In this document, we run the rational framework on the effect size judgment experiment in Kale et al. (2020). We first use two statistical models to predict behavioral agents' probability of superiority (PoS) response distribution and their incentivized decision distribution. We use the models from Kale et al. to generate these predictions.

In the effect size judgment experiment, we consider two kinds of behavioral agents. The first kind reports their belief of PoS, and the second kind directly reports their decision (whether to hire the new player in the context of Kale et al.'s experimental task). The framework can be thought of as a function that estimates five parameters under the assumptions of the experienetal design described in Kale et al. (including target sample size):

1. The *rational baseline* ($Rprior$): the performance of the rational agent without access to the visualization signals, i.e., with prior beliefs only.
2. The *rational benchmark* (R): the performance of the rational agent with access to the signal (provided by the visualization), i.e., with posterior beliefs.
3. The *behavioral decision score* (B): the expected score of the behavioral agent, who reports decisions, predicted by generative statistic model.
4. The *PoS raw score* (B^Q): the expected score of the behavioral agent, who reports PoS, predicted by a generative statistic model.
5. The *calibrated PoS score* (C^Q): the score of a rational agent on information structure π^B of the behavioral agent who reports PoS.

Generate Behavioral Data

Load in Data

Read in experimental data used to fit the models, obtained from https://github.com/kalealex/effect-size-jdm.

```
# read in data
model_df <- read_csv("./data/model-data.csv")
```

```
## Rows: 19892 Columns: 41
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (8): worker_id, condition, gender, age, education, chart_use, strategy_...
## dbl (29): batch, n_trials, n_data_conds, baseline, es_threshold, award_value...
## lgl  (4): start_means, outcome, means, exclude
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# preprocessing
model_df <- model_df %>%
  mutate(
    # factors for modeling
    means = as.factor(means),
    start_means = as.factor(start_means),
    sd_diff = as.factor(sd_diff),
    condition = factor(condition, levels = c("densities","intervals", "HOPs", "QDPs")), # reorder
    # evidence scale for decision model
    p_diff = p_award_with - (p_award_without + (1 / award_value)),
    evidence = qlogis(p_award_with) - qlogis(p_award_without + (1 / award_value))
  )
```

Models

Then we refit the models in Kale et al.

```
# hierarchical linear log odds model
m.p_sup <- brm(data = model_df, family = "gaussian",
             formula = bf(lo_p_sup ~  (1 + lo_ground_truth*trial + means*sd_diff|worker_id) + lo_ground
                          sigma ~ (1 + lo_ground_truth + trial|worker_id) + lo_ground_truth*condition*tr
             prior = c(prior(normal(1, 0.5), class = b),
                       prior(normal(1.3, 1), class = Intercept),
                       prior(normal(0, 0.15), class = sd, group = worker_id),
                       prior(normal(0, 0.3), class = b, dpar = sigma),
                       prior(normal(0, 0.15), class = sd, dpar = sigma),
                       prior(lkj(4), class = cor)),
             iter = 12000, warmup = 2000, chains = 2, cores = 2, thin = 2,
             control = list(adapt_delta = 0.99, max_treedepth = 12),
             file = "models/llo_mdl")
```

```
m.decisions <- brm(
  data = model_df, family = bernoulli(link = "logit"),
  formula = bf(intervene ~ (1 + evidence*means*sd_diff + evidence*trial|worker_id) + evidence*means*sd_c
  prior = c(prior(normal(0, 1), class = Intercept),
            prior(normal(1, 1), class = b, coef = evidence),
            prior(normal(0, 0.5), class = b),
            prior(normal(0, 0.5), class = sd),
            prior(lkj(4), class = cor)),
  iter = 8000, warmup = 2000, chains = 2, cores = 2, thin = 2,
  file = "models/logistic_mdl")
```

We generate the data generate the behavioral agents' belief of PoS and decisions by sampling from the posterior predictive distribution of the models. We take 500 draws from the posterior predictive distribution for each combination of ground truth effect size, means level (present or absent), sd_diff level (5 or 15), condition (of 4 visualization types), trial number (centered in log space), and block order (start_means TRUE or FALSE). We will use the dataframe of predictions to later simulate the experiment.

```
llo_model_df <- model_df %>%
  data_grid(lo_ground_truth, means, sd_diff, condition, trial, start_means) %>%
  add_predicted_draws(m.p_sup, re_formula = NA, n = 500) %>%
  mutate(est_error = plogis(.prediction) - plogis(lo_ground_truth))
```

```
## Warning:
## In add_predicted_draws(): The `n` argument is a deprecated alias for `ndraws`.
## Use the `ndraws` argument instead.
## See help("tidybayes-deprecated").
```

```
logistic_model_df <- model_df %>%
  data_grid(evidence, means, sd_diff, condition, trial, start_means) %>%
  add_predicted_draws(m.decisions, re_formula = NA, ndraws = 5000, seed = 1234)
```

Rational Agent Framework

Setup

We define helper functions needed for the rational agent calculations.

```
inv_log <- function(a) {
  # Computes the inverse of the logit function
```

```r
  return(1.00 / (1.00 + exp(0.00 - as.numeric(a))))
}

qlogis <- function(a) {
  # Computes the logit function
  return(log(a / (1.00 - a)))
}

binning <- function(bin_size) {
  # Computes the empirical distribution for each combination of condition, mean and ground truth
  # The freq[i, j, k, h] array records the frequency of i-th vis condition, j-th mean (1 for without me
  # k-th ground truth, and h-th PoS report (binned into 100 discrete levels).
  freq <- array(0, c(4, 2, 8, 100))
  freq_tibble = tibble(
    condition = match(condition_data, condition_dict),
    means = match(means_data, means_dict),
    ground_truth = ground_truth_data,
    behavioral_pos_reports=as.integer(inv_log(behavioral_pos_reports) / bin_size)
  ) %>%
    # counting condition, means, groundtruth, the response numbers, empirical distribution
    count(condition, means, ground_truth, behavioral_pos_reports)

  for (i in 1:nrow(freq_tibble)) {
    freq[freq_tibble[i, "condition"][[1]],
         freq_tibble[i, "means"][[1]],
         freq_tibble[i, "ground_truth"][[1]],
         freq_tibble[i, "behavioral_pos_reports"][[1]] + 1] <- freq_tibble[i, "n"][[1]]
  }
  return(freq)
}

compute_freq <- function(bin_num, bin_size) {
  # Computes the frequency given bin number and bin size
  return((bin_num + 0.5) * bin_size)
}

eq_err <- function(a, b) {
  # Compares if two values are equal within a margin of error 'eps'
  eps = 1e-5
  if (abs(a - b) <= eps) {
    return(TRUE)
  }
  return(FALSE)
}

rand_draw <- function(freq) {
  # Draws samples from the frequency distribution
  lst <- 0:99
  sample <- sample(lst, size = rand_size, prob = freq, replace = TRUE)
  freq_tmp <- array(0, c(1, 100))
  for (i in sample) {
    freq_tmp[1, i + 1] <- freq_tmp[1, i + 1] + 1
  }
  return(freq_tmp)
}
```

Scoring rule

We define the scoring rule for PoS reports and decision reports respectively.

```r
pos_score <- function(report, pos_truth) {
  # Computes the expected quadratic score given a report and the position of the ground truth
  if (is.numeric(pos_truth)) {
    tmp_ground_truth <- lo_ground_truth[pos_truth + 1]
  } else {
    tmp_ground_truth <- lo_ground_truth[unlist(pos_truth) + 1]
  }

  # we infer the decision from PoS in rational agent's way
  return(decision_score(prior_freq <= report, tmp_ground_truth))
}

decision_score <- function(decision, truth) {
  # Computes the decision score given a decision and the ground truth
  return(decision * (3.17 * truth - 1.00) + (1 - decision) * 3.17 * 0.5)
}
```

Preparation

We first load data and prepare with some constants for simulation.

```r
# The size of sample for each behavioral score calculation
rand_size <- 160 * 2

# Dictionaries used to map categorical variables (condition and means) to integers
condition_dict <- c("densities", "intervals", "HOPs", "QDPs")
means_dict <- c("FALSE", "TRUE")

# Dictionaries used to map continous variable (ground truth for PoS) to integers
lo_ground_truth_dict = unique(llo_model_df$lo_ground_truth)

# The total number of ground truth
n_ground_truth = length(lo_ground_truth_dict)

# compute probability of winning from pos
lo_ground_truth <- pnorm(sqrt(2) * qnorm(inv_log(lo_ground_truth_dict)))

# read columns from pos dataframe (mapping ground truth data into integers)
ground_truth_data <- match(llo_model_df$lo_ground_truth, lo_ground_truth_dict)
behavioral_pos_reports <- llo_model_df$.prediction
means_data <- llo_model_df$means
condition_data <- llo_model_df$condition
```

We use binning here to transform the belief on continous scale to discrete levels. While interpolation is more oftern to used here to infer a density function of PoS, we choose binning based on the sparse distribution of PoS reports, where interpolation can't recover the data well.

```r
bin_size <- 0.05
freq <- binning(bin_size)
```

Rational Baseline and Benchmark

We then calculate the rational agent's score with only prior knowledge and with both prior and posterior knowledge.

```r
# compute prior - np.sum(freq[1, 1], axis = 1) is the number of trials with each ground truth, lo_groun
prior_freq <- sum(rowSums(freq[1, 1, , ]) * lo_ground_truth) / sum(freq[1, 1, , ])

# run the analysis
prior_score <- sum(rowSums(freq[1, 1, , ]) * pos_score(prior_freq, 0:(n_ground_truth - 1))) / sum(freq[
cat("prior score: ", prior_score, "\n")
```

```
## prior score:  1.566376
```

```r
posterior_score <- sum(rowSums(freq[1, 1, , ]) * pos_score(lo_ground_truth, 0:(n_ground_truth - 1))) /
cat("posterior score: ", posterior_score, "\n")
```

```
## posterior score:  1.768289
```

Bevaioral and calibrated score (reporting PoS)

We calculate, when they report their belief about PoS, behavioral agents' score and the calibrated one.

```r
score_ind <- c()
score_behav <- c()
score_calib <- c()

calc_score <- function(cond_posterior, freq_tmp, bin_size) {
  # Calculates behavioral and calibrated behavioral scores given frequency and bin size
  behavioral_score <- 0.0
  for (i in 1:n_ground_truth) {
    behavioral_score <- behavioral_score + sum(freq_tmp[i, ] * pos_score(compute_freq(0:99, bin_size),
  }



  # calib_posterior is rational agent's PoS with belief as cond_posterior
  calib_posterior <- lo_ground_truth %*% cond_posterior
  calib_behav_score <- 0.00
  # we calculate the score of calibrated behavioral (rational agent with only behavioral agents' report
  for (i in 1:n_ground_truth) {
    calib_behav_score <- calib_behav_score + sum(freq_tmp[i, ] * pos_score(calib_posterior, i - 1)) / s
  }
  return(list(behavioral_score, calib_behav_score))
}

n_round <- 100
# loop over 4 vis conditions and 2 means (with and without)
for (t in 1:4) {
  for (m in 1:2) {
    tmp_behav <- c()
    tmp_calib <- c()

      # cond_posterior is the rational agent's belief over ground truth after viewing behavioral agents
    freq_tmp <- rand_draw(freq[t, m, 1, ])
    for (j in 2:n_ground_truth) {
      freq_tmp <- rbind(freq_tmp, rand_draw(freq[t, m, j, ]))
    }

    cond_posterior <- array(0, c(n_ground_truth, 100))
    for (i in 1:n_ground_truth) {
      for (j in 1:100) {
        if (!eq_err(freq_tmp[i, j], 0.0)) {
          cond_posterior[i, j] <- freq_tmp[i, j] / sum(freq_tmp[, j])
        }
      }
    }
    # run n_round to generate n_round samples and calculate a score for each sample
    for (i in 1:n_round) {
      freq_tmp <- rand_draw(freq[t, m, 1, ])
      for (j in 2:n_ground_truth) {
        freq_tmp <- rbind(freq_tmp, rand_draw(freq[t, m, j, ]))
      }
      behav_tmp <- calc_score(cond_posterior, freq_tmp, bin_size)
      tmp_behav <- c(tmp_behav, behav_tmp[[1]])
      tmp_calib <- c(tmp_calib, behav_tmp[[2]])
    }

    score_ind <- c(score_ind, rep((t - 1) * 4 + m, n_round))
    score_behav <- c(score_behav, tmp_behav)
    score_calib <- c(score_calib, tmp_calib)
  }
}
```

Behavioral score (reporting decisions)

This is how we calculate the behavioral score when they directly reporting their decisions.

Preparation Similar to PoS, we do some preparation for some helper arrays and calculate the belief over visualization conditions, mean condition, ground truth, and decision reports.

```r
# read the data of evidence
tmp_decision_ground_truth_dict = unique(logistic_model_df$evidence)
# transform the evidence into winning probability
trans_tmp_decision_ground_truth_dict = inv_log(tmp_decision_ground_truth_dict + qlogis(0.5 + 1.0 / 3.17

decision_ground_truth_dataframe = data.frame(matrix(ncol=2,nrow=0,
                                         dimnames=list(NULL, c("evidence", "index"))))
decision_ground_truth_dict = c()
decision_ground_truth_count = 0
# This loop eliminates the values that are close to others (difference less than eps) in tmp_decision_g
# then puts the final values into decision_ground_truth_dict
for (i in 1:length(tmp_decision_ground_truth_dict)) {
  flag = TRUE
  if (nrow(decision_ground_truth_dataframe) > 0) {
    for (j in 1:nrow(decision_ground_truth_dataframe)) {
      gt = decision_ground_truth_dataframe[j, "evidence"]
      index = decision_ground_truth_dataframe[j, "index"]
      if (eq_err(as.numeric(gt), tmp_decision_ground_truth_dict[i])) {
        flag = FALSE
        decision_ground_truth_dataframe[nrow(decision_ground_truth_dataframe) + 1, ] =
          c(tmp_decision_ground_truth_dict[i], index)
      }
    }
  }
  if (flag) {
    decision_ground_truth_count = decision_ground_truth_count + 1
    decision_ground_truth_dataframe[nrow(decision_ground_truth_dataframe) + 1, ] =
      c(tmp_decision_ground_truth_dict[i], decision_ground_truth_count)
    decision_ground_truth_dict = c(decision_ground_truth_dict, trans_tmp_decision_ground_truth_dict[i])
  }
}
decision_ground_truth_dataframe = decision_ground_truth_dataframe %>% transform(evidence = as.numeric(ev
                                                                             index = as.numeric(index))

# calculate the decision_freq
# The decision_freq[i, j, k, h] array records the frequency of i-th vis condition, j-th mean (1 for wit
# k-th ground truth, and h-th decision report (whether to hire new player).
decision_freq <- array(0, c(4, 2, 8, 2))
decision_freq_tibble = tibble(
  condition = match(logistic_model_df$condition, condition_dict),
  means = match(logistic_model_df$means, means_dict),
  ground_truth = (logistic_model_df %>% dplyr::select(evidence) %>% merge(decision_ground_truth_datafran
  behavioral_decision_reports=as.integer(logistic_model_df$.prediction)
) %>%
  # counting condition, means, groundtruth, the response numbers, empirical distribution
  count(condition, means, ground_truth, behavioral_decision_reports)
```

```
## Adding missing grouping variables: `means`, `sd_diff`, `condition`, `trial`,
## `start_means`, `.row`
```

```
for (i in 1:nrow(decision_freq_tibble)) {
  decision_freq[decision_freq_tibble[i, "condition"][[1]],
       decision_freq_tibble[i, "means"][[1]],
       decision_freq_tibble[i, "ground_truth"][[1]],
       decision_freq_tibble[i, "behavioral_decision_reports"][[1]] + 1] <- decision_freq_tibble[i, "n"]
}
```

Calculation We then calculate the behavioral score when they directly report the decisions.

```
score_decision <- c()
n_round <- 100
# loop over 4 vis conditions and 2 mean conditions
for (t in 1:4) {
  for (m in 1:2) {
    tmp_decision <- c()
    # run n_round to generate n_round samples to represent the uncertainty of behavioral agents
    for (i in 1:n_round) {
      score_tmp <- 0.0
      for (j in 1:decision_ground_truth_count) {
        sample <- sample(c(0, 1), size = rand_size, prob = decision_freq[t, m, j, ] / sum(decision_freq
        freq_tmp <- sum(sample) / rand_size
        score_tmp <- score_tmp + decision_score(freq_tmp, decision_ground_truth_dict[j])
      }
      score_tmp <- score_tmp / decision_ground_truth_count
      tmp_decision <- c(tmp_decision, score_tmp)
    }
    score_decision <- c(score_decision, tmp_decision)
  }
}
```

Save results

We save the results to data frame for visualization.

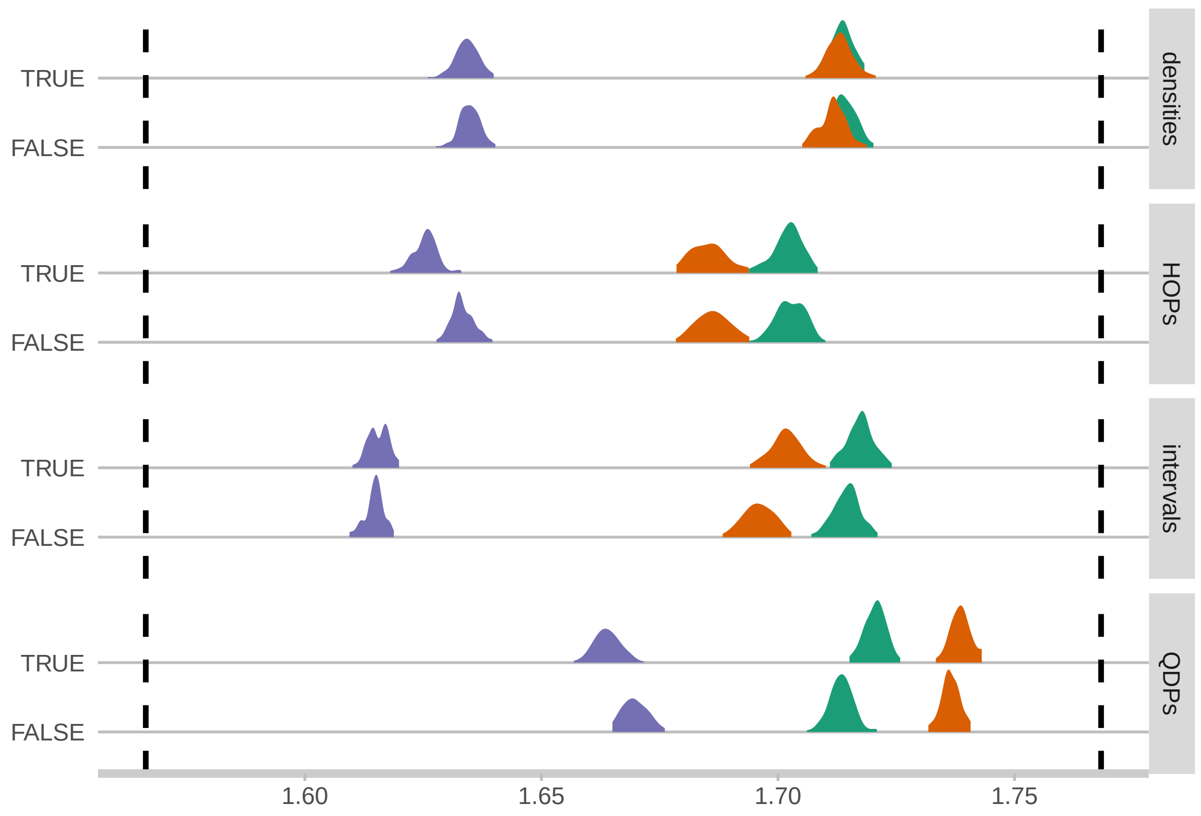
```
# Save results to csv files
all_behavioral <- data.frame(
  vis = condition_dict[unlist(lapply(score_ind, function(x) (x %/% 4) + 1))],
  mean = means_dict[unlist(lapply(score_ind, function(x) x %% 4))],
  behavioral = score_behav,
  calibrated_behavioral = score_calib,
  behavioral_decision = score_decision
)
all_rational <- data.frame(
  "rational"= c(prior_score, posterior_score)
)
```

Results

We show the results by visualization.

```
ggplot() +
  stat_slab(data = all_behavioral, aes(y = mean, x = behavioral), fill = "#7570b3", point_size=1.5) +
  stat_slab(data = all_behavioral, aes(y = mean, x = behavioral_decision), fill = "#1b9e77", point_size
  stat_slab(data = all_behavioral, aes(y = mean, x = calibrated_behavioral), fill = "#d95f02", point_si
  geom_vline(xintercept = all_rational[[1]][1], linetype = "dashed", size = 1) +
  geom_vline(xintercept = all_rational [[1]][2], linetype = "dashed", size = 1) +
  labs(x = "", y = "") +
  facet_grid(rows = vars(vis)) +
  theme(panel.grid.major.x = element_blank(),
        panel.grid.minor.x = element_blank(),
        panel.grid.major = element_line(colour = "grey"),
        axis.line.x = element_line(linewidth = 1.5, colour = "grey80"),
        panel.background = element_rect(fill = "white", color = "white"),
        axis.ticks.y = element_blank(),
        axis.ticks.x = element_line(colour = "grey"))
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```


TRUE
FALSE
densities
HOPs
intervals
QDPs
1.60
1.65
1.70
1.75