Skewness and staging: Does the floor effect induce bias in multilevel AR(1) models?

Reproducible codes

MH Manuel Haqiqatkhah

Table of contents

1 Introduction

This document contains the reproducible code for the manuscript Skewness and staging: Does the floor effect induce bias in multilevel AR(1) models? by M. M. Haqiqatkhah, O. Ryan, and E. L. Hamaker. please cite as:

…

In this study, we simulated multilevel data from three data generating mechanisms (DGMs), namely, the AR(1, χ^2AR(1), BinAR(1), and PoDAR(1) models with different parameter sets. For details, see the paper.

The simulation was conducted using the following modular pipeline design, inspired by Bien's R package `simulator` (2016), consisting of the following **components**:

A. *Simulation*: generating the datasets
B. *Analysis*: modeling the data
C. *Harvesting*: collecting the relevant parameter estimates
D. *Reporting*: making tables and plots

And the components were placed in a **pipeline**, that managed:

1. Making the simulation design matrix that include all relevant conditions
2. Book-keeping data files belonging to each replication of each condition
3. Performing simulations in batch
4. Performing Analyses in batch
5. Collecting the data in batch

This document is structured as follows. In Section 2, we explain the four components and the functions used therein. Then, in Section 3 we explain the wrapper functions used in the pipeline, and show how the pipeline was—and can be—executed. Finally, in Section 4, we discuss how the harvested data was used to make the figures used in the paper (and others that were not included).

Before we begin, we need to read the scripts to include them in this document. By default, none of the scripts run here (as they are time consuming). To run the scripts of each component, you can change the following variables to `TRUE` and re-render the document:

In case you want to change the scripts (e.g., to run a smaller portion of the simulation, or try other parameters, etc.), you should look up the `kintr` parameters called in each chunk (with `<<some_param>>`) and find the corresponding code (under `## @knitr some_param`) in the `scripts` folder.

2 Core components

2.1 The *Simulation* component

The Simulation component consists of four sets of functions:

i. Functions that implement the DGMs and generate univariate ($N = 1$) time series of length T from the parameters given to them;
ii. Wrappers that interface the DGM functions;
iii. A function that
iv. A wrapper to generate datasets (consisting on N time series of length T)

2.1.1 Data-generating models specifications

First we define functions for each data-generating models (DGMs) that can produce univariate, single-subject ($N = 1$) time series of desired length T (default: `T = 100`) with the two canonical parameters and a given random seed (default: `seed = 0`). All model(-implied) parameters are saved in a list (called `pa`).

For each model, the first observation (X_1) is randomly drawn from the model-implied marginal distribution, to eliminate the need for removing the burn-in window in the beginning of the data. After the data is generated, in case the argument `only.ts` is set to be `TRUE`, the raw data (as a vector of length `T`) is returned. Otherwise, the function calculates empirical dynamic (ϕ) and marginal (μ, σ^2, and γ) parameters based on the simulated data, and save it in a list (`Empirical.Parameters`). Furthermore, two $\LaTeX$-ready strings (`Model.Description` and `Model.Description.Short`) are made which include a summary of the model parameters (that can be used, e.g., in plots). Finally, in case `only.ts != TRUE`, the function returns a list consisting of the time series (stored in `x`), verbal description of the dataset (`Model.Description` and `Model.Description.Short`), theoretical (i.e., model-implied) parameters (`Model.Parameters`), and empirical (i.e., sample) estimated parameters (`Empirical.Parameters`).

2.1.1.1 The AR(1) model

The canonical parameters of the AR(1) model with normally distributed residuals (which we referred to as `NAR(1)` in the simulation) are the autoregressive parameter ϕ (default: `phi = 0.4`), mean μ (default: `Mean = 50`), and the marginal variance σ^2 (default: `var.marginal = 4`). Based on the marginal variance, the residual variance (`var.resid`) is calculated via $\sigma_\epsilon^2 = \sigma^2(1-\phi^2)$.

2.1.1.1.1 Construction

The time series is constructed by first generating a zero-centered time series $\tilde{X}_t$ (`x_cent`). To do so, first the initial observation in the time series (`x_cent[1]`) is sampled from normal distribution with mean zero and a variance equal to the marginal variance of the model:

$$\tilde{X}_1 \sim \mathcal{N}(0, \sigma^2)$$

Then, the remainder of the time series is generated using the definition of the AR(1) model (not that the here the residual variance is used in the normal distribution):

$$\begin{gathered}\tilde{X}_t = \phi\tilde{X}_{t-1} + \epsilon_t \\ \epsilon_t \sim \mathcal{N}(0, \sigma_\epsilon^2)\end{gathered}$$

Finally, the mean is added to the centered zero-centered time series to reach the final time series with mean μ:

$$X_t = \tilde{X}_t + \mu$$

2.1.1.1.2 Code

2.1.1.2 The χ^2AR(1) model

The canonical parameters of the χ^2AR(1) model (which we referred to as `ChiAR(1)` in the simulation) are the autoregressive parameter ϕ (default: `phi = 0.4`), and degrees of freedom ν (default: `nu = 3`). We set the intercept to zero (`c = 0`).[1]

[1]The χ^2AR(1), in a more general form, can have an intercept ($X_t = c + \phi X_{t-1} + a_t, \quad a_t \sim \chi^2(\nu)$. Since the intercept was set to zero in the simulation study, we discussed a zero-intercept version of this model ($c = 0$) in the paper. See the Supplemental Materials for more details.

2.1.1.2.1 Construction

Similar to the AR(1) model, we need to sample the first observation of the χ^2AR(1) model from its marginal distribution. However, since this model does not have a closed-form marginal distribution, as an approximation, we instead sample `x[1]` from a χ^2 distribution with ν degrees of freedom:

$$X_1 \sim \chi^2(\nu)$$

Then, we generate the remainder of the time series using the definition of the χ^2AR(1) model:

$$\begin{aligned} X_t &= c + \phi X_{t-1} + a_t \\ a_t &\sim \chi^2(\nu). \end{aligned}$$

2.1.1.2.2 Code

2.1.1.3 The BinAR(1) model

The canonical parameters of the BinAR(1) model (which we referred to as `BinAR(1)` in the simulation) are the survival probability α (default: `alpha = 0.5`) and the revival probability β (default: `beta = 0.4`). By default, the maximum value on scale k was set to `k = 10`.

2.1.1.3.1 Construction

We first calculate the θ parameter, which characterizes the marginal distribution of the BinaR(1) model:

$$\theta = \frac{k\beta}{1-(\alpha-\beta)}$$

Then we draw X_1 (`x[1]`) from the marginal distribution of the model:

$$X_1 \sim Binom(k, \theta)$$

The rest of time series is generated sequentially, for each time point t, by drawing values for the number of survived (`S_t[t]`) and revived (`R_t[t]`) elements of the BinAR(1) model based on the previous observations (X_{t-1}), and then adding them:

$$\begin{aligned} S_t &\sim Binom(X_{t-1}, \alpha) \\ R_t &\sim Binom(k - X_{t-1}, \beta) \\ X_t &= S_t + R_t \end{aligned}$$

2.1.1.3.2 Code

2.1.1.4 The PoDAR(1) model

The canonical parameters of the PoDAR(1) model (which we referred to as `PoDAR(1)` in the simulation) are the persistence probability τ (default: `tau = 0.7`) and the average rate λ (default: `lambda = 0.5`).

2.1.1.4.1 Construction

To generate the time series, we first draw the first observation X_1 (`x[1]`) from a Poisson distribution with rate λ:

$$X_1 \sim Poisson(\lambda)$$

And generate the rest of the time series by first drawing Z_t from a Poisson distribution with rate λ and P_t from a binomial distribution with size probability of success τ (that is equivalent to a Bernoulli distribution with probability τ). Then, we calculate X_t based on the previous observation (`x[t-1]`) and values of Z_t (`Z_t[t]`) and P_t (`P_t[t]`), using the definition of the PoDAR(1) model:

$$\begin{aligned} Z_t &\sim Poisson(\lambda) \\ P_t &\sim Binom(1, \tau) \\ X_t &= P_t X_{t-1} + (1 - P_t) Z_t \end{aligned}$$

2.1.1.4.2 Code

2.1.2 General DGM wrappers

Given that, in each model, two canonical parameters characterize the dynamic and marginal features of the generated time series, and given that we have analytic formulas that link the canonical parameters to the model-implied ϕ, μ, σ^2, and γ, we use a function (`dgm_parameterizer`) to calculate canonical parameters from two given parameters, and make a complete list of parameters (called pa). This list also includes non-parameter variables, importantly, the time series length T (saved in `pa$T`) and the random seed used in the `dgm_*` functions (saved in `pa$seed`). A wrapper function (`dgm_generator`) is used as an interface to all `dgm_*` functions, which first makes sure the given parameters are sufficient for data generation, makes a complete parameter list `pa` with the help of `dgm_parameterizer`, and passes `pa` to the respective DGM generating function.

2.1.2.1 Parameter conversions

The function `dgm_parameterizer` calculates canonical/model-implied parameters of a given DGM (specified using the `Model` argument) based on the parameters given to it as arguments, and saves them in a list of parameters (`pa`), which s returned by the function. The function makes sure that the set of parameters provided are sufficient to characterize the dynamic parameter of the model (i.e., the autoregression ϕ) and at least one of the marginal parameters (importantly, the mean μ) but giving default values to some parameters.

2.1.2.2 Wrapper around `dgm_*` functions

The function `dgm_generator` gets a set of parameters (either as separate arguments, or a list of parameters, like the one returned by `dgm_parameterizer`), saves them in a list called `pa`. It checks whether ϕ is included in the list (if not, sets the default value `pa$phi = 0.2`), and checks if at least one other parameter (which, together with ϕ, is required to characterize the marginal properties of the DGMs) is calculated for it (if not, it sets the default value `pa$Mean = 5` for μ). Furthermore, if the DGM name, time series length, and the random seed are not provided, it gives them default values (respectively: `Model = "ChiAR(1)"`, `T = 100`, and `seed = 0`) and adds them to `pa`.

Then, it passes the `pa` list to `dgm_parameterizer` to do the necessary conversions to complete the list of canonical and model-implied parameters. Finally, given the model name, it checks if non-canonical parameters k and c are set (otherwise assigns appropriate defaults to them), and passes the complete parameter list to the respective DGM function.

2.1.3 Dataset generation

The machinery described above can be used to generate individual ($N = 1$) time series. However, for the simulation study, we need datasets comprising of multiple ($N = 25, 50, 100$) individuals. As we discussed in the paper, in our study, all individuals in a dataset of a DGM share the same autoregressive parameter ($\phi_i = 0.4$) and the individual differences are only in the individual means (μ_i). Thus, we write a function (`dgm_make.sample`) that can generate, for each DGM, a dataset of N individuals based on an N-dimensional vector of individual means. Then, with a wrapper function (`make_datasets`), we generate datasets

2.2 The *Analysis* component

2.3 The *Harvesting* component

2.4 The *Reporting* component

3 Pipeline

We implemented each of these tasks in separate functions that were essentially wrapper functions (with parallel-computing implementation) around the modular components. Using these wrapper functions, each replication of each simulated condition was saved in a separate `.rds` file. These data files were fed to the analysis wrapper function whose output was saved in separate `.rds` data files. To collect relevant parameter estimates, another wrapper function was used to read the data files and save the desired parameters in a dataframe, which then used in reporting.

The outcomes of the components are saved in separate `.rds` files and indexed by unique, descriptive names, and each replication is given a unique numeric identifier that is also used as the random seed used to generate the dataset within each replication. The file names and address are stored in two dataframes along with model parameters used to generate each dataset, and these dataframes are used when reading and writing data files in other components.

3.1 Wrapper functions

3.2 Pipeline code

4 Making the figures

4.1 Simulation results

Profiles of simulated datasets

Bien, Jacob. 2016. "The Simulator: An Engine to Streamline Simulations." *arXiv:1607.00021 [Stat]*, June. http://arxiv.org/abs/1607.00021.