

Politecnico of Milan

School of Industrial and Information Engineering
Master of Science in Mathematical Engineering

MASTER THESIS

The DRPM Strikes Back: Improvements on a Bayesian Spatio-Temporal Clustering Model

Advisor

Prof. Alessandra Guglielmi

Coadvisor

Prof. Alessandro Carminati

Candidate

Federico Angelo Mor

Matr. 221429

*to my cats Otto
and La Micia*

Abstract

Sommario

Contents

1	Introduction	1
2	Description of the model	3
2.1	Update rules derivation	4
3	Implementation and MCMC algorithm	9
3.1	Optimizing spatial cohesions	11
3.2	Optimizing covariates similarities	13
4	Testing	17
4.1	Assessing the equivalence of the models	17
4.1.1	Target variable only	17
4.1.2	Target variable plus space	21
4.2	Performance with missing values	21
4.2.1	Target variable only	21
4.2.2	Target variable plus space	23
4.3	Effects of the covariates	23
4.4	Scaling performances	23
5	Conclusion	27
A	Theoretical details	29
A.1	Extended computations of the full conditionals	29
B	Computational details	37
B.1	Fitting algorithm code	37
B.2	Interface	51
C	Further plots	55
	Bibliography	57

Chapter 1

Introduction

Chapter 2

Description of the model

“Come on, gentlemen, why shouldn’t we get rid of all this calm reasonableness with one kick, just so as to send all these logarithms to the devil and be able to live our own lives at our own sweet will?”
— Fëdor Dostoevskij, *Notes from the Underground*

$$\begin{aligned}
Y_{it}|Y_{it-1}, \boldsymbol{\mu}_t^*, \boldsymbol{\sigma}_t^{2*}, \boldsymbol{\eta}, \mathbf{c}_t &\stackrel{\text{ind}}{\sim} \mathcal{N}(\mu_{c_{it}}^* + \eta_{1i}Y_{it-1}, \sigma_{c_{it}}^{2*}(1 - \eta_{1i}^2)) \\
Y_{i1} &\stackrel{\text{ind}}{\sim} \mathcal{N}(\mu_{c_{i1}}^*, \sigma_{c_{i1}}^{2*}) \\
\xi_i = \text{Logit}(\tfrac{1}{2}(\eta_{1i} + 1)) &\stackrel{\text{ind}}{\sim} \text{Laplace}(a, b) \\
(\mu_{jt}^*, \sigma_{jt}^{2*}) &\stackrel{\text{ind}}{\sim} \mathcal{N}(\vartheta_t, \tau_t^2) \times \mathcal{U}(0, A_\sigma) \\
\vartheta_t|\vartheta_{t-1} &\stackrel{\text{ind}}{\sim} \mathcal{N}((1 - \varphi_1)\varphi_0 + \varphi_1\vartheta_{t-1}, \lambda^2(1 - \varphi_1^2)) \\
(\vartheta_1, \tau_t) &\stackrel{\text{iid}}{\sim} \mathcal{N}(\varphi_0, \lambda^2) \times \mathcal{U}(0, A_\tau) \\
(\varphi_0, \varphi_1, \lambda) &\sim \mathcal{N}(m_0, s_0^2) \times \mathcal{U}(-1, 1) \times \mathcal{U}(0, A_\lambda) \\
\{\mathbf{c}_t, \dots, \mathbf{c}_T\} &\sim \text{tRPM}(\boldsymbol{\alpha}, M) \text{ with } \alpha_t \stackrel{\text{iid}}{\sim} \text{Beta}(a_\alpha, b_\alpha) \quad (2.1)
\end{aligned}$$

$$\begin{aligned}
Y_{it}|Y_{it-1}, \boldsymbol{\mu}_t^*, \boldsymbol{\sigma}_t^{2*}, \boldsymbol{\eta}, \mathbf{c}_t &\stackrel{\text{ind}}{\sim} \mathcal{N}(\mu_{c_{it}}^* + \eta_{1i}Y_{it-1} + \mathbf{x}_{it}^T \boldsymbol{\beta}_t, \sigma_{c_{it}}^{2*}(1 - \eta_{1i}^2)) \\
Y_{i1} &\stackrel{\text{ind}}{\sim} \mathcal{N}(\mu_{c_{i1}}^* + \mathbf{x}_{i1}^T \boldsymbol{\beta}_1, \sigma_{c_{i1}}^{2*}) \\
\boldsymbol{\beta}_t &\stackrel{\text{ind}}{\sim} \mathcal{N}_p(\mathbf{b}, s^2 I) \\
\xi_i = \text{Logit}(\tfrac{1}{2}(\eta_{1i} + 1)) &\stackrel{\text{ind}}{\sim} \text{Laplace}(a, b) \\
(\mu_{jt}^*, \sigma_{jt}^{2*}) &\stackrel{\text{ind}}{\sim} \mathcal{N}(\vartheta_t, \tau_t^2) \times \text{invGamma}(a_\sigma, b_\sigma) \\
\vartheta_t|\vartheta_{t-1} &\stackrel{\text{ind}}{\sim} \mathcal{N}((1 - \varphi_1)\varphi_0 + \varphi_1\vartheta_{t-1}, \lambda^2(1 - \varphi_1^2)) \\
(\vartheta_1, \tau_t^2) &\stackrel{\text{iid}}{\sim} \mathcal{N}(\varphi_0, \lambda^2) \times \text{invGamma}(a_\tau, b_\tau) \\
(\varphi_0, \varphi_1, \lambda^2) &\sim \mathcal{N}(m_0, s_0^2) \times \mathcal{U}(-1, 1) \times \text{invGamma}(a_\lambda, b_\lambda) \\
\{\mathbf{c}_t, \dots, \mathbf{c}_T\} &\sim \text{tRPM}(\boldsymbol{\alpha}, M) \text{ with } \alpha_t \stackrel{\text{iid}}{\sim} \text{Beta}(a_\alpha, b_\alpha) \quad (2.2)
\end{aligned}$$

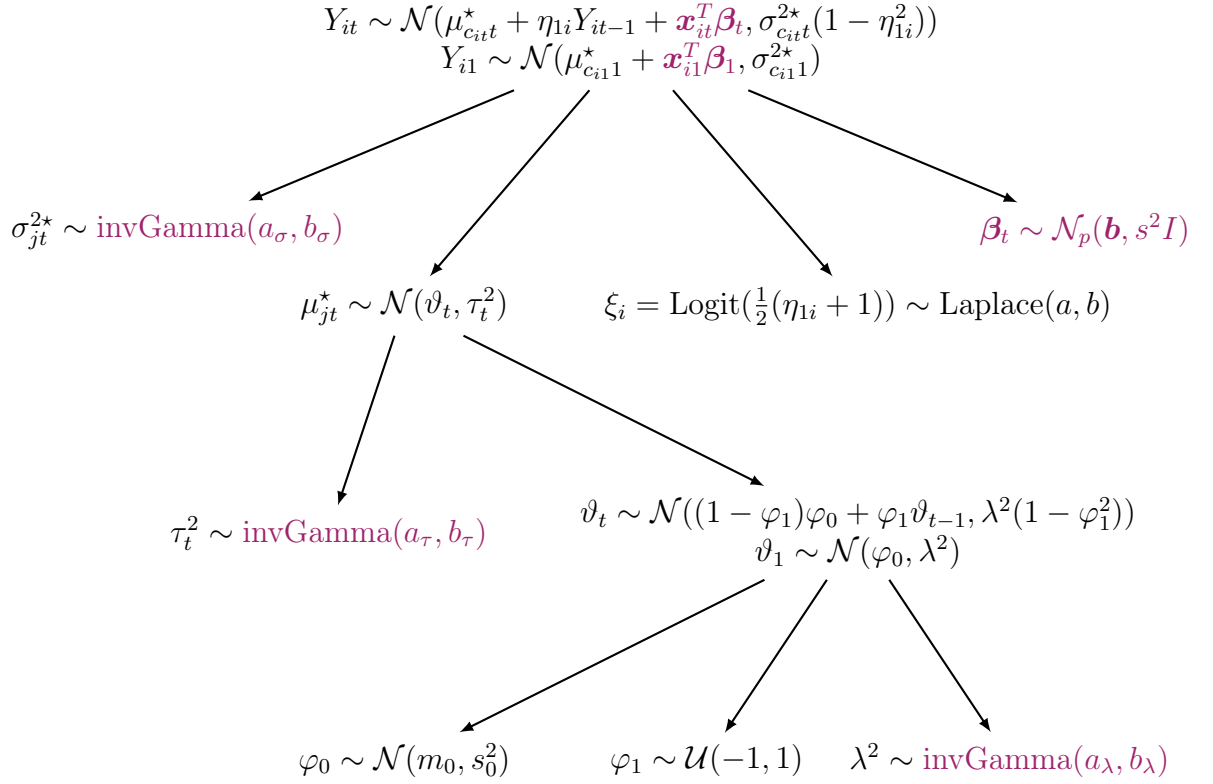


Figure 2.1: Graph visualization of the DRPM model, with highlighted in dark red the changes that we made to the original formulation.

2.1 Update rules derivation

We report here the full conditionals derivation for the parameters which had a conjugacy in the model (for the full computations see Appendix A). The other ones not included here involved instead the classical Metropolis update.

- update σ_{jt}^{2*}

for $t = 1$: $f(\sigma_{jt}^{2*} | -) \propto$ kernel of a $\text{invGamma}(a_{\sigma(\text{post})}, b_{\sigma(\text{post})})$ with

$$a_{\tau(\text{post})} = a_\sigma + \frac{|S_{jt}|}{2} \quad b_{\tau(\text{post})} = b_\sigma + \frac{1}{2} \sum_{i \in S_{jt}} (Y_{it} - \mu_{jt}^* - \mathbf{x}_{it}^T \boldsymbol{\beta}_t)^2$$

for $t > 1$: $f(\sigma_{jt}^{2*} | -) \propto$ kernel of a $\text{invGamma}(a_{\sigma(\text{post})}, b_{\sigma(\text{post})})$ with

$$a_{\tau(\text{post})} = a_\sigma + \frac{|S_{jt}|}{2} \quad b_{\tau(\text{post})} = b_\sigma + \frac{1}{2} \sum_{i \in S_{jt}} (Y_{it} - \mu_{jt}^* - \eta_{1i} Y_{it-1} - \mathbf{x}_{it}^T \boldsymbol{\beta}_t)^2 \quad (2.3)$$

- update μ_{jt}^*

for $t = 1$: $f(\mu_{jt}^* | -) \propto$ kernel of a $\mathcal{N}(\mu_{\mu_{jt}^*}^*(\text{post}), \sigma_{\mu_{jt}^*}^2(\text{post}))$ with

$$\sigma_{\mu_{jt}^*}^2(\text{post}) = \frac{1}{\frac{1}{\tau_t^2} + \frac{|S_{jt}|}{\sigma_{jt}^{2*}}} \quad \mu_{\mu_{jt}^*}^*(\text{post}) = \sigma_{\mu_{jt}^*}^2(\text{post}) \left(\frac{\vartheta_t}{\tau_t^2} + \frac{\sum_{i \in S_{jt}} Y_{i1}}{\sigma_{jt}^{2*}} \right)$$

for $t > 1$: $f(\mu_{jt}^*|-) \propto \text{kernel of a } \mathcal{N}(\mu_{\mu_{jt}^*}(\text{post}), \sigma_{\mu_{jt}^*}^2(\text{post}))$ with

$$\sigma_{\mu_{jt}^*}^2(\text{post}) = \frac{1}{\frac{1}{\tau_t^2} + \frac{\sum_{i \in S_{jt}} \frac{1}{1-\eta_{1i}^2}}{\sigma_{jt}^{2*}}} \quad \mu_{\mu_{jt}^*}(\text{post}) = \sigma_{\mu_{jt}^*}^2(\text{post}) \left(\frac{\vartheta_t}{\tau_t^2} + \frac{\sum_{i \in S_{jt}} \frac{Y_{it} - \eta_{1i} Y_{i,t-1}}{1-\eta_{1i}^2}}{\sigma_{jt}^{2*}} \right) \quad (2.4)$$

- update β_t

for $t = 1$: $f(\beta_t|-) \propto \text{kernel of a } \mathcal{N}(\mathbf{b}_{(\text{post})}, A_{(\text{post})})$ with

$$A_{(\text{post})} = \left(\frac{1}{s^2} I + \sum_{i=1}^n \frac{\mathbf{x}_{it} \mathbf{x}_{it}^T}{\sigma_{c_{it}}^{2*}} \right)^{-1} \quad \mathbf{b}_{(\text{post})} = A_{(\text{post})} \left(\frac{\mathbf{b}}{s^2} + \sum_{i=1}^n \frac{(Y_{it} - \mu_{c_{it}}^*) \mathbf{x}_{it}}{\sigma_{c_{it}}^{2*}} \right)$$

i.e. $f(\beta_t|-) \propto \text{kernel of a } \mathcal{N}\text{Canon}(\mathbf{h}_{(\text{post})}, J_{(\text{post})})$ with

$$\mathbf{h}_{(\text{post})} = \left(\frac{\mathbf{b}}{s^2} + \sum_{i=1}^n \frac{(Y_{it} - \mu_{c_{it}}^*) \mathbf{x}_{it}}{\sigma_{c_{it}}^{2*}} \right) \quad J_{(\text{post})} = \left(\frac{1}{s^2} I + \sum_{i=1}^n \frac{\mathbf{x}_{it} \mathbf{x}_{it}^T}{\sigma_{c_{it}}^{2*}} \right)$$

for $t > 1$: $f(\beta_t|-) \propto \text{kernel of a } \mathcal{N}(\mathbf{b}_{(\text{post})}, A_{(\text{post})})$ with

$$A_{(\text{post})} = \left(\frac{1}{s^2} I + \sum_{i=1}^n \frac{\mathbf{x}_{it} \mathbf{x}_{it}^T}{\sigma_{c_{it}}^{2*}} \right)^{-1} \quad \mathbf{b}_{(\text{post})} = A_{(\text{post})} \left(\frac{\mathbf{b}}{s^2} + \sum_{i=1}^n \frac{(Y_{it} - \mu_{c_{it}}^* - \eta_{1i} Y_{i,t-1}) \mathbf{x}_{it}}{\sigma_{c_{it}}^{2*}} \right)$$

i.e. $f(\beta_t|-) \propto \text{kernel of a } \mathcal{N}\text{Canon}(\mathbf{h}_{(\text{post})}, J_{(\text{post})})$ with

$$\mathbf{h}_{(\text{post})} = \left(\frac{\mathbf{b}}{s^2} + \sum_{i=1}^n \frac{(Y_{it} - \mu_{c_{it}}^* - \eta_{1i} Y_{i,t-1}) \mathbf{x}_{it}}{\sigma_{c_{it}}^{2*}} \right) \quad J_{(\text{post})} = \left(\frac{1}{s^2} I + \sum_{i=1}^n \frac{\mathbf{x}_{it} \mathbf{x}_{it}^T}{\sigma_{c_{it}}^{2*}} \right) \quad (2.5)$$

Here $\mathcal{N}\text{Canon}(\mathbf{h}, J)$ is the canonical formulation of the $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$, with $\mathbf{h} = \Sigma^{-1} \boldsymbol{\mu}$ and $J = \Sigma^{-1}$. This other distribution facilitates the sampling, since these full conditional computations allow to derive directly the parameters of the canonical one, e.g. the inverse of the variance matrix, rather than the variance matrix itself; and therefore sampling through it does not require any inversion of matrices which would produce more computational load, numerical instabilities, and loss of accuracy. So in Julia we can write `rand(MvNormalCanon(h_star, J_star))`, rather than the riskier one `rand(MvNormal(inv(J_star)*h_star, inv(J_star)))`; which a part from the previously mentioned disadvantages would be a statistically equivalent form.

- update τ_t^2

$f(\tau_t^2|-) \propto \text{kernel of a } \text{invGamma}(a_{\tau(\text{post})}, b_{\tau(\text{post})})$ with

$$a_{\tau(\text{post})} = \frac{k_t}{2} + a_{\tau} \quad b_{\tau(\text{post})} = \frac{\sum_{j=1}^{k_t} (\mu_{jt}^* - \vartheta_t)^2}{2} + b_{\tau} \quad (2.6)$$

- update ϑ_t

for $t = T$: $f(\vartheta_t|-) \propto \text{kernel of a } \mathcal{N}(\mu_{\vartheta_t}(\text{post}), \sigma_{\vartheta_t}^2(\text{post}))$ with

$$\begin{aligned}
\sigma_{\vartheta_t(\text{post})}^2 &= \frac{1}{\frac{1}{\lambda^2(1-\varphi_1^2)} + \frac{k_t}{\tau_t^2}} \\
\mu_{\vartheta_t(\text{post})} &= \sigma_{\vartheta_t(\text{post})}^2 \left(\frac{\sum_{j=1}^{k_t} \mu_{jt}^*}{\tau_t^2} + \frac{(1-\varphi_1)\varphi_0 + \varphi_1\vartheta_{t-1}}{\lambda^2(1-\varphi_1^2)} \right) \\
\text{for } 1 < t < T: f(\vartheta_t|-) &\propto \text{kernel of a } \mathcal{N}(\mu_{\vartheta_t(\text{post})}, \sigma_{\vartheta_t(\text{post})}^2) \text{ with} \\
\sigma_{\vartheta_t(\text{post})}^2 &= \frac{1}{\frac{1+\varphi_1^2}{\lambda^2(1-\varphi_1^2)} + \frac{k_t}{\tau_t^2}} \\
\mu_{\vartheta_t(\text{post})} &= \sigma_{\vartheta_t(\text{post})}^2 \left(\frac{\sum_{j=1}^{k_t} \mu_{jt}^*}{\tau_t^2} + \frac{\varphi_1(\vartheta_{t-1} + \vartheta_{t+1}) + \varphi_0(1-\varphi_1)^2}{\lambda^2(1-\varphi_1^2)} \right) \\
\text{for } t = 1: f(\vartheta_t|-) &\propto \text{kernel of a } \mathcal{N}(\mu_{\vartheta_t(\text{post})}, \sigma_{\vartheta_t(\text{post})}^2) \text{ with} \\
\sigma_{\vartheta_t(\text{post})}^2 &= \frac{1}{\frac{1}{\lambda^2} + \frac{\varphi_1^2}{\lambda^2(1-\varphi_1^2)} + \frac{k_t}{\tau_t^2}} \\
\mu_{\vartheta_t(\text{post})} &= \sigma_{\vartheta_t(\text{post})}^2 \left(\frac{\varphi_0}{\lambda^2} + \frac{\varphi_1(\vartheta_{t+1} - (1-\varphi_1)\varphi_0)}{\lambda^2(1-\varphi_1^2)} + \frac{\sum_{j=1}^{k_t} \mu_{jt}^*}{\tau_t^2} \right) \quad (2.7)
\end{aligned}$$

- update φ_0

$$\begin{aligned}
f(\varphi_0|-) &\propto \text{kernel of a } \mathcal{N}(\mu_{\varphi_0(\text{post})}, \sigma_{\varphi_0(\text{post})}^2) \text{ with} \\
\sigma_{\varphi_0(\text{post})}^2 &= \frac{1}{\frac{1}{s_0^2} + \frac{1}{\lambda^2} + \frac{(T-1)(1-\varphi_1)^2}{\lambda^2(1-\varphi_1^2)}} \\
\mu_{\varphi_0(\text{post})} &= \sigma_{\varphi_0(\text{post})}^2 \left(\frac{m_0}{s_0^2} + \frac{\vartheta_1}{\lambda^2} + \frac{1-\varphi_1}{\lambda^2(1-\varphi_1^2)} \sum_{t=2}^T (\vartheta_t - \varphi_1\vartheta_{t-1}) \right) \quad (2.8)
\end{aligned}$$

- update λ^2

$$\begin{aligned}
f(\lambda^2|-) &\propto \text{kernel of a invGamma}(a_{\lambda(\text{post})}, b_{\lambda(\text{post})}) \text{ with} \\
a_{\lambda(\text{post})} &= \frac{T}{2} + a_\lambda \\
b_{\lambda(\text{post})} &= \frac{(\vartheta_1 - \varphi_0)^2}{2} + \sum_{t=2}^T \frac{(\vartheta_t - (1-\varphi_1)\varphi_0 - \varphi_1\vartheta_{t-1})^2}{2} + b_\lambda \quad (2.9)
\end{aligned}$$

- update a missing observation Y_{it}

$$\begin{aligned}
\text{for } t = 1: f(Y_{it}|-) &\propto \text{kernel of a } \mathcal{N}(\mu_{Y_{it}(\text{post})}, \sigma_{Y_{it}(\text{post})}^2) \text{ with} \\
\sigma_{Y_{it}(\text{post})}^2 &= \frac{1}{\frac{1}{\sigma_{c_{it}t}^{2*}} + \frac{\eta_{1i}^2}{2\sigma_{c_{it+1}t+1}^{2*}(1-\eta_{1i}^2)}} \\
\mu_{Y_{it}(\text{post})} &= \sigma_{Y_{it}(\text{post})}^2 \left(\frac{\mu_{c_{it}t}^* + \mathbf{x}_{it}^T \boldsymbol{\beta}_t}{\sigma_{c_{it}t}^{2*}} + \frac{\eta_{1i}(Y_{it+1} - \mu_{c_{it+1}t+1}^* - \mathbf{x}_{it+1}^T \boldsymbol{\beta}_{t+1})}{\sigma_{c_{it+1}t+1}^{2*}(1-\eta_{1i}^2)} \right) \\
\text{for } 1 < t < T: f(Y_{it}|-) &\propto \text{kernel of a } \mathcal{N}(\mu_{Y_{it}(\text{post})}, \sigma_{Y_{it}(\text{post})}^2) \text{ with}
\end{aligned}$$

$$\sigma_{Y_{it}(\text{post})}^2 = \frac{1 - \eta_{1i}^2}{\frac{1}{\sigma_{c_{it}t}^{2\star}} + \frac{\eta_{1i}^2}{\sigma_{c_{it+1}t+1}^{2\star}}}$$

$$\mu_{Y_{it}(\text{post})} = \sigma_{Y_{it}(\text{post})}^2 \left(\frac{\mu_{c_{it}t}^{\star} + \eta_{1i} Y_{it-1} + \mathbf{x}_{it}^T \boldsymbol{\beta}_t}{\sigma_{c_{it}t}^{2\star} (1 - \eta_{1i}^2)} + \frac{\eta_{1i} (Y_{it+1} - \mu_{c_{it+1}t+1}^{\star} - \mathbf{x}_{it+1}^T \boldsymbol{\beta}_{t+1})}{\sigma_{c_{it+1}t+1}^{2\star} (1 - \eta_{1i}^2)} \right)$$

for $t = T$: $f(Y_{it}|-)$ is just the likelihood of Y_{it} (2.10)

Chapter 3

Implementation and MCMC algorithm

To implement our updated model, and to do it in a performance-oriented way, we decided to opt for the Julia language [Bez+17]. Julia is a etc.

As we will see in Section 4, even despite the higher complexity of the model we managed to get better performance in Julia compared to the original C implementation. This at the cost of a smaller increase in memory requirements.

In fact, one major issue encountered during the earlier stages of designing the fitting function in Julia was controlling the amount of memory and allocations that some functions, structures, or algorithms would require. At the beginning of the development and testing, where the correctness of the algorithm was the only priority, we saw that most of the execution time was actually spent by the garbage collector of Julia, which had the burden of tracking all the allocated memory and reclaiming the unused one in order to make it available again for new computations.

Together with avoiding useless allocations, or in general managing more finely the memory, another important aspect to focus on to improve performances is ensuring type stability of the function. Luckily, Julia disposes of several tools to inspect and address both problems.

Regarding the first point, there are packages such as `Cthulhu`, or even the simple `@code_warntype` macro, which allow to ensure that a function is type-stable, i.e. that the types of all variables can be correctly predicted by the compiler and stay consistent throughout the execution. Type stability is crucial for performance as it enables the compiler to generate optimized machine code, removing the load derived from dynamic type checks. In fact, Julia is dynamically typed, which means that variables do not have to be necessarily declared with their type, in contrast to fully statically typed languages such as C, and they could change it during execution. For example a variable initialized as an integer could later become a float, or even a string. However, for performance purposes, these dynamicisms should be avoided, and those tools help to ensure that this happens.

Regarding allocations, we relied on profiling suites such as `ProfileCanvas`. This profiler generates a plot, the *flame graph* of Figure 3.1, which depicts the different

sections of code with regions whose size is proportional to a certain metric, e.g. the time spent on them during execution or the number (or size) of allocations that was required. This allows to see if the running time is spent on actual useful operations or instead on “bad” ones, such as garbage collection; highlighting therefore the sections of code which could possibly be optimized. All the modelling variables were of course preallocated, but the key has been to refactor the code to make it work more *in place*, passing directly as arguments the variables which would be modified the function, and modify them inside the function, rather than returning some values and then use them on the initial variables.

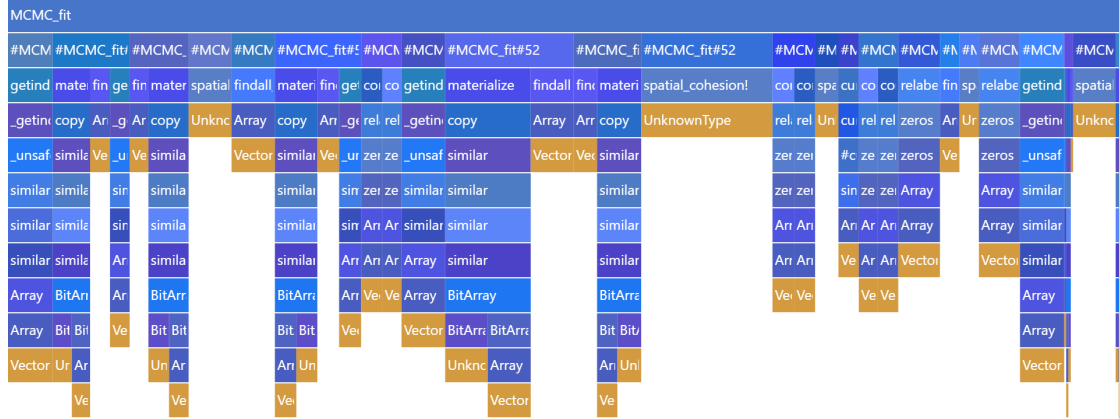


Figure 3.1: Flame graph derived from a simple test fit with $n = 20$, $T = 50$, ran for 10000 iterates. Each section, identified with a colored box, represents a function call with its stack trace below, i.e. all the other subsequent function calls generated from the initial top one. The widths are proportional to allocation counts, i.e. to how many times a memory allocation was required for their corresponding box. The plot should be read from top to bottom regarding function calls, and from left to right regarding the fitting steps.

More in general, **BenchmarkTools** [CR16] had been also a great ally to quickly analyse and compare whole fits or just small sections of code and simple instructions. For example, we can test different versions of equivalent instructions to see which could be the most efficient, as in this case

```
using BenchmarkTools
nh_tmp = rand(100)
@btime nclus_temp = sum($nh_tmp .> 0)
# 168.956 ns (2 allocations: 112 bytes)
@btime nclus_temp = count(x->(x>0), $nh_tmp)
# 11.612 ns (0 allocations: 0 bytes)
```

or this other one

```
n = 100; rho_tmp = rand((1:5),n); k = 1
@btime findall(j -> ($rho_tmp)[j]==$k, 1:n) # with anonymous function
# 272.302 ns (5 allocations: 384 bytes)
@btime findall_faster(j -> ($rho_tmp)[j]==$k, 1:n) # custom implementation
# 214.259 ns (3 allocations: 960 bytes)
@btime findall($rho_tmp .== $k) # with element-wise comparison
# 184.112 ns (3 allocations: 320 bytes)
```

where the `$` symbol is used for interpolate the variable, ensuring to treat it as a local variable inside the benchmark and therefore avoiding any bias caused by accessing a global variable. This was a quick and simple comparison, yet very precise and effective, which would be instead more complex to replicate in C.

During the finer and final refinements of the code we even used the surgical `--track-allocation` option, which asks Julia to execute the code and annotate the source file to see at which lines (and of which amount) occur allocations.

3.1 Optimizing spatial cohesions

The memory allocation problem was especially seen in the spatial cohesion computation. In fact, such computation appears twice in both sections of updating γ and updating ρ , which are inside of outer loops on draws, time, and units, and involve themselves some other loops on clusters. All this implied that those cohesion computations will be executed possibly millions of times during each fit. A simple and quick inspection suggests a value between $\Omega(dTn)$ and $O(dTn^2)$, being d the number of iterations, T the time horizon and n the number of units. We can't provide a Θ estimate due to the variability and randomness of the inside loops, which depend on the distribution of the clusters. With all that in mind, it was crucial to optimize the performance of the cohesion functions.

Function calls in Julia are very cheap, and so the only optimizing task consisted in carefully designing the cohesion function implementations, rather than maybe rethink the whole algorithm. Cohesions 1, 2, 5 and 6 didn't exhibit any complication or need for optimization. The natural conversion in code from their mathematical models proved to be already optimally performing. The main problem was instead with cohesions 3 and 4, the auxiliary and double dippery, which by nature would involve some linear algebra computations with vectors and matrices.

The first implementation of those cohesions turned out to be really slow, due to the overhead generated by allocating and then freeing, at each call, the memory devoted to all vectors and matrices. This in the end meant that most of the execution time was actually spent by the garbage collector, rather than in the real and useful operations. A first solution was then to resort to a scalar implementation, which would remove the overhead of the more complex memory structures, but in the end we managed to combine the readability of the first idea with the performance of the second one into a final version, as we can see from Listing 1.

Listing 1: Sections of the Julia code for the three implementation cases of the spatial cohesions 3 and 4. The first one has the classical vector computations derived from the mathematical formulation, the second one is the conversion to only have scalar variables, while the third one is the final version, keeping the vector form but improved using static structures.

```
# original vector version
sbar = [mean(s1), mean(s2)]
vtmp = sbar - mu_0
Mtmp = vtmp * vtmp'
Psi_n = Psi + S + (k0*sdim) / (k0+sdim) * Mtmp
```

```

# scalar-only version
sbar1 = mean(s1); sbar2 = mean(s2)
vtmp_1 = sbar1 - mu_0[1]
vtmp_2 = sbar2 - mu_0[2]
Mtmp_1 = vtmp_1^2
Mtmp_2 = vtmp_1 * vtmp_2
Mtmp_3 = copy(Mtmp_2)
Mtmp_4 = vtmp_2^2
aux1 = k0 * sdim; aux2 = k0 + sdim
Psi_n_1 = Psi[1] + S1 + aux1 / (aux2) * Mtmp_1
Psi_n_2 = Psi[2] + S2 + aux1 / (aux2) * Mtmp_2
Psi_n_3 = Psi[3] + S3 + aux1 / (aux2) * Mtmp_3
Psi_n_4 = Psi[4] + S4 + aux1 / (aux2) * Mtmp_4

# static improved version
sbar1 = mean(s1); sbar2 = mean(s2)
sbar = SVector((sbar1, sbar2))
vtmp = sbar .- mu_0
Mtmp = vtmp * vtmp'
aux1 = k0 * sdim; aux2 = k0 + sdim
Psi_n = Psi .+ S .+ aux1 / (aux2) .* Mtmp

```

This final version exploits the `StaticArrays` package of Julia, which allows to use vectors and matrices more efficiently if their size is known at compile time; which is the case of the spatial cohesions computation, since working with planar spatial coordinates we will always have 2×1 vectors and 2×2 matrices. The benefits of this final version are that we maintain the natural mathematical form of the first one, improving the clearness of the code, together with the efficiency of the second one, since now with static structures the compiler is able to optimize all memory allocations as it was doing with the simple scalar variables.

Figure 3.2 shows the comparison of their performances, where we can see how the scalar and static versions indeed perform very similarly.

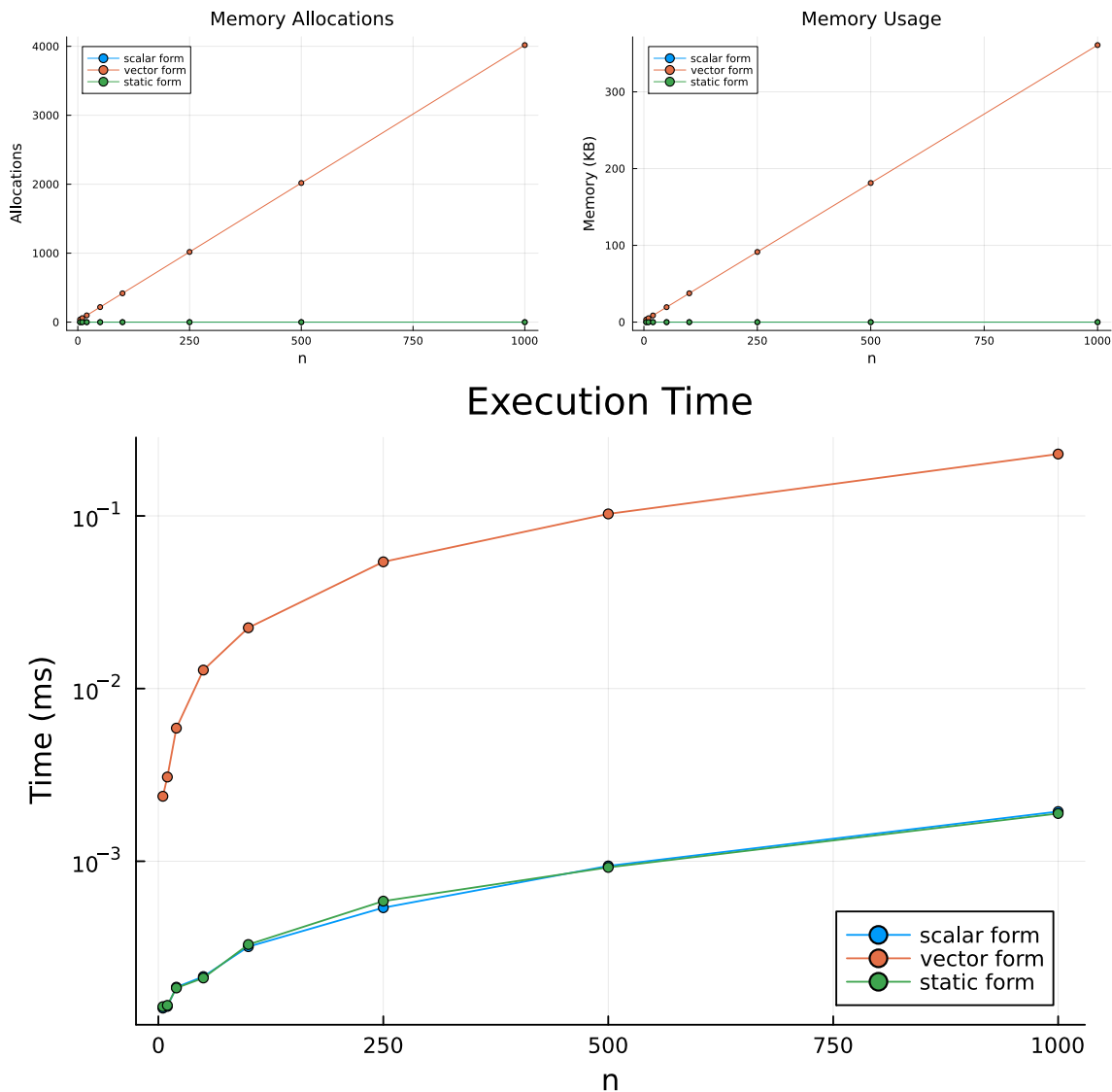


Figure 3.2: Performance comparison between the three versions of the cohesion 4 function. Tests ran through the `BenchmarkTools` package of Julia by randomly generating the spatial coordinates of the various test sizes n , with similar results standing for cohesion 3. The memory allocation and usage plots are constant at zero for both the scalar and vector static cases.

Noticeably, the C implementation of the model didn't have to worry about all this reasoning, since C can't natively, nor gracefully, work with vectors and matrices.

3.2 Optimizing covariates similarities

Another problem was how to speed up the computation of the similarity functions, since those would also be called the possibly millions of times as the cohesion ones, considering also that we could incorporate more than one covariate into the clustering process, so there would be an additional loop based on p , the number of covariates decided to be included.

As in the previous case, some of the functions didn't show any special need or room for relevant optimizations. The fourth one instead, the auxiliary similarity function, was essential to be optimized; not only being one the most frequent choice among all the similarities, but also because it involves a computationally heavy sum of the squares of the covariate values, as we can see in Listing 2.

Listing 2: Final version of the similarity 4 function. The performance analysis will focus just on that inside loop, since the rest is not negotiable.

```
function similarity4(X_jt::AbstractVector{<:Real}, mu_c::Real, lambda_c::Real,
↳ a_c::Real, b_c::Real; lg::Bool)
    n = length(X_jt)
    nm = n/2
    xbar = mean(X_jt)
    aux2 = 0.
    @inbounds @simd for i in eachindex(X_jt)
        aux2 += X_jt[i]^2
    end
    aux1 = b_c + 0.5 * (aux2 - (n*xbar + lambda_c*mu_c)^2/(n+lambda_c) +
↳ lambda_c*mu_c^2 )
    out = -nm*log2pi + 0.5*log(lambda_c/(lambda_c+n)) + lgamma(a_c+nm) -
↳ lgamma(a_c) + a_c*log(b_c) + (-a_c-nm)*log(aux1)
    return lg ? out : exp(out)
end
```

The idea to optimize it has been to annotate the loop with some macros provided by Julia. They are the following:

- `@inbounds` eliminates the array bounds checking within expressions. This allows to skip the checks, at the cost to guarantee, by code design, that no out-of-bounds or wrong accesses will occur in the code and therefore no undefined behaviour will take place. The above loop is indeed very simple and safe, so this assumption is clearly satisfied.
- `@fastmath` executes a transformed version of the expression, which calls functions that may violate strict IEEE semantics¹. For example, its use could make $(a + b) + c \neq a + (b + c)$, but just in very pathological cases. Again, this is not a problem in our case where we are computing $\sum X_i^2$, since it does not have an intrinsically “right order” in which it has to be done.
- `@simd` (single instruction multiple data) annotate a for loop to allow the compiler to take extra liberties to allow loop re-ordering. This is a sort of parallelism technique, but rather than distributing the computational load on more processors we just *vectorize* the loop, i.e. we enable the CPU to perform that single instruction (summing the square of the i th component to a reduction variable) on multiple data chunks at once, using vector registers, rather than working on each element of the vector individually.

¹Institute of Electrical and Electronics Engineers. The IEEE-754 standard defines floating-point formats, i.e. the ways to represent real numbers in hardware, and the expected behavior of arithmetic operations on them, including precision, rounding, and handling of special values (e.g. NaN (Not a Number) and infinity).

As we can see from Figure 3.3, the actual performance difference basically derives just from the use of `@simd`, with the other two annotations making not much of a difference. For that reason, and to reassure all pure mathematicians, we decided to remove the `@fastmath` annotation, leaving just `@inbounds` and `@simd`.

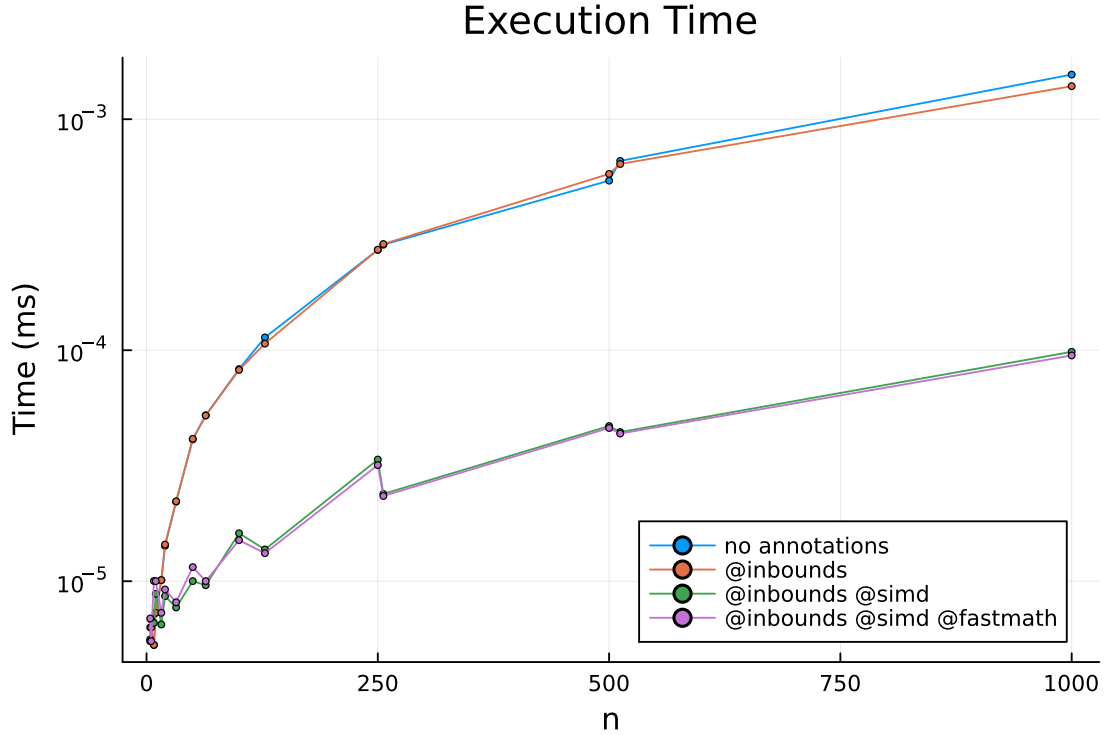


Figure 3.3: Comparison of the performances of the different possible loop annotations in the similarity 4 implementation. Their numerical output results are indeed the same for all cases. There are no memory allocation and usage plots since the analysis has been conducted only to evaluate the performances of the inside loop, which has no memory issues.

We can also notice interestingly how the tests with `@simd` annotation runs quicker in the case of n being a power of two compared to its closest rounded integer (e.g. 256 against 250 or 512 against 500), despite having some more data. This is a proof of the effectiveness of the SIMD paradigm: according to the different architectures, the CPUs can provide different register sizes (e.g. 64, 128, 256 or 512 bits) and therefore the data subdivision can fit perfectly in them when the total memory occupied by the elements is a multiple of that register size (i.e. the number of data values is a power of two). Otherwise there will be some “leftovers chunks”, which will of course be processed, but will also cause, as a consequence of the imperfect fit, a bit of overhead.

Chapter 4

Testing

4.1 Assessing the equivalence of the models

Our model, and the corresponding Julia code, is just an improvement of the original DRPM one with his relative C implementation. These improvements, as described in the previous chapters, refer to the insertion of covariates, both at the clustering and likelihood levels, the handling of missing values in the target variable, and the computational efficiency. In this sense, our updates are just add-ons to the original model, and therefore at a common testing level they should perform similarly by agreeing in the clusters that they produce on a given dataset.

To assess this ideally equivalent behaviour we ran two tests: the first one with only the target values, using generated data, while the second with also spatial information, using a real spatio-temporal dataset. From now on, for the sake of clarity, we will refer to CDRPM as the original model and implementation from [PQD22], while to JDRPM as the updated version derived from this thesis work.

4.1.1 Target variable only

For the first test we generated a dataset of $n = 10$ units and $T = 12$ time instants, and fitted both models collecting 1000 iterates derived from 10000 total iterations, discarding the first 5000 as burnin, and thinning by 5. Those parameters, as well as the generating function, were the same of [PQD22]. The generating function allowed to create data points with temporal dependence which could be tuned through some dedicated parameters.

So at this testing stage there was just the target values from Y_{it} to dictate the clusters definition, and both models managed to provide good fits, as we can read from Table 4.1. The JDRPM model had better fit metrics (lower WAIC and higher LPML) and also faster execution time. Regarding the fitted values, displayed in Figure 4.1 together with the original data, they turned out to be also more precise than the one generated by CDRPM, having lower MSEs.

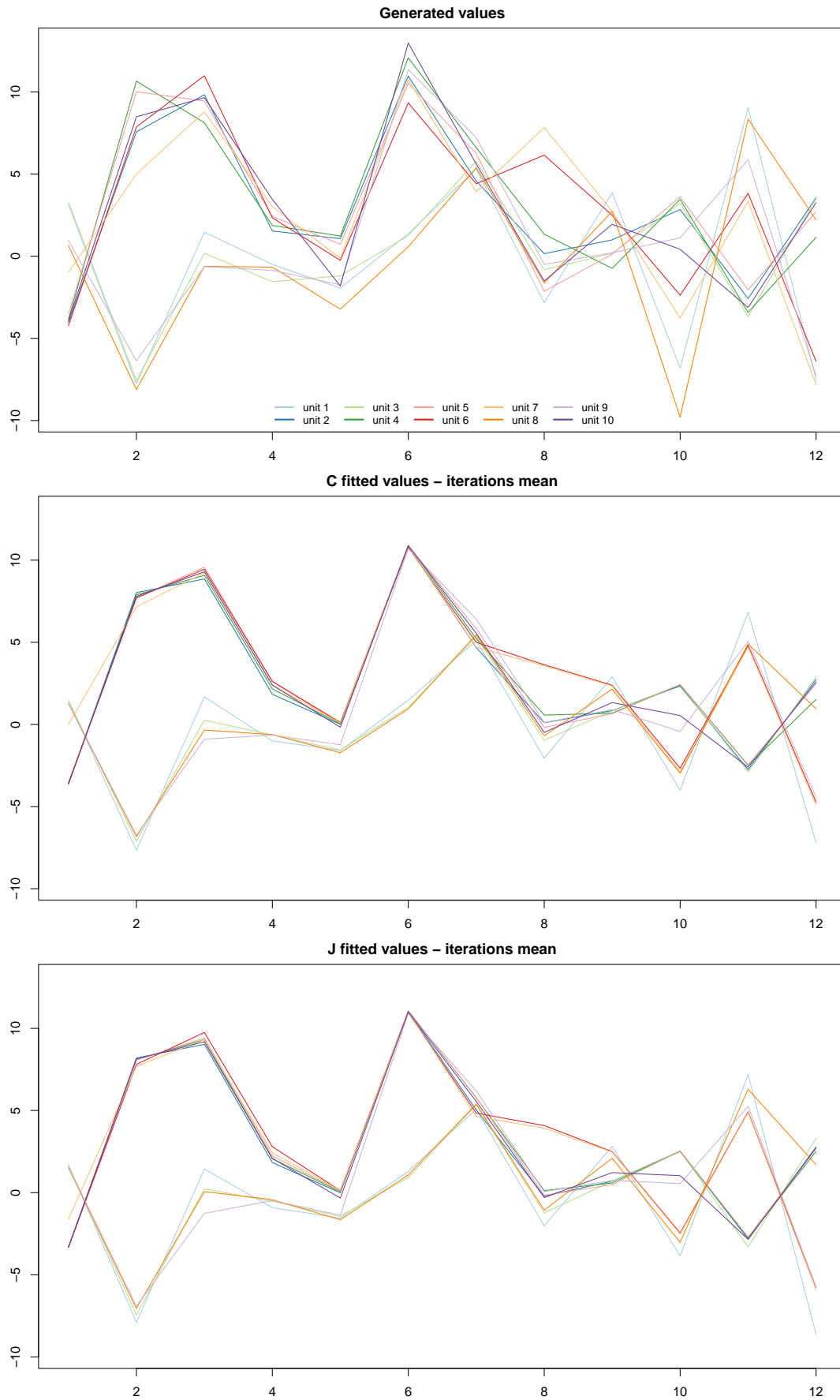


Figure 4.1: Generated target values (top), together with their fitted values estimate trough the mean of the 1000 iterates generated by model CDRPM (middle) and JDRPM (bottom).

Table 4.1: Summary of the comparison between the two fits. The MSE columns refer to the accuracy between the fitted values generated by the models (estimated by taking the mean and the median of the returned 1000 iterates) and the true values of the target variable. Higher LPML and lower WAIC indicate a better fit.

	MSE mean	MSE median	LPML	WAIC	execution time
CDRPM	1.673099	1.586079	-249.61	469.69	5.326 s
JDRPM	1.432986	1.401098	-229.20	420.02	2.568 s

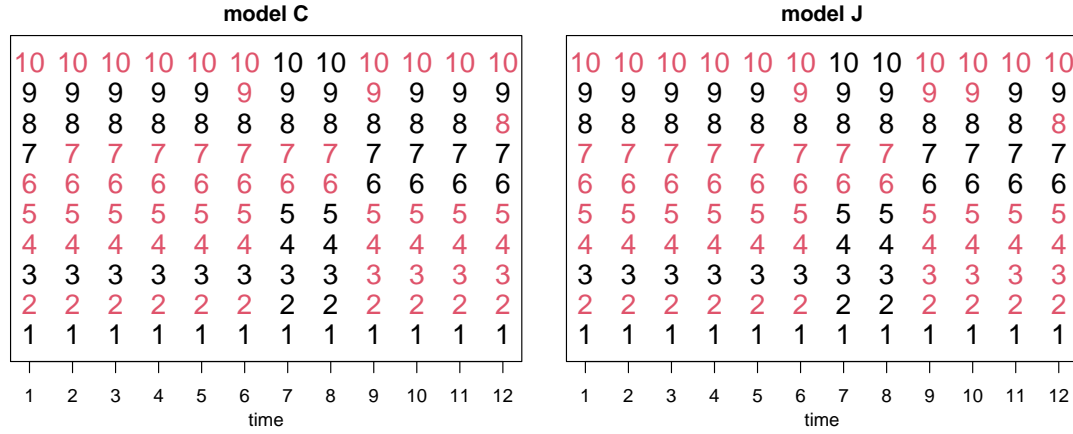


Figure 4.2: Clustering produced by the two models, with time points on the x axis, units indicated vertically by their ID number, and colors representing the cluster label.

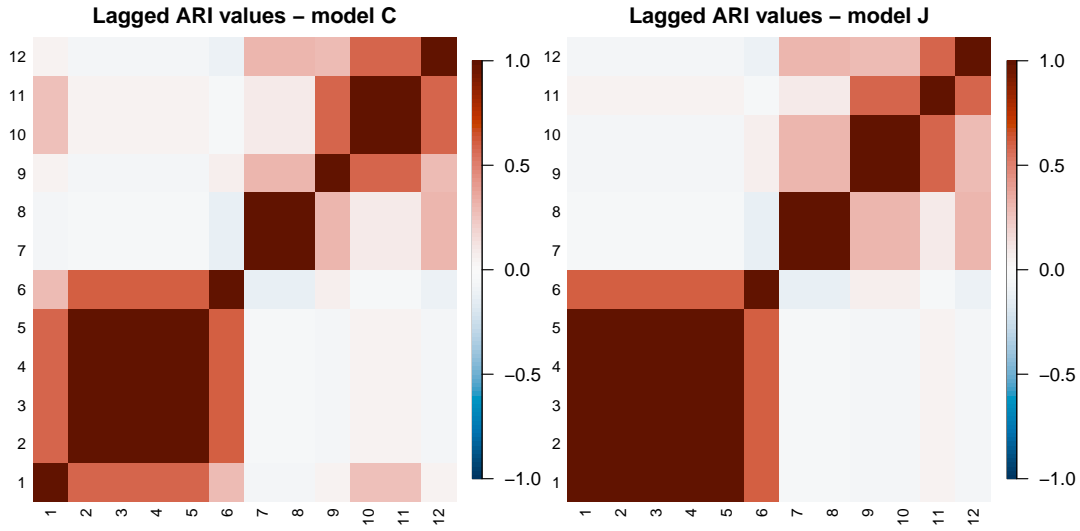


Figure 4.3: Lagged ARI values for the two models. The partitions were estimated using the `salso` function from the corresponding R library.

The resulting clusters were indeed very similar. Figure 4.3 shows how they manifest the same temporal trend, while Figure 4.2 highlights how the clusters produced are really the same except for two differences occurred at times 1 and 10. By visually inspecting the clusters from Figure 4.4, we can see how at $t = 1$

the JDRPM model assigned a red label to a unit closer to the black pack, to which in fact model CDRPM assigned a black label. However, that same unit in the following time instants would be clearly part of the red cluster, thus making the JDRPM choice very reasonable. The opposite happens at $t = 10$, where now the CDRPM model seems to give more importance on the temporal trend, assigning a black label to a unit really closer to the red pack but that is destined to enter the black cluster in the following two time instants.

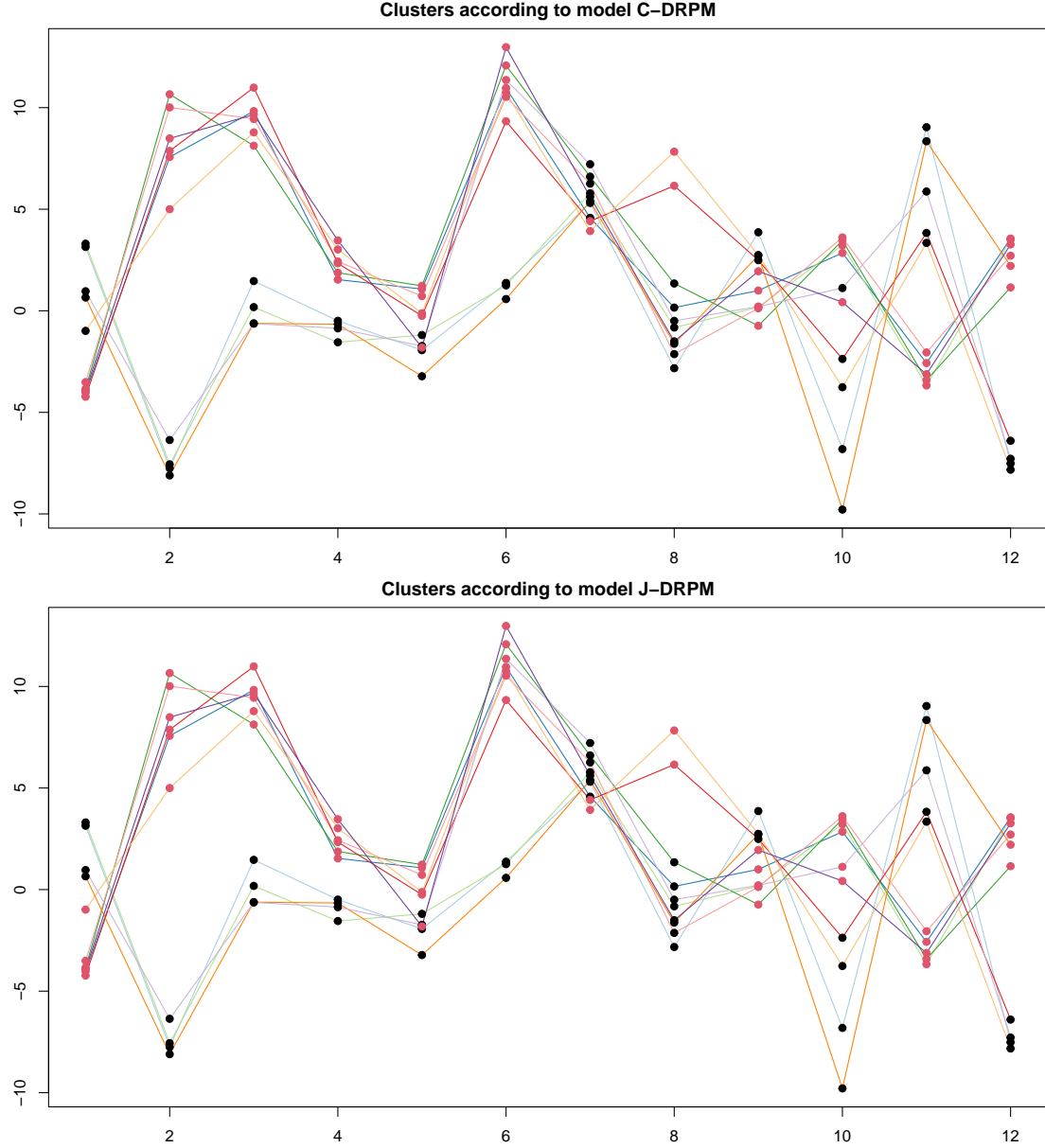


Figure 4.4: Visual representation of the clusters produced by the models, with units' labels represented as colored dots overlaid to the trend of the original generated target variables. The only differences occur at times 1 and 10.

4.1.2 Target variable plus space

4.2 Performance with missing values

We now repeat the tests of the previous section on a dataset with missing values, by randomly introducing some NAs into it to see how the JDRPM model reacts to the absence of a full dataset, and to see if it can still perform well. Based on the amount of missing values in the Agrimonia [Fas+23] dataset, used for the spatio-temporal dataset tests, we chose to set to 10% the percentage of values that would be replaced by NAs. To perform such replacements, we randomly drew $nT/10$ indexes from the sets $[1, \dots, n]$ and $[1, \dots, T]$ to get all the pairs (i, t) which would become missing values in the Y_{it} dataset. The treating of NAs was an upgrade of the JDRPM model, so we can't perform these tests with the original CDRPM model, since it cannot handle missing data.

4.2.1 Target variable only

The JDRPM seems to exhibit good performance also with missing data. From Table 4.2 we can see how the performances, naturally worse than the fit with the full case, are still good. The MSE is inevitably higher since the model did not have all the data points, and so we get less precise fitted values for the corresponding missing spots in the dataset. The WAIC is actually lower, i.e. better, but this could be just due to some randomness of the computation (e.g. the chosen seed).

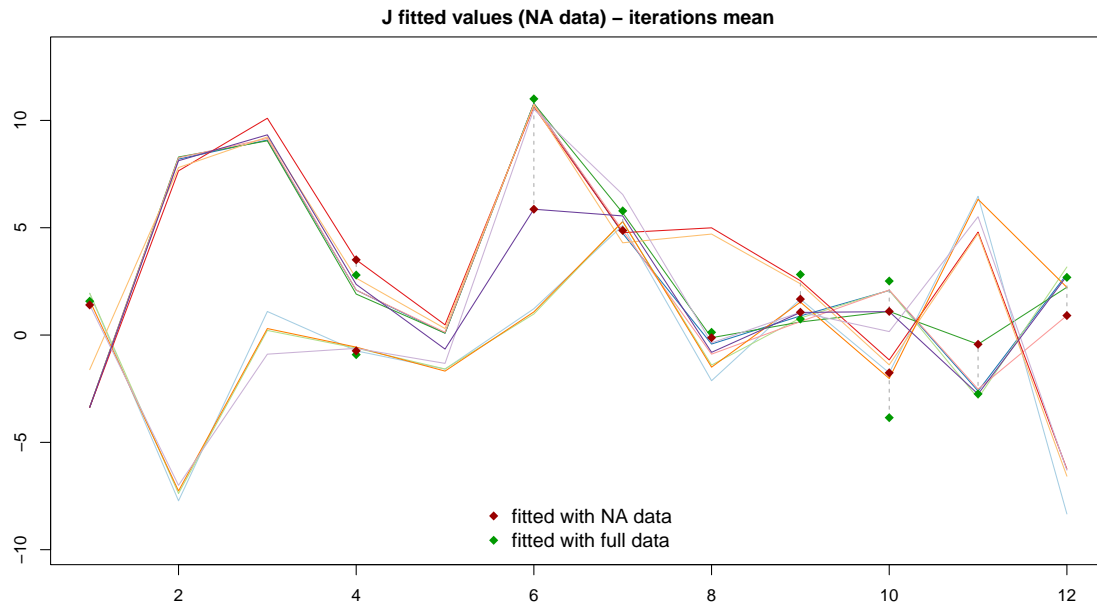


Figure 4.5: Fitted values estimate through the mean of the 1000 iterates generated by model JDRPM. The generated target values were the same of Figure 4.1. Special point markers are used for the data points corresponding to missing values, to highlight the gaps between the fit on the full dataset and the fit on the NA dataset.

Table 4.2: Summary of the comparison between the two Julia fits. The MSE columns refer to the accuracy between the fitted values generated by the models (estimated by taking the mean and the median of the returned 1000 iterates) and the true values of the target variable. Higher LPML and lower WAIC indicate a better fit.

<i>JDRPM</i>	MSE mean	MSE median	LPML	WAIC	execution time
NA data	2.181942	1.787575	-239.21	417.21	3.040 s
full data	1.432986	1.401098	-229.20	420.02	2.568 s

From Figures 4.6 and 4.7 we can see how we get almost the same temporal trend since indeed the assigned clusters are almost the same. The only difference occurs on a single unit at time 10, where now the JDRPM model does the same thing that CDRPM did in its corresponding fit of Section 4.1.1.

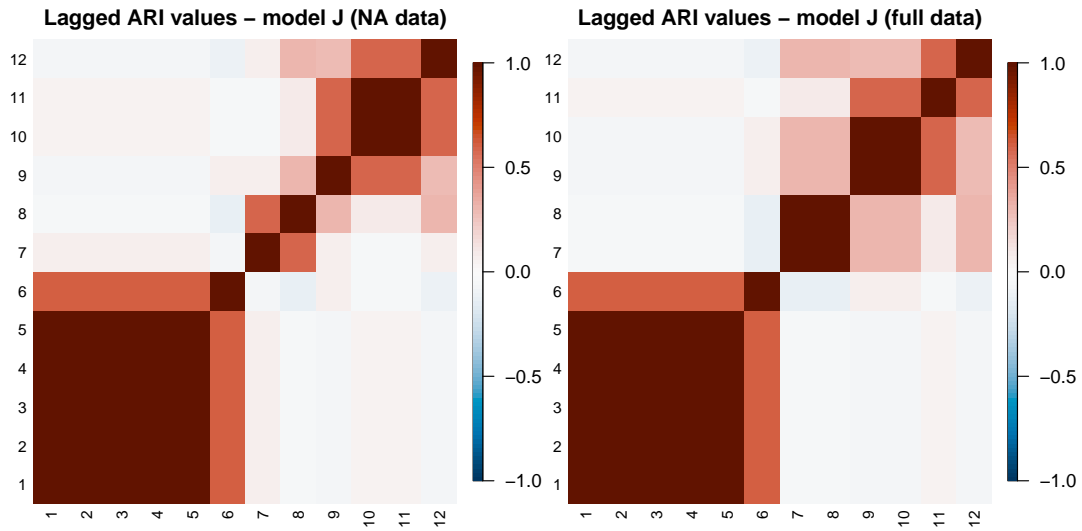


Figure 4.6: Lagged ARI values for the two tests on the JDRPM model. The partitions were estimated using the `salso` function from the corresponding R library.

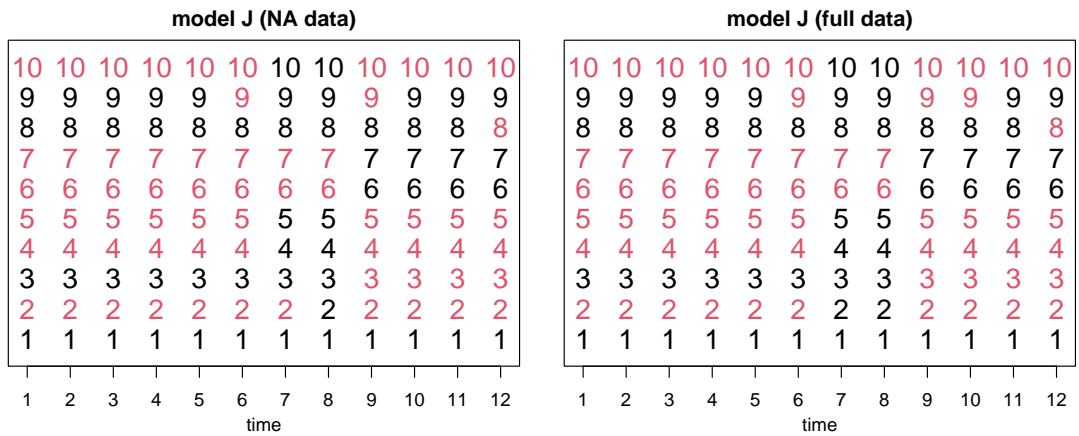


Figure 4.7: Clustering produced by the two tests on the JDRPM model, with time points on the x axis, units indicated vertically by their ID number, and colors representing the cluster label.

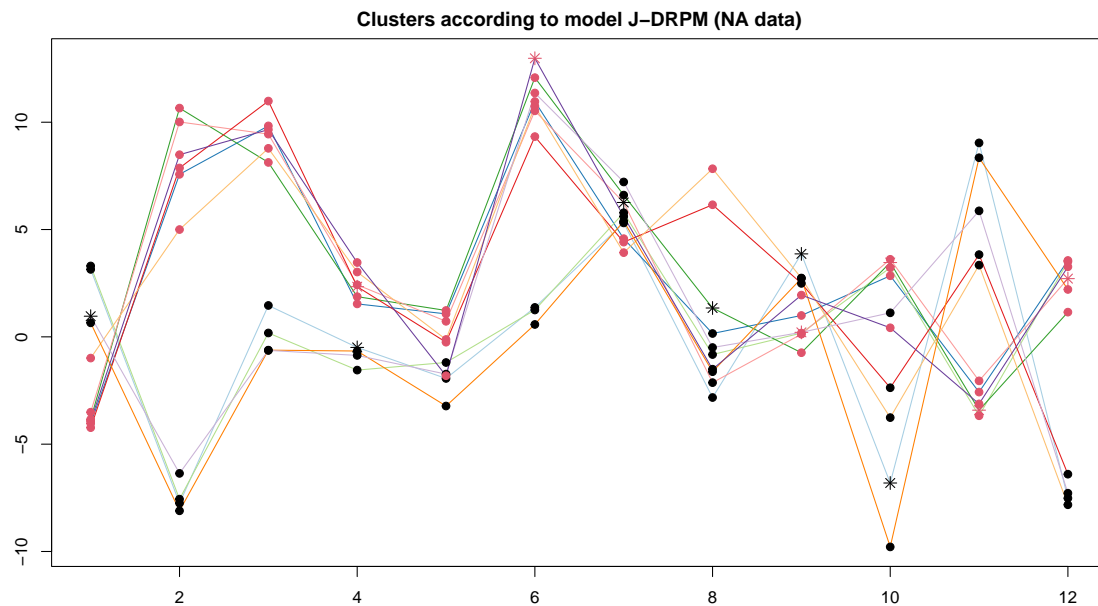


Figure 4.8: Visual representation of the clusters produced by the model, with units' labels represented as colored dots overlaid to the trend of the original generated target variables. Special point markers are used for the data points corresponding to missing values.

4.2.2 Target variable plus space

4.3 Effects of the covariates

4.4 Scaling performances

To really assess if the goal of a faster fit was reached, we setup several tests to compare the two models and the different ways in which they could be called. The original CDRPM allowed to perform clustering using just the target values or by also including spatial information. In addition to them, the options introduced by JDRPM are to include covariates at clustering and/or likelihood levels.

Apart, of course, from the target values, all the other information levels are indeed independent: one could perform the fit ignoring space and accounting just for covariates, for example; but with an additive perspective we decide to perform incremental tests, adding new layers on top of the base ones. Therefore, in what follows, we will see comparisons of fits using just the target values, then those plus space, then all those plus covariates in the clustering, and finally all them plus also covariates in the likelihood.

As it appeared from Section 3, the Julia implementation seems to be a little more eager for memory than the C one. This meant that for the larger tests we could have ran out of RAM memory, resulting in some data being stored in the swap space on the disk and therefore losing performance. However, on newer or just more technical-devoted machines than my simple laptop, this should not be an issue. Moreover, we have to remember that the Julia code is called from R, where

all the tests were performed, using the transmission control protocol (TCP) of the `JuliaConnectoR` library, rather than direct calls to the compiled source code as it happens when calling C from R. As a result, this indirect connection can also cause some small overhead due to the extra time and resources consumed by this communication step, especially in the really fast fits which occurred in the smallest tests.

Accounting all this, we should consider really accurate just the intermediate-values tests, the ones with n or T being 50 or 100, while take with a pinch of salt the two extremes ones of 10 and 250. Nonetheless, the results are really encouraging.

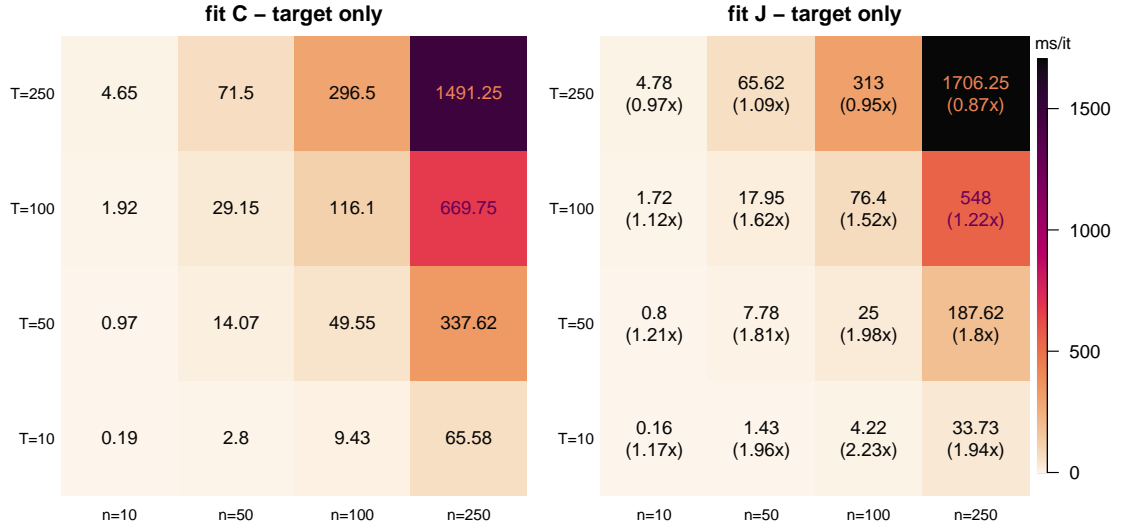


Figure 4.9: Execution times, measured in milliseconds per iteration, when fitting the models using just the target values to generate the clusters. On the JDRPM plot (on the right), in brackets, are present the speedups relative to the CDRPM timings (on the left), where higher values represent better performance.

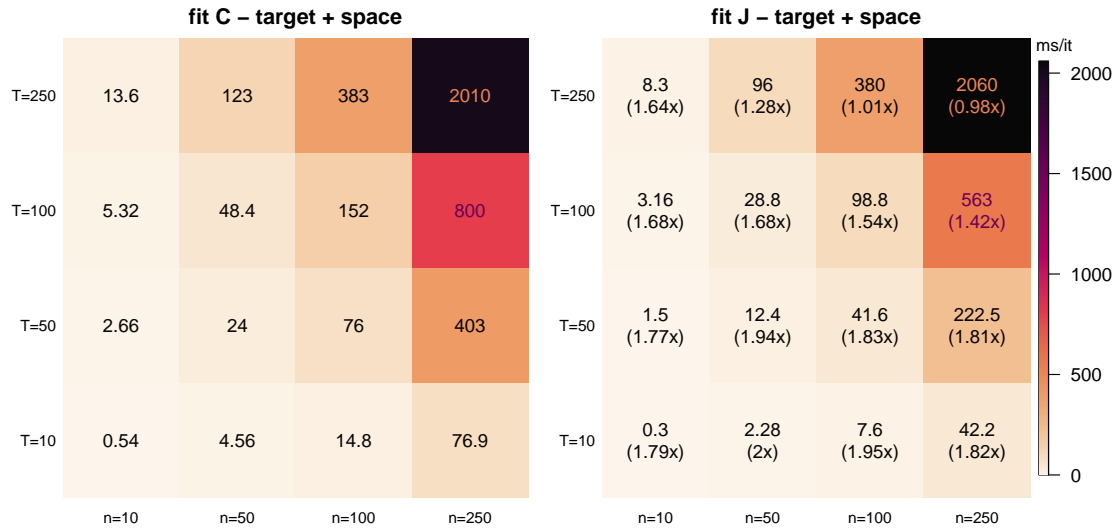


Figure 4.10: Execution times, measured in milliseconds per iteration, when fitting the models using the target values plus the spatial information to generate the clusters. On the JDRPM plot (on the right), in brackets, are present the speedups relative to the CDRPM timings (on the left), where higher values represent better performance.

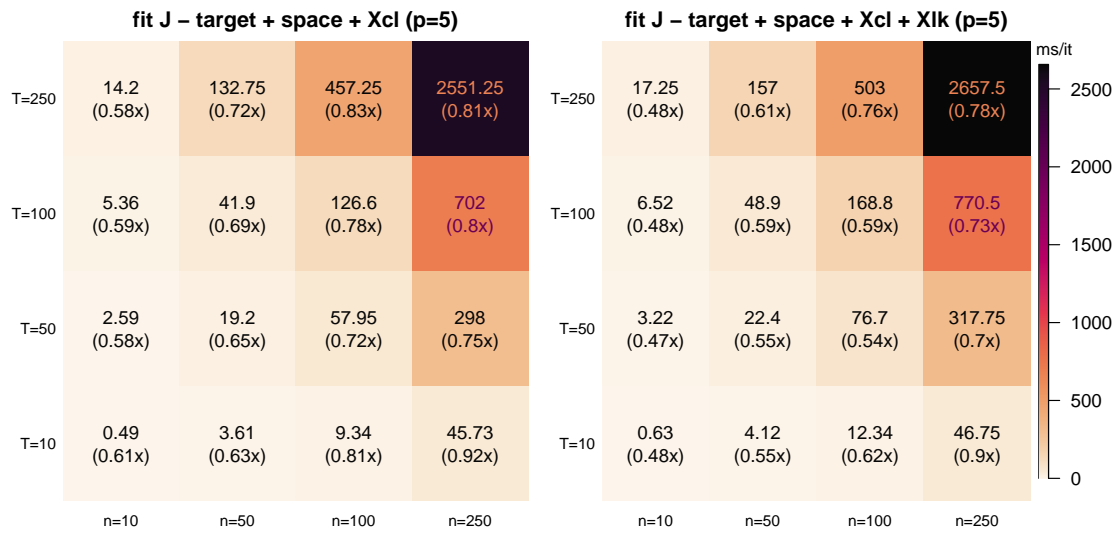


Figure 4.11: Enter Caption

Chapter 5

Conclusion

Honestly, I thought that I would be dead by now (wow)

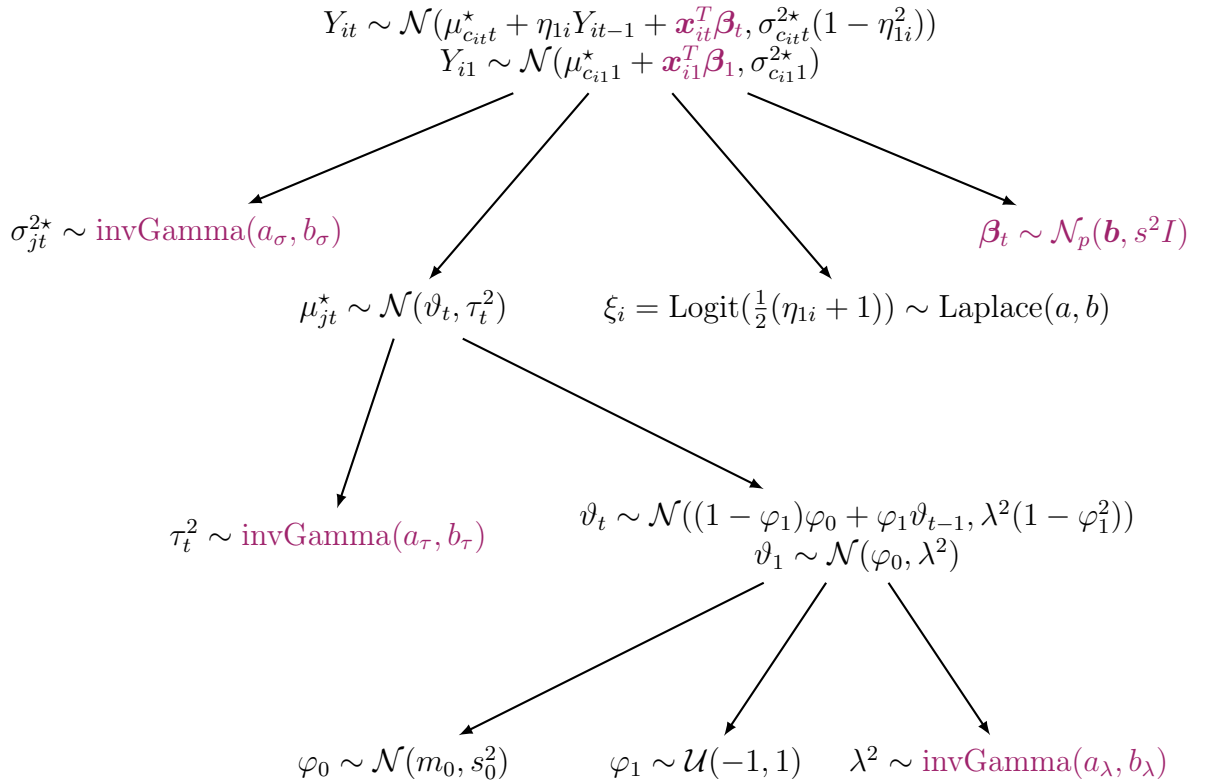
— Billie Eilish, *bury a friend*

Appendix A

Theoretical details

A.1 Extended computations of the full conditionals

We propose here the extended computations which allowed to extract the full conditionals presented in Chapter 2. We report also the model graph to make it quickly accessible as a reference for the laws involved in the following computations.



- update σ_{jt}^{2*}

for $t = 1$:

$$f(\sigma_{jt}^{2*} | -) \propto f(\sigma_{jt}^{2*}) f(\{Y_{it} : c_{it} = j\} | \sigma_{jt}^{2*}, -)$$

$$\begin{aligned}
&= \mathcal{L}_{\text{invGamma}(a_\sigma, b_\sigma)}(\sigma_{jt}^{2*}) \prod_{i \in S_{jt}} \mathcal{L}_{\mathcal{N}(\mu_{c_{it}t}^* + \mathbf{x}_{it}^T \boldsymbol{\beta}_t, \sigma_{c_{it}1}^{2*})}(Y_{it}) \\
&\propto \left[\left(\frac{1}{\sigma_{jt}^{2*}} \right)^{a_\sigma + 1} \exp \left\{ -\frac{1}{\sigma_{jt}^{2*}} b \right\} \right] \\
&\cdot \left[\prod_{i \in S_{jt}} \left(\frac{1}{\sigma_{jt}^{2*}} \right)^{1/2} \exp \left\{ -\frac{1}{2\sigma_{jt}^{2*}} (Y_{it} - \mu_{jt}^* - \mathbf{x}_{it}^T \boldsymbol{\beta}_t)^2 \right\} \right] \\
&\propto \left(\frac{1}{\sigma_{jt}^{2*}} \right)^{(a_\sigma + |S_{jt}|/2) + 1} \exp \left\{ -\frac{1}{\sigma_{jt}^{2*}} \left(b_\sigma + \frac{1}{2} \sum_{i \in S_{jt}} (Y_{it} - \mu_{jt}^* - \mathbf{x}_{it}^T \boldsymbol{\beta}_t)^2 \right) \right\} \\
\Rightarrow f(\sigma_{jt}^{2*} | -) &\propto \text{kernel of a invGamma}(a_{\sigma(\text{post})}, b_{\sigma(\text{post})}) \text{ with} \\
a_{\tau(\text{post})} &= a_\sigma + \frac{|S_{jt}|}{2} \\
b_{\tau(\text{post})} &= b_\sigma + \frac{1}{2} \sum_{i \in S_{jt}} (Y_{it} - \mu_{jt}^* - \mathbf{x}_{it}^T \boldsymbol{\beta}_t)^2 \tag{A.1}
\end{aligned}$$

for $t > 1$:

$$\begin{aligned}
f(\sigma_{jt}^{2*} | -) &\propto f(\sigma_{jt}^{2*}) f(\{Y_{it} : c_{it} = j\} | \sigma_{jt}^{2*}, -) \\
&= \mathcal{L}_{\text{invGamma}(a_\sigma, b_\sigma)}(\sigma_{jt}^{2*}) \prod_{i \in S_{jt}} \mathcal{L}_{\mathcal{N}(\mu_{c_{it}t}^* + \eta_{1i} Y_{it-1} + \mathbf{x}_{it}^T \boldsymbol{\beta}_t, \sigma_{c_{it}1}^{2*})}(Y_{it}) \\
&\propto \left[\left(\frac{1}{\sigma_{jt}^{2*}} \right)^{a_\sigma + 1} \exp \left\{ -\frac{1}{\sigma_{jt}^{2*}} b \right\} \right] \\
&\cdot \left[\prod_{i \in S_{jt}} \left(\frac{1}{\sigma_{jt}^{2*}} \right)^{1/2} \exp \left\{ -\frac{1}{2\sigma_{jt}^{2*}} (Y_{it} - \mu_{jt}^* - \eta_{1i} Y_{it-1} - \mathbf{x}_{it}^T \boldsymbol{\beta}_t)^2 \right\} \right] \\
&\propto \left(\frac{1}{\sigma_{jt}^{2*}} \right)^{(a_\sigma + |S_{jt}|/2) + 1} \\
&\cdot \exp \left\{ -\frac{1}{\sigma_{jt}^{2*}} \left(b_\sigma + \frac{1}{2} \sum_{i \in S_{jt}} (Y_{it} - \eta_{1i} Y_{it-1} - \mu_{jt}^* - \mathbf{x}_{it}^T \boldsymbol{\beta}_t)^2 \right) \right\} \\
\Rightarrow f(\sigma_{jt}^{2*} | -) &\propto \text{kernel of a invGamma}(a_{\sigma(\text{post})}, b_{\sigma(\text{post})}) \text{ with} \\
a_{\tau(\text{post})} &= a_\sigma + \frac{|S_{jt}|}{2} \\
b_{\tau(\text{post})} &= b_\sigma + \frac{1}{2} \sum_{i \in S_{jt}} (Y_{it} - \mu_{jt}^* - \eta_{1i} Y_{it-1} - \mathbf{x}_{it}^T \boldsymbol{\beta}_t)^2 \tag{A.2}
\end{aligned}$$

- update μ_{jt}^*

for $t = 1$:

$$f(\mu_{jt}^* | -) \propto f(\mu_{jt}^*) f(\{Y_{it} : c_{it} = j\} | \mu_{jt}^*, -)$$

$$\begin{aligned}
&= \mathcal{L}_{\mathcal{N}(\vartheta_t, \tau_t^2)}(\mu_{jt}^*) \prod_{i \in S_{jt}} \mathcal{L}_{\mathcal{N}(\mu_{jt}^*, \sigma_{jt}^{2*})}(Y_{it}) \\
&\propto \exp \left\{ -\frac{1}{2\tau_t^2} (\mu_{jt}^* - \vartheta_t)^2 \right\} \exp \left\{ -\frac{1}{2\sigma_{jt}^{2*}} \left(\sum_{i \in S_{jt}} (\mu_{jt}^* - Y_{i1})^2 \right) \right\} \\
&\propto \exp \left\{ -\frac{1}{2\tau_t^2} (\mu_{jt}^* - \vartheta_t)^2 \right\} \exp \left\{ -\frac{|S_{jt}|}{2\sigma_{jt}^{2*}} \left(\mu_{jt}^* - \frac{\text{SUM}_y}{|S_{jt}|} \right)^2 \right\} \\
&\text{where } \text{SUM}_y = \sum_{i \in S_{jt}} Y_{i1} \\
&\implies f(\mu_{jt}^* | -) \propto \text{kernel of a } \mathcal{N}(\mu_{\mu_{jt}^*}^*(\text{post}), \sigma_{\mu_{jt}^*}^{2*}(\text{post})) \text{ with} \\
&\sigma_{\mu_{jt}^*}^{2*}(\text{post}) = \frac{1}{\frac{1}{\tau_t^2} + \frac{|S_{jt}|}{\sigma_{jt}^{2*}}} \\
&\mu_{\mu_{jt}^*}^*(\text{post}) = \sigma_{\mu_{jt}^*}^{2*}(\text{post}) \left(\frac{\vartheta_t}{\tau_t^2} + \frac{\text{SUM}_y}{\sigma_{jt}^{2*}} \right) \tag{A.3}
\end{aligned}$$

for $t > 1$:

$$\begin{aligned}
&f(\mu_{jt}^* | -) \propto f(\mu_{jt}^*) f(\{Y_{it} : c_{it} = j\} | \mu_{jt}^*, -) \\
&= \mathcal{L}_{\mathcal{N}(\vartheta_1, \tau_1^2)}(\mu_{jt}^*) \prod_{i \in S_{jt}} \mathcal{L}_{\mathcal{N}(\mu_{jt}^* + \eta_{1i} Y_{i,t-1}, \sigma_{jt}^{2*}(1 - \eta_{1i}^2))}(Y_{it}) \\
&\propto \exp \left\{ -\frac{1}{2\tau_t^2} (\mu_{jt}^* - \vartheta_t)^2 \right\} \\
&\cdot \exp \left\{ -\frac{1}{2\sigma_{jt}^{2*}} \left(\sum_{i \in S_{jt}} \frac{1}{1 - \eta_{1i}^2} (\mu_{jt}^* - (Y_{it} - \eta_{1i} Y_{i,t-1}))^2 \right) \right\} \\
&\propto \exp \left\{ -\frac{1}{2\tau_t^2} (\mu_{jt}^* - \vartheta_t)^2 \right\} \exp \left\{ -\frac{\text{SUM}_{e2}}{2\sigma_{jt}^{2*}} \left(\mu_{jt}^* - \frac{\text{SUM}_y}{\text{SUM}_{e2}} \right)^2 \right\} \\
&\text{where } \text{SUM}_y = \sum_{i \in S_{jt}} \frac{Y_{it} - \eta_{1i} Y_{i,t-1}}{1 - \eta_{1i}^2}, \text{SUM}_{e2} = \sum_{i \in S_{jt}} \frac{1}{1 - \eta_{1i}^2} \\
&\implies f(\mu_{jt}^* | -) \propto \text{kernel of a } \mathcal{N}(\mu_{\mu_{jt}^*}^*(\text{post}), \sigma_{\mu_{jt}^*}^{2*}(\text{post})) \text{ with} \\
&\sigma_{\mu_{jt}^*}^{2*}(\text{post}) = \frac{1}{\frac{1}{\tau_t^2} + \frac{\text{SUM}_{e2}}{\sigma_{jt}^{2*}}} \\
&\mu_{\mu_{jt}^*}^*(\text{post}) = \sigma_{\mu_{jt}^*}^{2*}(\text{post}) \left(\frac{\vartheta_t}{\tau_t^2} + \frac{\text{SUM}_y}{\sigma_{jt}^{2*}} \right) \tag{A.4}
\end{aligned}$$

- update β_t

for $t = 1$:

$$f(\beta_t | -) \propto f(\beta_t) f(\{Y_{1t}, \dots, Y_{nt}\} | \beta_t, -)$$

$$\begin{aligned}
&= \mathcal{L}_{\mathcal{N}(\mathbf{b}, s^2 I)}(\boldsymbol{\beta}_t) \prod_{i=1}^n \mathcal{L}_{\mathcal{N}(\mu_{c_{it}}^* + \mathbf{x}_{it}^T \boldsymbol{\beta}_t, \sigma_{c_{it}}^{2*})}(Y_{it}) \\
&\propto \exp \left\{ -\frac{1}{2} \left[(\boldsymbol{\beta}_t^T - \mathbf{b})^T \frac{1}{s^2} (\boldsymbol{\beta}_t - \mathbf{b}) \right] \right\} \exp \left\{ -\frac{1}{2} \sum_{i=1}^n (Y_{it} - \mu_{c_{it}}^* - \mathbf{x}_{it}^T \boldsymbol{\beta}_t)^2 \right\} \\
&\propto \exp \left\{ -\frac{1}{2} \left[\boldsymbol{\beta}_t^T \left(\frac{1}{s^2} I + \sum_{i=1}^n \frac{\mathbf{x}_{it} \mathbf{x}_{it}^T}{\sigma_{c_{it}}^{2*}} \right) \boldsymbol{\beta}_t \right. \right. \\
&\quad \left. \left. - 2 \left(\frac{\mathbf{b}}{s^2} + \sum_{i=1}^n \frac{(Y_{it} - \mu_{c_{it}}^*) \mathbf{x}_{it}}{\sigma_{c_{it}}^{2*}} \right) \cdot \boldsymbol{\beta}_t \right] \right\} \\
&\implies f(\boldsymbol{\beta}_t | -) \propto \text{kernel of a } \mathcal{N}(\mathbf{b}_{(\text{post})}, A_{(\text{post})}) \text{ with} \\
&\quad A_{(\text{post})} = \left(\frac{1}{s^2} I + \sum_{i=1}^n \frac{\mathbf{x}_{it} \mathbf{x}_{it}^T}{\sigma_{c_{it}}^{2*}} \right)^{-1} \\
&\quad \mathbf{b}_{(\text{post})} = A_{(\text{post})} \left(\frac{\mathbf{b}}{s^2} + \sum_{i=1}^n \frac{(Y_{it} - \mu_{c_{it}}^*) \mathbf{x}_{it}}{\sigma_{c_{it}}^{2*}} \right) \\
&\iff f(\boldsymbol{\beta}_t | -) \propto \text{kernel of a } \mathcal{N}\text{Canon}(\mathbf{h}_{(\text{post})}, J_{(\text{post})}) \text{ with} \\
&\quad \mathbf{h}_{(\text{post})} = \left(\frac{\mathbf{b}}{s^2} + \sum_{i=1}^n \frac{(Y_{it} - \mu_{c_{it}}^*) \mathbf{x}_{it}}{\sigma_{c_{it}}^{2*}} \right) \\
&\quad J_{(\text{post})} = \left(\frac{1}{s^2} I + \sum_{i=1}^n \frac{\mathbf{x}_{it} \mathbf{x}_{it}^T}{\sigma_{c_{it}}^{2*}} \right) \tag{A.5}
\end{aligned}$$

for $t > 1$:

$$\begin{aligned}
&f(\boldsymbol{\beta}_t | -) \propto f(\boldsymbol{\beta}_t) f(\{Y_{1t}, \dots, Y_{nt}\} | \boldsymbol{\beta}_t, -) \\
&= \mathcal{L}_{\mathcal{N}(\mathbf{b}, s^2 I)}(\boldsymbol{\beta}_t) \prod_{i=1}^n \mathcal{L}_{\mathcal{N}(\mu_{c_{it}}^* + \eta_{1i} Y_{it-1} + \mathbf{x}_{it}^T \boldsymbol{\beta}_t, \sigma_{c_{it}}^{2*})}(Y_{it}) \\
&\propto \exp \left\{ -\frac{1}{2} \left[(\boldsymbol{\beta}_t^T - \mathbf{b})^T \frac{1}{s^2} (\boldsymbol{\beta}_t - \mathbf{b}) \right] \right\} \\
&\quad \cdot \exp \left\{ -\frac{1}{2} \sum_{i=1}^n (Y_{it} - \mu_{c_{it}}^* - \eta_{1i} Y_{it-1} - \mathbf{x}_{it}^T \boldsymbol{\beta}_t)^2 \right\} \\
&\propto \exp \left\{ -\frac{1}{2} \left[\boldsymbol{\beta}_t^T \left(\frac{1}{s^2} I + \sum_{i=1}^n \frac{\mathbf{x}_{it} \mathbf{x}_{it}^T}{\sigma_{c_{it}}^{2*}} \right) \boldsymbol{\beta}_t \right. \right. \\
&\quad \left. \left. - 2 \left(\frac{\mathbf{b}}{s^2} + \sum_{i=1}^n \frac{(Y_{it} - \mu_{c_{it}}^* - \eta_{1i} Y_{it-1}) \mathbf{x}_{it}}{\sigma_{c_{it}}^{2*}} \right) \cdot \boldsymbol{\beta}_t \right] \right\} \\
&\implies f(\boldsymbol{\beta}_t | -) \propto \text{kernel of a } \mathcal{N}(\mathbf{b}_{(\text{post})}, A_{(\text{post})}) \text{ with} \\
&\quad A_{(\text{post})} = \left(\frac{1}{s^2} I + \sum_{i=1}^n \frac{\mathbf{x}_{it} \mathbf{x}_{it}^T}{\sigma_{c_{it}}^{2*}} \right)^{-1}
\end{aligned}$$

$$\begin{aligned}
\mathbf{b}_{(\text{post})} &= A_{(\text{post})} \left(\frac{\mathbf{b}}{s^2} + \sum_{i=1}^n \frac{(Y_{it} - \mu_{c_{it}t}^* - \eta_{1i} Y_{it-1}) \mathbf{x}_{it}}{\sigma_{c_{it}t}^{2*}} \right) \\
\iff f(\boldsymbol{\beta}_t | -) &\propto \text{kernel of a } \mathcal{N}\text{Canon}(\mathbf{h}_{(\text{post})}, J_{(\text{post})}) \text{ with} \\
\mathbf{h}_{(\text{post})} &= \left(\frac{\mathbf{b}}{s^2} + \sum_{i=1}^n \frac{(Y_{it} - \mu_{c_{it}t}^* - \eta_{1i} Y_{it-1}) \mathbf{x}_{it}}{\sigma_{c_{it}t}^{2*}} \right) \\
J_{(\text{post})} &= \left(\frac{1}{s^2} I + \sum_{i=1}^n \frac{\mathbf{x}_{it} \mathbf{x}_{it}^T}{\sigma_{c_{it}t}^{2*}} \right)
\end{aligned} \tag{A.6}$$

- update τ_t^2

$$\begin{aligned}
f(\tau_t^2 | -) &\propto f(\tau_t^2) f((\mu_{1t}^*, \dots, \mu_{k_{tt}}^*) | \tau_t^2, -) \\
&= \mathcal{L}_{\text{invGamma}(a_\tau, b_\tau)}(\tau_t^2) \prod_{j=1}^{k_t} \mathcal{L}_{\mathcal{N}(\vartheta_t, \tau_t^2)}(\mu_{jt}^*) \\
&\propto \left[\left(\frac{1}{\tau_t^2} \right)^{a_\tau+1} \exp \left\{ -\frac{b_\tau}{\tau_t^2} \right\} \right] \left[\prod_{j=1}^{k_t} \left(\frac{1}{\tau_t^2} \right)^{1/2} \exp \left\{ -\frac{1}{2\tau_t^2} (\mu_{jt}^* - \vartheta_t)^2 \right\} \right] \\
&\propto \left(\frac{1}{\tau_t^2} \right)^{\left(\frac{k_t}{2} + a_\tau \right) + 1} \exp \left\{ -\frac{1}{\tau_t^2} \left(\frac{\sum_{j=1}^{k_t} (\mu_{jt}^* - \vartheta_t)^2}{2} + b_\tau \right) \right\} \\
\implies f(\tau_t^2 | -) &\propto \text{kernel of a } \text{invGamma}(a_{\tau(\text{post})}, b_{\tau(\text{post})}) \text{ with} \\
a_{\tau(\text{post})} &= \frac{k_t}{2} + a_\tau \\
b_{\tau(\text{post})} &= \frac{\sum_{j=1}^{k_t} (\mu_{jt}^* - \vartheta_t)^2}{2} + b_\tau
\end{aligned} \tag{A.7}$$

- update ϑ_t

for $t = T$:

$$\begin{aligned}
f(\vartheta_t | -) &\propto f(\vartheta_t) f((\mu_{1t}^*, \dots, \mu_{k_{tt}}^*) | \vartheta_t, -) \\
&= \mathcal{L}_{\mathcal{N}((1-\varphi_1)\varphi_0 + \varphi_1 \vartheta_{t-1}, \lambda^2(1-\varphi_1^2))}(\vartheta_t) \prod_{j=1}^{k_t} \mathcal{L}_{\mathcal{N}(\vartheta_t, \tau_t^2)}(\mu_{jt}^*) \\
&\propto \exp \left\{ -\frac{1}{2(\lambda^2(1-\varphi_1^2))} \left(\vartheta_t - ((1-\varphi_1)\varphi_0 + \varphi_1 \vartheta_{t-1}) \right)^2 \right\} \\
&\quad \cdot \exp \left\{ -\frac{k_t}{2\tau_t^2} \left(\vartheta_t - \frac{\sum_{j=1}^{k_t} \mu_{jt}^*}{k_t} \right)^2 \right\} \\
\implies f(\vartheta_t | -) &\propto \text{kernel of a } \mathcal{N}(\mu_{\vartheta_t(\text{post})}, \sigma_{\vartheta_t(\text{post})}^2) \text{ with} \\
\sigma_{\vartheta_t(\text{post})}^2 &= \frac{1}{\frac{1}{\lambda^2(1-\varphi_1^2)} + \frac{k_t}{\tau_t^2}} \\
\mu_{\vartheta_t(\text{post})} &= \sigma_{\vartheta_t(\text{post})}^2 \left(\frac{\sum_{j=1}^{k_t} \mu_{jt}^*}{\tau_t^2} + \frac{(1-\varphi_1)\varphi_0 + \varphi_1 \vartheta_{t-1}}{\lambda^2(1-\varphi_1^2)} \right)
\end{aligned} \tag{A.8}$$

for $1 < t < T$:

$$\begin{aligned}
f(\vartheta_t|-) &\propto \underbrace{f(\vartheta_t)f((\mu_{1t}^*, \dots, \mu_{k_{tt}}^*)|\vartheta_t, -)}_{\text{as in the case } t=T} f(\vartheta_{t+1}|\vartheta_t, -) \\
&= \mathcal{L}_{\mathcal{N}(\mu_{\vartheta_t(\text{post})}, \sigma_{\vartheta_t(\text{post})}^2)}(\vartheta_t) \mathcal{L}_{\mathcal{N}((1-\varphi_1)\varphi_0 + \varphi_1\vartheta_t, \lambda^2(1-\varphi_1^2))}(\vartheta_{t+1}) \\
&\propto \exp \left\{ -\frac{1}{2\sigma_{\vartheta_t(\text{post})}^2} (\vartheta_t - \mu_{\vartheta_t(\text{post})})^2 \right\} \\
&\quad \cdot \exp \left\{ -\frac{1}{2\frac{\lambda^2(1-\varphi_1^2)}{\varphi_1^2}} \left(\vartheta_t - \frac{\vartheta_{t+1} - (1-\varphi_1)\varphi_0}{\varphi_1} \right)^2 \right\} \\
\implies f(\vartheta_t|-) &\propto \text{kernel of a } \mathcal{N}(\mu_{\vartheta_t(\text{post})}, \sigma_{\vartheta_t(\text{post})}^2) \text{ with} \\
\sigma_{\vartheta_t(\text{post})}^2 &= \frac{1}{\frac{1+\varphi_1^2}{\lambda^2(1-\varphi_1^2)} + \frac{k_t}{\tau_t^2}} \\
\mu_{\vartheta_t(\text{post})} &= \sigma_{\vartheta_t(\text{post})}^2 \left(\frac{\sum_{j=1}^{k_t} \mu_{jt}^*}{\tau_t^2} + \frac{\varphi_1(\vartheta_{t-1} + \vartheta_{t+1}) + \varphi_0(1-\varphi_1)^2}{\lambda^2(1-\varphi_1^2)} \right) \quad (\text{A.9})
\end{aligned}$$

for $t = 1$:

$$\begin{aligned}
f(\vartheta_t|-) &\propto f(\vartheta_t)f(\vartheta_{t+1}|\vartheta_t, -)f(\boldsymbol{\mu}_{jt}^*|\vartheta_t, -) \\
&= \mathcal{L}_{\mathcal{N}(\varphi_0, \lambda^2)}(\vartheta_t) \mathcal{L}_{\mathcal{N}((1-\varphi_1)\varphi_0 + \varphi_1\vartheta_{t-1}, \lambda^2(1-\varphi_1^2))}(\vartheta_{t+1}) \prod_{j=1}^{k_t} \mathcal{L}_{\mathcal{N}(\vartheta_t, \tau_t^2)}(\mu_{jt}^*) \\
&\propto \exp \left\{ -\frac{1}{2\lambda^2} (\vartheta_t - \varphi_0)^2 \right\} \\
&\quad \cdot \exp \left\{ -\frac{1}{2\frac{\lambda^2(1-\varphi_1^2)}{\varphi_1^2}} \left(\vartheta_t - \frac{\vartheta_{t+1} - (1-\varphi_1)\varphi_0}{\varphi_1} \right)^2 \right\} \\
&\quad \cdot \exp \left\{ -\frac{k_t}{2\tau_t^2} \left(\vartheta_t - \frac{\sum_{j=1}^{k_t} \mu_{jt}^*}{k_t} \right)^2 \right\} \\
\implies f(\vartheta_t|-) &\propto \text{kernel of a } \mathcal{N}(\mu_{\vartheta_t(\text{post})}, \sigma_{\vartheta_t(\text{post})}^2) \text{ with} \\
\sigma_{\vartheta_t(\text{post})}^2 &= \frac{1}{\frac{1}{\lambda^2} + \frac{\varphi_1^2}{\lambda^2(1-\varphi_1^2)} + \frac{k_t}{\tau_t^2}} \\
\mu_{\vartheta_t(\text{post})} &= \sigma_{\vartheta_t(\text{post})}^2 \left(\frac{\varphi_0}{\lambda^2} + \frac{\varphi_1(\vartheta_{t+1} - (1-\varphi_1)\varphi_0)}{\lambda^2(1-\varphi_1^2)} + \frac{\sum_{j=1}^{k_t} \mu_{jt}^*}{\tau_t^2} \right) \quad (\text{A.10})
\end{aligned}$$

- update φ_0

$$\begin{aligned}
f(\varphi_0|-) &\propto f(\varphi_0)f((\vartheta_1, \dots, \vartheta_T)|\varphi_0, -) \\
&= \mathcal{L}_{\mathcal{N}(m_0, s_0^2)}(\varphi_0) \mathcal{L}_{\mathcal{N}(\varphi_0, \lambda^2)}(\vartheta_1) \prod_{t=2}^T \mathcal{L}_{\mathcal{N}((1-\varphi_1)\varphi_0 + \varphi_1\vartheta_{t-1}, \lambda^2(1-\varphi_1^2))}(\vartheta_t) \\
&\propto \exp \left\{ -\frac{1}{2s_0^2} (\varphi_0 - m_0)^2 \right\} \exp \left\{ -\frac{1}{2\lambda^2} (\varphi_0 - \vartheta_1)^2 \right\}
\end{aligned}$$

$$\begin{aligned}
& \cdot \exp \left\{ -\frac{1}{2 \frac{\lambda^2(1-\varphi_1^2)}{(T-1)(1-\varphi_1)^2}} \left(\varphi_0 - \frac{(1-\varphi_1)(\text{SUM}_t)}{(T-1)(1-\varphi_1)^2} \right)^2 \right\} \\
& \text{where } \text{SUM}_t = \sum_{t=2}^T (\vartheta_t - \varphi_1 \vartheta_{t-1}) \\
& \Rightarrow f(\varphi_0 | -) \propto \text{kernel of a } \mathcal{N}(\mu_{\varphi_0(\text{post})}, \sigma_{\varphi_0(\text{post})}^2) \text{ with} \\
& \sigma_{\varphi_0(\text{post})}^2 = \frac{1}{\frac{1}{s_0^2} + \frac{1}{\lambda^2} + \frac{(T-1)(1-\varphi_1)^2}{\lambda^2(1-\varphi_1^2)}} \\
& \mu_{\varphi_0(\text{post})} = \sigma_{\varphi_0(\text{post})}^2 \left(\frac{m_0}{s_0^2} + \frac{\vartheta_1}{\lambda^2} + \frac{1-\varphi_1}{\lambda^2(1-\varphi_1^2)} \text{SUM}_t \right) \tag{A.11}
\end{aligned}$$

- update λ^2

$$\begin{aligned}
& f(\lambda^2 | -) \propto f(\lambda^2) f(\vartheta_1, \dots, \vartheta_T | \lambda^2, -) \\
& = \mathcal{L}_{\text{invGamma}(a_\lambda, b_\lambda)}(\lambda^2) \mathcal{L}_{\mathcal{N}(\varphi_0, \lambda^2)}(\vartheta_1) \prod_{t=2}^T \mathcal{L}_{\mathcal{N}((1-\varphi_1)\varphi_0 + \varphi_1 \vartheta_{t-1}, \lambda^2(1-\varphi_1^2))}(\vartheta_t) \\
& \propto \left[\left(\frac{1}{\lambda^2} \right)^{a_\lambda+1} \exp \left\{ -\frac{b_\lambda}{\lambda^2} \right\} \right] \left[\left(\frac{1}{\lambda^2} \right)^{1/2} \exp \left\{ -\frac{1}{2\lambda^2} (\vartheta_1 - \varphi_0)^2 \right\} \right] \\
& \cdot \left[\prod_{t=2}^T \left(\frac{1}{\lambda^2} \right)^{1/2} \exp \left\{ -\frac{1}{2\lambda^2} (\vartheta_t - (1-\varphi_1)\varphi_0 - \varphi_1 \vartheta_{t-1})^2 \right\} \right] \\
& \propto \left(\frac{1}{\lambda^2} \right)^{\left(\frac{T}{2} + a_\lambda \right) + 1} \\
& \cdot \exp \left\{ -\frac{1}{\lambda^2} \left(\frac{(\vartheta_1 - \varphi_0)^2}{2} + \sum_{t=2}^T \frac{(\vartheta_t - (1-\varphi_1)\varphi_0 - \varphi_1 \vartheta_{t-1})^2}{2} + b_\lambda \right) \right\} \\
& \Rightarrow f(\lambda^2 | -) \propto \text{kernel of a invGamma}(a_{\lambda(\text{post})}, b_{\lambda(\text{post})}) \text{ with} \\
& a_{\lambda(\text{post})} = \frac{T}{2} + a_\lambda \\
& b_{\lambda(\text{post})} = \frac{(\vartheta_1 - \varphi_0)^2}{2} + \sum_{t=2}^T \frac{(\vartheta_t - (1-\varphi_1)\varphi_0 - \varphi_1 \vartheta_{t-1})^2}{2} + b_\lambda \tag{A.12}
\end{aligned}$$

- update a missing observation Y_{it}

for $t = 1$:

$$\begin{aligned}
& f(Y_{it} | -) \propto f(Y_{it}) f(Y_{it+1} | Y_{it}, -) \\
& = \mathcal{L}_{\mathcal{N}(\mu_{c_{it}}^* + \mathbf{x}_{it}^T \boldsymbol{\beta}_t, \sigma_{c_{it}}^{2*})}(Y_{it}) \\
& \cdot \mathcal{L}_{\mathcal{N}(\mu_{c_{it+1}t+1}^* + \eta_{1i} Y_{it} + \mathbf{x}_{it+1}^T \boldsymbol{\beta}_{t+1}, \sigma_{c_{it+1}t+1}^{2*}(1-\eta_{1i}^2))}(Y_{it+1}) \\
& \propto \exp \left\{ -\frac{1}{2\sigma_{c_{it}}^{2*}} (Y_{it} - \mu_{c_{it}}^* - \mathbf{x}_{it}^T \boldsymbol{\beta}_t)^2 \right\} \\
& \cdot \exp \left\{ -\frac{1}{2\sigma_{c_{it+1}t+1}^{2*}(1-\eta_{1i}^2)} (Y_{it+1} - \mu_{c_{it+1}t+1}^* - \eta_{1i} Y_{it} - \mathbf{x}_{it+1}^T \boldsymbol{\beta}_{t+1})^2 \right\}
\end{aligned}$$

$$\begin{aligned}
&= \exp \left\{ -\frac{1}{2\sigma_{c_{it}t}^{2*}} (Y_{it} - (\mu_{c_{it}t}^* + \mathbf{x}_{it}^T \boldsymbol{\beta}_t))^2 \right\} \\
&\cdot \exp \left\{ -\frac{\eta_{1i}^2}{2\sigma_{c_{it+1}t+1}^{2*}(1-\eta_{1i}^2)} \left(Y_{it} - \frac{Y_{it+1} - \mu_{c_{it+1}t+1}^* - \mathbf{x}_{it+1}^T \boldsymbol{\beta}_{t+1}}{\eta_{1i}} \right)^2 \right\} \\
\Rightarrow f(Y_{it}|-) &\propto \text{kernel of a } \mathcal{N}(\mu_{Y_{it}(\text{post})}, \sigma_{Y_{it}(\text{post})}^2) \text{ with} \\
\sigma_{Y_{it}(\text{post})}^2 &= \frac{1}{\frac{1}{\sigma_{c_{it}t}^{2*}} + \frac{\eta_{1i}^2}{2\sigma_{c_{it+1}t+1}^{2*}(1-\eta_{1i}^2)}} \\
\mu_{Y_{it}(\text{post})} &= \sigma_{Y_{it}(\text{post})}^2 \left(\frac{\mu_{c_{it}t}^* + \mathbf{x}_{it}^T \boldsymbol{\beta}_t}{\sigma_{c_{it}t}^{2*}} + \frac{\eta_{1i}(Y_{it+1} - \mu_{c_{it+1}t+1}^* - \mathbf{x}_{it+1}^T \boldsymbol{\beta}_{t+1})}{\sigma_{c_{it+1}t+1}^{2*}(1-\eta_{1i}^2)} \right)
\end{aligned} \tag{A.13}$$

for $1 < t < T$:

$$\begin{aligned}
f(Y_{it}|-) &\propto f(Y_{it})f(Y_{it+1}|Y_{it}, -) \\
&= \mathcal{L}_{\mathcal{N}(\mu_{c_{it}t}^* + \eta_{1i}Y_{it-1} + \mathbf{x}_{it}^T \boldsymbol{\beta}_t, \sigma_{c_{it}t}^{2*})}(Y_{it}) \\
&\cdot \mathcal{L}_{\mathcal{N}(\mu_{c_{it+1}t+1}^* + \eta_{1i}Y_{it} + \mathbf{x}_{it+1}^T \boldsymbol{\beta}_{t+1}, \sigma_{c_{it+1}t+1}^{2*}(1-\eta_{1i}^2))}(Y_{it+1}) \\
&\propto \exp \left\{ -\frac{1}{2\sigma_{c_{it}t}^{2*}(1-\eta_{1i}^2)} (Y_{it} - \mu_{c_{it}t}^* - \eta_{1i}Y_{it-1} - \mathbf{x}_{it}^T \boldsymbol{\beta}_t)^2 \right\} \\
&\cdot \exp \left\{ -\frac{1}{2\sigma_{c_{it+1}t+1}^{2*}(1-\eta_{1i}^2)} (Y_{it+1} - \mu_{c_{it+1}t+1}^* - \eta_{1i}Y_{it} - \mathbf{x}_{it+1}^T \boldsymbol{\beta}_{t+1})^2 \right\} \\
&= \exp \left\{ -\frac{1}{2\sigma_{c_{it}t}^{2*}(1-\eta_{1i}^2)} (Y_{it} - (\mu_{c_{it}t}^* + \eta_{1i}Y_{it-1} + \mathbf{x}_{it}^T \boldsymbol{\beta}_t))^2 \right\} \\
&\cdot \exp \left\{ -\frac{\eta_{1i}^2}{2\sigma_{c_{it+1}t+1}^{2*}(1-\eta_{1i}^2)} \left(Y_{it} - \frac{Y_{it+1} - \mu_{c_{it+1}t+1}^* - \mathbf{x}_{it+1}^T \boldsymbol{\beta}_{t+1}}{\eta_{1i}} \right)^2 \right\} \\
\Rightarrow f(Y_{it}|-) &\propto \text{kernel of a } \mathcal{N}(\mu_{Y_{it}(\text{post})}, \sigma_{Y_{it}(\text{post})}^2) \text{ with} \\
\sigma_{Y_{it}(\text{post})}^2 &= \frac{1 - \eta_{1i}^2}{\frac{1}{\sigma_{c_{it}t}^{2*}} + \frac{\eta_{1i}^2}{\sigma_{c_{it+1}t+1}^{2*}}} \\
\mu_{Y_{it}(\text{post})} &= \sigma_{Y_{it}(\text{post})}^2 \left(\frac{\mu_{c_{it}t}^* + \eta_{1i}Y_{it-1} + \mathbf{x}_{it}^T \boldsymbol{\beta}_t}{\sigma_{c_{it}t}^{2*}(1-\eta_{1i}^2)} + \frac{\eta_{1i}(Y_{it+1} - \mu_{c_{it+1}t+1}^* - \mathbf{x}_{it+1}^T \boldsymbol{\beta}_{t+1})}{\sigma_{c_{it+1}t+1}^{2*}(1-\eta_{1i}^2)} \right)
\end{aligned} \tag{A.14}$$

for $t = T$:

$$\begin{aligned}
f(Y_{it}|-) &\propto f(Y_{it}) \\
&= \mathcal{L}_{\mathcal{N}(\mu_{c_{it}t}^* + \eta_{1i}Y_{it-1} + \mathbf{x}_{it}^T \boldsymbol{\beta}_t, \sigma_{c_{it}t}^{2*}(1-\eta_{1i}^2))}(Y_{it}) \\
\Rightarrow f(Y_{it}|-) &\text{ is just the likelihood of } Y_{it}
\end{aligned} \tag{A.15}$$

Appendix B

Computational details

B.1 Fitting algorithm code

We report here the full code of the `MCMC_fit` function, which implements the updated DRPM model described in this work. We include it to let the readers appreciate the ease, clarity and even elegance of the Julia language in translating the mathematical formulation into code. Especially compared to the original C implementation, Julia is not only more readable and efficient, being still a compiled language, but also faster in the testing and development phase, thanks to the possibility of her interactive use trough the Julia REPL, similar to the one of the R console. Moreover, Julia’s extensive ecosystem provides users flexibility and all the supports they could need, thanks to the vast documentation or even an online active forum¹ (where I wrote myself some questions during the development of the thesis). For example, packages like `Distributions` [Bes+21] [Lin+19] and `Statistics` came in very handy for this work; while the C implementation had to write all the statistical functionalities from scratch.

All this with the the hope for Julia to become the new natural choice in the statistical, or in general scientific, computing field, giving to it a refreshing approach.

Listing 3: Julia code which implements the fitting model algorithm. We report here just the “functional” part, i.e. not all the previous setup lines regarding the function definition, variable preallocations, checks on the input arguments, etc.

```
##### start MCMC algorithm #####
println("Starting MCMC algorithm")

t_start = now()
progresso = Progress(round{Int64}(draws)),
    showspeed=true,
    output=stdout, # not stderr, to make it work nicer on R
    dt=1, # every how many seconds update the feedback
    barlen=0 # no progress bar, just time feedback
)

for i in 1:draws
```

¹<https://discourse.julialang.org/>

```

##### sample the missing values #####
# from the "update rho" section onwards also the Y[j,t] will be needed (to
↪ compute weights, update laws, etc)
# so we need now to simulate the values (from their full conditional) for the
↪ data which are missing
if Y_has_NA
  # we have to use the missing_idx to remember which units and at which
  ↪ times had a missing value,
  # in order to simulate just them and instead use the given value for the
  ↪ other units and times
  for (j,t) in missing_idx
    # What if when filling a NA we occur in another NA value? eg when we
    ↪ also need Y[j,t+1]?
    # I decided here to set that value to 0, if occurs, since anyway target
    ↪ should be centered
    # so it seems a reasonable patch, and I think there was no other
    ↪ solution

    c_it = Si_iter[j,t]
    Xlk_term_t = (lk_xPPM ? dot(view(Xlk_covariates,j,:,t), beta_iter[t]) :
    ↪ 0)
    aux1 = eta1_iter[j]^2

    if t==1
      c_itp1 = Si_iter[j,t+1]
      Xlk_term_tp1 = (lk_xPPM ? dot(view(Xlk_covariates,j,:,t+1),
      ↪ beta_iter[t+1]) : 0)

      sig2_post = 1 / (1/sig2h_iter[c_it,t] +
      ↪ aux1/(sig2h_iter[c_itp1,t+1]*(1-aux1)))
      mu_post = sig2_post * (
        (1/sig2h_iter[c_it,t])*(muh_iter[c_it,t] + Xlk_term_t) +
        (eta1_iter[j]/(sig2h_iter[c_itp1,t+1]*(1-aux1))*((ismissing(Y[j,t+1])
        ↪ ? 0 : Y[j,t+1]) - muh_iter[c_itp1,t+1] - Xlk_term_tp1)
        )

      Y[j,t] = rand(Normal(mu_post,sqrt(sig2_post)))

    elseif 1<t<T
      c_itp1 = Si_iter[j,t+1]
      Xlk_term_tp1 = (lk_xPPM ? dot(view(Xlk_covariates,j,:,t+1),
      ↪ beta_iter[t+1]) : 0)

      sig2_post = (1-aux1) / (1/sig2h_iter[c_it,t] +
      ↪ aux1/sig2h_iter[c_itp1,t+1])
      mu_post = sig2_post * (
        (1/(sig2h_iter[c_it,t]*(1-aux1)))*(muh_iter[c_it,t] +
        ↪ eta1_iter[j]*(ismissing(Y[j,t-1]) ? 0 : Y[j,t-1]) +
        ↪ Xlk_term_t) +
        (eta1_iter[j]/(sig2h_iter[c_itp1,t+1]*(1-aux1))*((ismissing(Y[j,t+1])
        ↪ ? 0 : Y[j,t+1]) - muh_iter[c_itp1,t+1] - Xlk_term_tp1)
        )

      Y[j,t] = rand(Normal(mu_post,sqrt(sig2_post)))

    else # t==T
      Y[j,t] = rand(Normal(
        muh_iter[c_it,t] + eta1_iter[j]*(ismissing(Y[j,t-1]) ? 0 :
        ↪ Y[j,t-1]) + Xlk_term_t,
        sqrt(sig2h_iter[c_it,t]*(1-aux1))
      ))
    end
  end
end

```



```

end
end

for t in 1:T
##### update gamma #####
for j in 1:n
    if t==1
        gamma_iter[j,t] = 0
        # at the first time units get reallocated
    else
        # we want to find  $\rho_t^{R_t(-j)}$  ...
        indexes = findall_faster(jj -> jj != j && gamma_iter[jj, t] == 1,
            ↪ 1:n)
        Si_red = Si_iter[indexes, t]
        copy!(Si_red1, Si_red)
        push!(Si_red1, Si_iter[j,t]) # ... and  $\rho_t^{R_t(+j)}$ 

        # get also the reduced spatial info if sPPM model
        if sPPM
            sp1_red = @view sp1[indexes]
            sp2_red = @view sp2[indexes]
        end
        # and the reduced covariates info if cl_xPPM model
        if cl_xPPM
            Xcl_covariates_red = @view Xcl_covariates[indexes,:,t]
        end

        # compute n_red's and nclus_red's and relabel
        n_red = length(Si_red) # = "n" relative to here, i.e. the
            ↪ sub-partition size
        n_red1 = length(Si_red1)
        relabel!(Si_red, n_red)
        relabel!(Si_red1, n_red1)
        nclus_red = isempty(Si_red) ? 0 : maximum(Si_red) # = number of
            ↪ clusters
        nclus_red1 = maximum(Si_red1)

        # save the label of the current working-on unit j
        j_label = Si_red1[end]

        # compute also nh_red's
        nh_red .= 0
        nh_red1 .= 0
        for jj in 1:n_red
            nh_red[Si_red[jj]] += 1 # = numerosities for each cluster label
            nh_red1[Si_red1[jj]] += 1
        end
        nh_red1[Si_red1[end]] += 1 # account for the last added unit j, not
            ↪ included in the above loop

        # start computing weights
        lg_weights .= 0

        # unit j can enter an existing cluster...
        for k in 1:nclus_red
            aux_idx = findall(Si_red .== k) # fast
            lC .= 0.
            if sPPM
                # filter the spatial coordinates of the units of label k
                copy!(s1o, sp1_red[aux_idx])
                copy!(s2o, sp2_red[aux_idx])
            end
        end
    end
end
end

```

```

    copy!(s1n,s1o); push!(s1n, sp1[j])
    copy!(s2n,s2o); push!(s2n, sp2[j])
    spatial_cohesion!(spatial_cohesion_idx, s1o, s2o,
        ↪ sp_params_struct, true, M_dp, S, 1, false, lC)
    spatial_cohesion!(spatial_cohesion_idx, s1n, s2n,
        ↪ sp_params_struct, true, M_dp, S, 2, false, lC)
end
lS .= 0.
if cl_xPPM
    # and the covariates of the units of label k
    for p in 1:p_cl
        copy!(Xo, @view Xcl_covariates_red[aux_idx,p])
        copy!(Xn, Xo); push!(Xn,Xcl_covariates[j,p,t])
        covariate_similarity!(covariate_similarity_idx, Xo,
            ↪ cv_params, true, 1, true, lS)
        covariate_similarity!(covariate_similarity_idx, Xn,
            ↪ cv_params, true, 2, true, lS)
    end
end
lg_weights[k] = log(nh_red[k]) + lC[2] - lC[1] + lS[2] - lS[1]
end
# ... or unit j can create a singleton
lC .= 0.
if sPPM
    spatial_cohesion!(spatial_cohesion_idx, SVector(sp1[j]),
        ↪ SVector(sp2[j]), sp_params_struct, true, M_dp, S, 2, false,
        ↪ lC)
end
lS .= 0.
if cl_xPPM
    for p in 1:p_cl
        covariate_similarity!(covariate_similarity_idx,
            ↪ SVector(Xcl_covariates[j,p,t]), cv_params, true, 2, true,
            ↪ lS)
    end
end
lg_weights[nclus_red+1] = log_Mdp + lC[2] + lS[2]

# now use the weights towards sampling the new gamma_jt
max_ph = maximum(@view lg_weights[1:(nclus_red+1)])
sum_ph = 0.0
# exponentiate...
for k in 1:(nclus_red+1)
    # for numerical purposes we subtract max_ph
    lg_weights[k] = exp(lg_weights[k] - max_ph)
    sum_ph += lg_weights[k]
end
# ... and normalize
lg_weights ./= sum_ph

# compute probh
probh::Float64 = 0.0
if time_specific_alpha==false && unit_specific_alpha==false
    probh = alpha_iter / (alpha_iter + (1 - alpha_iter) *
        ↪ lg_weights[j_label])
elseif time_specific_alpha==true && unit_specific_alpha==false
    probh = alpha_iter[t] / (alpha_iter[t] + (1 - alpha_iter[t]) *
        ↪ lg_weights[j_label])
elseif time_specific_alpha==false && unit_specific_alpha==true
    probh = alpha_iter[j] / (alpha_iter[j] + (1 - alpha_iter[j]) *
        ↪ lg_weights[j_label])
elseif time_specific_alpha==true && unit_specific_alpha==true

```

```

        probh = alpha_iter[j,t] / (alpha_iter[j,t] + (1 -
        ↪ alpha_iter[j,t]) * lg_weights[j_label])
    end

    # compatibility check for gamma transition
    if gamma_iter[j, t] == 0
        # we want to find  $\rho_{(t-1)}^{R_{t(+j)}}$  ...
        indexes = findall_faster(jj -> jj==j gamma_iter[jj, t]==1, 1:n)
        Si_comp1 = @view Si_iter[indexes, t-1]
        Si_comp2 = @view Si_iter[indexes, t] # ... and  $\rho_{t-1}^{R_{t(+j)}}$ 

        rho_comp = compatibility(Si_comp1, Si_comp2)
        if rho_comp == 0
            probh = 0.0
        end
    end
    # sample the new gamma
    gt = rand(Bernoulli(probh))
    gamma_iter[j, t] = gt
end
end # for j in 1:n

##### update rho #####
# we only update the partition for the units which can move (i.e. with
↪ gamma_jt=0)
movable_units = findall(gamma_iter[:,t] .== 0)

for j in movable_units
    # remove unit j from the cluster she is currently in

    if nh[Si_iter[j,t],t] > 1 # unit j does not belong to a singleton
        ↪ cluster
        nh[Si_iter[j,t],t] -= 1
        # no nclus_iter[t] change since j's cluster is still alive
    else # unit j belongs to a singleton cluster
        j_label = Si_iter[j,t]
        last_label = nclus_iter[t]

        if j_label < last_label
            # here we enter if j_label is not the last label, so we need to
            # relabel clusters in order to then remove j's cluster
            # eg: units 1 2 3 4 5 j 7 -> units 1 2 3 4 5 j 7
            #      label 1 1 2 2 2 3 4      label 1 1 2 2 2 4 3

            # swap cluster labels...
            for jj in 1:n
                if Si_iter[jj, t] == last_label
                    Si_iter[jj, t] = j_label
                end
            end
            Si_iter[j, t] = last_label
            # ... and cluster-specific parameters
            sig2h_iter[j_label, t], sig2h_iter[last_label, t] =
            ↪ sig2h_iter[last_label, t], sig2h_iter[j_label, t]
            muh_iter[j_label, t], muh_iter[last_label, t] =
            ↪ muh_iter[last_label, t], muh_iter[j_label, t]
            nh[j_label, t] = nh[last_label, t]
            nh[last_label, t] = 1
        end

        # remove the j-th observation and the last cluster (being j in a
        ↪ singleton)
    end
end

```

```

    nh[last_label, t] -= 1
    nclus_iter[t] -= 1
end

# setup probability weights towards the sampling of rho_jt
ph .= 0.0
resize!(ph, nclus_iter[t]+1)
copy!(rho_tmp, @view Si_iter[:,t])

# compute nh_tmp (numerocities for each cluster label)
copy!(nh_tmp, @view nh[:,t])
# unit j contribute is already absent from the change we did above
nclus_temp = 0

# we now simulate the unit j to be assigned to one of the existing
↳ clusters...
for k in 1:nclus_iter[t]
    rho_tmp[j] = k
    indexes = findall(gamma_iter[:,t+1] .== 1)
    # we check the compatibility between rho_t^{h=k,R_(t+1)} ...
    Si_comp1 = @view rho_tmp[indexes]
    Si_comp2 = @view Si_iter[indexes,t+1] # and rho_(t+1)^{R_(t+1)}
    rho_comp = compatibility(Si_comp1, Si_comp2)

    if rho_comp != 1
        ph[k] = log(0) # assignment to cluster k is not compatible
    else
        # update params for "rho_jt = k" simulation
        nh_tmp[k] += 1
        nclus_temp = count(a->(a>0), nh_tmp)

        lPP .= 0.
        for kk in 1:nclus_temp
            aux_idx = findall(rho_tmp .== kk)
            if sPPM
                copy!(s1n, @view sp1[aux_idx])
                copy!(s2n, @view sp2[aux_idx])
                spatial_cohesion!(spatial_cohesion_idx, s1n, s2n,
                    ↳ sp_params_struct, true, M_dp, S, 1, true, lPP)
            end
            if cl_xPPM
                for p in 1:p_cl
                    Xn_view = @view Xcl_covariates[aux_idx,p,t]
                    covariate_similarity!(covariate_similarity_idx, Xn_view,
                        ↳ cv_params, true, 1, true, lPP)
                end
            end
            lPP[1] += log_Mdp + lgamma(nh_tmp[kk])
        end

        if t==1
            ph[k] = loglikelihood(Normal(
                muh_iter[k,t] + (lk_xPPM ? dot(view(Xlk_covariates,j,:,t),
                    ↳ beta_iter[t]) : 0),
                sqrt(sig2h_iter[k,t])),
                Y[j,t]) + lPP[1]
        else
            ph[k] = loglikelihood(Normal(
                muh_iter[k,t] + eta1_iter[j]*Y[j,t-1] + (lk_xPPM ?
                    ↳ dot(view(Xlk_covariates,j,:,t), beta_iter[t]) : 0),
                sqrt(sig2h_iter[k,t]*(1-eta1_iter[j]^2))),
                Y[j,t]) + lPP[1]
        end
    end
end

```

```

        end

        # restore params after "rho_jt = k" simulation
        nh_tmp[k] -= 1
    end
end
# ... plus the case of being assigned to a new (singleton for now)
↪ cluster
k = nclus_iter[t]+1
rho_tmp[j] = k
# declare (for later scope accessibility) the new params here
muh_draw = 0.0; sig2h_draw = 0.0

indexes = findall(gamma_iter[:,t+1] .== 1)
Si_comp1 = @view rho_tmp[indexes]
Si_comp2 = @view Si_iter[indexes,t+1]
rho_comp = compatibility(Si_comp1, Si_comp2)

if rho_comp != 1
    ph[k] = log(0) # assignment to a new cluster is not compatible
else
    # sample new params for this new cluster
    muh_draw = rand(Normal(theta_iter[t], sqrt(tau2_iter[t])))
    sig2h_draw = rand(InverseGamma(sig2h_priors[1], sig2h_priors[2]))

    # update params for "rho_jt = k" simulation
    nh_tmp[k] += 1
    nclus_temp = count(a->(a>0), nh_tmp)

    lPP .= 0.
    for kk in 1:nclus_temp
        aux_idx = findall(rho_tmp .== kk)
        if sPPM
            copy!(s1n, @view sp1[aux_idx])
            copy!(s2n, @view sp2[aux_idx])
            spatial_cohesion!(spatial_cohesion_idx, s1n, s2n,
                ↪ sp_params_struct, true, M_dp, S, 1, true, lPP)
        end
        if cl_xPPM
            for p in 1:p_cl
                Xn_view = @view Xcl_covariates[aux_idx,p,t]
                covariate_similarity!(covariate_similarity_idx, Xn_view,
                    ↪ cv_params, true, 1, true, lPP)
            end
        end
        lPP[1] += log_Mdp + lgamma(nh_tmp[kk])
    end

    if t==1
        ph[k] = loglikelihood(Normal(
            muh_draw + (lk_xPPM ? dot(view(Xlk_covariates,j,:),t),
                ↪ beta_iter[t]) : 0),
            sqrt(sig2h_draw)),
            Y[j,t]) + lPP[1]
    else
        ph[k] = loglikelihood(Normal(
            muh_draw + eta1_iter[j]*Y[j,t-1] + (lk_xPPM ?
                ↪ dot(view(Xlk_covariates,j,:),t), beta_iter[t]) : 0),
            sqrt(sig2h_draw*(1-eta1_iter[j]^2))),
            Y[j,t]) + lPP[1]
    end
end

```

```

    # restore params after "rho_jt = k" simulation
    nh_tmp[k] -= 1
end

# now exponentiate the weights...
max_ph = maximum(ph)
sum_ph = 0.0
for k in eachindex(ph)
    # for numerical purposes we subtract max_ph
    ph[k] = exp(ph[k] - max_ph)
    sum_ph += ph[k]
end
# ... and normalize them
ph ./= sum_ph

# now sample the new label Si_iter[j,t]
u = rand(Uniform(0,1))
cph = cumsum(ph)
cph[end] = 1 # fix numerical problems of having sums like 0.999999etc
new_label = 0
for k in eachindex(ph)
    if u <= cph[k]
        new_label = k
        break
    end
end

if new_label <= nclus_iter[t]
    # we enter an existing cluster
    Si_iter[j, t] = new_label
    nh[new_label, t] += 1
else
    # we create a new singleton cluster
    nclus_iter[t] += 1
    cl_new = nclus_iter[t]
    Si_iter[j, t] = cl_new
    nh[cl_new, t] = 1
    muh_iter[cl_new, t] = muh_draw
    sig2h_iter[cl_new, t] = sig2h_draw
end

# now we need to relabel after the possible mess created by the
↳ sampling
# eg:          (before)          (after)
#      units j 2 3 4 5 -> units j 2 3 4 5
#      labels 1 1 1 2 2      labels 3 1 1 2 2
Si_tmp = @view Si_iter[:,t]

relabel_full!(Si_tmp, n, Si_relab, nh_reorder, old_lab)
# - Si_relab gives the relabelled partition
# - nh_reorder gives the numerosities of the relabelled partition, ie
↳ "nh_reorder[k] = #(units of new cluster k)"
# - old_lab tells "the index in position i (which before was cluster i)"
↳ is now called cluster old_lab[i]"
# eg:          Original labels (Si): 4 2 1 1 1 3 1 4 5
#              Relabeled groups (Si_relab): 1 2 3 3 3 4 3 1 5
# Reordered cluster sizes (nh_reorder): 2 1 4 1 1 0 0 0 0
#              Old labels (old_lab): 4 2 1 3 5 0 0 0 0

# now fix everything (morally permute params)
Si_iter[:,t] = Si_relab
copy!(muh_iter_copy, muh_iter)

```

```

copy!(sig2h_iter_copy, sig2h_iter)
len = findlast(x -> x != 0, nh_reorder)
for k in 1:nclus_iter[t]
    muh_iter[k,t] = muh_iter_copy[old_lab[k],t]
    sig2h_iter[k,t] = sig2h_iter_copy[old_lab[k],t]
    nh[k,t] = nh_reorder[k]
end

end # for j in movable_units

##### update muh #####
if t==1
    for k in 1:nclus_iter[t]
        sum_Y = 0.0
        for j in 1:n
            if Si_iter[j,t]==k
                sum_Y += Y[j,t] - (lk_xPPM ? dot(view(Xlk_covariates,j,:,t),
                    ↪ beta_iter[t]) : 0.0)
            end
        end
        sig2_star = 1 / (1/tau2_iter[t] + nh[k,t]/sig2h_iter[k,t])
        mu_star = sig2_star * (theta_iter[t]/tau2_iter[t] +
            ↪ sum_Y/sig2h_iter[k,t])

        muh_iter[k,t] = rand(Normal(mu_star,sqrt(sig2_star)))
    end
else # t>1
    for k in 1:nclus_iter[t]
        sum_Y = 0.0
        sum_e2 = 0.0
        for j in 1:n
            if Si_iter[j,t]==k
                aux1 = 1 / (1-eta1_iter[j]^2)
                sum_e2 += aux1
                sum_Y += (Y[j,t] - eta1_iter[j]*Y[j,t-1] - (lk_xPPM ?
                    ↪ dot(view(Xlk_covariates,j,:,t), beta_iter[t]) : 0.0)) *
                    ↪ aux1
            end
        end
        sig2_star = 1 / (1/tau2_iter[t] + sum_e2/sig2h_iter[k,t])
        mu_star = sig2_star * (theta_iter[t]/tau2_iter[t] +
            ↪ sum_Y/sig2h_iter[k,t])

        muh_iter[k,t] = rand(Normal(mu_star,sqrt(sig2_star)))
    end
end

##### update sigma2h #####
if t==1
    for k in 1:nclus_iter[t]
        a_star = sig2h_priors[1] + nh[k,t]/2
        sum_Y = 0.0
        S_kt = findall(Si_iter[:,t] .== k)
        for j in S_kt
            sum_Y += (Y[j,t] - muh_iter[k,t] - (lk_xPPM ?
                ↪ dot(view(Xlk_covariates,j,:,t), beta_iter[t]) : 0.0))^2
        end

        b_star = sig2h_priors[2] + sum_Y/2
    end
end

```

```

        sig2h_iter[k,t] = rand(InverseGamma(a_star, b_star))
    end

else # t>1
    for k in 1:nclus_iter[t]
        a_star = sig2h_priors[1] + nh[k,t]/2
        sum_Y = 0.0
        S_kt = findall(Si_iter[:,t] .== k)
        for j in S_kt
            sum_Y += (Y[j,t] - muh_iter[k,t] - eta1_iter[j]*Y[j,t-1] -
                ↪ (lk_xPPM ? dot(view(Xlk_covariates,j,:,t), beta_iter[t]) :
                ↪ 0.0))^2
        end

        b_star = sig2h_priors[2] + sum_Y/2
        sig2h_iter[k,t] = rand(InverseGamma(a_star, b_star))
    end
end

##### update beta #####
# if lk_xPPM && i>=beta_update_threshold # conditional update version
if lk_xPPM
    if t==1
        sum_Y = zeros(p_lk)
        A_star = I(p_lk)/s2_beta
        for j in 1:n
            X_jt = @view Xlk_covariates[j,:,t]
            sum_Y += (Y[j,t] - muh_iter[Si_iter[j,t],t]) * X_jt /
                ↪ sig2h_iter[Si_iter[j,t],t]
            A_star += (X_jt * X_jt') / sig2h_iter[Si_iter[j,t],t]
        end
        b_star = beta0/s2_beta + sum_Y
        A_star = Symmetric(A_star)
        # Symmetric is needed for numerical problems
        # but A_star is indeed symm and pos def (by construction) so there
        ↪ is no problem
        beta_iter[t] = rand(MvNormalCanon(b_star, A_star)) # quicker and
        ↪ more accurate method
        # old method with the MvNormal and the inversion required
        # beta_iter[t] = rand(MvNormal(inv(Symmetric(A_star))*b_star,
        ↪ inv(Symmetric(A_star))))
    else
        sum_Y = zeros(p_lk)
        A_star = I(p_lk)/s2_beta
        for j in 1:n
            X_jt = @view Xlk_covariates[j,:,t]
            sum_Y += (Y[j,t] - muh_iter[Si_iter[j,t],t] -
                ↪ eta1_iter[j]*Y[j,t-1]) * X_jt / sig2h_iter[Si_iter[j,t],t]
            A_star += (X_jt * X_jt') / sig2h_iter[Si_iter[j,t],t]
        end
        b_star = beta0/s2_beta + sum_Y
        A_star = Symmetric(A_star)
        beta_iter[t] = rand(MvNormalCanon(b_star, A_star)) # quicker and
        ↪ more accurate method
        # old method with the MvNormal and the inversion required
        # beta_iter[t] = rand(MvNormal(inv(Symmetric(A_star))*b_star,
        ↪ inv(Symmetric(A_star))))
    end
end
end

```



```

##### update theta #####
aux1 = 1 / (lambda2_iter*(1-phi1_iter^2))
kt = nclus_iter[t]
sum_mu = 0.0
for k in 1:kt
    sum_mu += muh_iter[k,t]
end

if t==1
    sig2_post = 1 / (1/lambda2_iter + phi1_iter^2*aux1 + kt/tau2_iter[t])
    mu_post = sig2_post * (phi0_iter/lambda2_iter + sum_mu/tau2_iter[t] +
        ↪ (phi1_iter*(theta_iter[t+1] - (1-phi1_iter)*phi0_iter))*aux1)

    theta_iter[t] = rand(Normal(mu_post, sqrt(sig2_post)))

elseif t==T
    sig2_post = 1 / (aux1 + kt/tau2_iter[t])
    mu_post = sig2_post * (sum_mu/tau2_iter[t] + ((1- phi1_iter)*phi0_iter
        ↪ + phi1_iter*theta_iter[t-1])*aux1)

    theta_iter[t] = rand(Normal(mu_post, sqrt(sig2_post)))

else # 1<t<T
    sig2_post = 1 / ((1+phi1_iter^2)*aux1 + kt/tau2_iter[t])
    mu_post = sig2_post * (sum_mu/tau2_iter[t] +
        ↪ (phi1_iter*(theta_iter[t-1]+theta_iter[t+1]) +
        ↪ phi0_iter*(1-phi1_iter)^2)*aux1)

    theta_iter[t] = rand(Normal(mu_post, sqrt(sig2_post)))
end

##### update tau2 #####
kt = nclus_iter[t]
aux1 = 0.0
for k in 1:kt
    aux1 += (muh_iter[k,t] - theta_iter[t])^2
end
a_star = tau2_priors[1] + kt/2
b_star = tau2_priors[2] + aux1/2
tau2_iter[t] = rand(InverseGamma(a_star, b_star))

end # for t in 1:T

##### update eta1 #####
# the input argument eta1_priors[2] is already the std dev
if update_eta1
    for j in 1:n
        eta1_old = eta1_iter[j]
        eta1_new = rand(Normal(eta1_old, eta1_priors[2])) # proposal value

        if (-1 <= eta1_new <= 1)
            ll_old = 0.0
            ll_new = 0.0
            for t in 2:T
                # likelihood contribution
                ll_old += loglikelihood(Normal(
                    muh_iter[Si_iter[j,t],t] + eta1_old*Y[j,t-1] + (lk_xPPM ?
                    ↪ dot(view(Xlk_covariates,j,:,t), beta_iter[t]) : 0),
                    sqrt(sig2h_iter[Si_iter[j,t],t]*(1-eta1_old^2))
                ), Y[j,t])
            end
        end
    end
end

```

```

    ll_new += loglikelihood(Normal(
        muh_iter[Si_iter[j,t],t] + eta1_new*Y[j,t-1] + (lk_xPPM ?
        ↪ dot(view(Xlk_covariates,j,:,t), beta_iter[t]) : 0),
        sqrt(sig2h_iter[Si_iter[j,t],t]*(1-eta1_new^2))
    ), Y[j,t])
end
logit_old = aux_logit(eta1_old)
logit_new = aux_logit(eta1_new)

# prior contribution
ll_old += -log(2*eta1_priors[1]) -1/eta1_priors[1]*abs(logit_old)
ll_new += -log(2*eta1_priors[1]) -1/eta1_priors[1]*abs(logit_new)

ll_ratio = ll_new-ll_old
u = rand(Uniform(0,1))
if (ll_ratio > log(u))
    eta1_iter[j] = eta1_new # accept the candidate
    acceptance_ratio_eta1 += 1
end
end
end
end

##### update alpha #####
if update_alpha
    if time_specific_alpha==false && unit_specific_alpha==false
        # a scalar
        sumg = sum(@view gamma_iter[:,1:T])
        a_star = alpha_priors[1] + sumg
        b_star = alpha_priors[2] + n*T - sumg
        alpha_iter = rand(Beta(a_star, b_star))

    elseif time_specific_alpha==true && unit_specific_alpha==false
        # a vector in time
        for t in 1:T
            sumg = sum(@view gamma_iter[:,t])
            a_star = alpha_priors[1] + sumg
            b_star = alpha_priors[2] + n - sumg
            alpha_iter[t] = rand(Beta(a_star, b_star))
        end

    elseif time_specific_alpha==false && unit_specific_alpha==true
        # a vector in units
        for j in 1:n
            sumg = sum(@view gamma_iter[j,1:T])
            a_star = alpha_priors[1,j] + sumg
            b_star = alpha_priors[2,j] + T - sumg
            alpha_iter[j] = rand(Beta(a_star, b_star))
        end

    elseif time_specific_alpha==true && unit_specific_alpha==true
        # a matrix
        for j in 1:n
            for t in 1:T
                sumg = gamma_iter[j,t] # nothing to sum in this case
                a_star = alpha_priors[1,j] + sumg
                b_star = alpha_priors[2,j] + 1 - sumg
                alpha_iter[j,t] = rand(Beta(a_star, b_star))
            end
        end
    end
end
end
end

```

```

##### update phi0 #####
aux1 = 1/lambda2_iter
aux2 = 0.0
for t in 2:T
    aux2 += theta_iter[t] - phi1_iter*theta_iter[t-1]
end
sig2_post = 1 / ( 1/phi0_priors[2] + aux1 * (1 +
    ↪ (T-1)*(1-phi1_iter)/(1+phi1_iter)) )
mu_post = sig2_post * ( phi0_priors[1]/phi0_priors[2] + theta_iter[1]*aux1 +
    ↪ aux1/(1+phi1_iter)*aux2 )
phi0_iter = rand(Normal(mu_post, sqrt(sig2_post)))

##### update phi1 #####
# the input argument phi_priors is already the std dev
if update_phi1
    phi1_old = phi1_iter
    phi1_new = rand(Normal(phi1_old, phi1_priors)) # proposal value

    if (-1 <= phi1_new <= 1)
        ll_old = 0.0; ll_new = 0.0
        for t in 2:T
            # likelihood contribution
            ll_old += loglikelihood(Normal(
                (1-phi1_old)*phi0_iter + phi1_old*theta_iter[t-1],
                sqrt(lambda2_iter*(1-phi1_old^2))
            ), theta_iter[t])
            ll_new += loglikelihood(Normal(
                (1-phi1_new)*phi0_iter + phi1_new*theta_iter[t-1],
                sqrt(lambda2_iter*(1-phi1_new^2))
            ), theta_iter[t])
        end

        # prior contribution
        ll_old += loglikelihood(Uniform(-1,1), phi1_old)
        ll_new += loglikelihood(Uniform(-1,1), phi1_new)

        ll_ratio = ll_new-ll_old
        u = rand(Uniform(0,1))
        if (ll_ratio > log(u))
            phi1_iter = phi1_new # accept the candidate
            acceptance_ratio_phi1 += 1
        end
    end
end

##### update lambda2 #####
aux1 = 0.0
for t in 2:T
    aux1 += (theta_iter[t] - (1-phi1_iter)*phi0_iter -
    ↪ phi1_iter*theta_iter[t-1])^2
end
a_star = lambda2_priors[1] + T/2
b_star = lambda2_priors[2] + ((theta_iter[1] - phi0_iter)^2 + aux1) / 2
lambda2_iter = rand(InverseGamma(a_star,b_star))

##### save MCMC iterates #####
if i>burnin && i%thin==0

```

```

Si_out[:, :, i_out] = Si_iter[:, 1:T]
gamma_out[:, :, i_out] = gamma_iter[:, 1:T]
if time_specific_alpha==false && unit_specific_alpha==false
  # for each iterate, a scalar
  alpha_out[i_out] = alpha_iter
elseif time_specific_alpha==true && unit_specific_alpha==false
  # for each iterate, a vector in time
  alpha_out[:, i_out] = alpha_iter[1:T]
elseif time_specific_alpha==false && unit_specific_alpha==true
  # for each iterate, a vector in units
  alpha_out[:, i_out] = alpha_iter
elseif time_specific_alpha==true && unit_specific_alpha==true
  # for each iterate, a matrix
  alpha_out[:, :, i_out] = alpha_iter[:, 1:T]
end
for t in 1:T
  for j in 1:n
    sigma2h_out[j, t, i_out] = sig2h_iter[Si_iter[j, t], t]
    muh_out[j, t, i_out] = muh_iter[Si_iter[j, t], t]
  end
end
eta1_out[:, i_out] = eta1_iter
if lk_xPPM
  for t in 1:T
    beta_out[t, :, i_out] = beta_iter[t]
  end
end
theta_out[:, i_out] = theta_iter[1:T]
tau2_out[:, i_out] = tau2_iter[1:T]
phi0_out[i_out] = phi0_iter
phi1_out[i_out] = phi1_iter
lambda2_out[i_out] = lambda2_iter

##### save fitted values and metrics #####
for j in 1:n
  for t in 1:T
    muh_jt = muh_iter[Si_iter[j, t], t]
    sig2h_jt = sig2h_iter[Si_iter[j, t], t]
    X_lk_term = lk_xPPM ? dot(view(Xlk_covariates, j, :, t), beta_iter[t])
    ↪ : 0.0

    if t==1
      llike[j, t, i_out] = logpdf(Normal(
        muh_jt + X_lk_term,
        sqrt(sig2h_jt)
      ), Y[j, t])
      fitted[j, t, i_out] = muh_jt + X_lk_term
    else # t>1
      llike[j, t, i_out] = logpdf(Normal(
        muh_jt + eta1_iter[j]*Y[j, t-1] + X_lk_term,
        sqrt(sig2h_jt*(1-eta1_iter[j]^2))
      ), Y[j, t])
      fitted[j, t, i_out] = muh_jt + eta1_iter[j]*Y[j, t-1] + X_lk_term
    end

    mean_likelhd[j, t] += exp(llike[j, t, i_out])
    mean_loglikelhd[j, t] += llike[j, t, i_out]
    CPO[j, t] += exp(-llike[j, t, i_out])
  end
end
end

```

```

        i_out += 1
    end

next!(progresso)

end # for i in 1:draws

println("\ndone!")
t_end = now()
println("Elapsed time: ",
    ↪ Dates.canonicalize(Dates.CompoundPeriod(t_end-t_start)))

##### compute LPML and WAIC #####
for j in 1:n
    for t in 1:T
        LPML += log(CPO[j,t])
    end
end
LPML -= n*T*log(nout) # scaling factor
LPML = -LPML # fix sign
println("LPML: ", LPML, " (the higher the better)")

# adjust mean variables
mean_likelhd ./= nout
mean_loglikelhd ./= nout
for j in 1:n
    for t in 1:T
        WAIC += 2*mean_loglikelhd[j,t] - log(mean_likelhd[j,t])
    end
end
WAIC *= -2
println("WAIC: ", WAIC, " (the lower the better)")

if update_eta1 @printf "acceptance ratio eta1: %.2f%%\n"
    ↪ acceptance_ratio_eta1/(n*draws) *100 end
if update_phi1 @printf "acceptance ratio phi1: %.2f%%"
    ↪ acceptance_ratio_phi1/draws*100 end
println()

close(log_file)

if simple_return
    return Si_out, LPML, WAIC
else
    return Si_out, Int.(gamma_out), alpha_out, sigma2h_out, muh_out, include_eta1
    ↪ ? eta1_out : NaN, lk_xPPM ? beta_out : NaN, theta_out, tau2_out,
    ↪ phi0_out, include_phi1 ? phi1_out : NaN, lambda2_out, fitted, llike,
    ↪ LPML, WAIC
end

```

B.2 Interface

Now some more technical details about the whole implementation design. The fitting code was written in Julia, but its main intended use is from R. This choice was made because R is currently the best language for statistical purposes, or at least the most spread; in fact all other models “competitors” to the DRPM are available through some R package. Therefore we thought that letting our work be

also accessible from R, and not only from Julia, would have eased the possibilities of testing and comparisons with other models or datasets.

To do so, we relied on the `JuliaConnectoR` library [LHB22] on R, which allows the interaction between the two languages. In this sense, we can load in R the Julia package JDRPM, which stores the Julia project (i.e. all codes and packages dependencies) about the improved version of the DRPM model, and call it using data and arguments from R. The output produced by the Julia functions has then to be converted back into R structures, which is done automatically through a dedicated function of the package. The structure of this workflow is summarized in Listing 4.

Listing 4: Instructions to use the JDRPM model from R.

```
##### Requirements
# install the required package
install.packages("JuliaConnectoR")

##### Setup
# load the package
library(JuliaConnectoR)
# check it returns TRUE
juliaSetupOk()

# load the Package manager on Julia
juliaEval("using Pkg")
# enter into the JDRPM project
juliaEval("Pkg.activate(\"<path/to/where/you/stored/JDRPM>\")")

# downloads and install, only once, all the dependencies
juliaEval("Pkg.instantiate()")
# now, as a check, this should print the list of packages that JDRPM uses,
# such as Distributions, Statistics, LinearAlgebra, SpecialFunctions, etc.
juliaEval("Pkg.status()")

# locate the "main" file
module = normalizePath("<path/to/where/you/stored/JDRPM>/src/JDRPM.jl")
# load the "main" file into a callable R object
module_JDRPM = juliaImport(juliaCall("include", module))

##### Fit
# perform the fit
out = module_JDRPM$MCMC_fit(Y=..., sp_coords=..., ...)
# convert the output to R structures
rout = juliaGet(out)
names(rout) = c("Si", "gamma", "alpha", "sigma2h", "muh", "eta1", "beta", "theta",
  ↪ "tau2", "phi0", "phi1", "lambda2", "fitted", "llike", "lpml", "waic")
# and reshape it to uniform to the DRPM output
rout$Si      = aperm(rout$Si,      c(2, 1, 3))
rout$gamma   = aperm(rout$gamma,   c(2, 1, 3))
rout$sigma2h = aperm(rout$sigma2h, c(2, 1, 3))
rout$muh     = aperm(rout$muh,     c(2, 1, 3))
rout$fitted  = aperm(rout$fitted,  c(2, 1, 3))
rout$llike   = aperm(rout$llike,   c(2, 1, 3))
rout$alpha   = aperm(rout$alpha,   c(2, 1))
rout$theta   = aperm(rout$theta,   c(2, 1))
rout$tau2    = aperm(rout$tau2,    c(2, 1))
```

```

rout$eta1    = aperm(rout$eta1,    c(2, 1))
rout$phi0    = matrix(rout$phi0,   ncol = 1)
rout$phi1    = matrix(rout$phi1,   ncol = 1)
rout$lambda2 = matrix(rout$lambda2, ncol = 1)

```

Regarding instead the “visual” interface, in the sense of the feedback provided by the function, we decided to make it more user-friendly, and possibly informative, than the original C code. The most relevant add-on is probably the progress monitoring, which indicates and updates in real-time the estimated remaining time to complete the fit (i.e. the ETA, estimated time of arrival). As a further proof of the ease of this language, this feature was a simple insertion of few lines of code thanks to the Julia package `ProgressMeter`.

Listing 5: Feedback from the Julia implementation of the updated DRPM.

```

Parameters:
sig2h ~ invGamma(0.01, 0.01)
Logit(1/2(eta1+1)) ~ Laplace(0, 0.9)
tau2 ~ invGamma(1.0, 1.5)
phi0 ~ Normal( $\mu$ =0.0,  $\sigma$ =10.0)
lambda2 ~ invGamma(0.1, 0.1)
alpha ~ Beta(2.0, 2.0)

- using seed 113.0 -
fitting 100000 total iterates (with burnin=60000, thinning=40)
thus producing 1000 valid iterates in the end

on n=105 subjects
for T=12 time instants

with space? true
with covariates in the likelihood? false
with covariates in the clustering process? false
are there missing data in Y? false

Starting MCMC algorithm
Progress: 100% Time: 0:41:05 (24.65 ms/it)

done!
Elapsed time: 41 minutes, 5 seconds, 500 milliseconds
LPML: 727.1743696442145 (the higher the better)
WAIC: -1957.1501437043987 (the lower the better)
acceptance ratio eta1: 69.95%
acceptance ratio phi1: 76.60%

```


Appendix C

Further plots

Bibliography

- [Bez+17] Jeff Bezanson et al. “Julia: A fresh approach to numerical computing”. In: *SIAM Review* 59.1 (2017), pp. 65–98. DOI: 10.1137/141000671. URL: <https://epubs.siam.org/doi/10.1137/141000671> (cit. on p. 9).
- [CR16] Jiahao Chen and Jarrett Revels. “Robust benchmarking in noisy environments”. In: *arXiv e-prints*, arXiv:1608.04295 (Aug. 2016). arXiv: 1608.04295 [cs.PF] (cit. on p. 10).
- [PQD22] Garrit L. Page, Fernando A. Quintana, and David B. Dahl. “Dependent Modeling of Temporal Sequences of Random Partitions”. In: *Journal of Computational and Graphical Statistics* 31.2 (2022), pp. 614–627. DOI: 10.1080/10618600.2021.1987255. eprint: <https://doi.org/10.1080/10618600.2021.1987255>. URL: <https://doi.org/10.1080/10618600.2021.1987255> (cit. on p. 17).
- [Fas+23] A. Fassò et al. *AgrImOnIA: Open Access dataset correlating livestock and air quality in the Lombardy region, Italy (3.0.0)*. 2023. DOI: <https://doi.org/10.5281/zenodo.7956006> (cit. on p. 21).
- [Bes+21] Mathieu Besançon et al. “Distributions.jl: Definition and Modeling of Probability Distributions in the JuliaStats Ecosystem”. In: *Journal of Statistical Software* 98.16 (2021), pp. 1–30. ISSN: 1548-7660. DOI: 10.18637/jss.v098.i16. URL: <https://www.jstatsoft.org/v098/i16> (cit. on p. 37).
- [Lin+19] Dahua Lin et al. *JuliaStats/Distributions.jl: a Julia package for probability distributions and associated functions*. July 2019. DOI: 10.5281/zenodo.2647458. URL: <https://doi.org/10.5281/zenodo.2647458> (cit. on p. 37).
- [LHB22] Stefan Lenz, Maren Hackenberg, and Harald Binder. “The JuliaConnector: A Functionally-Oriented Interface for Integrating Julia in R”. In: *Journal of Statistical Software* 101.6 (2022), pp. 1–24. DOI: 10.18637/jss.v101.i06 (cit. on p. 52).

Acknowledgements

Ringraziamenti

“Ecco il mio ponte d’approdo per molti lunghi anni, il mio angoletto, nel quale faccio il mio ingresso con una sensazione così diffidente, così morbosa... Ma chi lo sa? Forse, quando tra molti anni mi toccherà abbandonarlo, magari potrei anche rimpiangerlo!”

— Fëdor Dostoevskij, *Memorie da una casa di morti*