DynaMORE

Dynamic MOdelling of REsilience

H2020 - 777084

D 1.2– Open-source code of calibrated resilience model

Dissemination level	Public
Contractual date of delivery	31.03.2022
Actual date of delivery	29.03.2022
Type	Report
Version	1.0
Filename	DynaMORE_Deliverable_report_D1.2
Workpackage	WP1
Workpackage leader	ALU-FR (Jens Timmer)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 777084.

Author list

Organisation	Name	Contact information
University of Freiburg, Germany	Shakoor Pooseh	shakoor.pooseh@fdm.uni-freiburg.de
University of Freiburg, Germany	Jens Timmer	jeti@fdm.uni-freiburg.de

Executive Summary

This report summarizes the implementation of an exogenous linear autoregressive mixed-effects model for the estimation of intra-individual dynamic networks of affects in the programming language Julia. The model is suitable for data acquired by ecological momentary assessment. We lay down the theoretical background and the build up of the data types and functions that represent the data generating process in a step-by-step fashion. We also provide the tools which are necessary to prepare simulated and real data to be fitted by the Julia package *MixedModels.jl*. At this stage the model is calibrated through simulation and the source code is stored in a private password protected GitLab repository, maintained by WP2, under https://gitlab.imbi.uni-freiburg.de/DynaMORE/larmex. This model will be further calibrated and validated using real data and eventually will be published as a Julia package by the end of DynaMORE project.

The following sections of this text are written and maintained as a Jupyter notebook in Julia and will accompany the final published open-source code as its manual.

1 Introduction

We consider very simple directed intra-individual networks comprising two symptom nodes and one node for external factors. The two symptom nodes interact by enhancing or dampening each other and the external factors are assumed to be acting on both nodes in a one-way fashion, i.e. the possible effects of a symptom on the perception of external factors are neglected.

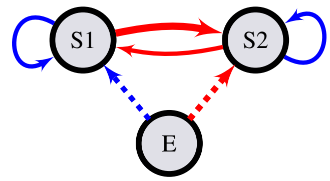

Figure 1: Variables, $S1$ and $S2$, interact with each other leading to temporal correlation meaning that the value of the source variable (node) at time t has an impact on the value of its target at time $t+1$. Looped arrows indicate autoregressive effects. Exogenous factors act on moods at the same time in a one-way fashion. Blue and red arrows indicate activation and suppression respectively. Different thicknesses stand for distinct interaction strengths. Dashed arrows indicate contemporaneous effects while solid ones are indicative of possible temporal causalities.

Assuming intensive longitudinal data through **ecological momentary assessment**, we formalize the mathematical representation of such networks, Figure 1, by exogenous Linear Auto-Regressive Mixed-Effects (**LARMEx**) models such that the autoregressive coefficients represent the interactions and the external factors enter the model as exogenous covariates. We let every parameter in the model to have fixed and random components aiming at networks that are allowed to have variable structures over reasonable units of time like days or weeks depending on the study design. Given the fact that a rich theoretical and computational literature furnishes classic linear mixed-effects models, we transform the autoregressive formulation to a classical one and use the already developed methods and tools. Then assuming our model is the true data generating process, we simulate data using a predefined set of parameters and investigate the performance and feasibility of this approach in delivering reliable estimates for different choices of the number of observations and the intensity of noise.

2 Data generating process

Let $s_{i,t} = [s_1, s_2]^T_{i,t}$ be the 2×1 vector corresponding to the symptom values from the ith day at any observation occasion $t = 1, 2, \ldots, n_i$ where n_i is the number of observations per day and T denotes the transpose of a matrix.

Assuming that a collection of external factors (E) act on symptom nodes and their effect could vary between days, the evolution of $s_{i,t}$ is represented by a LARMEx model as

$$s_{i,t} = (\beta^{ar} + b_i^{ar})s_{i,t-1} + (\beta^e + b_i^e)e_{i,t} + (\beta^c + b_i^c) + \epsilon_{i,t},$$

in which β and b represent the fixed and random effects (FE and RE henceforth). The temporal (Granger-causal) connections between the symptoms are governed by the lag(1) autoregressive term, $(\beta^{ar} + b_i^{ar})s_{i,t-1}$. The remaining terms represent the contemporaneous effects of E and the constants (intercepts). For simplicity we assume no connections from symptoms to E.

3 Generating a set of known parameters

For simulating data with this model we need to specify the parameters. In Julia we implement this using a `struct`.

```
mutable struct parLARMEx
    nVar::Int                # number of temporally connected nodes
    seed::Int                # an integer to replicate, 0 to initialize the RNG anew
    rng::AbstractRNG         # RNG throught the simulation for consintency
    nL2Max::Int              # how many random-effects to generate
    nSamp::Int               # initial sample size for generating REs, see genRE_CS()
    B_AR::Matrix{Float64}    # fixed-effects autoregressive coefficients
    B_E::Vector{Float64}     # fixed-effects coefficients of exogenous factors
    b_var::Matrix{Float64}   # variance of group of random-effects
    b_cov::Matrix{Float64}   # variance-covariance matrix of random-effects
    b_ar::Matrix{Float64}    # random-effects for autoregressive coefficients
    b_e::Matrix{Float64}     # random-effects for exogenous coefficients
    b_c::Matrix{Float64}     # random-effects for constant terms
end
```

For the initialization of this `struct` we implement its constructor as

```
parLARMEx(;b=[], nVar=2, seed=0, nL2Max=100, nSamp=20000, B_ar=.3, B_e=.3,
           b_var=[.03 .03 .03])
```

where the keyword arguments are:

- `b = []`: if provided with a matrix, RE are extracted from it otherwise generated
- `nVar = 2`: number of temporally connected nodes
- `seed = 0`: an integer to replicate, 0 to initialize the random number generator (RNG) anew
- `nL2Max = 100`: how many RE to generate
- `nSamp = 20000`: initial sample size for generating RE, see genRE_CS() for explanation
- `B_ar = .3`: absolute value of FE autoregressive coefficients, = `[]` for random values between 0.1 and 0.6
- `B_e = .3`: absolute value of FE exogenous coefficients, = `[]` for random values between 0.1 and 0.6
- `b_var = [.03 .03 .03]`: for constructing a default variance-covariance of RE

For simplicity we restrict the fixed-effects to be

$$\beta^{ar} = \begin{bmatrix} 0.3 & -0.3 \\ -0.3 & 0.3 \end{bmatrix}, \quad \beta^{e} = \begin{bmatrix} 0.3 \\ -0.3 \end{bmatrix}, \quad \beta^{c} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

which are generated using keywork arguments to the constructor of `parLARMEx`. It is also possible to initialize these with random values by passing `[]`. The variance-covariance of the random-effects is built using `b_var` to be

$$G = \begin{bmatrix} 0.03 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.03 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.03 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.03 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.03 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.03 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.03 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.03 \end{bmatrix}$$

This is not the final variance-covariance of the RE because one needs to control for the stability of the autoregressive process. To this end, we sample a number of `nSamp = 20000` set from a multivariate normal distribution of $\mathrm{MVN}(0, G)$ and retain `nL2Max = 100` set for which the absolute eigenvalues of $\beta^{ar} + b^{ar}$ are less than one. These are stored in `b_ar`, `b_e`, `b_c`, and the variance-covariance of this latter sample as `b_cov`.

This process is accomplished by calling the function

```
genRE_CS(rng,G,B_AR,maxSbj,nSamp)
```

inside the constructor which updates the fields corresponding to the random-effects and their variance-covariance structure. In what follows we import the function definitions saved at the file `helperSim.jl`, included in the appendix, and then construct the parameters using a replicable RNG following `seed = 1984`.

[1]:
```
include("helperSim.jl");
```

[2]:
```
par = parLARMEx(seed=1984);
```

4 Generating data

We assume that the data generating process has a two-level structure. The multiple observations are made on level I and these are nested in level II units. For EMA data, level I could be the observations within one day, as many as `nObsL1`, which are nested in single days with a total number of `nL2`. These characteristics together with the variance-covariance of noise, `sigma`, initial values for days, `S0`, and the vector of known exogenous factors, `E`, are stored in another `struct`.

```
mutable struct simLARMEx
    nObsL1::Int             # number of observation on level I
    nL2::Int                # number of units on level II
    sigma::Float64          # variance of noise
    S0::Matrix{Float64}     # initial values each day
    E::Vector{Float64}      # known exogenous factors
end
```

The constructor of this `struct` has the following signature,

```
simLARMEx(;rng=MersenneTwister(), nObsL1=10, nL2=36, sigma=.2, S0_max=.5, E=[])
```

and keyword arguments as:

- `rng = MersenneTwister()`: feed `parLARMEx.rng` for consistency and replication, initialized anew by default
- `nObsL1 = 10`: number of observation on level I
- `nL2 = 36`: number of units on level II
- `sigma = .02`: variance of noise
- `S0_max = .5`: determines the amplitude of initial values and exogenous factors, 0.5 to keep trajectories mostly in [-1,1]
- `E = []`: known exogenous factors of size [`nL2` x `nObsL1`], drawn randomly from [0,`S0_max`] by default

The following code snippet sets up the configuration of the desired data set. The RNG is carried over to provide replicability. Finally, `genData()` gives the simulated data along with the noiseless data as dataframes and the signal-to-noise-ratio (SNR).

[3]:
```
sim = simLARMEx(rng=par.rng);
simData,simData0,SNR = genData(par,sim);
```

[4]:
```
# show(first(simData,5),allcols=true)
latexify(round.(simData[1:5,:],digits=2))
```

[4]:

idL2	tL1	S1	S2	E
1.0	1.0	−0.07	0.44	0.27
1.0	2.0	−0.03	0.2	0.49
1.0	3.0	0.2	0.08	0.19
1.0	4.0	0.3	−0.04	0.23
1.0	5.0	0.19	−0.17	0.26

The returned values are

- `simData`: the simulated data according to the setup so far
- `simData0`: the simulated data without the noise component used to calculate the SNR
- `SNR`: the ratio of the variance of noiseless data to that of noise

5 Transforming to classical mixed-effects model

By stacking symptom values, parameters and covariates for every day separately, and forming the design matrices for fixed and random effects, X and Z, data from a single day takes the following matrix form which is equivalent to a linear mixed effects formulation.

$$Y_i = X_i\beta + Z_i b_i + \epsilon_i$$

In this formulation for a network of two symptoms and one external nodes one has

$$Y_i = \begin{bmatrix} s_{1,1} & s_{1,2} & \dots & s_{1,n_i} & s_{2,1} & s_{2,2} & \dots & s_{2,n_i} \end{bmatrix}_i^T, \quad \beta = \begin{bmatrix} \beta_{11} & \beta_{12} & \beta_{21} & \beta_{22} & \beta_1^e & \beta_2^e & \beta_1^c & \beta_2^c \end{bmatrix}^T$$

and

$$X_i = Z_i = \begin{bmatrix} s_{1,0} & s_{2,0} & 0 & 0 & 1 & e_1 & 0 & 0 \\ s_{1,1} & s_{2,1} & 0 & 0 & 1 & e_2 & 0 & 0 \\ \vdots & \vdots & & & & \vdots & & \\ s_{1,n_i-1} & s_{2,n_i-1} & 0 & 0 & 1 & e_{n_i} & 0 & 0 \\ 0 & 0 & s_{1,0} & s_{2,0} & 0 & 0 & 1 & e_1 \\ 0 & 0 & s_{1,1} & s_{2,1} & 0 & 0 & 1 & e_2 \\ \vdots & \vdots & \vdots & \vdots & & & & \vdots \\ 0 & 0 & s_{1,n_i-1} & s_{2,n_i-1} & 0 & 0 & 1 & e_{n_i} \end{bmatrix}$$

In its general form, the above equation for k symptoms, Y_i is a $k(n_i - 1) \times 1$ vector of symptom values, β is a $(k^2 + 2k) \times 1$ vector of fixed effects, b_i is a $(k^2 + 2k) \times 1$ vector of random effects, X_i and Z_i are $k(n_i - 1) \times (k^2 + 2k)$ design matrices. The random effects b_i and residuals ϵ_i are assumed to be independent with a multivariate normal (MVN) distribution of

$$\begin{bmatrix} b_i \epsilon_i \end{bmatrix} \sim MVN\left(\begin{bmatrix} 00 \end{bmatrix}, \begin{bmatrix} G & 00 & \Sigma_i \end{bmatrix}\right).$$

This step is done by the following function which gives another dataframe suitable for performing the estimation.

```
prepData2Fit(rawData,idL2,endList,exgList)
```

with the following arguments:

- `rawData`: the simulated data as a dataframe
- `idL2`: the column name for level II units, e.g., an id for each day, `"idL2"` here
- `endList`: the list of temporelly connected symptoms, `["S1","S2"]` here
- `exgList`: the list of exogenous factors together with the constant terms, `["E","C"]` here

[5]:
```
fitData = prepData2Fit(simData,"idL2",["S1","S2"],["E","C"]);
# show(first(fitData,5),allcols=true)
```

[6]:
```
df = fitData[1:5,:]; df[:,3:end] = round.(df[:,3:end],digits=2);
latexify(df)
```

[6]:

idL2	tL1	S	S11	S12	S21	S22	E1	E2	C1	C2
1.0	2.0	−0.03	−0.07	0.44	−0.0	0.0	0.49	0.0	1.0	0.0
1.0	3.0	0.2	−0.03	0.2	−0.0	0.0	0.19	0.0	1.0	0.0
1.0	4.0	0.3	0.2	0.08	0.0	0.0	0.23	0.0	1.0	0.0
1.0	5.0	0.19	0.3	−0.04	0.0	−0.0	0.26	0.0	1.0	0.0
1.0	6.0	0.65	0.19	−0.17	0.0	−0.0	0.49	0.0	1.0	0.0

6 Setting up the formula for fitting

The prepared data has more columns representing the response variable `S`, and the predictors including the connections in the network `S11, S12, S21, S22, E1, E2`, as well as the constant terms `C1, C2`. It also contains the level I time points `tL1`, and the level II identifiers `idL2`. We provide a function

```
setFormula(fitData)
```

which constructs the formula suitable to feed in the Julia package MixedModels.jl. This function is restricted only to a dataframe which has the exact columns as shown above.

[7]:
```
frm = setFormula(fitData);
```

7 Fitting

We use the MixedModels.jl package in Julia to perform the estimation. It is similar to the lme4 in R but exhibits faster performance in our case.

[8]:
```
fit = MixedModels.fit(MixedModel, frm, fitData, progress=false);
show(fit)
```

```
Linear mixed model fit by maximum likelihood
 S ~ 0 + S11 + S12 + S21 + S22 + E1 + E2 + (0 + S11 + S12 + S21 + S22 + E1␣
 ↪+ E2
+ C1 + C2 | idL2)
   logLik   -2 logLik     AIC       AICc        BIC
   246.8663  -493.7327  -407.7327  -401.4678  -215.3554

Variance components:
         ColumnVariance Std.Dev.   Corr.
idL2     S11   0.019109 0.138235
         S12   0.065390 0.255715 +0.26
         S21   0.025455 0.159545 -0.50 +0.68
```

```
        S22  0.048207 0.219560 -0.70 +0.01 +0.37
        E1   0.027348 0.165371 +0.49 +0.27 -0.05 -0.58
        E2   0.021337 0.146070 +0.77 +0.57 -0.11 -0.45 +0.28
        C1   0.031647 0.177895 -0.11 +0.20 +0.41 -0.15 +0.38 -0.45
        C2   0.031128 0.176433 +0.33 +0.03 -0.15 -0.17 -0.31 -0.01 +0.34
Residual     0.017568 0.132545
 Number of obs: 648; levels of grouping factors: 36

  Fixed-effects parameters:
───────────────────────────────────────────
         Coef.  Std. Error      z  Pr(>|z|)
───────────────────────────────────────────
S11   0.16115    0.0441711   3.65    0.0003
S12  -0.482173   0.059415   -8.12    <1e-15
S21  -0.281178   0.0449767  -6.25    <1e-09
S22   0.342208   0.0553473   6.18    <1e-09
E1    0.394389   0.0587841   6.71    <1e-10
E2   -0.254405   0.0542693  -4.69    <1e-05
───────────────────────────────────────────
```

8 Parameter recovery

In order to investigate how well one can recover the parameters, we generate data with the following specifications:

- number of observations per day: 10
- $(\sigma^2, \mathrm{SNR}) \in \{(0.01, 12), (0.02, 6), (0.06, 2)\}$
- number of days, $N \in \{4, 6, \ldots, 36\}$
- β^{ar}, β^{e} and β^{c} as before
- data is generated for one simulated subject, 1000 times repeat
- at every iteration new parameters are generated

The following code in Julia creates a folder called `bootstrap` in the current working directory and for each noise intensity, `sigma = .02` here, saves the true and estimated parameters of each simulation in different folders, `re` and `reh`, based on the number of days as `CSV` files. Fixed-effects and the variances of random-effects are collected in two separate `CSV` files as well.

```
function loopSim(csvDir, nSim, S0_max, sigma, n, P)
    feh = zeros(nSim, 6)
    sig = zeros(nSim, 9)
    col = []
    endList = ["S1", "S2"];
    exgList = ["X", "C"];
    for j in 1:nSim
```

```
        par = parLARMEx(b=hcat(P.b_ar,P.b_x,P.b_c), nL2Max=n);
        fName = joinpath(csvDir, "re", @sprintf("re_n%02d_%03d.csv", n, j));
        re2csv(par, n, ["M1","M2"], ["X","C"], fName);
        sim = simLARMEx(nL2=n, S0_max=S0_max, sigma=sigma);
        simData,_,_ = genData(par, sim);
        fitData = prepData2Fit(simData, "idL2", endList, exgList);
        frm = setFormula(fitData);
        res = MixedModels.fit(MixedModel, frm, fitData, progress=false);
        reh = DataFrame(only(raneftables(res)))
        CSV.write(joinpath(csvDir, "reh", @sprintf("reh_n%02d_%03d.csv",n,j)), reh)
        feh[j,:] = coef(res)
        sig[j,:] = vcat(collect(res.sigmas.idL2), res.sigma)
        col = coefnames(res)
    end
    return feh, sig, col
end
function wrapScropt()
    baseDir = @sprintf("./bootstrap") # where to save data
    mkpath(baseDir)
    nSim = 1000  # number of simulations
    b_var=.03 * [1 1 1]; S0_max=.5; sigma=.02;
    nDay = 4:2:37 # number of days
    par = parLARMEx(b_var=b_var, nL2Max=1000); # generate RE for a large number of days
    sim = simLARMEx(S0_max=S0_max, sigma=sigma); # specify the simulation setting
    for n in nDay
        csvDir = joinpath(baseDir, @sprintf("sig%.2f/n%02d/",sigma,n))
        mkpath(csvDir)
        fehName = @sprintf("feh_n%02d.csv",n) # CSV to store fixed-effects
        for d in ["re" "reh"] # folders for true and estimated random-effects
            cD = joinpath(csvDir, d)
            mkpath(cD)
        end
        try
            feh, reSig, col = loopSim(csvDir, nSim, S0_max, sigma, n, par)
            feh = DataFrame(feh, col)
            CSV.write(joinpath(csvDir, fehName), feh)
            reSig = DataFrame(reSig,vcat(col,["C1", "C2", "sigma"]))
            CSV.write(joinpath(csvDir, @sprintf("sig_n%02d.csv",n)), reSig)
        catch e
            bt = catch_backtrace()
            msg = sprint(showerror, e, bt)
            println(msg)
            break
         end
```

```
    end
end
wrapScript()
```

The code takes a couple of hours, depending of course on the computational power. Using the estimated parameters we construct bootstrap confidence intervals for the fixed-effects, Figure 2, and the variance of random-effects, Figure 3.

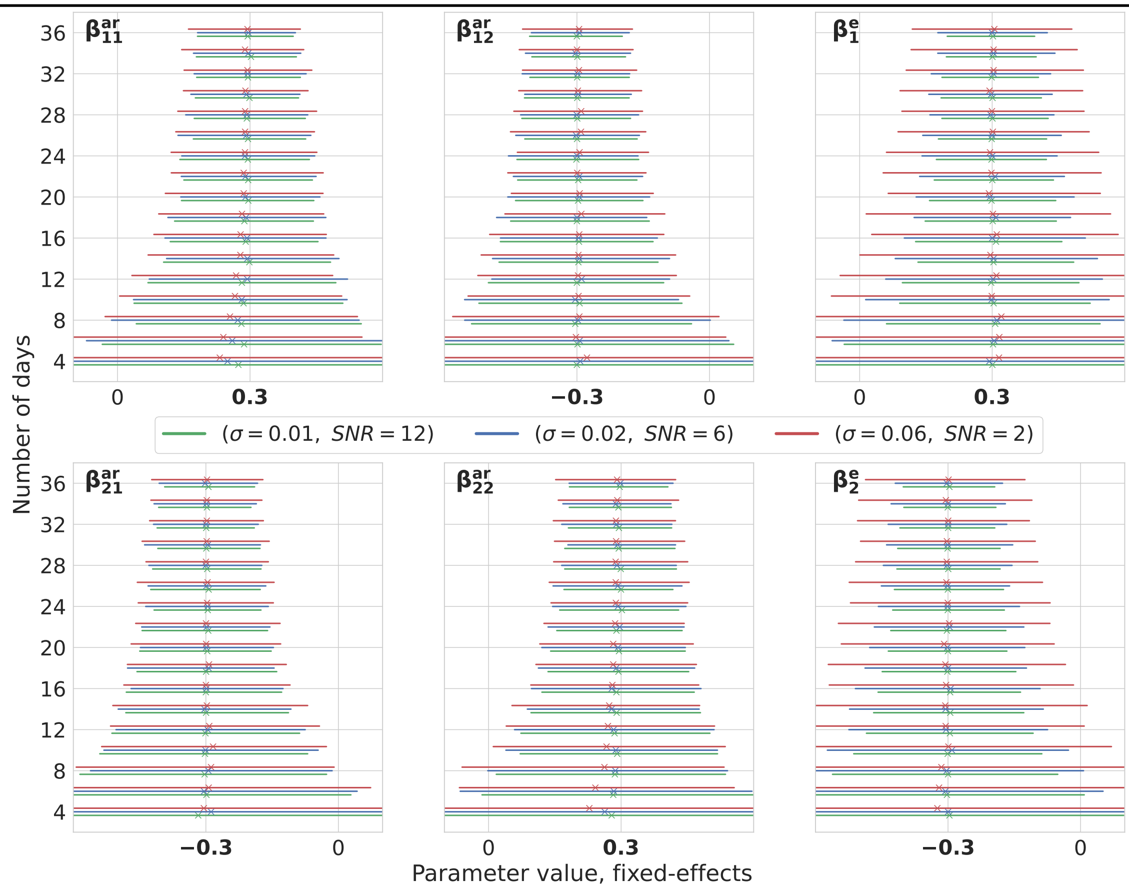

Figure 2: Estimations of fixed effects for different levels of noise intensity and number of days. The number of observations per day is 10. True values of parameters are highlighted bold on x axes. For every number of days at the horizontal axes, three lines are drawn representing the bootstrap 95% confidence intervals around the median depicted by cross signs. Every line is color-coded to demonstrate one noise intensity. Data has been generated for one simulated subject, 1000 times repeatedly, with $(\sigma^2, \text{SNR}) \in \{(0.01, 12), (0.02, 6), (0.06, 2)\}$ and estimations are done using *MixedModels.jl* package in Julia.

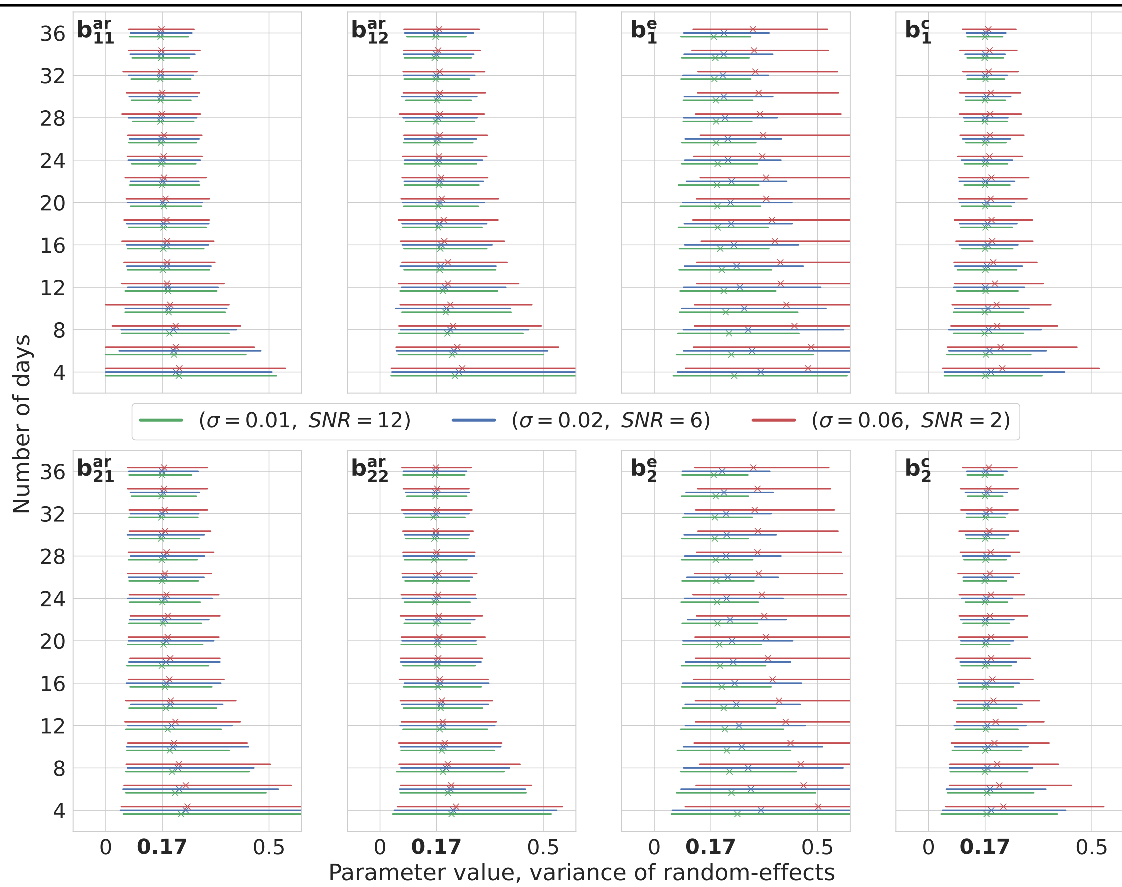

Figure 3: Estimations of the variance of random effects for different levels of noise intensity and number of days. The number of observations per day is 10. True values of parameters are highlighted bold on x axes, $\sqrt{0.03} \approx 0.17$. For every number of days at the vertical axes, three lines are drawn representing the bootstrap 95% confidence intervals around the median depicted by cross signs. Every line is color-coded to demonstrate one noise intensity. Data has been generated for one simulated subject, 1000 times repeatedly, with $(\sigma^2, \text{SNR}) \in \{(0.01, 12), (0.02, 6), (0.06, 2)\}$ and estimations are done using *MixedModels.jl* package in Julia.

We also calculate the relative estimation error, δ, in the aforementioned cases as $\delta_\theta = |\hat{\theta} - \theta|/|\theta|$ in which θ and $\hat{\theta}$ are the true and estimated parameters respectively with $(\sigma^2, \text{SNR}) = (0.02, 6)$. To avoid computational difficulties only random effects larger than 0.02 are considered in these analyses. We also report only the coefficients present in a network representation, i.e., β^{ar}, β^{e}, b^{ar} and b^{e}. This is to show that in a mixed effects model the prediction of random effects is not as reliable as fixed effects in terms of being able to recover the parameters varying over days in our case. The reason is that only the variance of random effects are present in the likelihood function and individual values are not estimated directly but only up to their variance. More precisely what one gets as the output of software packages like MixedModels.jl in Julia or lme4 in R holds only for the expected values and should not be considered as single parameter estimations. In the following figure it is clearly seen that the median relative estimation error for random effects approaches to about 40% as the number of days grows and the change is minimal after 20 days. Whereas, for the fixed effects, the value is below 20% which indicates that with a reasonable amount of data one could hope for even better estimates for the fixed effects but this is still far from being optimal. The two separate solid and dashed lines correspond to the coefficients of the exogenous factors which have poor estimations compared to others, Figure 4.

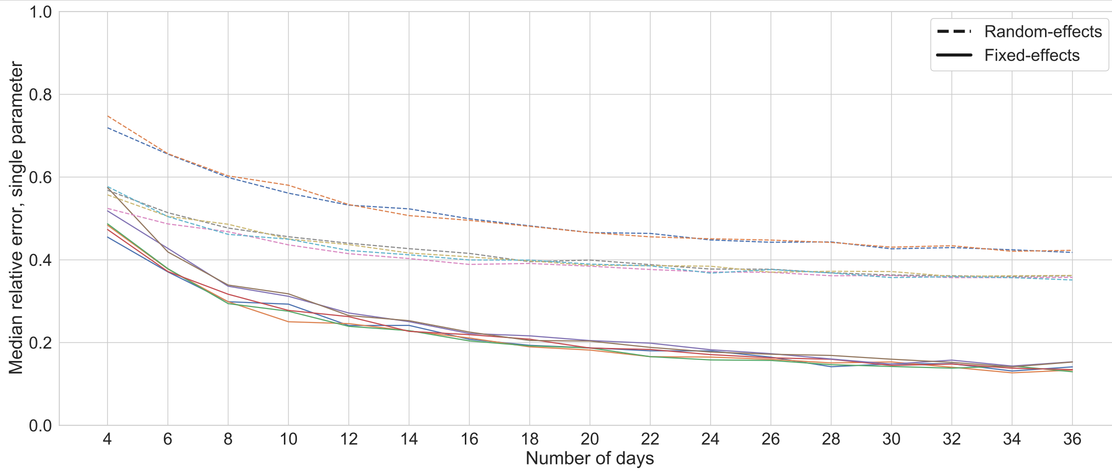

Figure 4: *Mean relative estimation errors*. Solid and dashed lines represent the mean relative estimation errors for the six parameters of a network as in Figure **??** fixed. Data is generated for one simulated subject, 1000 times repeatedly, with $(\sigma^2, \mathrm{SNR}) = (0.02, 6)$ and estimations are done using *MixedModels.jl* package in Julia. The two lines with higher relative errors correspond to exogenous factors.

9 Conclusion

In this work, we studied an extension of linear mixed-effects models by adding an autoregressive component to model the network representation of mental state which is suitable to model intensive longitudinal data, the so-called ecological momentary assessments acquired by experience sampling methods. Specifically, we assumed the simplest possible network of two causally interacting state nodes under the influence of external factors represented by a node acting on both states in a contemporaneous fashion.

We showed briefly how this representation is mathematically formulated and detailed the implementation of such a model in Julia. The practical implementation of the data generating process was shown by explicit codes which will be available on online public repository, e.g., GitLab, as Julia package or a webapp accompanied by manuals as notebooks like this report as well as all their implementations in Julia. We also provided exemplary code for performing a bootstrap analysis to construct confidence intervals for the estimations. The goal here was to calibrate the model through simulations by quantifying the estimation errors. More calibration and validation will be performed as more data acquired in the course of the ongoing DynaMORE project.

Given that our target was building individualized networks, we based our study on a two-level model with daily observations nested in days for a specific respondent. Using simulated data, we constructed bootstrap confidence intervals for the fixed effect parameters and showed what one could expect from this model as more days are added to the observation. We argued that random effects are not directly estimated in this procedure and highlighted the difference between the relative estimation errors for fixed and random effects which showed that in a mixed model random effects are identified less precisely and as the number of observations increases the gain in precision is very little. Therefore, in order to infer individual networks from ecological momentary assessments, one should build a two-level model with daily observations nested in days for a single respondent.

We mainly considered one aspect of study design related to sample size and varied the number of observations per subject by adding more days. We also showed how sensitive such a model might be to the intensity of noise in data. However, the provided code could be modified to study other variants to these scenarios.

9.1 Appendix: `helperSim.jl`

```
using LinearAlgebra, Random, Distributions, DataFrames
using MixedModels, CSV, CategoricalArrays, Latexify

"""
    parLARMEx(;b=[], nVar=2, seed=0, nL2Max=100, nSamp=20000,
               B_ar=.3, B_e=.3, b_var=[.03 .03 2])

Construct the struct holding the settings for generating parameters
for simulation.

The default values of the keyword arguments specify a model with two
temporally connected network nodes with one exogenous factor acting on both.
## keyword arguments:
- `b = []`: if provided with a matrix random-effects are extracted,
            otherwise generated
- `nVar = 2`: number of temporally connected nodes
- `seed = 0`: an integer to to replicate, 0 to initialize the
              random-number-generator anew
- `nL2Max = 100`: how many random-effects to generate
- `nSamp = 20000`: initial sample size for generating random-effects,
                   see genRE_CS() for explanation
- `B_ar = .3`: absolute value of fixed-effects autoregressive coefficients,
               `= []` for random values between 0.1 and 0.6
- `B_e = .3`: absolute value of fixed-effects exogenous coefficients,
              `= []` for random values between 0.1 and 0.6
- `b_var = [.03 .03 .03]`: for constructing a default variance-covariance of RE
"""
mutable struct parLARMEx
    nVar::Int
    seed::Int
    rng ::AbstractRNG
    nL2Max::Int
    nSamp::Int
    "Fixed-effects autoregressive coefficients"
    B_AR::Matrix{Float64}
    "Fixed-effects exogenous coefficients"
    B_E::Vector{Float64}
    b_var::Matrix{Float64}
    "Variance-covariance of random-effects"
    b_cov::Matrix{Float64}
    "Random-effects autoregressive coefficients of size [`nVar^2` x `nL2Max`]"
    b_ar::Matrix{Float64}
    "Random-effects exogenous coefficients of size [`nVar` x `nL2Max`]"
```

```
    b_e::Matrix{Float64}
    "Random-effects constant terms of size [`nVar` x `nL2Max`]"
    b_c::Matrix{Float64}
    function parLARMEx(;b=[], nVar=2, seed=0, nL2Max=100, nSamp=20000,
                        B_ar=.3, B_e=.3, b_var=[.03 .03 .03])
        if seed === 0
            rng = MersenneTwister();
        else
            rng = MersenneTwister(seed);
        end
        if isempty(B_ar)
            B_AR =  rand(rng,.1:.01:.6,2,2) .* [1 -1 ; -1 1]
        else
            B_AR = B_ar .* [1 -1 ; -1 1] #.6 .* Matrix(1.0I,nVar,nVar) .- .3;
        end
        G = Diagonal(vcat(b_var[1]*ones(nVar^2),repeat(b_var[2:end],inner=nVar)));
        if isempty(b)
            b_ar,b_e,b_c,b_cov = genRE_CS(rng,G,B_AR,nL2Max,nSamp)
        else
            samp = sample(rng,1:size(b)[1],nL2Max)
            b_ar = b[samp,1:nVar^2];
            b_e  = b[samp,(nVar^2+1):(nVar^2+nVar)];
            b_c  = b[samp,(nVar^2+nVar+1):(nVar^2+2*nVar)];
            b_var = b_cov = reshape(Float64[],0,2)
            nSamp = 0
        end
        if isempty(B_e)
            B_E =  rand(rng,.1:.01:.6,2,1) .* [1 ; -1]
        else
            B_E = B_e .* [1 ; -1]
        end
        return new(nVar,seed,rng,nL2Max,nSamp,B_AR,B_E,b_var,b_cov,b_ar,b_e,b_c)
    end
end

"""
    simLARMEx(;rng=MersenneTwister(), nObsL1=10, nL2=36, sigma=.02,
               S0_max=.5, E=[])

Construct the `struct` holding the settings for generating simulated data.

By the default values of the keyword arguments it is assumed that the data
generating process has a two-level structure. The multiple observations are
made on level I and these are nested in level II units.
## keyword arguments:
```

```
- `rng = MersenneTwister()`: feed `parLARMEx.rng` for consistency and
                             replication, initialized anew by default
- `nObsL1 = 10`: number of observation on level I
- `nL2 = 36`: number of units on level II
- `sigma = .02`: variance of noise
- `S0_max = .5`: determines the amplitude of initial values and exogenous
                 factors, 0.5 to keep trajectories mostly in [-1,1]
- `E = []`: known exogenous factors of size [`nL2` x `nObsL1`],
            drawn randomly from [0,`S0_max`] by default
"""
mutable struct simLARMEx
    nObsL1::Int
    nL2::Int
    sigma::Float64
    S0::Matrix{Float64}
    E::Vector{Float64}
    function simLARMEx(;rng=MersenneTwister(), nObsL1=10, nL2=36, sigma=.02,
                        S0_max=.5, E=[])
        rangeS = range(0,stop=S0_max,length=50)
        if isempty(E)
            E = sample(rng,rangeS,(nL2*nObsL1));
        end
        S0 = sample(rng,[-1 1],nL2) .*
             hcat(sample(rng,rangeS,nL2),sample(rng,-rangeS,nL2))
        return new(nObsL1,nL2,sigma,S0,E)
    end
end

"""
    genRE_CS(rng,G,B_AR,nL2Max,nSamp)

Generate random-effects parameter.

For consistency it is advised that one instance of a random-number-generator
should be used throughout the simulation. This function is called inside the
constructor of `parLARMEx` and draws a sample of size `nSamp` from a
multivariate normal Distributions MVN(0,`G`). Then using the fixed-effect
autoregressive coefficients, `B_AR`, retains tha random-effects for which
the eigenvalues of the sum of fixed- and random-effects are less tha one.
This guarantees that the autoregressive component of the process is stable.
Finally a number of `nL2Max` parameter sets are returned in the matrices
`b_ar`, `b_e`, `b_c` and their variance-covariance matrix as `b_cov`.
## Example:
b_ar,b_e,b_c,b_cov = genRE_CS(rng,G,B_AR,nL2Max,nSamp)
"""
```

```
function genRE_CS(rng,G,B_AR,nL2Max,nSamp)
    nVar = size(B_AR)[1] # nE  = length(b_var)-1; d = nVar^2 + nVar*nE;
    mvn = MvNormal(G);
    ri  = rand(rng,mvn,nSamp);
    ind = all.(eachslice(abs.(ri[1:(nVar^2+1*nVar),:]) .< .9,dims=2));
    ri  = ri[:,ind];
    r1  = reshape(ri[1:nVar^2,:],nVar,nVar,size(ri)[2]) .+ B_AR;
    eg  = eigvals.(eachslice(r1,dims=3));
    ind = [!any(abs.(i).>1) for i in eg];
    ri  = ri[:,ind];

    a = 1:nVar^2;
    e = (nVar^2+1):(nVar^2+nVar);
    c = (nVar^2+nVar+1):(nVar^2+2*nVar);

    samp = sample(rng,axes(ri,2),nL2Max)
    b_ar = ri[a,samp]'
    b_e  = ri[e,samp]'
    b_c  = ri[c,samp]'
    return b_ar,b_e,b_c,cov(ri');
end

"""
    genData(P,S)

Generate simulated data given two `struct` of parameter and data
specifications.

It returns the simulated data along with the noiseless data as
dataframes and the signal-to-noise-ratio `SNR`.
## Example:
simData,simData0,SNR = genData(parLARMEx(),simLARMEx());
"""
function genData(P,S)
    data1 = zeros(S.nL2*S.nObsL1,P.nVar)
    data0 = copy(data1)
    rho = 0 # Noise_rho;
    Sigma =  Matrix((S.sigma-rho)I,P.nVar,P.nVar) .+ rho;
    for i in 1:S.nL2
        BAR = P.B_AR + reshape(P.b_ar[i,:],P.nVar,P.nVar)';
        E1   = S.E[((i-1)*S.nObsL1+1):(i*S.nObsL1)]
        mvn = MvNormal(Sigma);
        d1 = copy(transpose(rand(P.rng,mvn, S.nObsL1)));
        d0 = 0 .* d1;
        d1[1,:] = S.S0[i,:];
```

```
        d0[1,:] = S.S0[i,:];
        ib = 1
        for t in 2:(S.nObsL1)
            d1[t,:] += BAR * d1[t-1,:] + (P.B_E .+ P.b_e[i,:]) *
                       E1[t] + P.b_c[i,:]
            d0[t,:] += BAR * d0[t-1,:] + (P.B_E .+ P.b_e[i,:]) *
                       E1[t] + P.b_c[i,:]
        end
        data1[((i-1)*S.nObsL1+1):(i*S.nObsL1),:] = d1[1:S.nObsL1,:]
        data0[((i-1)*S.nObsL1+1):(i*S.nObsL1),:] = d0[1:S.nObsL1,:]
    end

    tL1 = repeat(1:S.nObsL1,S.nL2);
    idL2 = repeat(1:S.nL2,inner=S.nObsL1);
    if all(P.B_E .== 0)
        simData  = DataFrame(idL2=idL2,tL1=tL1);
        simData0 = DataFrame(idL2=idL2,tL1=tL1);
        cols = ["S1","S2"]
    else
        simData  = DataFrame(idL2=idL2,tL1=tL1,E=S.E);
        simData0 = DataFrame(idL2=idL2,tL1=tL1,E=S.E);
        cols = ["S1","S2","E"]
    end
    for i in 1:P.nVar
        insertcols!(simData ,i+2,cols[i] => data1[:,i]);
        insertcols!(simData0,i+2,cols[i] => data0[:,i]);
    end
    SNR = var(data0) / Sigma[1,1]
    return simData, simData0, SNR;
end

"""
    prepData2Fit(rawData,idL2,endList,exgList)

Prepare simulated\real data to be fit by LARMEx.

It is assumed that data has been acquired by ecological momentary
assessment where respondent are observed multiple times in the course
of several days or weeks.
## Arguments:
- `rawData`: simulated or real data as a dataframe
- `idL2`: column name for level II units, e.g., an id for each day,
          `"idL2"` here
- `endList`: list of temporelly connected symptoms, `["S1","S2"]` here
- `exgList`: list of exogenous factors together with the constant terms,
```

```
            `["E","C"]` here
"""
function prepData2Fit(rawData,idL2,endList,exgList)
    nVar = length(endList)
    nExg = length(exgList)
    colS = map(string,repeat(endList,inner=nVar),repeat(1:nVar,nVar));
    col  = ["idL2";"tL1";"S";colS]
    if "E" in exgList
        col = vcat(col,map(string,repeat(["E"],nVar),1:nVar))
    end
    if "C" in exgList
        col = vcat(col,map(string,repeat(["C"],nVar),1:nVar))
    end
    D = reshape(Float64[],0,3+nVar^2+nVar*nExg)
    l2ID = unique(rawData.idL2);
    for sj in l2ID
        d1 = rawData[rawData.idL2.==sj,:];
        nObs = size(d1,1);
        iD = repeat(Matrix(d1[2:end,1:2]),nVar);
        dS = Matrix(d1[2:nObs,endList]);
        S = reshape(dS,nVar*(nObs-1),1);
        dL = kron(Matrix(1I,nVar,nVar), Matrix(d1[1:(nObs-1),endList]));
        cat = hcat(iD,S,dL)
        if "E" in exgList
            dE = kron(Matrix(1.0I,nVar,nVar),Matrix(d1[2:nObs,["E"]]));
            cat = hcat(cat,dE)
        end
        if "C" in exgList
            C = repeat([1],nObs-1);
            dC = kron(Matrix(1I,nVar,nVar),C);
            cat = hcat(cat,dC);
        end
        D = vcat(D,cat)
    end
    fitData = DataFrame(D,col);
    fitData[!,1:2] = convert.(Int16,fitData[:,1:2]);
    fitData[!,idL2] = categorical(fitData[:,idL2]);
    return fitData;
end

"""
    setFormula(fitData)

Generate a mixed-effects formula from a specifically formatted dataframe.
```

```
The input is supposed to have a special format where in a two-level
longitudinal data the level II identifiers are in the first and the
stacked observation in the third columns. The columns `4:end` are
supposed to represent the network structure and constant terms.
## Example:
`names(fitData)`:
11-element Vector{String}: ["idL2","tL1","S","S11","S12","S21","S22",
                            "E1","E2","C1","C2"]
"""
function setFormula(fitData)
    cols = names(fitData);
    idL2 = cols[1];
    colS  = cols[3];
    colRE = cols[4:end];
    indC = findall(x -> occursin("C",x),colRE);
    colFE = copy(colRE);
    deleteat!(colFE,indC);
    frm = string(colS," ~ 0 +",join(colFE,'+')," + (0+",join(colRE,'+'),
                 "|",idL2,")")
    return @eval(@formula($(Meta.parse(frm))));
end

"""
    re2csv(P,n,endList,exgList,fName)

Save the random-effects from a `parLARMEx` instance as a CSV file.
"""
function re2csv(P,n,endList,exgList,fName)
    nVar = P.nVar
    col = map(string,repeat(endList,inner=nVar),repeat(1:nVar,nVar));
    D = P.b_ar[1:n,:]
    if "E" in exgList
        col = vcat(col,map(string,repeat(["E"],nVar),1:nVar))
        D = hcat(D,P.b_e[1:n,:])
    end
    if "C" in exgList
        col = vcat(col,map(string,repeat(["C"],nVar),1:nVar))
        D = hcat(D,P.b_c[1:n,:])
    end
    re = DataFrame(D,col)
    CSV.write(fName,re)
end

"""
    fe2csv(P,n,endList,exgList,fName)
```

```
Save the fixed-effects from a `parLARMEx` instance as a CSV file.
"""
function fe2csv(P,endList,exgList,fName)
    nVar = P.nVar
    col = map(string,repeat(endList,inner=nVar),repeat(1:nVar,nVar));
    D = reshape(P.B_AR',(1,nVar^2))
    if "E" in exgList
        col = vcat(col,map(string,repeat(["E"],nVar),1:nVar))
        D = hcat(D,reshape(P.B_E,(1,2)))
    end
    if "C" in exgList
        col = vcat(col,map(string,repeat(["C"],nVar),1:nVar))
        D = hcat(D,[0 0])
    end
    if length(D) == length(P.b_var)
        D = vcat(D,reshape(P.b_var,(1,8)))
    else
        bv = hcat(P.b_var[1]*ones(nVar^2),repeat(P.b_var[2:end],inner=nVar))
        D = vcat(D,reshape(bv,(1,8)))
    end
    fe = DataFrame(D,col)
    CSV.write(fName,fe)
end
```