Interpersonal coordination in perception and memory

A. Paxton, T. J. H. Morgan, J. Suchow, & T. L. Griffiths

This R markdown provides the basis for our manuscript, "Interpersonal coordination in perception and memory" (Paxton, Morgan, Suchow & Griffiths, *in preparation*).

To run these analyses from scratch, you will need the following files:

- `./data/`: Contains experimental data. All data for included dyads are freely available in the OSF repository for the project: `https://osf.io/8fu7x/`.
- `./supplementary-code/required_packages-pmc.r`: Installs required libraries, if they are not already installed. **NOTE**: This should be run *before* running this script.
- `./supplementary-code/libraries_and_functions-pmc.r`: Loads in necessary libraries and creates new functions for our analyses.

Additional files will be created during the initial run that will help reduce processing time. Several of these files are available as CSVs from the OSF repository listed above.

Code written by: A. Paxton (University of California, Berkeley)

Date last modified: 26 October 2017

---

1 Data preparation

1.1 Preliminaries

```
# clear our workspace
rm(list=ls())

# read in libraries and create functions
source('./supplementary-code/required_packages-pmc.r')
source('./supplementary-code/libraries_and_functions-pmc.r')
```

1.2 Concatenate experiment files

```
# get list of individual experiments included in the data
experiment_files = list.dirs('./data', recursive=FALSE)

# concatenate the files
vector_files = data.frame()
info_files = data.frame()
questionnaire_files = data.frame()
node_files = data.frame()
for (experiment in experiment_files){
```

```
  # read in the next experiment's files and add ID to each
  exp_id = strsplit(as.character(experiment),"/|-")[[1]][3]
  next_vector = read.table(paste(experiment,'/vector.csv',sep=''), sep=',',
                           header=TRUE, stringsAsFactors = FALSE) %>%
    mutate(experiment = exp_id)
  next_info = read.table(paste(experiment,'/info.csv',sep=''), sep=',',
                         header=TRUE, stringsAsFactors = FALSE) %>%
    mutate(experiment = exp_id)
  next_q = read.table(paste(experiment,'/question.csv',sep=''), sep=',',
                           header=TRUE, stringsAsFactors = FALSE) %>%
    mutate(experiment = exp_id)
  next_node = read.table(paste(experiment,'/node.csv',sep=''), sep=',',
                           header=TRUE, stringsAsFactors = FALSE) %>%
    mutate(experiment = exp_id)

  # append to group files
  vector_files = rbind.data.frame(vector_files, next_vector)
  info_files = rbind.data.frame(info_files, next_info)
  questionnaire_files = rbind.data.frame(questionnaire_files, next_q)
  node_files = rbind.data.frame(node_files, next_node)

}
```

1.3 Identify dyads from vector data

In order to figure out which participants' nodes were connected to one another in dyads, we use the vectors created between nodes (excluding the stimulus-creating node). We then use that information to identify which stimuli were sent to which dyads.

```
# use the vectors connecting the nodes to identify pairs
vector_df = vector_files %>%

  # convert time to integer and winnow out unnecessary variables and nodes
  mutate(t = round(as.numeric(ymd_hms(creation_time)), 0)) %>%
  select(experiment, t, origin_id, destination_id, network_id) %>%
  dplyr::filter(!origin_id == 1) %>%

  # find pairs from vector files
  group_by(experiment, t) %>%
  mutate(min_id = pmin(origin_id,destination_id)) %>%
  mutate(max_id = pmax(origin_id,destination_id)) %>%
  ungroup() %>%

  # get unique pairs and number them
  select(-origin_id, -destination_id) %>%
  distinct() %>%
  mutate(dyad = seq_along(min_id)) %>%

  # gather the participants into a single column
  gather(key="id",value="participant", min_id, max_id) %>%
  select(-id)

# figure out which stimuli were sent to which dyads
```

```
dyad_df = info_files %>%
  mutate(t = round(as.numeric(ymd_hms(creation_time)), 0)) %>%
  dplyr::filter(origin_id == 1) %>%
  select(experiment, t, contents) %>%
  full_join(., vector_df,
            by = c('experiment', 't')) %>%
  select(-t)
```

1.4 Prepare dataframe

We now take the concatenated files and begin processing, including de-duplication of dataset.

The structure of the experiment sometimes led to near-duplicate rows to be sent to the server to manage partner communication. We must now identify these near-duplicates and strip them out. We can best identify these by using the `response_counter` variable: A properly de-duplicated dataset should have only 1 row per `response_counter` value in each trial for each participant.

```
info_df = info_files %>% ungroup() %>%

  # filter out stimulus nodes
  dplyr::filter(!origin_id == 1) %>%

  # convert time and get rid of unnecessary variables
  mutate(t = round(as.numeric(ymd_hms(creation_time)), 0)) %>%
  select(experiment, t, property3, origin_id, network_id, contents) %>%

  # read in `contents` as JSONs
  cbind(., jsonlite::stream_in(textConnection(.$contents))) %>%

  # rename a whole slew of variables
  dplyr::rename(participant = origin_id,
                trial_type = trialType,
                trial_number = trialNumber,
                guess_counter = guessCounter,
                response_counter = responseCounter,
                accept_type = acceptType,
                response_type = responseType) %>%

  # get rid of unnecessary variables and arrange rows
  select(-property3, -finalAccuracy, -contents) %>%
  dplyr::arrange(experiment, participant, trial_number, response_counter) %>%

  # remove the automatically generated infos that produced NAs in `guess`
  dplyr::filter(!is.na(guess)) %>%

  # determine uniqueness without considering time or response_type
  group_by(experiment, participant, network_id, trial_type,
           trial_number, guess_counter, response_counter) %>%
  summarise_all(first) %>%
  ungroup() %>%

  # replace NAs from guesses and calculate error with each guess
  mutate(guess = replace(guess, guess<0, NA)) %>%
```

```
  mutate(guess_error = length - guess) %>%

  # merge info dataframe with dyad number information
  full_join(., dyad_df,
            by = c('experiment', 'participant','network_id')) %>%
  dplyr::rename(stimulus_list = contents)
```

```
##
 Found 500 records...
 Found 1000 records...
 Found 1500 records...
 Found 2000 records...
 Found 2500 records...
 Found 3000 records...
 Found 3500 records...
 Found 4000 records...
 Found 4500 records...
 Found 5000 records...
 Found 5500 records...
 Found 6000 records...
 Found 6500 records...
 Found 7000 records...
 Found 7500 records...
 Found 8000 records...
 Found 8500 records...
 Found 9000 records...
 Found 9500 records...
 Found 10000 records...
 Found 10500 records...
 Found 11000 records...
 Found 11500 records...
 Found 12000 records...
 Found 12500 records...
 Found 13000 records...
 Found 13500 records...
 Found 14000 records...
 Found 14500 records...
 Found 15000 records...
 Found 15500 records...
 Found 15501 records...
 Imported 15501 records. Simplifying...
## Problematic rows identified (i.e., duplicates with differing accept types): 0
```

1.5 Identify and winnow down data to usable dyads

Next, we identify all dyads in which both participants responded the same number of times. This ensures that we include only dyads who experienced the full and correct experimental protocol.

```
# identify usable dyads
usable_dyads = info_df %>%

  # count the number of infos and trials per participant
  group_by(experiment,participant) %>%
```

```
  summarise(trials = max(trial_number),
            dyad = ifelse(length(unique(dyad)==1),
                          unique(dyad),
                          NA),
            infos = n()) %>%
  ungroup() %>%
  na.omit() %>%

  # count the infos sent by each participant in each dyad
  group_by(experiment, dyad) %>%
  mutate(participant = paste('p',(participant - min(participant)),sep='')) %>%
  spread(key = participant, value = infos) %>%
  ungroup() %>%
  mutate(difference_in_responses = abs(p1-p0)) %>%

  # filter out anyone who didn't have the same number of infos and who didn't complete 24 trials
  dplyr::filter(trials==24 & difference_in_responses==0) %>%
  na.omit()
```

Now that we've figured out which dyads we should use, let's winnow down the dataframe to include only those dyads.

```
# winnow and recorder columns
winnowed_info_df = info_df %>%
  dplyr::filter(dyad %in% usable_dyads$dyad) %>%
  mutate(t = round(t,-1)) %>%
  select(experiment, t, dyad, participant,
         trial_type, trial_number, response_counter, guess_counter, accept_type,
         length, guess, guess_error, response_type, network_id) %>%
  na.omit()

winnowed_info_df = unique(setDT(winnowed_info_df), by = c('experiment', 'dyad', 'participant',
         'trial_type', 'trial_number', 'response_counter', 'guess_counter', 'accept_type',
         'length', 'guess', 'guess_error', 'response_type', 'network_id'))
```

```
## Total dyads with strictly paired data for all trials: 34
```

For sanity, let's also check that everyone included in our winnowed dataset completed both training and test trials.

```
# ensure that everyone completed both training and test
only_one_trial_type = winnowed_info_df %>% ungroup() %>%
  select(experiment, participant, trial_type) %>%
  distinct() %>%
  group_by(experiment, participant) %>%
  summarize(n=n()) %>%
  dplyr::filter(n!=2)
```

```
## Included participants who did not undergo training and testing rounds: 0
```

1.6 Add questionnaire data

In the experiment's current form, different tables include different information, and some tables present the same information under different labels. This is true for questionnaire data. To accurately pair individuals'

guess data with their questionnaire responses, we match the `participant_id` variables in `node_df` and `question_df`, and we join the `id` variable in `node_df` with the `participant` variable in `info_df`.

```
# clean up questionnaire data by converting the stringified JSONs to a new variable
question_df = questionnaire_files %>% ungroup() %>%
  select(experiment, participant_id, response) %>%
  cbind(., jsonlite::stream_in(textConnection(.$response))) %>%
  select(-response)
```

```
##
 Found 148 records...
 Imported 148 records. Simplifying...
```

```
# clean up the node dataframe
node_df = node_files %>% ungroup() %>%
  select(experiment, participant_id, id) %>%
  na.omit()

# join questionnaire data wth infos and remove any participants whose survey data we don't have
winnowed_info_df = left_join(question_df, node_df,
                             by=c('experiment','participant_id')) %>%
  left_join(winnowed_info_df, .,
                            by=c('experiment','participant' = 'id')) %>%
  drop_na(cooperative_partner, cooperative_self, trust_partner, trust_self, engagement, difficulty)
```

```
# identify how many dyads have matching infos and complete questionnaire data
usable_question_dyads = winnowed_info_df %>% ungroup() %>%
  select(experiment, dyad, participant) %>%
  distinct() %>%
  group_by(experiment, dyad) %>%
  summarise(included_p = n()) %>%
  ungroup() %>%
  dplyr::filter(included_p==2)

# if needed, remove dyads who didn't have questionnaire data
winnowed_info_df = winnowed_info_df %>% ungroup() %>%
  dplyr::filter(dyad %in% usable_question_dyads$dyad)
```

```
## Total dyads with all guess and questionnaire data: 33
```

1.7 Create unique dyad and participant IDs across all experiments

Dallinger provides numeric IDs for each participant that are unique only within each experiment. Therefore, we create participant and dyad identifiers that are unique across the entire dataset.

```
# create unique dyad IDs
unique_dyad_ids = winnowed_info_df %>% ungroup() %>%
  select(experiment, dyad) %>%
  distinct() %>%
  mutate(unique_dyad = row_number())

# create unique participant IDs
unique_participant_ids = winnowed_info_df %>% ungroup() %>%
  select(experiment, participant) %>%
  distinct() %>%
```

```
  mutate(unique_participant = row_number())

# merge both into the main dataframe and rename
winnowed_info_df = right_join(unique_participant_ids, winnowed_info_df,
                              by=c('experiment', 'participant')) %>%
  right_join(unique_dyad_ids, ., by=c('experiment','dyad')) %>%
  dplyr::rename(original_participant = participant,
                original_dyad = dyad,
                participant = unique_participant,
                dyad = unique_dyad) %>%
  dplyr::arrange(experiment, participant, trial_number, response_counter)
```

1.8 Increment all counters by 1

Data were collected using Pythonic counters (i.e., starting from 0). We'll here update the dataframe to reflect R conventions (i.e., starting from 1).

```
winnowed_info_df = winnowed_info_df %>%
  mutate(trial_number = trial_number + 1) %>%
  mutate(response_counter = response_counter + 1) %>%
  mutate(guess_counter = guess_counter + 1)
```

1.9 Normalize error by maximum possible error

Because stimuli line lengths could range from 1-100, each trial provided a bound on the total possible guess error. As a result, we need to normalize each guess error by the maximum *possible* error for that trial.

```
winnowed_info_df = winnowed_info_df %>% ungroup() %>%
  mutate(normalized_error = guess_error/max(abs(100-length),abs(length-100)))
```

1.10 Create training accuracy metric

We next create a training metric that quantifies the *non-directional* improvement over the training rounds. Essentially, this captures the change in relative accuracy over training, regardless of whether participants began by over- or under-estimating line lengths.

```
# create a slope to see how quickly they improved
winnowed_info_df = winnowed_info_df %>% ungroup() %>%
  select(participant, trial_type, trial_number, normalized_error) %>%
  na.omit() %>%
  dplyr::filter(trial_type == 'train') %>%
  group_by(participant) %>%
  do(broom::tidy(lm(abs(.$normalized_error) ~ .$trial_number))) %>%
  dplyr::filter(term=='.$trial_number') %>%
  select(participant, estimate) %>%
  dplyr::rename(training_improvement = estimate) %>%
  left_join(winnowed_info_df, .,
            by='participant')
```

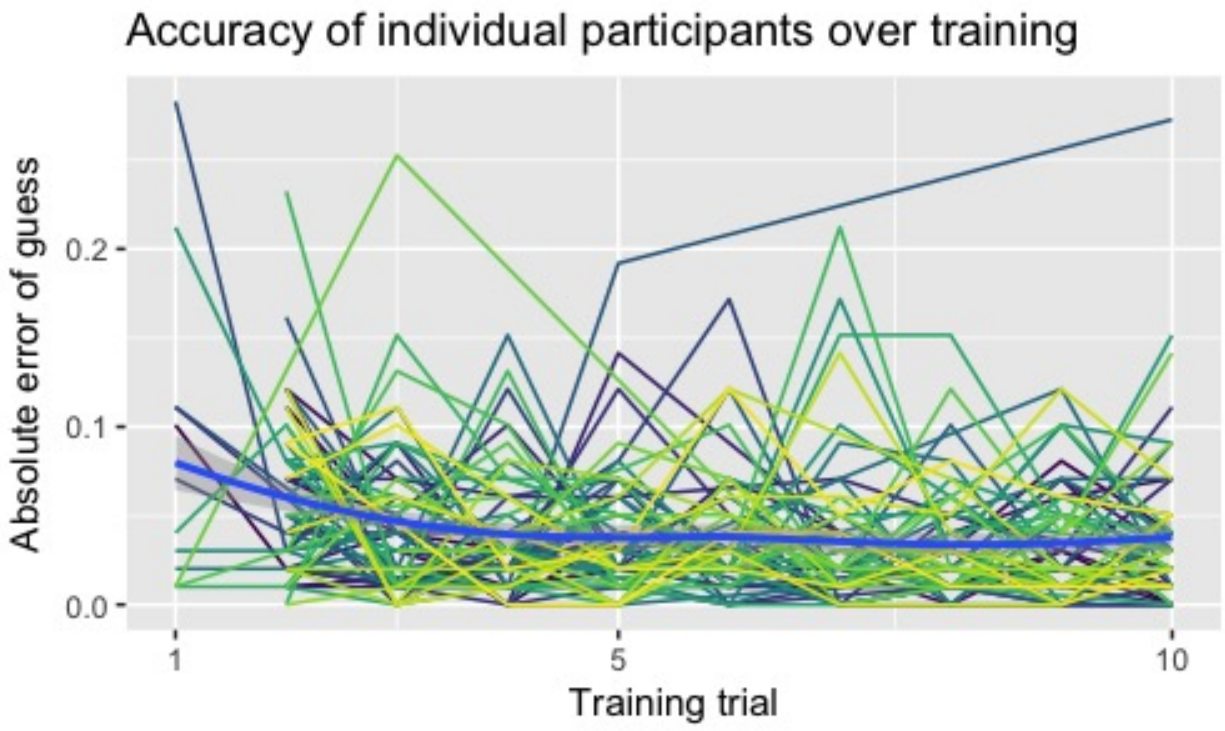

Figure 1: **Figure**. Included participants' absolute normalized error over all training trials and best-fit line (in blue).

1.11 Widen data to include partner's guess

```
# create a column for the partner's guess at that time
winnowed_info_df = winnowed_info_df %>% ungroup() %>%

  # create participant binary values
  group_by(experiment, dyad) %>%
  mutate(partner_id = (min(participant)+max(participant)) - participant) %>%
  mutate(self_id = participant) %>%
  ungroup() %>%

  # gather into multiple values
  select(self_id, partner_id, normalized_error, trial_number, response_counter) %>%
  dplyr::rename(partner_error = normalized_error) %>%
  distinct() %>%

  # merge
  left_join(winnowed_info_df, .,
            by=c('participant'='self_id',
                 'trial_number',
                 'response_counter'))
```

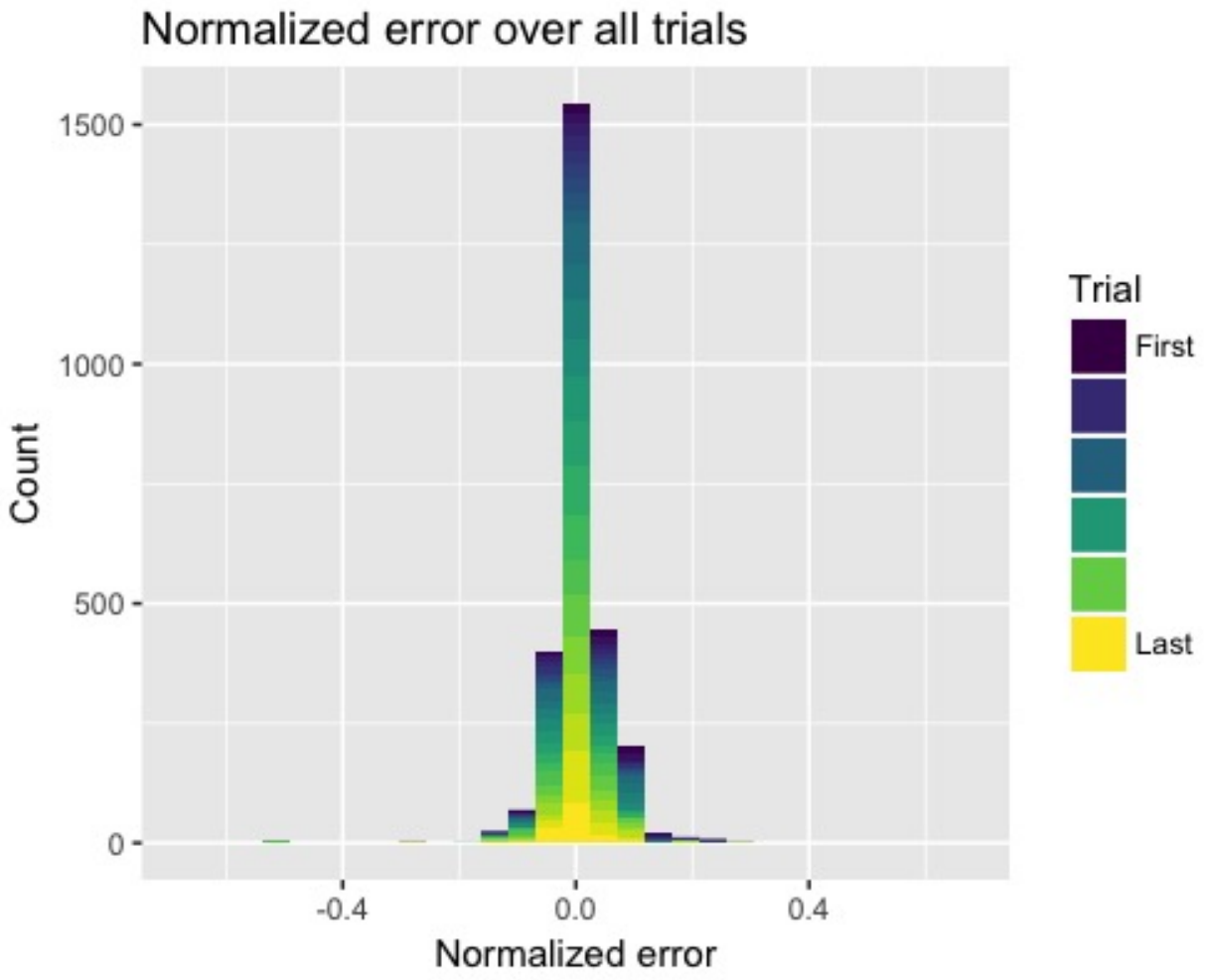

Figure 2: **Figure**. Histogram of individual participants' normalized error for each guess over all trials. Histogram is further broken down by the trial number at which each guess was given.

1.12 Export data

```
write.table(winnowed_info_df, './data/winnowed_data.csv', sep=',',
            append = FALSE, quote = FALSE, na = "NA", row.names = FALSE, col.names = TRUE)
```

---

2 Data exploration and descriptive statistics

2.1 Preliminaries

```
# clear our workspace
rm(list=ls())

# read in libraries and create functions
source('./supplementary-code/libraries_and_functions-pmc.r')

# read in dataset
winnowed_info_df = read.table('./data/winnowed_data.csv', sep=',',header = TRUE)
```

2.2 Descriptive statistics

```
# get list of individual experiments included in the data
experiment_files = list.dirs('./data', recursive=FALSE)

# concatenate the files
participant_files = data.frame()
for (experiment in experiment_files){

  # read in the next experiment's files and add ID to each
  exp_id = strsplit(as.character(experiment),"/|-")[[1]][3]
  next_participant = read.table(paste(experiment,'/participant.csv',sep=''), sep=',',
                           header=TRUE, stringsAsFactors = FALSE) %>%
    mutate(experiment = exp_id)

  # append to group files
  participant_files = rbind.data.frame(participant_files, next_participant)
}

# keep only the info we'll need
participant_time_df = participant_files %>%
  select(id, creation_time, end_time, experiment, worker_id, bonus) %>%
  mutate(creation_time = ymd_hms(creation_time)) %>%
  mutate(end_time = ymd_hms(end_time)) %>%
  mutate(duration = (end_time - creation_time)/60)
```

```
# identify how long all participants took to complete the experiment
all_participant_time = participant_time_df %>%
  select(-bonus) %>%
  na.omit()
```

```
## Average participation: 13.00024 minutes
```

```
# identify how long included participants took to complete the experiment
included_participant_time = participant_time_df %>%
  na.omit()
```

```
## Average participation: 12.59589 minutes
```

We'd intended for each experimental session to last 20 minutes, but the mean duration for included participants in these pilot data is about half of that. Future pilot studies should increase the number of trials.

2.3 Variable distributions

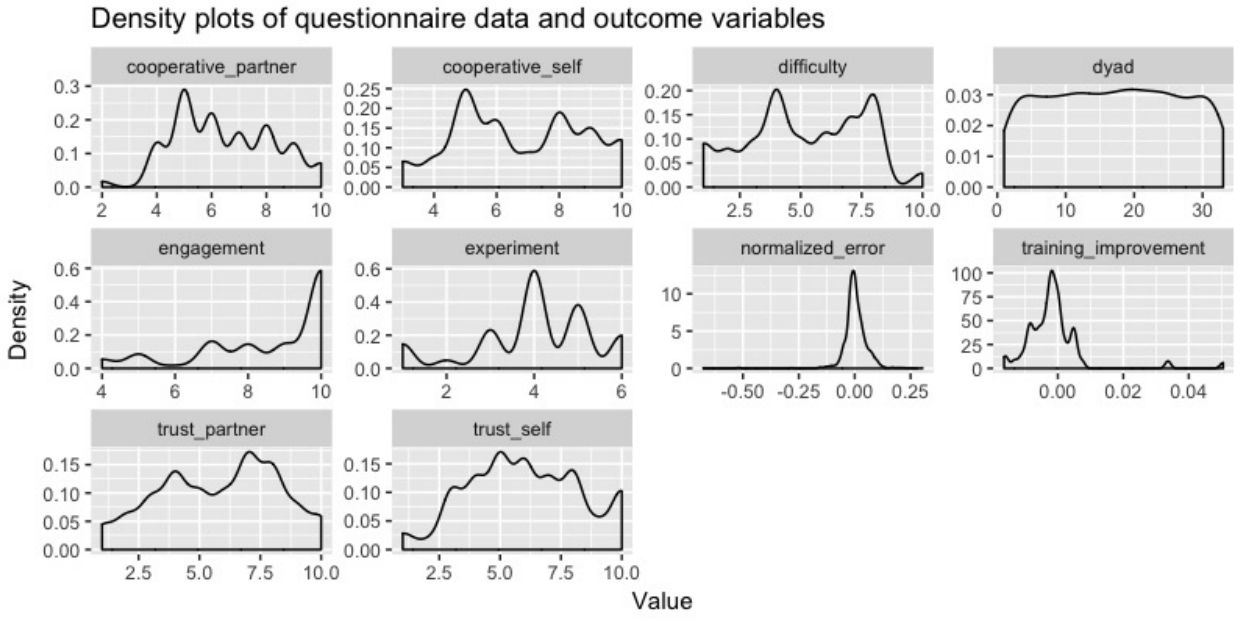

Figure 3: **Figure**. Density plots of questionnaire responses, normalized error, and improvement over training trials.

---

3 Psychonomics 2017 poster

This section presents the data analysis for the Psychonomics 2017 poster, "Exploring social behavior with Dallinger, an open-source experiment automation tool" (A. Paxton, J. Suchow, T. J. H. Morgan, & T. L. Griffiths, forthcoming). The poster largely presents Dallinger and its specific capabilities of interest to individuals running social psychology experiments, but an analysis of the pilot data from the present study is included as a demonstration.

Because of each dyad in the pilot data only completes 15 trials together, we are unable to do cross-recurrence quantification analysis (CRQA) on the `normalized_error` data. As a result, we instead do cross-correlation. Future pilot studies should increase the number of trials to create a sufficiently long time `normalized_error` time series for each dyad to permit CRQA.

3.1 Preliminaries

Clear the workspace, load in required libraries and functions, and read in the prepared dataset.

```
# clear our workspace
rm(list=ls())

# read in libraries and create functions
source('./supplementary-code/libraries_and_functions-pmc.r')

# read in dataset
winnowed_info_df = read.table('./data/winnowed_data.csv', sep=',',header = TRUE)
```

3.2 Data preparation

3.2.1 Calculate cross-correlation between partners' normalized error

First, we prepare the dataframe for cross-correlation by transitioning from long-form data for both participants within the dyad to using wide-form data for each dyad, with one column for each constituent participant's `normalized_error` at each `trial_number` and `response_counter`.

```
# strip out unnecessary information
infos = winnowed_info_df %>% ungroup() %>%
  select(experiment,dyad,participant,t,normalized_error,trial_number,response_counter) %>%

  # create participant binary values
  group_by(experiment, dyad) %>%
  mutate(partner_id = (min(participant)+max(participant)) - participant) %>%
  mutate(self_id = participant) %>%
  ungroup() %>%

  # create binary ID
  group_by(experiment, dyad) %>%
  mutate(partner_binary = participant - min(participant)) %>%
  ungroup() %>%

  # remove training trials
  dplyr::filter(trial_number > 10)

# create a dataframe with one dyad per row and separate columns for each participant's error data
binary_dfs = split(infos, infos$partner_binary)
p0_df = data.frame(binary_dfs[[1]]) %>%
  dplyr::rename(error0 = normalized_error) %>%
  select(experiment, dyad, trial_number, response_counter, error0)
p1_df = data.frame(binary_dfs[[2]]) %>%
  dplyr::rename(error1 = normalized_error) %>%
  select(experiment, dyad, trial_number, response_counter, error1)
```

Once the data are prepared, we calculate the cross-correlation coefficients between participants' `normalized_error` during all test rounds. The maximum lag is specified within the `libraries_and_functions-pmc.r` file.

```
# calculate cross-correlation
ccf_df = full_join(p0_df,p1_df,
                   by= c("experiment", "dyad", "trial_number", "response_counter")) %>%
```

```
  # calculate cross-correlation for each dyad's error scores
  group_by(experiment,dyad) %>%
  do(ccf = ccf(.$error0, .$error1, lag.max = ccf_max_lag, type = 'correlation',
               na.action = na.pass, plot=FALSE)) %>%
  ungroup() %>%

  # extract cross-correlations from the embedded list
  select(ccf) %>%
  dplyr::pull(ccf) %>%
  unlist() %>%
  matrix(.,ncol=length(unique(winnowed_info_df$dyad))) %>%

  # convert it into a proper dataframe and select only the coefficients
  as.data.frame() %>%
  slice(1:(ccf_max_lag*2+1)) %>%
  t() %>%
  as.data.frame %>%
  rowid_to_column(var='dyad') %>%

  # rename variables and strip rownames
  rename_(.dots=setNames(names(.),
                         gsub("V", "", names(.)))) %>%
  remove_rownames() %>%

  # reshape the data to combine lag and r
  gather(key = 'lag' , value='r', -dyad) %>%
  mutate_all(as.numeric) %>%
  mutate(lag = lag - ccf_max_lag - 1)
```

```
## Warning: attributes are not identical across measure variables; they will
## be dropped
```

Because we don't have any theoretical expectations about or experimental manipulations to change *who* might be leading and following, we ignore directionality for this first-pass analysis.

```
# ignore lag directionality
ccf_df = ccf_df %>% ungroup() %>%
  mutate(lag = abs(lag)) %>%
  group_by(dyad,lag) %>%
  summarise(r = mean(r))
```

Once we've calculated the cross-correlation coefficients for each dyad, we merge it into the questionnaire data.

```
# grab what we need for the cross-correlation analyses
questions_only = winnowed_info_df %>%
  select(one_of(c('experiment','dyad','participant',
                  questionnaire_variables, 'training_improvement'))) %>%

  # create a mean training improvement score for the dyad
  group_by(experiment, dyad) %>%
  mutate(training_improvement = mean(training_improvement)) %>%
  ungroup() %>%

  # select only the unique rows
  distinct()
```

```
# merge into the ccf dataframe
ccf_df = full_join(questions_only, ccf_df,
                   by='dyad','experiment')
```

Let's clean up a bit before we move on.

```
# clean up unneeded variables
rm(p0_df,p1_df,binary_dfs, infos, questions_only)
```

3.2.2 Create interaction terms

3.2.2.1 For `winnowed_info_df`

```
# create interactions
winnowed_info_df = winnowed_info_df %>% ungroup() %>%

  # create a turn variable across trials and responses
  as.data.frame() %>%
  group_by(experiment,dyad,participant) %>%
  mutate(turn = row_number()) %>%
  ungroup() %>%

  # exclude training data
  dplyr::filter(trial_type=='test') %>%

  # survey interactions
  mutate(cooperative.both = cooperative_self * cooperative_partner) %>%
  mutate(trust.both = trust_self * trust_partner) %>%
  mutate(cooperative.trust.self = cooperative_self * trust_self) %>%
  mutate(cooperative.trust.partner = cooperative_partner * trust_partner) %>%

  # error interactions
  mutate(error.length = (normalized_error+.00001) * length) %>%
  mutate(error.turn = (normalized_error+.00001) * turn) %>%
  mutate(error.length.turn = (normalized_error+.00001) * turn * length) %>%

  # other interactions
  mutate(turn.training = turn * training_improvement)
```

```
# spin off a dataset for only first answers
first_guess_df = winnowed_info_df %>% ungroup() %>%

  # grab just the final guess on each trial guess
  dplyr::filter(response_counter==1) %>%

  # filter out "turn" variables
  select(-contains("turn")) %>%

  # recreate the interactions at the trial level
  mutate(error.length = (normalized_error+.00001) * length) %>%
  mutate(error.trial = (normalized_error+.00001) * trial_number) %>%
  mutate(error.length.trial = (normalized_error+.00001) * trial_number * length) %>%
  mutate(trial.training = trial_number * training_improvement)
```

```r
# spin off a dataset for only final answers
final_guess_df = winnowed_info_df %>% ungroup() %>%

  # grab just the final guess on each trial guess
  group_by(experiment,dyad,participant,trial_number) %>%
  slice(n()) %>%
  ungroup() %>%

  # filter out "turn" variables
  select(-contains("turn")) %>%

  # recreate the error interactions at the trial level
  mutate(error.length = (normalized_error+.00001) * length) %>%
  mutate(error.trial = (normalized_error+.00001) * trial_number) %>%
  mutate(error.length.trial = (normalized_error+.00001) * trial_number * length) %>%
  mutate(trial.training = trial_number * training_improvement)
```

3.2.2.2 For `ccf_df`

```r
# create first- and second-order orthogonal polynomials for lag
raw_lag = min(ccf_df$lag):max(ccf_df$lag)
lag_vals = data.frame(raw_lag)
lag_offset = (0-min(raw_lag)) + 1
t = stats::poly((raw_lag + lag_offset), 2)
lag_vals[, paste("lag_ot", 1:2, sep="")] = t[lag_vals$raw_lag + lag_offset, 1:2]

# join it to the original data table
ccf_df = left_join(ccf_df,lag_vals, by = c("lag" = "raw_lag"))
```

```r
ccf_df = ccf_df %>% ungroup() %>%

  # create interactions among static variables of interest
  mutate(cooperative.both = cooperative_self * cooperative_partner) %>%
  mutate(trust.both = trust_self * trust_partner) %>%
  mutate(cooperative.trust.self = cooperative_self * trust_self) %>%
  mutate(cooperative.trust.partner = cooperative_partner * trust_partner) %>%

  # first-order polynomials with lag
  mutate(cooperative_self.lag_ot1 = cooperative_self * lag_ot1) %>%
  mutate(cooperative_partner.lag_ot1 = cooperative_partner * lag_ot1) %>%
  mutate(trust_self.lag_ot1 = trust_self * lag_ot1) %>%
  mutate(trust_partner.lag_ot1 = trust_partner * lag_ot1) %>%

  # first-order polynomials with lag
  mutate(cooperative_self.lag_ot2 = cooperative_self * lag_ot2) %>%
  mutate(cooperative_partner.lag_ot2 = cooperative_partner * lag_ot2) %>%
  mutate(trust_self.lag_ot2 = trust_self * lag_ot2) %>%
  mutate(trust_partner.lag_ot2 = trust_partner * lag_ot2) %>%

  # polynomial interactions
  mutate(lag_ot1.lag_ot2 = lag_ot1 * lag_ot2) %>%
  mutate(cooperative.both.lag_ot1.lag_ot2 = cooperative.both * lag_ot1 * lag_ot2) %>%
  mutate(trust.both.lag_ot1.lag_ot2 = trust_self * trust_partner * lag_ot1 * lag_ot2) %>%
```

```
  mutate(cooperative.trust.self.lag_ot1.lag_ot2 = cooperative_self * trust_self * lag_ot1 * lag_ot2) %>%
  mutate(cooperative.trust.partner.lag_ot1.lag_ot2 = cooperative_partner * trust_partner * lag_ot1 * lag
```

3.2.3 Create standardized datasets

3.2.3.1 For `winnowed_info_df`

```
info_plot = winnowed_info_df %>% ungroup() %>%
  mutate_at(vars(participant,dyad),
            funs(factor))

info_st = winnowed_info_df %>% ungroup() %>%
  mutate_all(funs(as.numeric(scale(as.numeric(.))))) %>%
  mutate_at(vars(participant,dyad),
            funs(factor))

first_guess_plot = first_guess_df %>% ungroup() %>%
  mutate_at(vars(participant,dyad),
            funs(factor))

first_guess_st = first_guess_df %>% ungroup() %>%
  mutate_all(funs(as.numeric(scale(as.numeric(.))))) %>%
  mutate_at(vars(participant,dyad),
            funs(factor))

final_guess_plot = final_guess_df %>% ungroup() %>%
  mutate_at(vars(participant,dyad),
            funs(factor))

final_guess_st = final_guess_df %>% ungroup() %>%
  mutate_all(funs(as.numeric(scale(as.numeric(.))))) %>%
  mutate_at(vars(participant,dyad),
            funs(factor))
```

3.2.3.2 For `ccf_df`

```
# create unstandardized dataframe and convert relevant variables to factors
ccf_plot = ccf_df %>% ungroup() %>%
  mutate_at(vars(participant,dyad),
            funs(factor))

# create standardized dataframe and convert relevant variables to factors
ccf_st = ccf_df %>%
  mutate_all(funs(as.numeric(scale(as.numeric(.))))) %>%
  mutate_at(vars(participant,dyad),
            funs(factor))
```

3.3 Data analysis

Here are our preliminary questions:

- **When accounting for improvement during training, do individuals become more accurate over time. . .**
  - **. . . in their final guesses?** No, with non-significant trend toward "yes."
  - **. . . across all guesses?** No, with non-significant trend toward "yes."
  - **. . . in their first guesses?** While training improvement significantly predicts accuracy of first guess, there is also a non-significant trend toward "yes."
- **Are partners temporally coordinated in their guess accuracy across time?** Yes.
- **Do partners' ratings of their own and their partners' trustworthiness and cooperativity influence the dynamics of perceptual coordination?** No.

We also have some ideas that will shape future analyses.

- When we have more time:
  - *Are post-experiment ratings of cooperativity related to the directionality of change? (That is, are people perceived as being more cooperative if they change their opinion to be more like their partner?)*
  - *Are post-experiment ratings of trustworthiness related to unwillingness to change? (That is, are people perceived as being more trustworthy if they stick to their initial guess?)*
  - *Are partners of consistently more reliable guessers more likely to adopt a strategy to minimize their own first-guess effort and then to simply mimic their partner?*
  - *Are partners of consistently poor guessers more likely to adopt a strategy to maximize their first guess and ignore their partner?*
- If we can get longer time series:
  - *Do partners become more similar across trials in the accuracy of their first guesses?*
  - *Do partners become more similar across trials in the accuracy of their last guesses?*
  - *Do partners become more similar within trials in their changes to their guesses?*

Notes for future work:

- Partners might think of this as a competition. If so, we should as a question to capture that.

3.3.1 Do individuals become more accurate over time?

3.3.1.1 Do individuals become more accurate in their final guesses over time?

```
# standardized estimates
more_accurate_final_over_time_st = lmer(normalized_error ~ trial_number + training_improvement + trial.
                                  (1 + trial_number | participant)  +
                                  (1 + trial_number | dyad),
                                data = final_guess_st)
pander_lme(more_accurate_final_over_time_st, stats.caption = TRUE)
```

Table 1: **Marginal *R*-squared: 0.01. Conditional *R*-squared: 0.11.**

	Estimate	Std..Error	t.value	p	sig
(Intercept)	-0.001403	0.04862	-0.02885	0.98	
trial_number	-0.06186	0.03956	-1.564	0.118	
training_improvement	0.1671	0.1459	1.145	0.25	
trial.training	-0.1022	0.1555	-0.657	0.51	

```
# raw estimates
more_accurate_final_over_time_raw = lmer(normalized_error ~ trial_number + training_improvement + trial
```

```
                                  (1 + trial_number | participant)  +
                                  (1 + trial_number | dyad),
                                data = final_guess_plot)
pander_lme(more_accurate_final_over_time_raw, stats.caption = TRUE)
```

Table 2: **Marginal *R*-squared: 0.01. Conditional *R*-squared: 0.11.**

	Estimate	Std..Error	t.value	p	sig
(Intercept)	0.01341	0.007826	1.713	0.087	.
trial_number	-0.0006859	0.0004386	-1.564	0.118	
training_improvement	0.854	0.7456	1.145	0.25	
trial.training	-0.02818	0.04289	-0.657	0.51	

3.3.1.2 Do individuals become more accurate across all turns?

```
# standardized estimates
more_accurate_over_all_time_st = lmer(normalized_error ~ turn + training_improvement + turn.training +
                                  (1 + turn | participant)  +
                                  (1 | dyad),
                                data = info_st)
pander_lme(more_accurate_over_all_time_st, stats.caption = TRUE)
```

Table 3: **Marginal *R*-squared: 0.01. Conditional *R*-squared: 0.16.**

	Estimate	Std..Error	t.value	p	sig
(Intercept)	-0.009465	0.04729	-0.2002	0.84	
turn	-0.05947	0.0346	-1.719	0.086	.
training_improvement	0.123	0.09093	1.353	0.176	
turn.training	-0.08714	0.08848	-0.9849	0.32	

```
# unstandardized estimates
more_accurate_over_all_time_raw = lmer(normalized_error ~ turn + training_improvement + turn.training +
                                  (1 + turn | participant)  +
                                  (1 | dyad),
                                data = info_plot)
```

```
## Warning: Some predictor variables are on very different scales: consider
## rescaling
```

```
pander_lme(more_accurate_over_all_time_raw, stats.caption = TRUE)
```

Table 4: **Marginal *R*-squared: 0.01. Conditional *R*-squared: 0.16.**

	Estimate	Std..Error	t.value	p	sig
(Intercept)	0.008359	0.004907	1.704	0.088	.
turn	-0.0003114	0.0001812	-1.719	0.086	.
training_improvement	0.6586	0.4869	1.353	0.176	
turn.training	-0.01832	0.0186	-0.9849	0.32	

3.3.1.3 Do individuals become more accurate across first guesses?

```
# standardized estimates
more_accurate_first_guesses_st = lmer(normalized_error ~ trial_number + training_improvement + trial.tra
                                      (1 + trial_number | participant)  +
                                      (1 | dyad),
                                    data = first_guess_st)
pander_lme(more_accurate_first_guesses_st, stats.caption = TRUE)
```

Table 5: **Marginal *R*-squared: 0.01. Conditional *R*-squared: 0.08.**

	Estimate	Std..Error	t.value	p	sig
(Intercept)	0.00111	0.04365	0.02543	0.98	
trial_number	-0.06219	0.03538	-1.757	0.079	.
training_improvement	0.3044	0.1526	1.994	0.046	*
trial.training	-0.2462	0.1513	-1.627	0.104	

```
# unstandardized estimates
more_accurate_first_guesses_raw = lmer(normalized_error ~ trial_number + training_improvement + trial.t
                                      (1 + trial_number | participant)  +
                                      (1 | dyad),
                                    data = first_guess_plot)
pander_lme(more_accurate_first_guesses_raw, stats.caption = TRUE)
```

Table 6: **Marginal *R*-squared: 0.01. Conditional *R*-squared: 0.08.**

	Estimate	Std..Error	t.value	p	sig
(Intercept)	0.01663	0.008166	2.037	0.042	*
trial_number	-0.0007673	0.0004366	-1.757	0.079	.
training_improvement	1.706	0.8552	1.994	0.046	*
trial.training	-0.07428	0.04565	-1.627	0.104	

Note: We should consider what to do with some of these high-error guesses. At least some of them may be an artifact of the incentives of the experiment (i.e., not wanting to not guess during a turn).

3.3.2 Do partners provide similar guesses?

```
# standardized estimates
similar_partners_st = lmer(r ~ lag_ot1 + lag_ot2 + lag_ot1.lag_ot2 + training_improvement +
                             (1 + lag_ot1 + lag_ot2 + lag_ot1.lag_ot2 | dyad),
                           data = ccf_st)
pander_lme(similar_partners_st, stats.caption=TRUE)
```

Table 7: **Marginal *R*-squared: 0.14. Conditional *R*-squared: 0.95.**

	Estimate	Std..Error	t.value	p	sig
(Intercept)	-0.00000000000002796	0.09882	-0.0000000000002829	1	
lag_ot1	-0.3383	0.1101	-3.073	0.002	**

	Estimate	Std..Error	t.value	p	sig
lag__ot2	0.2002	0.06365	3.146	0.002	**
lag__ot1.lag__ot2	0.03646	0.05453	0.6687	0.5	
training__improvement	0.07553	0.08009	0.9431	0.35	

```
# unstandardized estimates
similar_partners_raw = lmer(r ~ lag_ot1 + lag_ot2 + lag_ot1.lag_ot2 + training_improvement +
                              (1 + lag_ot1 + lag_ot2 + lag_ot1.lag_ot2 | dyad),
                            data = ccf_plot)
pander_lme(similar_partners_raw, stats.caption=TRUE)
```

Table 8: **Marginal *R*-squared: 0.14. Conditional *R*-squared: 0.95.**

	Estimate	Std..Error	t.value	p	sig
(Intercept)	0.04852	0.02199	2.206	0.027	*
lag__ot1	-0.1808	0.05885	-3.073	0.002	**
lag__ot2	0.107	0.03403	3.146	0.002	**
lag__ot1.lag__ot2	0.04146	0.06199	0.6687	0.5	
training__improvement	2.6	2.757	0.9431	0.35	

3.3.3 Do ratings of cooperativity and trustworthiness influence adaptation?

```
# standardized estimates
similar_partners_moderators_st = lmer(r ~ lag_ot1 + lag_ot2 + lag_ot1.lag_ot2 + training_improvement +
                                        trust_self + trust_self.lag_ot1 + trust_self.lag_ot2 +
                                        trust_partner + trust_partner.lag_ot1 + trust_partner.lag_ot2 +
                                        trust.both +
                                        (1 + lag_ot1 + lag_ot2 + lag_ot1.lag_ot2 | dyad) +
                                        (1 | participant),
                                      data = ccf_st)
pander_lme(similar_partners_moderators_st, stats.caption=TRUE)
```

Table 9: **Marginal *R*-squared: 0.14. Conditional *R*-squared: 0.95.**

	Estimate	Std..Error	t.value	p	sig
(Intercept)	-0.00000000000001615	0.09872	-0.0000000000001636	1	
lag__ot1	-0.3355	0.1168	-2.873	0.004	**
lag__ot2	0.1922	0.07433	2.586	0.01	*
lag__ot1.lag__ot2	0.03646	0.05453	0.6687	0.5	
training__improvement	0.07585	0.07994	0.9488	0.34	
trust__self	0.007585	0.04039	0.1878	0.85	
trust__self.lag__ot1	0.004718	0.04376	0.1078	0.91	
trust__self.lag__ot2	0.01393	0.04337	0.3212	0.75	
trust__partner	0.003644	0.04352	0.08372	0.93	
trust__partner.lag__ot1	-0.007782	0.04144	-0.1878	0.85	
trust__partner.lag__ot2	-0.005387	0.04101	-0.1314	0.9	
trust.both	-0.008834	0.07051	-0.1253	0.9	

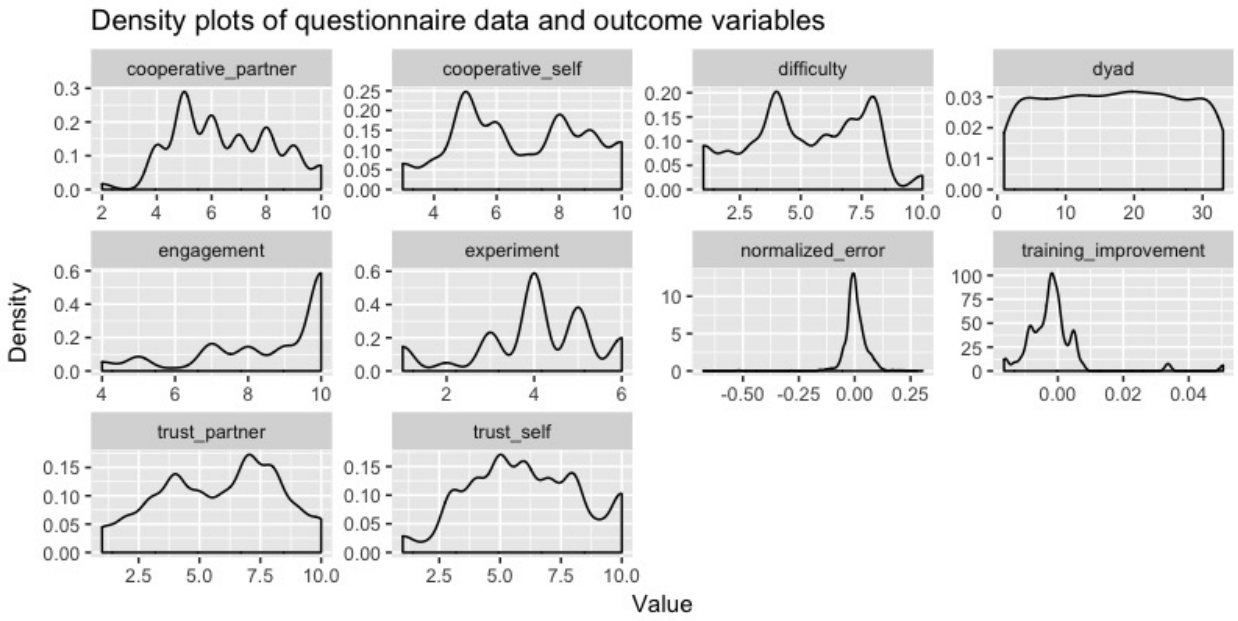

Figure 4: **Figure**. Density plots of questionnaire responses, normalized error, and improvement over training trials.

```
# unstandardized estimates
similar_partners_moderators_raw = lmer(r ~ lag_ot1 + lag_ot2 + lag_ot1.lag_ot2 + training_improvement +
                                         trust_self + trust_self.lag_ot1 + trust_self.lag_ot2 +
                                         trust_partner + trust_partner.lag_ot1 + trust_partner.lag_ot2 +
                                         trust.both +
                                         (1 + lag_ot1 + lag_ot2 + lag_ot1.lag_ot2 | dyad) +
                                         (1 | participant),
                                       data = ccf_plot)
```

```
## Warning: Some predictor variables are on very different scales: consider
## rescaling
```

```
pander_lme(similar_partners_moderators_raw, stats.caption=TRUE)
```

Table 10: **Marginal *R*-squared: 0.14. Conditional *R*-squared: 0.95.**

	Estimate	Std..Error	t.value	p	sig
(Intercept)	0.04515	0.03078	1.467	0.142	
lag__ot1	-0.1794	0.06243	-2.873	0.004	**
lag__ot2	0.1028	0.03973	2.586	0.01	*
lag__ot1.lag__ot2	0.04146	0.06199	0.6687	0.5	
training__improvement	2.611	2.752	0.9488	0.34	
trust__self	0.0007138	0.003801	0.1878	0.85	
trust__self.lag__ot1	0.0003912	0.003629	0.1078	0.91	
trust__self.lag__ot2	0.001155	0.003596	0.3212	0.75	
trust__partner	0.0003255	0.003888	0.08372	0.93	
trust__partner.lag__ot1	-0.0006593	0.003511	-0.1878	0.85	
trust__partner.lag__ot2	-0.0004564	0.003474	-0.1314	0.9	
trust.both	-0.00007303	0.0005829	-0.1253	0.9	

	Estimate	Std..Error	t.value	p	sig

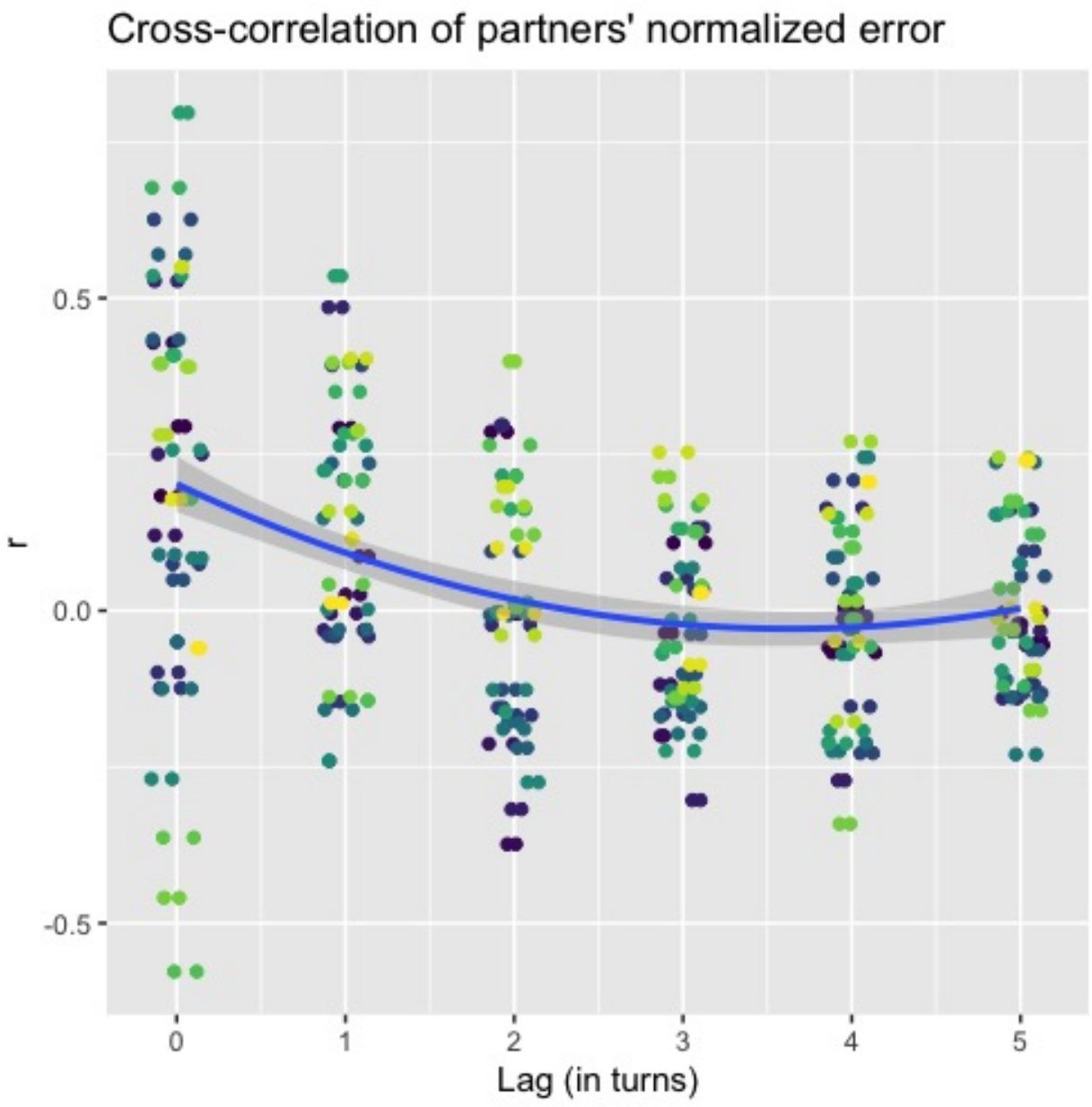

Figure 5: **Figure**. Cross-correlation of partners' normalized error across turns, overlaid with best-fit line.