Simulation-based power analysis for $N = 1$ intensive longitudinal designs

Ginette Lafit Jordan Revol

2023-06-08

Table of contents

Settings things up

Before we proceed, we need to ensure we have several packages installed and loaded into our `R` session. For the scripts below, we will use the following packages:

- `data.table`
- `psych`
- `tidyverse`
- `MASS`
- `stringr`

Which we can install in one go as follows:

```
# Prepare the package list.
packages = c("data.table", "psych", "tidyverse", "MASS", "stringr")

# Install packages.
```

```
install.packages(packages)
```

> **Tip**
>
> You may consider first checking if the packages are installed before actually installing them. Nevertheless, the code above will not reinstall packages that are already installed and up-to-date.

Now that we have all packages installed, we continue by loading them.

```
# To create lagged outcome variables.
library(data.table)

# To compute descriptive statistics.
library(psych)

# A useful package.
library(tidyverse)

# Handy functions for data analysis.
library(MASS)

# For handy string manipulation.
library(stringr)

# Set a seed for reproducibility.
set.seed(123)
```

Preparing the simulation functions

We need two functions to conduct a simulation-based power analysis.

- one function that generates the datasets for the simulation
- and a second function that runs the Monte Carlo simulation to estimate the *empirical power*

The data generating function

This function generates a dataset from an VAR(1) model and takes a few arguments as follows:

- `vars` is the number of variables of the VAR(1) model
- `Tobs` is the number of repeated measurements
- `delta` the intercept matrix
- `psi` the transition matrix (i.e., containing the auto-regressive and cross-regressive effects)
- `sigma` the variance-covariance matrix of the innovation

```
# Function to generate data from a `VAR(1)` model.
sim_VAR_data <- function(vars, Tobs, delta, psi, sigma) {
    # Number of burning observations.
    T.burning <- 10000

    # Define the number of observations.
    T.total <- T.burning + Tobs

    # Simulate errors.
    if (vars == 1) {
        E <- as.matrix(rnorm(T.total, 0, sigma))
    } else {
        E <- mvrnorm(T.total, mu = rep(0, vars), sigma)
    }

    # Recursive equation.
    Y <- matrix(0, T.total, vars)

    # Initialized values.
    Y[1, ] <- delta + E[1, ]

    # Simulate dependent variables.
    for (t in 2:T.total) {
        Y[t, ] <- delta + psi %*% Y[t - 1, ] + E[t, ]
    }

    # Exclude burning observations.
    Y <- Y[-seq(1:T.burning), ]

    # Create lagged variables.
    Y <- cbind(Y, lag(Y))

    # Rename variables.
    colnames(Y) <- c(
        sprintf("Y%d", seq(1:vars)),
        sprintf("Y%dlag", seq(1:vars))
```

```
    )

    # Return the data.
    return(as.data.frame(Y))
}
```

The Monte Carlo simulation function

This function conducts the Monte Carlo simulation for a set of sample sizes (i.e., several different number of observations) and computes the statistical power for a given hypothesis. It takes several arguments as follows:

- `vars` is the number of variables of the VAR(1) model
- `Tobs_list` is a list of numbers of repeated measurements (i.e., `Tobs`)
- `delta` the intercept matrix
- `psi` the transition matrix (which contains the auto-regressive and cross-regressive effects)
- `sigma` the variance-covariance matrix of the innovation
- `R` is the number of Monte Carlo replicates (e.g., 1000)
- `alpha` is the Type I error rate (or significance level of a test statistic)

```
# Function to conduct the Monte Carlo power simulation.
mc_power <- function(vars, Tobs_list, delta, psi, sigma, R, alpha) {
    # Prepare simulation storage.
    df_pow <- data.frame()

    # For each sample size in the list.
    for (i in 1:length(Tobs_list)) {
        # Extract the sample size.
        Tobs <- Tobs_list[i]

        # Print the progress.
        print(paste0("Power analysis for N = ", Tobs))

        # For each Monte Carlo replication.
        for (r in 1:R) {
            # Generate data.
            data <- sim_VAR_data(vars, Tobs, delta, psi, sigma)

            # Create names lists.
            var_names <- sprintf("Y%d", seq(1:vars))
            lag_names <- sprintf("Y%dlag", seq(1:vars))
```

```
        # Prepare storage for the model fits.
        list_pow_rep <- list()

        # For each variable in the model.
        for (nbName in 1:length(var_names)) {
            # Extract variable name.
            name_ <- var_names[nbName]

            # Create the formula for the linear regression.
            formula <- as.formula(
                paste(name_, paste(lag_names, collapse = " + "), sep = " ~ ")
            )

            # Estimate model.
            model <- lm(formula, data)

            # Extract the coefficients.
            coefficients <- summary(model)$coefficients

            # Extract the `p` values.
            df_pval <- as.data.frame(rbind(coefficients[, 4]))

            # Compute the empirical power.
            list_pow_rep[[nbName]] <- as.data.frame(df_pval < alpha)

            # Add names to the columns.
            names(list_pow_rep[[nbName]]) <- c(
                paste0("pow_int_", name_),
                paste0("pow_", lag_names, "_", name_)
            )
        }

        # Bind the columns.
        rep_data <- bind_cols(
            data.frame(Tobs = Tobs), do.call(cbind, list_pow_rep)
        )

        # Bind the rows.
        df_pow <- rbind(df_pow, rep_data)
    }
}
```

```r
    # Compute empirical power.
    df_pow <- aggregate(
        df_pow[, 2:ncol(df_pow)], list(df_pow$Tobs), mean
    )

    # Rename the columns.
    df_pow <- rename(df_pow, Tobs = Group.1)

    return(df_pow)
}
```

Conducting the simulation power analysis

To conduct the simulation-based power analysis you first need to set the following arguments:

- `Tobs`: list of numbers of repeated measurements.
- `vars`: number of variables of the VAR(1) model.
- `delta`: a vars $\times$ 1 matrix with one intercept per variable.
- `psi`: a vars $\times$ vars matrix. The diagonal elements are the autoregressive coefficients, and the off-diagonal are the cross-regressive coefficients.
- `sigma`: a vars $\times$ vars matrix. The diagonal elements are the variance of the residuals, and the off-diagonal elements are the covariance values. Note that the matrix should be symmetric.
- `alpha`: type I error rate (i.e., conventionally set at .05).
- `R`: the number of Monte Carlo replicates (i.e., 1000).
- `pow_target`: the empirical power targeted, which is often .8. This variable is only used in the results (not in the simulation).

For the reminder of this document, we demonstrate how to run a simulation-based power analysis for the AR(1) and VAR(1) models.

Simulation for the AR(1) model

We ran a simulation for an AR(1) model with Tobs $= \{50, 100, 150\}$, specified as follows:

$$y_t = 3 + .3 * y_{t-1} + \varepsilon$$

with:

$$\varepsilon \sim N(0, 10)$$

> **!** Important
>
> In the example below we set the number of Monte Carlo replicates to `R = 10` to speed up the computation and the generation of this document. In practice, you should set **at lest** `R = 1000` to obtain reliable results. Make sure to change it accordingly when you run the code.

```
# Specify the simulation inputs.
Tobs_list <- c(50, 100, 150)
vars <- 1
delta <- as.matrix(3)
psi <- as.matrix(.3)
sigma <- as.matrix(10)
alpha <- 0.05
R <- 10
pow_target <- .8

# Run the power analysis simulation.
df_pow_result <- mc_power(vars, Tobs_list, delta, psi, sigma, R, alpha)
```

```
[1] "Power analysis for N = 50"
[1] "Power analysis for N = 100"
[1] "Power analysis for N = 150"
```

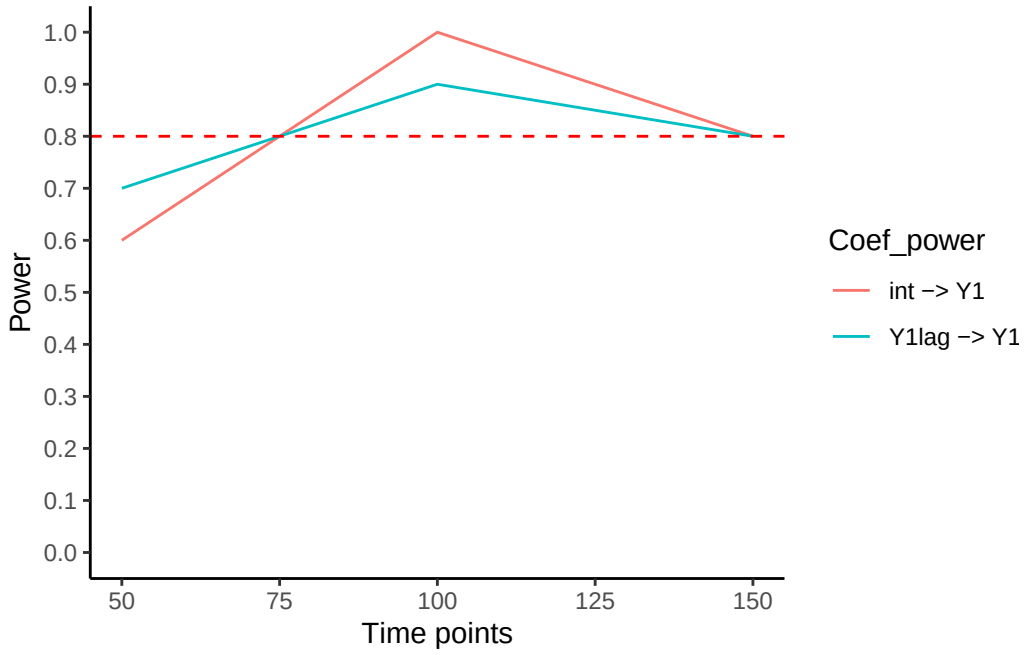
With the simulation completed, we can continue to display the results of the power analysis in a plot and a recapitulation table.

```
# Prepare the results data for the plot.
dt <- df_pow_result %>%
    gather(Coef_power, power, starts_with("pow"))

# Rename.
dt$Coef_power = stringr::str_replace_all(
    dt$Coef_power, c("pow_" = "", "_" = " -> ")
)

# Plot the results of the `AR(1)` power analysis.
ggplot(dt,
    aes(
        y = power,
        x = Tobs,
```

```
    color = Coef_power
)) +
geom_line() +
geom_hline(
    yintercept = pow_target,
    color = "red",
    linetype = 2
) +
scale_y_continuous(
    breaks = seq(0, 1, by = .1),
    limits = c(0, 1)
) +
labs(
    y = "Power",
    x = "Time points"
) +
theme_classic()
```



Finally, we can display the results in a table as shown below.

```r
# Create table of outputs.
dt = df_pow_result %>%
    gather(pow, value, starts_with("pow")) %>%
    spread(Tobs, value)

# Rename variables.
dt$pow <- stringr::str_replace_all(dt$pow, c("pow_"="", "_"=" -> "))

# Rename the columns.
names(dt)[1] <- "Coefficients"

# Print the table.
print(dt)
```

```
  Coefficients  50 100 150
1    int -> Y1 0.6 1.0 0.8
2  Y1lag -> Y1 0.7 0.9 0.8
```

Simulation for the VAR(1) model

We ran a simulation for a VAR(1) model with three variables and Tobs $= \{50, 100, 150\}$, specified as follows:

$$\begin{bmatrix} y_{1t} \\ y_{2t} \\ y_{3t} \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \\ 10 \end{bmatrix} + \begin{bmatrix} 0.5 & 0.14 & .2 \\ 0.1 & 0.4 & .03 \\ 0.05 & 0.12 & .6 \end{bmatrix} \begin{bmatrix} y_{1,t-1} \\ y_{2,t-1} \\ y_{3,t-1} \end{bmatrix} + \begin{bmatrix} \varepsilon_{1t} \\ \varepsilon_{2t} \\ \varepsilon_{3t} \end{bmatrix}$$

with:

$$\begin{bmatrix} \varepsilon_{1t} \\ \varepsilon_{2t} \\ \varepsilon_{3t} \end{bmatrix} \sim N\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 15 & 3 & 6 \\ 3 & 20 & 9 \\ 6 & 9 & 18 \end{bmatrix} \right)$$

Let's start by setting the values for the simulation. Check out the documentation for the `mc_power` function for more details on the inputs.

> **!** Important
>
> Just like before, in the example below we set the number of Monte Carlo replicates to `R = 10` to speed up the computation and the generation of this document. In practice, you should set **at lest** `R = 1000` to obtain reliable results. Make sure to change it accordingly

when you run the code.

```
# Set the values for conducting the power analysis.
Tobs_list = c(50, 100, 150)
vars = 3
delta = as.matrix(c(4, 6, 10))
psi = rbind(
    c(.50, .14, .20),
    c(.10, .40, .03),
    c(.05, .12, .50)
)
sigma = rbind(
    c(15,  3,  6),
    c( 3, 20,  9),
    c( 6,  9, 18)
)
alpha = 0.05
R = 10
pow_target = .8

# Run the power analysis simulation.
df_pow_result <- mc_power(vars, Tobs_list, delta, psi, sigma, R, alpha)
```

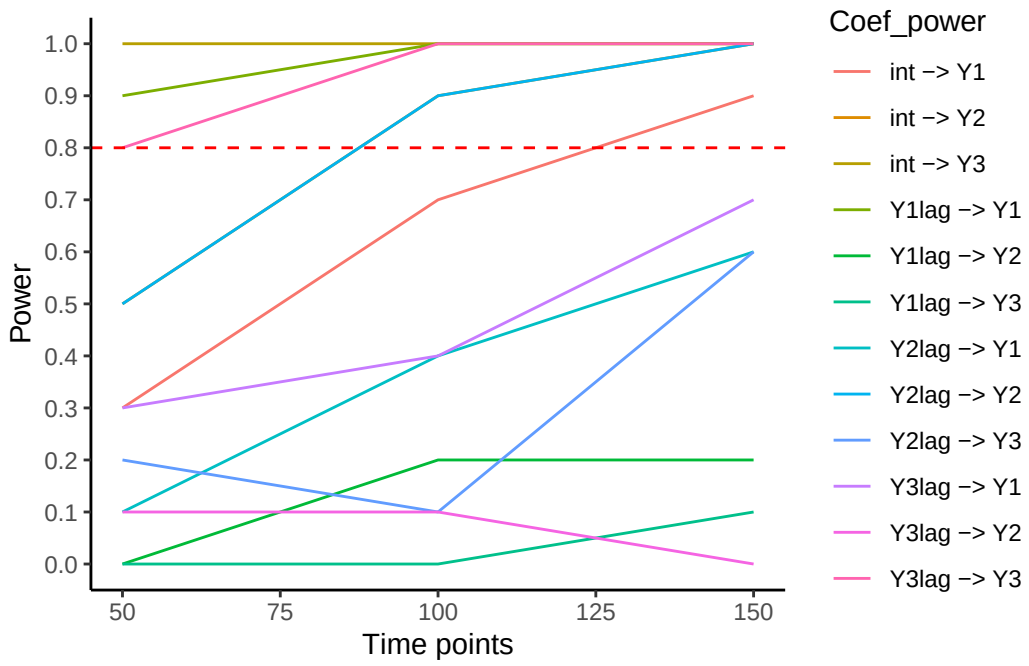
```
[1] "Power analysis for N = 50"
[1] "Power analysis for N = 100"
[1] "Power analysis for N = 150"
```

We can continue to display the results of the power analysis in a plot and then summarize them in a table.

```
# Prepare the results data for the plot.
dt = df_pow_result %>%
    gather(
        Coef_power, power, starts_with("pow")
    )

# Rename the variables.
dt$Coef_power = stringr::str_replace_all(
    dt$Coef_power, c("pow_" = "", "_" = " -> ")
)
```

```r
# Plot the results of the power analysis.
ggplot(dt,
    aes(
        y = power,
        x = Tobs,
        color = Coef_power
    )) +
    geom_line() +
    geom_hline(
        yintercept = pow_target,
        color = "red",
        linetype = 2
    ) +
    scale_y_continuous(
        breaks = seq(0, 1, by = .1),
        limits = c(0, 1)
    ) +
    labs(
        y = "Power",
        x = "Time points"
    ) +
    theme_classic()
```



```r
# Create table of outputs.
dt = df_pow_result %>%
    gather(
        pow,
        value,
        starts_with("pow")
    ) %>%
    spread(
        Tobs,
        value
    )

# Rename the variables.
dt$pow <- stringr::str_replace_all(
    dt$pow, c("pow_" = "", "_" = " -> ")
)

# Rename the columns.
names(dt)[1] <- "Coefficients"

# Print the table.
print(dt)
```

```
   Coefficients  50 100 150
1     int -> Y1 0.3 0.7 0.9
2     int -> Y2 0.5 0.9 1.0
3     int -> Y3 1.0 1.0 1.0
4   Y1lag -> Y1 0.9 1.0 1.0
5   Y1lag -> Y2 0.0 0.2 0.2
6   Y1lag -> Y3 0.0 0.0 0.1
7   Y2lag -> Y1 0.1 0.4 0.6
8   Y2lag -> Y2 0.5 0.9 1.0
9   Y2lag -> Y3 0.2 0.1 0.6
10  Y3lag -> Y1 0.3 0.4 0.7
11  Y3lag -> Y2 0.1 0.1 0.0
12  Y3lag -> Y3 0.8 1.0 1.0
```

Session information

Using the command below, we can print the `session` information (i.e., operating system, details about the `R` installation, and so on) for reproducibility purposes.

```r
# Session information.
sessionInfo()
```

```
R version 4.3.0 (2023-04-21)
Platform: aarch64-apple-darwin20 (64-bit)
Running under: macOS Ventura 13.4

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack.dylib;  L

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

time zone: Europe/Amsterdam
tzcode source: internal

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
 [1] MASS_7.3-60       lubridate_1.9.2   forcats_1.0.0     stringr_1.5.0
 [5] dplyr_1.1.2       purrr_1.0.1       readr_2.1.4       tidyr_1.3.0
 [9] tibble_3.2.1      ggplot2_3.4.2     tidyverse_2.0.0   psych_2.3.3
[13] data.table_1.14.8

loaded via a namespace (and not attached):
 [1] gtable_0.3.3     jsonlite_1.8.5   compiler_4.3.0   tidyselect_1.2.0
 [5] parallel_4.3.0   scales_1.2.1     yaml_2.3.7       fastmap_1.1.1
 [9] lattice_0.21-8   R6_2.5.1         labeling_0.4.2   generics_0.1.3
[13] knitr_1.43       munsell_0.5.0    pillar_1.9.0     tzdb_0.4.0
[17] rlang_1.1.1      utf8_1.2.3       stringi_1.7.12   xfun_0.39
[21] timechange_0.2.0 cli_3.6.1        withr_2.5.0      magrittr_2.0.3
[25] digest_0.6.31    grid_4.3.0       rstudioapi_0.14  hms_1.1.3
[29] lifecycle_1.0.3  nlme_3.1-162     vctrs_0.6.2      mnormt_2.1.1
[33] evaluate_0.21    glue_1.6.2       farver_2.1.1     fansi_1.0.4
[37] colorspace_2.1-0 rmarkdown_2.22   tools_4.3.0      pkgconfig_2.0.3
[41] htmltools_0.5.5
```