Notes on simulation

Jono Tuke

Fri 31 Dec 2021

Contents

Introduction

The following is notes on the analysis with examples. The complete analysis was performed using the package targets:

https://books.ropensci.org/targets/

The complete analysis is visualised in the Appendix, along with all of the code.

To repeat the analysis, you can run the `makefile`. This is achieved by opening the project `2020 animal_simulation.Rproj` in RStudio, and then running the build command using `CMD+SHIFT+B` on a Mac. The shortcut for Windows is not known by the author.

Basic form for simulated data

The basic form, referred to as a `simDB`, is a data frame such that that each row is a second, it has the following columns

- **time:** the second of interest.
- **type:** the type of behaviour

The types of behaviour will of two forms:

- background
- event / state

So we are interested in the event / state behaviour.

Notation

We will represent the data by

$$x_1, x_2, \ldots, x_n,$$

where $n = 3600$ in our case and

$$x_i = \begin{cases} 0, & \text{if time-point } i \text{ is background,} \\ 1, & \text{if time-point } i \text{ is an event.} \end{cases}$$

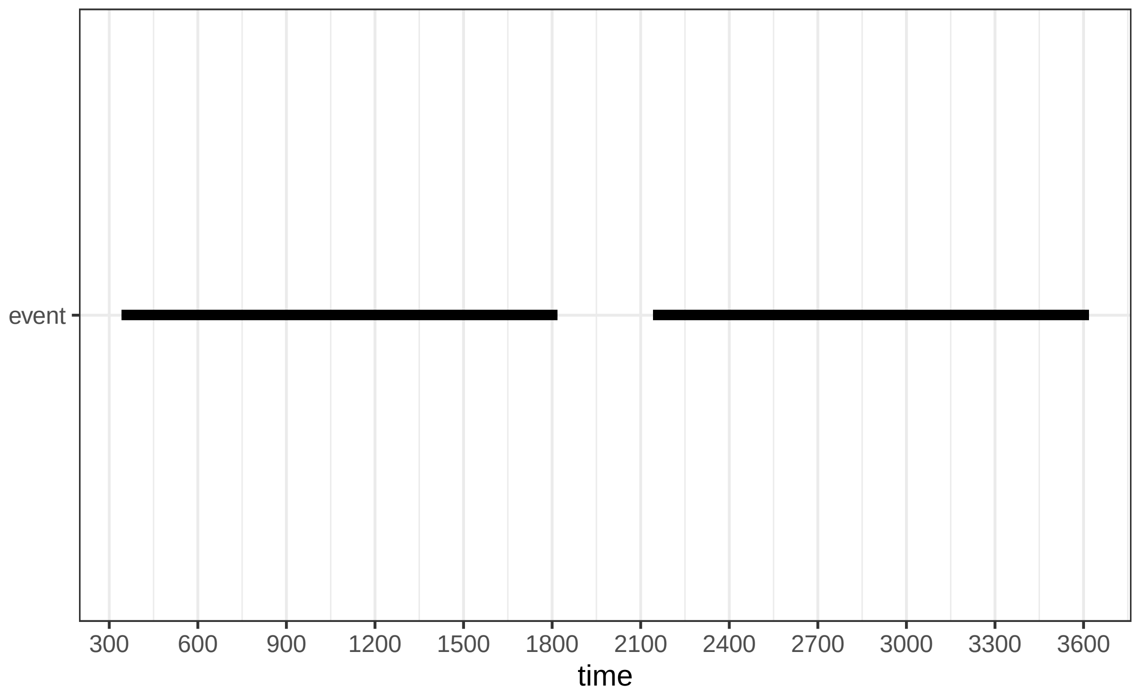
Example

Consider the following data frame

```
tar_load(example_simDB)
example_simDB
```

```
## # A tibble: 3,600 x 2
##     time type
##    <int> <chr>
##  1     1 background
##  2     2 background
##  3     3 background
##  4     4 background
##  5     5 background
##  6     6 background
##  7     7 background
##  8     8 background
##  9     9 background
## 10    10 background
## # ... with 3,590 more rows
```

```
plot_simDB(example_simDB)
```



Percent of time event occurs

The key measure we are interested in is the percent of the time that the event occurs. We stick to proportion as this is nicer mathematically, and then convert to percent for presentation.

It is defined as

$$p_{GS} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

```
get_prop(example_simDB) %>% scales::percent()
```

```
## [1] "80%"
```

Simulation method

Response frequency

In these simulations, we have a behaviour that is short (1 second). The behaviour occurs

- frequent (every 3 seconds),
- moderate (every 30 seconds), or
- infrequent (every 300 seconds).

The function `sim_response_freq()` produces a simDB with these properties. It works by splitting the time into blocks of length equal to the frequency, so for moderate frequency, we have blocks of 30 seconds, then one of the seconds in this time are randomly selected to be the event.

So we split the data

$$x_1, x_2, \ldots, x_n,$$

where

$$x_i = \begin{cases} 0, & \text{if time-point } i \text{ is background,} \\ 1, & \text{if time-point } i \text{ is an event.} \end{cases}$$

into K blocks

Block 1

$$x_1, x_2, \ldots, x_\delta$$

Block 2

$$x_{\delta+1}, x_{\delta+2}, \ldots, x_{2\delta}$$

So we have

Block i

$$x_{(i-1)\delta+1}, x_{(i-1)\delta+2}, \ldots, x_{i\delta}$$

The final block is

$$x_{(k-1)\delta+1}, x_{(k-1)\delta+2}, \ldots, x_n,$$

where

$$k = \left\lceil \frac{n}{\delta} \right\rceil .$$

We then randomly select one of the time points in the block with equal probability,

$$x_j, j \in \{x_{(i-1)\delta+1}, x_{(i-1)\delta+2}, \ldots, x_{i\delta}\}$$

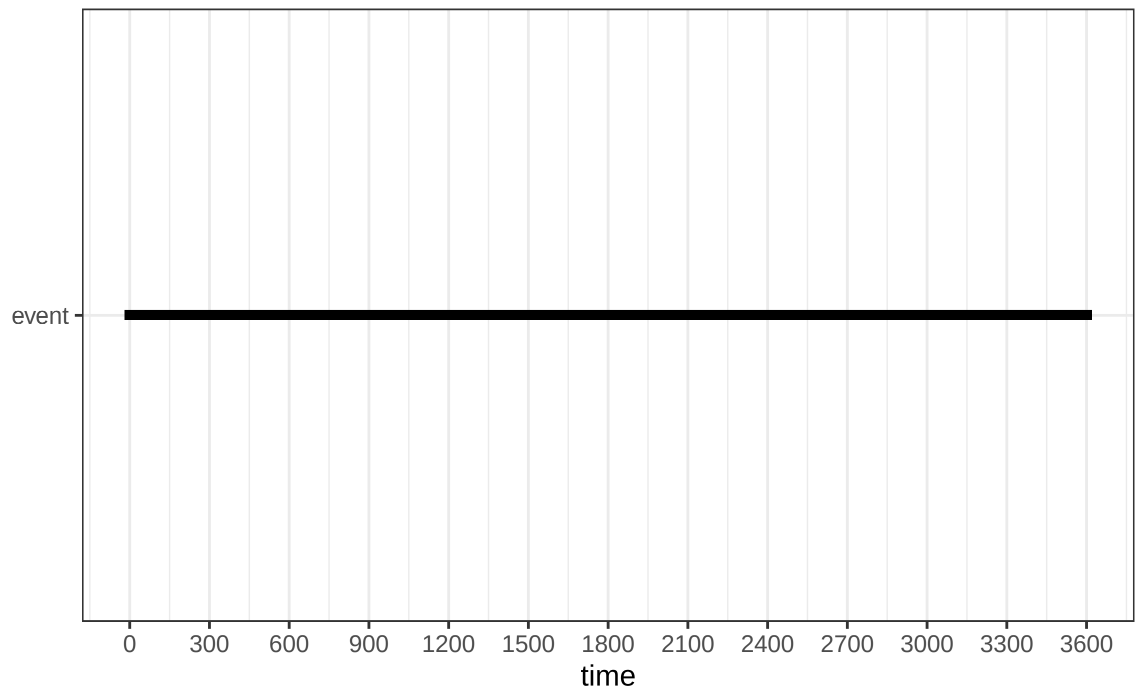
then we set

$$x_j = 1,$$

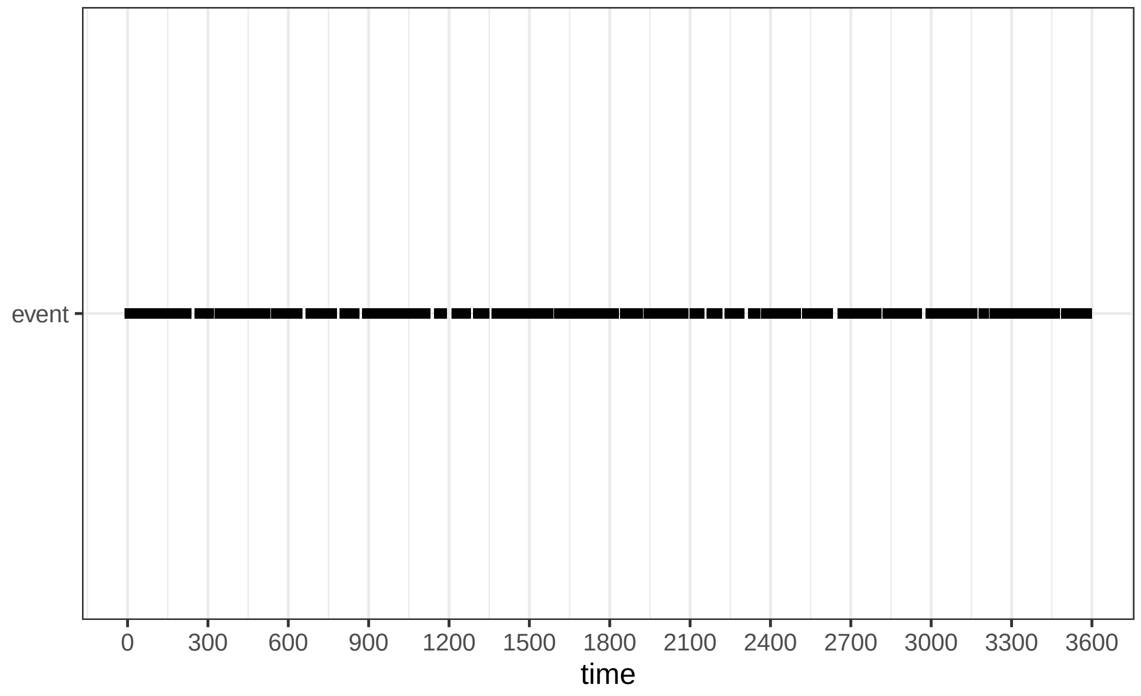
and

$$x_i = 0, i \in \{x_{(i-1)\delta+1}, x_{(i-1)\delta+2}, \ldots, x_{i\delta}\}, i \neq j.$$

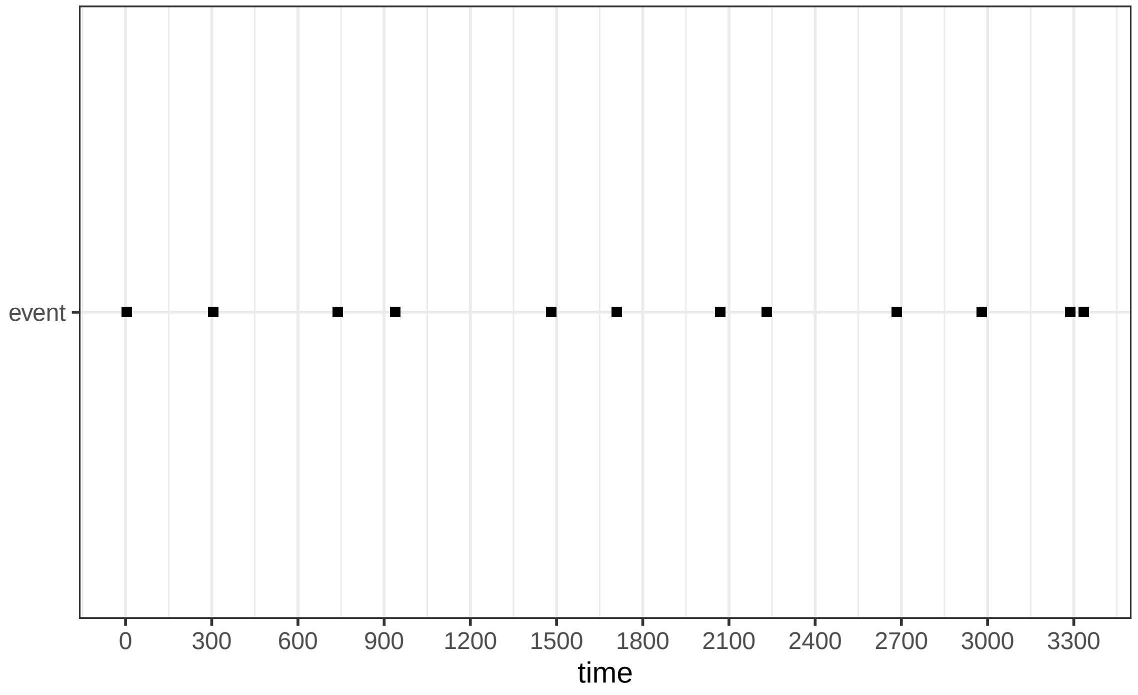
```
sim_response_freq(3) %>% plot_simDB()
```



```r
sim_response_freq(30) %>% plot_simDB()
```



```r
sim_response_freq(300) %>% plot_simDB()
```



```
sim_response_freq(3) %>% get_prop()
```

```
## [1] 0.3333333
```

```
sim_response_freq(30) %>% get_prop()
```

```
## [1] 0.03333333
```

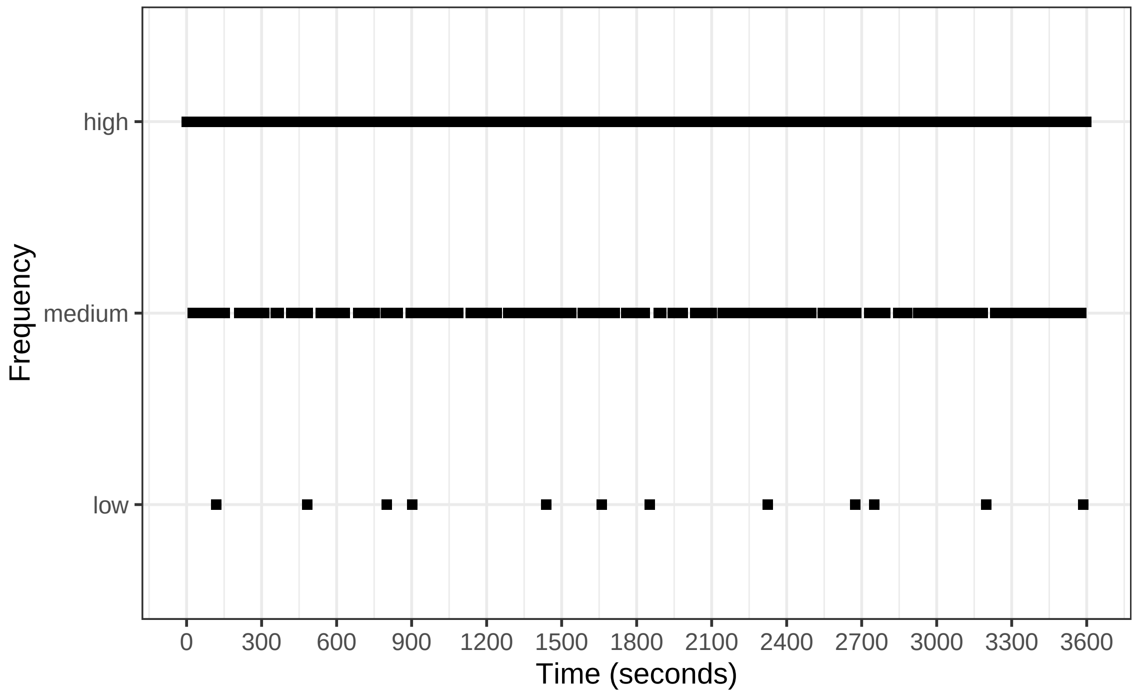
```
sim_response_freq(300) %>% get_prop()
```

```
## [1] 0.003333333
```

Joint plot

```
sim_response_freq(3) %>%
  add_column(rate = "high") %>%
  bind_rows(
    sim_response_freq(30) %>%
      add_column(rate = "medium")
  ) %>%
  bind_rows(
    sim_response_freq(300) %>%
      add_column(rate = "low")
  ) %>%
  filter(
    type == "event"
  ) %>%
  mutate(
    rate = factor(rate, levels = c("low", "medium", "high"))
  ) %>%
  ggplot(
    aes(time, rate)
  ) + geom_point(shape = "square") +
```

```r
  labs(y = "Frequency", x = "Time (seconds)") +
    scale_x_continuous(breaks = seq(0, 3600, 300))
```



```r
ggsave("figs/sim_RF.jpg", width = 10, height = 6)
```

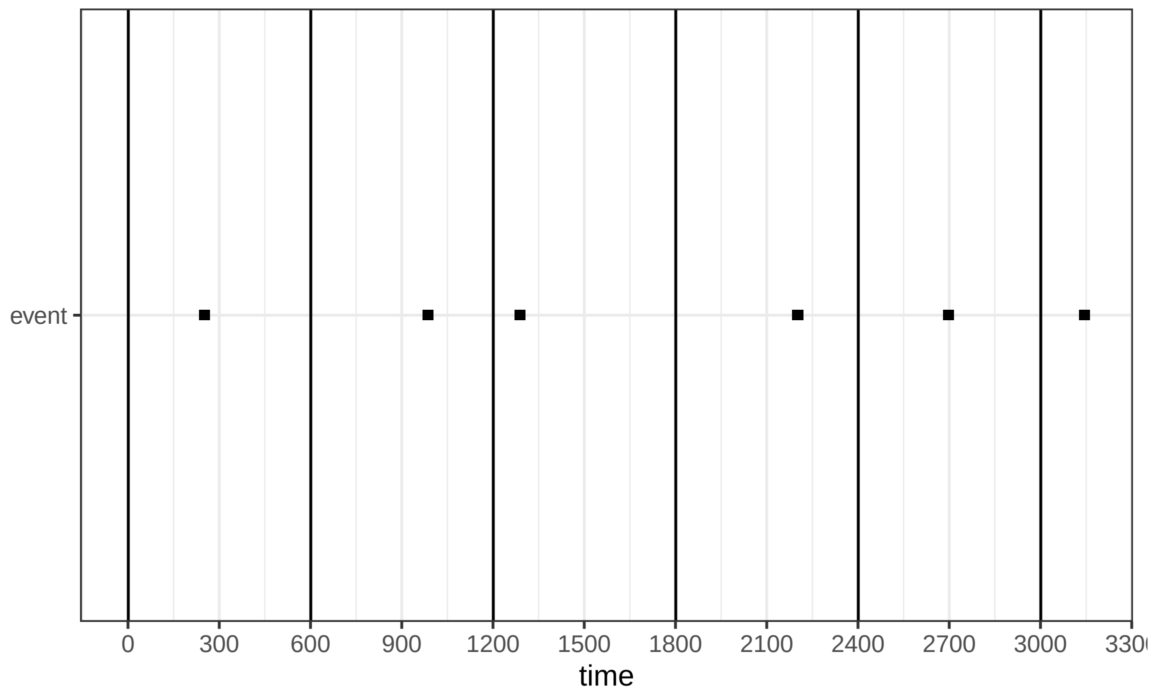
Response duration

In this simulation, the duration is

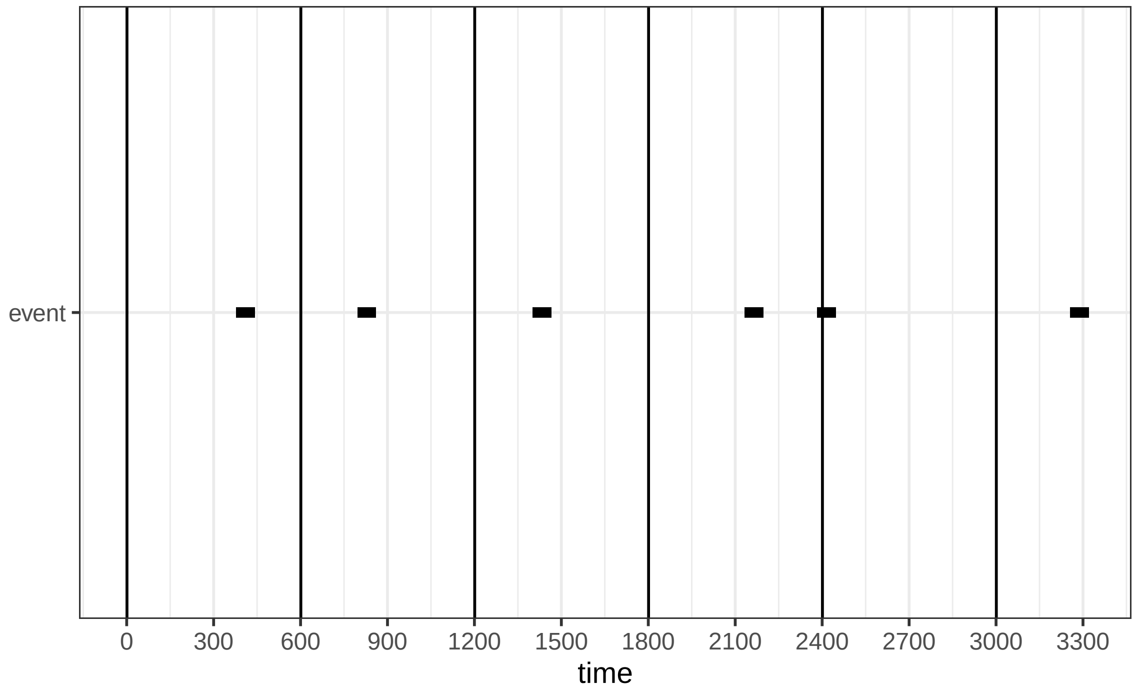
- short (3 second),
- medium (30 seconds) and
- long (300 seconds).

A single event occurs in each 10 minute interval, and is randomly placed in this interval.

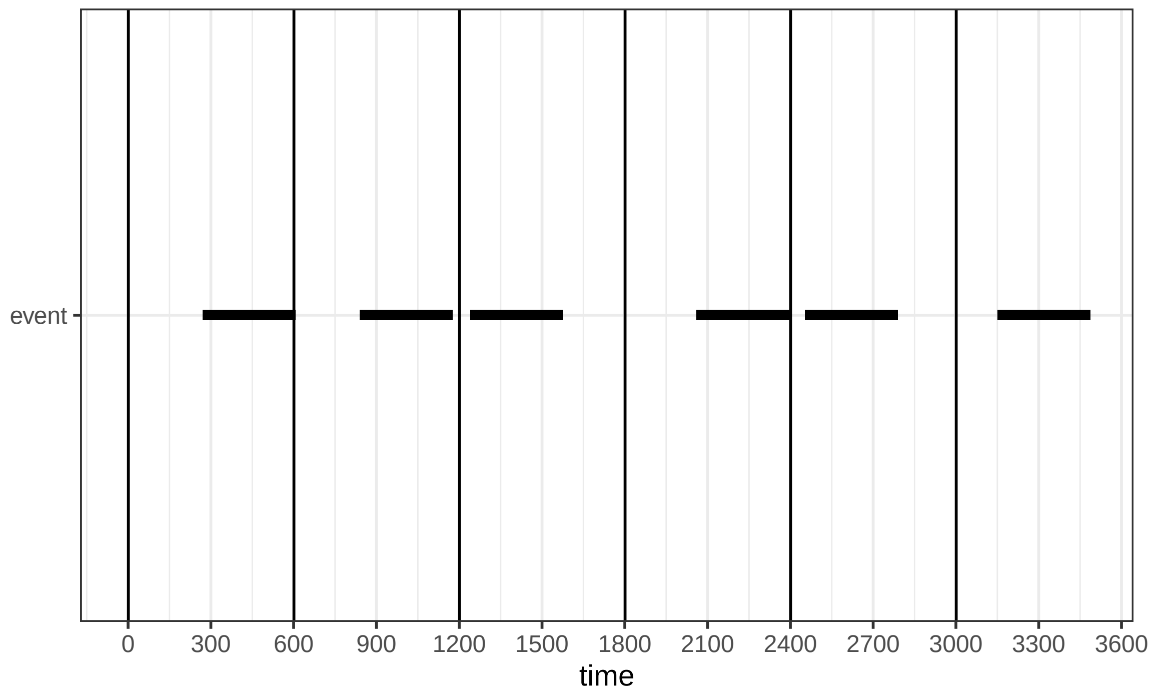
```r
sim_response_duration(3) %>% plot_simDB() + geom_vline(xintercept = seq(1, 3600, 600))
```



```r
sim_response_duration(30) %>% plot_simDB() + geom_vline(xintercept = seq(1, 3600, 600))
```



```r
sim_response_duration(300) %>% plot_simDB() + geom_vline(xintercept = seq(1, 3600, 600))
```



```r
sim_response_duration(3) %>% get_prop()
```

```
## [1] 0.005
```

```r
sim_response_duration(30) %>% get_prop()
```

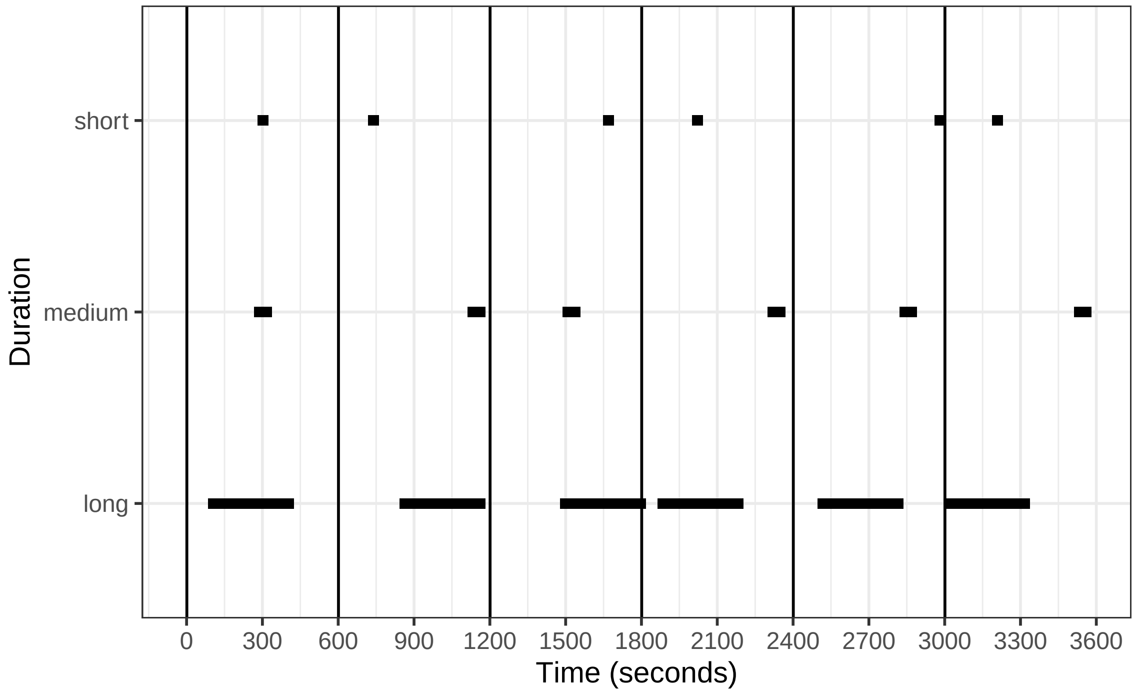
```
## [1] 0.05
```

```r
sim_response_duration(300) %>% get_prop()
```

```
## [1] 0.5
```

Joint plot

```r
sim_response_duration(3) %>%
  add_column(rate = "short") %>%
  bind_rows(
    sim_response_duration(30) %>%
      add_column(rate = "medium")
  ) %>%
  bind_rows(
    sim_response_duration(300) %>%
      add_column(rate = "long")
  ) %>%
  filter(
    type == "event"
  ) %>%
  mutate(
    rate = factor(rate, levels = c("long", "medium", "short"))
  ) %>%
  ggplot(
    aes(time, rate)
  ) + geom_point(shape = "square") +
  labs(y = "Duration", x = "Time (seconds)") +
    scale_x_continuous(breaks = seq(0, 3600, 300)) +
  geom_vline(xintercept = seq(1, 3600, 600))
```



```
ggsave("figs/sim_RD.jpg", width = 10, height = 6)
```

Sampling methods

So we have two sampling methods:

Pinpoint sampling

In pinpoint sampling, we sample at regular intervals, so for example, if the continuous data is represented by

$$x_1, x_2, \ldots, x_n,$$

where $n = 3600$ in our case and

$$x_i = \begin{cases} 0, & \text{if time-point } i \text{ is background,} \\ 1, & \text{if time-point } i \text{ is an event.} \end{cases}$$

Then if we have an interval of δ, then the pinpoint sample is

$$x_1, x_{1+\delta}, x_{1+2\delta}, \ldots, x_{1+k\delta},$$

where

$$k = \left\lfloor \frac{n-1}{\delta} \right\rfloor$$

The function `pin_point_sampling()` takes an interval and a `simDB` and returns the proportion of the time-points that are an event, i.e.

$$p_{pp}(\delta) = \frac{x_1 + x_{1+\delta} + x_{1+2\delta} + \ldots + x_{1+k\delta},}{k+1}$$

```
pin_point_sampling(example_simDB, 3)
```

```
## # A tibble: 1 x 3
##   method delta     p
##   <chr>  <dbl> <dbl>
## 1 PP         3   0.8
```

```
pin_point_sampling(example_simDB, 30)
```

```
## # A tibble: 1 x 3
##   method delta     p
##   <chr>  <dbl> <dbl>
## 1 PP        30   0.8
```

```
pin_point_sampling(example_simDB, 300)
```

```
## # A tibble: 1 x 3
##   method delta     p
##   <chr>  <dbl> <dbl>
## 1 PP       300 0.667
```

One-zero sampling

In one-zero sampling, we split the continuous data into contiguous blocks of length δ, i.e.,

Block 1

$$x_1, x_2, \ldots, x_\delta$$

Block 2

$$x_{\delta+1}, x_{\delta+2}, \ldots, x_{2\delta}$$

So we have

Block i

$$x_{(i-1)\delta+1}, x_{(i-1)\delta+2}, \ldots, x_{i\delta}$$

The final block is

$$x_{(k-1)\delta+1}, x_{(k-1)\delta+2}, \ldots, x_n,$$

where

$$k = \left\lceil \frac{n}{\delta} \right\rceil.$$

Note that this final block may not be the same length as the previous blocks.

If we let the result for Block i be denoted by y_i, then we have

$$y_i = \begin{cases} 0, & \text{if all } x_{(i-1)\delta+1}, x_{(i-1)\delta+2}, \ldots, x_{i\delta} = 0 \\ 1, & \text{if at least one of } x_{(i-1)\delta+1}, x_{(i-1)\delta+2}, \ldots, x_{i\delta} = 1. \end{cases}$$

Then

$$p_{01}(\delta) = \frac{\sum_{i=1}^{k} y_i}{k}$$

The function `one_zero_sampling()` takes a `simDB` and a δ, and returns the proportion.

```
one_zero_sampling(example_simDB, 3)
```

```
## # A tibble: 1 x 3
##   method delta     p
##   <chr>  <dbl> <dbl>
## 1 01         3 0.802
```

```
one_zero_sampling(example_simDB, 30)
```

```
## # A tibble: 1 x 3
##   method delta     p
##   <chr>  <dbl> <dbl>
## 1 01        30 0.817
```

```
one_zero_sampling(example_simDB, 300)
```

```
## # A tibble: 1 x 3
##   method delta     p
##   <chr>  <dbl> <dbl>
## 1 01       300 0.833
```

Simulation protocol

So for each simulation, we have

- method used - response frequency (RF), or response duration (RD),
- parameter - 3, 30, 300 interval (RF), or duration (RD).

This gives 6 different types of simulation, which we will repeat 100 times each giving 600 simulations.

Discussion with Eduardo

Frequency

Table 1: Classification of frequency

Percent	Classification
25-50%	Frequent
10-15%	Moderate
1-2%	Infrequent

Sampling protocol

For each simulation, we have

- sampling method - pinpoint - intervals 3, 30, and 300, while for one-zero, we sample every 10 minutes for duration of 3, 30, 300 seconds.

So again we have 6 sampling procedures, that will give use altogether 3600 measurements.

Results

For each combination of simulation method, simulation parameters, sample method, and sample parameters, we have 100 simulations. We have calculated the error rate (true proportion of time - estimated proportion of time events occur). These are given in the table below. The 95% intervals are based on percentile intervals.

Summary statistics

Summary statistics of error rate for each simulation type

Grouped by simulation method and sampling method

Simulation parameters	Sampling parameters	Proportion of time event occurs	Mean error	Lower 95% percentile
RD - 01				
3	5	0.005000000	6.666667e-03	0.003
3	50	0.005000000	8.236111e-02	0.078
3	500	0.005000000	6.837500e-01	0.495
30	5	0.050000000	6.805556e-03	0.004
30	50	0.050000000	7.736111e-02	0.047
30	500	0.050000000	6.587500e-01	0.450
300	5	0.500000000	6.652778e-03	0.003
300	50	0.500000000	8.208333e-02	0.069
300	500	0.500000000	4.262500e-01	0.375
RD - PP				
3	5	0.005000000	-1.778092e-19	-0.002
3	50	0.005000000	1.111111e-03	-0.005
3	500	0.005000000	2.500000e-03	-0.005
30	5	0.050000000	0.000000e+00	0.000
30	50	0.050000000	-4.722222e-03	-0.036
30	500	0.050000000	-1.500000e-02	-0.050
300	5	0.500000000	0.000000e+00	0.000
300	50	0.500000000	0.000000e+00	0.000
300	500	0.500000000	-7.500000e-02	-0.250
RF - 01				
3	5	0.333333333	6.666667e-01	0.666
3	50	0.333333333	6.666667e-01	0.666
3	500	0.333333333	6.666667e-01	0.666
30	5	0.033333333	1.333333e-01	0.133
30	50	0.033333333	9.666667e-01	0.966
30	500	0.033333333	9.666667e-01	0.966
300	5	0.003333333	1.333333e-02	0.013
300	50	0.003333333	1.633333e-01	0.163
300	500	0.003333333	9.016667e-01	0.871
RF - PP				
3	5	0.333333333	1.916667e-03	-0.038
3	50	0.333333333	6.944444e-03	-0.118

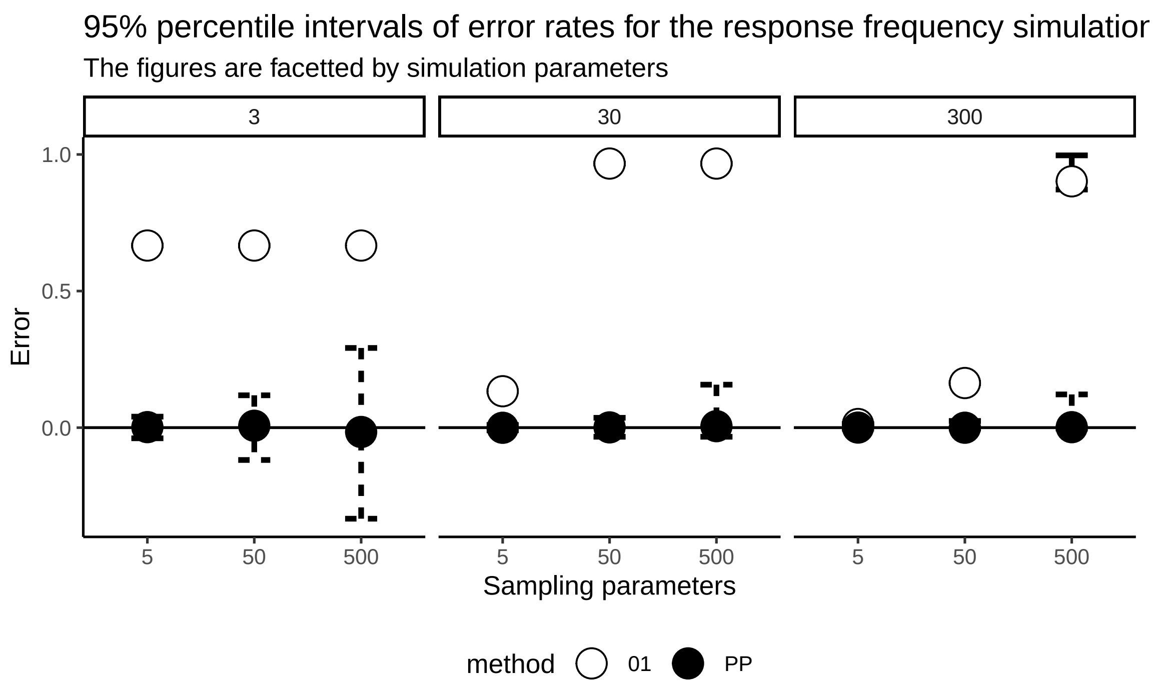
3	500	0.333333333	-1.583333e-02	-0.333
30	5	0.033333333	-4.166667e-04	-0.009
30	50	0.033333333	9.722222e-04	-0.033
30	500	0.033333333	5.416667e-03	-0.033
300	5	0.003333333	2.916667e-04	-0.003
300	50	0.003333333	-2.777778e-04	-0.003
300	500	0.003333333	1.666667e-03	-0.003

RD: Response duration

Notes, easiest way to get table out, is to save as RTF, and then paste into word.

```r
results_tab %>%
  gtsave("tabs/results.rtf")
```

Response frequency simulation

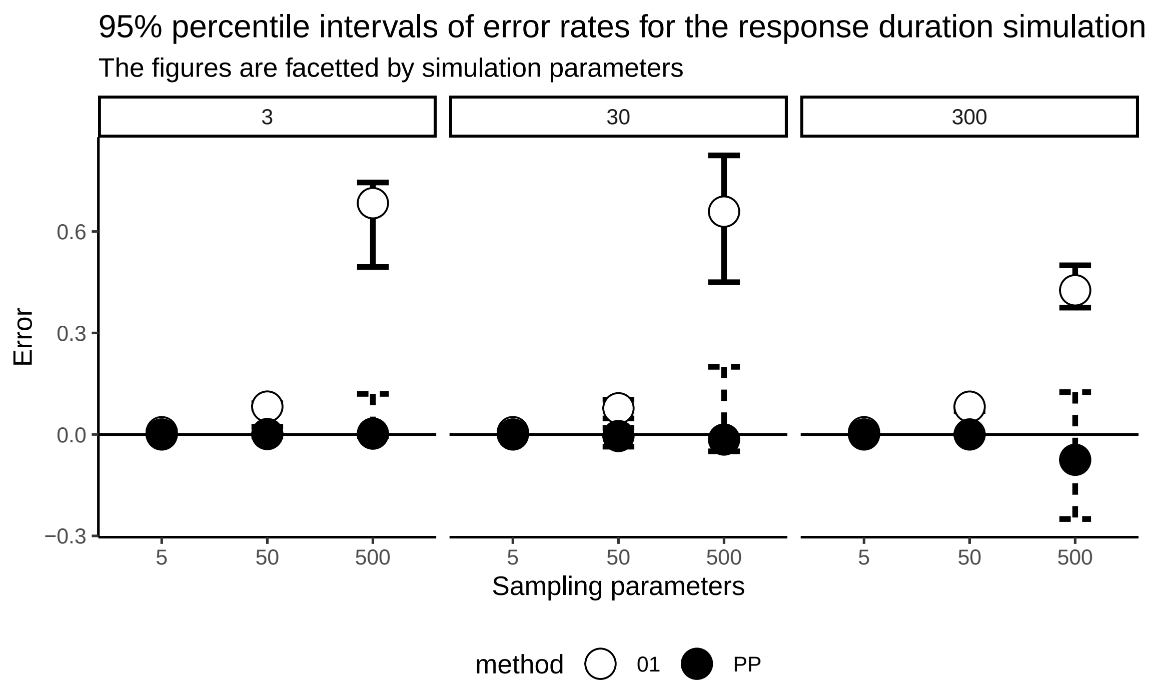


```r
ggsave("figs/RF_plot.jpg", width = 10, height = 6)
```

Notice that the zero-one sampling is biased in nearly all cases, over-estimating the true percent of the time that the event occurs. This overestimation gets worse as the sampling parameter increases.

With pinpoint sampling, we find that the method is unbiased, but the variability of the estimate increases as the interval between samples increases.

Response duration simulation



```
ggsave("figs/RD_plot.jpg", width = 10, height = 6)
```

Again we see that one-zero sampling is a poor estimator, not quite as consistently bad as with response frequency, but still pretty bad. Again, we see the over-estimation increases as the duration of the one-zero sampling increases.

Test for difference between sampling methods

To do this, we use a Friedman test to deal with the non-normality of the errors. I have treated the simulation and sampling parameter values as blocks.

Response duration

```
tar_read(test_results_RD)
```

```
## # A tibble: 1 x 6
##   statistic    df         p effsize method    magnitude
##       <dbl> <dbl>     <dbl>   <dbl> <chr>     <ord>
## 1       900     1 9.81e-198       1 Kendall W large
```

We see that there is a statistically significant difference between the sampling methods at the 5% level for the response duration method.

Response frequency

```
tar_read(test_results_RF)
```

```
## # A tibble: 1 x 6
##   statistic    df         p effsize method    magnitude
##       <dbl> <dbl>     <dbl>   <dbl> <chr>     <ord>
## 1       900     1 9.81e-198       1 Kendall W large
```

Post-hoc

```
post_hoc_tab <- tar_read(post_hoc_tab) %>%
  select(
    `Simulation method` = methods,
    `Simulation parameters` = params,
    `Sampling parameters` = delta,
    `P-value` = p
  ) %>%
  group_by(`Simulation method`) %>%
  pivot_wider(names_from = `Sampling parameters`,values_from = `P-value`) %>%
  gt() %>%
  tab_spanner(
    label = "Sampling parameters",
    columns = vars(`5`, `50`, `500`)
  ) %>%
  tab_header(
    title = "P-value for comparision of pinpoint sampling to one-zero sampling for each simulation and s
    subtitle = "P-values adjusted by FDR method"
    )
```

```
## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead
```

```
post_hoc_tab
```

P-value for comparision of pinpoint sampling to one-zero sampling for each simulation and sampling method
P-values adjusted by FDR method

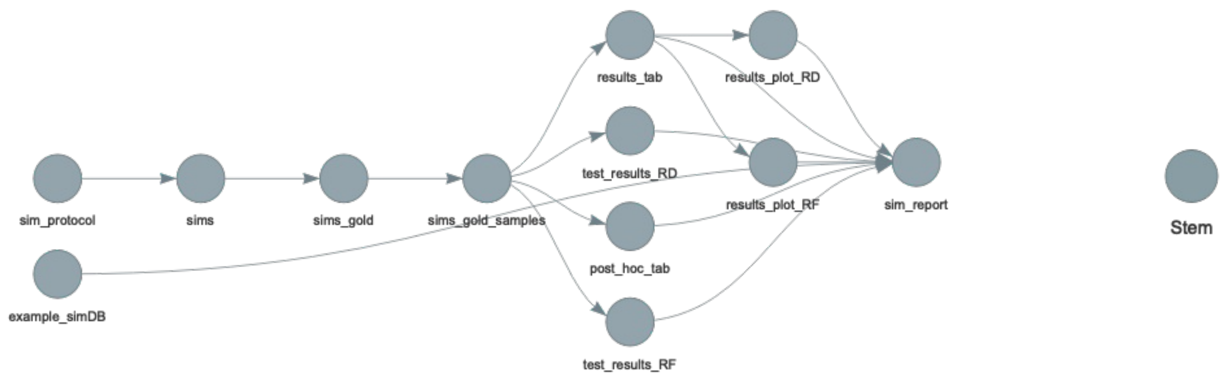
	Sampling parameters		
Simulation parameters	5	50	500
RD			
3	3.91e-18	3.83e-20	5.01e-19
30	2.11e-18	1.13e-18	6.29e-19
300	2.71e-18	1.31e-21	1.03e-18
RF			
3	3.91e-18	3.69e-18	2.46e-18
30	3.76e-18	2.90e-18	6.97e-20
300	3.54e-18	7.48e-21	7.11e-20

```
post_hoc_tab %>%
  gtsave("tabs/post_hoc.rtf")
```

Appendix

Analysis

```
targets::tar_glimpse()
```



Code

```r
code <- dir("R/", full.names = TRUE)
code
```

```
##  [1] "R//add_gold.R"               "R//add_sampling.R"
##  [3] "R//check_simDB.R"            "R//create_sampleDB.R"
##  [5] "R//create_sim_protocol.R"    "R//create_simDB.R"
##  [7] "R//fit_lm.R"                 "R//get_post_hoc.R"
##  [9] "R//get_prop.R"               "R//get_results_plot.R"
## [11] "R//get_results_tab.R"        "R//measure_stats.R"
## [13] "R//one_zero_sample.R"        "R//perform_test.R"
## [15] "R//pinpoint_sample.R"        "R//plot_sampling.R"
## [17] "R//plot_simDB.R"             "R//run_sims.R"
## [19] "R//sample_simDB.R"           "R//sim_response_duration.R"
## [21] "R//sim_response_freq.R"
```

```r
#' add_gold
#'
#' @param sims
#'
#' @return
#' @export
#'
#' @examples
add_gold <- function(sims) {
  sims %>% mutate(
    true_p = map(data, get_prop)
  )
}
# source("R/get_prop.R")
# pacman::p_load(tidyverse, targets)
# tar_load(sims)
# add_gold(sims) %>% unnest(p)
```

```r
#' add_sampling
#'
#' @param sims
#'
#' @return
#' @export
#'
#' @examples
add_sampling <- function(simDB, delta = c(3, 30, 300)) {
  PP <- delta %>%
    map_df(~pin_point_sampling(simDB, .x))
  one_zero <- delta %>%
    map_df(~one_zero_sampling(simDB, .x))
  sampling <- bind_rows(PP, one_zero)
  sampling
}
add_sampling_nest <- function(nested_simDB, delta){
  nested_simDB %>%
    mutate(
      samples = map(data, add_sampling, delta = delta)
    )
}
# pacman::p_load(tidyverse, targets)
# source("R/pinpoint_sample.R")
# source("R/one_zero_sample.R")
# tar_load(example_simDB)
# tar_load(sims_gold)
# add_sampling(example_simDB, delta = c(6, 60, 600))
# tmp <- add_sampling_nest(sims_gold, delta = c(6, 60, 600))
#' check_simDB
#'
#' @param simDB
#'
#' @return
#' @export
#'
#' @examples
check_ID <- function(simDB) {
  simDB
  # Check starts at zero
  start <- min(simDB$start)
  # Check ends at one
  end <- max(simDB$end)
  # Check lie between one and 3600
  if(all(
    between(simDB$start, 0, 3600),
    between(simDB$end, 0, 3600)
  )){
    in_range <- "YES"
  } else {
    in_range <- "NO"
  }
  # Check no missing
  simDB <-
    simDB %>%
    mutate(
      next_start = lead(start),
      diff = end - next_start
    )
  if(any(simDB$diff > 0, na.rm = TRUE)){
    overlap <- "YES"
  } else {
    overlap <- "NO"
  }
  if(any(simDB$diff < 0, na.rm = TRUE)){
    missing <- "YES"
  } else {
    missing <- "NO"
  }
  # Check overlapping
  report <- tibble(
    ID = simDB$ID[1],
    start = start,
    end = end,
    in_range = in_range,
    overlap = overlap,
    missing = missing
  )
  return(report)
}

check_simDB <- function(simDB){
  results <- simDB %>% group_split(ID) %>%
    map_df(check_ID)
  return(results)
}
# pacman::p_load(tidyverse, targets)
# tar_load(example_simDB)
# example_simDB
# check_simDB(example_simDB)
# # Break start
# example_simDB$start[1] <- 360
# check_simDB(example_simDB)
# # Break END
# tar_load(example_simDB)
# example_simDB$end[4] <- 3240
# check_simDB(example_simDB)
# # Break range
# tar_load(example_simDB)
# example_simDB$start[1] <- -360
# example_simDB$end[8] <- 3960
# check_simDB(example_simDB)
# # Make overlap
# tar_load(example_simDB)
# example_simDB$start[2] <- 180
# example_simDB$end[6] <- 1620
```

```r
# example_simDB
# plot_simDB(example_simDB)
# check_simDB(example_simDB)
# # Make missing
# tar_load(example_simDB)
# example_simDB$start[2] <- 540
# example_simDB$end[6] <- 1260
# example_simDB
# plot_simDB(example_simDB)
# check_simDB(example_simDB)
#' create_sampleDB
#'
#' @param
#'
#' @return
#' @export
#'
#' @examples
create_sampleDB <- function() {
  sampleDB <-
    tribble(
      ~start, ~end,
      0, 1440,
      1800, 2520
    )
  return(sampleDB)
}
# pacman::p_load(tidyverse, targets)
# create_sampleDB()
#' Sim_protocol
#'
#' @param params
#' @param n_sims
#'
#' @return
#' @export
#'
#' @examples
create_sim_protocol <- function(params, n_sims) {
  crossing(methods = c("RF", "RD"), params = params, ID = 1:n_sims)
}
# pacman::p_load(tidyverse, targets)
# create_sim_protocol(params = c(3, 30, 300), n_sims = 2)
#' create_simDB
#'
#' @param canvas_raw
#'
#' @return
#' @export
#'
#' @examples
create_simDB <- function() {
  simDB <- tibble(
    time = 1:3600,
    type = "background"
  )
  simDB$type[360:1800] <- "event"
  simDB$type[2160:3600] <- "event"
  return(simDB)
}
# pacman::p_load(tidyverse, targets)
# create_simDB()

#' fit_LM
#'
#' @param sims_gold_samples
#'
#' @return
#' @export
#'
#' @examples
fit_LM <- function(sims_gold_samples, sim_method) {
    df <- sims_gold_samples %>%
      unnest(c(true_p, samples)) %>%
    mutate(
      error = p - true_p,
      params = factor(params),
      delta = factor(delta)
    ) %>%
    filter(methods == sim_method)
  print(df)
  sims_lm <- lm(error ~ params * method * delta, data = df)
  return(sims_lm)
}
# pacman::p_load(tidyverse, targets, gglm)
# tar_load(sims_gold_samples)
# sims_lm <- fit_LM(sims_gold_samples, sim_method = "RF")
# sims_lm
# gglm(sims_lm)
#' Get_post_hoc
#'
#' @param sims_gold_samples
#'
#' @return
#' @export
#'
#' @examples
get_post_hoc <- function(sims_gold_samples) {
  sims_gold_samples %>%
    unnest(c(true_p, samples)) %>%
    mutate(
      error = p - true_p
    ) %>%
    group_by(methods, params, delta) %>%
    wilcox_test(error ~ method, paired = TRUE, p.adjust.method = 'fdr')
}
```

```r
# pacman::p_load(tidyverse, targets)
# tar_load(sims_gold_samples)
# get_post_hoc(sims_gold_samples)
#' get_percent
#'
#' @param simDB
#'
#' @return
#' @export
#'
#' @examples
get_prop <- function(simDB) {
  simDB %>%
    count(type) %>%
    mutate(N = sum(n)) %>%
    mutate(p = n / N) %>%
    filter(type == "event") %>%
    pull(p)
}
# pacman::p_load(tidyverse, targets)
# tar_load(example_simDB)
# get_percent(example_simDB)

#' get_results_plot
#'
#' @param results_tab
#'
#' @return
#' @export
#'
#' @examples
get_results_plot <- function(results_tab, sim_method) {
  results_tab %>%
    filter(methods == sim_method) %>%
    ggplot(aes(factor(delta), m, fill = method)) +
    geom_errorbar(aes(ymin = lwr, ymax = upr, linetype = method), width = 0.3, size = 1, show.legend =
    geom_point(pch = 21, size = 5) +
    facet_wrap(~params) +
    scale_fill_manual(values = c("white", "black")) +
    labs(y = "Error",
         x = "Sampling parameters",
         col = "Sampling method",
         subtitle = "The figures are facetted by simulation parameters"
    ) +
    theme_classic(base_size = 10) +
    geom_hline(yintercept = 0) +
    theme(legend.position = "bottom")
}
# pacman::p_load(tidyverse, targets)
# tar_load(results_tab)
# results_tab
# get_results_plot(results_tab, "RD")
# get_results_plot(results_tab, "RF")
```

```r
#' get_results_tab
#'
#' @param sims_gold_samples
#'
#' @return
#' @export
#'
#' @examples
get_results_tab <- function(sims_gold_samples) {
  sims_gold_samples %>%
    unnest(c(true_p, samples)) %>%
    mutate(
      error = p - true_p
    ) %>%
    group_by(
      methods, params, method, delta, true_p
    ) %>%
    summarise(
      m = mean(error),
      lwr = quantile(error, 0.025),
      upr = quantile(error, 0.975),
      .groups = "keep"
    )
}
# pacman::p_load(tidyverse, targets)
# tar_load(sims_gold_samples)
# get_results_tab(sims_gold_samples)
#' measure_stats
#'
#' @param simDB
#'
#' @return
#' @export
#'
#' @examples
measure_stats <- function(simDB) {
  simDB %>%
    count(type) %>%
    mutate(
      p = n / sum(n)
    )
}
# pacman::p_load(tidyverse, targets)
# tar_load(example_simDB)
# measure_stats(example_simDB)
#' one_zero sampling
#'
#' @param simDB
#' @param start_pt
#' @param interval
#' @param duration
#'
#' @return sample_DB
#' @export
#'
#' @examples
one_zero_sampling <- function(simDB, delta) {
  n <- nrow(simDB)
  k <- ceiling(n / delta)
  blocks <- rep(1:k, each = delta)
  blocks <- blocks[1:n]
  simDB <- simDB %>%
    add_column(
      block = blocks
    )
  sample <- simDB %>%
    group_by(block) %>%
    summarise(
      y = ifelse(any(type == "event"), 1, 0),
      .groups = "keep"
    )
  p <- mean(sample$y)
  return(
    tibble(
      method = "01", delta = delta, p = p
    )
  )
}
# pacman::p_load(tidyverse, targets)
# tar_load(example_simDB)
# example_simDB <- example_simDB %>% filter(time < 3600)
# one_zero_sampling(example_simDB, 3)

#' Friedman_test
#'
#' @param tar_load(sims_gold_samples)
#'
#' @return
#' @export
#'
#' @examples
perform_test <- function(sims_gold_samples, method = "RD") {
  block_data <- sims_gold_samples %>%
    filter(methods == method) %>%
    unnest(c(true_p, samples)) %>%
    mutate(
      error = p - true_p
    ) %>%
    unite(
      col = "block",
      c(methods, params, ID, delta)
    )
  results1 <- block_data %>%
    friedman_test(error ~ method | block) %>%
    select(statistic, df, p)
  results2 <- block_data %>%
    friedman_effsize(error ~ method | block) %>%
    select(effsize, method, magnitude)
  bind_cols(results1, results2)
}
# pacman::p_load(tidyverse, targets, rstatix)
# tar_load(sims_gold_samples)
# perform_test(sims_gold_samples, "RD")
# perform_test(sims_gold_samples, "RF")
#' pin_point_sampling
#'
#' @param interval
#'
#' @return points
#' @export
#'
#' @examples
pin_point_sampling <- function(simDB, delta) {
  n <- nrow(simDB)
  k <- floor((n - 1) / delta)
  i <- seq(1, 3600, delta)
  sample <- simDB[i, ]
  p <- sum(sample$type == "event") / length(sample$type)
  return(
    tibble(
      method = "PP", delta = delta, p = p
    )
  )
}
# pacman::p_load(tidyverse, targets)
# tar_load(example_simDB)
# example_simDB
# pin_point_sampling(example_simDB, 3)
#' plot_sampling
#'
#' @param simDB original simulation
#' @param sampling sampled data
#'
#' @return
#' @export
#'
#' @examples
plot_sampling <- function(simDB, sampling, points) {
  sampleDB <- tibble(time = points)
  simDB %>%
    filter(type == "event") %>%
    ggplot(aes(time, "gold standard")) +
    geom_point(shape = "square") +
    geom_point(aes(x = time, "sample points"), data = sampleDB) +
    geom_point(aes(y = "sample"), data = sampling %>% filter(type == "event")) +
    scale_x_continuous(breaks = seq(0, 3600, 300)) +
    labs(y = NULL)
}
# pacman::p_load(tidyverse, targets)
# tar_load(example_simDB)
# pts <- pin_point_sampling(300)
# example_sample <- sample_simDB(simDB = example_simDB, pts = pts)
# plot_sampling(example_simDB, example_sample, pts)
#' plot_sim
#'
#' @param sim_df
#'
#' @return ggplot
#' @export
#'
#' @examples
plot_simDB <- function(sim_df) {
  sim_df %>%
    filter(type == "event") %>%
    ggplot(aes(time, "event")) + geom_point(shape = "square") +
    scale_x_continuous(breaks = seq(0, 3600, 300)) +
    labs(y = NULL)
}

# pacman::p_load(tidyverse, targets)
# tar_load(example_simDB)
# plot_simDB(example_simDB)
run_sim <- function(methods, params){
  if(methods == "RD"){
    df <- sim_response_duration(duration = params)
  } else {
    df <- sim_response_freq(interval = params)
  }
  return(df)
}
# run_sim("RD", 3)
#' run_sims
#'
#' @param sim_protocol
#'
#' @return
#' @export
#'
#' @examples
run_sims <- function(sim_protocol) {
  sims <- sim_protocol %>%
    rowwise(methods, params, ID) %>%
    summarise(run_sim(methods, params)) %>%
    group_by(methods, params, ID) %>%
    nest()
  sims
}
# pacman::p_load(tidyverse, targets)
# tar_load(sim_protocol)
# run_sims(sim_protocol)
# sims
# plot_simDB(sims$data[[1]])
# plot_simDB(sims$data[[7]])

sample_simDB <- function(simDB, pts){
  simDB <- simDB %>%
    filter(time %in% pts)
  return(simDB)
}
# pacman::p_load(tidyverse, targets)
# tar_load(example_simDB)
# example_simDB
# sample_simDB(example_simDB, 1700:2000)
#' simulate response duration
#'
#' Splits time into blocks and then choose one of the seconds in that block for the behaviour.
#'
#' @param interval rate of event, i.e., every 3 seconds set interval to 3.
#'
#' @return simDB
#' @export
#'
#' @examples
sim_response_duration <- function(duration){
  starts <- seq(1, 3600, 600)
  ends <- seq(600, 3600, 600)  - duration
  df <- tibble(
    time = 1:3600,
    type = "background"
  )
  for(i in 1:length(starts)){
    from <- sample(starts[i]:ends[i], size = 1)
    to <- from + duration - 1
    df$type[from:to] <- "event"
  }
  return(df)
}

# pacman::p_load(tidyverse)
# source("R/plot_simDB.R")
# plot_simDB(sim_response_duration(300)) +
#   geom_vline(xintercept = seq(1, 3600, 600))

#' simulate response frequency
#'
#' Splits time into blocks and then choose one of the seconds in that block for the behaviour.
#'
#' @param interval rate of event, i.e., every 3 seconds set interval to 3.
#'
#' @return simDB
#' @export
#'
#' @examples
sim_response_freq <- function(interval){
  df <- tibble(
    time = 1:3600,
    type = "background"
  )
  ## Add blocks
  blocks <- rep(1:1200, each = interval)
  blocks <- blocks[1:3600]
  df <- df %>%
    add_column(
      block = blocks
    )
  i <- df %>%
    group_by(block) %>%
    sample_n(1) %>%
    pull(time)
  df$type[i] <- "event"
  df <-
    df %>% select(-block)
  return(df)
}

# plot_simDB(sim_response_freq(300))
```

Notes

Folder

```r
here::here()
```

```
## [1] "/Users/jonathantuke/Dropbox/Research/2020 animal_simulation"
```