Power Analysis with Iteration

Introduction

This document demonstrates how to use iteration with `purrr::map()` to run power analyses across multiple effect sizes. We'll use rodent population data and the `simr` package to assess statistical power for detecting trends over time.

Load Required Packages

```r
library(dplyr)
library(readr)
library(purrr)
library(lme4)
library(simr)
```

Prepare the Data

We'll analyze counts of Merriam's Kangaroo Rat (*Dipodomys merriami*, species code "DM") from the Portal Project, grouping by year and plot.

```r
rodents <- read_csv(here::here("PortalData/Rodents/Portal_rodent.csv"))

# Our z-score function from before
to_z <- function(x) {
  (x - mean(x, na.rm = TRUE)) / sd(x, na.rm = TRUE)
}

rodent_counts <- rodents |>
  filter(species == "DM") |>
  group_by(year, plot) |>
  summarise(n = n()) |>
  ungroup() |>
  mutate(
    plot = factor(plot),
    nyear = to_z(year)
  )

head(rodent_counts)
```

```
# A tibble: 6 × 4
   year plot      n nyear
  <dbl> <fct> <int> <dbl>
```

```
1  1977 1         9 -1.55
2  1977 2        17 -1.55
3  1977 3         6 -1.55
4  1977 4         7 -1.55
5  1977 5        10 -1.55
6  1977 6        15 -1.55
```

Fit a Mixed Model

We fit a negative binomial generalized linear mixed model with plot as a random effect.

```
rodent_mod <- glmer.nb(
  n ~ nyear + (1 | plot),
  data = rodent_counts
)

summary(rodent_mod)
```

```
Generalized linear mixed model fit by maximum likelihood (Laplace
  Approximation) [glmerMod]
 Family: Negative Binomial(1.6163)  ( log )
Formula: n ~ nyear + (1 | plot)
   Data: rodent_counts

      AIC       BIC    logLik -2*log(L)  df.resid
   6308.8    6327.4   -3150.4    6300.8       783

Scaled residuals:
    Min      1Q  Median      3Q     Max
-1.2088 -0.7461 -0.1650  0.5049  4.7279

Random effects:
 Groups Name        Variance Std.Dev.
 plot   (Intercept) 0.6152   0.7844
Number of obs: 787, groups:  plot, 24

Fixed effects:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  2.74168    0.16406  16.712   <2e-16 ***
nyear       -0.01685    0.03082  -0.547    0.585
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Correlation of Fixed Effects:
      (Intr)
nyear -0.004
```

Create a Power Analysis Function

This function automates the power analysis workflow:

1. Sets a specified effect size for the year coefficient
2. Extends the dataset to simulate additional time points
3. Runs power simulations
4. Returns results as a data frame

```r
run_pa <- function(
  model,
  fxsize,
  yearcol = "nyear",
  extend_n = 10,
  nsim = 500,
  alpha = 0.2
) {
  library(simr)
  fixef(model)[yearcol] <- (log(fxsize))
  model_extended <- extend(model, along = yearcol, n = extend_n)
  result <- model_extended |>
    powerSim(
      nsim = nsim,
      alpha = alpha
    ) |>
    summary() |>
    as.data.frame()

  dplyr::mutate(result, fxsize = fxsize, Design = "Full dataset")
}
```

Run Power Analysis Across Multiple Effect Sizes

Using `map()`, we can iterate over a sequence of effect sizes and run the power analysis for each one. We'll start with a small example using just 10 simulations per effect size.

```r
fx_sizes <- seq(1.01, 1.05, by = 0.01)

res_list <- map(
  fx_sizes,
  \(x) run_pa(rodent_mod, x, nsim = 10)
)

power_results <- list_rbind(res_list)
power_results
```

```
  successes trials mean     lower     upper fxsize       Design
1         0     10  0.0 0.0000000 0.3084971   1.01 Full dataset
2         4     10  0.4 0.1215523 0.7376219   1.02 Full dataset
3         4     10  0.4 0.1215523 0.7376219   1.03 Full dataset
4        10     10  1.0 0.6915029 1.0000000   1.04 Full dataset
5        10     10  1.0 0.6915029 1.0000000   1.05 Full dataset
```

Advanced: Running in Parallel (Optional)

The following section demonstrates how to speed up power analyses using parallel processing. The parallel approach can significantly reduce computation time when running many simulations.

Comparing Sequential vs Parallel Execution

We'll compare the performance of sequential execution against parallel execution with 100 simulations across 10 effect sizes.

Sequential Execution

```r
fx_sizes <- seq(1.01, 1.10, by = 0.01)

tictoc::tic("Sequential")
res_list <- map(
  fx_sizes,
  \(x) run_pa(rodent_mod, x, nsim = 100)
)
tictoc::toc()
```

```
Sequential: 44.925 sec elapsed
```

```r
list_rbind(res_list)
```

```
   successes trials mean     lower     upper fxsize       Design
1         23    100 0.23 0.1517316 0.3248587   1.01 Full dataset
2         35    100 0.35 0.2572938 0.4518494   1.02 Full dataset
3         55    100 0.55 0.4472802 0.6496798   1.03 Full dataset
4         83    100 0.83 0.7418246 0.8977351   1.04 Full dataset
5         83    100 0.83 0.7418246 0.8977351   1.05 Full dataset
6         98    100 0.98 0.9296161 0.9975687   1.06 Full dataset
7         96    100 0.96 0.9007428 0.9889955   1.07 Full dataset
8         99    100 0.99 0.9455406 0.9997469   1.08 Full dataset
9        100    100 1.00 0.9637833 1.0000000   1.09 Full dataset
10        99    100 0.99 0.9455406 0.9997469   1.10 Full dataset
```

Parallel Execution Setup

For parallel processing, we need to attach the data to the model object so it's available to the parallel workers.

```r
# This is sort of weird, we have to add the data back into the model object so
that
# when it gets sent into the parallel workers it has everything it needs
getData(rodent_mod) <- rodent_counts

parallel::detectCores()
```

```
[1] 10
```

Running with Parallel Workers

Using `purrr::in_parallel()` with the `mirai` backend, we can distribute the work across multiple CPU cores.

```r
tictoc::tic("Parallel, 10 workers")
mirai::daemons(10)

res_list <- map(
  fx_sizes,
  in_parallel(
    \(x) run_pa(model, x, nsim = 100),
    # need to send the objects into the workers
    run_pa = run_pa,
    model = rodent_mod
  )
)

# Clean up the parallel workers
mirai::daemons(0)
tictoc::toc()
```

```
Parallel, 10 workers: 8.928 sec elapsed
```

```r
list_rbind(res_list)
```

```
  successes trials mean     lower     upper fxsize       Design
1        32    100 0.32 0.2302199 0.4207669   1.01 Full dataset
2        34    100 0.34 0.2482235 0.4415333   1.02 Full dataset
3        54    100 0.54 0.4374116 0.6401566   1.03 Full dataset
4        74    100 0.74 0.6426879 0.8226056   1.04 Full dataset
```

```
5         91    100 0.91 0.8360177 0.9580164   1.05 Full dataset
6         94    100 0.94 0.8739701 0.9776651   1.06 Full dataset
7         99    100 0.99 0.9455406 0.9997469   1.07 Full dataset
8        100    100 1.00 0.9637833 1.0000000   1.08 Full dataset
9         99    100 0.99 0.9455406 0.9997469   1.09 Full dataset
10       100    100 1.00 0.9637833 1.0000000   1.10 Full dataset
```

The parallel approach should show significant time savings, especially as the number of different effect sizes tested and the number of simulations increases.