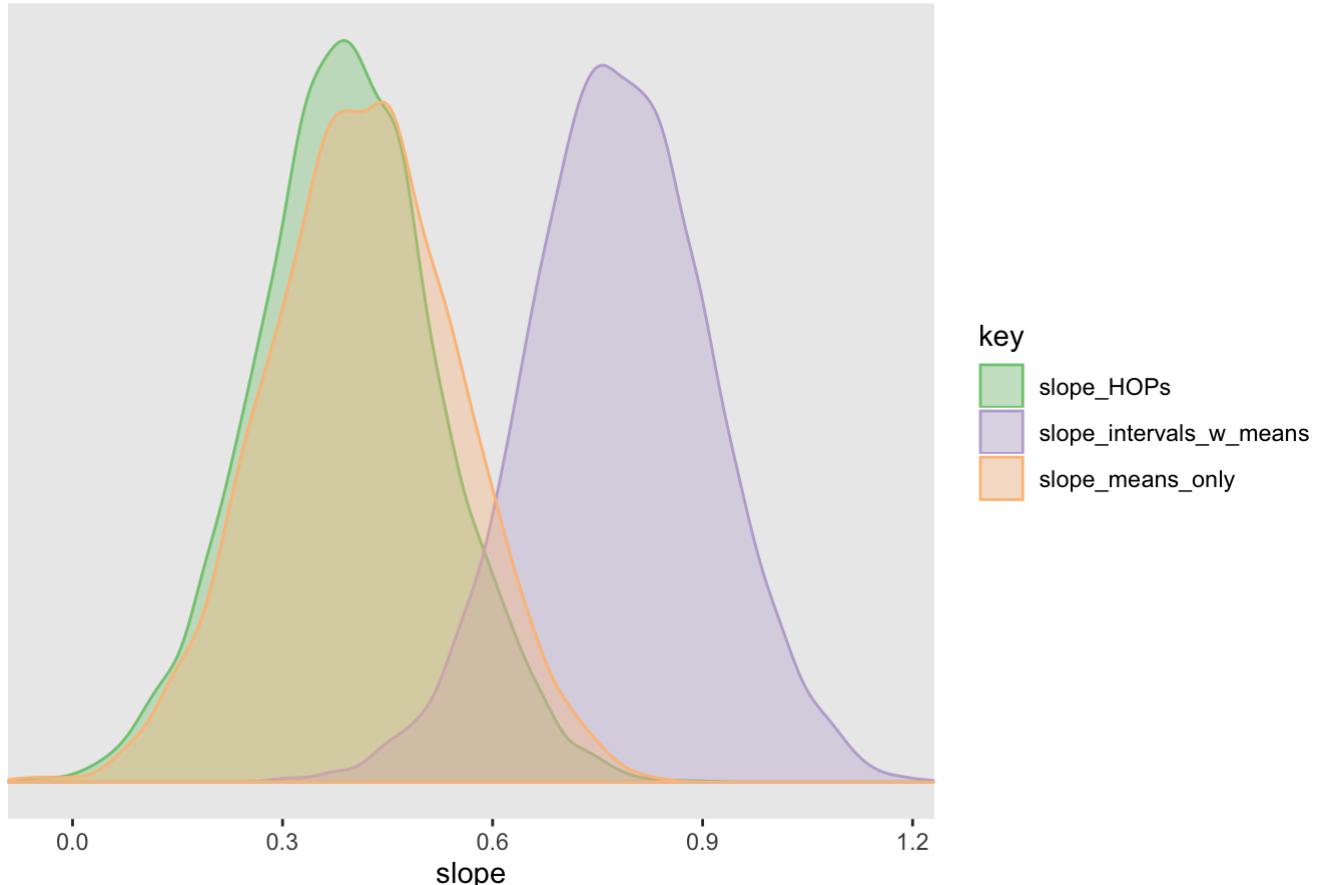
Posterior predictive distribution for probability of superiority



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

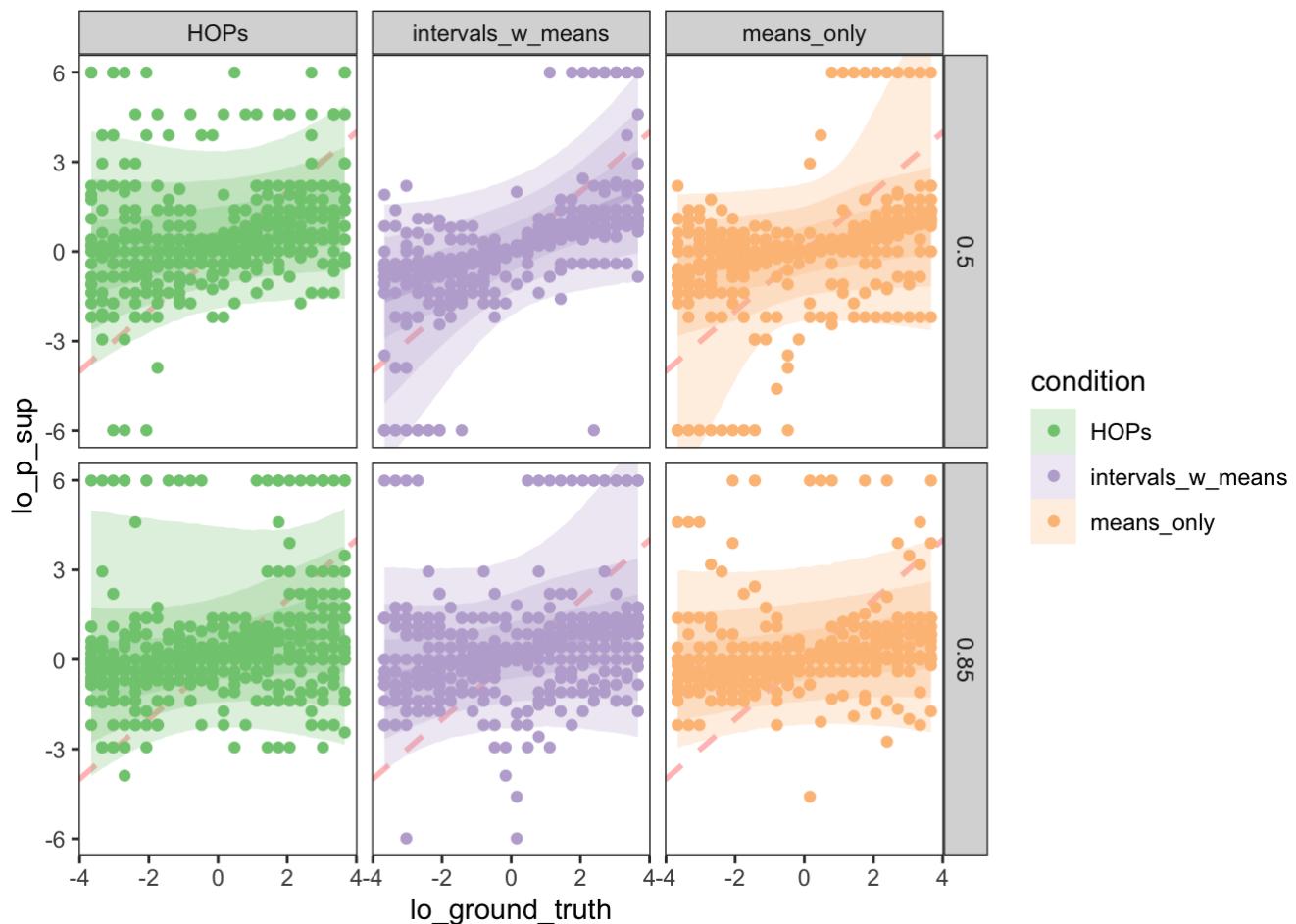
```
# use posterior samples to define distributions for the slope in each visualization condition
posterior_samples(m.vis.base.wrkr.ll0_p_sup) %>%
  transmute(slope_HOPs = `b_lo_ground_truth:conditionHOPs:baseline0.5` + `b_lo_ground_truth:conditionHOPs:baseline0.85`,
            slope_intervals_w_means = `b_lo_ground_truth:conditionintervals_w_means:baseline0.5` + `b_lo_ground_truth:conditionintervals_w_means:baseline0.85`,
            slope_means_only = `b_lo_ground_truth:conditionmeans_only:baseline0.5` + `b_lo_ground_truth:conditionmeans_only:baseline0.85`) %>%
  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



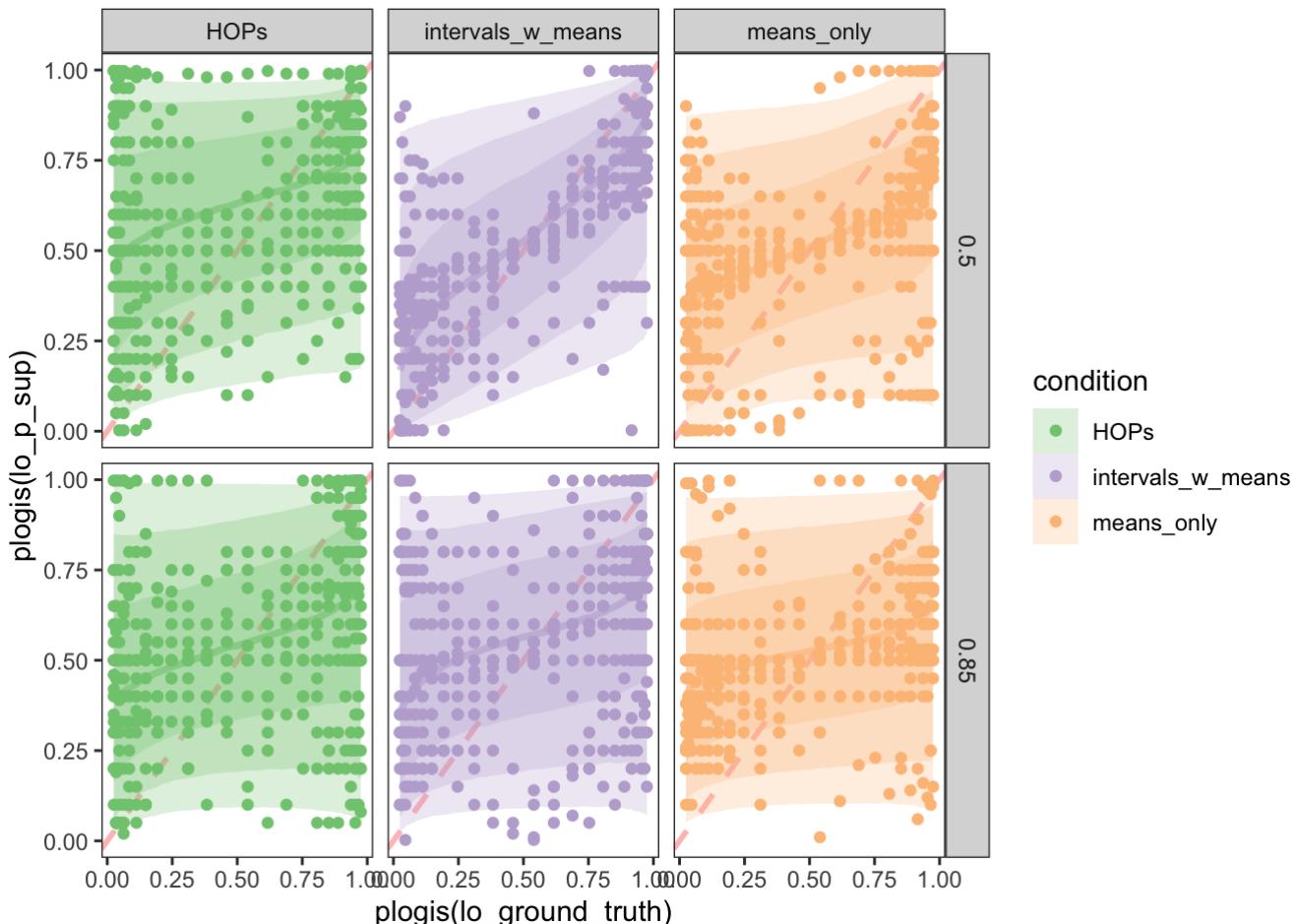
Let's take a look at some of the estimated linear models per visualization \* baseline condition.

```
# this time we'll adopt functions from the tidybayes package to make plotting posterior
or predictions easier
model_df_ll0 %>%
  group_by(condition, baseline, worker_id) %>%
  data_grid(lo_ground_truth = seq_range(lo_ground_truth, n = 51)) %>%
  add_predicted_draws(m.vis.base.wrkr.ll0_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_ll0) +
  scale_fill_brewer(type = "qual", palette = 1) +
  scale_color_brewer(type = "qual", palette = 1) +
  coord_cartesian(xlim = quantile(model_df_ll0$lo_ground_truth, c(0, 1)),
                  ylim = quantile(model_df_ll0$lo_p_sup, c(0, 1))) +
  theme_bw() +
  theme(panel.grid = element_blank()) +
  facet_grid(baseline ~ condition)
```



What does this look like in probability units?

```
model_df_lllo %>%
  group_by(condition, baseline, worker_id) %>%
  data_grid(lo_ground_truth = seq_range(lo_ground_truth, n = 51)) %>%
  add_predicted_draws(m.vis.base.wrkr.lllo_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_lllo) +
  scale_fill_brewer(type = "qual", palette = 1) +
  scale_color_brewer(type = "qual", palette = 1) +
  coord_cartesian(xlim = quantile(plogis(model_df_lllo$lo_ground_truth), c(0, 1)),
                  ylim = quantile(plogis(model_df_lllo$lo_p_sup), c(0, 1))) +
  theme_bw() +
  theme(panel.grid = element_blank()) +
  facet_grid(baseline ~ condition)
```



Adding an interaction for the baseline condition doesn't really fix the problem that our posterior predictions have high density where there are few responses. However, it does seem like probability of superiority judgments may be slightly more distorted when the baseline is 0.5.

Let's check whether adding parameters to account for baseline condition 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.lllo_p_sup, m.vis.base.wrkr.lllo_p_sup)
```

```

##                               WAIC      SE
## m.vis.wrkr.llo_p_sup      7753.64 172.09
## m.vis.base.wrkr.llo_p_sup 7751.80 172.68
## m.vis.wrkr.llo_p_sup - m.vis.base.wrkr.llo_p_sup  1.84   2.25

```

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 baseline condition to this model. This makes sense as the baseline conditions are 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( # heuristic functions
    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, baseline, ground_truth, process, process_est) %>%
  mutate(baseline = as.factor(baseline))

# 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", "baseline", "ground_truth", "process")) %>% # add process estimates to model_df
  select(-p_superiority) %>% # remove actual responses
  mutate(
    lo_p_sup = qlogis(process_est / 100) + rnorm(n(), 0, sigma_err), # likelihood function
    p_sup = plogis(lo_p_sup) * 100, # simulated responses from known data generating process
    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.

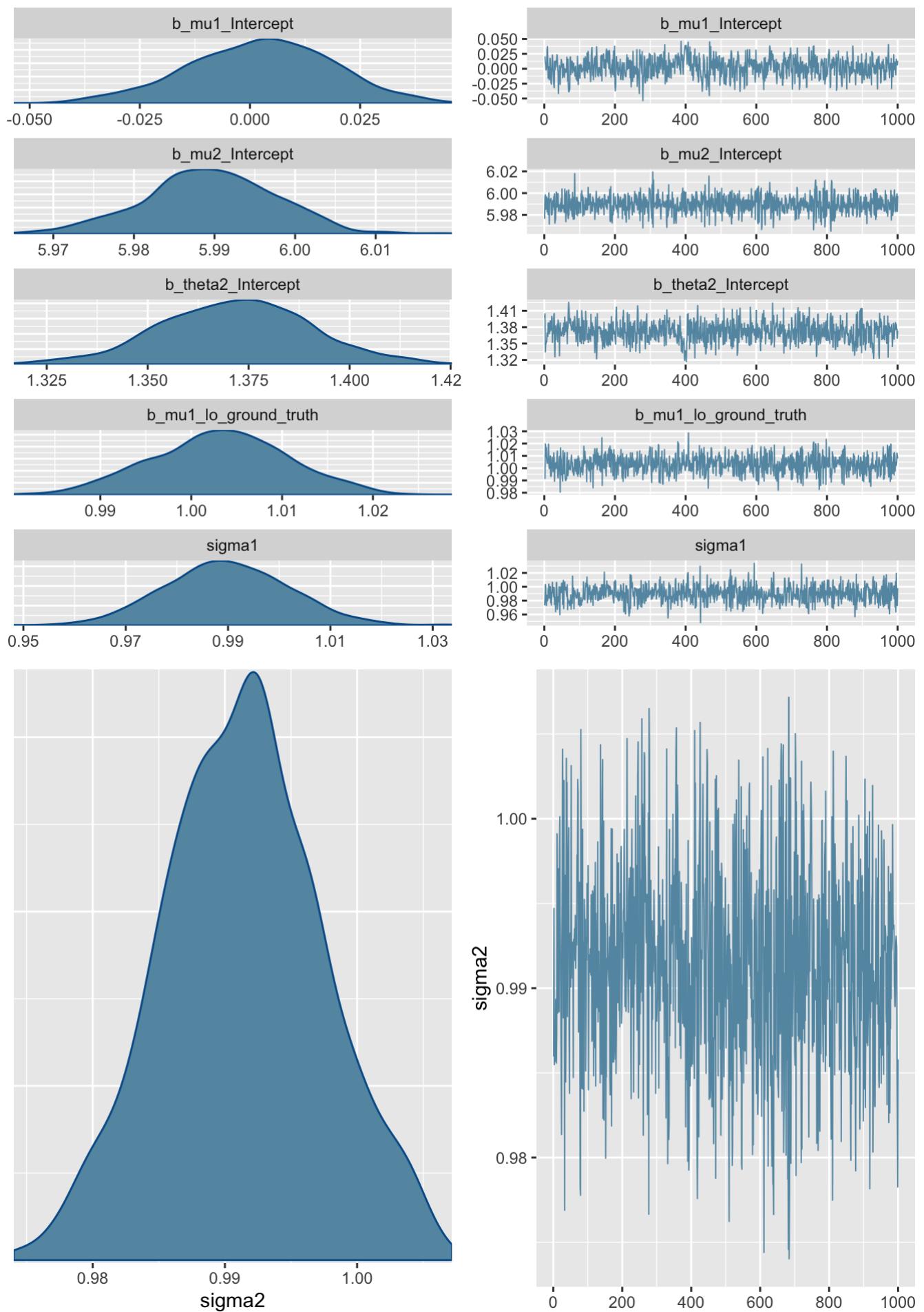
```
# get_prior
#   bf(lo_p_sup ~ 1, mu1 ~ lo_ground_truth, mu2 ~ 1, theta2 ~ 1),
#   data = fake_df_llo_mix,
#   family = mixture(gaussian, gaussian)
# )

# 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

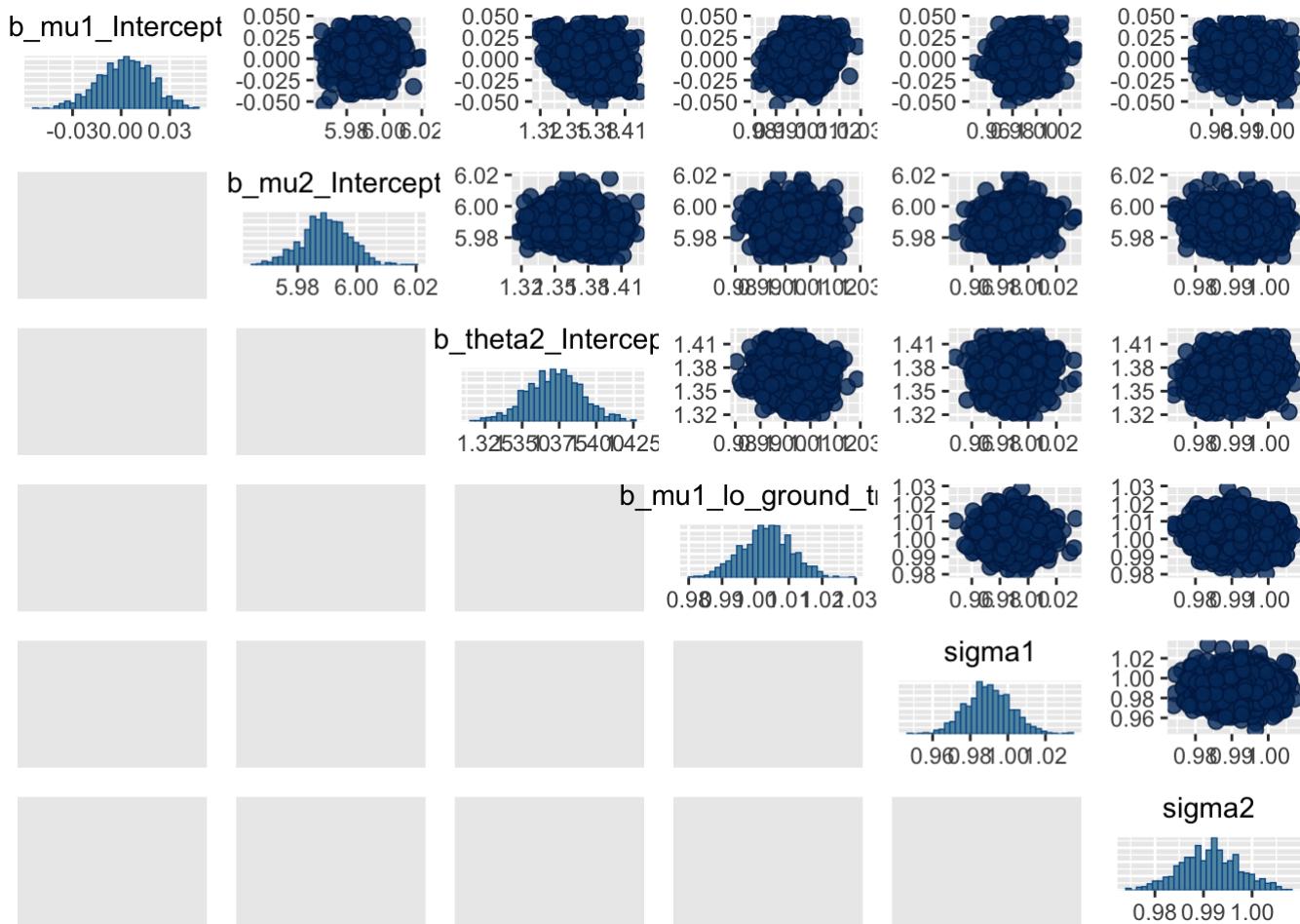
```
# trace plots
plot(m.fake.llo_mix)
```



- Pairs plot

```
# pairs plot
pairs(m.fake.lllo_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.lllo_mix)
```

```

## Family: mixture(gaussian, gaussian)
## Links: mu1 = identity; sigma1 = identity; mu2 = identity; sigma2 = identity; thet
ta1 = 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).

```

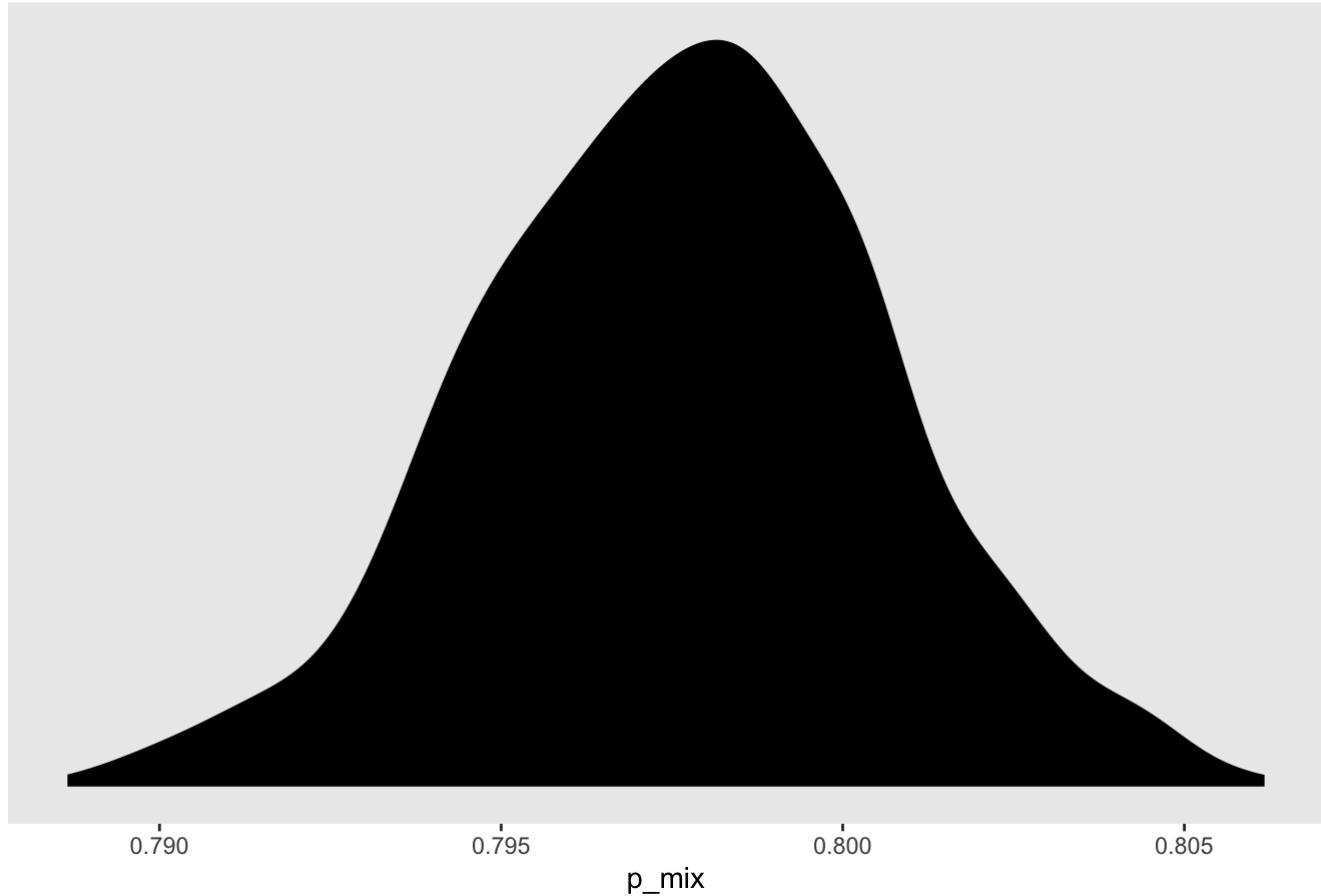
So, did we accurately recover the mixture proportions? Let's find out by plotting the posterior for theta. Because theta is in log odds units we'll transform it into probability units.

```

# 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())

```

### Posterior distribution for mixture proportion

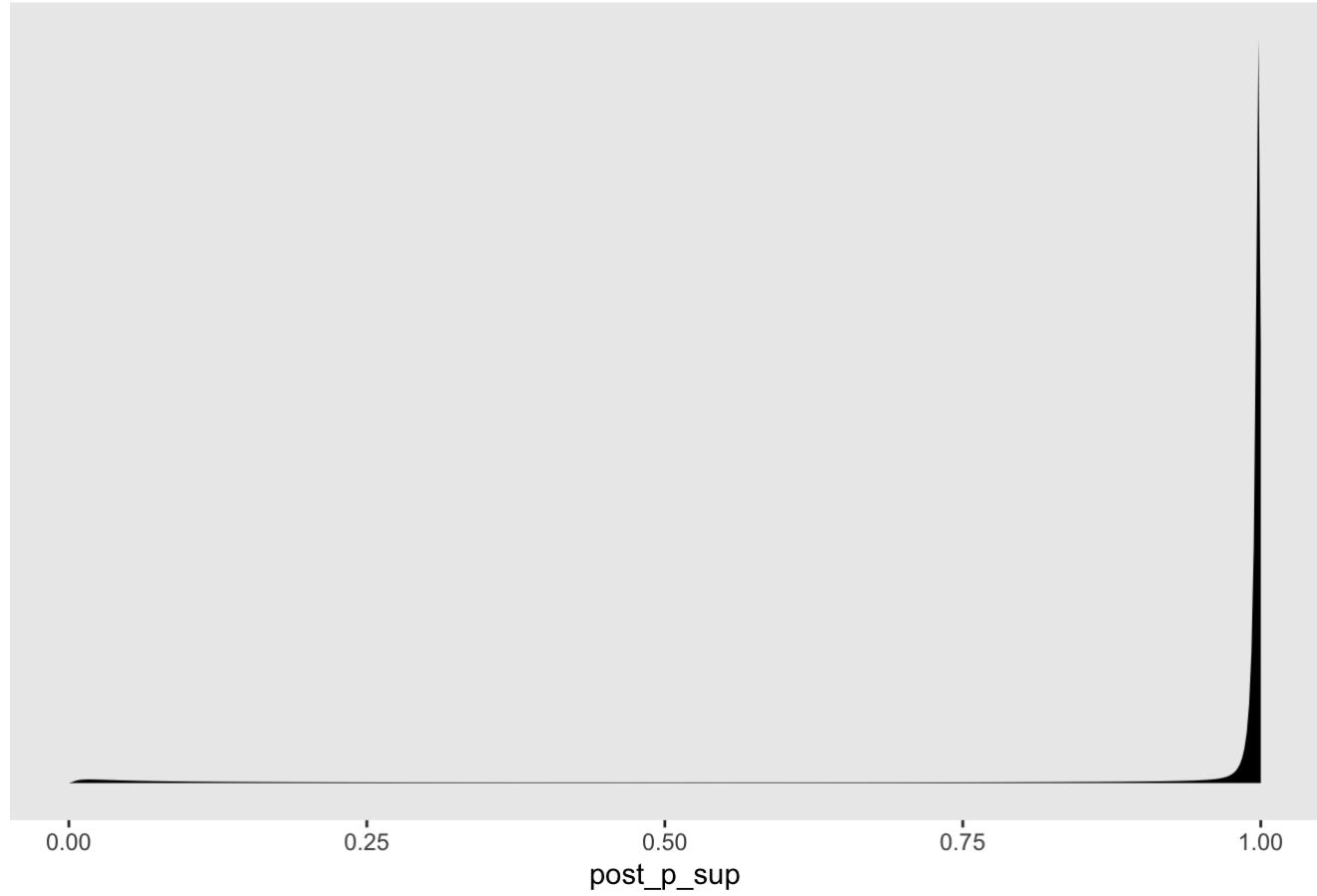


We succeeded in estimating the proportion of each alternative within a simulated mixture. This approach should help us to dramatically improve our model of real data.

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, condition, baseline, 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())
```

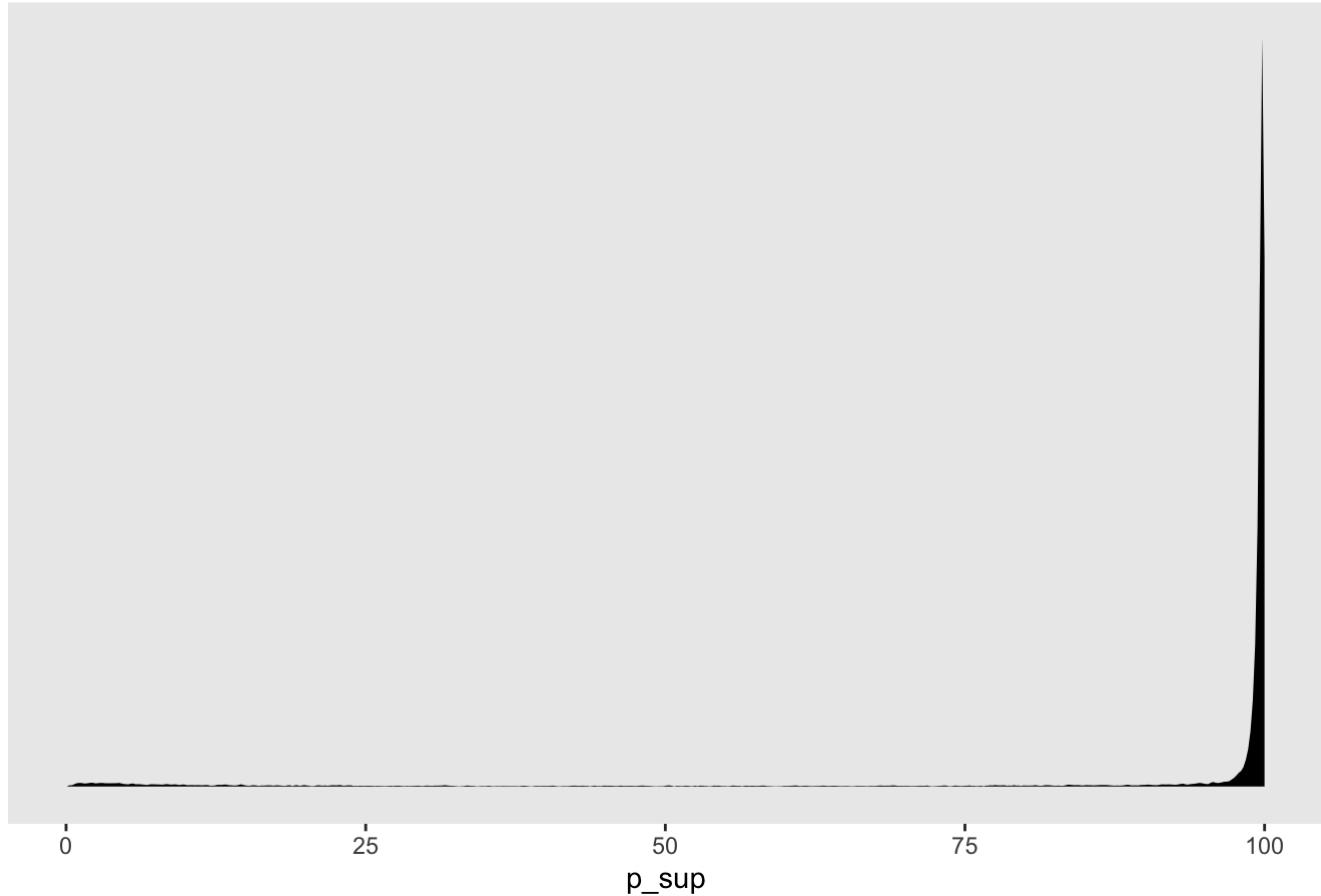
### Posterior predictive distribution for probability of superiority



And let's compare this to the data distribution.

```
# posterior predictive check
fake_df_lllo_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

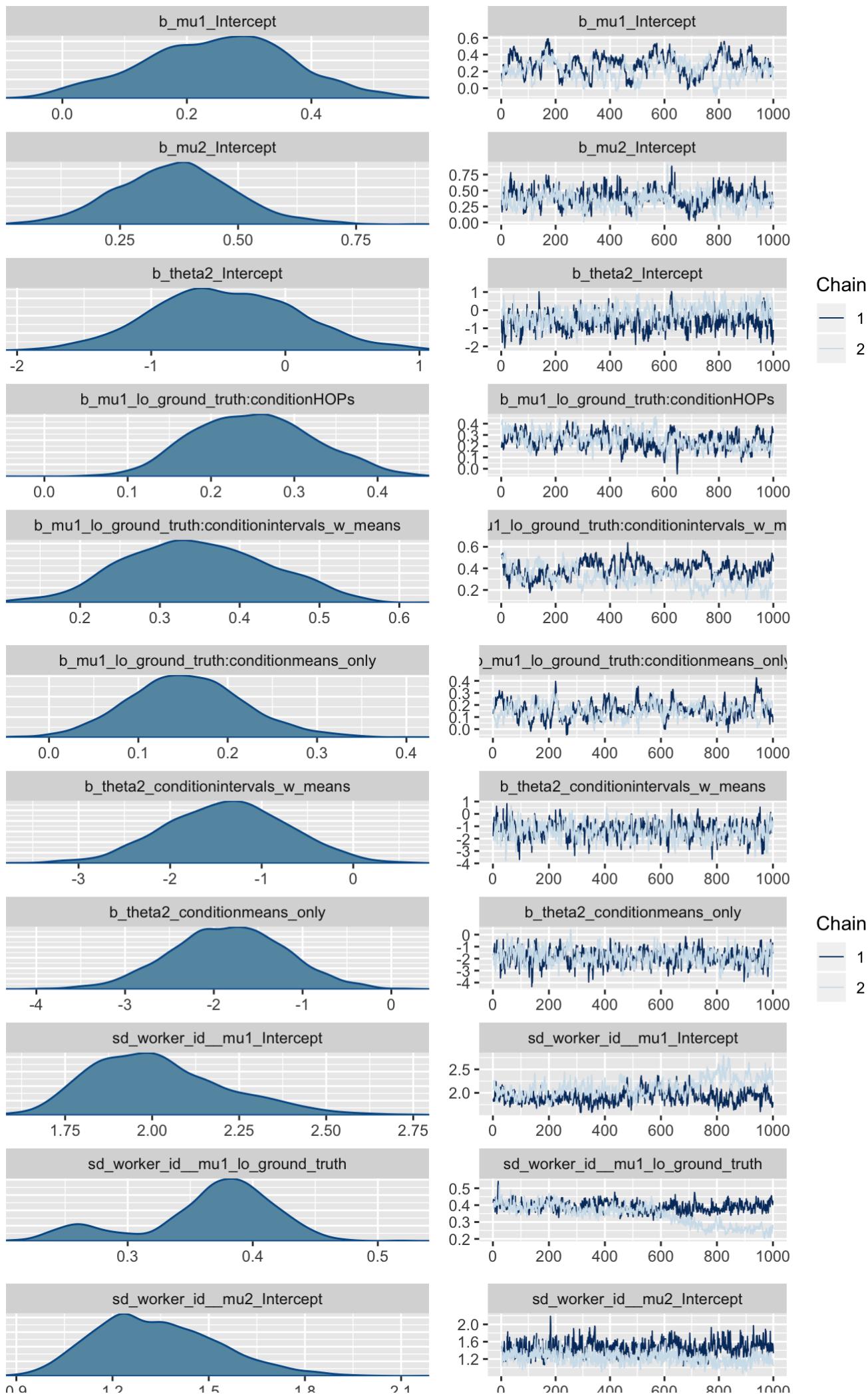
Let's adopt this mixture model for our actual probability of superiority responses by adding a constant response distribution to the iteration of the LLO model with different slopes per visualization condition and random slopes and intercepts per worker.

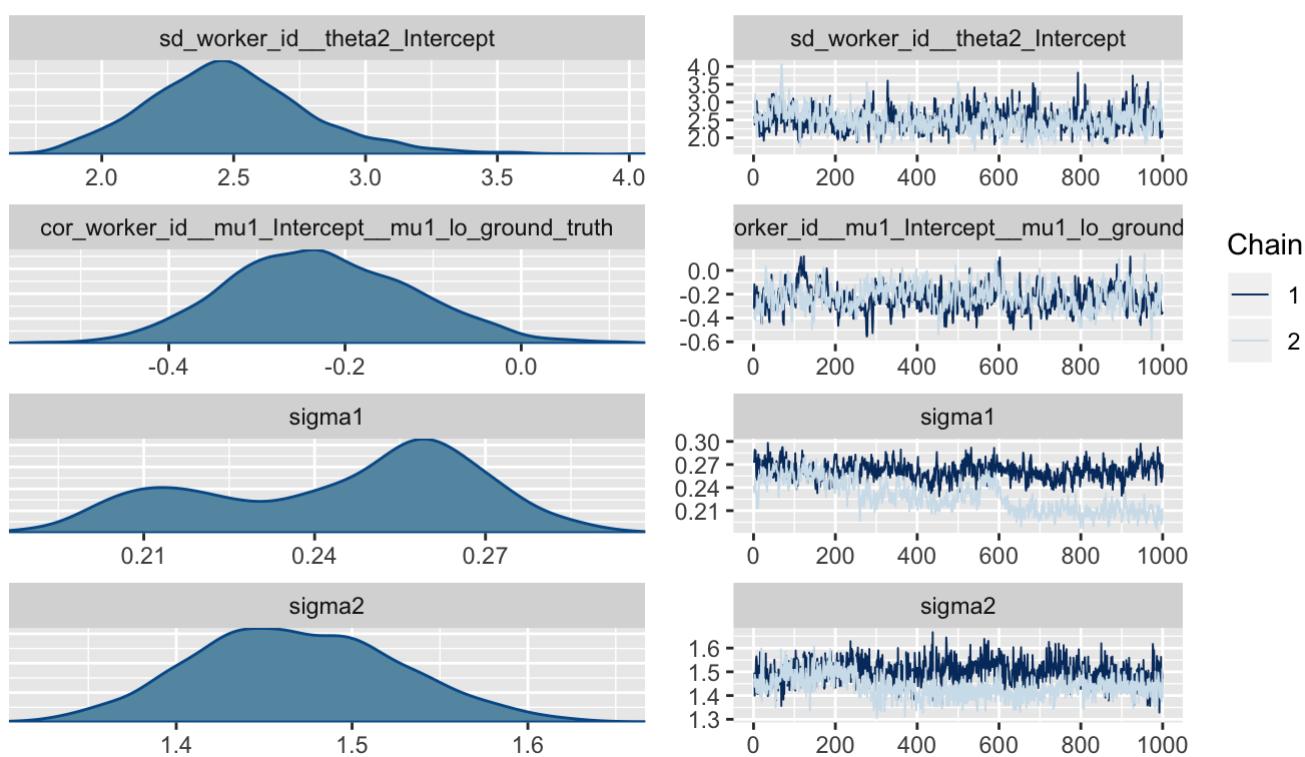
```
m.vis.wrkr.llo_mix <- brm(
  bf(lo_p_sup ~ 1,
    mu1 ~ (1 + lo_ground_truth|worker_id) + lo_ground_truth:condition, # llo model
    mu2 ~ (1|worker_id), # random constant response per worker (to account for people
    who always answer the same, often but not always 50%)
    theta2 ~ (1|worker_id) + 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

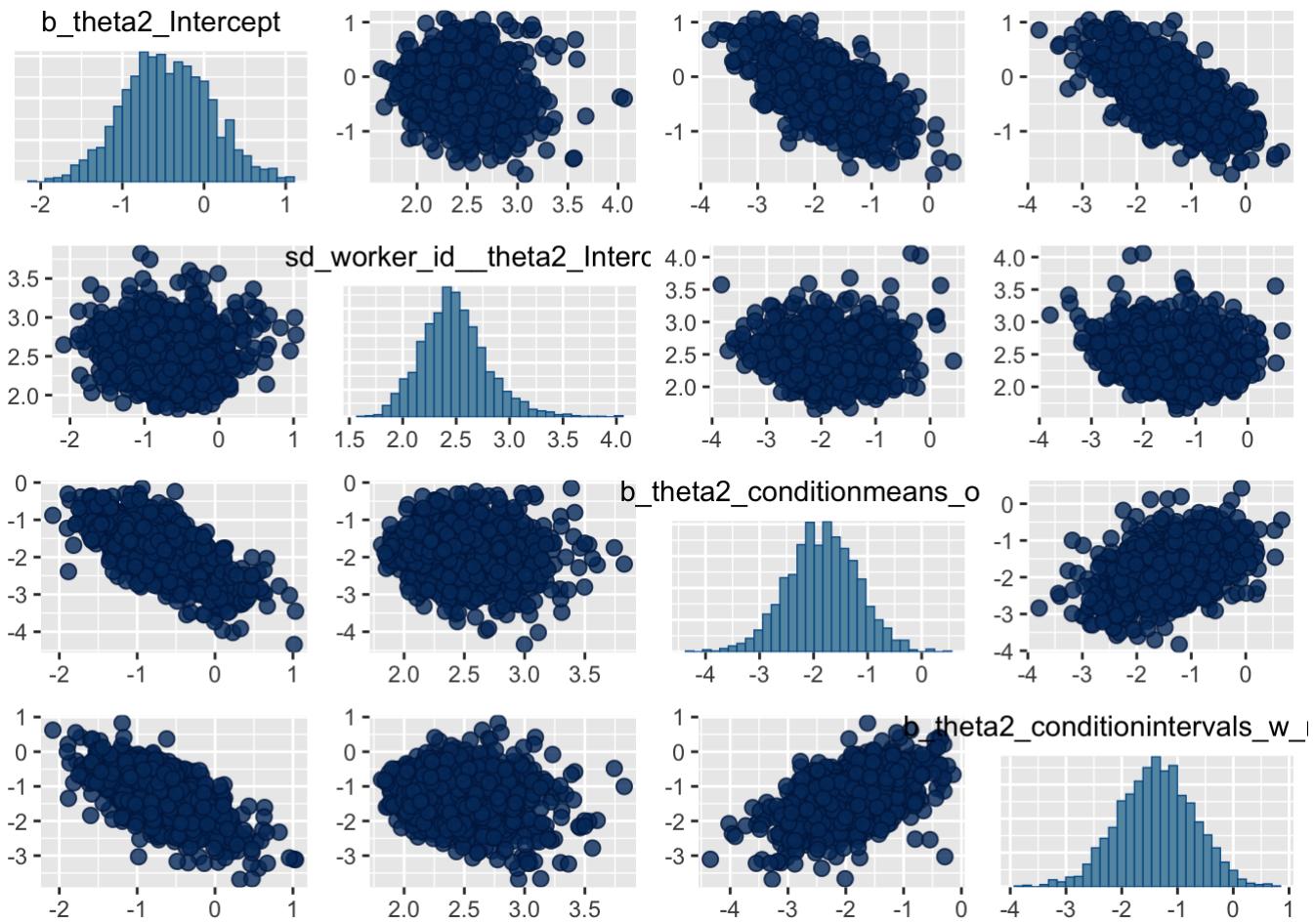
```
# trace plots
plot(m.vis.wrkr.llo_mix)
```



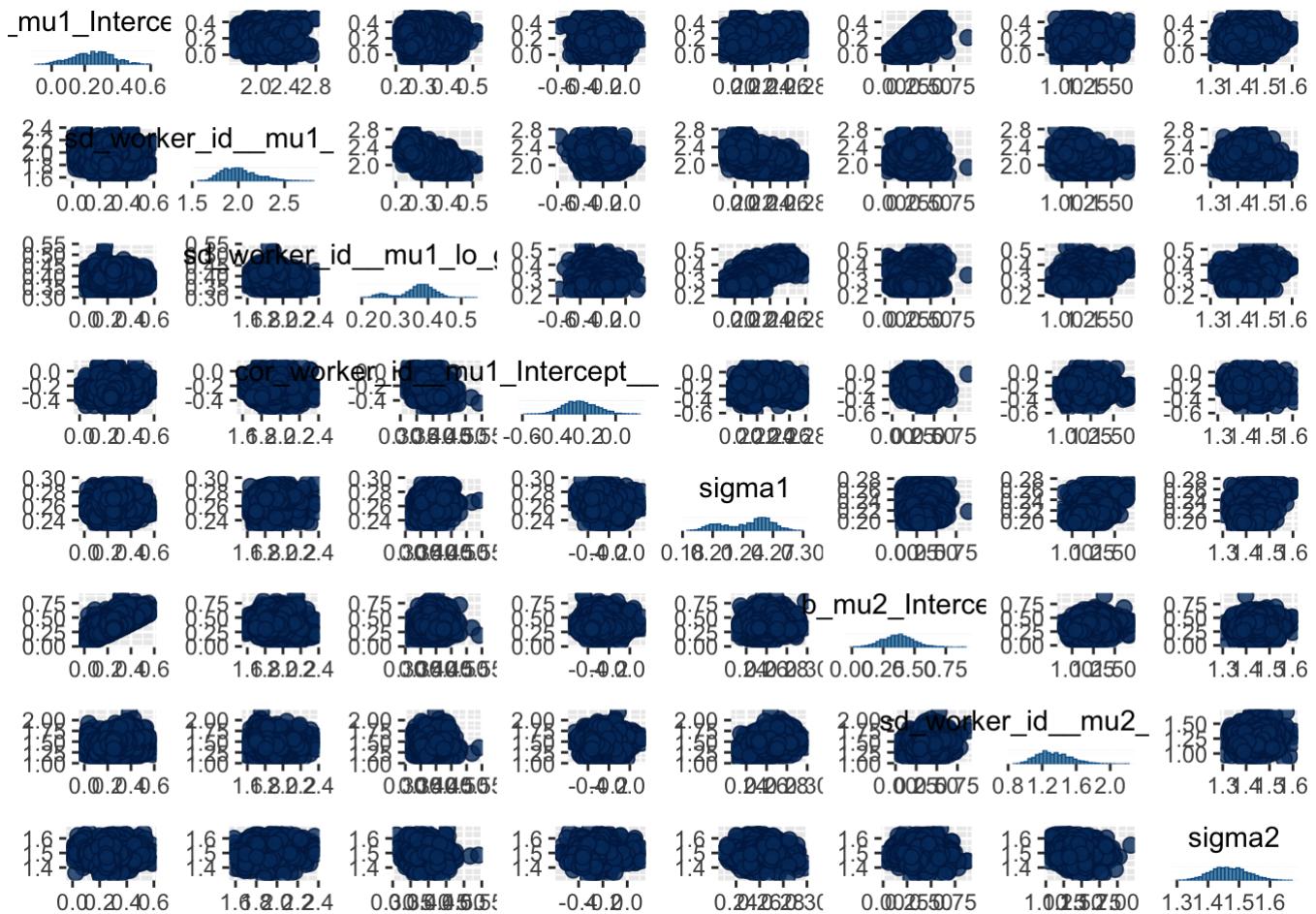


- Pairs plot

```
# 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
model because of dummy coding
                                    "b_theta2_conditionmeans_only",
                                    "b_theta2_conditionintervals_w_means"))
```

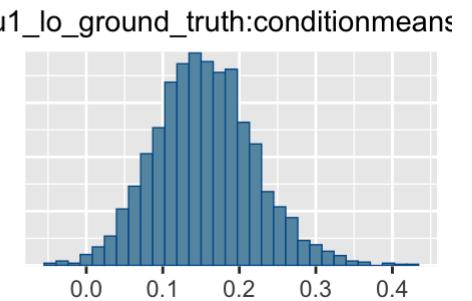
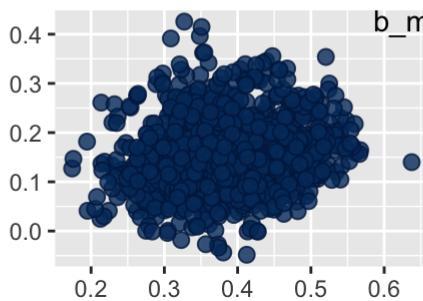
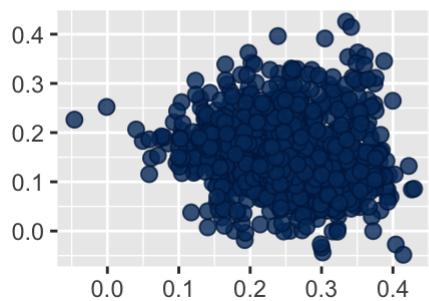
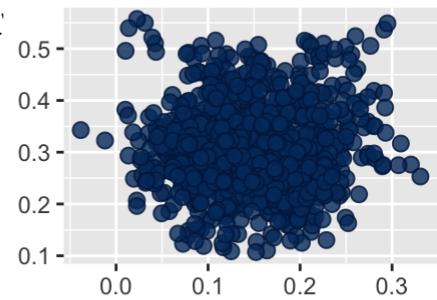
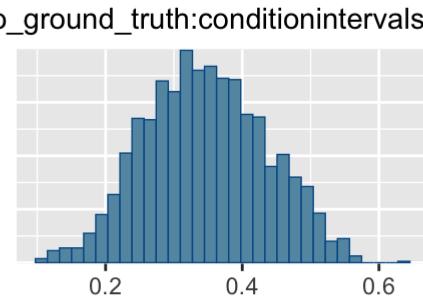
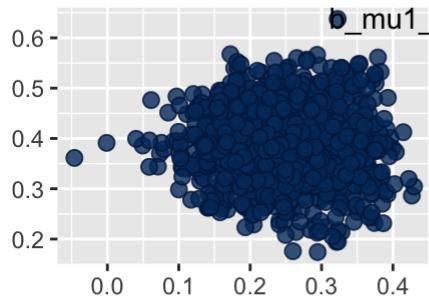
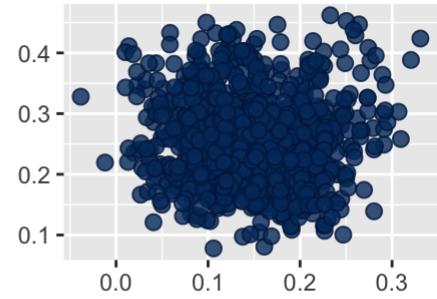
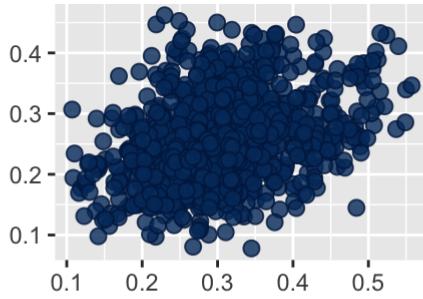
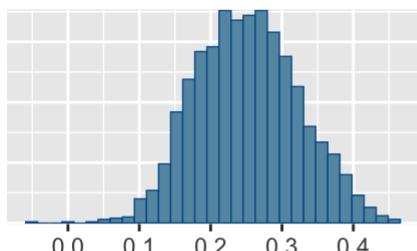


```
# pairs plot (too many things to view at once, so we've grouped them)
# hyperparameters
pairs(m.vis.wrkr.llo_mix, pars = c("b_mu1_Intercept",
                                    "sd_worker_id_mu1_Intercept",
                                    "sd_worker_id_mu1_lo_ground_truth",
                                    "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"))
```

`.mu1_lo_ground_truth:conditionHO`



- Summary

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

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

```

## Family: mixture(gaussian, gaussian)
## Links: mu1 = identity; sigma1 = identity; mu2 = identity; sigma2 = identity; thet
ta1 = 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: 2616)
## Samples: 2 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 2000
##
## Group-Level Effects:
## ~worker_id (Number of levels: 109)
##                               Estimate Est.Error l-95% CI
## sd(mu1_Intercept)            2.02    0.20    1.71
## sd(mu1_lo_ground_truth)     0.37    0.05    0.24
## sd(mu2_Intercept)           1.34    0.19    1.02
## sd(theta2_Intercept)        2.49    0.31    1.93
## cor(mu1_Intercept,mu1_lo_ground_truth) -0.23    0.11   -0.42
##                                         u-95% CI Eff.Sample Rhat
## sd(mu1_Intercept)            2.46      3 1.39
## sd(mu1_lo_ground_truth)     0.45      4 1.59
## sd(mu2_Intercept)           1.75      2 1.32
## sd(theta2_Intercept)         3.18     187 1.02
## cor(mu1_Intercept,mu1_lo_ground_truth) -0.02    154 1.01
##
## Population-Level Effects:
##                               Estimate Est.Error l-95% CI
## mu1_Intercept                0.25    0.12    0.01
## mu2_Intercept                0.37    0.13    0.12
## theta2_Intercept              -0.45    0.53   -1.48
## mu1_lo_ground_truth:conditionHOPs 0.25    0.07    0.12
## mu1_lo_ground_truth:conditionintervals_w_means 0.35    0.09    0.18
## mu1_lo_ground_truth:conditionmeans_only 0.15    0.07    0.03
## theta2_conditionintervals_w_means -1.38    0.68   -2.70
## theta2_conditionmeans_only    -1.84    0.65   -3.11
##                                         u-95% CI Eff.Sample Rhat
## mu1_Intercept                 0.48     45 1.07
## mu2_Intercept                 0.63     84 1.03
## theta2_Intercept               0.63      6 1.17
## mu1_lo_ground_truth:conditionHOPs 0.39     44 1.09
## mu1_lo_ground_truth:conditionintervals_w_means 0.52      4 1.27
## mu1_lo_ground_truth:conditionmeans_only 0.29    142 1.01
## theta2_conditionintervals_w_means -0.10    366 1.01
## theta2_conditionmeans_only    -0.62    415 1.00
##
## Family Specific Parameters:
##                               Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## sigma1       0.24      0.02      0.20      0.28      2 1.86
## sigma2       1.47      0.06      1.36      1.59      3 1.26
##
## 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).

```

There are a couple problems with this model. First, the trace plots show some divergence between chains in estimating individual variability in slopes `sd_worker_id_mu1_lo_ground_truth` and intercepts `sd_worker_id_mu1_Intercept`, respectively as well as the residual error `sigma1`. This lack of convergence is symptomatic of a parameter space with difficult geometry. This means we need to reparameterize our model.

A second problem may be the root cause of our issues. The posterior samples are strongly correlated among the estimates of the rate of the constant response strategy in each visualization condition (e.g., `b_theta2_conditionmeans_only`). In order to make the geometry of parameter space easier for the MCMC sampler, the model needs to know about this correlation between conditions. We reparameterize the model in order to represent the correlation between these three parameters 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;")

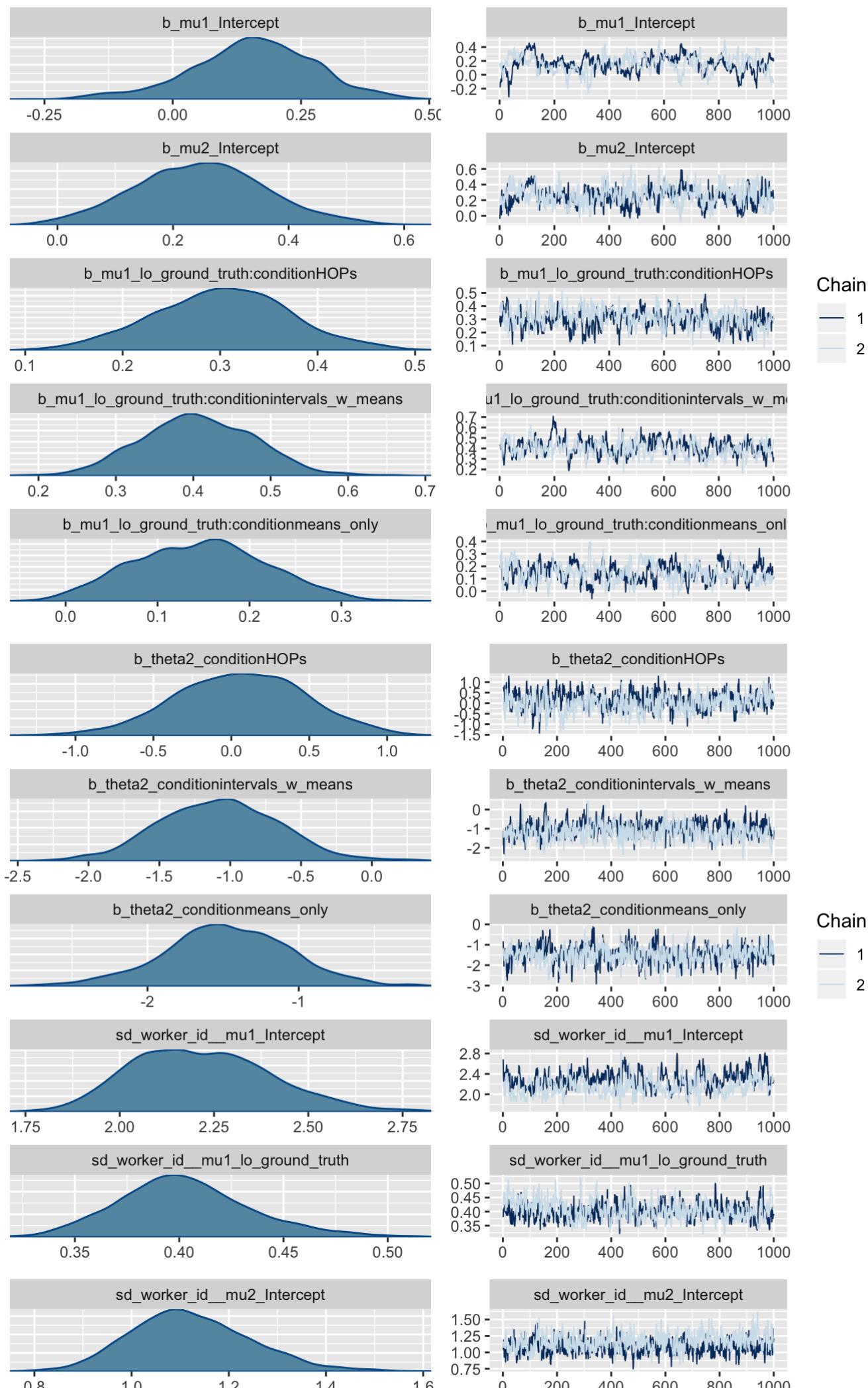
# fit the model
m.vis.wrkr.llo_mix2 <- brm(
  bf(lo_p_sup ~ 1,
    mu1 ~ (1 + lo_ground_truth|worker_id) + lo_ground_truth:condition, # our most recent llo model
    mu2 ~ (1|worker_id), # random constant response per worker (to account for people who always answer the same, often but not always 50%)
    theta2 ~ (1|worker_id) + 0 + 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),
    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"
)
```

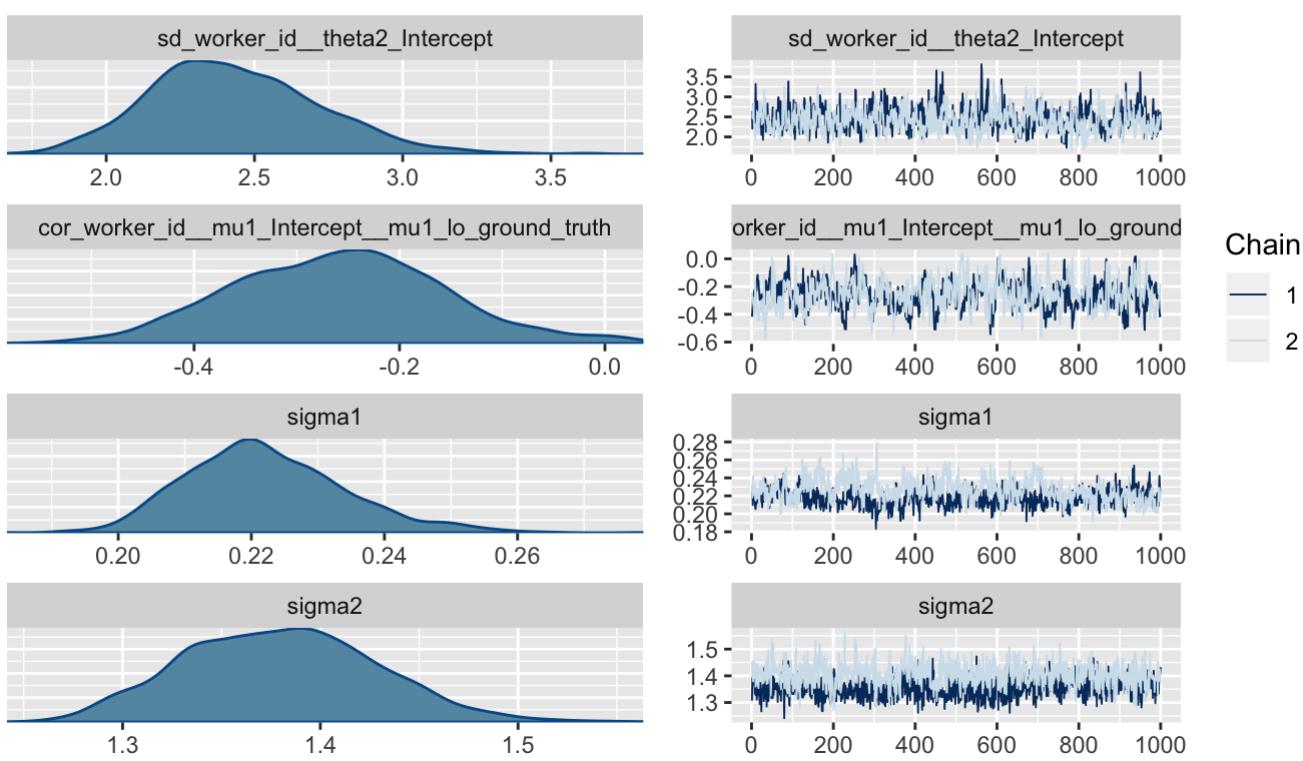
Check diagnostics:

- Trace plots

```
# trace plots
plot(m.vis.wrkr.llo_mix2)
```

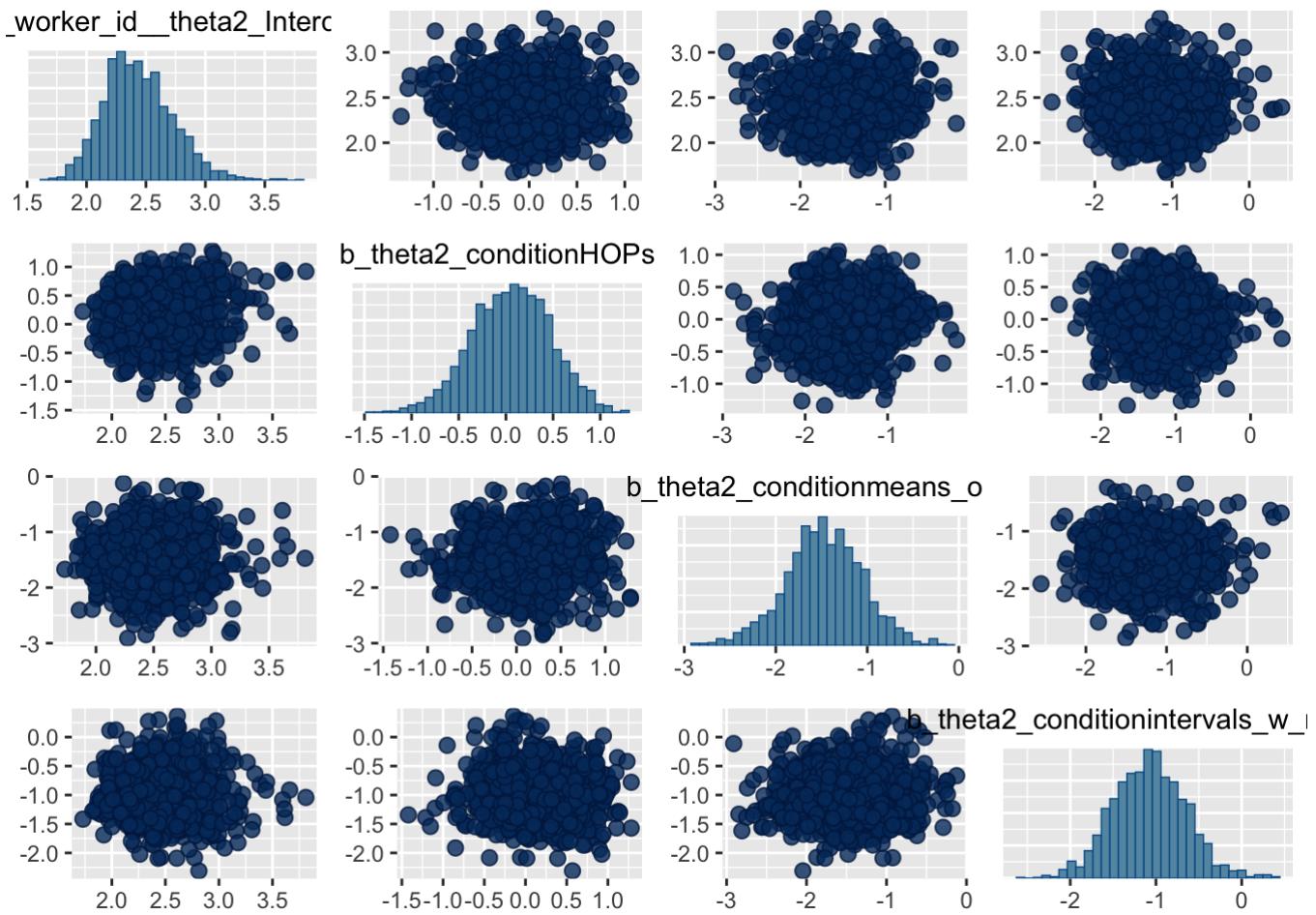




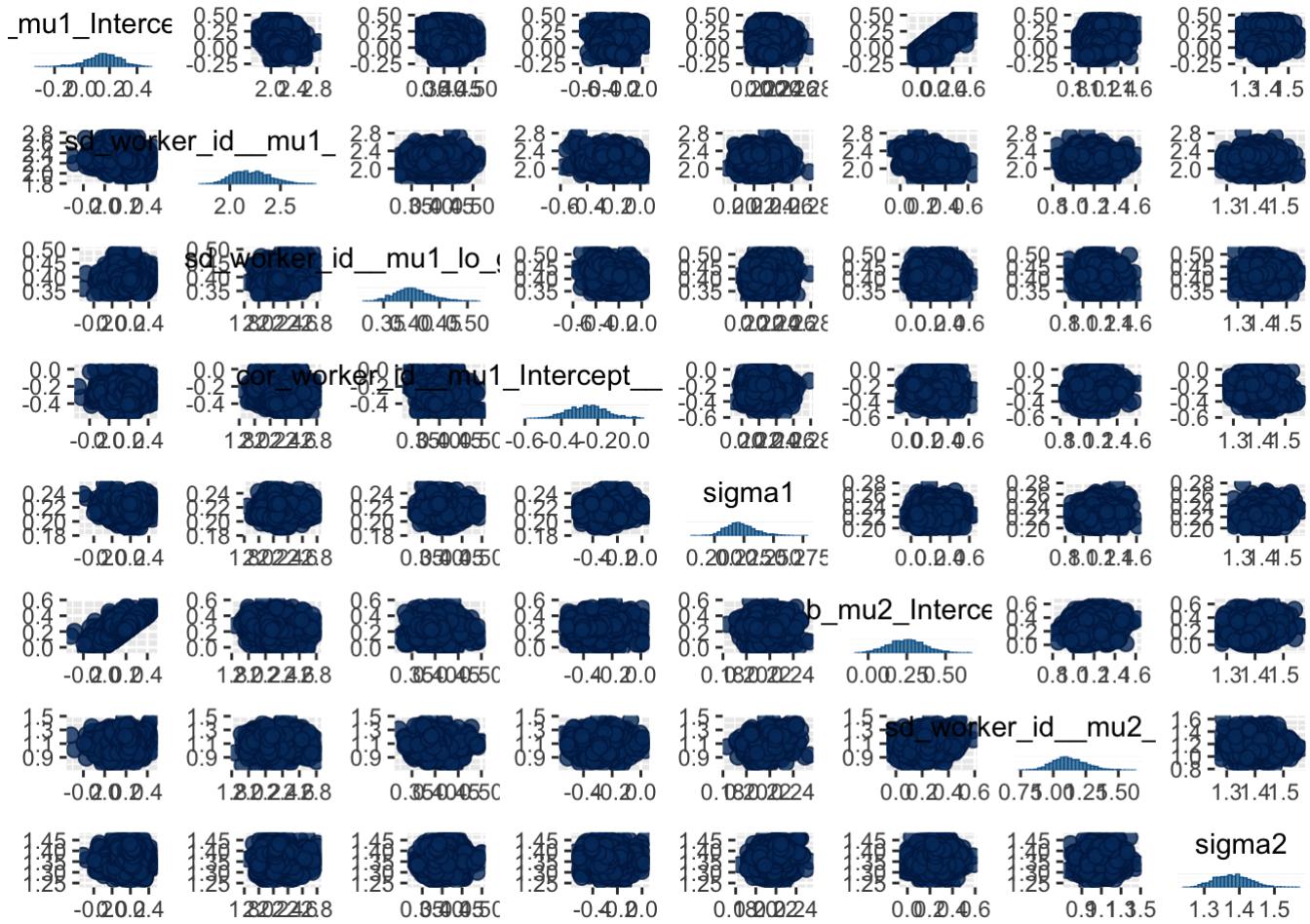


- Pairs plot

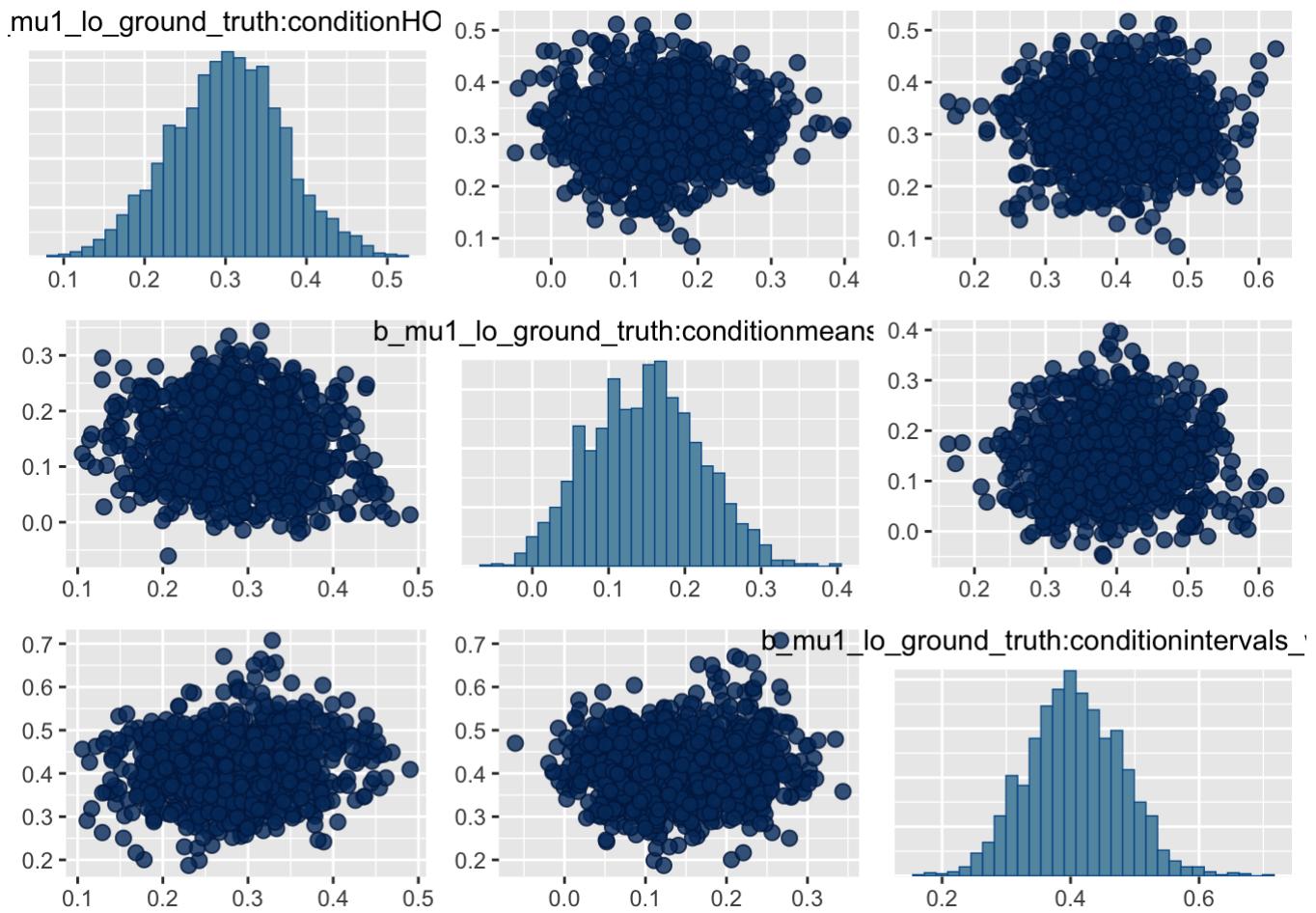
```
# pairs plot (too many things to view at once, so we've grouped them)
# mixture proportions
pairs(m.vis.wrkr.llo_mix2, pars = c("sd_worker_id_theta2_Intercept",
                                     "b_theta2_conditionHOPs",
                                     "b_theta2_conditionmeans_only",
                                     "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_mix2, 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_mix2, pars = c("b_mu1_lo_ground_truth:conditionHOPs",
                                    "b_mu1_lo_ground_truth:conditionmeans_only",
                                    "b_mu1_lo_ground_truth:conditionintervals_w_means"))
```



- Summary

```
# model summary
print(m.vis.wrkr.llo_mix2)
```

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

```

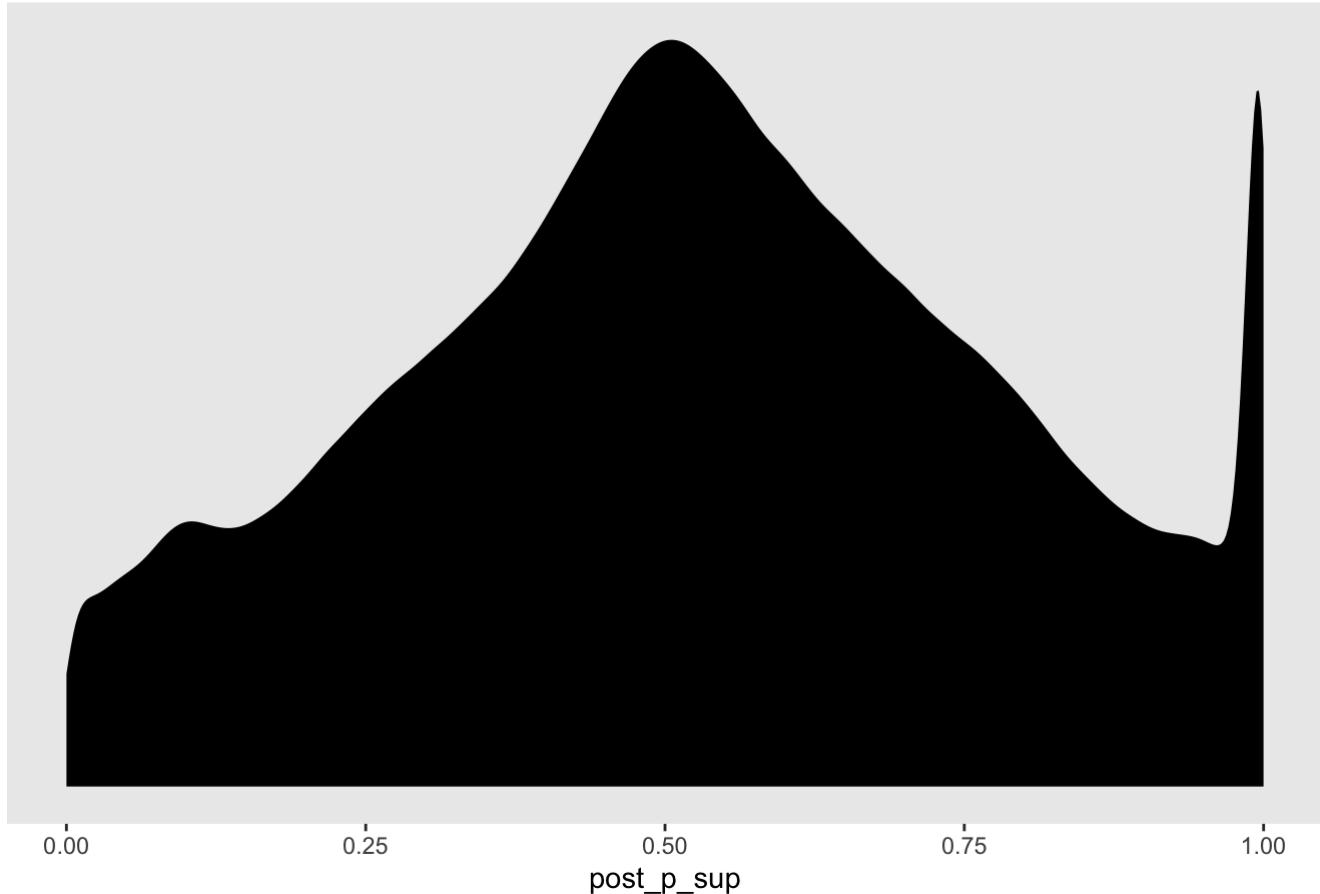
## Family: mixture(gaussian, gaussian)
## Links: mu1 = identity; sigma1 = identity; mu2 = identity; sigma2 = identity; thetal = identity; theta2 = identity
## Formula: lo_p_sup ~ 1
##           mu1 ~ (1 + lo_ground_truth | worker_id) + lo_ground_truth:condition
##           mu2 ~ (1 | worker_id)
##           theta2 ~ (1 | worker_id) + 0 + condition
## Data: model_df_llo (Number of observations: 2616)
## Samples: 2 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 2000
##
## Group-Level Effects:
## ~worker_id (Number of levels: 109)
##                               Estimate Est.Error l-95% CI
## sd(mu1_Intercept)            2.22    0.18    1.91
## sd(mu1_lo_ground_truth)     0.40    0.03    0.35
## sd(mu2_Intercept)           1.12    0.12    0.91
## sd(theta2_Intercept)        2.44    0.29    1.93
## cor(mu1_Intercept,mu1_lo_ground_truth) -0.26    0.10   -0.45
##                                         u-95% CI Eff.Sample Rhat
## sd(mu1_Intercept)            2.60      5 1.14
## sd(mu1_lo_ground_truth)     0.47    151 1.03
## sd(mu2_Intercept)           1.39      8 1.10
## sd(theta2_Intercept)        3.05    299 1.02
## cor(mu1_Intercept,mu1_lo_ground_truth) -0.05    254 1.01
##
## Population-Level Effects:
##                               Estimate Est.Error l-95% CI
## mu1_Intercept                0.16    0.12   -0.12
## mu2_Intercept                0.25    0.11    0.04
## mu1_lo_ground_truth:conditionHOPs 0.30    0.07    0.16
## mu1_lo_ground_truth:conditionintervals_w_means 0.41    0.07    0.27
## mu1_lo_ground_truth:conditionmeans_only 0.15    0.07    0.01
## theta2_conditionHOPs         0.06    0.42   -0.79
## theta2_conditionintervals_w_means -1.08    0.42   -1.90
## theta2_conditionmeans_only   -1.47    0.42   -2.35
##                                         u-95% CI Eff.Sample Rhat
## mu1_Intercept                 0.39     67 1.00
## mu2_Intercept                 0.48    119 1.01
## mu1_lo_ground_truth:conditionHOPs 0.44     89 1.04
## mu1_lo_ground_truth:conditionintervals_w_means 0.55    157 1.02
## mu1_lo_ground_truth:conditionmeans_only 0.29    146 1.00
## theta2_conditionHOPs         0.87     76 1.03
## theta2_conditionintervals_w_means -0.23    23 1.05
## theta2_conditionmeans_only   -0.63    364 1.01
##
## Family Specific Parameters:
##                               Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## sigma1       0.22      0.01     0.20     0.25      6 1.14
## sigma2       1.38      0.05     1.29     1.48      2 1.23
##
## 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_lllo %>%
  select(lo_ground_truth, condition, worker_id) %>%
  add_predicted_draws(m.vis.wrkr.lllo_mix2, 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())
```

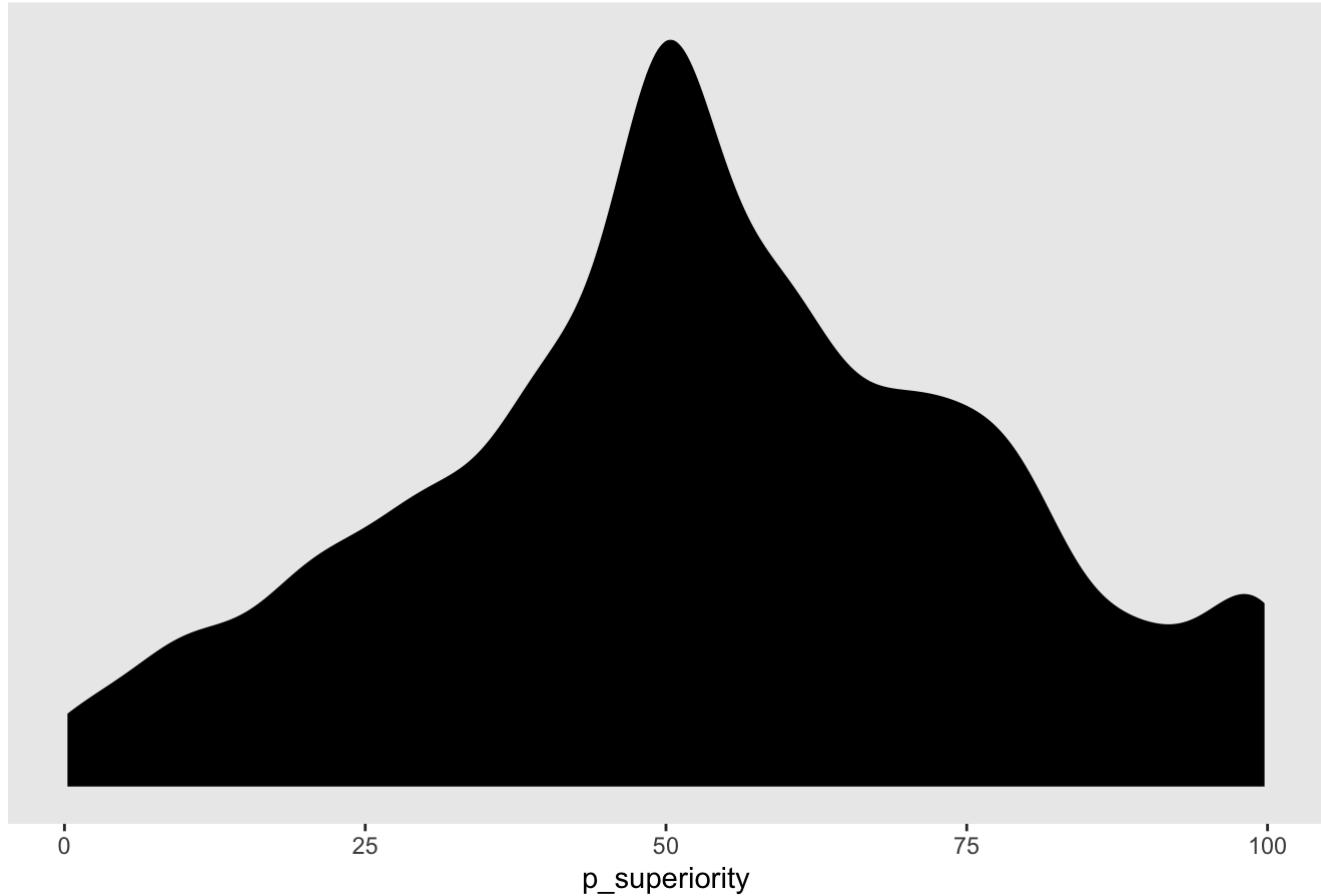
Posterior predictive distribution for probability of superiority



How does this compare to the empirical distribution of probability of superiority responses?

```
# posterior predictive check
model_df_lllo %>%
  ggplot(aes(x = p_superiority)) +
  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())
```

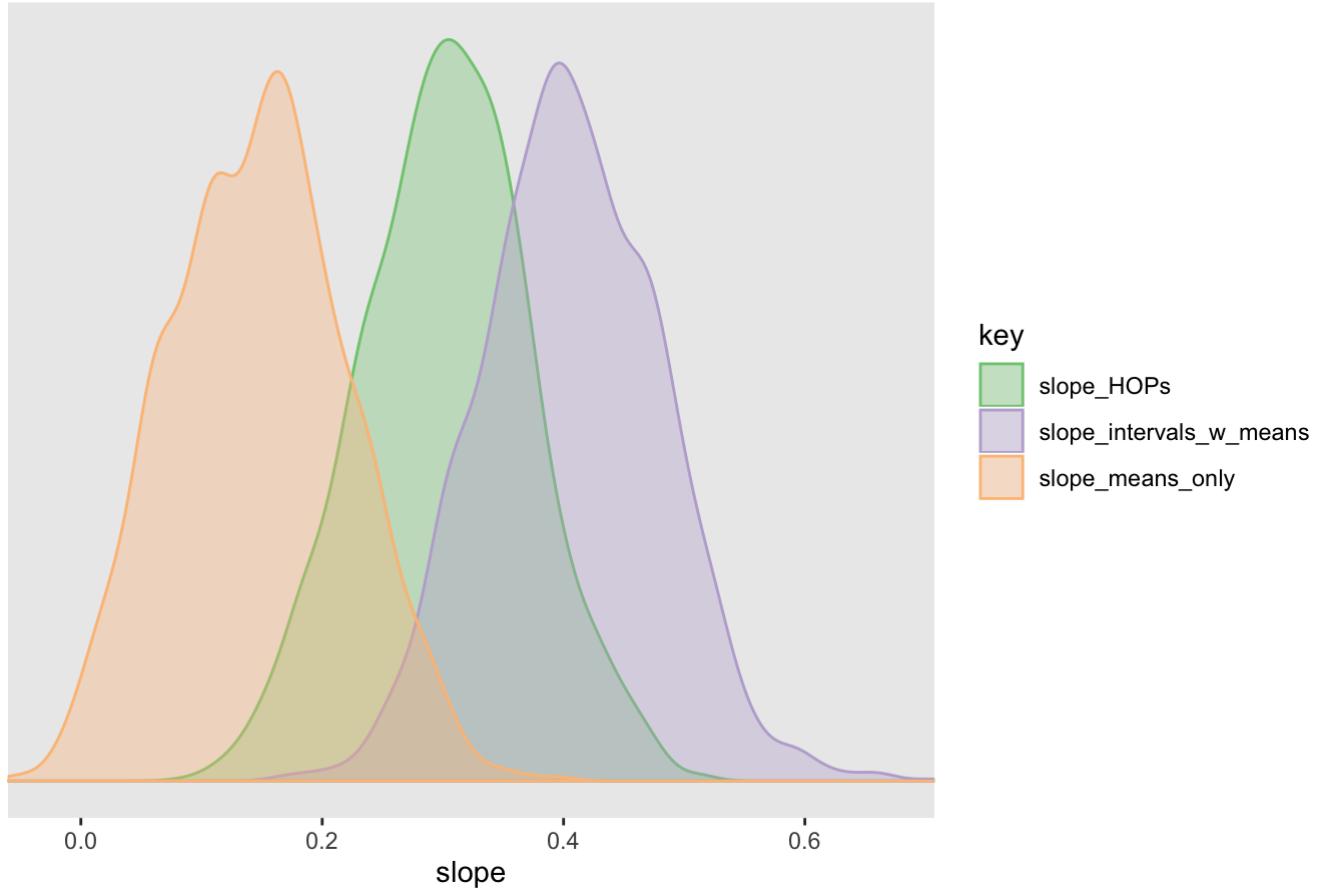
### Posterior predictive distribution for probability of superiority



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

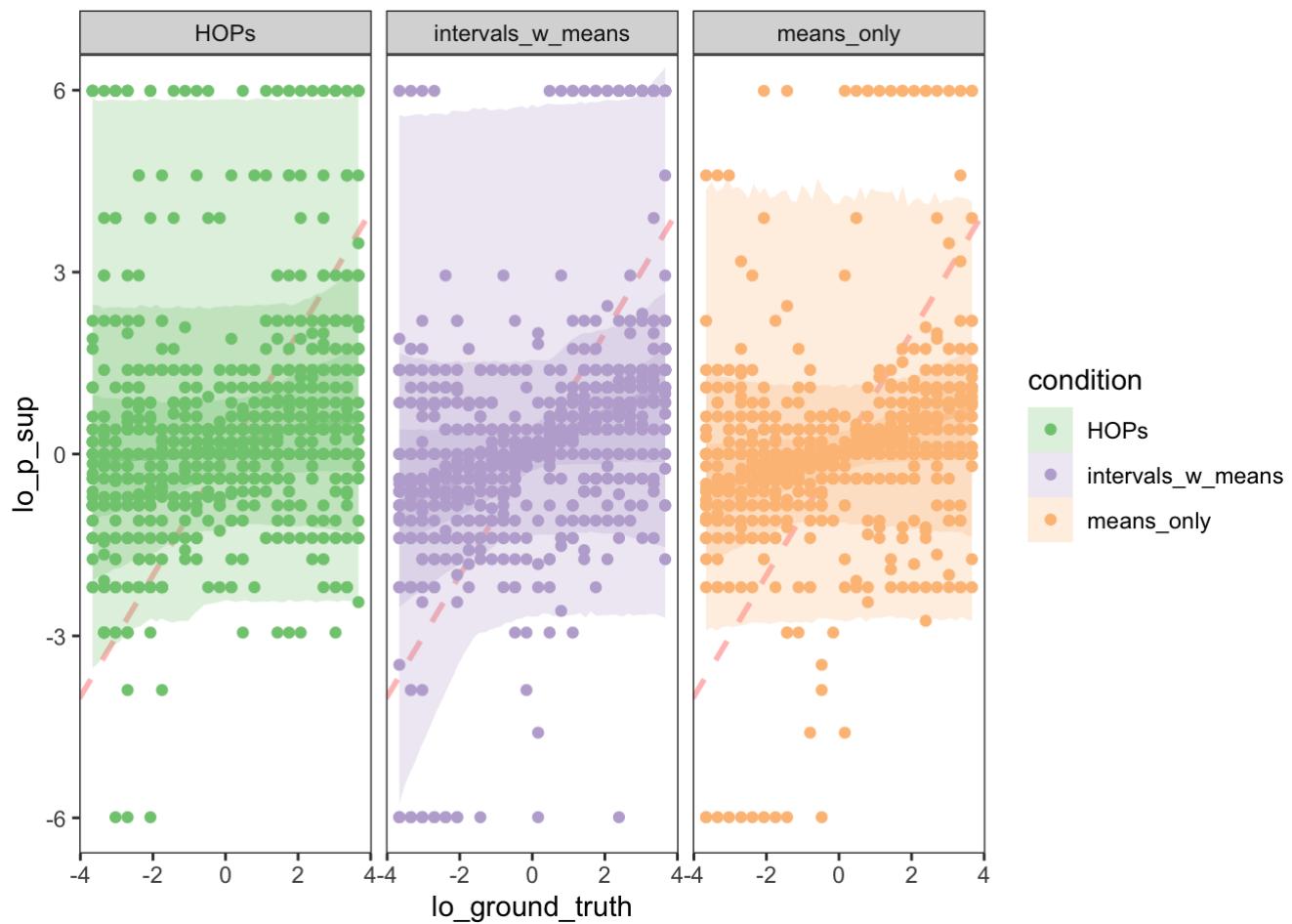
```
# use posterior samples to define distributions for the slope in each visualization condition
posterior_samples(m.vis.wrkr.ll0_mix2) %>%
  transmute(slope_HOPs = `b_mul_lo_ground_truth:conditionHOPs`,
            slope_intervals_w_means = `b_mul_lo_ground_truth:conditionintervals_w_means`,
            slope_means_only = `b_mul_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



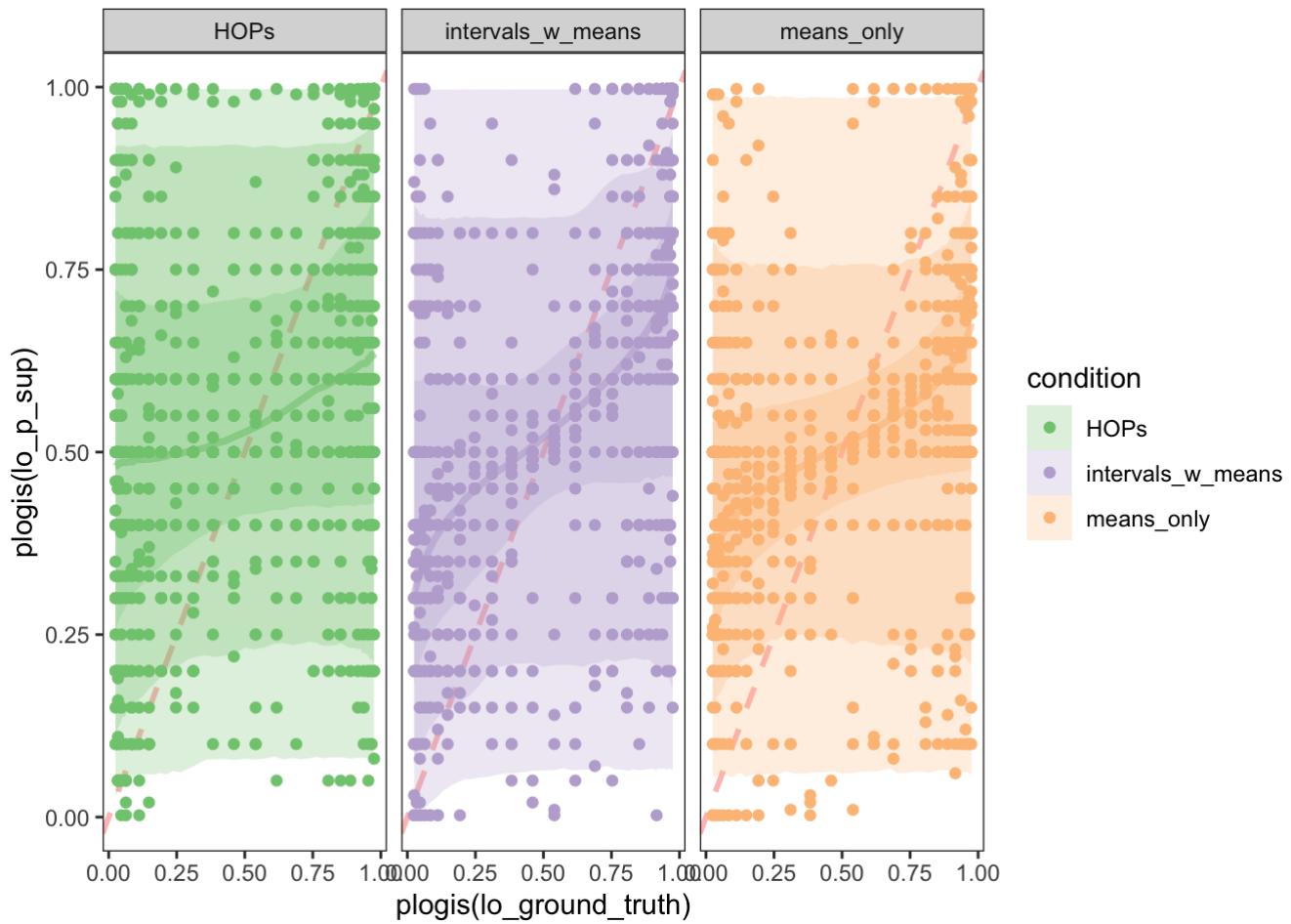
Let's take a look at some of the estimated linear models per visualization condition.

```
# this time we'll adopt functions from the tidybayes package to make plotting posterior
or predictions easier
model_df_lllo %>%
  group_by(condition, worker_id) %>%
  data_grid(lo_ground_truth = seq_range(lo_ground_truth, n = 51)) %>%
  add_predicted_draws(m.vis.wrkr.lllo_mix2) %>%
  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_lllo) +
  scale_fill_brewer(type = "qual", palette = 1) +
  scale_color_brewer(type = "qual", palette = 1) +
  coord_cartesian(xlim = quantile(model_df_lllo$lo_ground_truth, c(0, 1)),
                  ylim = quantile(model_df_lllo$lo_p_sup, c(0, 1))) +
  theme_bw() +
  theme(panel.grid = element_blank()) +
  facet_grid(. ~ condition)
```



What does this look like in probability units?

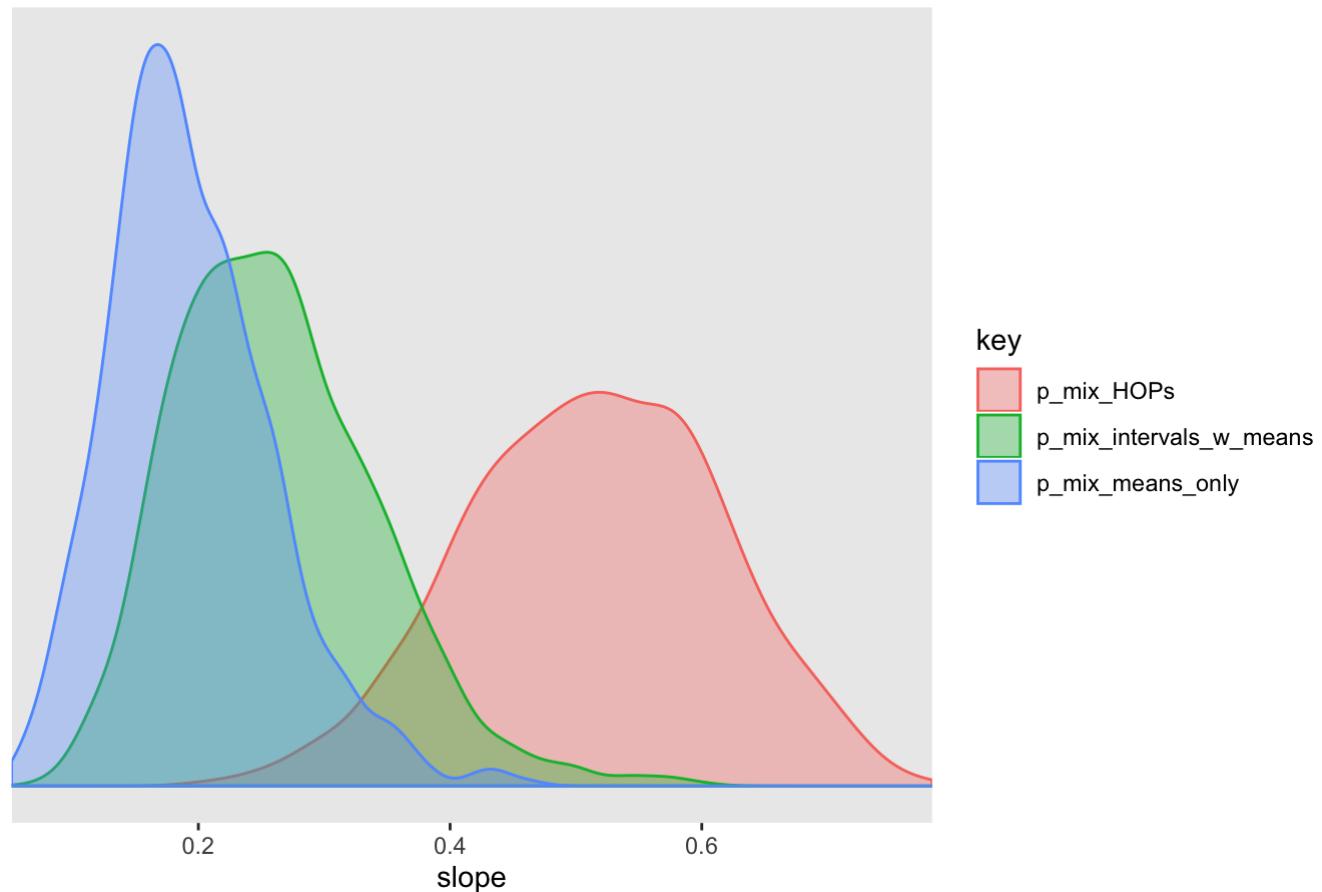
```
model_df_ll0 %>%
  group_by(condition, worker_id) %>%
  data_grid(lo_ground_truth = seq_range(lo_ground_truth, n = 51)) %>%
  add_predicted_draws(m.vis.wrkr.ll0_mix2) %>%
  ggplot(aes(x = plogis(lo_ground_truth), y = plogis(lo_p_sup), color = condition, fill = condition)) +
  geom_abline(intercept = 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_ll0) +
  scale_fill_brewer(type = "qual", palette = 1) +
  scale_color_brewer(type = "qual", palette = 1) +
  coord_cartesian(xlim = quantile(plogis(model_df_ll0$lo_ground_truth), c(0, 1)),
                  ylim = quantile(plogis(model_df_ll0$lo_p_sup), c(0, 1))) +
  theme_bw() +
  theme(panel.grid = element_blank()) +
  facet_grid(. ~ condition)
```



What about the mixture proportions? Let's plot 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.vis.wrkr.llo_mix2) %>%
  transmute(
    #p_mix_HOPs = plogis(b_theta2_Intercept),
    p_mix_HOPs = plogis(b_theta2_conditionHOPs),
    p_mix_intervals_w_means = plogis(b_theta2_conditionintervals_w_means),
    p_mix_means_only = plogis(b_theta2_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 proportion of constant response by visualization condition") +
  theme(panel.grid = element_blank())
```

## Posterior for proportion of constant response by visualization condition



All of this looks good!