0_power_analysis

Stefan Vermeent

1-11-2021

Introduction

This is a power analysis for the cognitive tasks in the pilot, where I calculate the required number of participants given a certain number of trials. As the DDM is fit to each individual dataset, running a power analysis with several hundreds of iterations per subject/condition would take a substantial amount of time. In order to prevent having to fit the DDM that many times, the power analysis below consists of two steps.

In part I, I calculate the correlation between simulated and recovered DDM parameters for varying numbers of trials and across two common estimation techniques: Maximum Likelihood and Kolmogorov-Smirnoff. As estimated DDM parameters are not perfectly correlated with the true, data-generating parameter values (especially with fewer trials), this correlation gives an estimate of the average "imprecision" of estimated DDM parameters. I do this for the Drift Rate and the Boundary Separation parameters, which are of main interest to us.

In part II, I use these correlation values to simulate generate two sets of DDM parameters: the "true", data-generating parameter values, and the "recovered" parameter values which correlate to a fixed extend with the "true" parameters, as informed by the first part of the analysis. Thus, this approach allows us to run a power analysis on the DDM parameters without having to fit the DDM tens- to hundreds of thousands of times.

Estimated DDM parameters are not perfectly correlated with the true underlying parameter values, especially with fewer trials. Here, I attempt to get an estimate of the correlation between simulated DDM parameters and recovered parameters with varying number of trials and with two different estimation techniques: Maximum Likelihood and Kolmogorov-Smirnoff. These correlations can be used in the subsequent power analysis to quickly simulate "empirical" DDM parameters, which will prevent us from having to run thousands of DDM models.

Part I: Estimation of DDM Precision

We simulate RT data for 500 subjects and with varying numbers of trials per subject: 10, 20, 30, 40, 50 and 60 trials.

```
n_subjects = 500
n_trials = c(10, 20, 30, 40, 50, 60)
```

Create RT dataset

First, we randomly sample Drift Diffusion parameters for each subject that will be used to generate RTs. Drift rates are sampled from a uniform distribution ranging from 1 to 4.9. Boundary separation is sampled from a uniform distribution ranging from 0.6 to 1.9. Non-decision time is fixed to 0.3 and Starting-point is fixed to 0.5.

```
set.seed(42)
```

```
DDM_data <- expand.grid(subject = 1:n_subjects, trials = n_trials) %>%
  mutate(
    drift_rate = runif(n(), min = 1, max = 4.9),
    boundary_sep = runif(n(), 0.6, 1.9),
    tau = 0.3,
    beta = 0.5,
    id = str_c("simulationsubj", subject, "trials", trials, sep = "_")
  )
```

Using the DDM parameters, we next generate the RTs:

```
pwalk(DDM_data[c("trials", "boundary_sep", "tau", "beta", "drift_rate", "id")], generate_RTs)
```

The DDM is fit using the fast-dm software. Fast-dm runs via the command prompt. Follow the following steps to recalculate DDM parameters:

1. Open the command prompt and set the working directory to /data/1_pilot/0_simulation
2. Run the DDM model once for both estimation techniques by typing the following commands one after the other: fast-dm.exe exp_ks.ctl fast-dm.exe exp_ml.ctl

Read recovered DDM parameters

```
DDM_data_recovered <- bind_rows(
  read.table(here('data', "1_pilot", 'simulation', 'ddm_pars_ks.lst'), header = TRUE),
  read.table(here('data', "1_pilot", 'simulation', 'ddm_pars_ml.lst'), header = TRUE)
) %>%
  rename(id = dataset) %>%
  # Join simulated DDM parameters
  left_join(DDM_data %>% select(id, trials, drift_rate, boundary_sep)) %>%
  mutate(
    trials = as.numeric(str_extract_all(id, "[0-9]+$"))
  ) %>%
  group_by(method, trials) %>%
  # Calculate correlation between simulated and recovered parameters
  mutate(
    error_drift = drift_rate - v,
    error_boundary = boundary_sep - a
  ) %>%
  rename(
    `Simulated Drift Rate` = drift_rate,
    `Simulated Boundary Separation` = boundary_sep,
    `Recovered Drift Rate` = v,
    `Recovered Boundary Separation` = a
  )
```

```
## Joining, by = "id"
```

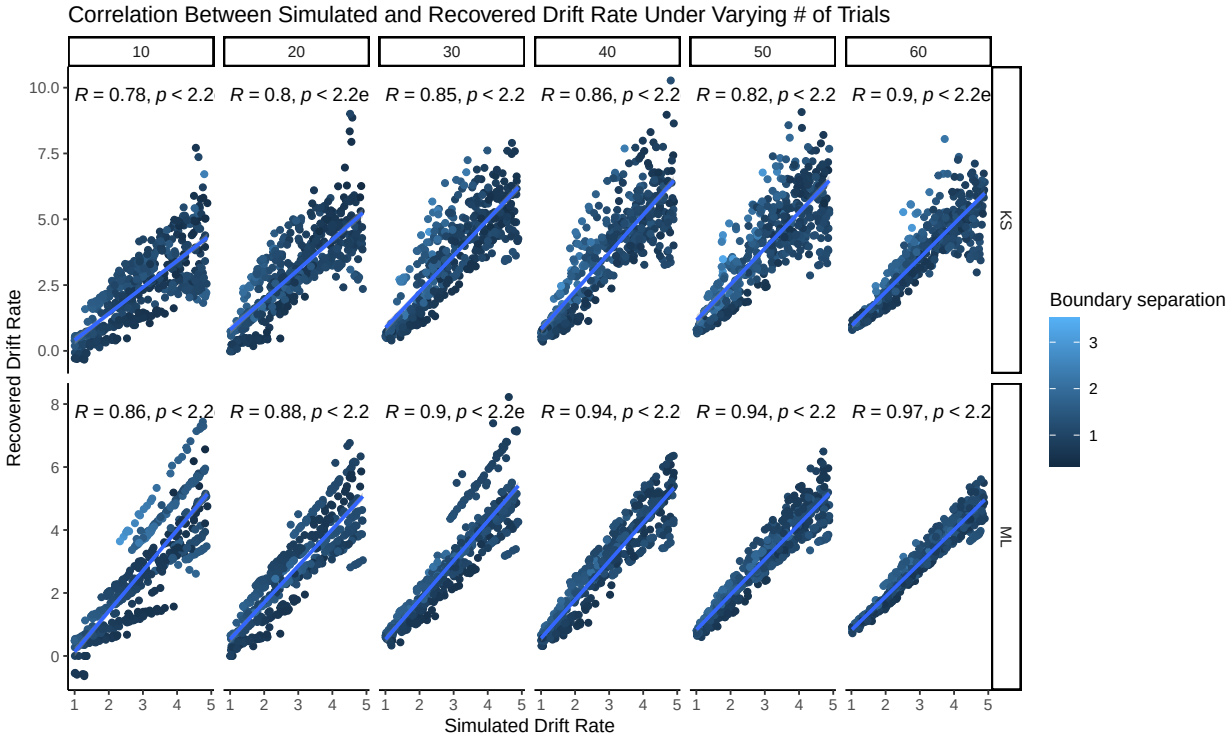
Plots

Drift Rate - Correlation

```
DDM_data_recovered %>%
  ggplot(aes(`Simulated Drift Rate`, `Recovered Drift Rate`, color = `Recovered Boundary Separation`)) +
```

```
  geom_point() +
  geom_smooth(method="lm") +
  facet_grid(method~trials, scales = "free") +
  theme_classic() +
  stat_cor(method="pearson") +
  labs(
    title = "Correlation Between Simulated and Recovered Drift Rate Under Varying # of Trials",
    color = "Boundary separation"
  )
```

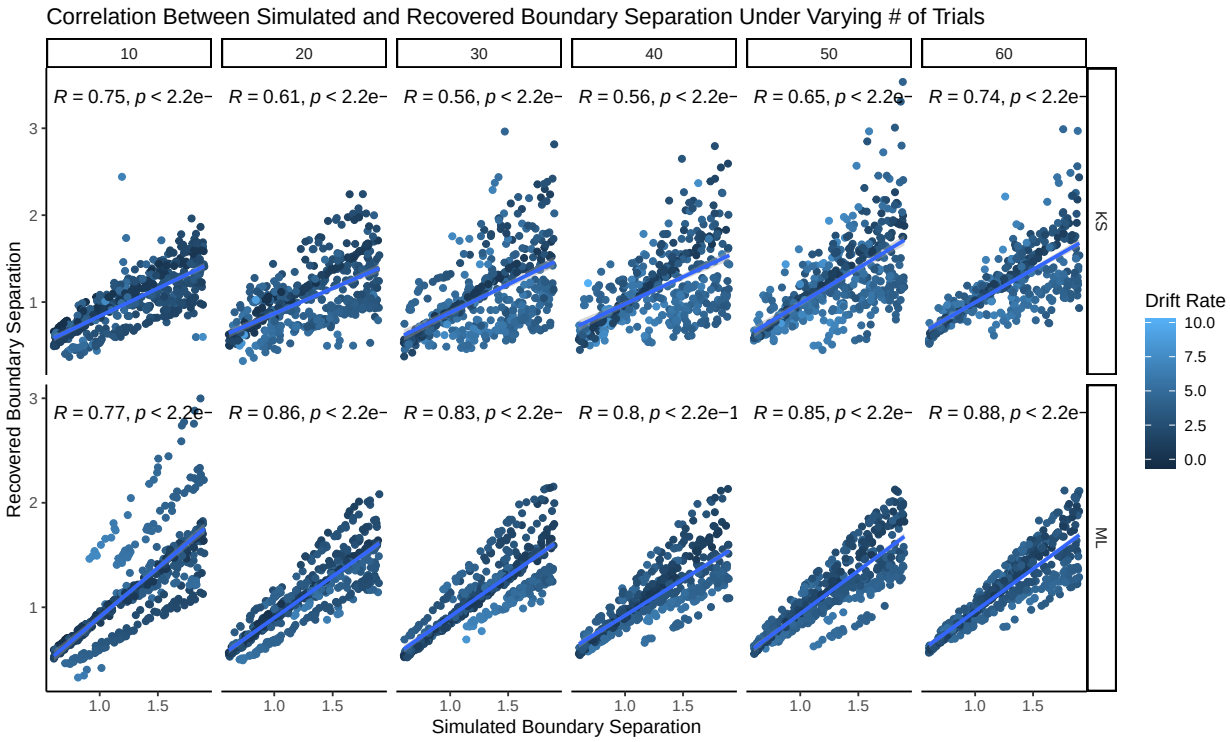
```
## `geom_smooth()` using formula 'y ~ x'
```



Boundary separation - Correlation

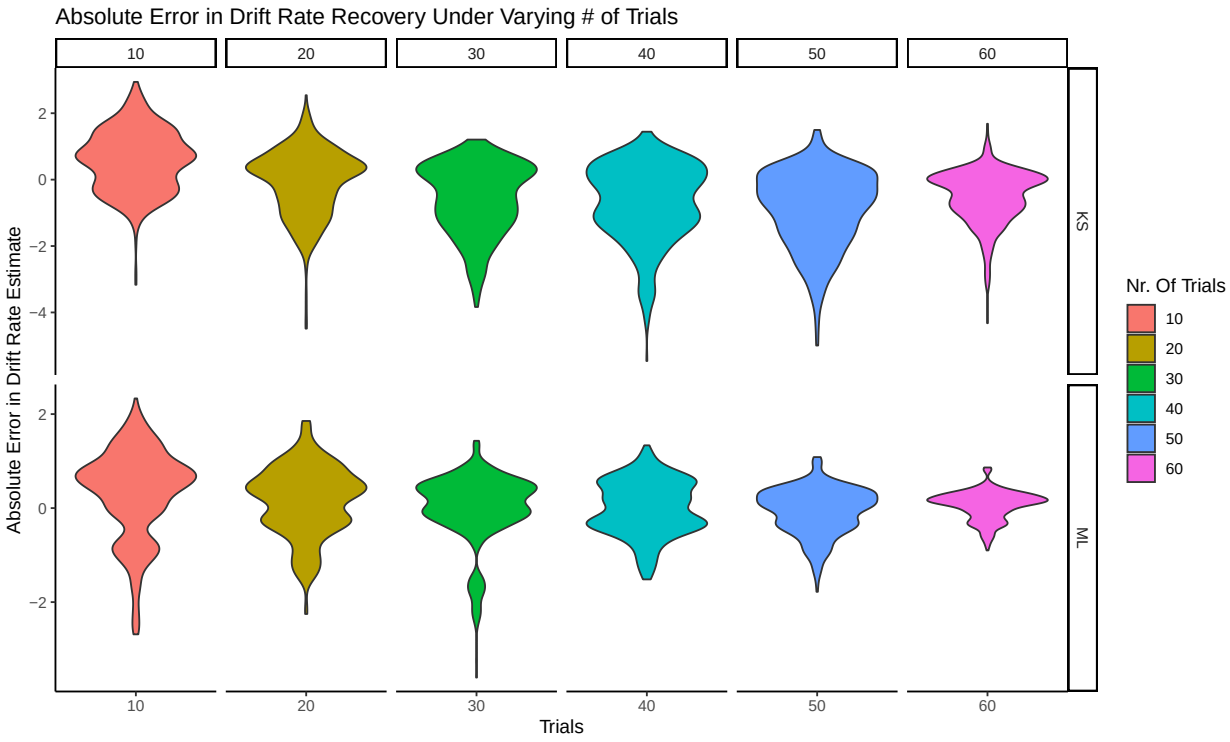
```
DDM_data_recovered %>%
  ggplot(aes(`Simulated Boundary Separation`, `Recovered Boundary Separation`, color = `Recovered Drift
  geom_point() +
  geom_smooth(method="lm") +
  facet_grid(method~trials, scales = "free") +
  theme_classic() +
  stat_cor(method="pearson") +
  labs(
    title = "Correlation Between Simulated and Recovered Boundary Separation Under Varying # of Trials"
    color = "Drift Rate"
  )
```

```
## `geom_smooth()` using formula 'y ~ x'
```



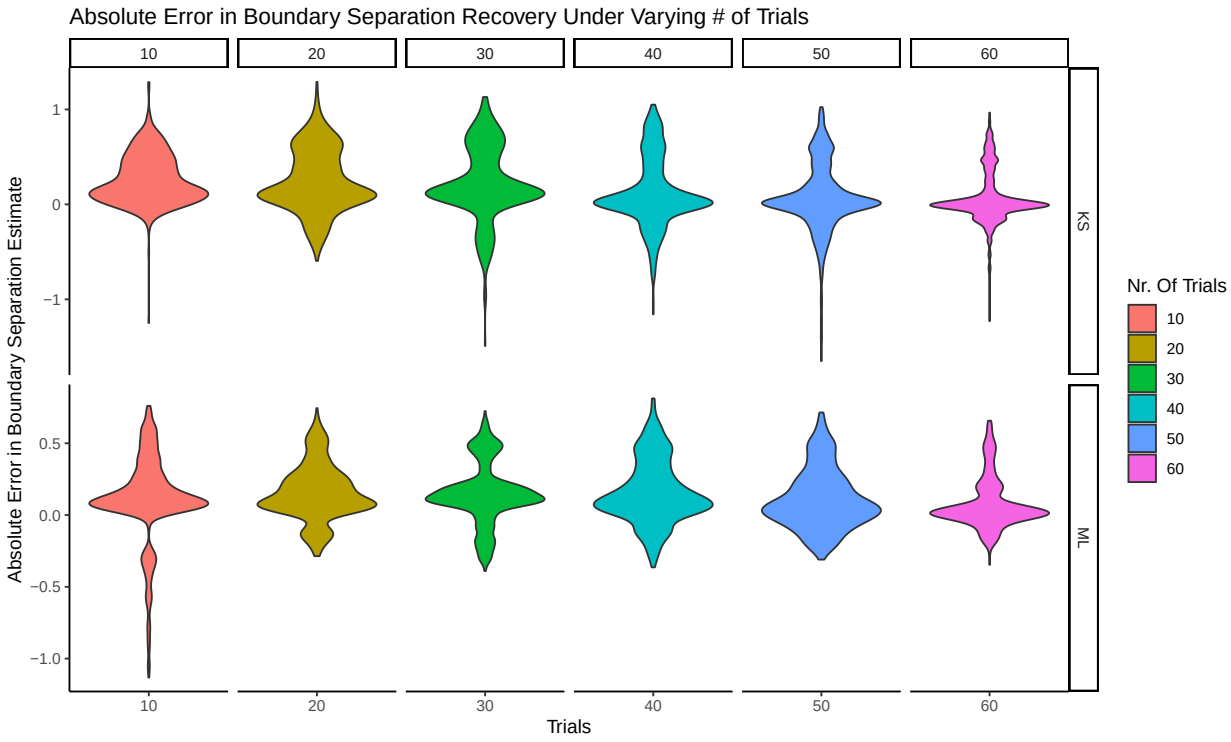
Drift Rate - Mean error

```
DDM_data_recovered %>%
  ggplot(aes(factor(trials), error_drift, fill = factor(trials))) +
  geom_violin() +
  facet_grid(method~trials, scales = "free") +
  theme_classic() +
  labs(
    title = "Absolute Error in Drift Rate Recovery Under Varying # of Trials",
    x = "Trials",
    y = "Absolute Error in Drift Rate Estimate",
    fill = "Nr. Of Trials"
  )
```



Boundary Separation - Mean error

```
DDM_data_recovered %>%
  ggplot(aes(factor(trials), error_boundary, fill = factor(trials))) +
  geom_violin() +
  facet_grid(method~trials, scales = "free") +
  theme_classic() +
  labs(
    title = "Absolute Error in Boundary Separation Recovery Under Varying # of Trials",
    x = "Trials",
    y = "Absolute Error in Boundary Separation Estimate",
    fill = "Nr. Of Trials"
  )
```



part II: Power analysis

The analyses under Part I indicate that the recovered drift rate values correlate with >=.90 with the simulated, "true" values starting at 30 trials. For the boundary separation, the correlation is ~ .80 starting at 30 trials. Below, we calculate 80% and 90% power an effect size of β for a number of scenarios:

1) Linear mixed effects model with main effect of adversity and interaction effect of condition*adversity
2) Correlation of recovered DDM parameters with the "true", data-generating DDM parameters of .60 and .85.
3) Different levels of random noise added to the IV-DV relationship: *SD* of 0.5 and 1.

Set-up

```
# Simulation parameters
intercept = 0        # Intercept
fixed_effect = 0.1   # Fixed effect of main effects and interaction
intercept_sd = 1     # Random intercept SD for subjects
```

Mixed Effects Model

```
# Use multiple cores for simulation
plan(multisession, workers = cores - 4)

mixed_df <- function(n_subjects, DDM_recovery_correlation, sigma_sd, fixed_effect,n_sim) {
  data_mixed <- add_random(subjects = n_subjects) %>%
    add_within("subjects", condition = c("cued", "neutral")) %>%
    add_ranef("subjects", intercept_s = intercept_sd) %>%
    add_ranef(sigma = sigma_sd) %>%
```

```
    add_contrast("condition", "sum", colnames = "condition_sum") %>%
    mutate(
      adversity = scale(rnorm(n(), 0, 1)) %>% as.numeric(),
      drift_rate_true = intercept + intercept_s + (fixed_effect * condition_sum) + (fixed_effect * adver
    ) %>%
    group_by(condition_sum) %>%
    mutate(
      drift_rate_recov = rnorm_pre(drift_rate_true, mu = mean(drift_rate_true), sd = sd(drift_rate_true)
      n_sim=n_sim
    ) %>%
    ungroup() %>%
    select(subjects, drift_rate_true, drift_rate_recov, condition_sum, adversity)

  return(data_mixed)
}
```

Simulate data and fit models

Note: The code below takes quite some time to process (~13 minutes on my PC using 4 processing cores).

```
tic()
# Grid containing the simulation parameters to loop over
simulation_results_mixed <- expand_grid(
  n_subjects = c(400, 450, 500, 600),
  DDM_recovery_correlation = c(.60, .85),
  sigma_sd = c(0.5, 1),
  fixed_effect =fixed_effect,
  n_sim = 1:500
) %>%
  # simulate data
  mutate(mixed_data = future_pmap(., mixed_df, .options = furrr_options(seed = TRUE)),
         ) %>%
  # Fit Mixed model to data
  mutate(mixed_model = future_map(mixed_data, function(x) {

    fit_true <- lmerTest::lmer(data = x, scale(drift_rate_true) ~ condition_sum*scale(adversity) + (1|su
    fit_recov <- lmerTest::lmer(data = x, scale(drift_rate_recov) ~ condition_sum*scale(adversity) + (1

      # Extract p-values and convergence
      list(
        p_mixed_main_true         = (fit_true %>% summary %>% coef() %>% as_tibble %>% pull(`Pr(>|t|)`)
        p_mixed_interaction_true  = (fit_true %>% summary %>% coef() %>% as_tibble %>% pull(`Pr(>|t|)`)
        convergence_true          = ifelse(is.null(fit_true@optinfo$conv$lme4$messages), TRUE, FALSE),

        p_mixed_main_recov        = (fit_recov %>% summary %>% coef() %>% as_tibble %>% pull(`Pr(>|t|)`
        p_mixed_interaction_recov = (fit_recov %>% summary %>% coef() %>% as_tibble %>% pull(`Pr(>|t|)`
        convergence_recov         = ifelse(is.null(fit_recov@optinfo$conv$lme4$messages), TRUE, FALSE)
      )
  }
  ))

toc()
```

```
## 823.42 sec elapsed
```

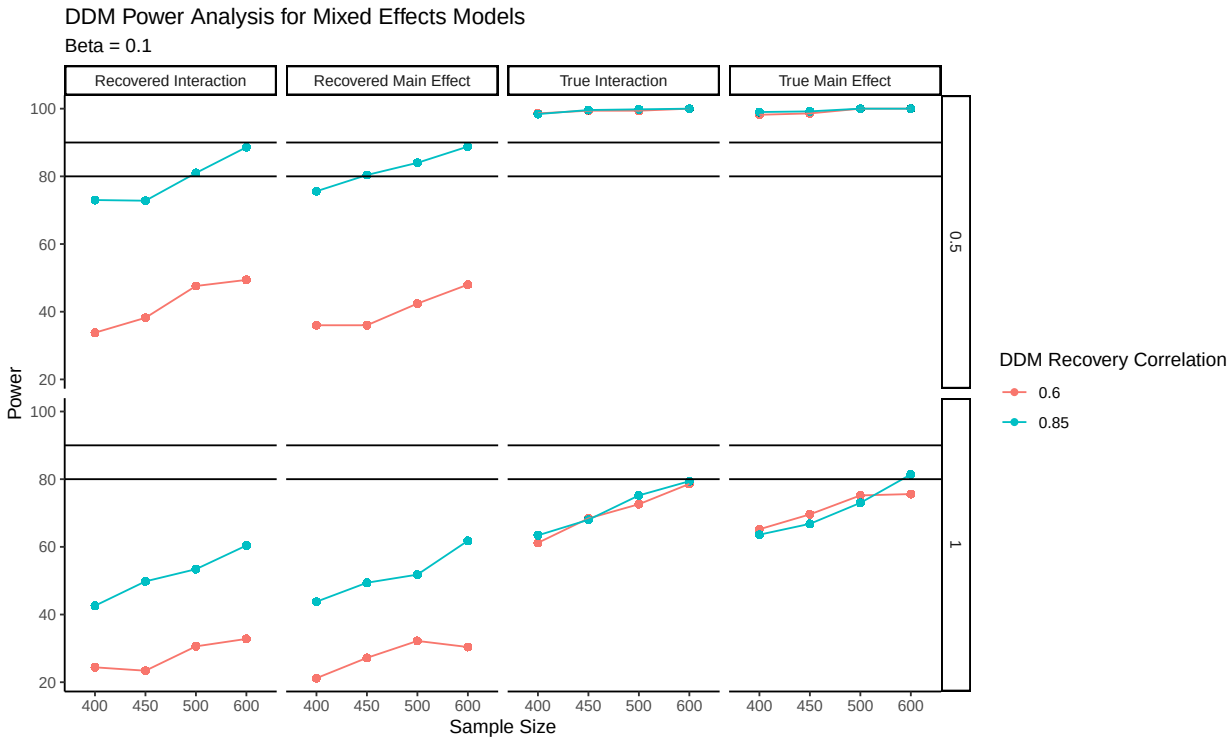
```r
plan("sequential")
```

Calculate power

```r
power_mixed <- simulation_results_mixed %>%
  unnest_wider(mixed_model) %>%
  # Only include converged models
  mutate(across(c(p_mixed_main_true, p_mixed_interaction_true), ~ifelse(convergence_true == FALSE, NA,
  mutate(across(c(p_mixed_main_recov, p_mixed_interaction_recov), ~ifelse(convergence_recov == FALSE, NA
  group_by(n_subjects, DDM_recovery_correlation, sigma_sd) %>%
  # Calculate power
  mutate(
    mixed_true_main_power         = (sum(p_mixed_main_true < .05) / n()) * 100,
    mixed_recov_main_power        = (sum(p_mixed_main_recov < .05) / n()) * 100,
    mixed_true_interaction_power  = (sum(p_mixed_interaction_true < .05) / n()) * 100,
    mixed_recov_interaction_power = (sum(p_mixed_interaction_recov < .05) / n()) * 100
  ) %>%
  ungroup() %>%
  pivot_longer(ends_with("power"), names_to = "effect", values_to = "power")
```

Plot power curves

```r
power_mixed %>%
  mutate(
    effect = case_when(
      effect == "mixed_true_main_power" ~ "True Main Effect",
      effect == "mixed_recov_main_power" ~ "Recovered Main Effect",
      effect == "mixed_true_interaction_power" ~ "True Interaction",
      effect == "mixed_recov_interaction_power" ~ "Recovered Interaction"
    )
  ) %>%
  ggplot(aes(factor(n_subjects), power, color=factor(DDM_recovery_correlation), group = factor(DDM_recov
  geom_point() +
  geom_line() +
  geom_hline(yintercept = c(80, 90)) +
  #geom_text(aes(label = power)) +
  facet_grid(sigma_sd~effect) +
  theme_classic() +
  labs(
    x = "Sample Size",
    y = "Power",
    color = "DDM Recovery Correlation",
    title = "DDM Power Analysis for Mixed Effects Models",
    subtitle = str_c("Beta = ", fixed_effect)
  )
```



Linear model

```
# Use multiple cores for simulation
plan(multisession, workers = cores - 4)


linear_df <- function(n_subjects, DDM_recovery_correlation, sigma_sd, fixed_effect, n_sim) {
  faux_options(sep=".")
  data_linear <- sim_design(within = list(vars = c("drift_rate_true", "adversity")),
                            n = n_subjects,
                            mu = list(drift_rate_true = 0, adversity = 0),
                            sd = list(drift_rate_true = 1, adversity = 1),
                            r = fixed_effect, plot = FALSE) %>%
    mutate(drift_rate_recov = rnorm_pre(drift_rate_true, mu = mean(drift_rate_true), sd = sd(drift_rate

  return(data_linear)
}
faux_options(sep = ".")
```

Simulate data and fit models

Note: The code below takes quite some time to process (~3.5 minutes on my PC using 4 processing cores).

```
tic()
# Grid containing the simulation parameters to loop over
simulation_results_linear <- expand_grid(
  n_subjects = c(400, 500, 600, 800, 1000, 1200),
  DDM_recovery_correlation = c(.60, .85),
  sigma_sd = c(0.5, 1),
```

```
  fixed_effect =fixed_effect,
  n_sim = 1:500
) %>%
  # simulate data
  mutate(linear_data = future_pmap(., linear_df, .options = furrr_options(seed = TRUE)),
  ) %>%
  # Fit Mixed model to data
  mutate(linear_model = future_map(linear_data, function(x) {

    fit_true <- lm(data  = x, scale(drift_rate_true) ~ scale(adversity))
    fit_recov <- lm(data = x, scale(drift_rate_recov) ~ scale(adversity))

    list(
      p_mixed_main_true  = (fit_true %>% summary %>% coef() %>% as_tibble %>% pull(`Pr(>|t|)`))[2],
      p_mixed_main_recov = (fit_recov %>% summary %>% coef() %>% as_tibble %>% pull(`Pr(>|t|)`))[2]
    )
  }
  ))
toc()
```

```
## 276.19 sec elapsed
```

```
plan("sequential")
```

Calculate power

```
power_linear <- simulation_results_linear %>%
  unnest_wider(linear_model) %>%
  group_by(n_subjects, DDM_recovery_correlation, sigma_sd) %>%
  # Calculate power
  mutate(
    mixed_true_main_power         = (sum(p_mixed_main_true < .05) / n()) * 100,
    mixed_recov_main_power        = (sum(p_mixed_main_recov < .05) / n()) * 100,
  ) %>%
  ungroup() %>%
  pivot_longer(ends_with("power"), names_to = "effect", values_to = "power")
```

Plot power curves

```
power_linear %>%
  mutate(
    effect = case_when(
      effect == "mixed_true_main_power" ~ "True Main Effect",
      effect == "mixed_recov_main_power" ~ "Recovered Main Effect",
    )
  ) %>%
  ggplot(aes(factor(n_subjects), power, color=factor(DDM_recovery_correlation), group = factor(DDM_recov
  geom_point() +
  geom_line() +
  geom_hline(yintercept = c(80, 90)) +
  #geom_text(aes(label = power)) +
  facet_grid(sigma_sd~effect) +
  theme_classic() +
```

```
  labs(
    x = "Sample Size",
    y = "Power",
    color = "DDM Recovery Correlation",
    title = "DDM Power Analysis for Linear Models",
    subtitle = str_c("Beta = ", fixed_effect)
  )
```