Simulation-based power analysis for the AR(1) and VAR(1) models

Ginette Lafit Jordan Revol

2023-06-02

Table of contents

Setup the environment 1
Preparing the functions 2
Function 1: Data generation for the VAR(1) model 2
Function 2: Conduct the Monte Carlo simulation 3
Conduct the simulation-based power analysis 4
Example of a AR(1) model 5
Example of a VAR(1) model with 3 variables 7
Get the session info 9

Setup the environment

The code below chunk simply makes sure that all the libraries used here are installed. We should first check if the R packages are installed before we proceed.

```
# Do not run because we do not want to install packages (this should be your decision)

list.of.packages = c("data.table","psych","ggplot2","tidyverse","MASS")
new.packages = list.of.packages[!(list.of.packages %in% installed.packages()[,"Package"])]
if(length(new.packages)) install.packages(new.packages)
```

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

```
library(data.table) # to create lagged outcome
library(psych) # to compute descriptive statistics
library(ggplot2) # for making plots
```

```r
library(tidyverse) # a useful package

library(MASS)

set.seed(1235) # Set a seed to reproduce analyses
```

Preparing the functions

We need two functions to conduct a simulation-based power analysis. The first one generates the datasets, and the second run the Monte-Carlo simulation to estimate the empirical power.

Function 1: Data generation for the VAR(1) model

This function generates a dataset from an VAR(1) model where '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 (which contains the auto- and cross-regressive effects), and 'sigma' the variance-covariance matrix of the innovation.

```r
# Function to generate data from an $\text{AR}(1)$ Model
sim_VAR_data = function(vars,Tobs,delta,psi,sigma){

  # Create number of observations: N + T.burning
  T.burning = 10000 # Number of burning 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, create lag variable and rename columns
  Y = Y[-seq(1:T.burning),]
  Y = cbind(Y,lag(Y))
  colnames(Y) = c(sprintf("Y%d",seq(1:vars)),sprintf("Y%dlag",seq(1:vars)))

  return(as.data.frame(Y))
}
```

Function 2: Conduct the Monte Carlo simulation

This function conducts the Monte Carlo simulation for a set of 'Tobs' and computed statistical power for a given hypothesis. The arguments of the function are: 'vars' is the number of variables of the VAR(1) model, 'Tobs_list' is a list of numbers of repeated measurements ('Tobs'), 'delta' the intercept matrix, 'psi' the transition matrix (which contains the auto- and cross-regressive effects), and '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).

```
# Function to conduct the Monte Carlo simulation and compute statistical power for a given

mc_power = function(vars,Tobs_list,delta,psi,sigma,R,alpha){
  # Generate dataset
  df_pow = data.frame()

  # Loop over the sample size list
  for (i in 1:length(Tobs_list)){
    Tobs = Tobs_list[i]
    print(paste0("Power analysis for N = ", Tobs))

    # R replicates for each sample size
    for (r in 1:R){

      # Generate VAR data
      data = sim_VAR_data(vars,Tobs,delta,psi,sigma)

      # Create names list
      var_names = sprintf("Y%d",seq(1:vars))
```

```
      lag_names = sprintf("Y%dlag",seq(1:vars))

      # Estimate models
      list_pow_rep = list()

      for (nbName in 1:length(var_names)){
        name_ = var_names[nbName]

        # Create formula
        formula = as.formula(paste(name_ ,paste(lag_names, collapse = " + "), sep = " ~ ")

        # Estimate model
        model = lm(formula, data)

        # Extract coefs
        sum = summary(model)$coefficients

        # Extract p.values
        df_pval = as.data.frame(rbind(sum[,4]))

        # Compute power
        list_pow_rep[[nbName]] =  as.data.frame(df_pval < alpha)
        names(list_pow_rep[[nbName]]) = c(paste0("pow_int_",name_), paste0("pow_",lag_name
      }

      rep_data = bind_cols(data.frame(Tobs = Tobs), do.call(cbind, list_pow_rep))
      df_pow = rbind(df_pow, rep_data)
    }
  }

  # Compute power
  df_pow = aggregate(df_pow[, 2:ncol(df_pow)], list(df_pow$Tobs), mean)
  df_pow = rename (df_pow, Tobs = Group.1)

  return(df_pow)
}
```

Conduct the simulation-based 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*1 matrix with one intercept per variable.
- 'psi': a vars*vars matrix. The diagonal elements are the autoregressive coefficients, and the off-diagonal are the cross-regressive coefficients.
- 'sigma': a var*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 (often .05).
- 'R': number of Monte Carlo replicates (often 1000).
- 'pow_target': the empirical power targeted, which is often .8. This variable is only used in the results (not in the simulation).

We demonstrate how to run this analysis with two examples: an AR(1) and a VAR(1) model.

Example of a AR(1) model

We ran a AR(1) model with Tobs = 50, 100 and 150 and for a AR(1) model as follows:

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

with:

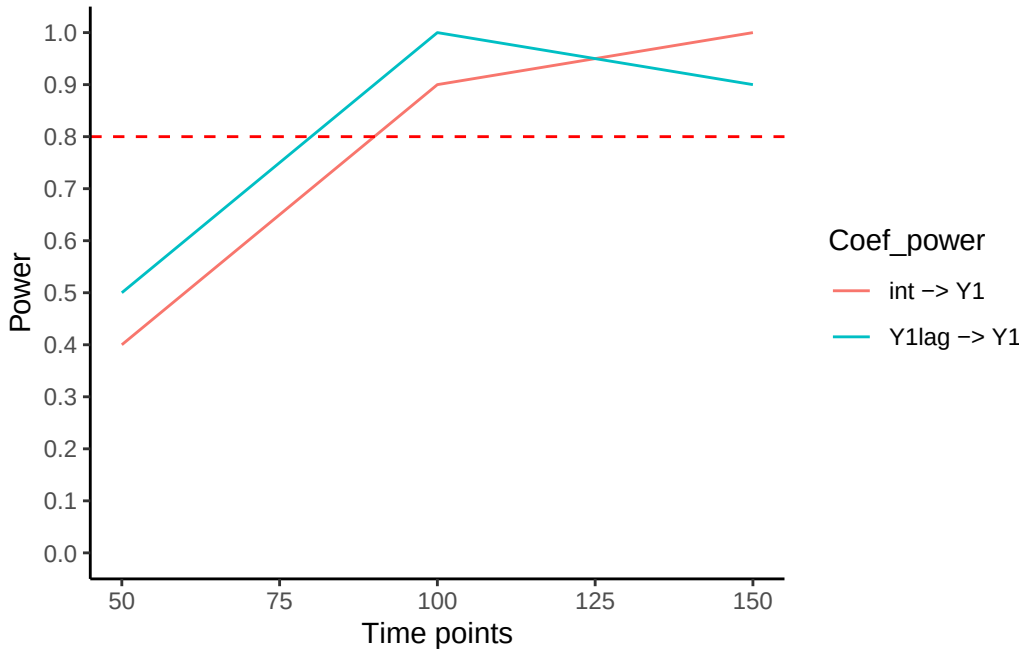
$$\varepsilon \sim N(0, 10)$$

```
# Set the values for conducting the power analysis
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

# Conduct the power analysis
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"
```

Finally, we display the results of the power analysis in a plot and a recap table:

```
# Plot the results of the power analysis
dt = df_pow_result %>%
        gather(Coef_power, power, starts_with("pow"))
dt$Coef_power = stringr::str_replace_all(dt$Coef_power, c("pow_"="", "_"=" -> ")) # rename
ggplot() +
        geom_line(data=dt, aes(y=power,x=Tobs, color=Coef_power)) +
        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()
```




```
# Create table of outputs
dt = df_pow_result %>%
        gather(pow,value,starts_with("pow")) %>%
        spread(Tobs, value)
dt$pow = stringr::str_replace_all(dt$pow, c("pow_"="", "_"=" -> ")) # rename
```

```
names(dt)[1] = "Coefficients"
dt
```

```
  Coefficients  50 100 150
1    int -> Y1 0.4 0.9 1.0
2  Y1lag -> Y1 0.5 1.0 0.9
```

Example of a VAR(1) model with 3 variables

We run a power analysis with Tobs = 50 and 100 and for a VAR(1) model with 3 variables 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:

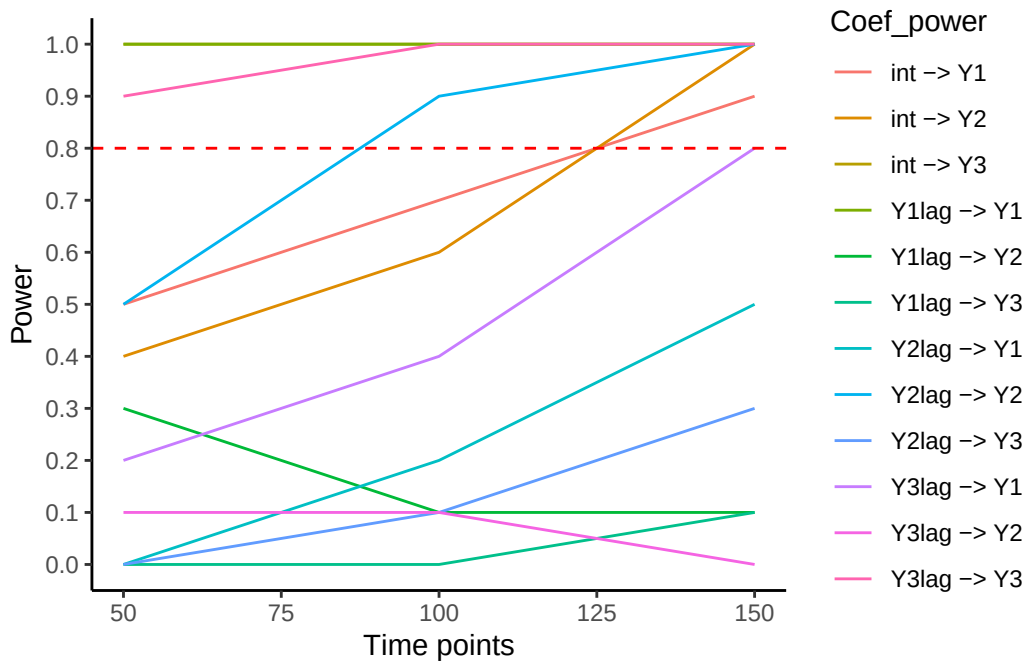
$$\begin{bmatrix} \varepsilon_{1t} \\ \varepsilon_{2t} \\ \varepsilon_{3t} \end{bmatrix} \sim N\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 15 & 3 & 6 \\ 3 & 20 & 9 \\ 6 & 9 & 18 \end{bmatrix} \right)$$

```
# Set the values for conducting the power analysis
Tobs_list = c(50, 100, 150)
vars = 3
delta = as.matrix(c(4,6,10))
psi = as.matrix(rbind(c(.5, .14, .2),
                      c(.1, .4, .03),
                      c(.05, .12, .5)))
sigma = as.matrix(rbind(c(15, 3, 6),
                        c(3, 20, 9),
                        c(6, 9, 18)))
alpha = 0.05
R = 10

pow_target = .8

# Conduct the power analysis
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"
```

Finally, we display the results of the power analysis in a plot and a recap table:

```
# Plot the results of the power analysis
dt = df_pow_result %>%
        gather(Coef_power, power, starts_with("pow"))
dt$Coef_power = stringr::str_replace_all(dt$Coef_power, c("pow_"="", "_"=" -> ")) # rename
ggplot() +
        geom_line(data=dt, aes(y=power,x=Tobs, color=Coef_power)) +
        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()
```




```
# Create table of outputs
dt = df_pow_result %>%
        gather(pow,value,starts_with("pow")) %>%
        spread(Tobs, value)
dt$pow = stringr::str_replace_all(dt$pow, c("pow_"="", "_"=" -> ")) # rename
```

```
  names(dt)[1] = "Coefficients"
  dt
```

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

Get the session info

Below we provide the `session` information (i.e., operating system, details about the `R` installation, and so on) for reproducibility purposes.

```
  sessionInfo()
```

```
R version 4.3.0 (2023-04-21)
Platform: aarch64-apple-darwin20 (64-bit)
Running under: macOS Ventura 13.3.1

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_7.3-58.4     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      tidyverse_2.0.0   ggplot2_3.4.2     psych_2.3.3
[13] data.table_1.14.8

loaded via a namespace (and not attached):
 [1] gtable_0.3.3     jsonlite_1.8.4   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
```