Sampling-based Type I and power analyses

T. Florian Jaeger

3/15/2021

This document

This document is created in R markdown. R markdown combines text with R code, allowing us to see the code and its output, embedded within the text describing the code. If you have the original R markdown file (file extension .Rmd), you can 'knit' the document into an HMTL, PDF, or Word file. Just be aware that the code executes a good number of sampling-based simulations, and these simulations take some time to complete. The parameter that determines how many samples are used in each of the sampling simulations is `n.samples`. You can change the value of `n.samples` to a small number (e.g., `10`) to speed up compilation.

An example problem

In this document, we are using the same example data set and problem already introduced in the lecture on sampling-based approaches to confidence intervals.

Let's say we have an experiment with 48 subjects, in which each subject saw 24 trials for each of two conditions (for a total of 48 trials per subject). On each trial, subjects saw a picture of an object and had to decide as quickly and accurately whether the picture depicts something animate (a living thing) or not. The two conditions manipulate whether the object is something that we see frequently in our live ("high frequency") or not ("low frequency"). We collected both reaction times of subjects' responses (`RT`), and whether those responses were accurate (`correct`). We hypothesize that more frequent objects will be responded to more quickly and accurately.

Sampling-based Type I error and power analyses

Another powerful application of sampling-based approaches is to obtain estimates of the **Type I error** rate (how often the analysis returns significance when in reality there is no effect in the underlying population) or **Type II error** rate of an analysis approach (how often we fail to find an effect that is actually present in the underlying population). Before we proceed, it is important to note that these error rates depend on both the *data* and the *analysis approach*.[1] That is, while one approach might yield the targeted Type I error rate (e.g., a Type I error rate of .05 for a significance criterion of $\alpha = .05$), another approach might fail to achieve this error rate on the same data. Conversely, approach A might work better for data set 1, whereas approach B works better for data set 2 (where "better" means achieving an error rate closer to the targeted error rate).

For Type I errors, we want to avoid in particular *anti-conservativity*—error rates above .05—but also *conservatity*—error rates lower than .05. For **power** (which is simply 1 - Type II error), we want to have as

[1] Additionally, Type I and II error rates depend on the assumptions we make while estimating them—a fact that is often overlooked. For example, journals might require a certain minimum of statistical power for publication, but typically do not specify how this power is estimated. We'll return to this point below.

much statistical power as possible. Whereas conventions for targeted Type I error rates tend to be pretty clear, conventions for what constitutes sufficient power tend to be less clear.

In this section, we go through how one can use the sampling-based approach to assess power and Type I error rates. These analyses are based conducted before collecting data (e.g., so that one can still change how much data to collect), but can also be conducted after data collection prior to the analysis (e.g., to guide which analysis approach one should choose), or even after analyses are already conducted (e.g., to assess how much we can trust our results).

The basic idea of the sampling-based approach to Type I error or power analysis is that we

(i) determine our assumed 'ground truth', i.e., the assumptions we want to make about the effects in the underlying population that we are interested in.
(ii) generate hypothetical results (e.g., data we might obtain from an experiment we plan to run, or have run) based on the assumed ground truth. This data generation could employ a *parametric* or *non-parametric* process (e.g., bootstrap).
(iii) analyze the hypothetical results with the intended analysis approach (or approach*es* if we want to compare different approaches).
(iv) repeat (ii) and (iii) many times.
(v) calculate how often we find a significant effect.

The only difference between Type I error and power simulations is in step (i)—whether we assume a null effect or an effect different from zero. If we generate the data under the assumption of a null effect, then the proportion of significant effects in step (v) provides an estimate of the Type I error rate. If, on the other hand, we generate the data under a non-null effect, then the proportion of significant effects in step (v) provides an estimate of the power to detect an effect of that size.

Parametrically estimating Type I error

Let's say we want to analyze both the RT and the accuracy data from our experiment with a *t*-test. Being conservative researchers, we are interested in estimating the Type I error rate of this approach. We don't want to be anti-conservative, as this would mean we might draw conclusions that are unlikely to hold, and describe result patterns that are unlikely to replicate. To illustrate this process, we again start with a scenario in which we have data from only one subject)this time, we'll go with Subject `3`). We first get the mean and SD for both RTs and accuracy from our data.

An example

If we apply a standard *t*-test to the original RT and accuracy data, we find a (highly) significant diffference in RTs but not in accuracy:

```
##
##  Two Sample t-test
##
## data:  RT by condition
## t = -32.828, df = 46, p-value < 2.2e-16
## alternative hypothesis: true difference in means between group high frequency and group low frequency
## 95 percent confidence interval:
##  -693.1695 -613.0756
## sample estimates:
## mean in group high frequency  mean in group low frequency
##                      367.970                     1021.093
```

```
##
##  Two Sample t-test
##
## data:  correct by condition
## t = 1.8127, df = 46, p-value = 0.07641
## alternative hypothesis: true difference in means between group high frequency and group low frequency
## 95 percent confidence interval:
##  -0.0138086  0.2638086
## sample estimates:
## mean in group high frequency  mean in group low frequency
##                        1.000                        0.875
```

For **step (i)** of our Type I error calculation, we take the observed mean and standard deviation (SD) of the data in our sample, and then make a function that generates data with those parameters under, e.g., the assumption of normality. For this, we ignore condition, since we want to simulate a scenario in which the ground truth is that there is *no* difference between the two conditions (i.e., a null effect):

```
## # A tibble: 1 x 4
##   RT_mean correct_mean RT_sd correct_sd
##     <dbl>        <dbl> <dbl>      <dbl>
## 1    695.        0.938  337.      0.245
```

For **step (ii)**, we write a function that can generate RT and accuracy data from these parameters. For the RTs we assume that they are drawn from a normal distribution with the mean and SD observed in our experiment for Subject 3. For accuracy, we assume that it is drawn from a Bernoulli distribution with the mean observed in our experiment for Subject 3.

```r
# A function to parametrically generate data from our thought
# experiment:
make_single_subject_data = function(n.trial, d.pars) {
  # This generates a data frame with one row for each unique
  # combination of the variable values handed to it. E.g., if
  # there are 24 trials (n.trial = 24), we will end up with
  # 48 rows (= 1 subject * 2 conditions * 48 trials)
  crossing(
    subject = 1,
    condition = c("low frequency","high frequency"),
    trial = 1:n.trial
  ) %>%
    # Sample the responses / outcomes / dependent variables, i.e.
    # the accuracy and RTs. d.pars is a data frame with the mean
    # and SD of the RT and accuracy distributions we want to sample
    # from.
    mutate(
      correct = rbinom(nrow(.), 1, d.pars$correct_mean[1]),
      RT = rnorm(nrow(.), d.pars$RT_mean[1], d.pars$RT_sd[1])
    ) %>%
    # Do some formatting and sort the data by subject, trial, and
    # condition. These lines of code are not necessary.
    as_tibble() %>%
    mutate_at(c("condition", "subject"), factor) %>%
    arrange(subject, trial, condition)
}
```

Here's an example draw from that function, giving us one random instance of our thought experiment. Notice that the RT and accuracy values for the two conditions are not identical. That's to be expected since each trial is generated as a draw from a random variable. The critical question is how this affects our conclusions about significant differences between the means of the conditions.

```
# Let's sample some data from our parametric generative process (step ii)
make_single_subject_data(n.trial, d.pars)
```

```
## # A tibble: 48 x 5
##    subject condition      trial correct    RT
##    <fct>   <fct>          <int>   <int> <dbl>
##  1 1       high frequency     1       1  143.
##  2 1       low frequency      1       1  709.
##  3 1       high frequency     2       1  743.
##  4 1       low frequency      2       1  500.
##  5 1       high frequency     3       1  808.
##  6 1       low frequency      3       1 1079.
##  7 1       high frequency     4       1  908.
##  8 1       low frequency      4       0  905.
##  9 1       high frequency     5       1  705.
## 10 1       low frequency      5       1  359.
## # ... with 38 more rows
```

For **step (iii)**, we write a function that applies the two t-tests to the data set, and stores the p-value for each test in a data.frame. To illustrate how this function works, we apply it to the original data and to one example data generated by our data generation process:

```
# a function that applies a t-test to both the RT and the
# accuracy data and then stores the two p-values in a data
# frame. We'll be using this function for all of the examples
# below.
do_test = function(data) {
  d = tibble(.rows = 1)

  if ("RT" %in% names(data)) {
    t1 = t.test(RT ~ condition, data = data, var.equal = T)
    d %<>%
      mutate(RT.p.value = t1$p.value)
  }
  if ("correct" %in% names(data)) {
    t2 = t.test(correct ~ condition, data = data, var.equal = T)
    d %<>%
      mutate(Accuracy.p.value = t2$p.value)
  }

  return(d)
}

# Apply our test function to the actual data (the individual subject we
# selected above).
do_test(d.subj)
```

```
## # A tibble: 1 x 2
```

```
##   RT.p.value Accuracy.p.value
##        <dbl>            <dbl>
## 1   1.43e-33           0.0764
```

```
# One draw from the data generated under the assumption of a null effect
# that is then analyzed in our do_test() function.
make_single_subject_data(n.trial, d.pars) %>%
  do_test()
```

```
## # A tibble: 1 x 2
##   RT.p.value Accuracy.p.value
##        <dbl>            <dbl>
## 1      0.450           0.0764
```

For **step (iv)**, we repeat the step (ii) and (iii) 1000 times, giving us the following (showing only the first 20 rows):

```
##    .n RT.p.value Accuracy.p.value
## 1   1 0.97295188       0.07641426
## 2   2 0.15280285       1.00000000
## 3   3 0.27909557       0.15495731
## 4   4 0.80447697       0.32254205
## 5   5 0.30896796       0.07641426
## 6   6 0.86036908       1.00000000
## 7   7 0.89218302       0.07641426
## 8   8 0.57385993       0.30640510
## 9   9 0.52708254       1.00000000
## 10 10 0.46379358       0.39350945
## 11 11 0.56347759       1.00000000
## 12 12 0.06144075       0.56080389
## 13 13         NA               NA
## 14 14 0.46192213       0.64509890
## 15 15 0.85594479       0.32254205
## 16 16         NA               NA
## 17 17 0.34495436       0.30640510
## 18 18 0.76092846       0.56080389
## 19 19 0.73388361       1.00000000
## 20 20 0.10788079       0.56080389
```

Finally, for **step (v)**, we calculate the proportion of the 1000 samples for which the effects of conditions on RTs or accuracy were significant, which corresponds to the estimated Type I error rate of the *t*-tests:

```
##   RT.Type_I Accuracy.Type_I
## 1     0.044           0.035
```

These Type I error rates look OKish for the RT analysis, but clearly conservative for the analysis of accuracy.

As mentioned in the footnote above, any Type I error calculation is only as good as its assumptions. In this particular case, we've made some problematic assumptions. For example, we assumed that RTs are drawn from a normal distribution, but that would predict that there can be negative RTs (the tails of a normal distribution are infinitely long). Such problematic assumptions do not *necessarily* lead to wrong estimates of the Type I error rate, but they *can*.

The consequences of making problematic assumptions about the ground truth (step (i))

In this particular example, we actually *know* the ground truth. That's because the data analyzed here are actually not data from a psycholinguistic experiment. Instead, we generated them silently at the top of this document, using a lognormal (rather than normal) distribution for the RTs and a Bernoulli distribution for the accuracies (plus normally distributed individual differences across the subjects). This means that we can compare the Type I error rate of the t-test obtained above to the Type I error rate of the same t-test if the data is generated following the ground truth.

The following repeats steps (ii) - (v) under the correct ground truth. For RTs, we find that we get a pretty similar Type I error rate as above, despite the fact that our assumptions have changed. For the analysis of accuracy, however, we see a substantially lower Type I error rate than the target of .05. That is, the t-test is actually rather conservative for this particular data set.

```
##   RT.Type_I Accuracy.Type_I
## 1      0.06           0.025
```

These results suggest that we should not use a t-test for the analysis of our accuracy results. Indeed, this problem of t-tests and ANOVA for the analysis of categorical outcomes is well-known. But even if we didn't know anything about it, the sampling-based approach has provided us with a tool to notice that something is off.

What if we don't know the ground truth? Bootstrap!

In the above example, we knew the ground truth and this let us assess whether the assumptions we made for our Type I error analysis were problematic. But that is rarely the case. How then, can we avoid problematic assumptions? While there is no magic bullet—nothing that will always work—the non-parametric bootstrap we've used above to obtain confidence intervals also allows us to get Type I error estimates, without assuming, for example, normality. Specifically, we can bootstrap from the data we have. Specifically, we can substitute steps (i) and (ii) by sampling without replacement data points from subject 2.

```
# A bootstrap function to generate data
bootstrap_single_subject_data = function(n.trial, d) {
  d %>%
    ungroup() %>%
    # Sample twice as many rows as n.trials since there are two conditions
    sample_n(n.trial * 2, replace = T) %>%
    # Randomly assign data to conditions (since we are simulating a null effect)
    mutate(condition = rep(c("high frequency", "low frequency"), n.trial))
}

bootstrap_single_subject_data(n.trial, d.subj)
```

```
## # A tibble: 48 x 5
##    subject condition      trial correct    RT
##    <fct>   <chr>          <int>   <int> <dbl>
##  1 3       high frequency    23       1  964.
##  2 3       low frequency     22       0 1264.
##  3 3       high frequency     1       1  380.
##  4 3       low frequency      6       1  907.
##  5 3       high frequency    12       1 1041.
```

```
##  6 3       low frequency     23       1  964.
##  7 3       high frequency     2       1  374.
##  8 3       low frequency     17       1  340.
##  9 3       high frequency    14       1  370.
## 10 3       low frequency     21       1  347.
## # ... with 38 more rows
```

Applying the same steps (iii) - (v) as above, we find that the Type I error rate of both analyses seems acceptable:

```
d.type1.boot =
  plyr::rdply(
    .n = n.samples,
    function(i) {
      dd = try(bootstrap_single_subject_data(n.trial, d.subj) %>%
                 do_test(), silent = TRUE)
      if (class(dd) == "try-error")
        dd = data.frame(
          RT.p.value = NA,
          Accuracy.p.value = NA
        )
      return(dd)
    })

d.type1.boot %>%
  get_TypeI()
```

```
##   RT.Type_I Accuracy.Type_I
## 1     0.053           0.024
```

Power

To calculate power, we proceed in parallel to the approach described above for Type I error rates. For power, too, we can use parametric or non-parametric approaches. The following example shows the parametric approach to simulated the data from Subject 3. We can modify the function used above, so that it can accommodate both Type I error analyses (for effect = 0) and power analyses (for effects != 0). For the sake of simplicity, we focus on only RTs.

```
# A function to parametrically generate data with an effect
make_single_subject_data = function(n.trial, d.pars, effect = 0) {
  crossing(
    subject = 1,
    condition = c("low frequency","high frequency"),
    trial = 1:n.trial
  ) %>%
  mutate(
    RT = rnorm(
      nrow(.),
      # Here we are adding the effect to the mean of the normal
      # distribution we are sampling from:
      d.pars$RT_mean[1] + ifelse(condition == "low frequency", -effect, +effect),
      d.pars$RT_sd[1])
```

```
  ) %>%
  as_tibble() %>%
  mutate_at(c("condition", "subject"), factor) %>%
  arrange(subject, trial, condition)
}
```

Here's an example draw from this revised function, for a hypothetical (huge) difference of 250 milliseconds between the means of the two conditions:

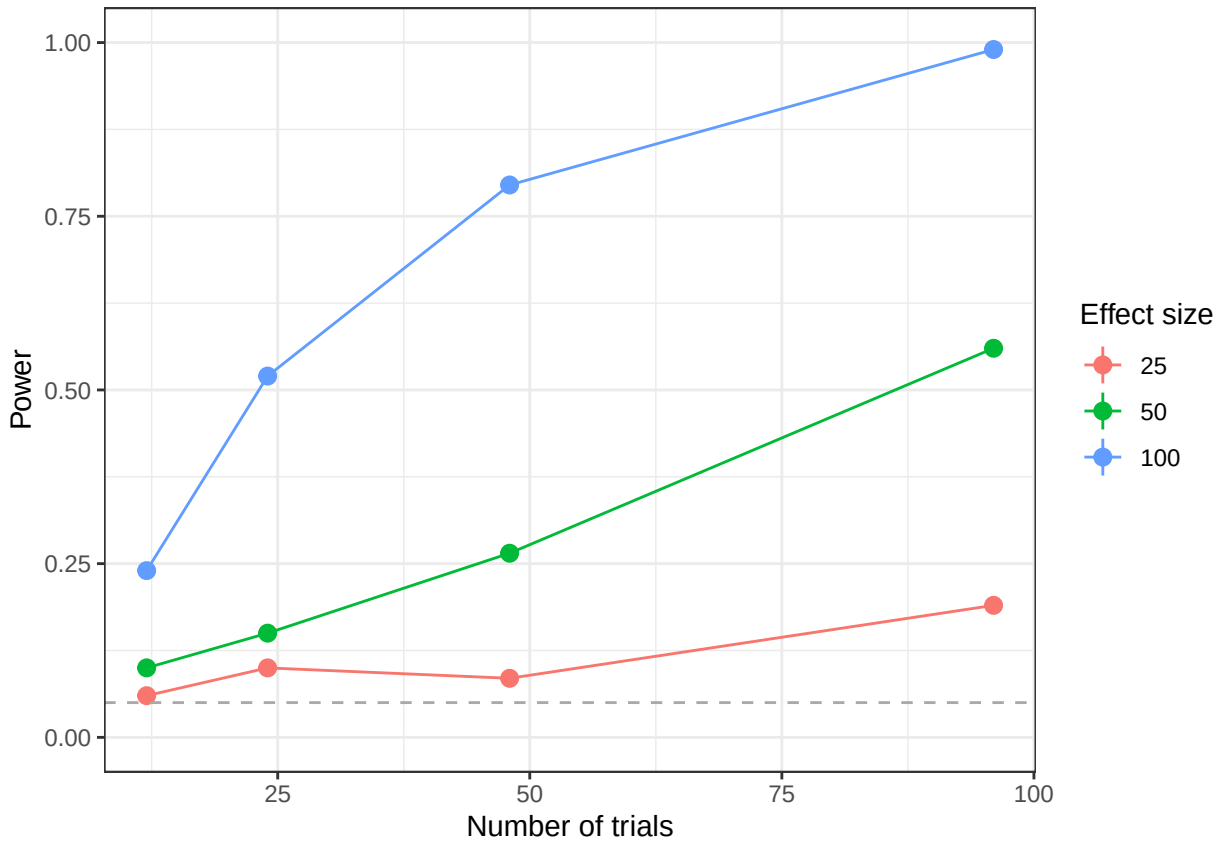
```
make_single_subject_data(n.trial, d.pars, effect = 250)
```

```
## # A tibble: 48 x 4
##    subject condition      trial     RT
##    <fct>   <fct>          <int>  <dbl>
##  1 1       high frequency     1  842.
##  2 1       low frequency      1 1003.
##  3 1       high frequency     2  625.
##  4 1       low frequency      2  975.
##  5 1       high frequency     3 1170.
##  6 1       low frequency      3  -71.2
##  7 1       high frequency     4 1403.
##  8 1       low frequency      4  985.
##  9 1       high frequency     5 1041.
## 10 1       low frequency      5  559.
## # ... with 38 more rows
```

With this simple change in the data generation—i.e., step (i)—we can calculate the power for, e.g., a combination of different amounts of data (12, 24, 48, or 96 trials) and different effect sizes (25, 50, and 100 ms):

```
get_power = . %>%
  get_TypeI() %>%
  rename_all(., .funs = ~gsub("Type_I", "Power", .x))

sample_power = function(d.pars, n.trials, effects, n.samples = n.samples) {
  d = tibble(.rows = 0)
  for (n.trial in n.trials) {
    for (effect in effects) {
      d %<>% rbind(
        plyr::rdply(
          .n = n.samples,
          function(i) {
            dd = try(
              make_single_subject_data(n.trial, d.pars, effect = effect)  %>%
                do_test(), silent = TRUE)
            if (class(dd) == "try-error")
              dd = data.frame(RT.p.value = NA)
            return(dd)
          }) %>%
          mutate(effect = effect, n.trial = n.trial))
    }
  }
```

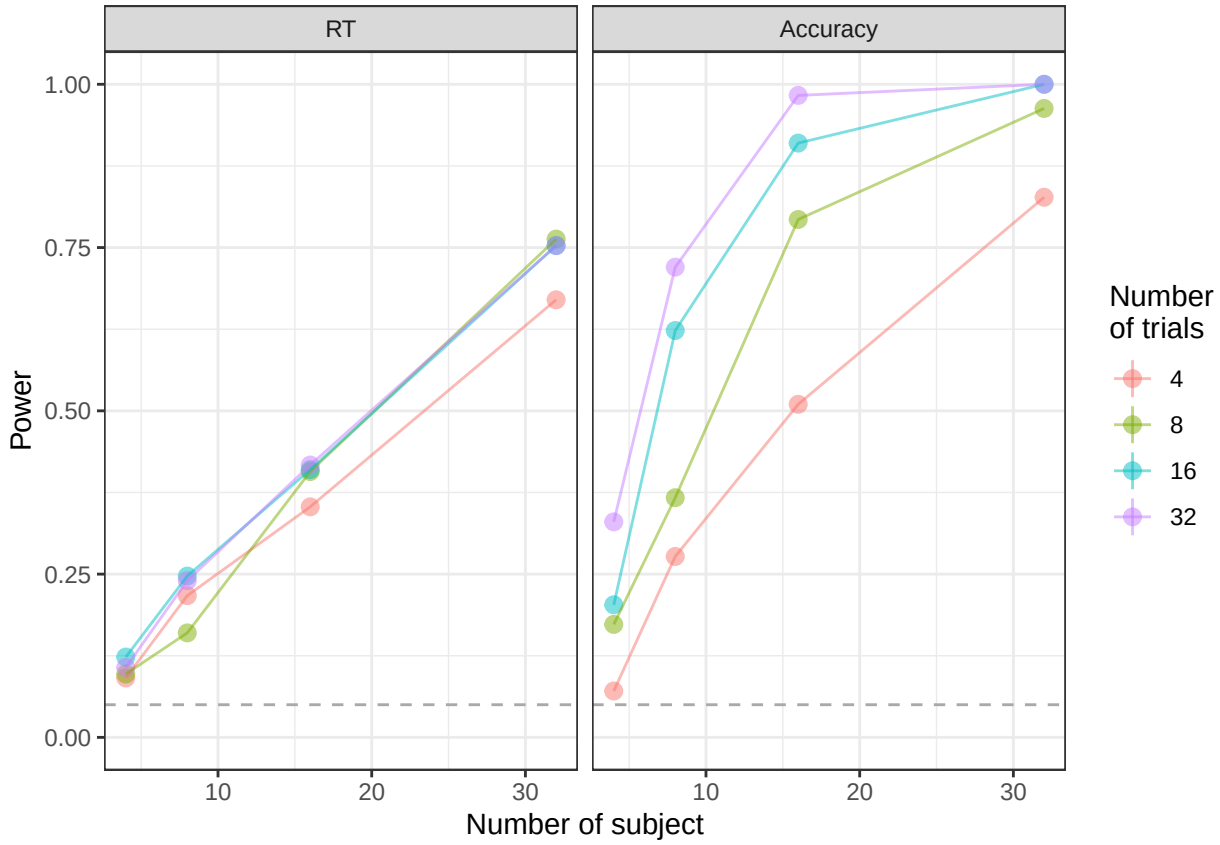
```
  d %<>%
    group_by(effect, n.trial) %>%
    get_power()

  return(d)
}

sample_power(
  n.trials = c(12, 24, 48, 96),
  effects = c(25, 50, 100),
  d.pars = d.pars,
  n.samples = 200
) %>%
  ggplot(aes(x = n.trial, y = RT.Power, color = factor(effect))) +
  stat_summary(fun.data = mean_cl_boot, geom = "pointrange") +
  stat_summary(fun = mean, geom = "line") +
  # Adding a line to indicate the 'power' we'd get from the Type I error
  # alone
  geom_hline(yintercept = .05, linetype = 2, color = "darkgray") +
  scale_x_continuous("Number of trials") +
  scale_y_continuous("Power", limits = c(0,1)) +
  scale_color_discrete("Effect size") + theme_bw()
```



Power in grouped data

The same approach can be applied to grouped data. Here is an example using a generation function that is similar to the one used to generate the (simulated) results that we introduced at the beginning of this document. We can then ask how the power of a *paired* t-test over the by-participant means of RTs and accuracy scales as a function of the number of trials and the number of subjects. If you are working with similar repeated-measures data, have a look at the code in the R markdown source.

Note how we stand to gain much less power from increasing the number of trials per subjects, compared to increasing the number of subjects (e.g., 4 subjects with 8 trials each is the same amount of total data as 8 subjects with 4 trials each, but the latter gives us more power). Of course, this depends on the relative variability *within* vs. *between* subjects, but the scenario depicted here holds for many of our experiments.



Researcher's degrees of freedom: a sampling-based case study

Let's say we collect 30 subjects' worth of data (with 24 trials each), from the same hypothetical experiment used in the power analyses right above. We then conduct a paired t-test on both the RTs and the accuracy:

```
d = make_multi_subject_data(n.trial = 16, n.subject = 24)

d %>%
  do_multi_subject_test()
```

```
## # A tibble: 1 x 2
##   RT.p.value Accuracy.p.value
```

```
##        <dbl>            <dbl>
## 1      0.620            0.638
```

The result of the t-test for the reaction times looks 'promising'. They are significant, and one might say it is "approaching significance". So we decide that we need more statistical power to understand the effect. We collect data from 8 more subjects and then test again:

```
d %<>%
  rbind(
    make_multi_subject_data(n.trial = 16, n.subject = 8))

d %>%
  do_multi_subject_test()
```

```
## # A tibble: 1 x 2
##   RT.p.value Accuracy.p.value
##        <dbl>            <dbl>
## 1      0.369            0.509
```

It seems like we were on the right track! There is indeed a significant effect of high vs. low frequency on reaction times. We decide to submit the result for publication.

Eight years later, a team of undergraduate researchers decides to replicate our study. They succeed in following the same design, procedure, etc., and sample from the same underlying population. They are weary of the small number of trials and subjects. They decided to run twice the number of trials per subject (32) and four times the number of subjects of the original study (128), for a total of *eight times the original data*. Using the same paired t-test approach, they find:

```
make_multi_subject_data(n.trial = 32, n.subject = 128) %>%
  do_multi_subject_test()
```

```
## # A tibble: 1 x 2
##   RT.p.value Accuracy.p.value
##        <dbl>            <dbl>
## 1      0.683            0.175
```

Which study should we trust? A failure to replicate is *not necessarily* due to any fault of the original researchers (or the replication team): even if the test we employ achieves the targeted Type I error rate of 5%, one in twenty experiments will return a significant effect even if the underlying effect is null.

But is there something that the original team could have done better? If you don't know the answer, make sure to read Nelson et al. (2008) article in the *Annual Review of Psychology*.

Illustrating the consequences of the run-test-run-test-... scheme (incremental testing)

Let's use the sampling-based approach to investigate the consequences of this incremental approach. We first write a function that collects data in 10 steps (of 10 subjects with 16 trials each), each time testing all the data collected so far.

In the output, we see the p-values for each batch of 10 subjects, both for the t-test on RTs and the t-test on accuracy. Next to it, we see the column `RT.significant`, indicating whether the p-value for the RT

analysis was significant ($< .05$) *in that batch*. Next to that, the columns `RT.significant.incremental` tells us whether the *p*-value *in this or any preceding batch* was significant. This captures the **incremental testing** approach, in which we would stop any time we have found significance. For example, even if I had planned to in theory collect data from up to 10 batches of 10 subjects (testing for an effect after each batch has been collected), under the incremental testing approach I'd stop as soon as I found significance on any of the earlier batches. I'd then conclude and report a significant effect. That's why the column `RT.significant.incremental` has the value `TRUE` for any batch that follow a batch on which I have found significance.

```
run_test_run_test = function(n.trial = 16, n.subject = 10, n.batch = 10) {
  d = plyr::rdply(
    .n = n.batch, .id = "batch",
    make_multi_subject_data(n.trial = n.trial, n.subject = n.subject)) %>%
    group_by(batch, subject, condition) %>%
    dplyr::summarise_at(c("correct", "RT"), mean)

  d.t = tibble(.rows = 0)
  for (i in 1:max(d$batch)) {
    d.t %<>%
      rbind(
      do_multi_subject_test(d %>%
                              filter(batch <= i)) %>%
        mutate(batch = i))
  }

  d.t %>%
    mutate(
      RT.significant = RT.p.value < .05,
      RT.significant.incremental = cumany(RT.significant),
      Acc.significant = Accuracy.p.value < .05,
      Acc.significant.incremental = cumany(Acc.significant)
      )
}

run_test_run_test()
```

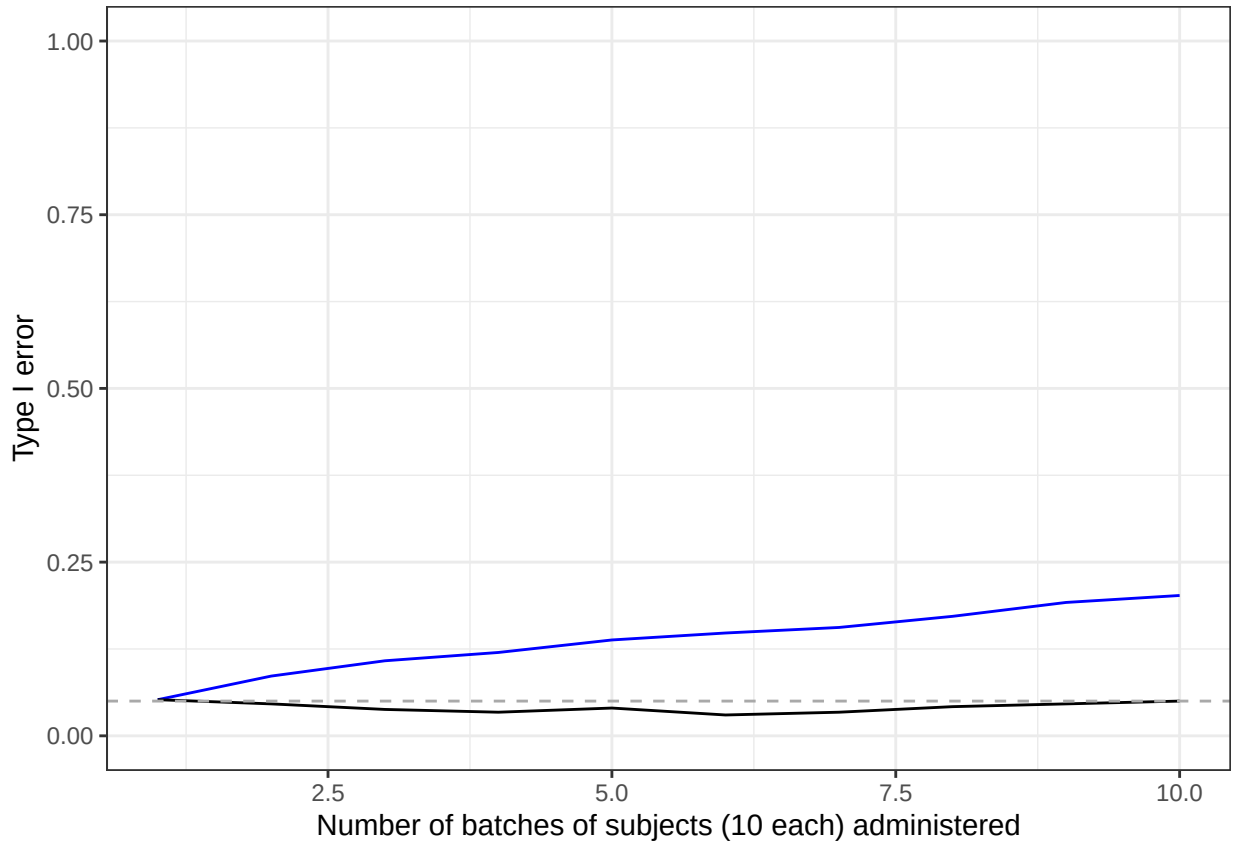
```
## # A tibble: 10 x 7
##    RT.p.value Accuracy.p.value batch RT.significant RT.significant.incremental
##         <dbl>            <dbl> <int> <lgl>          <lgl>
##  1      0.464            0.298     1 FALSE          FALSE
##  2      0.416            0.690     2 FALSE          FALSE
##  3      0.706            0.447     3 FALSE          FALSE
##  4      0.860            0.588     4 FALSE          FALSE
##  5      0.551            0.699     5 FALSE          FALSE
##  6      0.921            0.875     6 FALSE          FALSE
##  7      0.589            0.593     7 FALSE          FALSE
##  8      0.343            0.846     8 FALSE          FALSE
##  9      0.514            0.860     9 FALSE          FALSE
## 10      0.393            1        10 FALSE          FALSE
## # ... with 2 more variables: Acc.significant <lgl>,
## #   Acc.significant.incremental <lgl>
```

Now we can ask what happens if we repeated this thought experiment many times. The black line in the following plot shows the estimated Type I error rate if we plan how many subjects we run and only test once

(e.g., we might plan to run 6 batches, i.e., 60 subjects and then test and stop, regardless of the result we find). The blue line shows the Type I error rate if we test after every batch of subject and stop if we find significance.

```
d.sim = plyr::rdply(
  .n = 500,
  run_test_run_test()
)

d.sim %>%
  group_by(.n, batch) %>%
  summarise(
    RT.TypeI.cumulative = mean(RT.significant),
    Acc.TypeI.cumulative = mean(Acc.significant),
    RT.TypeI.incremental = mean(RT.significant.incremental),
    Acc.TypeI.incremental = mean(Acc.significant.incremental)) %>%
  ggplot(aes(x = batch, y = RT.TypeI.incremental)) +
  stat_summary(fun = mean, geom = "line", color = "blue") +
  stat_summary(fun = mean, geom = "line", color = "black", aes(y = RT.TypeI.cumulative)) +
  # Adding a line to indicate the 'power' we'd get from the Type I error
  # alone
  geom_hline(yintercept = .05, linetype = 2, color = "darkgray") +
  scale_x_continuous("Number of batches of subjects (10 each) administered") +
  scale_y_continuous("Type I error", limits = c(0,1)) + theme_bw()
```

```
## `summarise()` has grouped output by '.n'. You can override using the `.groups`
## argument.
```



This illustrates the problem—an inflated Type I error—of the incremental testing approach. Simmons et al. (2011) in *Psychological Science*, summarized also in the Nelson et al. (2018) article, discuss incremental testing as one of several common practices and researchers' degrees of freedom that have under-appreciated consequences on the Type I error of our studies.