

Politecnico di Milano

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

MASTER THESIS

forse qui non starebbe bene un verbo? ad esempio
giving more flexibility...
adding more flexibility...
granting/providing/ecc more flexibility...

The DRPM Strikes Back: More Flexibility for 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

Clustering is a key technique for identifying patterns and structures in complex datasets, whose relevance is intensified in spatio-temporal contexts where observations are simultaneously influenced by multiple factors such as space, time, and covariates. This complexity can be effectively tamed by model-based clustering methods, which often provide more accurate and interpretable results with respect to traditional frequentist approaches thanks to the possibility of encoding data information directly inside the model. To this end, the Dependent Random Partition Model [PQD22] is one of the most relevant Bayesian models due to its explicit consideration of temporal dependence in the partitions. However, the current formulation of the model and the implementation of the associated MCMC algorithm lack of the inclusion of covariates, the handling of missing data, and the efficiency in execution times. Therefore, in this work we improve the original DRPM by addressing those issues trough updates on the model formulation and a brand new implementation in Julia. These advancements are then tested on synthetic and real-world datasets, including air quality data from the AgrImOnIA project [Fas+23] in Lombardy, Italy.

KEYWORDS: Bayesian modelling, clustering, spatio-temporal data, MCMC, Julia

Sommario

Il clustering è una tecnica fondamentale per identificare strutture e pattern in dataset complessi, la cui importanza è intensificata nei contesti spazio-temporali in cui le osservazioni sono influenzate simultaneamente da molteplici fattori come spazio, tempo e covariate. Questa complessità può essere efficacemente gestita da metodi di clustering basati su modelli, che spesso forniscono risultati più precisi e interpretabili rispetto agli approcci frequentisti tradizionali grazie alla possibilità di inserire informazioni riguardo ai dati direttamente all'interno del modello. In tal senso, il Dependent Random Partition Model [PQD22] (modello per partizioni aleatorie dipendenti) è uno dei modelli bayesiani più rilevanti in quanto tiene conto in modo esplicito della dipendenza temporale delle partizioni. Tuttavia, l'attuale formulazione del modello e la sua corrispondente implementazione dell'algoritmo di campionamento mancano dell'inclusione di covariate, della gestione dei dati mancanti, e di efficienza nei tempi di esecuzione. In questo lavoro abbiamo quindi migliorato l'originale DRPM affrontando tali problemi tramite aggiornamenti sulla formulazione del modello e una fiammante implementazione in Julia. Questi sviluppi sono stati poi testati su dataset sintetici e reali, compresi i dati sulla qualità dell'aria in Lombardia del progetto AgrImOnIA [Fas+23].

PAROLE CHIAVE: modellistica bayesiana, clustering, dati spazio-temporali, MCMC, Julia

Contents

1 Description of the model	5
1.1 MCMC sampling algorithm	9
1.2 Spatial cohesions analysis	12
1.3 Covariates similarities analysis	18
2 Implementation and optimizations	23
2.1 Optimizations	24
2.1.1 Optimizing spatial cohesions	26
2.1.2 Optimizing covariates similarities	28
3 Testing	31
3.1 Assessing the equivalence of the models	31
3.1.1 Target variable only	32
3.1.2 Target variable plus space	33
3.2 Performance with missing values	40
3.2.1 Target variable only (NA case)	40
3.2.2 Target variable plus space (NA case)	42
3.3 Effects of the covariates	44
3.3.1 Covariates in the likelihood	45
3.3.2 Covariates in the clustering	50
3.4 Scaling performances	55
4 Conclusion	61
A Theoretical details	63
A.1 Extended computations of the full conditionals	63
B Computational details	73
B.1 Fitting algorithm code	73

B.2 Interface	88
C Fits interpretation	93
Bibliography	95

List of Figures

1.1	Updated DRPM model graph	8
1.2	Partition considered for the cohesion analysis	14
1.3	Cohesions 1, 2, and 3 illustration	16
1.4	Cohesions 4, 5, and 6 illustration	17
1.5	Partition considered for the similarity analysis	19
1.6	Similarities 1, 2 and 3 illustration	20
1.7	Similarity 4 illustration	21
2.1	Flame graph of a test fit	24
2.2	Cohesions 3 and 4 implementation comparison	27
2.3	Similarity 4 annotations comparison	29
3.1	Lagged ARI values of CDRPM and JDRPM fits, target values only	32
3.2	Clusters produced by JDRPM and CDRPM fits, target values only	33
3.3	Visual representation of the clusters of JDRPM and CDRPM fits, target values only	34
3.4	Generated and fitted values of JDRPM and CDRPM fits, target values only	35
3.5	Comparison of the two possible mean centering methods on the target variable	36
3.6	Lagged ARI values of CDRPM and JDRPM fits, target plus space values	37
3.7	Target and fitted values of JDRPM and CDRPM fits, target plus space values	38
3.8	Clusters generated by CDRPM and JDRPM fits, target plus space values	39
3.9	Fitted values of JDRPM fit, target values only, NA dataset	40
3.10	Clusters produced by JDRPM fits, target values only, full vs NA dataset	41

3.11 Lagged ARI values of JDRPM fits, target values only, full vs NA dataset	41
3.12 Visual representation of the clusters of JDRPM fit, target values only, NA dataset	42
3.13 Lagged ARI values of JDRPM fits, target plus space values, full vs NA dataset	43
3.14 Fitted values of JDRPM fit, target plus space values, NA dataset	43
3.15 Comparison of the two possible mean centering methods on a covariate	44
3.16 Lagged ARI values of JDRPM fit, target plus space values, full vs NA dataset, with covariates in the likelihood	45
3.17 Regression vector of the fit with multiple covariates in the likelihood, full dataset	46
3.18 Regression vector of the fit with multiple covariates in the likelihood, NA dataset	47
3.19 Clusters generated by JDRPM fits, target plus space values, full vs NA dataset, with covariates in the likelihood	48
3.20 Target and fitted values of JDRPM fits, target plus space values, NA dataset, with covariates in the likelihood	49
3.21 Trace plot of the fitted values for a fit with covariates in the likelihood	49
3.22 Variables used for the fit tests with covariates in the clustering	51
3.23 Lagged ARI values of JDRPM fits with vs without covariates in the clustering	52
3.24 Comparison of the clusterings provided by the base fits plus JDRPM with covariates in the clustering	53
3.25 Clusters generated by JDRPM fit, target plus space values, with covariates in the clustering process	54
3.26 CDRPM standard fit, clusters distribution with respect to the wind speed covariate	55
3.27 JDRPM fit with covariates, clusters distribution with respect to the wind speed covariate	56
3.28 JDRPM standard fit, clusters distribution with respect to the wind speed covariate	56
3.29 Execution times of JDRPM and CDRPM fits, target values only	57
3.30 Execution times of JDRPM and CDRPM fits, target plus space values	58
3.31 Execution times of JDRPM fits, target plus space plus covariates	58
3.32 Visual representation of all fitting performances	59

List of Tables

3.1	Accuracy metrics of CDRPM and JDRPM fits, target values only	32
3.2	Accuracy metrics of CDRPM and JDRPM fits, target plus space values	37
3.3	Accuracy metrics of JDRPM fits, target values only, full vs NA dataset	41
3.4	Accuracy metrics of JDRPM fits, target plus space values, full vs NA dataset	42
3.5	Accuracy metrics of JDRPM fits, target plus space values, full vs NA dataset, with vs without covariates in the likelihood	45
3.6	Accuracy metrics of CDRPM and JDRPM fits, target plus space values, with vs without covariates in the clustering process	52

Introduction

Clustering is the canonical example of unsupervised learning, where a set of data points has to be divided into homogenous groups of units which exhibit a similar behaviour with respect to a target variable. It has always been a powerful tool to identify structures and patterns into data, especially in contexts where relationships between the observations are complex, e.g. when the target variable is affected by many factors simultaneously. For this reason, clustering techniques saw a continuous increase in popularity in a variety of scientific fields, including social sciences, climate and environmental analysis, economics, and healthcare.

In general, clustering approaches fall into two main categories: algorithmic and model-based methods.

Algorithmic methods such as hierarchical, partition-based, or density-based methods treat the clustering problem as an optimization problem where a certain metric has to be minimised (or maximised). Partition-based methods as k -means generate the clusters around a set of centroids which are iteratively updated to minimize the within-cluster variance, i.e. the mean squared distance of the units from the cluster centroid. However, this method requires the desired number k of clusters to be set in advance and can only work on numerical data. Hierarchical clustering methods, on the other hand, build a tree of clustering solutions, represented as a dendrogram, which highlights the relationships among the possible clusters. This is done either through an agglomerative (bottom-up) strategy, where each unit starts in her own cluster and gets iteratively combined with other units and clusters, or in a divisive (top-down) strategy, where units start together in a single cluster which is then iteratively subdivided into smaller ones. This method allows for the number of clusters to be unspecified a priori, but both strategies become highly sensitive to the choice of the distance metric, responsible for all the merge and split operations. Density based methods as DBSCAN define instead a density metric, which can provide less constraints in the structure of the generated clusters, frequently resulting in weird shapes and less interpretable configurations. Moreover, the density metric is again sensitive to the choice of the density parameters. In summary, all these algorithmic approaches are largely heuristic and are suited for specific cases such as well-separated groups and standard geometric forms, while less for more complex scenarios of overlapping clusters. Moreover, lacking a solid statistical basis, can often lead to unsatisfying results and do not provide assessments about the clustering uncertainties.

An alternative and more flexible approach is therefore proposed by model-based methods, which assume a probabilistic modelling of the data. This is typically done

trough a mixture model, where each mixture component corresponds to a cluster with its specific cluster parameters [Wad23] [GMR22]. In this way, each observation is assumed to have arisen from one of J possible groups which are mixed together in various proportions. More formally, for each unit $i = 1, \dots, n$ we have that

$$f(y_i | \boldsymbol{\pi}, \boldsymbol{\vartheta}_1^*, \dots, \boldsymbol{\vartheta}_J^*) = \sum_{j=1}^J \pi_j f_j(y_j | \boldsymbol{\vartheta}_j^*) \quad (1)$$

where y_i is the data point of the i -th observation, $\boldsymbol{\pi}$ is the set of weights satisfying $\pi_j \in [0, 1]$ for $j = 1, \dots, J$ and $\sum_{j=1}^J \pi_j = 1$, $\boldsymbol{\vartheta}_j^*$ are the cluster-specific parameters, and $f_j(\cdot | \boldsymbol{\vartheta}_j^*)$ is the density of the j -th cluster. A common modelling choice is a gaussian mixture model (GMM), where each cluster follows a normal distribution and thus making $\boldsymbol{\vartheta}_j^* = (\mu_j^*, \sigma_j^{2*})$, or $\boldsymbol{\vartheta}_j^* = (\boldsymbol{\mu}_j^*, \Sigma_j^*)$ in the multivariate case. This choice is flexible and effective since, especially in the multivariate case, gaussian distribution are able to capture very different clustering structures. Anyway, in this model-based approach, the goal is to estimate the cluster-specific parameters $\boldsymbol{\vartheta}_1, \dots, \boldsymbol{\vartheta}_J$ and the mixing proportions π_1, \dots, π_J . The estimation step is often performed trough the Expectation-Maximization (EM) algorithm, which iteratively refines the estimates of the parameters via maximum likelihood estimation (MLE). Once the cluster-specific parameters are estimated, each observation can be assigned to a component, i.e. to a cluster, based on the highest posterior probability of belonging to that component, which can be computed trough the Bayes rule.

This approach of mixture model can be naturally moved in a Bayesian framework where to each parameter is treated as a random variable with a corresponding prior distribution [Fra08]. This leads (1) to be reformulated into

$$\begin{aligned} y_i | c_i, \boldsymbol{\pi}, \boldsymbol{\vartheta}_1^*, \dots, \boldsymbol{\vartheta}_J^* &\stackrel{\text{iid}}{\sim} f_{c_i}(y_i | \boldsymbol{\vartheta}_{c_i}^*) \\ c_1, \dots, c_n &\sim \text{Cat}(\pi_1, \dots, \pi_J) \\ \boldsymbol{\vartheta}_1^*, \dots, \boldsymbol{\vartheta}_J^* &\stackrel{\text{iid}}{\sim} F_0 \\ \pi_1, \dots, \pi_J &\sim \text{Dir}(\gamma, \dots, \gamma) \end{aligned} \quad (2)$$

where the cluster labels c_1, \dots, c_n are assigned a multinomial distribution with probabilities given by the vector of weights $\boldsymbol{\pi}$, the cluster-specific parameters $\boldsymbol{\vartheta}_j^*$ are assigned a prior distribution F_0 , while the weights are assigned a Dirichlet distribution, where e.g. all parameters are the same $\gamma_j = \gamma$ if no prior information about cluster assignments is wished to be inserted into the model.

The Bayesian framework is however much powerful and allows for even more complex formulations. In fact, moving to a Bayesian non-parametric approach, an infinite mixture model, i.e. that does not require a predetermined number J of components, can be introduced. This extension often relies on the Dirichlet Process (DP) [Fer73], which leads to a model formulation in the form

$$\begin{aligned} y_i | \boldsymbol{\vartheta}_i &\stackrel{\text{ind}}{\sim} f(y_i | \boldsymbol{\vartheta}_i) \\ \boldsymbol{\vartheta}_1, \dots, \boldsymbol{\vartheta}_n | P &\stackrel{\text{iid}}{\sim} P \\ P &\sim \text{discrete RPM} \end{aligned}$$

where RPM denotes a random probability measure. The discreteness of P implies the presence of ties among the atoms of the process $\boldsymbol{\vartheta}_1, \dots, \boldsymbol{\vartheta}_n$. These ties therefore induce a partition ρ_n which can be identified by the units manifesting the same values of the parameter $\boldsymbol{\vartheta}_i$. That is, denoting with $\boldsymbol{\vartheta}_1^*, \dots, \boldsymbol{\vartheta}_K^*$ the unique values of $\boldsymbol{\vartheta}_1, \dots, \boldsymbol{\vartheta}_n$, the generic h -cluster can be defined as $S_h = \{i \in \{1, \dots, n\} : \boldsymbol{\vartheta}_i = \boldsymbol{\vartheta}_h^*\}$. The Dirichlet Process plays a significant role in general in Bayesian nonparametrics [Gra23], but especially in clustering, thanks to its computationally-friendly implementation through the stick-breaking representation [Teh10], the Pólya urn representation [BM73], and the Chinese restaurant process (CRP) [Ald85].

This increased flexibility in the clustering modelling becomes even more powerful when working on spatio-temporal datasets, in which observations are collected over time and across different spatial locations, possibly concealing trends behind both information levels. This type of data, in fact, is inherently complex due to this interaction between spatial and temporal dimensions; a complexity that is further increased if covariates are also available. As a result, model-based methods are fairly more suited to tackle this task, rather than traditional algorithmic methods, since they can combine all these different levels of information. A model-based analysis of such data should also be able to account for the temporal dependency among the partitions, in order to extract possible trends occurring in time, resulting in more gentle and interpretable temporal evolution of the clusters, while also granting efficient implementations to be possibly applied on large scale datasets which are commonly accessible in this context.

Recently, the use of Bayesian models to perform clustering has become more popular, particularly in this field of spatio-temporal datasets. Bayesian clustering, in fact, allows to incorporate prior information into the model enhancing the flexibility and interpretability of the results with respect, for example, to more traditional frequentist approaches. Throughout the years, several models have been proposed in the Bayesian literature, but one of the most relevant is the Dependent Random Partition Model [PQD22] which stands out for being able to handle explicitly the temporal dependence of partitions into the model formulation, while also possibly accounting for the spatial information. However, the current DRPM implementation of the MCMC algorithm, written in C and available through an R interface, lacks some relevant utilities such as the inclusion of covariates, which could further improve the generation and informativity of the clusters, the handling of missing data, and an efficient implementation, which would speed up the model fitting to run multiple chains in parallel, or be more easily applied on large scale datasets.

In this work, we aim to address these three issues by making the original model more flexible. We will show how our updates can indeed be effective and also provide faster execution times. In fact, implementing the model using the Julia language, rather than C, we took advantage of its high-performance capabilities and well-equipped statistical ecosystem. Our comparison will focus on both synthetic and real-world datasets, with the latter case involving air quality measurements from the AgrImOnIA project [Fas+23], a comprehensive record of air pollutant levels and other environmental variables measured across the Lombardy region of Italy.

Chapter 1 briefly reviews the literature on Bayesian clustering models for spatio-temporal data, and then dives deeply into the analysis and description of the DRPM and of our generalization.

Chapter 2 provides insights about the computational aspects of the MCMC algorithm, motivating the choice of the Julia language and reporting some optimization possibilities emerged when developing the implementation.

Chapter 3 focuses on testing the original DRPM formulation and our generalized model. We firstly evaluate if they perform similarly at a common testing level, i.e. under the same data, hyperparameters values, and MCMC iterations setup. Then, we assess the performances of our updates by considering fits involving missing data and fits including covariates. A comparative analysis about execution times with respect to the size of the dataset and the type of the fit will also be provided.

Finally, in Chapter 4, we briefly review the strengths and drawbacks this work revealed and suggest possible further improvements.

Chapter 1

Description of the model

“Come on, gentlemen, why shouldn’t we get rid of all this calm reasonableness with one good 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*

In the Bayesian framework, clustering is possible by employing a random probability measure of discrete type that induces a distribution over random partitions. This discreteness is obtained with the Dirichlet Process (DP), which several clustering models implement either through the stick-breaking representation [Bar+12] [AW15] [GMR16] [Jo+16] [KG18] [DK18] [De +19] or through the Pólya urn scheme [Car+17]. However, these classical Bayesian methods rely on modelling the dependence in the random partitions by modelling the dependence inside the random probability measures, i.e. on the parameters which underlie those DP representations rather than to the clusters themselves. This approach is therefore kind of a “step back” from the main object of interest, the clusters, which are then only *induced* by the random partition model. As a consequence, there is no guarantee that the correlation that appears in the parameters would subsequently reflect into correlation among the partitions, often producing counterintuitive behaviours in the results. The Dependent Random Partition Model (DRPM) [PQD22], on the other hand, models *directly* the sequence of partitions, thus providing a more reasonable, accurate, and interpretable temporal evolution of the clusters.

Before diving into the model description, we define some notation conventions. We setup in a spatio-temporal context with $i = 1, \dots, n$ and $t = 1, \dots, T$ being the indexes for units and time instants. We will denote with $\rho_t = \{S_{1t}, \dots, S_{kt}\}$ the partition at time t , of the n experimental units, composed by k_t cluster. Another possible representation of the partition is through cluster membership labels $\mathbf{c}_t = \{c_{1t}, \dots, c_{nt}\}$, where $c_{it} = j$ if unit i belongs to cluster S_{jt} . Finally, we will denote with a \star superscript all the variables or quantities which are cluster-specific.

To implement dependence in the partitions one could simply propose a joint probability model for (ρ_1, \dots, ρ_T) , denoted as $P(\rho_1, \dots, \rho_T)$, where each ρ_t is set to be possibly affected by all the other partitions. This principle, however, it’s far

too complex and general to be modelled efficiently; therefore the DRPM authors limited this temporal connection to a first-order Markov-chain structure, where the conditional distribution of ρ_t given all the predecessors $\rho_{t-1}, \rho_{t-2}, \dots, \rho_1$ actually depends only on ρ_{t-1} . This brings the random partition model to the form

$$P(\rho_1, \dots, \rho_T) = P(\rho_T | \rho_{T-1}) \cdots P(\rho_2 | \rho_1) P(\rho_1) \quad (1.1)$$

To explicitly manage the relation between ρ_t and ρ_{t-1} some auxiliary variables are introduced. The idea is that if two partitions are highly time-dependent, few changes will occur between them. In turn, partitions which are quite independent will possibly exhibit very different configurations. To express this fixity or flexibility concept, for each unit $i = 1, \dots, n$ the following variable is introduced

$$\gamma_{it} = \begin{cases} 1 & \text{if unit } i \text{ is } not \text{ reallocated when moving from time } t-1 \text{ to } t \\ 0 & \text{otherwise (i.e. unit } i \text{ is reallocated)} \end{cases} \quad (1.2)$$

By construction, we set $\gamma_{i1} = 0$ for all i , meaning that at the first time instant all units will get reallocated since they have no partition to which they could be possibly fixed at. Regarding their modelling, the authors proposed $\gamma_{it} \stackrel{\text{ind}}{\sim} \text{Ber}(\alpha_t)$ with $\alpha_t \in [0, 1]$ behaving as a temporal dependence parameter. At the two extremes, $\alpha_t = 1$ will denote perfect temporal dependence, with $\rho_t = \rho_{t-1}$, while $\alpha_t = 0$ will imply full independence of ρ_t from ρ_{t-1} . The parameter α_t can actually be global, and we will write simply α without any subscripts, or instead time and/or unit specific, offering the cases of α_t , α_i , or α_{it} . For the sake of clarity, the vector $\boldsymbol{\gamma}_t = (\gamma_{1t}, \dots, \gamma_{nt})$ is created, and the augmented joint model becomes in the form

$$P(\boldsymbol{\gamma}_1, \rho_1, \dots, \boldsymbol{\gamma}_T, \rho_T) = P(\rho_T | \boldsymbol{\gamma}_T, \rho_{T-1}) P(\boldsymbol{\gamma}_T) \cdots P(\rho_2 | \boldsymbol{\gamma}_2, \rho_1) P(\boldsymbol{\gamma}_2) P(\rho_1) \quad (1.3)$$

The insertion of these additional variables makes the model very powerful in describing the temporal dependence of the partitions but slightly hinders the design of the sampling algorithm. To outline it, we firstly need a

Definition 1.1 (compatibility). Two partitions ρ_t and ρ_{t-1} are *compatible* with respect to $\boldsymbol{\gamma}_t$ if ρ_t can be obtained from ρ_{t-1} by reallocating items as indicated by $\boldsymbol{\gamma}_t$; i.e. only moving the units i with $\gamma_{it} = 0$.

To perform this compatibility check, it is enough to ensure that the reduced partitions from ρ_t and ρ_{t-1} are the same, with reduced meaning their restriction to the units which cannot move. Indeed, if those fixed units are clustered in the same ways, then surely the free-movers from ρ_t can be set to match the labels assigned by partition ρ_{t-1} . Denoting as $\mathfrak{R}_t = \{i : \gamma_{it} = 1\}$ the set of fixed units at time t , this check translates into asking that $\rho_t^{\mathfrak{R}_t} = \rho_{t-1}^{\mathfrak{R}_t}$.

The sampling algorithm requires that when we are drawing the new samples for the γ_{ita} , or also for the cluster labels c_{it} , we firstly need to check if those draws can actually be valid, i.e. if they would keep compatible and coherent all the partitions and parameters involved. For example, when updating γ_{it} during each iteration d of the algorithm, the only case which can raise problems is when we pass from

$\gamma_{it}^{(d-1)} = 0$ to $\gamma_{it}^{(d)} = 1$. This step corresponds to the case in which a unit i that was initially (i.e. according to the previous iteration parameters) free to be reassigned is now instead deemed to stay fixed in her cluster. However, this change may not align to the current sampled values of the partitions $\rho_{t-1}^{(d)}$ and $\rho_t^{(d-1)}$. Therefore, compatibility between their reductions to the units in the set $\mathfrak{R}_t \cup \{i\}$ needs to be checked and, if this check fails, the tentative update $\gamma_{it}^{(d-1)} = 0 \rightarrow \gamma_{it}^{(d)} = 1$ is not allowed to be performed and we force $\gamma_{it}^{(d)} = 0$ in the sampling algorithm. Similar checks are conducted when ρ_t is updated. In this step, only the units that can actually move, i.e. that have $\gamma_{it} = 0$, are updated, and therefore there are no compatibility problems between ρ_{t-1} and ρ_t . However, since the update of γ_{it} occurs before the one of the partition, compatibility needs to be checked between ρ_t and ρ_{t+1} .

In any case, once the partition model is specified, there is great flexibility in how to setup the rest of the hierarchical model. To allow temporal dependence to propagate trough the model, an autoregressive AR(1) component is also added to the formulation of the model (but only optionally in the implementation), both at the data and cluster-specific parameters level. All this led the authors to the following complete model

$$\begin{aligned}
Y_{it}|Y_{it-1}, \boldsymbol{\mu}_t^*, \boldsymbol{\sigma}_t^{2*}, \boldsymbol{\eta}, \mathbf{c}_t &\stackrel{\text{ind}}{\sim} \mathcal{N}(\mu_{c_{it}t}^* + \eta_{1i}Y_{it-1}, \sigma_{c_{it}t}^{2*}(1 - \eta_{1i}^2)) \\
Y_{i1} &\stackrel{\text{ind}}{\sim} \mathcal{N}(\mu_{c_{i1}1}^*, \sigma_{c_{i1}1}^{2*}) \\
\xi_i = \text{Logit}(\tfrac{1}{2}(\eta_{1i} + 1)) &\stackrel{\text{ind}}{\sim} \text{Laplace}(a, b) \\
(\mu_{jt}^*, \sigma_{jt}^*) &\stackrel{\text{ind}}{\sim} \mathcal{N}(\vartheta_t, \tau_t^2) \times \mathcal{U}(0, A_\sigma) \\
\vartheta_t|\vartheta_{t-1} &\stackrel{\text{ind}}{\sim} \mathcal{N}((1 - \varphi_1)\varphi_0 + \varphi_1\vartheta_{t-1}, \lambda^2(1 - \varphi_1^2)) \\
(\vartheta_1, \tau_t) &\stackrel{\text{iid}}{\sim} \mathcal{N}(\varphi_0, \lambda^2) \times \mathcal{U}(0, A_\tau) \\
(\varphi_0, \varphi_1, \lambda) &\sim \mathcal{N}(m_0, s_0^2) \times \mathcal{U}(-1, 1) \times \mathcal{U}(0, A_\lambda) \\
\{\mathbf{c}_t, \dots, \mathbf{c}_T\} &\sim \text{tRPM}(\boldsymbol{\alpha}, M) \text{ with } \alpha_t \stackrel{\text{iid}}{\sim} \text{Beta}(a_\alpha, b_\alpha)
\end{aligned} \tag{1.4}$$

where tRPM represents the temporal random partition model (1.3).

Moving towards our update, we decided to refine some parts of that formulation. Regarding the variances σ_{jt}^{2*} , τ_t^2 , and λ^2 , we chose to model them trough an inverse gamma distribution rather than the uniform employed originally. This is indeed a more sophisticate choice, since the tuning of the parameters of an $\text{invGamma}(a, b)$ is a bit more difficult than simply setting the bounds of a $\mathcal{U}(l, u)$, but should guarantee a better mixing in the chain. In fact, the invGamma distributions recovers conjugacy in the model, thanks to the normal law assigned to the other parameters, allowing the update step of the variances to be performed trough the analytically exact Gibbs sampler rather than the acceptance-rejection method of Metropolis algorithm. Finally, to improve the accuracy in fitting the target values, we added a regression parameter β_t in the likelihood. We decided to make it only time-dependent, and not also unit-dependent, to lighten the already quite-heavy formulation.

The final updated model is now proposed, with highlighted in dark red the

changes and insertions that we made.

$$\begin{aligned}
Y_{it} | Y_{it-1}, \boldsymbol{\mu}_t^*, \boldsymbol{\sigma}_t^{2*}, \boldsymbol{\eta}, \mathbf{c}_t &\stackrel{\text{ind}}{\sim} \mathcal{N}(\mu_{c_{it}}^* + \eta_{1i} Y_{it-1} + \mathbf{x}_{it}^T \boldsymbol{\beta}_t, \sigma_{c_{it}}^{2*}(1 - \eta_{1i}^2)) \\
Y_{i1} &\stackrel{\text{ind}}{\sim} \mathcal{N}(\mu_{c_{i1}}^* + \mathbf{x}_{i1}^T \boldsymbol{\beta}_1, \sigma_{c_{i1}}^{2*}) \\
\boldsymbol{\beta}_t &\stackrel{\text{ind}}{\sim} \mathcal{N}_p(\mathbf{b}, s^2 I) \\
\xi_i = \text{Logit}(\frac{1}{2}(\eta_{1i} + 1)) &\stackrel{\text{ind}}{\sim} \text{Laplace}(a, b) \\
(\mu_{jt}^*, \sigma_{jt}^{2*}) &\stackrel{\text{ind}}{\sim} \mathcal{N}(\vartheta_t, \tau_t^2) \times \text{invGamma}(a_\sigma, b_\sigma) \\
\vartheta_t | \vartheta_{t-1} &\stackrel{\text{ind}}{\sim} \mathcal{N}((1 - \varphi_1)\varphi_0 + \varphi_1 \vartheta_{t-1}, \lambda^2(1 - \varphi_1^2)) \\
(\varphi_1, \tau_t^2) &\stackrel{\text{iid}}{\sim} \mathcal{N}(\varphi_0, \lambda^2) \times \text{invGamma}(a_\tau, b_\tau) \\
(\varphi_0, \varphi_1, \lambda^2) &\sim \mathcal{N}(m_0, s_0^2) \times \mathcal{U}(-1, 1) \times \text{invGamma}(a_\lambda, b_\lambda) \\
\{\mathbf{c}_t, \dots, \mathbf{c}_T\} &\sim \text{tRPM}(\boldsymbol{\alpha}, M) \text{ with } \alpha_t \stackrel{\text{iid}}{\sim} \text{Beta}(a_\alpha, b_\alpha)
\end{aligned} \tag{1.5}$$

A visual representation of this new version of the DRPM model is also present in Figure 1.1, to more clearly appreciate the hierarchical structure and the relations among the parameters.

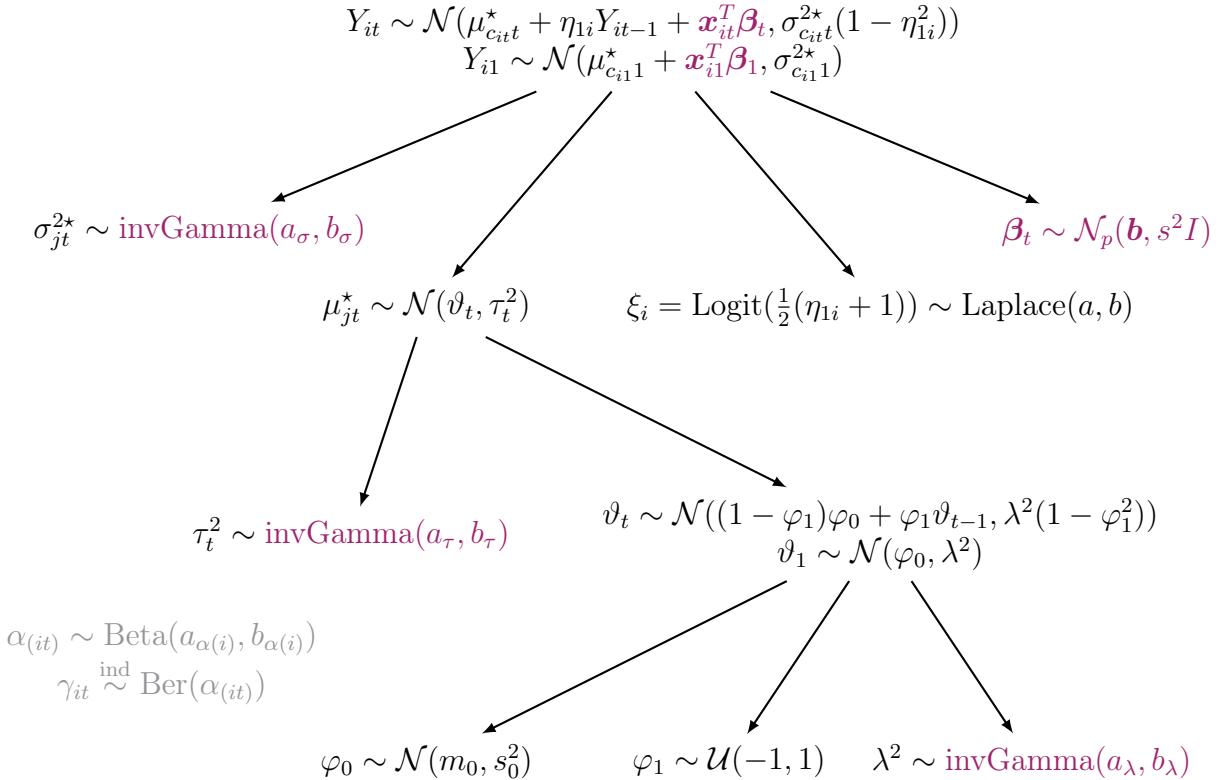


Figure 1.1: Graph visualization of the DRPM model, with highlighted in dark red the changes that we made to the original formulation and in gray the internal variables of the model.

In the course of this work, for the sake of clarity, we will refer to CDRPM for the original model formulation of [PQD22], and to JDRPM for our updated version. The starting letters C and J refer to the languages in which their corresponding MCMC algorithm has been implemented: C for the former, Julia for the latter.

We will now dive more deeply into the characteristics of our updated model by outlining the MCMC algorithm and, subsequently, inspecting the behaviours of the spatial cohesions and covariates similarities. This description refers to our generalization, the JDRPM model of (1.5), but we will remark analogies and differences with respect to the original CDRPM of (1.4).

1.1 MCMC sampling algorithm

We now report the full conditionals derivation for the parameters which had a conjugacy in the model (for the complete steps see Appendix A). Their computation is theoretically “simple”, where we use the fact that $\text{posterior} \propto \text{likelihood} \cdot \text{prior}$, but followed useful suggestions and tricks from [Dun16]. The other variables not included here, namely η_{1i} and φ_1 , involved instead the classical Metropolis update.

- update $\sigma_{jt}^{2\star}$. This full conditional derivation is characteristic of JDRPM only, since in CDRPM the variance had a uniform law and was therefore updated through a Metropolis step.

$$\begin{aligned} \text{for } t = 1: f(\sigma_{jt}^{2\star} | -) &\propto \text{kernel of a invGamma}(a_{\sigma(\text{post})}, b_{\sigma(\text{post})}) \text{ with} \\ a_{\tau(\text{post})} &= a_\sigma + \frac{|S_{jt}|}{2} \quad b_{\tau(\text{post})} = b_\sigma + \frac{1}{2} \sum_{i \in S_{jt}} (Y_{it} - \mu_{jt}^* - \mathbf{x}_{it}^T \boldsymbol{\beta}_t)^2 \\ \text{for } t > 1: f(\sigma_{jt}^{2\star} | -) &\propto \text{kernel of a invGamma}(a_{\sigma(\text{post})}, b_{\sigma(\text{post})}) \text{ with} \\ a_{\tau(\text{post})} &= a_\sigma + \frac{|S_{jt}|}{2} \quad b_{\tau(\text{post})} = b_\sigma + \frac{1}{2} \sum_{i \in S_{jt}} (Y_{it} - \mu_{jt}^* - \eta_{1i} Y_{it-1} - \mathbf{x}_{it}^T \boldsymbol{\beta}_t)^2 \end{aligned} \tag{1.6}$$

- update μ_{jt}^* . This update rule is the same for both JDRPM and CDRPM.

$$\begin{aligned} \text{for } t = 1: f(\mu_{jt}^* | -) &\propto \text{kernel of a } \mathcal{N}(\mu_{\mu_{jt}^*(\text{post})}, \sigma_{\mu_{jt}^*(\text{post})}^2) \text{ with} \\ \sigma_{\mu_{jt}^*(\text{post})}^2 &= \frac{1}{\frac{1}{\tau_t^2} + \frac{|S_{jt}|}{\sigma_{jt}^{2\star}}} \quad \mu_{\mu_{jt}^*(\text{post})} = \sigma_{\mu_{jt}^*(\text{post})}^2 \left(\frac{\vartheta_t}{\tau_t^2} + \frac{\sum_{i \in S_{jt}} (Y_{i1} - \mathbf{x}_{i1}^T \boldsymbol{\beta}_t)}{\sigma_{jt}^{2\star}} \right) \\ \text{for } t > 1: f(\mu_{jt}^* | -) &\propto \text{kernel of a } \mathcal{N}(\mu_{\mu_{jt}^*(\text{post})}, \sigma_{\mu_{jt}^*(\text{post})}^2) \text{ with} \\ \sigma_{\mu_{jt}^*(\text{post})}^2 &= \frac{1}{\frac{1}{\tau_t^2} + \frac{\sum_{i \in S_{jt}} \frac{1}{1 - \eta_{1i}^2}}{\sigma_{jt}^{2\star}}} \quad \mu_{\mu_{jt}^*(\text{post})} = \sigma_{\mu_{jt}^*(\text{post})}^2 \left(\frac{\vartheta_t}{\tau_t^2} + \frac{\sum_{i \in S_{jt}} \frac{Y_{it} - \eta_{1i} Y_{i,t-1} - \mathbf{x}_{it}^T \boldsymbol{\beta}_t}{1 - \eta_{1i}^2}}{\sigma_{jt}^{2\star}} \right) \end{aligned} \tag{1.7}$$

- update $\boldsymbol{\beta}_t$. This full conditional derivation is characteristic of JDRPM only, since the insertion of a regression term in the likelihood is a feature introduced by our generalized model.

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

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

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

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

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

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

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

$$\mathbf{h}_{(\text{post})} = \left(\frac{\mathbf{b}}{s^2} + \sum_{i=1}^n \frac{(Y_{it} - \mu_{c_{it}t}^* - \eta_{1i} Y_{it-1}) \mathbf{x}_{it}}{\sigma_{c_{it}t}^{2\star}} \right) \quad J_{(\text{post})} = \left(\frac{1}{s^2} I + \sum_{i=1}^n \frac{\mathbf{x}_{it} \mathbf{x}_{it}^T}{\sigma_{c_{it}t}^{2\star}} \right) \quad (1.8)$$

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. As a consequence, 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 apart from the previously mentioned disadvantages would be a statistically equivalent form.

- update τ_t^2 . This full conditional derivation is characteristic of JDRPM only, since in CDRPM the variance had a uniform law and was therefore updated through a Metropolis step.

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

$$a_{\tau(\text{post})} = \frac{k_t}{2} + a_\tau \quad b_{\tau(\text{post})} = \frac{\sum_{j=1}^{k_t} (\mu_{jt}^* - \vartheta_t)^2}{2} + b_\tau \quad (1.9)$$

- update ϑ_t . This update rule is the same for both JDRPM and CDRPM.

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

$$\begin{aligned} \sigma_{\vartheta_t(\text{post})}^2 &= \frac{1}{\frac{1}{\lambda^2(1-\varphi_1^2)} + \frac{k_t}{\tau_t^2}} \\ \mu_{\vartheta_t(\text{post})} &= \sigma_{\vartheta_t(\text{post})}^2 \left(\frac{\sum_{j=1}^{k_t} \mu_{jt}^*}{\tau_t^2} + \frac{(1-\varphi_1)\varphi_0 + \varphi_1 \vartheta_{t-1}}{\lambda^2(1-\varphi_1^2)} \right) \end{aligned}$$

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

$$\sigma_{\vartheta_t(\text{post})}^2 = \frac{1}{\frac{1+\varphi_1^2}{\lambda^2(1-\varphi_1^2)} + \frac{k_t}{\tau_t^2}}$$

$$\mu_{\vartheta_t(\text{post})} = \sigma_{\vartheta_t(\text{post})}^2 \left(\frac{\sum_{j=1}^{k_t} \mu_{jt}^*}{\tau_t^2} + \frac{\varphi_1(\vartheta_{t-1} + \vartheta_{t+1}) + \varphi_0(1 - \varphi_1)^2}{\lambda^2(1 - \varphi_1^2)} \right)$$

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

$$\begin{aligned} \sigma_{\vartheta_t(\text{post})}^2 &= \frac{1}{\frac{1}{\lambda^2} + \frac{\varphi_1^2}{\lambda^2(1 - \varphi_1^2)} + \frac{k_t}{\tau_t^2}} \\ \mu_{\vartheta_t(\text{post})} &= \sigma_{\vartheta_t(\text{post})}^2 \left(\frac{\varphi_0}{\lambda^2} + \frac{\varphi_1(\vartheta_{t+1} - (1 - \varphi_1)\varphi_0)}{\lambda^2(1 - \varphi_1^2)} + \frac{\sum_{j=1}^{k_t} \mu_{jt}^*}{\tau_t^2} \right) \end{aligned} \quad (1.10)$$

- update φ_0 . This update rule is also the same for both JDRPM and CDRPM.

$f(\varphi_0| -) \propto \text{kernel of a } \mathcal{N}(\mu_{\varphi_0(\text{post})}, \sigma_{\varphi_0(\text{post})}^2)$ with

$$\begin{aligned} \sigma_{\varphi_0(\text{post})}^2 &= \frac{1}{\frac{1}{s_0^2} + \frac{1}{\lambda^2} + \frac{(T-1)(1-\varphi_1)^2}{\lambda^2(1-\varphi_1^2)}} \\ \mu_{\varphi_0(\text{post})} &= \sigma_{\varphi_0(\text{post})}^2 \left(\frac{m_0}{s_0^2} + \frac{\vartheta_1}{\lambda^2} + \frac{1-\varphi_1}{\lambda^2(1-\varphi_1^2)} \sum_{t=2}^T (\vartheta_t - \varphi_1 \vartheta_{t-1}) \right) \end{aligned} \quad (1.11)$$

- update λ^2 . This full conditional derivation is characteristic of JDRPM only, since in CDRPM the variance had a uniform law and was therefore updated through a Metropolis step.

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

$$\begin{aligned} a_{\lambda(\text{post})} &= \frac{T}{2} + a_\lambda \\ b_{\lambda(\text{post})} &= \frac{(\vartheta_1 - \varphi_0)^2}{2} + \sum_{t=2}^T \frac{(\vartheta_t - (1 - \varphi_1)\varphi_0 - \varphi_1 \vartheta_{t-1})^2}{2} + b_\lambda \end{aligned} \quad (1.12)$$

- update α . This update rule is the same for both JDRPM and CDRPM.

if global α : $f(\alpha| -) \propto \text{kernel of a Beta}(a_{\alpha(\text{post})}, b_{\alpha(\text{post})})$ with

$$a_{\alpha(\text{post})} = a_\alpha + \sum_{i=1}^n \sum_{t=1}^T \gamma_{it} \quad b_{\alpha(\text{post})} = b_\alpha + nT - \sum_{i=1}^n \sum_{t=1}^T \gamma_{it}$$

if time specific α : $f(\alpha_t| -) \propto \text{kernel of a Beta}(a_{\alpha(\text{post})}, b_{\alpha(\text{post})})$ with

$$a_{\alpha(\text{post})} = a_\alpha + \sum_{i=1}^n \gamma_{it} \quad b_{\alpha(\text{post})} = b_\alpha + n - \sum_{i=1}^n \gamma_{it}$$

if unit specific α : $f(\alpha_i| -) \propto \text{kernel of a Beta}(a_{\alpha(\text{post})}, b_{\alpha(\text{post})})$ with

$$a_{\alpha(\text{post})} = a_{\alpha i} + \sum_{t=1}^T \gamma_{it} \quad b_{\alpha(\text{post})} = b_{\alpha i} + T - \sum_{t=1}^T \gamma_{it}$$

if time and unit specific α : $f(\alpha_{it}| -) \propto \text{kernel of a Beta}(a_{\alpha(\text{post})}, b_{\alpha(\text{post})})$ with

$$a_{\alpha(\text{post})} = a_{\alpha i} + \gamma_{it} \quad b_{\alpha(\text{post})} = b_{\alpha i} + 1 - \gamma_{it} \quad (1.13)$$

- update a missing observation Y_{it} . This full conditional derivation is characteristic of JDRPM only, since the handling of missing data feature introduced by our generalized model.

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

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

for $1 < t < T$: $f(Y_{it} | -) \propto$ kernel of a $\mathcal{N}(\mu_{Y_{it}(\text{post})}, \sigma_{Y_{it}(\text{post})}^2)$ with

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

for $t = T$: $f(Y_{it} | -)$ is just the likelihood of Y_{it} (1.14)

Finally, we briefly highlight in Algorithm 1 the steps which compose the MCMC sampling algorithm. Regarding the computation of the LPML and WAIC metrics, they follow classic ideas from [Chr+10] and [GHV13].

The core of the clustering process happens in the updating steps of γ_{it} and ρ_t . Their update step is indeed quite complex, and as we said before involves the check of compatibility issues. In any case, the general idea is that, for each unit i currently belonging to cluster j , we simulate to assign her to one of the existing clusters, plus to a new singleton cluster, and compute for each case the likelihood of this to happen, deriving probability weights to finally sample the decision for the next iteration. The key elements participating into the definition of such weights are the spatial cohesions and, with the JDRPM update, also the covariate similarities, which we will now both investigate.

1.2 Spatial cohesions analysis

The clustering procedure revolves around the product partition model (PPM) [Har90] [BH93] [Cro97]. The simplest idea is to set $P(\rho_t) \propto \prod_{j=1}^{k_t} C(S_{jt})$, with the function $C(S_{jt})$ that measures how tightly grouped the elements in cluster S_{jt} are considered to be. Then, to include spatial information, the idea is to extend the PPM from being a function of just $C(S_{jt})$ to the more informed one $C(S_{jt}, \mathbf{s}_{jt}^*)$, where S_{jt} is the j -th cluster at time instant t and \mathbf{s}_{jt}^* is the subset of spatial coordinates of the units inside S_{jt} . For the sake of clarity, in this section where we are just interested in analysing the cohesions we employ the S_h notation to indicate a general h -th cluster, rather than the more pedantic S_{jt} .

Regarding the computation of spatial cohesion, several choices are available [PQ15]. The main common idea of the following formulas is to favour few spatially

Algorithm 1: Pseudocode for the MCMC fitting algorithm of the JDRPM model (1.5).

```

1 for  $d = 1, \dots, draws$  do
2   if target variable has missing values then
3     | update the missing  $Y_{it}$ 's using (1.14)
4   end
5   for  $t = 1, \dots, T$  do
6     | for  $i = 1, \dots, n$  do
7       |   | if  $t = 1$  then
8         |     |    $\gamma_{it} = 0$ 
9       |   | else
10      |     |   update  $\gamma_{it}$ 
11    |   | end
12  |   | end
13  |   for  $i \in \{l : \gamma_{lt} = 0\}$  do
14    |     | update  $c_{it}$  (i.e. update  $\rho_t$ )
15  |   | end
16  |   update  $\mu_{jt}^*$  using (1.7)
17  |   update  $\sigma_{jt}^{2*}$  using (1.6)
18  |   if are there covariates in the likelihood then
19    |     | update  $\beta_t$ 
20  |   | end
21  |   update  $\vartheta_t$  using (1.10)
22  |   update  $\tau_t^2$  using (1.9)
23 end
24 if update_eta = true then
25   | update  $\eta_{1i}$  using Metropolis sampling
26 end
27 if update_alpha = true then
28   | update  $\alpha$  using (1.13)
29 end
30 update  $\varphi_0$  using (1.11)
31 if update_phi1 = true then
32   | update  $\varphi_1$  using Metropolis sampling
33 end
34 update  $\lambda^2$  using (1.12)
35 if  $d > burnin$  and  $d \% thin = 0$  then
36   | save MCMC current iterate
37 end
38 end
39 compute LPML and WAIC metrics
40 if perform diagnostics then
41   | print diagnostics (ESS and  $\hat{R}$ ) on parameters  $\lambda^2, \varphi_0, \tau_t^2, \vartheta_t, \eta_{1i}, \alpha$ 
42 end

```

connected clusters rather than a lot of singleton ones, to derive more interpretable and meaningful results. For this reason, most of the cohesions employ the $M \cdot \Gamma(|S_h|)$ term, which resembles the DP partitioning method that helps in reaching that goal.

We will now describe briefly all the cohesions which are implemented in the JDRPM model, and were implemented as well in the CDRPM model, and conduct tests on each of them, to see how the tuning of their parameters reflects on the computed values.

All the tests of Figures 1.3 and 1.4 refer to the partition of Figure 1.2, taken as test case here from a general fit on the same spatio-temporal dataset of Chapter 3. The following results, as well as the ones of the next section, are presented with the logarithm applied to better highlight the differences among them, otherwise for example all values could be really close together and make the analysis less understandable, and moreover because this is the actual perspective in which the implementation works. In fact, the fitting algorithm firstly saves the log-transformed values generated by the cohesions, in order to avoid numerical problems and instabilities, and secondly exponentiates and normalizes them into proper scaled probabilities, from which finally draw the cluster assignments.

Cohesions analysis test case

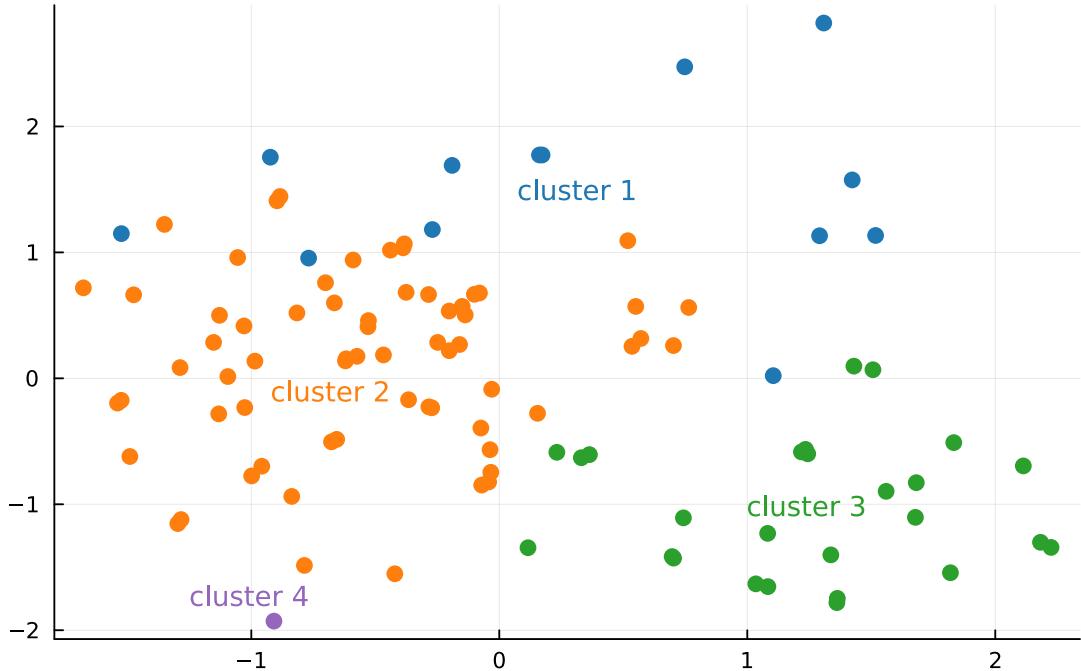


Figure 1.2: Partition considered to analyse the spatial cohesions.

The first cohesion uses a tessellation idea from [DH01] that considers $\mathcal{D}_h = \sum_{i \in S_h} \|s_i - \bar{s}_h\|$ as the total distance from the units to the cluster centroid \bar{s}_h . The computation is then an adjustment of a decreasing function in terms of \mathcal{D}_h , to give an higher weight on clusters which are denser, i.e. that have lower \mathcal{D}_h , with an

additional parameter α to provide more control on the penalization.

$$C_1(S_h, \mathbf{s}_h^*) = \begin{cases} \frac{M \cdot \Gamma(|S_h|)}{\Gamma(\alpha \mathcal{D}_h) \mathbb{1}_{[\mathcal{D}_h \geq 1]} + \mathcal{D}_h \mathbb{1}_{[\mathcal{D}_h < 1]}} & \text{if } |S_h| > 1 \\ M & \text{if } |S_h| = 1 \end{cases} \quad (1.15)$$

The second function provides, instead, a hard cluster boundary where the weight is set to 1 (i.e. 0 with the logarithm view) only if all the distances between all possible pairs of points inside the cluster are below the threshold parameter, i.e. if all units are “close enough” to each other. If this does not happen, even for a single pair of points, the returned value is 0, which corresponds to the maximum penalization since it would be $-\infty$ in the logarithm perspective. The strictness of this requirement can be adjusted trough the parameter a .

$$C_2(S_h, \mathbf{s}_h^*) = M \cdot \Gamma(|S_h|) \cdot \prod_{i,j \in S_h} \mathbb{1}_{[\|\mathbf{s}_i - \mathbf{s}_j\| \leq a]} \quad (1.16)$$

According to this function, from Figure 1.3 we can see how the purple cluster is considered the one with the highest cohesion, being a singleton. The runner-up is the green cluster, because it’s the first among all the non-singletons which activates cohesion 2 when we increase the parameter a . The orange and blue clusters, in contrast, appear to be less dense since they require an higher value of a to “pass” the distance check.

However, cohensions C_1 and C_2 do not preserve the exchangeability property, meaning that if we would marginalize the random partition model over the last of m units we would not get to the same model as if we only had $m - 1$ units. This coherence property, known as sample size consistency or addition rule [De +15], is instead often desirable, for theoretical or computational purposes, and the following two cohensions are able to provide it [MQR11].

Cohesion 3, called auxiliary similarity, treats the spatial coordinates \mathbf{s} as if they were random, applying on them a model such as the Normal/Normal-Inverse-Wishart with $\boldsymbol{\xi} = (\mathbf{m}, V)$, $\mathbf{s}|\boldsymbol{\xi} \sim \mathcal{N}(\mathbf{m}, V)$ and $\boldsymbol{\xi} \sim \mathcal{NIW}(\boldsymbol{\mu}_0, \kappa_0, \nu_0, \Lambda_0)$. The idea is to assign a larger weight on clusters which produce large marginal likelihood values, i.e. which according to the modelling are more probable to appear.

$$C_3(S_h, \mathbf{s}_h^*) = M \cdot \Gamma(|S_h|) \cdot \int \prod_{i \in S_h} q(\mathbf{s}_i | \boldsymbol{\xi}_h) q(\boldsymbol{\xi}_h) d\boldsymbol{\xi}_h \quad (1.17)$$

On the same line there is cohesion 4, the double dipper cohesion [QMP15], which now employs the posterior predictive distribution rather than the prior predictive distribution of cohesion C_3 .

$$C_4(S_h, \mathbf{s}_h^*) = M \cdot \Gamma(|S_h|) \cdot \int \prod_{i \in S_h} q(\mathbf{s}_i | \boldsymbol{\xi}_h) q(\boldsymbol{\xi}_h | \mathbf{s}_h^*) d\boldsymbol{\xi}_h \quad (1.18)$$

Another final idea comes from the cluster variance/entropy similarity function, a very general methodology to measure the closeness of a set of values which in fact

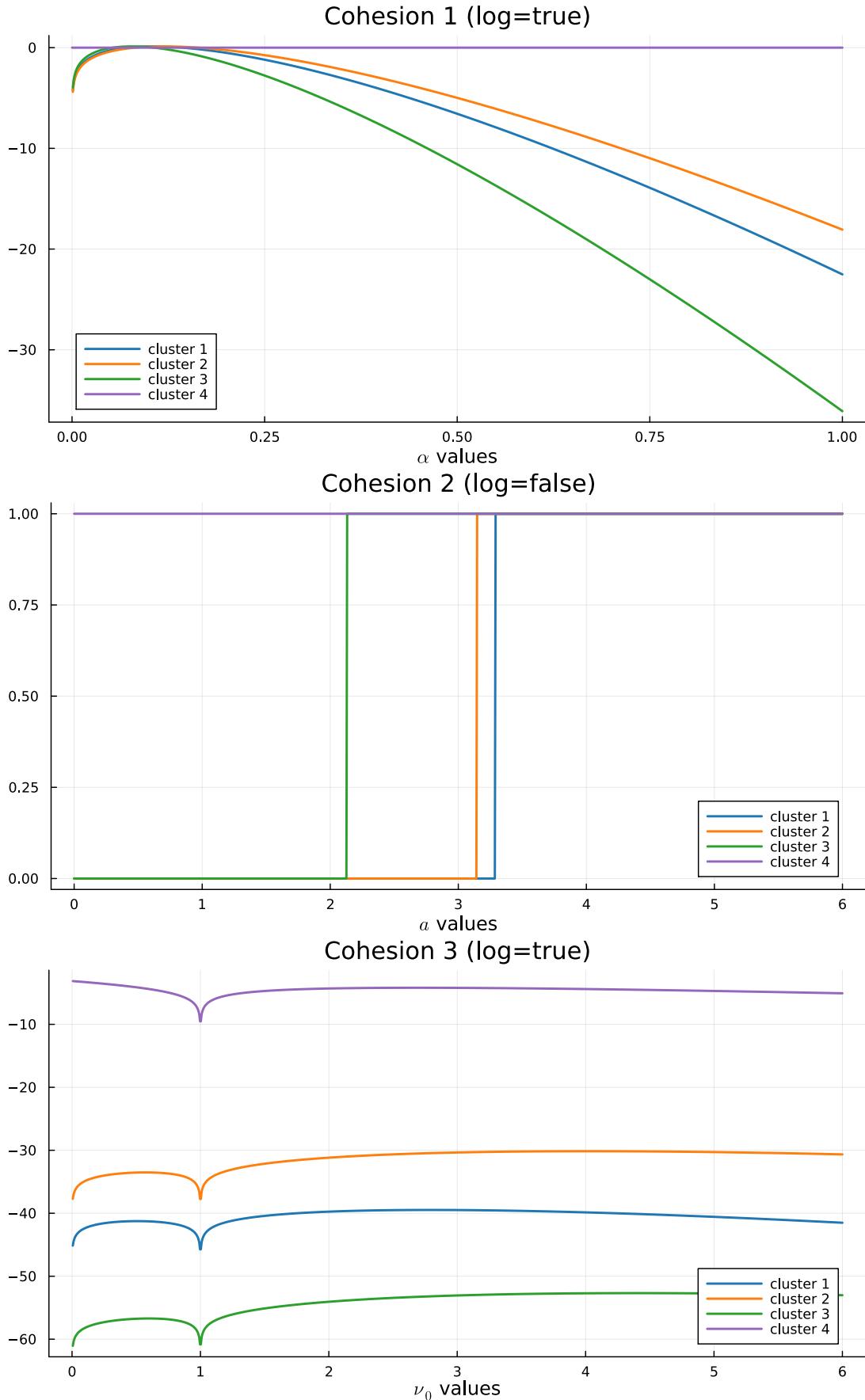


Figure 1.3: Cohesions 1, 2, and 3 computed on the test case partition, with respect to different values of their tuning parameter. Cohesion 2 is without the logarithm applied just for plotting purposes, since otherwise the values would have been $-\infty$ and 0.

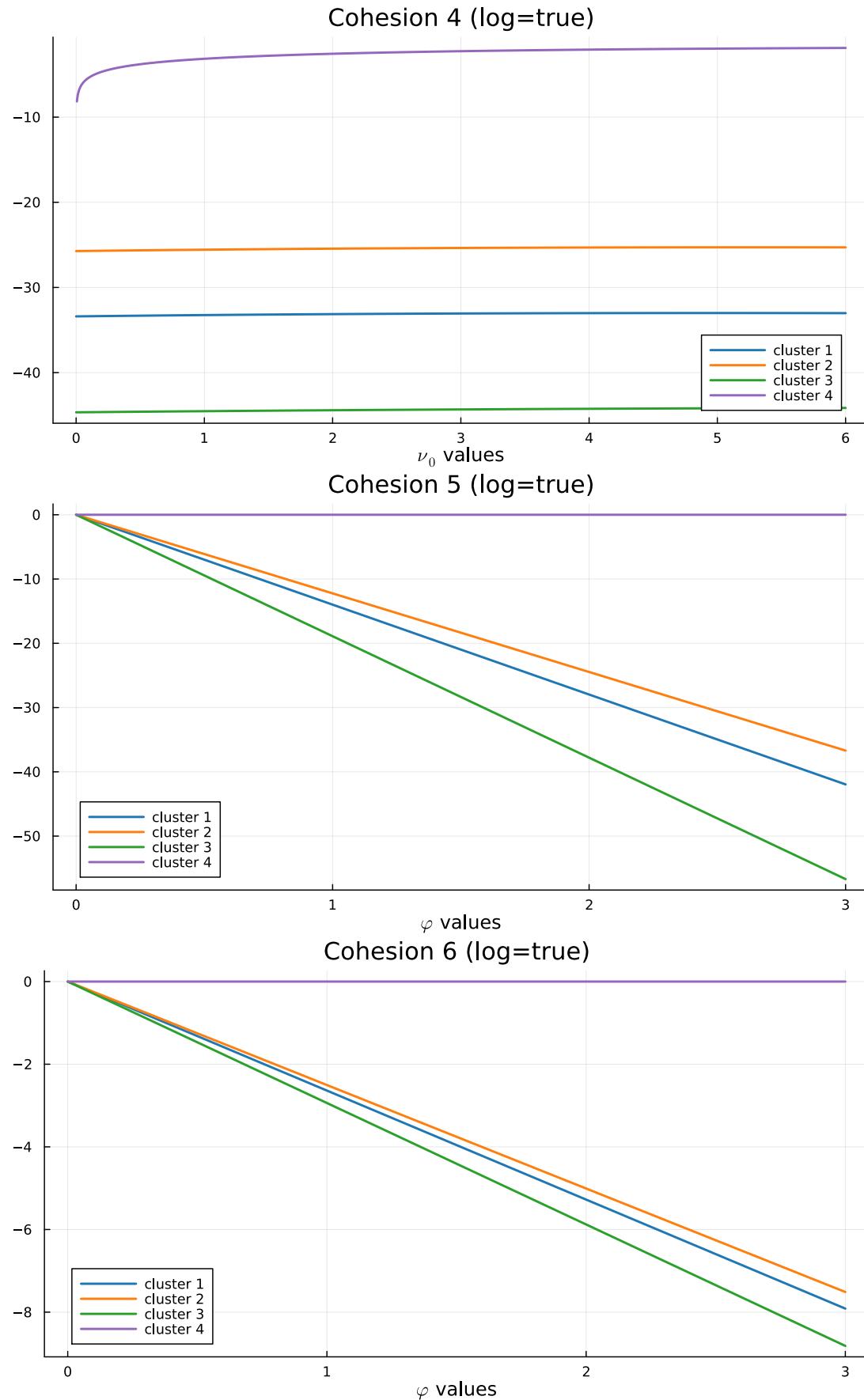


Figure 1.4: Cohesions 4, 5, and 6 computed on the test case partition, with respect to different values of their tuning parameter.

will be used also for the covariates case. As in cohesion C_1 , the idea is to derive a summary of the closeness of the units, summing the distances of the units from the cluster centroid, and then adjust the parameter φ to control how much penalize dissimilar values. In this way we arrive to the final two cohesions.

$$C_5(S_h, \mathbf{s}_h^*) = \exp \left\{ -\varphi \sum_{i \in S_h} \|\mathbf{s}_i - \bar{\mathbf{s}}_h\| \right\} \quad (1.19)$$

$$C_6(S_h, \mathbf{s}_h^*) = \exp \left\{ -\varphi \log \left(\sum_{i \in S_h} \|\mathbf{s}_i - \bar{\mathbf{s}}_h\| \right) \right\} \quad (1.20)$$

1.3 Covariates similarities analysis

A wide spectrum of choice is also available for covariates similarities [PQ18]. To account for them, the idea consists in extending the PPM to make it function of S_{jt} , \mathbf{s}_{jt}^* , and now also of X_{jt}^* , being this the $p \times |S_{jt}|$ matrix storing the covariates of the units belonging to cluster j at time t , i.e. $X_{jt}^* = \{\mathbf{x}_{it}^* = (x_{it1}, \dots, x_{itp})^T : i \in S_{jt}\}$.

For the current implementation of JDRPM we decided to treat each covariate individually, therefore the PPM will actually be function of the set of unidimensional vectors which record the different p covariates of the units inside cluster S_{jt} , meaning $\mathbf{x}_{jt1}^*, \dots, \mathbf{x}_{jtp}^*$, of which all contributions will be considered independently one to each other. In this way, the final PPM form will be

$$P(\rho) \propto \prod_{h=1}^{k_t} C(S_{jt}, \mathbf{s}_{jt}^*) \left(\prod_{r=1}^p g(S_{jt}, \mathbf{x}_{jtr}^*) \right) \quad (1.21)$$

where \mathbf{x}_{jtr}^* represents the vector recording the r -th covariate values for all the units inside cluster S_{jt} , i.e. row r of matrix X_{jt}^* . This choice is lighter from the theoretical and computational perspectives, and allows a mixture of numerical and categorical covariates without any problem. A unified and multidimensional consideration of the covariates is nonetheless possible, with proper adjustments on the similarities functions, and would have lead to a PPM of the form

$$P(\rho) \propto \prod_{h=1}^{k_t} C(S_{jt}, \mathbf{s}_{jt}^*) g(S_{jt}, X_{jt}^*) \quad (1.22)$$

As in the previous section, we will now discuss all the similarities implemented in the JDRPM model and perform tests on each of them. All the tests will be referred to the test case partition displayed in Figure 1.5 which is about the **Altitude** covariate from the spatio-temporal dataset that will be used in Chapter 3. Regarding the notation, we will again employ the simpler and general one dropping the spatio-temporal context.

The first similarity is the cluster variance/entropy similarity function, which as the name suggest can work with both numerical and categorical variables. The general form is

$$g_1(S_h, \mathbf{x}_h^*) = \exp \{-\varphi H(S_h, \mathbf{x}_h^*)\} \quad (1.23)$$

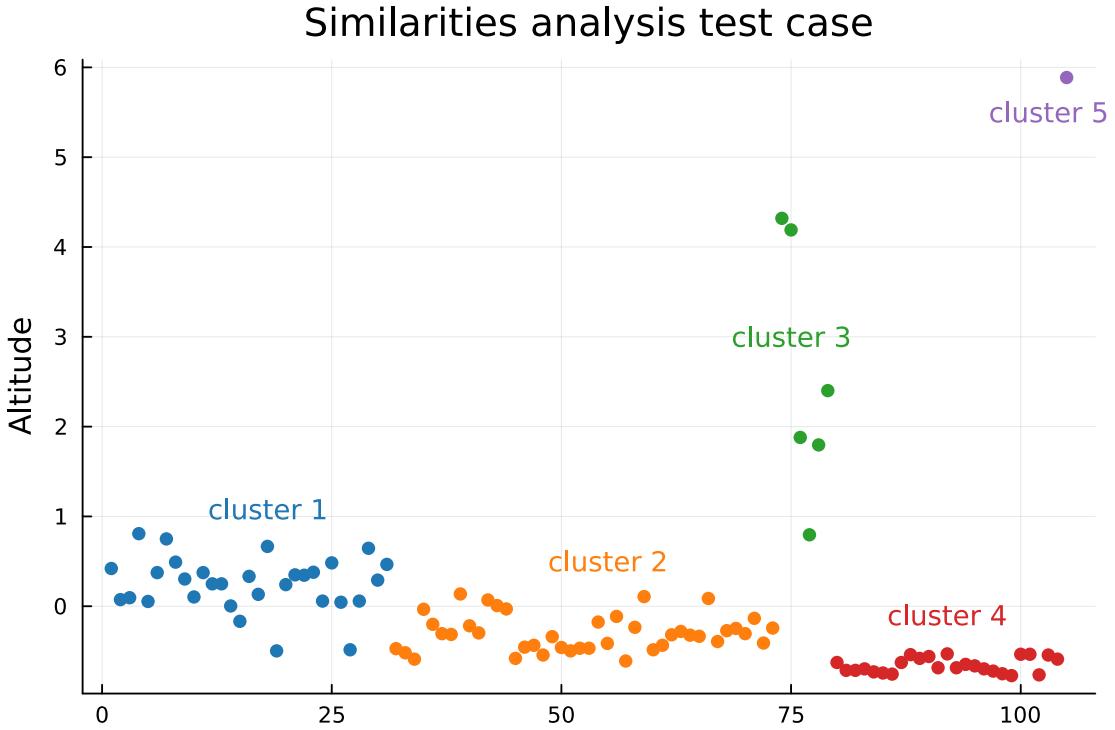


Figure 1.5: Partition considered to analyse the covariate similarities. The order of the units has been modified to plot together all units belonging to the same cluster.

with $H(S_h, \mathbf{x}_h^*) = \sum_{i \in S_h} (x_i - \bar{x}_h)^2$ for numerical covariates, being \bar{x}_h the mean value of the \mathbf{x}_h^* vector, and $H(S_h, \mathbf{x}_h^*) = -\sum_{c=1}^C \hat{p}_c \log(\hat{p}_c)$ for categorical covariates, with \hat{p}_c indicating the relative frequency at which each factor appears. The parameter φ controls the amount of penalization to apply. This function can be extended quite easily to the multidimensional case, at least for the case of numerical covariates only, where the H function becomes $H(S_h, X_h^*) = \sum_{r=1}^p \|\mathbf{x}_r - \bar{\mathbf{x}}_h\|$.

Another popular choice is the Gower similarity function [Gow71], which was originally designed for a multivariate context, where vectors of covariates are compared to each other. As a consequence, some work was required to adapt it to the univariate case. The simple idea of comparing all cluster-specific pair-wise similarities leads to the total Gower similarity.

$$g_2(S_h, \mathbf{x}_h^*) = \exp \left\{ -\alpha \sum_{i,j \in S_h, i \neq j} d(x_i, x_j) \right\} \quad (1.24)$$

This function, however, is strictly increasing with respect to the cluster size, meaning that it will naturally tend to propose a large number of small clusters. For that reason, a correction can be applied, accounting for the size of the cluster S_h , leading to the average Gower similarity.

$$g_3(S_h, \mathbf{x}_h^*) = \exp \left\{ -\frac{2\alpha}{|S_h|(|S_h| - 1)} \sum_{i,j \in S_h, i \neq j} d(x_i, x_j) \right\} \quad (1.25)$$

In both functions, $d(x_i, x_j)$ is the Gower dissimilarity between x_i and x_j , with $d(x_i, x_j) = |x_i - x_j|/R$ in the case of numerical covariates, being $R = \max(\mathbf{x}) -$

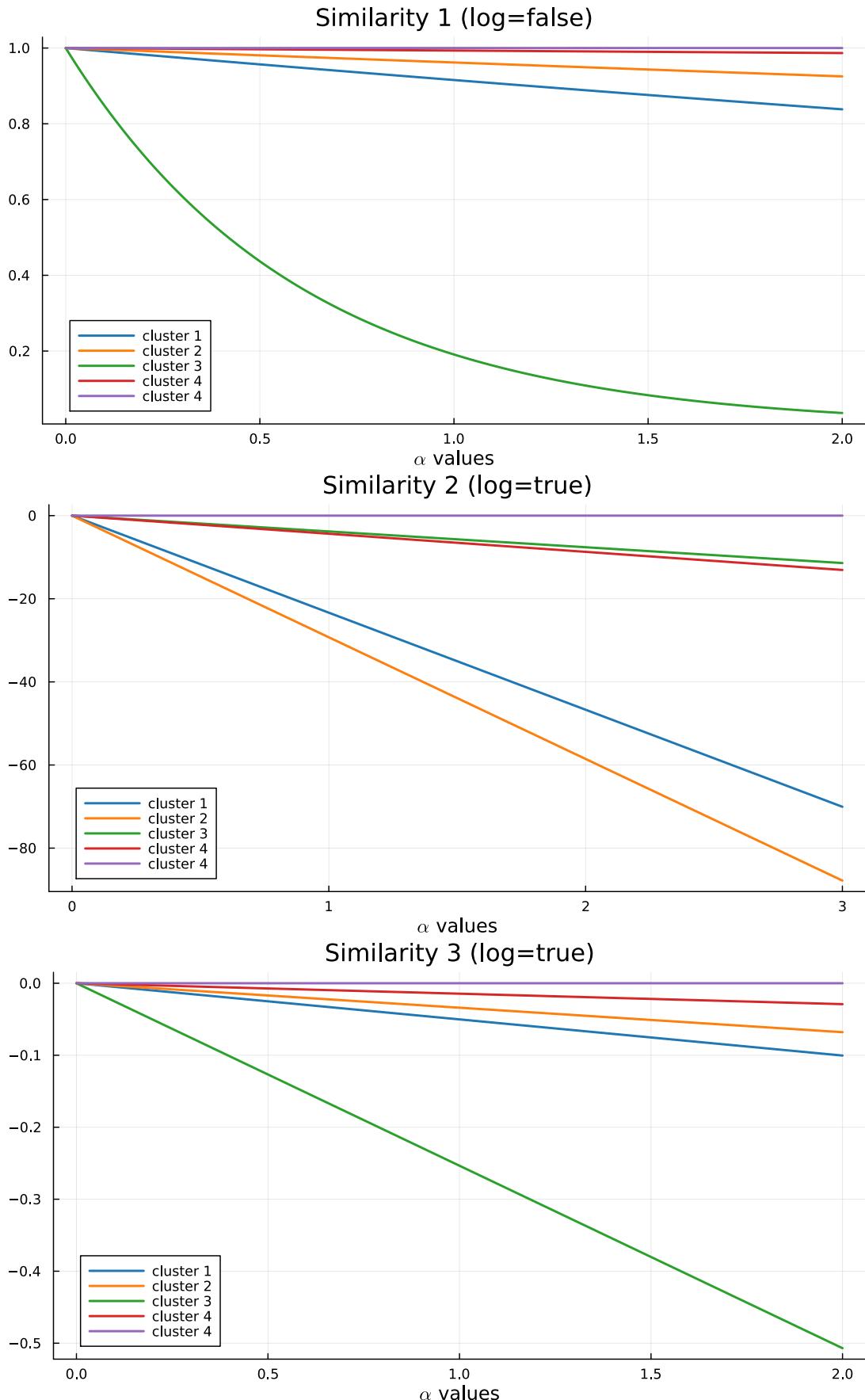


Figure 1.6: Similarities 1, 2 and 3 computed on the test case partition, with respect to different values of their tuning parameter. Similarity 1 is without the logarithm applied just for plotting purposes, to see more clearly the gap among the clusters.

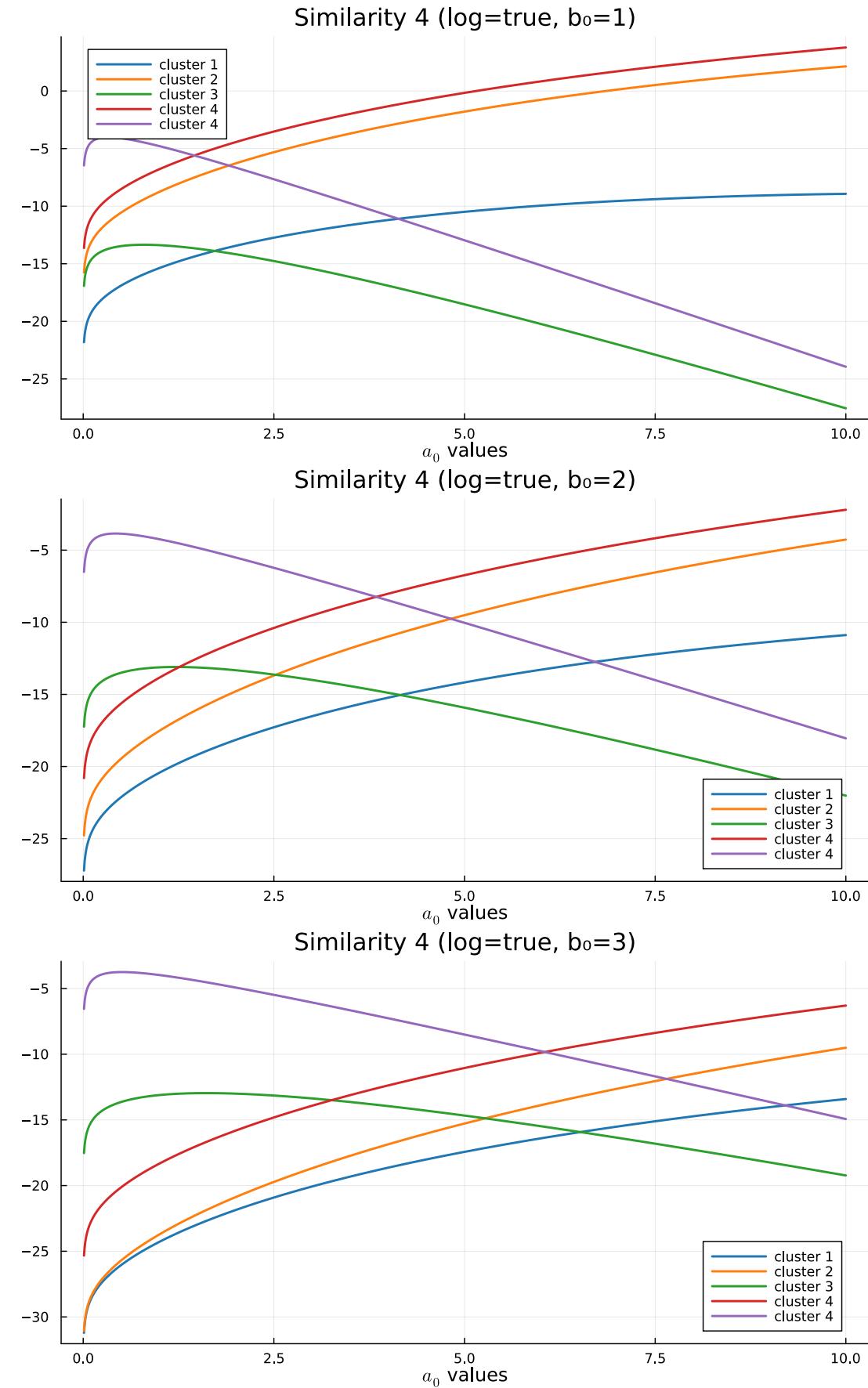


Figure 1.7: Similarity 4 computed on the test case partition, with respect to the different values of the tuning parameter a_0 and b_0 . The other parameters has been set to $\mu_0 = 0$ and $\lambda_0 = 1$, with the idea to apply the similarity on scaled (centered and standardized) covariates.

$\min(\mathbf{x})$ the range of the covariate values, considering all the units independently from their cluster, while $d(x_i, x_j) = \mathbb{1}_{[x_i \neq x_j]}$ in the case of categorical covariates. This is a dissimilarity since values closer to 0 refer to similar data, while values closer to 1 to dissimilar data; therefore the minus sign inside g_2 and g_3 exponents converts the function to a similarity. For the multidimensional design there are natural extensions of the $d(\mathbf{x}_i, \mathbf{x}_j)$ function.

The last similarity recalls the structure of the spatial cohesion C_3 , where now are the covariates to be treated as if they were random variables. However, since we chose to deal with each covariate individually, in this unidimensional setting a Normal/Normal-Inverse-Gamma model is employed, with $\boldsymbol{\xi} = (\mu, \sigma^2)$, $x|\boldsymbol{\xi} \sim \mathcal{N}(\mu_0, \sigma^2)$, and $\mu \sim \mathcal{N}(\mu_0, \sigma^2/\lambda_0)$, $\sigma^2 \sim \text{invGamma}(a_0, b_0)$, i.e. $\boldsymbol{\xi} \sim \mathcal{N}\text{invGamma}(\mu_0, \lambda_0, a_0, b_0)$. A multivariate extension is thus possible through the same modelling style of the spatial coordinates case.

$$g_4(S_h, \mathbf{x}_h^\star) = \int \prod_{i \in S_h} q(x_i | \boldsymbol{\xi}_h) q(\boldsymbol{\xi}_h) d\boldsymbol{\xi}_h \quad (1.26)$$

All the similarities, except for g_2 , agree to classify the non-singleton clusters with the red being the one with the highest similarity, followed by the orange, blue, and green, as we can see from Figures 1.6 and 1.7. Intuitively, Figure 1.5 seems to confirm this ranking, by visually reasoning about the sparsity of each cluster. Similarities g_4 and g_2 seem to be the only ones which are able to give a considerable weight also to the green partition, which is indeed sparser but collects all the large, sort of outliers, values of the covariate at test.

Regarding their flexibility, we tested their behaviour changing the testing covariate. Similarity g_1 appeared to be the most adaptive one, working well (i.e. returning very reasonable orders in the clusters) in every covariate we tested. However, it tends to penalize a lot sparse clusters, as we saw in the previous test case. This could suggest to maybe implement other distance metrics rather than the L^2 norm in the computation of $H(S_h, \mathbf{x}_h^\star)$. On the other hand, similarity 4 appeared to be quite sensible to the parameters regulating the invGamma distribution for σ^2 , suggesting that each covariate that we want to include in the clustering process should have her properly-tuned pair of a_0 and b_0 parameters. For this reason, the JDRPM implementation can provide this flexibility in assigning separately those parameters for each clustering covariate.

Chapter 2

Implementation and optimizations

To implement our updated model, and to do it efficiently, we decided to opt for the Julia language [Bez+17].

Julia is a relatively new programming language that combines the ease and expressiveness of high-level languages to the efficiency and performance of low-level ones. Such balance is largely achieved through the just-in-time (JIT) compilation, implemented using the LLVM framework, together with many nice features like dynamic multiple dispatch and heavy code specialization against multiple runtime types. This design choice allows Julia to be used interactively, in the same fashion to the R, Matlab, or Python consoles, while also supporting the more traditional program execution style of statically compiled languages such as C, C++ and Fortran. This solution provides faster development phases, since for examples code sections can be evaluated and tested line by line, but at the same time guarantees efficient implementations. Performances are further enhanced by the native integration of the optimized BLAS [Law+79] and LAPACK libraries for linear algebra operations, which are often at the core of all scientific applications. Moreover, Julia provides an extensive ecosystem, currently comprised by more than ten thousand packages, spanning over almost all branches of science and engineering. Most of those packages are already highly tested and optimized, and can therefore reduce the implementation time required to the users. As an example, in this work we employed the `Distributions` [Bes+21] [Lin+19] and `Statistics` packages, while the original C implementation had to write all the statistical functionalities from scratch. Considering all this, the Julia choice was deemed very natural.

As we will see in Chapter 3, even despite the higher complexity of the model we managed to get better performance in Julia compared to the original C implementation, at the reasonable cost of a smaller increase in the memory requirements, which nowadays should not be a big deal. Anyway, this improvement was possible through several refinements and optimizations, which we will now briefly discuss.

2.1 Optimizations

One of the major issues encountered during the design of 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 in fact that most of the execution time was actually spent by the garbage collector of Julia, which had the burden of track all the allocated memory and reclaim the unused one in order to make it available again for new computations.

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

Regarding the second point, there are several 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 necessarily to be 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. We managed to reduce all the useless type instabilities, but some are instead still present to guarantee some flexibility to the users, e.g. to select at run time which cohesion and similarity functions use in the fit.

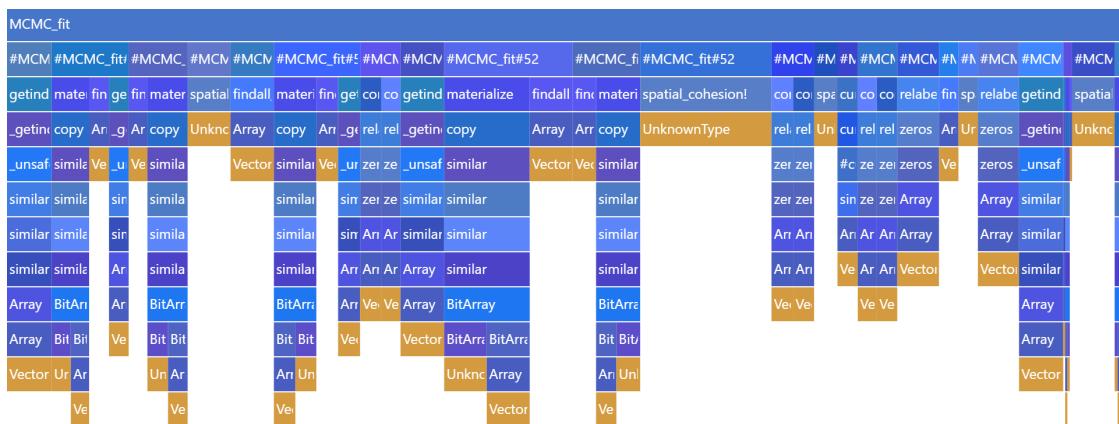


Figure 2.1: Flame graph derived from a simple test fit with $n = 20$, $T = 50$, ran for 10000 iterates. Each section, identified with a colored box, represents a function call with its stack trace below, i.e. all the other subsequent function calls generated from the initial top one. The widths are proportional 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 with respect to function calls, and from left to right with respect to the fitting steps.

Regarding the first point, the memory issues, we heavily inspected it through the `ProfileCanvas` package. This profiler generates a plot, the flame graph, represented in Figure 2.1, which illustrates 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, in this case of memory investigation, the number or size of allocations that was required for that section. This allows to see if the running time is spent on actual useful operations or instead on “bad” ones, such as garbage collection and run-time evaluations; therefore highlighting the sections of code which could possibly be optimized. All the modelling and working variables were of course preallocated, but the key idea has been to refactor the code to make it work more *in-place*, passing as arguments the variables which would be modified by a function and applying those changes directly inside the function, rather than returning some values and then use them to modify the initial variables. On top of that, some other easier improvements were possible through various features of the Julia language, such as the `@view` macro which allowed to remove unnecessary copies by just passing a reference to the portion of vectors or matrices which was needed to perform a certain computation, rather than passing the whole structure.

Also the `BenchmarkTools` [CR16] package has been in general a great tool to quickly analyse and compare entire fits or just small sections of code. For example, that package allows to simply test different versions of equivalent instructions to see which one 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 interpolating a variable, ensuring to treat it as a local variable inside the benchmark and therefore avoiding any performance 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 also used more low-level analysis tools such as the `--track-allocation` option which asks Julia to execute the code and annotate, line by line, the source file to see where and of which amount allocations occurred.

2.1.1 Optimizing spatial cohesions

The memory allocation problem was especially visible in the spatial cohesion computation. In fact, such computation appears in both sections of updating γ_{it} and updating ρ_t , 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 dTn and dTn^2 , being d the number of iterations, T the time horizon and n the number of units. We can't provide more precise estimate due to the variability and randomicity 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.

The only optimizing task consisted in carefully designing the cohesion function implementations. 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. 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 first-thought vector implementation 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
```

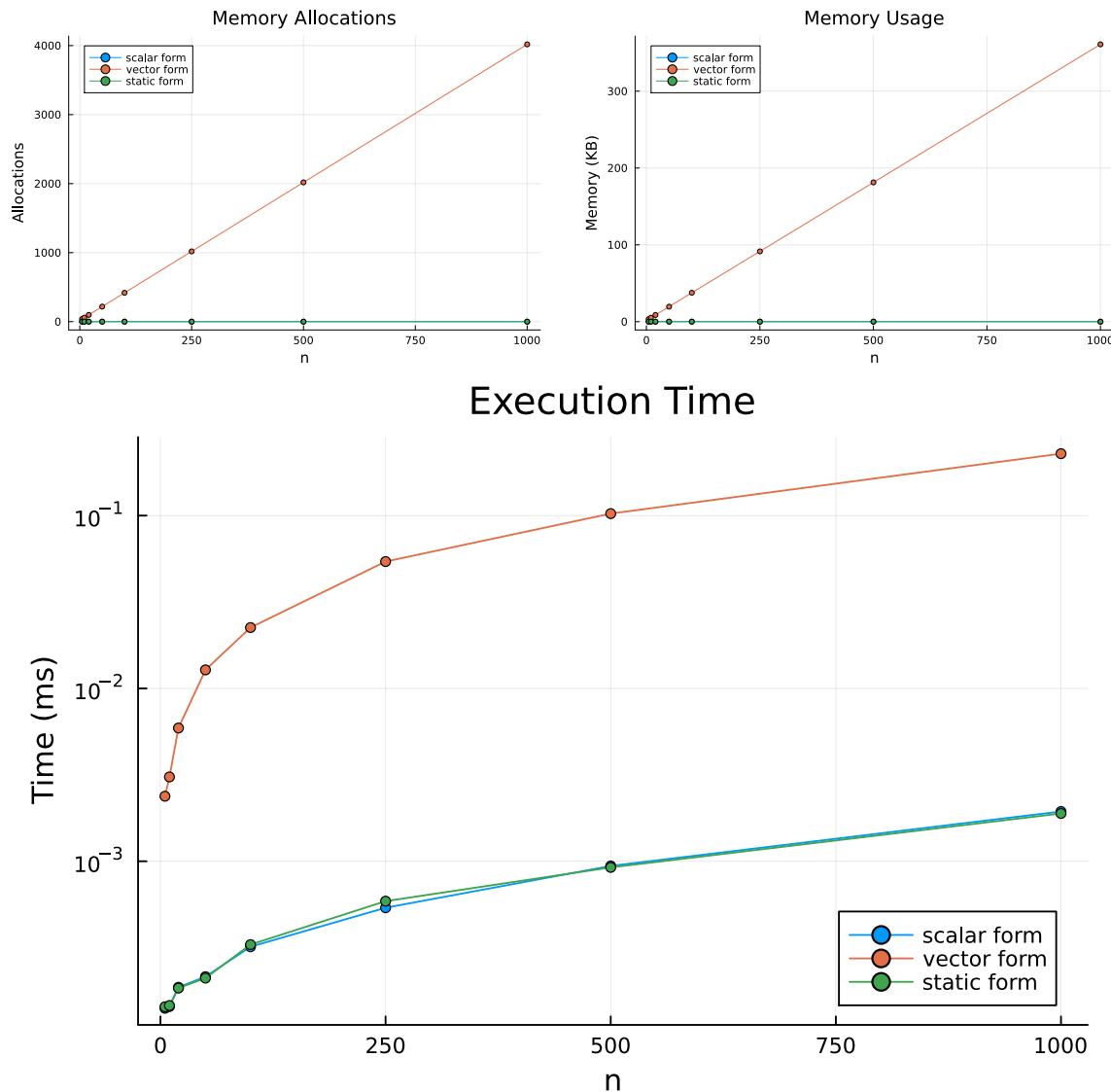


Figure 2.2: Performance comparison between the three versions of the cohesion 4 function. Tests ran through the `BenchmarkTools` package of Julia by randomly generating the spatial coordinates of the various test sizes n , with similar results standing for cohesion 3. The memory allocation and usage plots are constant at zero for both the scalar and vector static cases.

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 2.2 shows the comparison of their performances, where we can see how the scalar and static versions indeed perform very similarly to each other, and more quickly with respect to the first version.

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 and was therefore forced to the scalar implementation.

2.1.2 Optimizing covariates similarities

Another problem has been understanding how to speed up the computation of the similarity functions, since those would also be called possibly millions of times as the cohesion ones or even more, considering that we can incorporate more than one covariate into the clustering process, and this would add 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 because it is one the most common choice among 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: Version of the similarity 4 function with all the possible optimizing macros. 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 @fastmath @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 those checks to save some execution time. This insertion is harmless as long as we are sure that in our code design no out-of-bounds

or wrong accesses will occur. Otherwise some 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 function, where we are just 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.

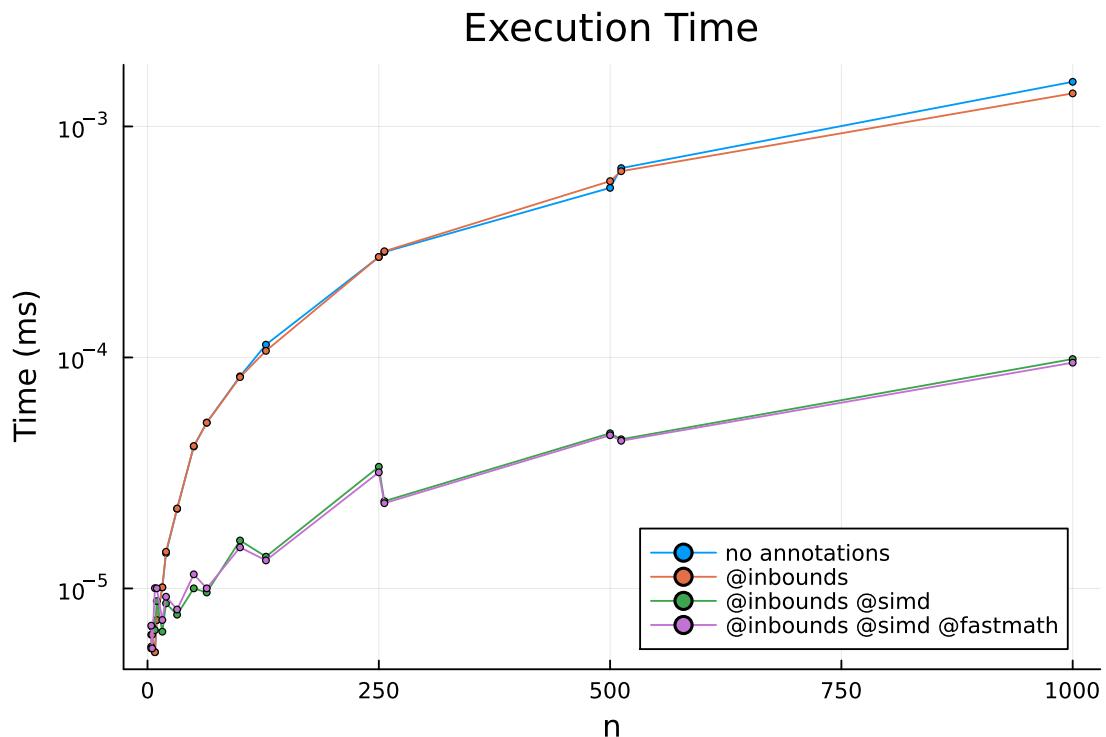


Figure 2.3: Comparison of the performances of the different possible loop annotations in the similarity 4 implementation. Their numerical output results are indeed the same for all cases. There are no memory allocation and usage plots since the analysis has been conducted only to evaluate the performances of the inside loop, which has no memory issues.

As we can see from Figure 2.3, the actual performance difference basically derives just from the use of `@simd`, with the other two annotations making not

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 behaviour of arithmetic operations on them, including precision, rounding, and handling of special values (e.g. NaN (Not a Number) and infinity).

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`.

We can also notice interestingly how the tests with `@simd` annotation run quicker in the case of n being a power of two, compared to their closest rounded integers (e.g. 256 against 250 or 512 against 500), despite having relatively 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 3

Testing

3.1 Assessing the equivalence of the models

Our model, and the corresponding Julia code, is just an improvement of the original DRPM 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 synthetic data, while the second including spatial information, using a real spatio-temporal dataset. For the sake of clarity, also in these sections we will refer to CDRPM for the original model and implementation from [PQD22], while to JDRPM for the updated version derived from this thesis work.

In the following analysis we will heavily rely on the Adjusted Random Index (ARI) [HA85] to compare the partitions generated by the models. The ARI index is a correlation index to measure similarities between clusterings. More precisely, given two partitions ρ_1 and ρ_2 , the $\text{ARI}(\rho_1, \rho_2)$ returns a value between $[-1, 1]$ where higher values indicates higher levels of agreement between the partitions, i.e. they produced similar clusters. Being a *random* index, it has an expected value and it is equal to zero, which corresponds to the case of comparing two randomly generated partitions.

We will use this index both to study the time evolution of the clusterings, to see e.g. if ρ_{t+k} is correlated to ρ_t , that is, how the clusters show the time dependent structure that the models implement, and also to check the level of agreement between the two models, comparing the clusters produced by CDRPM and JDRPM.

All the following tests have been performed on my laptop, which for performance references has 8 GB of RAM and 1.80 GHz CPU base clock speed, on the software R [R C24] and trough the `JuliaConnectoR` library [LHB22], which allowed the interface between R and Julia where JDRPM is implemented. The DPRM model

corresponding to the original formulation was already available in R through the `drpm` library.

3.1.1 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. Both models were fitted using the full formulation, i.e. including and updating also the optional autoregression parameters η_{1i} , and φ_1 , and using a time specific α .

Table 3.1: Summary of the comparison between the two fits on target values only. 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	exec. time
CDRPM	1.6731	1.5861	-249.61	469.69	4.8s
JDRPM	1.2628	1.2181	-227.83	415.03	2.5s

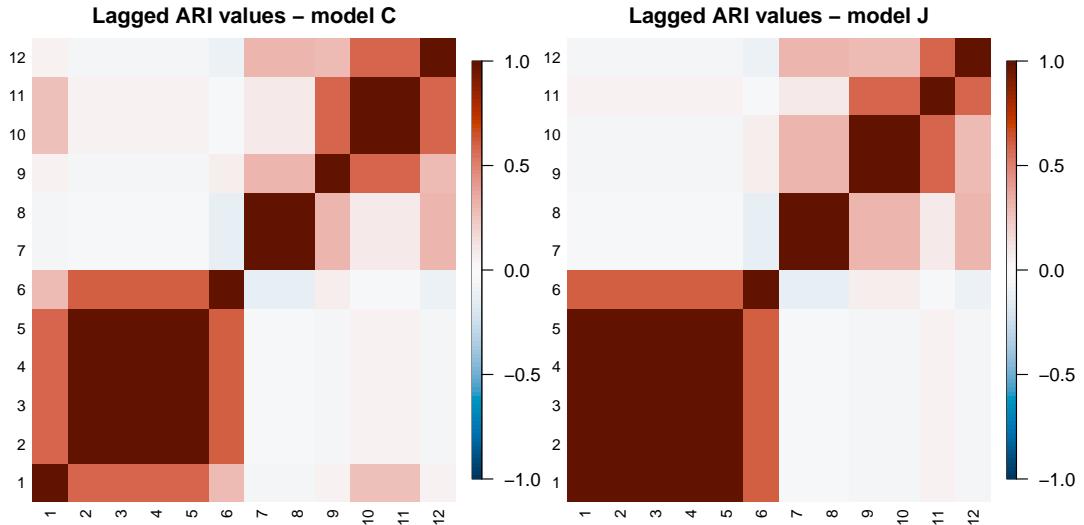


Figure 3.1: Lagged ARI values for the two models, on the fit with target only data. The partitions were estimated using the `salso` function from the corresponding R library.

At this testing stage there were 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 3.1. The JDRPM model had better fit metrics (lower WAIC and higher LPML) and also faster execution time, which is however not really relevant in this small-sized fit. Regarding the fitted values, displayed in Figure 3.4 together with

the original data, they turned out to be also more precise than the one generated by CDRPM, having lower mean squared errors.

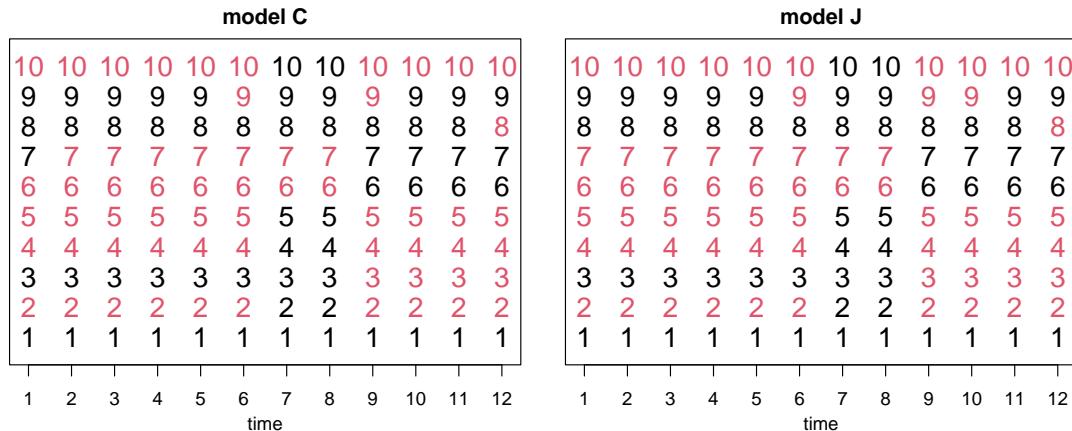


Figure 3.2: Clustering produced by the two models, with time points on the x axis, units indicated vertically by their number, and colors representing the cluster label.

The resulting clusters were indeed very similar. Figure 3.1 shows how they manifest the same temporal trend, while Figure 3.2 highlights how the clusters produced are effectively the same except for two differences occurred at times 1 and 10. By visually inspecting the clusters from Figure 3.3, 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 to 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.

3.1.2 Target variable plus space

We now consider a more realistic scenario in which we fit the models using a spatio-temporal dataset. In particular, we used the AgrImOnIA [Fas+23] dataset which stores measurements about air pollutants, together with many other environmental variables, in the Lombardy region from 2016 to 2021. For the following testing fits, we employed a summary dataset composed by weekly averages of the data from year 2018.

The chosen target variable was PM_{10} whose values had been log-transformed, to recover a normal distribution, and centered with respect to time-wise means. More precisely, each unit Y_{it} was corrected by subtracting the mean \bar{Y}_t of all the observations of that time instant. This procedure, which was the same employed by the authors of the original DRPM during their spatial tests, helps to emphasize the variations within each time point rather than across the entire dataset. This can be particularly useful to understand how units deviate from their typical behaviour at specific times, potentially highlighting temporal trends and anomalies. This method

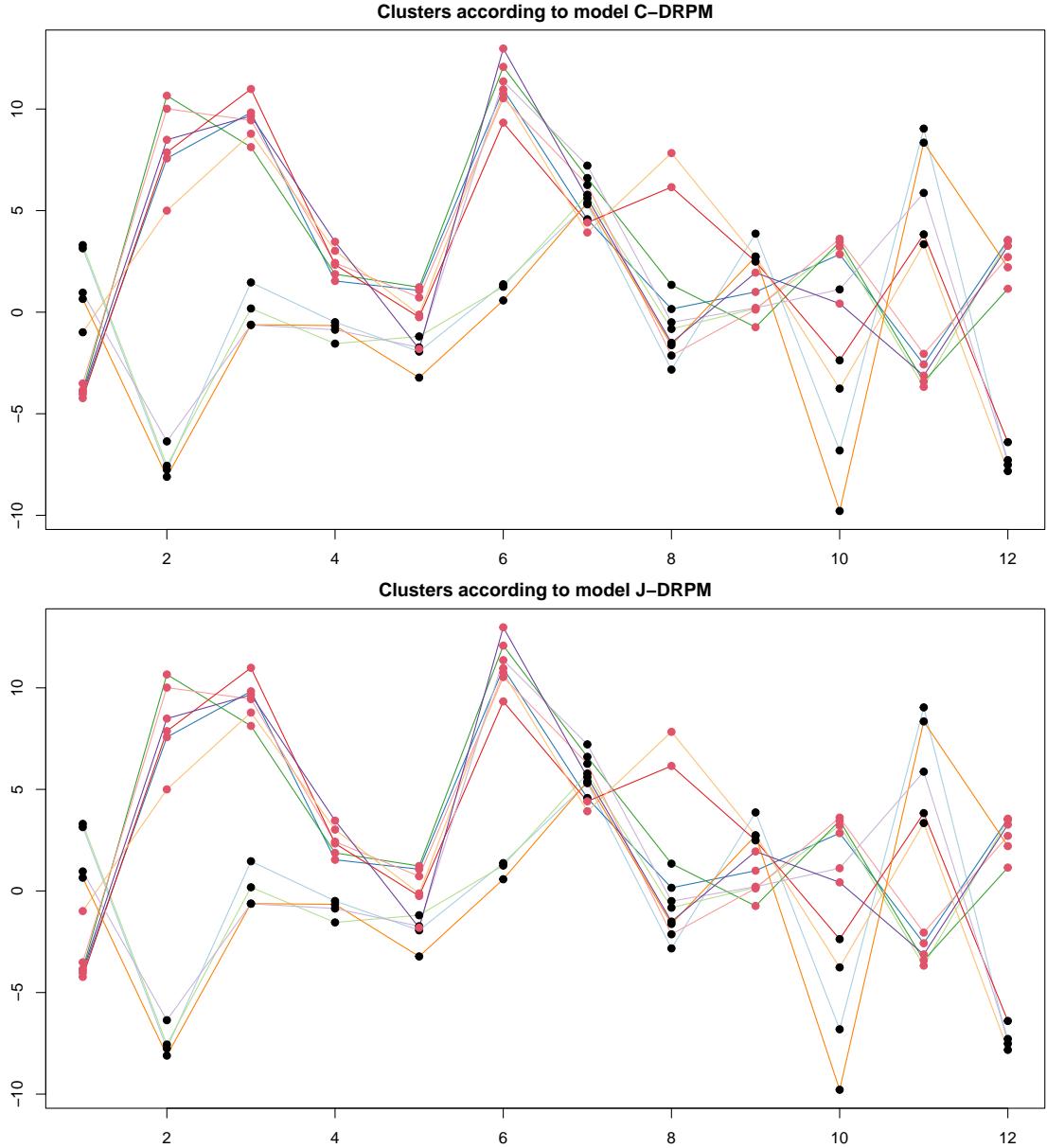


Figure 3.3: Visual representation of the clusters produced by the models, on the fit with target only data, with units' labels represented as colored dots overlaid to the trend of the generated target variable. The only differences occur at times 1 and 10.

also removes any temporal bias, e.g. time instants in which all target values were particularly high or low for any particular reason. In contrast, the more traditional choice of centering with respect to the global mean \bar{Y} of the dataset would be useful to just detect global trends, without inspecting deeply into each step. From some test fits performed on the target values with the classical global centering indeed only three clusters appeared, for all time instants. This is of course correct, in some sense, since it's reasonable according to the trend displayed in the top image of Figure 3.5. However, as we will see shortly, the other version was able to produce several clusters, or at least different partitions at different time instants, improving the specializations of the clusters and the interpretability of the obtained results.

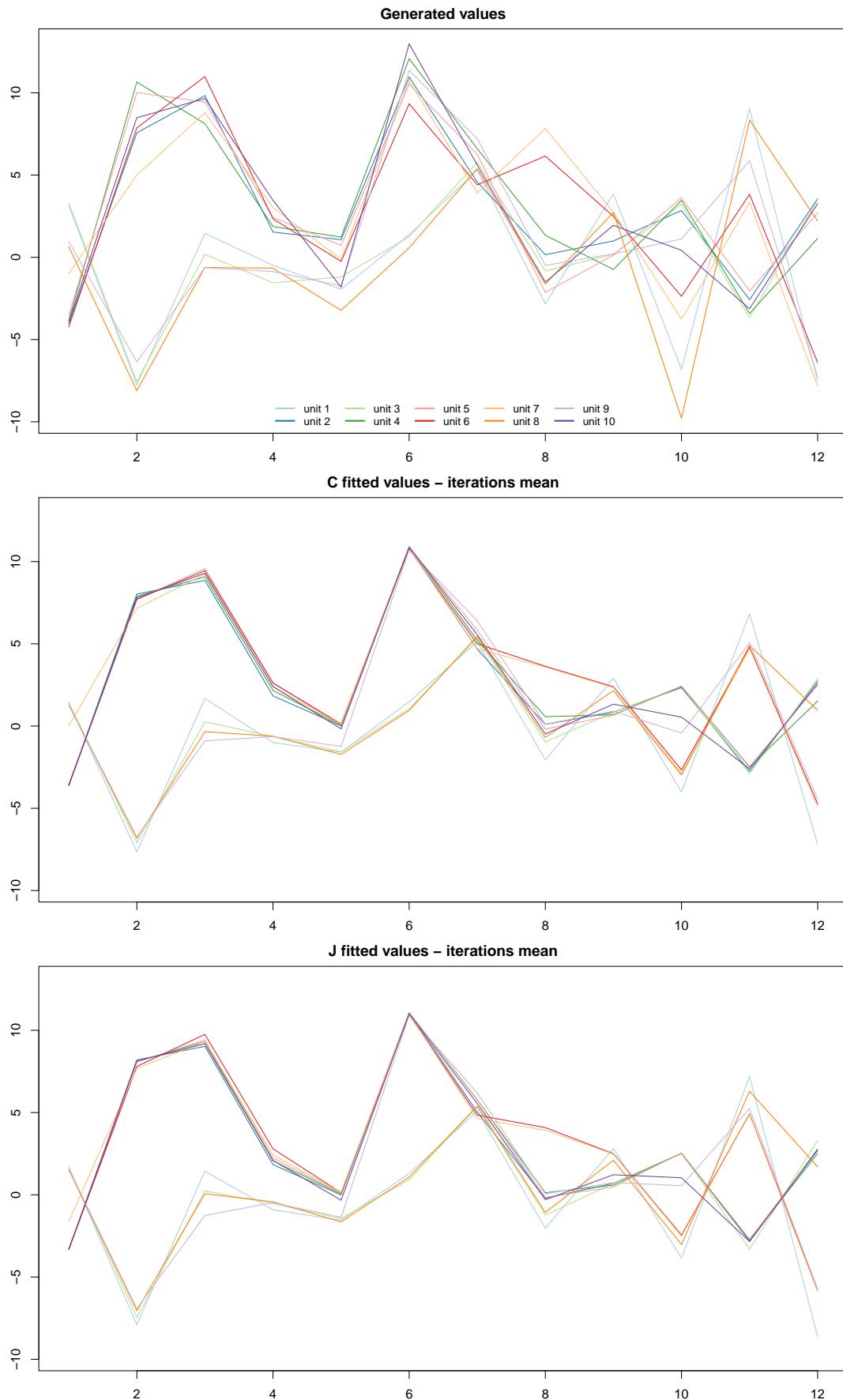


Figure 3.4: Generated target values (top), together with their fitted values estimate through the mean of the 1000 iterates generated by model CDRPM (middle) and JDRPM (bottom).

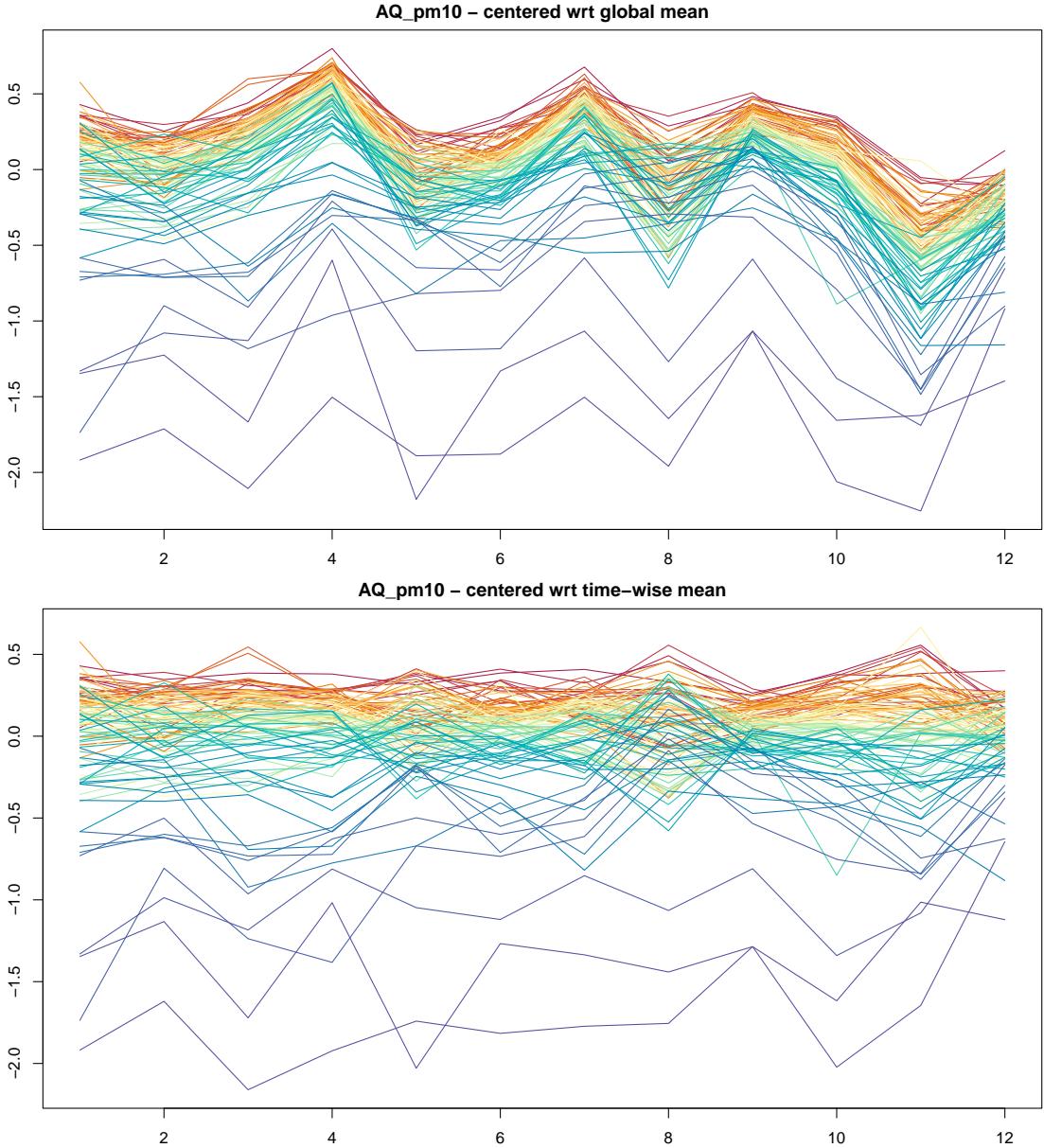


Figure 3.5: Values of the target variable AQ_{pm10} corrected using the global mean (top) and using the time-wise mean (bottom). This second choice helps to adjust the values range while keeping, or even highlighting, the clustering structural information. Coloring is based on the ranking of PM_{10} values of the units according to their median, from highest (red) to lowest (blue).

We took all the $n = 105$ stations available in the dataset but we limited the time horizon to $T = 12$, which corresponds to a three-month monitoring period, just to reduce the time needed to perform all tests. We ran both CDRPM and JDRPM models using the same complete modelling setup, as in Section 3.1.1, and using a time specific α . Convergence was ensured visually by inspecting trace plots and, for the JDRPM model, by numerical diagnostics such as ESS and \hat{R} , which instead the CDRPM model is not able to provide directly from the fit. We collected 4000 iterates, from 110000 total iterates with a burnin of 90000 and thinning by 5.

Table 3.2 shows how both model managed to be accurate not only with respect

Table 3.2: Summary of the comparison between the two fits on target plus space values. 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 4000 iterates) and the true values of the target variable. Higher LPML and lower WAIC indicate a better fit.

	MSE mean	MSE median	LPML	WAIC	exec. time
CDRPM	0.0142	0.0149	694.81	-1768.42	1h38m
JDRPM	0.0131	0.0138	624.91	-1898.05	48m

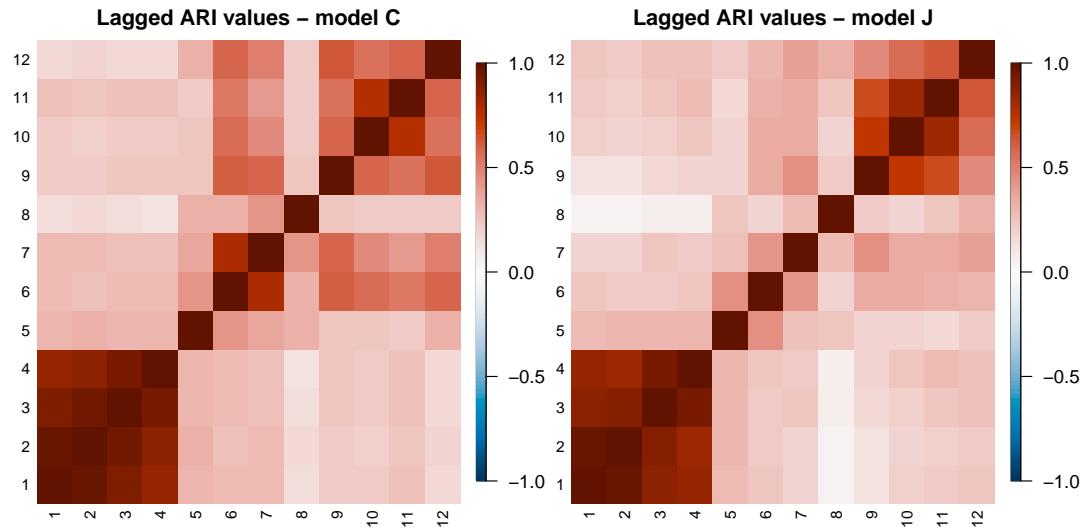


Figure 3.6: Lagged ARI values for the two models, on the fit with target plus space data. The partitions were estimated using the `salso` function from the corresponding R library.

to the fitting metrics of LPML and WAIC but also in terms of the fitted values. Execution times are also reported, but they are somewhat inaccurate since during the fitting I was busy in writing this thesis, meaning that not all the computational resources of my laptop were devoted to the fitting task. In any case, more precise performance evaluations on the timings will be conducted in Section 3.4.

From Figure 3.6 we can see how the temporal trend was very similar, confirming again the correctness of the JDRPM implementation. Regarding the similarity of the produced clusters, the partition plot as the one of Figure 3.2 would now be more crowded due to the high number of units. For this reason, in order to still convey that information, we computed $\text{ARI}(\rho_{\text{JDRPM}}(t), \rho_{\text{CDRPM}}(t))$ for all time instants $t = 1, \dots, 12$, and we obtained a mean of 0.80 and a median of 0.86, denoting high levels of agreement in the clusters produced.

A visual representation of the clusters is provided in Figure 3.8, but will be further detailed and commented in Section 3.3.2 together with the fits including covariates in the clustering.

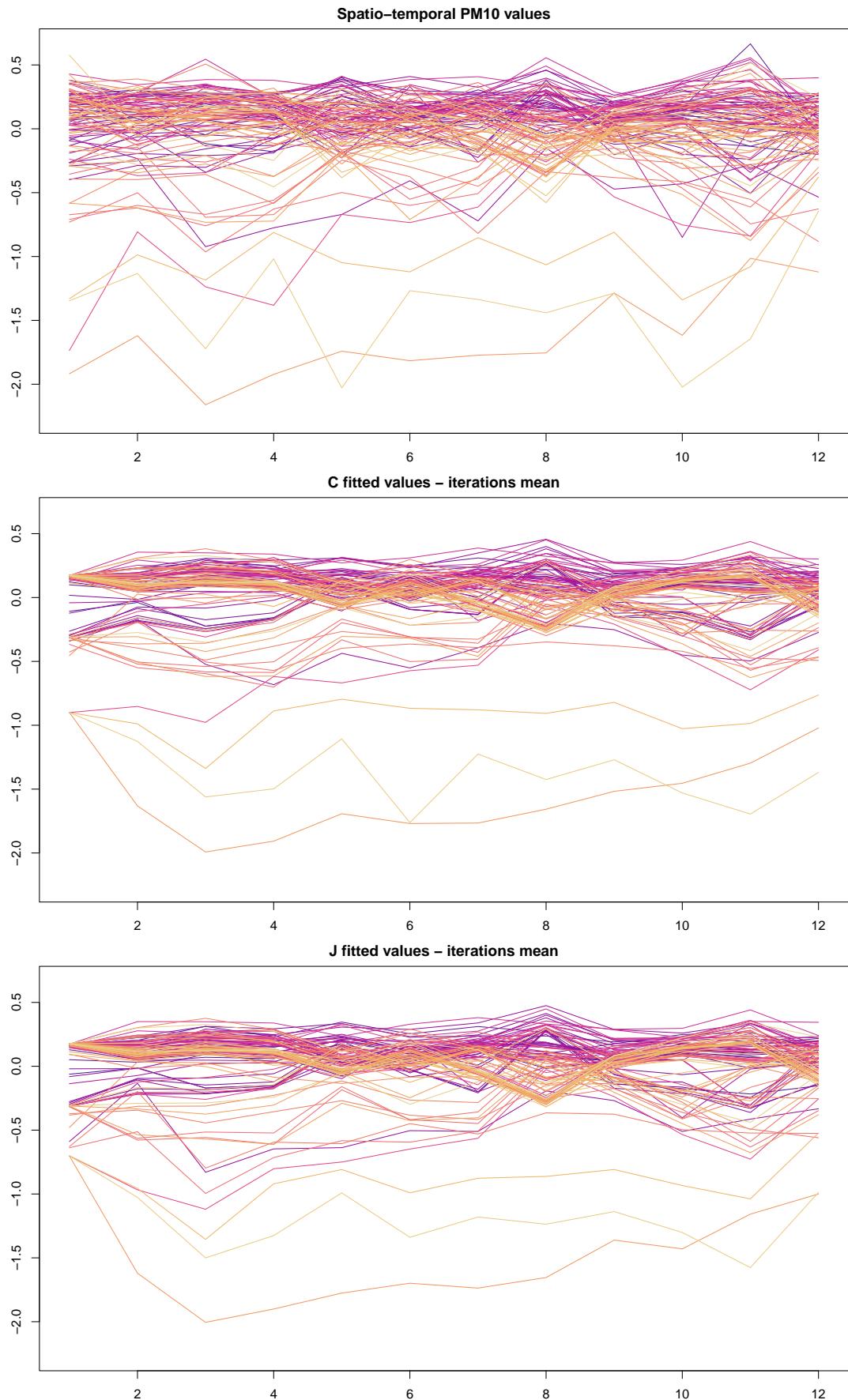


Figure 3.7: Target Y_{it} values (top), together with their fitted values estimate through the mean of the 4000 iterates generated by model CDRPM (middle) and JDRPM (bottom).

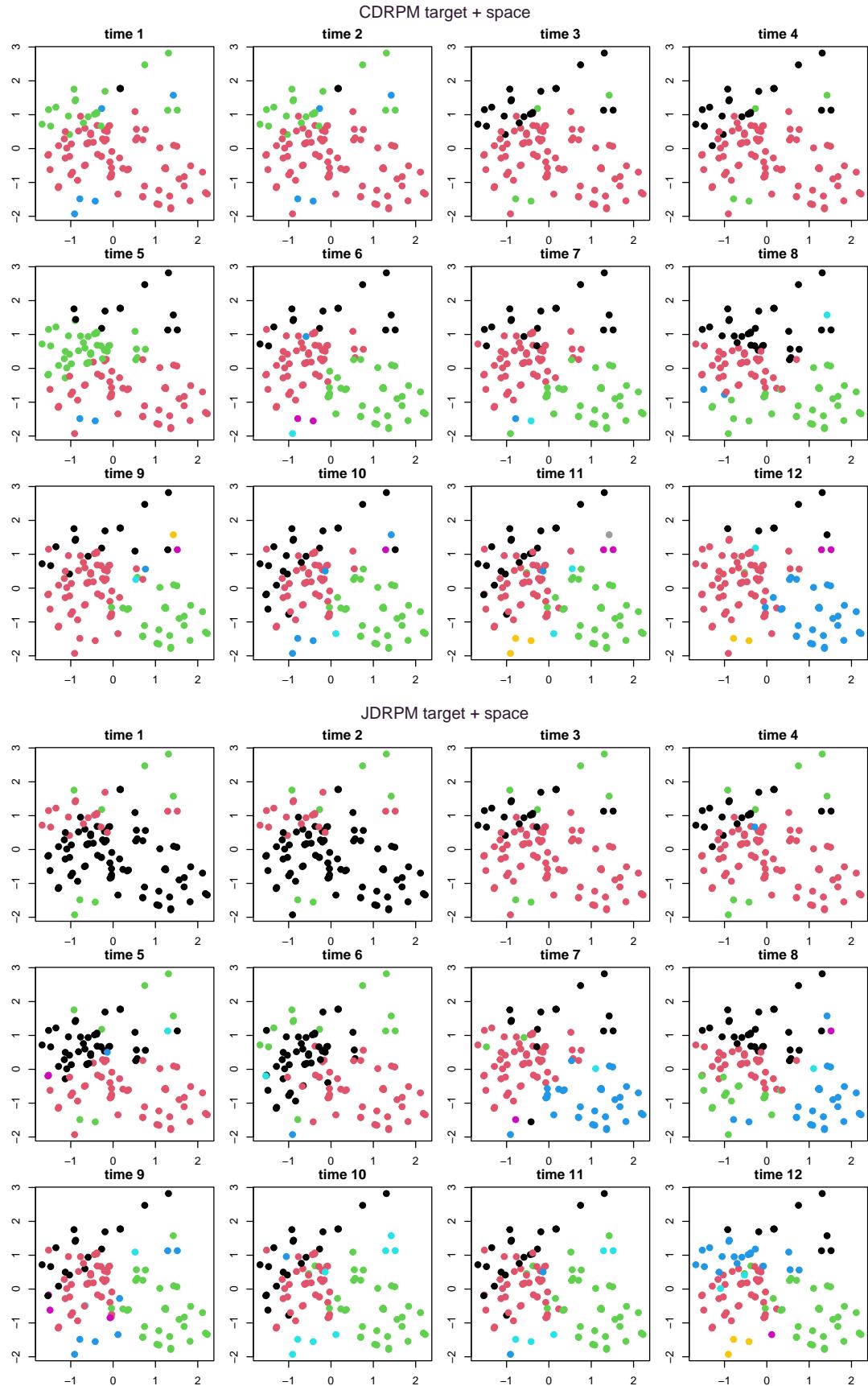


Figure 3.8: Clusters generated by the CDRPM and JDRPM fits with target plus space values.

3.2 Performance with missing values

We now repeat the tests of the previous section on the same datasets but this time with missing values, to see how the JDRPM model reacts to the absence of a full dataset and check if it can still perform well. Based on the amount of missing values in the AgrImOnIA dataset, used for the spatio-temporal tests, we chose to set at 10% the amount 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 target variable Y_{it} . The dealing of NAs was an upgrade brought by the JDRPM model, so we can't perform these tests with the original CDRPM model since it cannot handle missing data.

All the following fits have been conducted under the same conditions of their full-dataset counterpart, i.e. using the same models formulation and parameters.

3.2.1 Target variable only (NA case)

The JDRPM seems to exhibit good performance also with missing values. From Table 3.3 we can see how the performances, naturally worse than the fit in 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, but the fitted values of Figure 3.9 are still close the one obtained with the fit on full dataset.

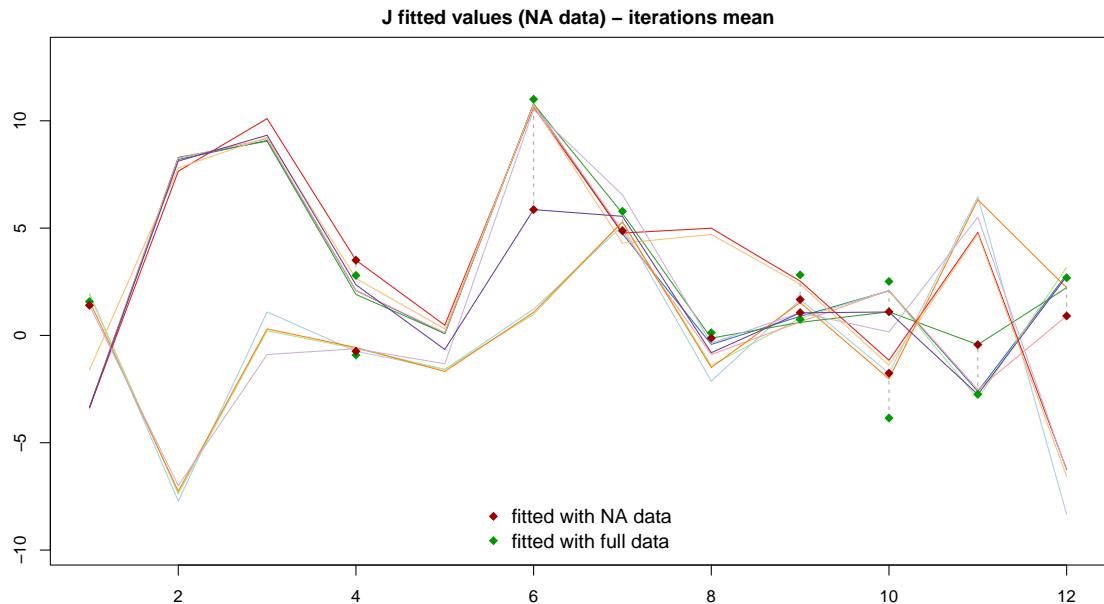


Figure 3.9: Fitted values estimate through the mean of the 1000 iterates generated by model JDRPM. The generated target values were the same of Figure 3.4. Special point markers are used for the data points which were missing, to highlight the gaps between the fitted values on the full dataset (green squares) and the fitted ones on the NA dataset (red squares).

From Figures 3.11 and 3.10 we can see how we get almost the same temporal

trend which we saw in the case without missing data, and 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 3.1.1.

Table 3.3: Summary of the comparison between the two Julia fits on target values only. 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.

JDRPM	MSE mean	MSE median	LPML	WAIC	exec. time
NA data	2.1819	1.7876	-239.21	417.21	3.0s
full data	1.2628	1.2181	-227.83	415.03	2.5s

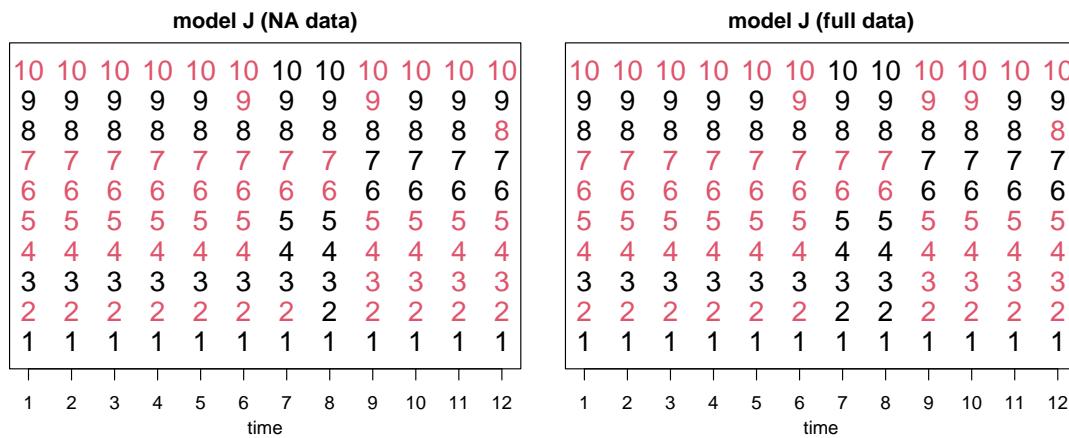


Figure 3.10: Clustering produced by the two tests on the JDRPM model, with time points on the x axis, units indicated vertically by their number, and colors representing the cluster label.

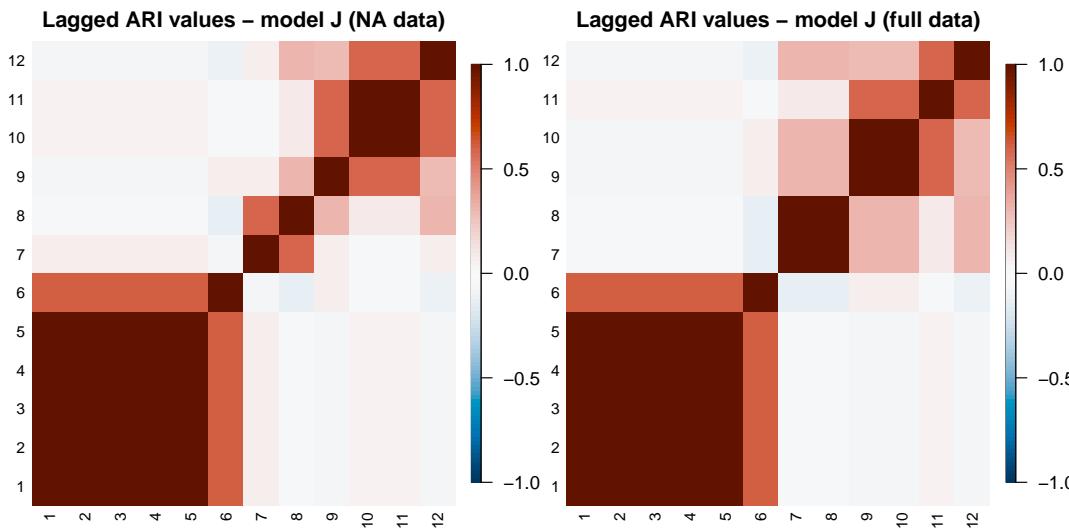


Figure 3.11: Lagged ARI values for the two fits on the JDRPM model with target values only. The partitions were estimated using the `salso` function from the corresponding R library.

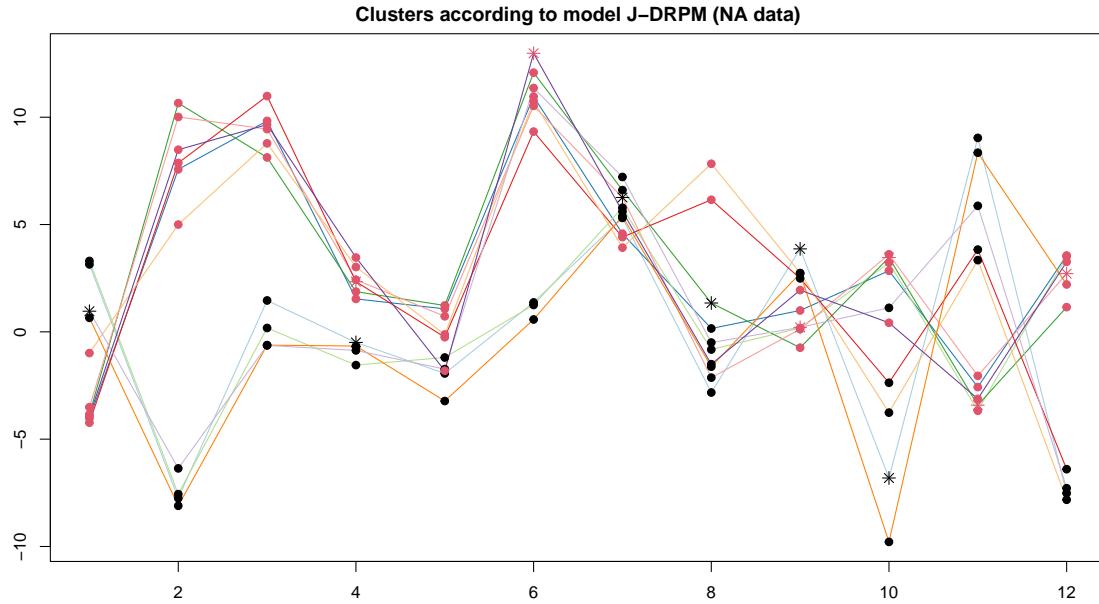


Figure 3.12: Visual representation of the clusters produced by the model, with units' labels represented as colored dots overlaid to the trend of the original generated target variables. Special point markers are used for the data points corresponding to missing values.

3.2.2 Target variable plus space (NA case)

Table 3.4: Summary of the comparison between the two Julia fits on target plus space values. 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 4000 iterates) and the true values of the target variable. Higher LPML and lower WAIC indicate a better fit.

JDRPM	MSE mean	MSE median	LPML	WAIC	exec. time
NA data	0.0160	0.0170	502.86	-1793.64	43m
full data	0.0131	0.0138	624.91	-1898.05	48m

The JDRPM model seems able to perform well also in the case of fits including spatial information. Table 3.4 shows how the accuracy reduces, which again is inevitable since we are fitting with missing data, but not drastically. In fact, the drops in LPML and WAIC metrics are just of 3.16% and 0.95% respectively. Fitted values are reported in Figure 3.14. Regarding the reported execution times, they are not completely truthful, as already said before, due to the not-full commitment of my laptop resources to the fitting task. In fact, the fit with NA data appeared to be faster than the one with full data, which is somewhat implausible since in the presence of NA data there is the additional load of updating the missing Y_{it} values. Again, more accurate results will be provided in Section 3.4.

The temporal trend managed as well to remain similar to the trend we saw in Section 3.1.2, as proved by Figure 3.13. Noticeably, the ARI plot of the fit with NA data managed to resemble more the original trend displayed by the CDRPM fit on the full data; this same fact happened also in the previous section on fits without the additional spatial information. Regarding the similarity of the produced clusters,

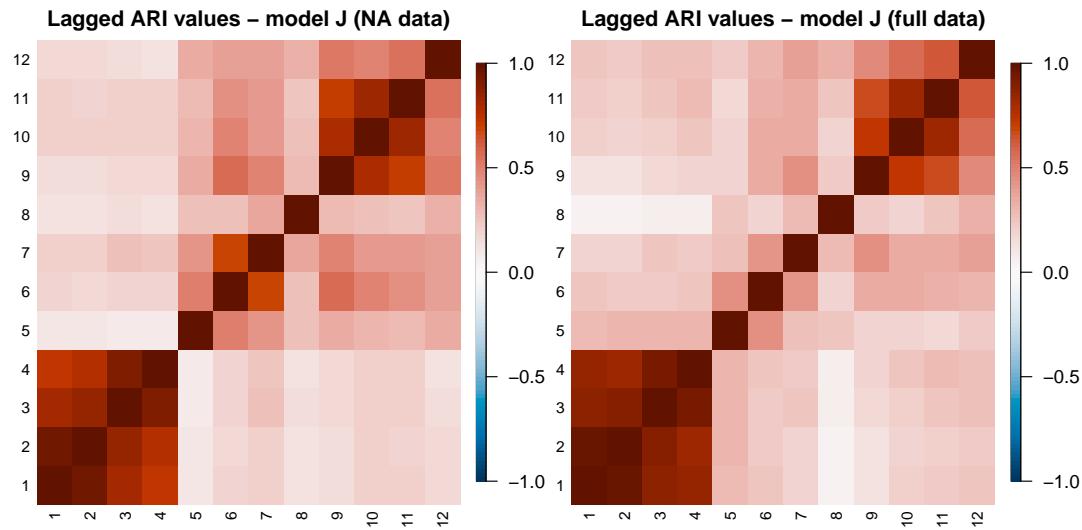


Figure 3.13: Lagged ARI values for the two fits on the JDRPM model with target plus space values. The partitions were estimated using the `salso` function from the corresponding R library.

we computed the values $\text{ARI}(\rho_{\text{JDRPM_NA}}(t), \rho_{\text{JDRPM_full}}(t))$ for all time instants $t = 1, \dots, 12$, and we obtained a mean of 0.82 and a median of 0.86, denoting still a relatively high level of agreement in the partitions despite the loss of quite some data (121 points out of the 1260 of the full dataset).

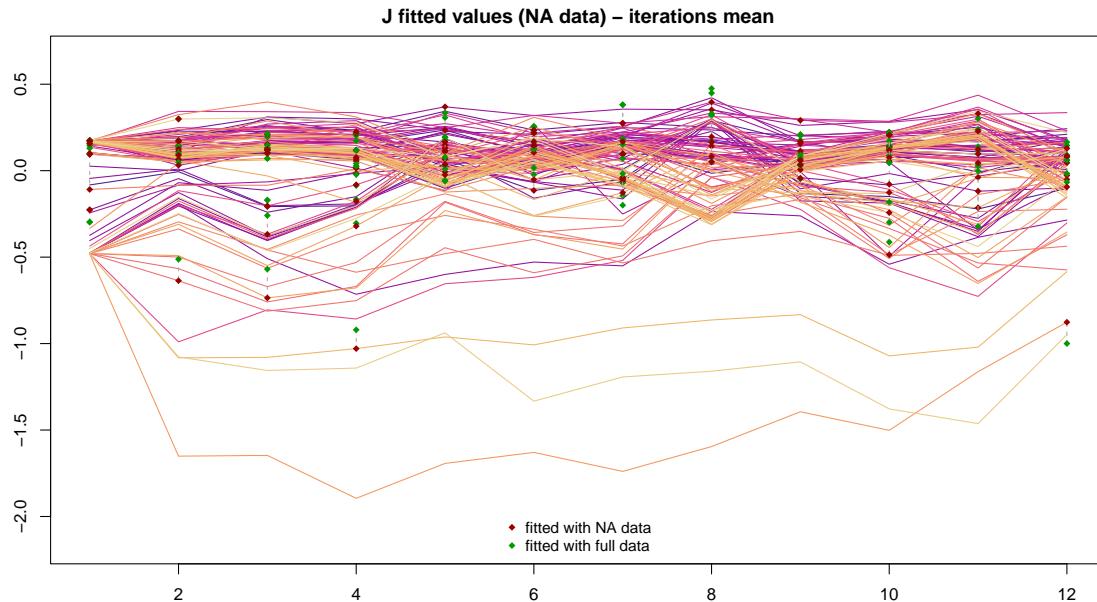


Figure 3.14: Fitted values estimate through the mean of the 4000 iterates generated by model JDRPM. The generated target values were the same of Figure 3.7. Special point markers are used for the data points which were missing, to highlight the gaps between the fitted values on the full dataset (green squares) and the fitted ones on the NA dataset (red squares).

3.3 Effects of the covariates

To perform the fits that will now follow, all the included covariates were treated in the same way as the target value, with the time-wise centering procedure and reasoning described in Section 3.1.2. Figure 3.15 provides an exemplification of the effect of this transformation on one of the covariates. Again, this was performed in order to remove any trend or bias on the covariates and to equalize their contribution at each time instant.

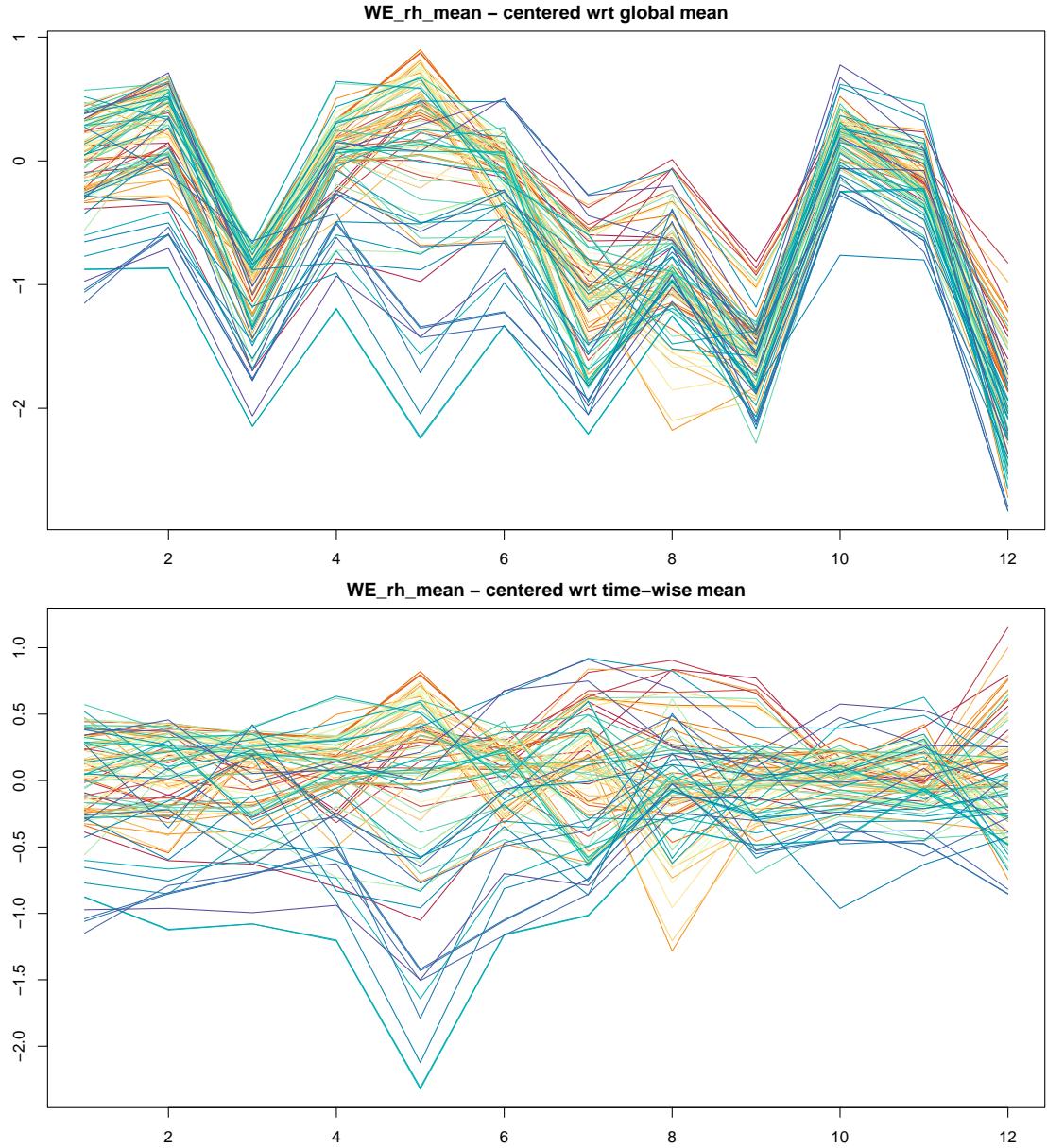


Figure 3.15: Values of the `WE_rh_mean` centered covariate corrected using the global mean (top) and using the time-wise mean (bottom). Coloring is based on the ranking of PM_{10} values of the units according to their median, from highest (red) to lowest (blue).

We will now see some testing fits which employ the core advancement of the JDRPM update, i.e. the inclusion of covariates. Being with radically different

purposes, we will distinguish the cases of covariates in the likelihood and covariates in the clustering.

3.3.1 Covariates in the likelihood

Continuing on the line of the previous section, we can test if the insertion of covariates in the likelihood can help to recover some accuracy in the case of fits with missing data. In fact, the addition of the regression parameter β_t had the main goal of improving the accuracy of the model when estimating the target values Y_{it} , without affecting too much the cluster generation. With that in mind, the most natural use case of such parameter would be when fitting with missing values.

Table 3.5: Summary of the comparison between the JDRPM fits, on target plus space values, plus the fits including also covariates in the likelihood. 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 4000 iterates) and the true values of the target variable. Higher LPML and lower WAIC indicate a better fit.

JDRPM	MSE mean	MSE median	LPML	WAIC	exec. time
full data	0.0131	0.0138	624.91	-1898.05	48m
full data + Xlk	0.0112	0.0113	778.96	-2029.84	56m
NA data	0.0160	0.0170	502.86	-1793.64	43m
NA data + Xlk	0.0127	0.0130	625.81	1902.74	58m

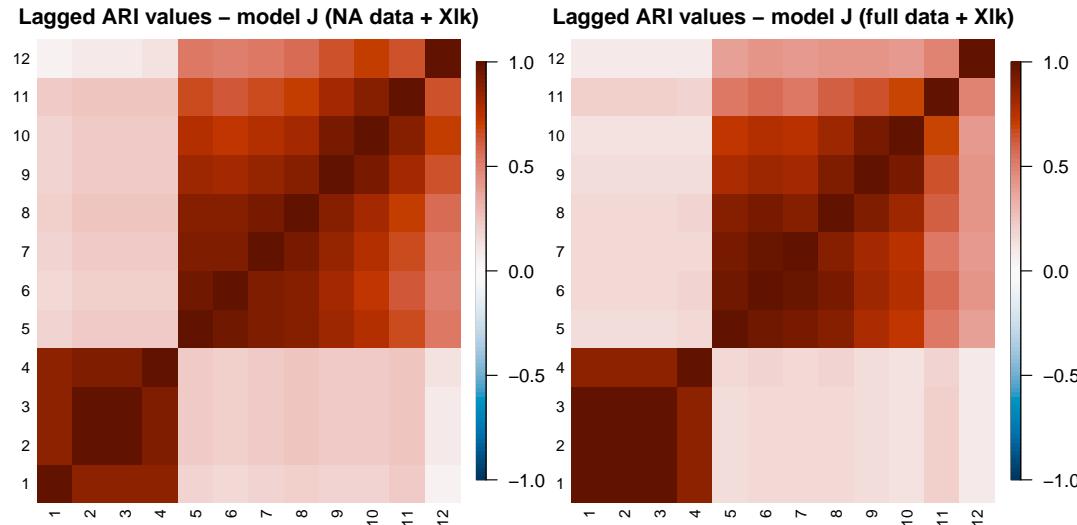


Figure 3.16: Lagged ARI values of the two JDRPM fits with target plus space values on the full and NA datasets, with covariates in the likelihood.

Indeed, after including one or multiple covariates in the likelihood, and repeating the fit on the NA dataset under the same conditions, apart from this addition, we saw some improvements. To better illustrate them, we also ran the corresponding fit using the same covariates set in the likelihood, but this time on the full dataset.

Results are summarized in Table 3.5 which also includes, as a reference, the results of the previous fits without the regressor component. Again, the execution times have to be taken with a pinch of salt.

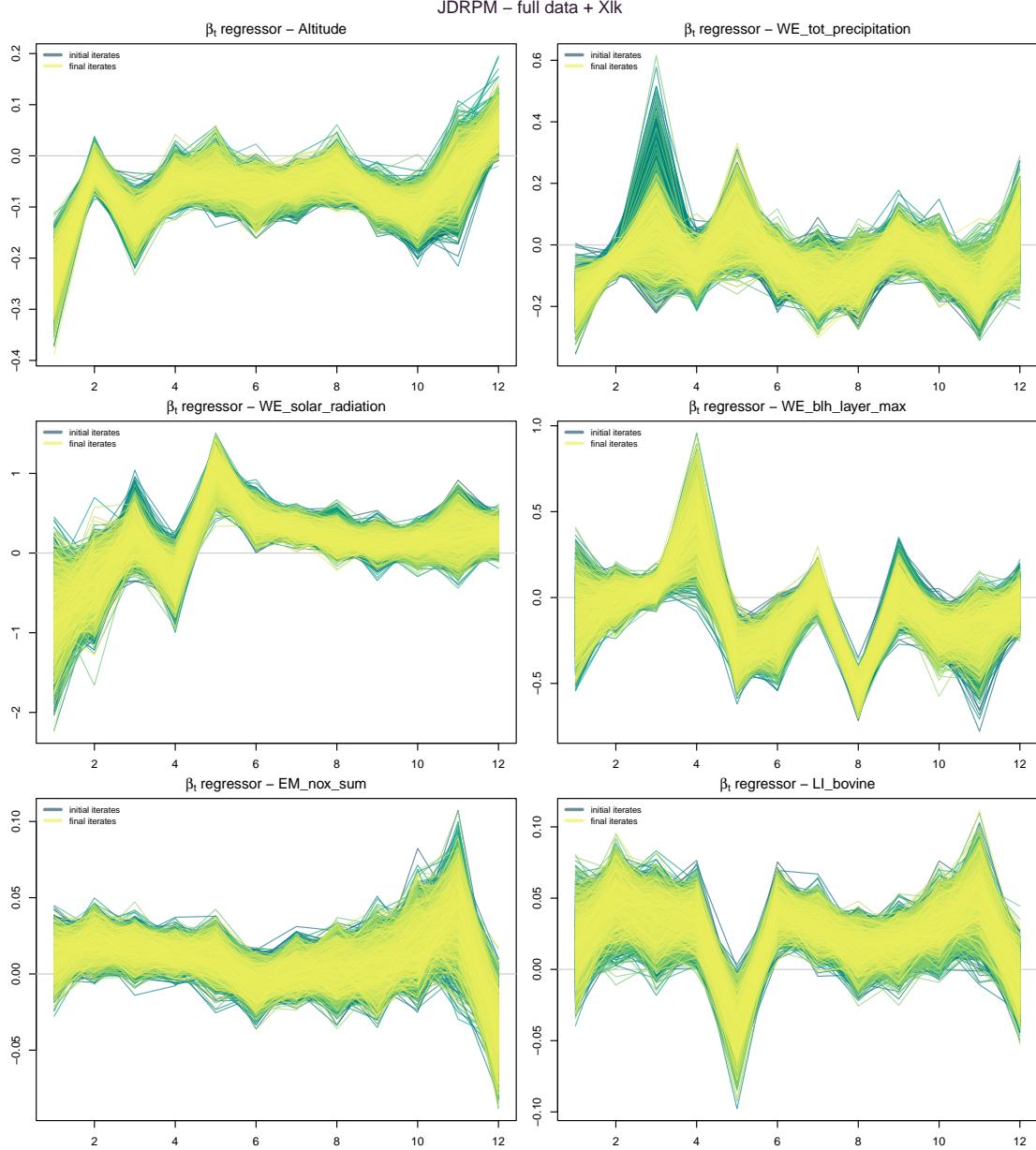


Figure 3.17: Regression vector β_t for the $p = 6$ covariates inserted in the likelihood in the full data spatio-temporal fit test, with time instants $t = 1, \dots, 12$ are on the x axis.

We can see how clearly the insertion of covariates in the likelihood managed to improve all fitting metrics with, as expected, a decrease in the MSEs, especially in the fits with NA data, proving how that insertion can indeed recover some performance in the case of missing values. The regression vectors β_t displayed good trace plots, represented in Figures 3.17 and 3.18, proving the correctness and effectiveness of the implementation. Moreover the values at which the regressors settled seems to be reasonable and interpretable. For example, it is known that higher altitudes implies lower air pollution, due to the nearly absence of emission

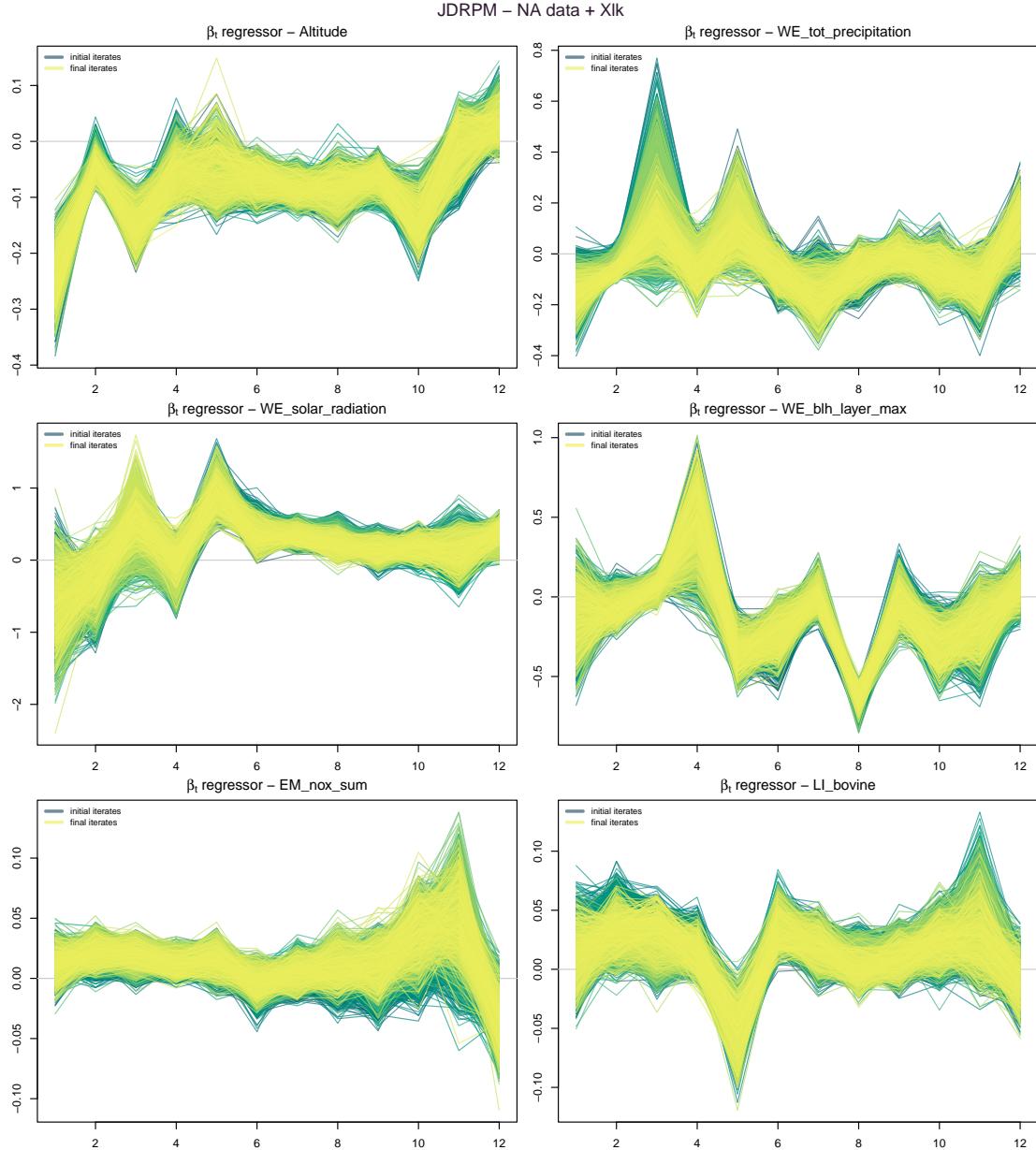


Figure 3.18: Regression vector β_t for the $p = 6$ covariates inserted in the likelihood in the NA data spatio-temporal fit test, with time instants $t = 1, \dots, 12$ are on the x axis.

sources like industries and vehicles, the more intense winds, the lower atmospheric pressure which facilitates the mixing of the air, and so on; and in fact the β_t component referring to `Altitude` wanders around negative values, indicating a reduction impact on PM_{10} concentrations. On the other hand, the `LI_bovine` covariate, which refers to the density of bovines per km^2 in the area of the measuring station, tends to stay around positive values, since indeed the presence of livestock industries contributes to the emission of air pollutants.

Regarding the generated partitions, they remained substantially unchanged, as briefly depicted in Figure 3.19. The spatio-temporal trend was as well preserved, displayed in Figure 3.16, except for a generally higher dependence in the second part of the time interval, after the same identified “change point” at $t = 4$.

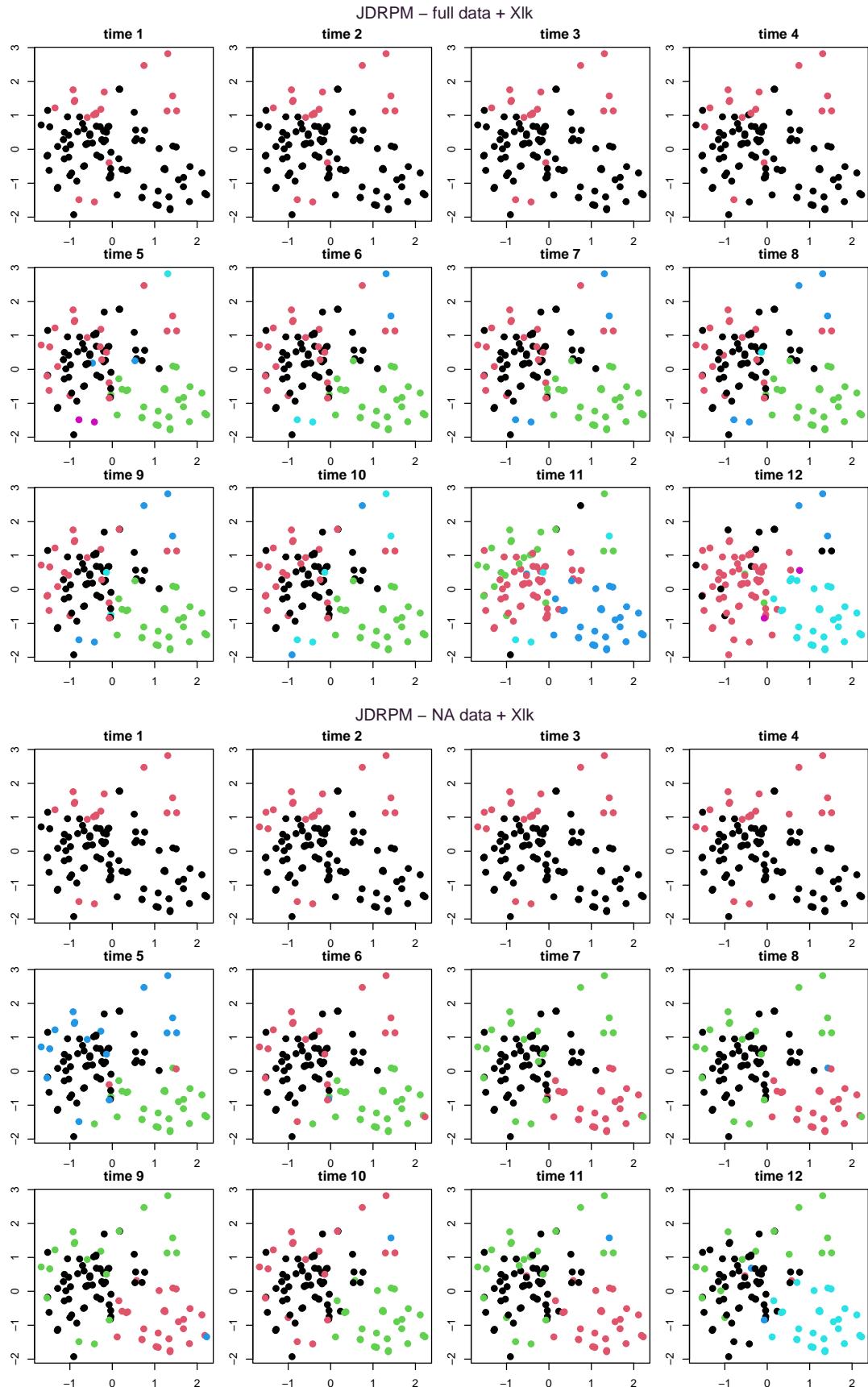


Figure 3.19: Clusters generated by the two JDRPM fits, with target plus space values, on the full and NA datasets, with covariates in the likelihood.

The JDRPM fitting function also includes a `beta_update_threshold` argument, defaulted to zero, to set after which iteration the algorithm should start to update the β_t parameter. This was an harmless and simple addition, which was considered useful to firstly let the model update and improve the more delicate and relevant parameters for the clustering, such as μ_{jt}^* and σ_{jt}^{2*} , and secondly tune also the less impactful β_t . Otherwise, maybe, the initial development of the model parameters and clusterings could be biased by early inaccurate samples of the likelihood regressor.

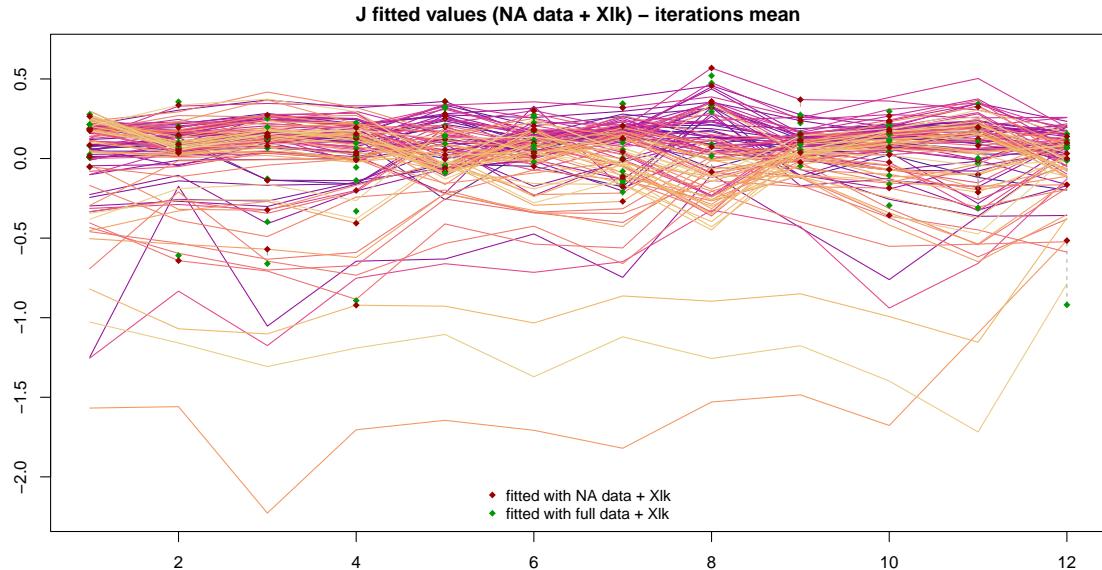


Figure 3.20: Target and fitted values of the JDRPM fits with target plus space values, on the NA and full dataset, to see the effects of the insertion of covariates in the likelihood.

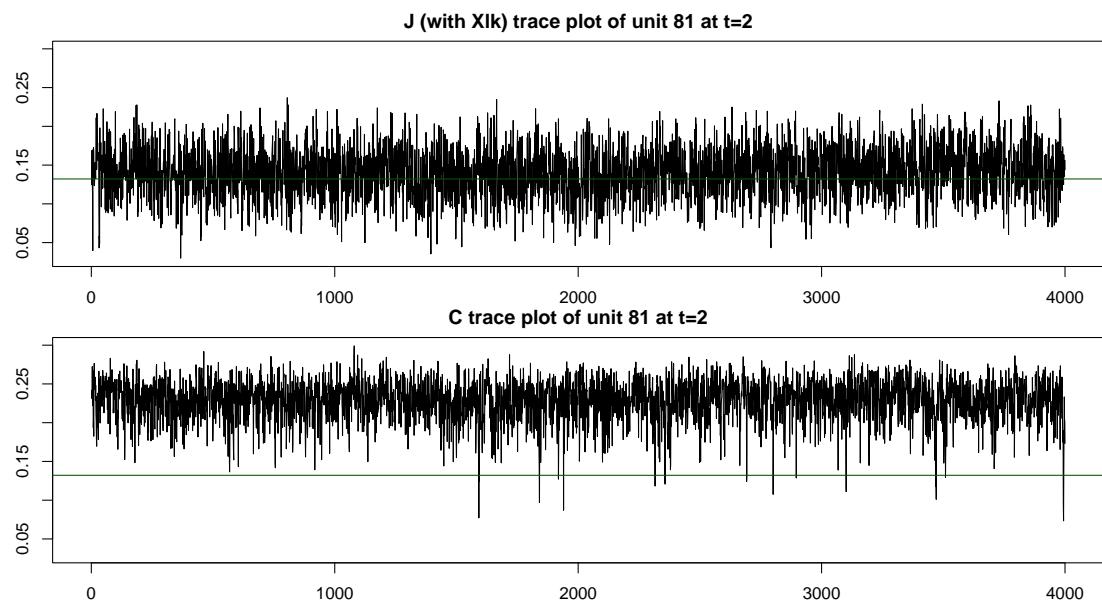


Figure 3.21: Example of a trace plot of the fitted values, on a problematic unit i and time instant t , comparing the JDRPM fit with covariates in the likelihood to the base (target plus space) CDRPM fit. The green line represents the true value of Y_{it} .

Finally, Figure 3.20 encloses the fitted values of NA run with covariates in the likelihood. We can see how indeed, visually but also as proved by the better MSEs, these fitted values resemble more the ones of the original target variable, especially if compared to the case of Figure 3.7, in which both fits were without any “external suggestion” of covariates in the likelihood. A further proof of this insertion is given by Figure 3.21, representing the trace plot of the fitted values for a problematic unit and time instant, with problematic in the sense that both the standard JDRPM and CDRPM fits didn’t manage to provide precise estimates, which now became more accurate. In the end, this analysis denotes how the addition of this regression parameter β_t could be beneficial to recover more accuracy in the results, and also help the clustering process since the regressor contribution manifests when using variables computed from the target values inside the update steps of the parameters.

3.3.2 Covariates in the clustering

Now we consider the more theoretically effective and impactful inclusion of covariates in the clustering process. For the choice of which covariates to include, we reasoned about which where the aspects that could affect more the PM₁₀ concentrations, and we ended up selecting the following three covariates:

- **WE_wind_speed_10m_max.** Wind speed plays a significant role in the concentration levels of air pollutants. Its effect is of dual: wind can disperse the pollutants away from their source, spreading them to different areas and thus locally reducing the levels, but it can also have an opposite effect, increasing the amount of contaminants in the air by resuspending the particles which were settled down on the earth surface, such as roads, soils, or buildings. This latter effect is particularly visible in dry and windy rural areas, of which Lombardy is a suitable example. The dataset offered also the wind speed at 100m, but this would be relevant for a wider analysis, where e.g. the trend of PM₁₀ concentrations are followed over multiple regions or countries. Our setup, as with the time-wise mean adjustment, was instead more focused on local and particular behaviours, on the more the ground-level perspective about Lombardy cities.
- **WE_tot_precipitation.** It is well known how rainwater can improve the air quality thanks to water droplets that attract aerosol particles, taking them away from the air, and bringing them to the ground. This process, known as wet deposition, precipitation, scavenging, or washout is indeed very effective in reducing the concentrations of air pollutants.
- **WE_blh_layer_max.** The boundary layer height (BLH) is another relevant variable in terms of the dispersion of air contaminants. This layer defines the height at which air mixing occurs. Usually air gets colder when rising in the atmosphere; however, there could be some warm air regions on top of colder ones. At this boundary, air is not more able to mix, with the warm layer causing a sort of lid, a covering, which traps the lower cold air, with all the contaminants, below it. This reduces the quality of the air since the

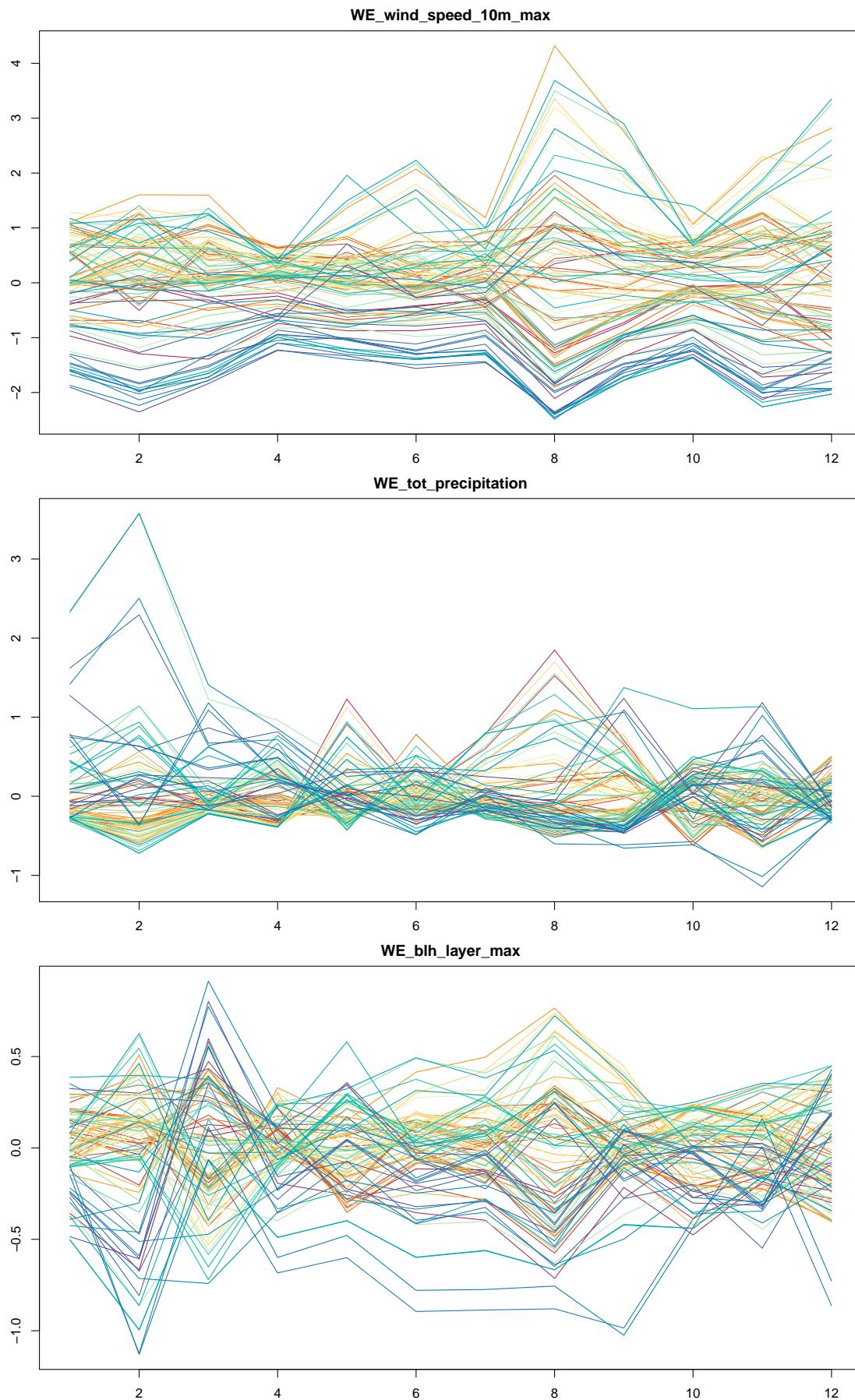


Figure 3.22: The three variables used to perform the test fit with covariates in the clustering. Colors are based on the ranking of PM₁₀ values of the units according to their median, from highest (red) to lowest (blue).

pollution builds up and accumulates since it has no way to disperse. This covariate, therefore, records the maximum height at which this problematic layer occurs: the lower the values, the higher we should expect the pollutants concentrations to be.

To perform this terminal fit with also covariates in the clustering we used the same parameters setup of the previous fits, meaning the same prior coefficients and MCMC configuration. For the covariates we employed similarity g_4 , the one with the Normal/Normal-Inverse-Gamma model, where we set $\mu_0 = 0$, $\lambda_0 = 1$, $a_0 = 7.5$ and $b_0 = 2$. These choices were motivated both by experimental tunings and by the standardization of the covariates, whose trend, as effectively illustrated by Figure 3.22, rarely moves outside the $[-1, 1]$ interval.

Table 3.6: Summary of the comparison between the two fits, on target plus space values, plus also the JDRPM fits including covariates in the clustering process. 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 4000 iterates) and the true values of the target variable. Higher LPML and lower WAIC indicate a better fit.

	MSE mean	MSE median	LPML	WAIC	exec. time
CDRPM	0.0142	0.0149	694.81	-1768.42	1h38m
JDRPM	0.0131	0.0138	624.91	-1898.05	48m
JDRPM + Xcl	0.0126	0.0135	677.71	-1969.76	1h20m

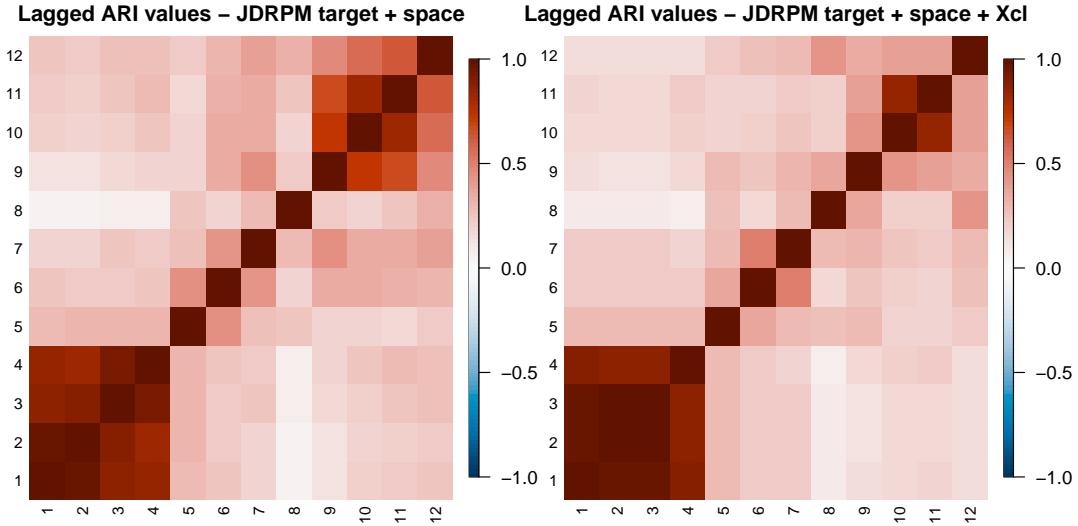


Figure 3.23: Comparison of the lagged ARI values of the JDRPM fits with covariates in the clustering and the standard fit on target plus space values.

The results are indeed encouraging, as we can see from Table 3.6 where all metrics appeared to have improved in this ultimate fit. Even with the inclusion of multiple covariates, i.e. with the risk of their “clustering suggestions” to differ, their contribution can still be appreciated by complementing the partition representation to cluster-specific boxplots of the relevant variables, i.e. the target PM_{10} plus and

three included covariates, in this case. Such a comparison is presented in Figure 3.24, where we can see how the JDRPM fit with also covariates seemed to have captured a slightly better clustering structure than the two competitors.

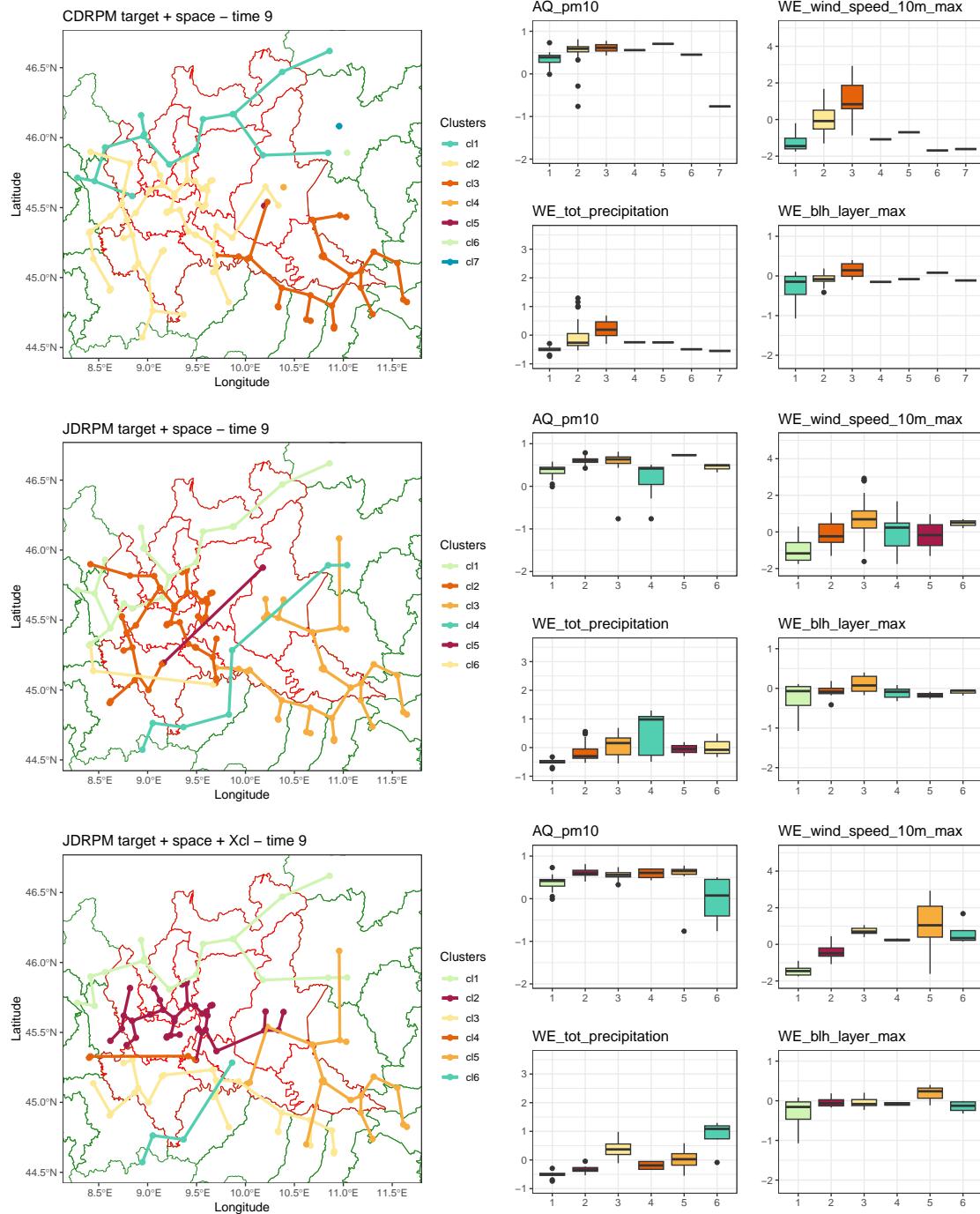


Figure 3.24: Comparison of the clusterings provided by the three main models under our analysis: CDRPM and JDRPM with target plus space values, and JDRPM plus with also covariates included in the clustering.

Regarding the highest polluted clusters, CDRPM did not manage to characterize better the stations in the right side of the map, while correctly identifying, as the other models did, the big cluster on the right side mostly spanning on Mantova

and Brescia areas. The middle-level polluted cluster, which encloses the mountain area was also accurately recovered, similarly to what the JDRPM plus covariates fit did. This cluster seems in fact to be characterized by the lowest level of `WE_wind_speed_10m_max`. In contrast, the corresponding covariate boxplot in the JDRPM standard fit seems to be more oriented towards higher values. This improvement of a more reasonable cluster could have been ascribed to the contribute of covariates into the clustering.

Regarding the lowest polluted cluster, CDRPM decided to separate it into a singleton, while both JDRPM fits decided to keep it into another cluster, therefore causing the appearance of an outlier in the boxplots. As this unit belongs to a mountain region, the inclusion of the `Altitude` covariate could have possibly helped to improve her cluster assignment. Nonetheless, this testing fit seems to provide a confirmation of the potential effectiveness and functionality of the insertion of covariates. The summary representation of the clusters of the JDRPM plus covariates fit is also available in Figure 3.25, and the temporal trend in Figure 3.23.

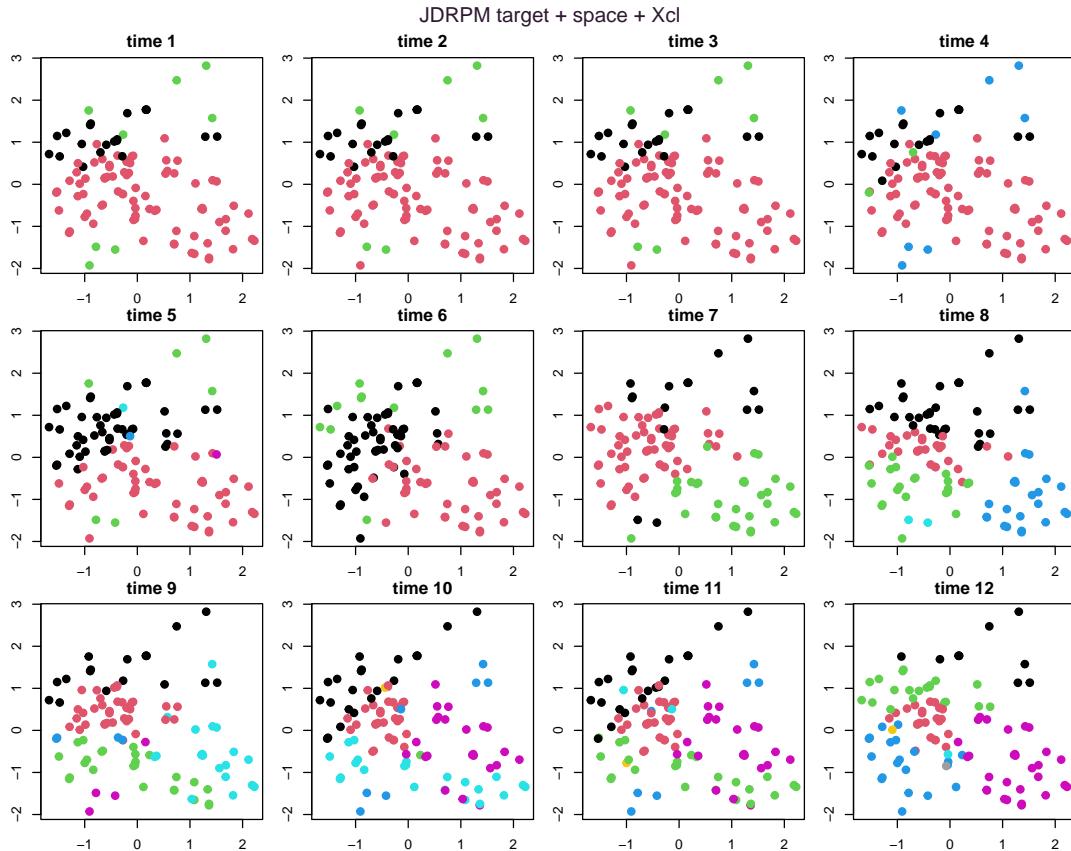


Figure 3.25: Clusters generated by the JDRPM fit, on target plus space values, with covariates in the clustering process.

The effect of the insertion of covariates is also proved by Figures 3.26 and 3.27, where we compare the distribution of the clusters with respect to one of the covariate, `WE_wind_speed_10m_max` in the case of the proposed figure. All the fits had of course the usual PM_{10} variable as target, meaning that the highest separation should appear there, but we could expect that, with the insertion of covariates, a

clustering pattern should also emerge slightly in the auxiliary covariates included. Indeed, the distribution of the clusters in the JDRPM fit with covariates shows a sharper division with respect to the wind speed variable; a division that appears more fuzzy in the CDRPM fit. Also with respect to the JDRPM without covariates of Figure 3.28 we can see a minor improvement, which could have been probably more significant in the case of datasets without an already-quite clear division in the target variable, since in any case the standard fits of target plus space values were already quite good, in terms of fitting metrics and clusters interpretation.

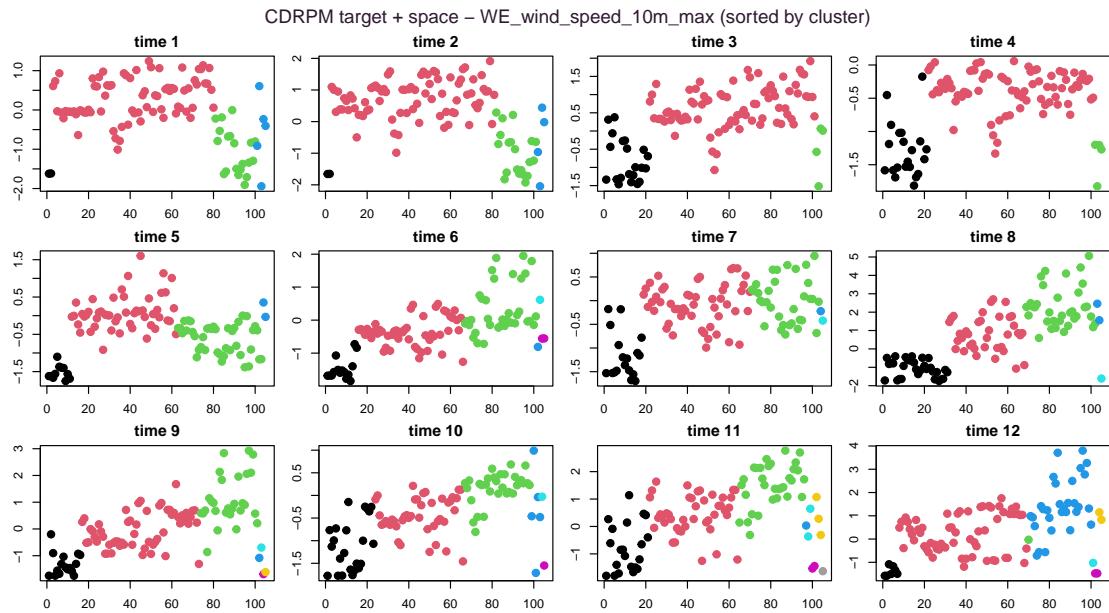


Figure 3.26: Distribution of the clusters according to the JDRPM standard fit, with respect to the wind speed covariate. The indication “sorted by clusters” indicates that the order of the units in the plot is based on their cluster membership and not on their original numbering $1, \dots, n$.

3.4 Scaling performances

To really assess if the goal of faster fits was reached, we setup several tests to compare the two models and the different information levels at 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 the inclusion of 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 in mind 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 those two plus covariates in the clustering, and finally all those plus covariates in the likelihood.

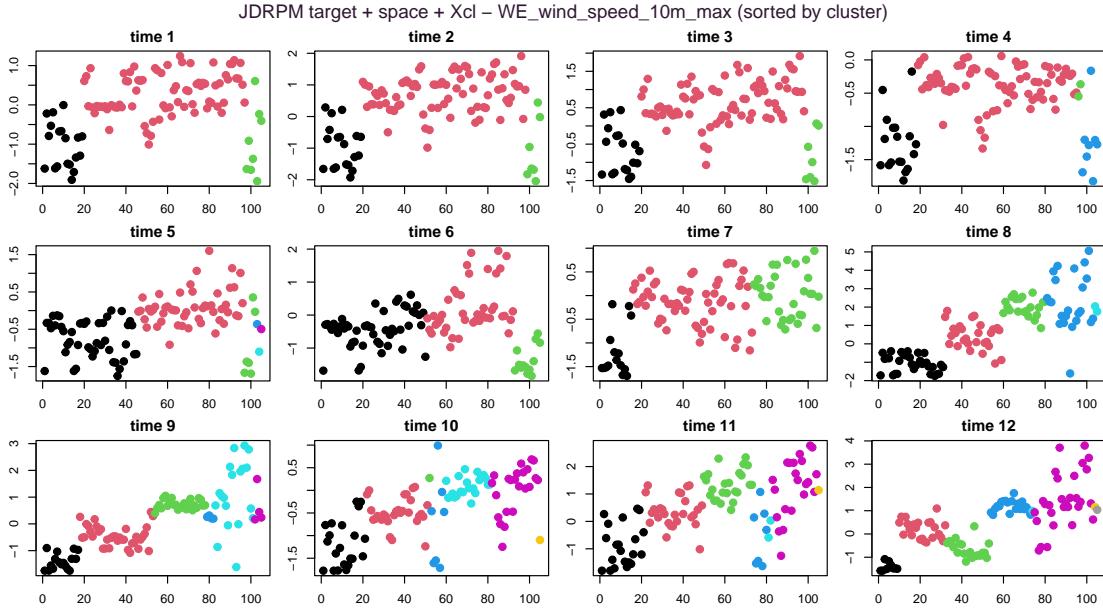


Figure 3.27: Distribution of the clusters according to the JDRPM fit with covariates, with respect to the wind speed covariate.

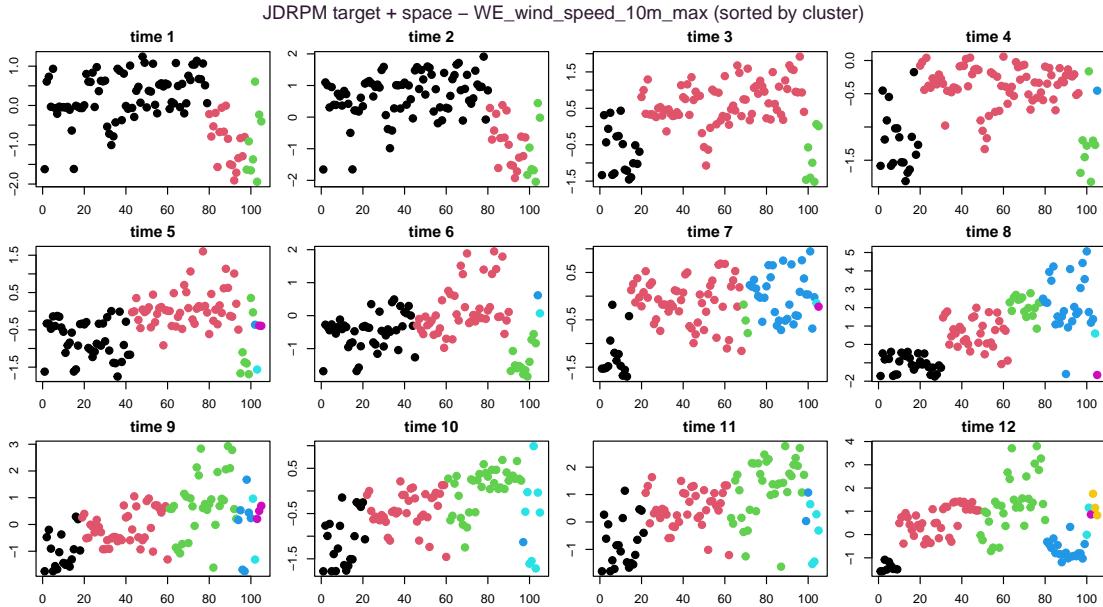


Figure 3.28: Distribution of the clusters according to the JDRPM standard fit, with respect to the wind speed covariate.

As it appeared from Section 2, the Julia implementation seemed to be a little more eager for memory than the C one. This meant that when we ran the larger tests my system could have ran out of RAM memory, resulting in some data needing to be stored in the slower swap space on the disk, and therefore losing performance. However, on newer or just more technical-oriented machines than my simple laptop, this memory limitation should not be an issue.

Accounting all this, in what follows we should consider really accurate just the intermediate-values tests, i.e. the ones with n or T being 10, 50 or 100, and

take with a pinch of salt the extreme one of 250, for which my laptop hardware limitations could have concealed the true accurate results. Nonetheless, the analysis is really encouraging for the JDRPM model.

The tests were conducted on randomly generated values, according to n and T , for the target data Y_{it} and the spatial coordinates s_i . In this way, we produced a sort of “mesh” of possible fitting parameters, ranging from small number of units and time horizons to larger ones. To record the expected execution time per iteration of each case we ran each fit with a number of iterations inversely proportional to the size of the test, e.g. 10000 iterations for the case $(n, T) = (10, 10)$ and 16 iterations for $(n, T) = (250, 250)$. This ensured that each fit lasted approximately the same time. Each fit was run four times, to then save the minimum time observed during those multiple runs. Such practice, common in benchmarking, helps to eliminate bias from the results due to the system computational demands and fluctuations, filtering in this way just the “ideal” execution time in which all the system performance are devoted to that task.

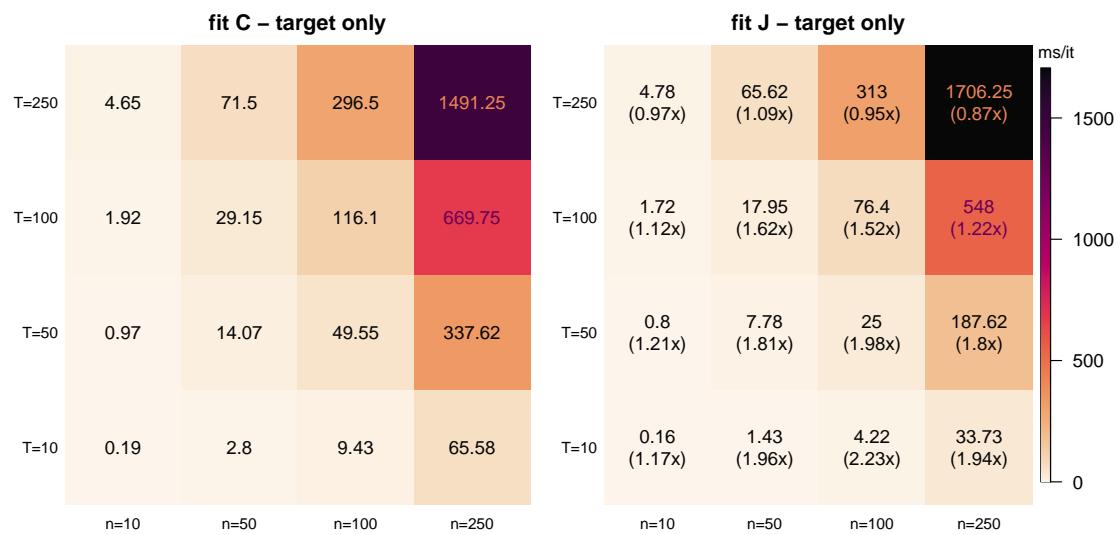


Figure 3.29: Execution times, measured in milliseconds per iteration, when fitting the models using just the target values to generate the clusters. On the JDRPM plot (on the right), in brackets, are reported the speedups relative to the CDRPM timings (on the left), where higher values indicate better performance.

From Figure 3.29 we can see how the basic fit, with just the target values, perform very quickly in Julia, with peaks even around a 2x speedup. Similar improvements appeared in the case of fits including also the spatial information, as Figure 3.30 shows. Therefore, especially looking at the more reliable non-extreme tests, we can confidently claim that the JDRPM implementation is faster than the CDRPM one.

Regarding fits with covariates, we also generated them randomly by creating matrices of dimensions $n \times p \times T$. For these tests, we avoided to treat different values of p : the previous figures summarizing the fits up to spatial information were in fact projection of 3d information (n , T , and execution time), while if we extended the mesh to also account for p , the number of covariates, we would have

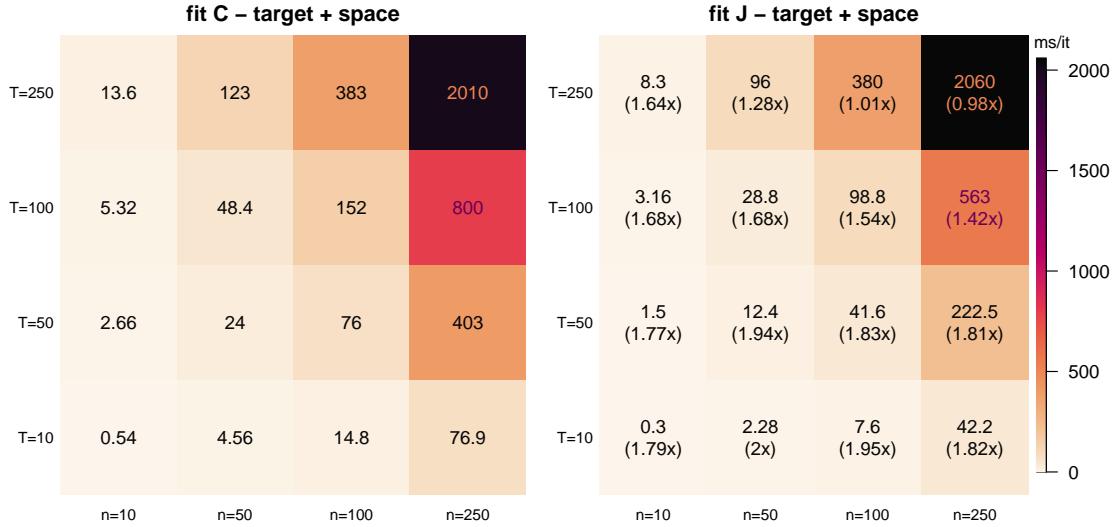


Figure 3.30: Execution times, measured in milliseconds per iteration, when fitting the models using the target values plus the spatial information to generate the clusters. On the JDRPM plot (on the right), in brackets, are reported the speedups relative to the CDRPM timings (on the left), where higher values indicate better performance.

ended up with 4d information, or even 5d if we considered separately clustering and likelihood covariates. Therefore, to keep things tidy and understandable, we just fixed $p = 5$ for both cases.

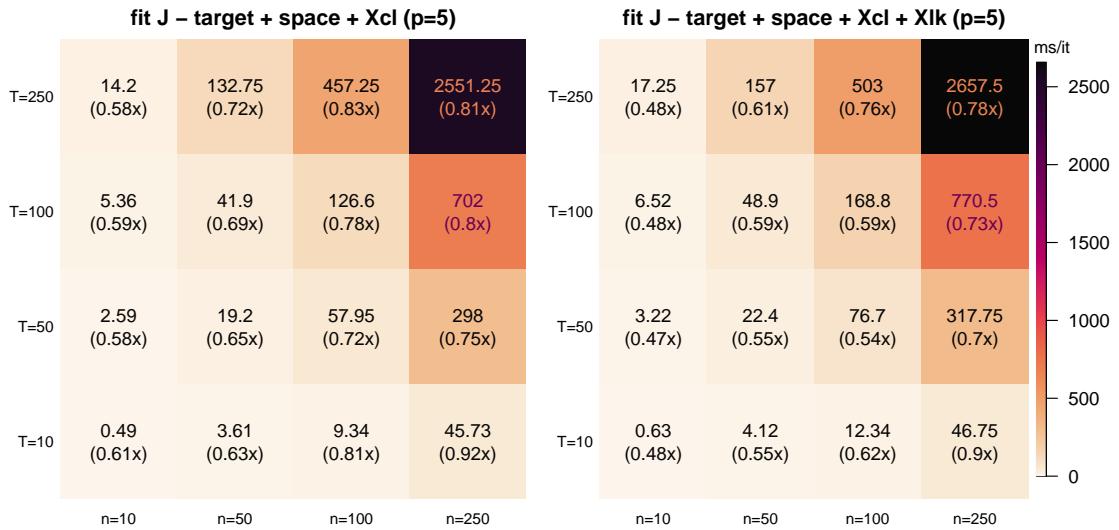


Figure 3.31: Execution times, measured in milliseconds per iteration, when fitting the JDRPM model using target values, spatial information, covariates in the clustering level (left), and covariates in both clustering and likelihood levels (right). In brackets are reported the speedups relative to the JDRPM timings of the fitting with target and space, with higher values still indicating better performance.

Figure 3.31 shows how fits including covariates saw a drop of performance, which was inevitable due to the additional work to perform, but they still remain at a good performance level. Indeed, Figure 3.32 will point out how some of the fits

with all information levels in Julia were actually faster than the basic fits in C.

We can also notice how with larger fits the performance drop reduces to just around 0.8x, rather than the 0.6x of the small cases: this indicates that on fits with heavy n and T the bottleneck is no more the algorithm complexity, meaning the choice of which information levels to include in the clusters generation, but rather the available hardware resources.

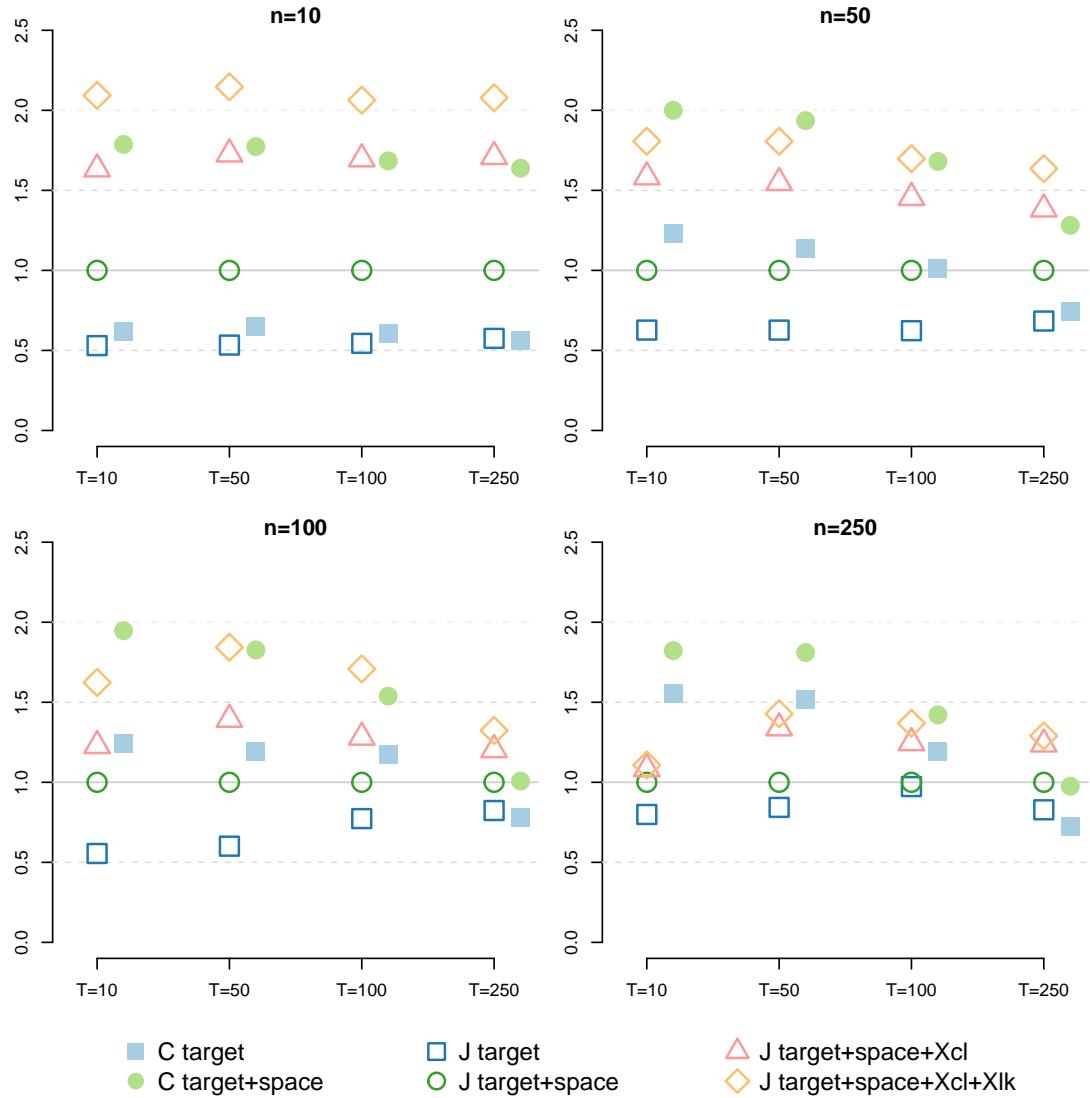


Figure 3.32: Visual representation of the performance of all the fits, for all the n and T cases, relatively to the JDRPM fit using target values and spatial information. Namely, the execution time per iteration metric of that fit has been taken as a reference, to which then all other fits have been compared to derive their speedup (or slowdown) factor. Points above the reference line indicate slower fits, while points below denote faster fits.

Chapter 4

Conclusion

*And what in human reckoning seems still afar off,
may by the Divine ordinance be close at hand, on the
eve of its appearance. And so be it, so be it!*

— Fëdor Dostoevskij, *Brothers Karamazov*

In the end, we can confidently assert that the JDRPM model is a valid improvement of the original DRPM formulation.

From a theoretical point of view, it offers the same base structure, just with the addition of a possible regression term in the likelihood, while possibly enhancing the quality of the sampled values of the parameters through the change in the law of the variances from uniform to inverse gamma. In fact, this choice recovers conjugacy in the model and therefore improves the mixing of the chain during the fitting of the model. Plus, of course, the insertion of covariates in the clustering should provide even more accurate results in the generation of the partitions with respect to only having the spatio-temporal information layers.

A possible drawback, inevitable with the increased model complexity, could be the tuning of the parameters, especially in the cohesion and similarity functions to find the right balance between the different contributes of the information levels. However, in this regards, an hopefully helpful analysis was conducted in Chapter 1, where the effects of the tuning parameters for all cohesions and similarities have been explored. Moreover, the Julia `MCMC_fit` function provides an optional argument `cv_weight`, defaulted to 1, which can be used to scale up (or down) the contribute of the covariates similarities, e.g. to make them closer to the interval to which the spatial cohesions values belong.

This higher complexity appeared also in the tuning of the `invGamma(a, b)` parameters, which are indeed more delicate than a simple $\mathcal{U}(l, u)$. To this end, a possible solution could be to run an initial fit with the original CDRPM model, to see where the values of the variances tend to settle, and according to that fixing the `invGamma` parameters. For example, in the spatio-temporal tests of Section 3.1.2 we saw, from the CDRPM fits, very low values for the variances parameters and as such we assigned, for λ^2 and τ_t^2 parameters in the JDRPM model, an `invGamma(a = 1.9, b = 0.4)`

which has 90% of his density in the interval [0.109151, 1.58222]. The more relevant $\sigma_{jt}^{2\star}$ parameter had also pretty low values, however on that we attached the more uninformative prior `invGamma(a = 0.01, b = 0.01)` which despite being not always suggested [Gel04] turned out to be fine and very precise, relatively to the sampled “reference” values of CDRPM. In general, an alternative solution, probably less theoretically appealing but possibly effective, could be to truncate the `invGamma` laws to a certain region in order to avoid sampling extremely high values, which could happen if uninformative priors get not properly adjusted by the data, and also to resemble more the density of a uniform random variable. This choice can be seamlessly implemented in the Julia code by changing `rand(InverseGamma(a, b))` to `rand(truncated(InverseGamma(a, b), l, u))`, with l and u being the lower and upper bounds of the designed truncation interval.

From a computational point of view, the goal of faster fits was also reached, reducing the execution times up to a factor of two with respect to the original implementation. This at the cost of a small increase in the memory requirements which however, nowadays, should not be a big deal.

A final drawback from the two-tiered implementation is time needed to convert back the results from Julia to R, since the `juliaGet` function from the `JuliaConnectoR` package can take up to some minutes, especially in fits where lots of iterations are saved. In any case, this latency becomes quite negligible considering the time saved with respect to the original implementation of the DRPM package.

Regarding some possible further improvements, from the theoretical point of view we can also consider satisfied. The complexity of the original DRPM model was already kind of daunting, and the improvements brought by the JDRPM update surely haven’t reduced it, considering all the parameters present in the model and especially the ones regulating the spatial cohesions and covariate similarities which strongly determine the final clusters. Therefore, with this complexity in mind, some room for upgrades is easily available in improving the usability of this model. To this end, the current JDRPM implementation has already some basic logging features, where e.g. some steps of the computations can be saved to a text file and analysed afterwards, but more elaborate profiling and monitoring tools could also be implemented, particularly considering the flexible possibilities offered by Julia. Some ideas could be heuristics or suggestions about how to initialize the parameters according to the dataset at hand, or automatic online tuning of the parameters with the progression of iterations. Additionally, visualization tools could be also comfortably implemented trough the extensive plotting ecosystem of Julia, e.g. to return trace plots directly with the execution, provide live-monitoring the distribution of the sampled parameters, and so on.

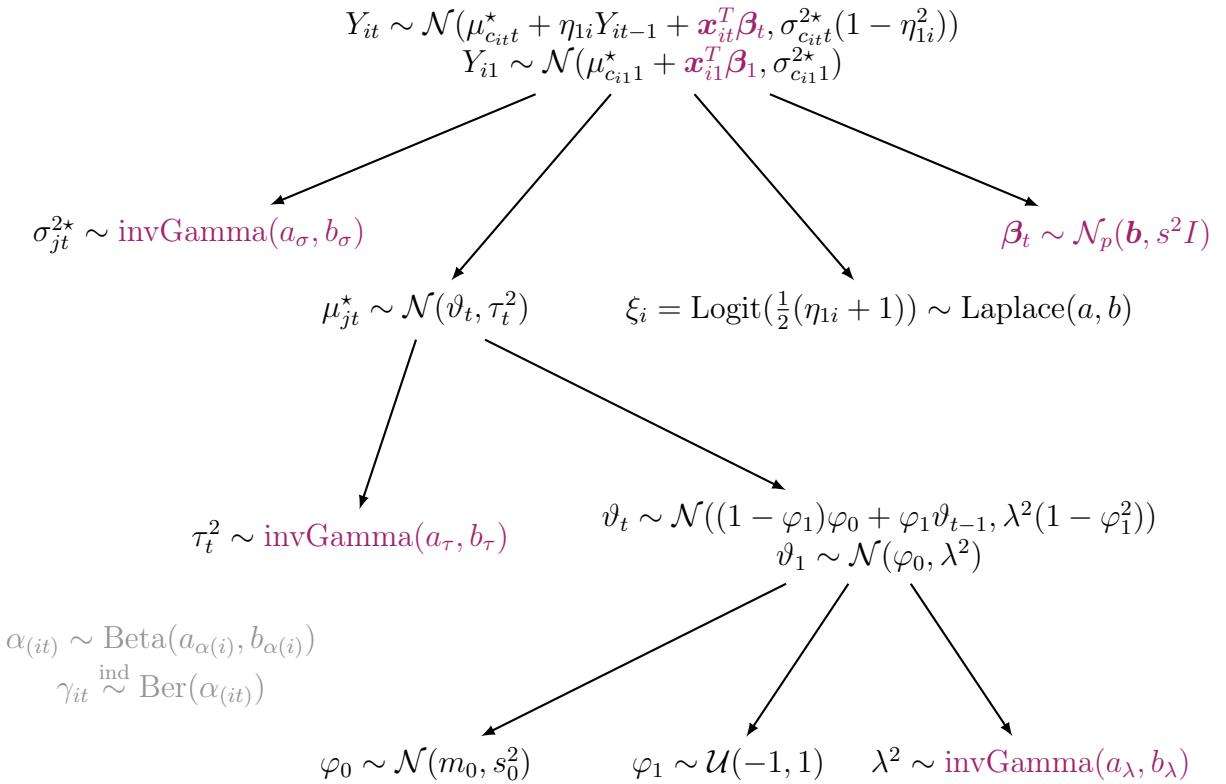
Moreover, considering the advancement in performance, a setup with multiple chains ran in parallel, as most of the lighter Bayesian models do, could be introduced. Possible further speedup could also be obtained integrating the GPU, if available, to reduce some load from the CPU. Needless to say, the road could already be paved to perform such integration trough the various packages under the `JuliaGPU` collection.

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 1. 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 $t = 1$:

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

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

for $t > 1$:

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

- update μ_{jt}^*

for $t = 1$:

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

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

for $t > 1$:

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

- update $\boldsymbol{\beta}_t$

for $t = 1$:

$$\begin{aligned}
f(\boldsymbol{\beta}_t | -) &\propto f(\boldsymbol{\beta}_t) f(\{Y_{1t}, \dots, Y_{nt}\} | \boldsymbol{\beta}_t, -) \\
&= \mathcal{L}_{\mathcal{N}(\mathbf{b}, s^2 I)}(\boldsymbol{\beta}_t) \prod_{i=1}^n \mathcal{L}_{\mathcal{N}(\mu_{c_{it}t}^* + \mathbf{x}_{it}^T \boldsymbol{\beta}_t, \sigma_{c_{it}t}^{2*})}(Y_{it})
\end{aligned}$$

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

for $t > 1$:

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

$\implies f(\boldsymbol{\beta}_t | -) \propto \text{kernel of a } \mathcal{N}(\mathbf{b}_{(\text{post})}, A_{(\text{post})}) \text{ with}$

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

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

$$\begin{aligned} \mathbf{h}_{(\text{post})} &= \left(\frac{\mathbf{b}}{s^2} + \sum_{i=1}^n \frac{(Y_{it} - \mu_{c_{it}t}^* - \eta_{1i} Y_{it-1}) \mathbf{x}_{it}}{\sigma_{c_{it}t}^{2*}} \right) \\ J_{(\text{post})} &= \left(\frac{1}{s^2} I + \sum_{i=1}^n \frac{\mathbf{x}_{it} \mathbf{x}_{it}^T}{\sigma_{c_{it}t}^{2*}} \right) \end{aligned} \quad (\text{A.6})$$

- update τ_t^2

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

- update ϑ_t

for $t = T$:

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

for $1 < t < T$:

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

for $t = 1$:

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

- update φ_0

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

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

- update λ^2

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

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

$$\begin{aligned}
a_{\lambda(\text{post})} &= \frac{T}{2} + a_\lambda \\
b_{\lambda(\text{post})} &= \frac{(\vartheta_1 - \varphi_0)^2}{2} + \sum_{t=2}^T \frac{(\vartheta_t - (1-\varphi_1)\varphi_0 - \varphi_1 \vartheta_{t-1})^2}{2} + b_\lambda \tag{A.12}
\end{aligned}$$

- update α

if global α : prior is $\alpha \sim \text{Beta}(a_\alpha, b_\alpha)$

$$\begin{aligned}
f(\alpha | -) & \propto f(\alpha) f((\gamma_{11}, \dots, \gamma_{1T}, \dots, \gamma_{n1}, \dots, \gamma_{nT}) | \alpha) \\
& \propto \alpha^{a_\alpha-1} (1-\alpha)^{b_\alpha-1} \prod_{i=1}^n \prod_{t=1}^T \alpha^{\gamma_{it}} (1-\alpha)^{1-\gamma_{it}} \\
& = \alpha^{(a_\alpha + \sum_{i=1}^n \sum_{t=1}^T \gamma_{it}) - 1} (1-\alpha)^{(b_\alpha + nT - \sum_{i=1}^n \sum_{t=1}^T \gamma_{it}) - 1} \\
\implies f(\alpha | -) & \propto \text{kernel of a } \text{Beta}(a_{\alpha(\text{post})}, b_{\alpha(\text{post})}) \text{ with}
\end{aligned}$$

$$a_{\alpha(\text{post})} = a_\alpha + \sum_{i=1}^n \sum_{t=1}^T \gamma_{it} \quad b_{\alpha(\text{post})} = b_\alpha + nT - \sum_{i=1}^n \sum_{t=1}^T \gamma_{it} \quad (\text{A.13})$$

if time specific α : prior is $\alpha_t \sim \text{Beta}(a_\alpha, b_\alpha)$

$$\begin{aligned} f(\alpha_t | -) &\propto f(\alpha_t) f((\gamma_{1t}, \dots, \gamma_{nt}) | \alpha_t) \\ &\propto \alpha_t^{a_\alpha-1} (1-\alpha_t)^{b_\alpha-1} \prod_{i=1}^n \alpha_t^{\gamma_{it}} (1-\alpha_t)^{1-\gamma_{it}} \\ &= \alpha_t^{(a_\alpha + \sum_{i=1}^n \gamma_{it})-1} (1-\alpha_t)^{(b_\alpha + n - \sum_{i=1}^n \gamma_{it})-1} \end{aligned}$$

$\implies f(\alpha_t | -) \propto \text{kernel of a Beta}(a_{\alpha(\text{post})}, b_{\alpha(\text{post})})$ with

$$a_{\alpha(\text{post})} = a_\alpha + \sum_{i=1}^n \gamma_{it} \quad b_{\alpha(\text{post})} = b_\alpha + n - \sum_{i=1}^n \gamma_{it} \quad (\text{A.14})$$

if unit specific α : prior is $\alpha_i \sim \text{Beta}(a_{\alpha i}, b_{\alpha i})$

$$\begin{aligned} f(\alpha_i | -) &\propto f(\alpha_i) f((\gamma_{i1}, \dots, \gamma_{iT}) | \alpha_i) \\ &\propto \alpha_i^{a_{\alpha i}-1} (1-\alpha_i)^{b_{\alpha i}-1} \prod_{t=1}^T \alpha_i^{\gamma_{it}} (1-\alpha_i)^{1-\gamma_{it}} \\ &= \alpha_i^{(a_{\alpha i} + \sum_{t=1}^T \gamma_{it})-1} (1-\alpha_i)^{(b_{\alpha i} + T - \sum_{t=1}^T \gamma_{it})-1} \end{aligned}$$

$\implies f(\alpha_i | -) \propto \text{kernel of a Beta}(a_{\alpha(\text{post})}, b_{\alpha(\text{post})})$ with

$$a_{\alpha(\text{post})} = a_{\alpha i} + \sum_{t=1}^T \gamma_{it} \quad b_{\alpha(\text{post})} = b_{\alpha i} + T - \sum_{t=1}^T \gamma_{it} \quad (\text{A.15})$$

if time and unit specific α : prior is $\alpha_{it} \sim \text{Beta}(a_{\alpha i}, b_{\alpha i})$

$$\begin{aligned} f(\alpha_{it} | -) &\propto f(\alpha_{it}) f(\gamma_{it} | \alpha_{it}) \\ &\propto \alpha_{it}^{a-1} (1-\alpha_{it})^{b-1} \alpha_{it}^{\gamma_{it}} (1-\alpha_{it})^{1-\gamma_{it}} \\ &= \alpha_i^{(a+\gamma_{it})-1} (1-\alpha_i)^{(b+1-\gamma_{it})-1} \end{aligned}$$

$\implies f(\alpha_{it} | -) \propto \text{kernel of a Beta}(a_{\alpha(\text{post})}, b_{\alpha(\text{post})})$ with

$$a_{\alpha(\text{post})} = a_{\alpha i} + \gamma_{it} \quad b_{\alpha(\text{post})} = b_{\alpha i} + 1 - \gamma_{it} \quad (\text{A.16})$$

- update a missing observation Y_{it}

for $t = 1$:

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

$$\begin{aligned}
&= \exp \left\{ -\frac{1}{2\sigma_{c_{it}t}^{2\star}} (Y_{it} - (\mu_{c_{it}t}^* + \mathbf{x}_{it}^T \boldsymbol{\beta}_t))^2 \right\} \\
&\cdot \exp \left\{ -\frac{\eta_{1i}^2}{2\sigma_{c_{it+1}t+1}^{2\star}(1-\eta_{1i}^2)} \left(Y_{it} - \frac{Y_{it+1} - \mu_{c_{it+1}t+1}^* - \mathbf{x}_{it+1}^T \boldsymbol{\beta}_{t+1}}{\eta_{1i}} \right)^2 \right\} \\
\implies f(Y_{it} | -) &\propto \text{kernel of a } \mathcal{N}(\mu_{Y_{it}(\text{post})}, \sigma_{Y_{it}(\text{post})}^2) \text{ with} \\
\sigma_{Y_{it}(\text{post})}^2 &= \frac{1}{\frac{1}{\sigma_{c_{it}t}^{2\star}} + \frac{\eta_{1i}^2}{2\sigma_{c_{it+1}t+1}^{2\star}(1-\eta_{1i}^2)}} \\
\mu_{Y_{it}(\text{post})} &= \sigma_{Y_{it}(\text{post})}^2 \left(\frac{\mu_{c_{it}t}^* + \mathbf{x}_{it}^T \boldsymbol{\beta}_t}{\sigma_{c_{it}t}^{2\star}} + \frac{\eta_{1i}(Y_{it+1} - \mu_{c_{it+1}t+1}^* - \mathbf{x}_{it+1}^T \boldsymbol{\beta}_{t+1})}{\sigma_{c_{it+1}t+1}^{2\star}(1-\eta_{1i}^2)} \right) \tag{A.17}
\end{aligned}$$

for $1 < t < T$:

$$\begin{aligned}
f(Y_{it} | -) &\propto f(Y_{it}) f(Y_{it+1} | Y_{it}, -) \\
&= \mathcal{L}_{\mathcal{N}(\mu_{c_{it}t}^* + \eta_{1i} Y_{it-1} + \mathbf{x}_{it}^T \boldsymbol{\beta}_t, \sigma_{c_{it}t}^{2\star})}(Y_{it}) \\
&\cdot \mathcal{L}_{\mathcal{N}(\mu_{c_{it+1}t+1}^* + \eta_{1i} Y_{it} + \mathbf{x}_{it+1}^T \boldsymbol{\beta}_{t+1}, \sigma_{c_{it+1}t+1}^{2\star}(1-\eta_{1i}^2))}(Y_{it+1}) \\
&\propto \exp \left\{ -\frac{1}{2\sigma_{c_{it}t}^{2\star}(1-\eta_{1i}^2)} (Y_{it} - \mu_{c_{it}t}^* - \eta_{1i} Y_{it-1} - \mathbf{x}_{it}^T \boldsymbol{\beta}_t)^2 \right\} \\
&\cdot \exp \left\{ -\frac{1}{2\sigma_{c_{it+1}t+1}^{2\star}(1-\eta_{1i}^2)} (Y_{it+1} - \mu_{c_{it+1}t+1}^* - \eta_{1i} Y_{it} - \mathbf{x}_{it+1}^T \boldsymbol{\beta}_{t+1})^2 \right\} \\
&= \exp \left\{ -\frac{1}{2\sigma_{c_{it}t}^{2\star}(1-\eta_{1i}^2)} (Y_{it} - (\mu_{c_{it}t}^* + \eta_{1i} Y_{it-1} + \mathbf{x}_{it}^T \boldsymbol{\beta}_t))^2 \right\} \\
&\cdot \exp \left\{ -\frac{\eta_{1i}^2}{2\sigma_{c_{it+1}t+1}^{2\star}(1-\eta_{1i}^2)} \left(Y_{it} - \frac{Y_{it+1} - \mu_{c_{it+1}t+1}^* - \mathbf{x}_{it+1}^T \boldsymbol{\beta}_{t+1}}{\eta_{1i}} \right)^2 \right\} \\
\implies f(Y_{it} | -) &\propto \text{kernel of a } \mathcal{N}(\mu_{Y_{it}(\text{post})}, \sigma_{Y_{it}(\text{post})}^2) \text{ with} \\
\sigma_{Y_{it}(\text{post})}^2 &= \frac{1 - \eta_{1i}^2}{\frac{1}{\sigma_{c_{it}t}^{2\star}} + \frac{\eta_{1i}^2}{\sigma_{c_{it+1}t+1}^{2\star}}} \\
\mu_{Y_{it}(\text{post})} &= \sigma_{Y_{it}(\text{post})}^2 \left(\frac{\mu_{c_{it}t}^* + \eta_{1i} Y_{it-1} + \mathbf{x}_{it}^T \boldsymbol{\beta}_t}{\sigma_{c_{it}t}^{2\star}(1-\eta_{1i}^2)} + \frac{\eta_{1i}(Y_{it+1} - \mu_{c_{it+1}t+1}^* - \mathbf{x}_{it+1}^T \boldsymbol{\beta}_{t+1})}{\sigma_{c_{it+1}t+1}^{2\star}(1-\eta_{1i}^2)} \right) \tag{A.18}
\end{aligned}$$

for $t = T$:

$$\begin{aligned}
f(Y_{it} | -) &\propto f(Y_{it}) \\
&= \mathcal{L}_{\mathcal{N}(\mu_{c_{it}t}^* + \eta_{1i} Y_{it-1} + \mathbf{x}_{it}^T \boldsymbol{\beta}_t, \sigma_{c_{it}t}^{2\star}(1-\eta_{1i}^2))}(Y_{it}) \\
\implies f(Y_{it} | -) &\text{ is just the likelihood of } Y_{it} \tag{A.19}
\end{aligned}$$

Appendix B

Computational details

B.1 Fitting algorithm code

We report here part of the 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. 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.

Together with what said in Chapter 2, we also note that the productivity offered by the Julia language is not only obtained trough the vast land of packages and documentation, but even trough an online active forum¹, where I wrote myself some questions (and received answers) during the development of the thesis. Finally, at <https://github.com/federicomor/Tesi/tree/main/src/JDRPM> there are available all the codes behind this JDRPM update.

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

```
##### start MCMC algorithm #####
println(replace(string(now()), "T" => " ") [1:end-4])
println("Starting MCMC algorithm")

t_start = now()
progresso = Progress(round(Int64(draws)),
    showspeed=true,
    output=stdout, # default is stderr, which turns out in orange color on R
    dt=1, # every how many seconds update the feedback
    barlen=0 # no progress bar
)

@inbounds for i in 1:draws
    ##### 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 for the data which are missing (from
    # their full conditional)
```

¹<https://discourse.julialang.org/>

```

if Y_has_NA
    # we have to use the missing_idxs 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_idxs
        # Problem: 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
        # We could have decided to ignore this computation and just use the
        # likelihood as proposal
        # filling distribution, but this would have just worked in the Y[j,t+1]
        # case so the general
        # problem would have still been there

        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

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

                # filter the indexes of the units of label k
                aux_idxs = findall(Si_red .== k)
                lc .= 0.
                if sPPM
                    copy!(s1o, sp1_red[aux_idxs])
                    copy!(s2o, sp2_red[aux_idxs])

```

```

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,1C)
spatial_cohesion!(spatial_cohesion_idx, s1n, s2n,
→ sp_params_struct, true, M_dp, S,2,false,1C)
end

lS .= 0.
if cl_xPPM
    for p in 1:p_cl
        if isa(first(Xcl_covariates[j,p,t]),Real) # numerical
        → covariate
            copy!(Xo, @view Xcl_covariates_red[aux_idxs,p])
            copy!(Xn, Xo); push!(Xn,Xcl_covariates[j,p,t])

            if covariate_similarity_idx == 4
                covariate_similarity!(covariate_similarity_idx, Xo,
                → cv_params_sim4[p], Rs[p,t],
                → true,1,true,lS,cv_weight)
                covariate_similarity!(covariate_similarity_idx, Xn,
                → cv_params_sim4[p], Rs[p,t],
                → true,2,true,lS,cv_weight)
            else
                covariate_similarity!(covariate_similarity_idx, Xo,
                → cv_params, Rs[p,t], true,1,true,lS,cv_weight)
                covariate_similarity!(covariate_similarity_idx, Xn,
                → cv_params, Rs[p,t], true,2,true,lS,cv_weight)
            end
        else # categorical covariate
            copy!(Xo_cat, @view Xcl_covariates_red[aux_idxs,p])
            copy!(Xn_cat, Xo_cat);
            → push!(Xn_cat,Xcl_covariates[j,p,t])

            covariate_similarity!(covariate_similarity_idx, Xo_cat,
            → cv_params, Rs[p,t], true,1,true,lS,cv_weight)
            covariate_similarity!(covariate_similarity_idx, Xn_cat,
            → cv_params, Rs[p,t], true,2,true,lS,cv_weight)
        end
    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,1C)
end
lS .= 0.
if cl_xPPM
    for p in 1:p_cl
        if covariate_similarity_idx == 4
            covariate_similarity!(covariate_similarity_idx,
            → SVector(Xcl_covariates[j,p,t]), cv_params_sim4[p],
            → Rs[p,t], true, 2,true,lS,cv_weight)
        else
            covariate_similarity!(covariate_similarity_idx,
            → SVector(Xcl_covariates[j,p,t]), cv_params, Rs[p,t],
            → true, 2,true,lS,cv_weight)
        end
    end
end

```

```

        end
    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^{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) # fast

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)
    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 (numerosities 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) # fast
    # 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)
    end
end

lPP .= 0.

```

```

        for kk in 1:nclus_temp
            aux_idxs = findall(rho_tmp .== kk)
            if sPPM
                copy!(s1n, @view sp1[aux_idxs])
                copy!(s2n, @view sp2[aux_idxs])
                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_idxs,p,t]
                    if covariate_similarity_idx == 4
                        covariate_similarity!(covariate_similarity_idx,
                            → Xn_view, cv_params_sim4[p], Rs[p,t],
                            → true, 1, true, lPP, cv_weight)
                    else
                        covariate_similarity!(covariate_similarity_idx,
                            → Xn_view, cv_params, Rs[p,t],
                            → true, 1, true, lPP, cv_weight)
                    end
                end
            end
            1PP[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]) + 1PP[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]) + 1PP[1]
        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_idxs = findall(rho_tmp .== kk) # fast
    if sPPM
        copy!(s1n, @view sp1[aux_idxs])
        copy!(s2n, @view sp2[aux_idxs])
        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_idxs,p,t]
            if covariate_similarity_idx == 4
                covariate_similarity!(covariate_similarity_idx, Xn_view,
                                      → cv_params_sim4[p], Rs[p,t],
                                      → true, 1, true, lPP, cv_weight)
            else
                covariate_similarity!(covariate_similarity_idx, Xn_view,
                                      → cv_params, Rs[p,t], true, 1, true, lPP, cv_weight)
            end
        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

# 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 sampling) (after sampling)
#      units j 2 3 4 5 -> units j 2 3 4 5
#      labels 1 1 1 2 2      labels 3 1 1 2 2
# the after case has to be relabelled
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
# discard the zeros at the end of the auxiliary vectors nh_reorde and
# old_lab
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
        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

```

```

end

##### update beta #####
if lk_xPPM && i>=beta_update_threshold
    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))
        # this is the quicker and more accurate method

        # old method with the MuNormal and the inversion required
        # Am1_star = inv(A_star)
        # beta_iter[t] = rand(MuNormal(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))
    end
end

##### update theta #####
aux1::Float64 = 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)))
else
    if 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
        sum_mu = 0.0
        for k in 1:kt
            sum_mu += muh_iter[k,t]
        end
        sig2_post = 1 / (lambda2_iter*(1-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)))
    end
end

```

```

    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_tau = tau2_priors[1] + kt/2
b_star_tau = tau2_priors[2] + aux1/2
tau2_iter[t] = rand(InverseGamma(a_star_tau, b_star_tau))

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])
                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

#####
# 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 phi1_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 = lambda2_priors[1] + T/2
b_star_lambda2 = lambda2_priors[2] + ((theta_iter[1] - phi0_iter)^2 + aux1) /
→ 2
lambda2_iter = rand(InverseGamma(a_star_lambda2,b_star_lambda2))

##### 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,:,:), 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

i_out += 1
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

```

```

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()

if perform_diagnostics
    chn = Chains(
        hcat(lambda2_out,phi0_out,tau2_out',theta_out',eta1_out',alpha_out'),
        ["lambda2","phi0",
        [string("tau2_t", i) for i in 1:T]...,
        [string("theta_t", i) for i in 1:T]...,
        [string("eta1_j", i) for i in 1:n]...,
        [string("alpha_t", i) for i in 1:T]...,
        ]
    )
    ss = DataFrame(summarystats(chn))
    println("\nDiagnostics:")
    @show ss[!, [1,4,5,6,7]];
    if logging CSV.write(log_file,ss[!, [1,4,5,6,7]]) end
end

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` 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 on how 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))

# trigger the compilation of the function, otherwise all first runs
# will be slow since the function still would have to be compiled
module_JDRPM$trigger_compilation()

##### Fit #####
# perform the fit
out = module_JDRPM$MCMC_fit(...)

# 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 form
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)
```

```
# this reshape works in the case of full model fit, but in the case of special
# fitting options (e.g. unit_specific_alpha=true) it needs to be adjusted
```

The `trigger_compilation` function runs the fitting function on a small test case to simply give Julia a friendly nudge to compile the function, making all the subsequent and more serious fits faster. In fact, especially for more complex projects, there are packages such as `PrecompileTools` and `PackageCompiler` which helps to solve this “time to first execution” problem of Julia.

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 implementation. The most relevant add-ons are the possibility to perform diagnostics on a subset of the sampled parameters, and the real-time progress monitoring, which updates itself every second to show the estimated remaining time to complete the fit. As a further proof of the ease of this language, these features were just a simple insertion of few lines of code, thanks to the Julia packages `MCMCChains` [GXG18] and `ProgressMeter`.

Listing 5: Feedback from the JDRPM implementation.

```
# if verbose=true this initial parameter section is also printed
Parameters:
sig2h ~ invGamma(0.01, 0.01)
Logit(1/2(eta1+1)) ~ Laplace(0, 0.9)
tau2 ~ invGamma(1.9, 0.4)
phi0 ~ Normal(mu=0.0, sigma=10.0)
lambda2 ~ invGamma(1.9, 0.4)
alpha ~ Beta(2.0, 2.0)

- using seed 113.0 -
fitting 110000 total iterates (with burnin=90000, thinning=5)
thus producing 4000 valid iterates in the end

on n=105 subjects
for T=12 time instants

[✓] with space? true (cohesion 3.0)
[X] with covariates in the likelihood? false
[X] with covariates in the clustering process? false
[X] are there missing data in Y? false

2024-10-14 09:33:11
Starting MCMC algorithm
Progress: 100% Time: 0:25:39 (15.40 ms/it)

done!
Elapsed time: 48 minutes, 45 seconds, 862 milliseconds
LPML: 624.9135200152505 (the higher the better)
WAIC: -1898.049549751033 (the lower the better)
acceptance ratio eta1: 51.72%
acceptance ratio phi1: 36.99%

# if perform_diagnostics=true this final diagnostics section is also printed
Diagnostics:
```

```
ss[!, [1, 4, 5, 6, 7]] = 143×5 DataFrame
  Row | parameters      mcse      ess_bulk      ess_tail      rhat
       | Symbol          Float64    Float64    Float64    Float64
-----|-----
  1 | lambda2        0.000817173  4243.04   3970.25   0.999769
  2 | phi0           0.00218706   3533.95   3675.99   1.00018
  3 | tau2_t1        0.00540064   3716.56   3645.99   0.999968
       |
       :
 14 | tau2_t12       0.00259168   3965.77   3655.55   0.999951
 15 | theta_t1        0.00404988   3603.34   3765.78   0.999834
       |
       :
 26 | theta_t12       0.00337843   3588.83   3907.13   0.999866
 27 | eta1_j1         0.0112187    451.588   1107.96   1.00163
       |
       :
131 | eta1_j105       0.00328424   1539.36   2194.83   1.00002
132 | alpha_t1         0.000213087  3774.24   3920.48   1.00044
       |
       :
143 | alpha_t12        0.00266005   541.427   1867.54   1.0084
```


Appendix C

Fits interpretation

We provide here some comments and insights about the air pollution levels in Lombardy that we can derive from the spatio-temporal fits performed in Chapter 3. We recall that they refer to the weekly averages concentrations of PM_{10} , recorded for $n = 105$ monitoring stations and on a time period of $T = 12$ corresponding to the first three months of 2018.

tipo notare che l'area di brescia magari è inquinata, poi che invece il cluster più inquinato diventa verso milano o bergamo, ecc, commenti ambientali sui fit

Non so se questa parte è utile/interessante da inserire

Bibliography

- [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 (cit. on pp. v, vii, 3, 5, 8, 31, 32).
- [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 pp. v, vii, 3, 33).
- [Wad23] Sara Wade. “Bayesian cluster analysis”. In: *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences* 381 (Mar. 2023), p. 20220149. DOI: 10.1098/rsta.2022.0149 (cit. on p. 2).
- [GMR22] Isobel Gormley, Thomas Murphy, and Adrian Raftery. “Model-Based Clustering”. In: *Annual Review of Statistics and Its Application* 10 (Oct. 2022). DOI: 10.1146/annurev-statistics-033121-115326 (cit. on p. 2).
- [Fra08] Jessica Franzén. “Bayesian Cluster Analysis : Some Extensions to Non-standard Situations”. In: 2008. URL: <https://api.semanticscholar.org/CorpusID:12397258> (cit. on p. 2).
- [Fer73] Thomas S. Ferguson. “A Bayesian Analysis of Some Nonparametric Problems”. In: *The Annals of Statistics* 1.2 (1973), pp. 209–230. DOI: 10.1214/aos/1176342360. URL: <https://doi.org/10.1214/aos/1176342360> (cit. on p. 2).
- [Gra23] Clara Grazian. *A review on Bayesian model-based clustering*. Mar. 2023. DOI: 10.48550/arXiv.2303.17182 (cit. on p. 3).
- [Teh10] Yee Whye Teh. “Dirichlet Process”. In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 280–287. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_219 (cit. on p. 3).
- [BM73] David Blackwell and James B. MacQueen. “Ferguson Distributions Via Polya Urn Schemes”. In: *The Annals of Statistics* 1.2 (1973), pp. 353–355. DOI: 10.1214/aos/1176342372 (cit. on p. 3).
- [Ald85] David J. Aldous. “Exchangeability and related topics”. In: *École d’Été de Probabilités de Saint-Flour XIII — 1983*. Ed. by P. L. Hennequin. Berlin, Heidelberg: Springer Berlin Heidelberg, 1985, pp. 1–198. ISBN: 978-3-540-39316-0 (cit. on p. 3).

- [Bar+12] Luis E Nieto Barajas et al. “A Time-Series DDP for Functional Proteomics Profiles”. In: *Biometrics* 68.3 (Jan. 2012), pp. 859–868. DOI: 10.1111/j.1541-0420.2011.01724.x (cit. on p. 5).
- [AW15] Isadora Antoniano Villalobos and Stephen Walker. “A Nonparametric Model for Stationary Time Series”. In: *Journal of Time Series Analysis* 63 (Aug. 2015). DOI: 10.1111/jtsa.12146 (cit. on p. 5).
- [GMR16] Luis Gutiérrez, Ramsés H. Mena, and Matteo Ruggiero. “A time dependent Bayesian nonparametric model for air quality analysisa”. In: *Computational Statistics & Data Analysis* 95.C (2016), pp. 161–175. DOI: 10.1016/j.csda.2015.10.00. URL: <https://ideas.repec.org/a/eee/csdana/v95y2016icp161-175.html> (cit. on p. 5).
- [Jo+16] Seongil Jo et al. “Dependent Species Sampling Models for Spatial Density Estimation”. In: *Bayesian Analysis* 12 (May 2016). DOI: 10.1214/16-BA1006 (cit. on p. 5).
- [KG18] Maria Kalli and Jim Griffin. “Bayesian nonparametric vector autoregressive models”. In: *Journal of Econometrics* 203 (Jan. 2018). DOI: 10.1016/j.jeconom.2017.11.009 (cit. on p. 5).
- [DK18] Maria De Iorio and Athanasios Kottas. “Modeling for Dynamic Ordinal Regression Relationships: An Application to Estimating Maturity of Rockfish in California”. In: *Journal of the American Statistical Association* 113.521 (2018), pp. 68–80. DOI: 10.1080/01621459.2017.1328357 (cit. on p. 5).
- [De +19] Maria De Iorio et al. *Bayesian nonparametric temporal dynamic clustering via autoregressive Dirichlet priors*. Oct. 2019. DOI: 10.48550/arXiv.1910.10443 (cit. on p. 5).
- [Car+17] François Caron et al. “Generalized Pólya urn for time-varying Pitman-Yor processes”. In: *Journal of Machine Learning Research* 18.1 (Jan. 2017), pp. 836–867. ISSN: 1532-4435 (cit. on p. 5).
- [Dun16] Earl Duncan. *Deriving the Full Conditionals*. BRAG: Bayesian Research & Application Group. 2016. URL: <https://bragqut.wordpress.com/wp-content/uploads/2018/04/deriving-the-full-conditionals.pdf> (cit. on p. 9).
- [Chr+10] Ronald Christensen et al. *Bayesian ideas and data analysis. An introduction for scientists and statisticians*. CRC Press, 2010. DOI: 10.1201/9781439894798 (cit. on p. 12).
- [GHV13] Andrew Gelman, Jessica Hwang, and Aki Vehtari. “Understanding predictive information criteria for Bayesian models”. In: *Statistics and Computing* 24 (July 2013). DOI: 10.1007/s11222-013-9416-2 (cit. on p. 12).
- [Har90] J.A. Hartigan. “Partition models”. In: *Communications in Statistics - Theory and Methods* 19.8 (1990), pp. 2745–2756. DOI: 10.1080/03610929008830345 (cit. on p. 12).

- [BH93] Daniel Barry and J. A. Hartigan. “A Bayesian Analysis for Change Point Problems”. In: *Journal of the American Statistical Association* 88.421 (1993), pp. 309–319. ISSN: 01621459, 1537274X. URL: <http://www.jstor.org/stable/2290726> (cit. on p. 12).
- [Cro97] Evelyn M. Crowley. “Product Partition Models for Normal Means”. In: *Journal of the American Statistical Association* 92.437 (1997), pp. 192–198. ISSN: 01621459. URL: <http://www.jstor.org/stable/2291463> (cit. on p. 12).
- [PQ15] Garrett Page and Fernando Quintana. “Spatial Product Partition Models”. In: *Bayesian Analysis* 11 (Apr. 2015). DOI: [10.1214/15-BA971](https://doi.org/10.1214/15-BA971) (cit. on p. 12).
- [DH01] D. Denison and C Holmes. “Bayesian Partitioning for Estimating Disease Risk”. In: *Biometrics* 57 (Apr. 2001), pp. 143–9. DOI: [10.1111/j.0006-341X.2001.00143.x](https://doi.org/10.1111/j.0006-341X.2001.00143.x) (cit. on p. 14).
- [De +15] Pierpaolo De Blasi et al. “Are Gibbs-Type Priors the Most Natural Generalization of the Dirichlet Process?” In: *IEEE Trans. Pattern Anal. Mach. Intell.* 37.2 (Feb. 2015), pp. 212–229. ISSN: 0162-8828. DOI: [10.1109/TPAMI.2013.217](https://doi.org/10.1109/TPAMI.2013.217) (cit. on p. 15).
- [MQR11] Peter Müller, Fernando Quintana, and Gary Rosner. “A Product Partition Model With Regression on Covariates”. In: *Journal of computational and graphical statistics: a joint publication of American Statistical Association, Institute of Mathematical Statistics, Interface Foundation of North America* 20 (Mar. 2011), pp. 260–278. DOI: [10.1198/jcgs.2011.09066](https://doi.org/10.1198/jcgs.2011.09066) (cit. on p. 15).
- [QMP15] Fernando Quintana, Peter Müller, and Ana Luisa Papoila. “Cluster-Specific Variable Selection for Product Partition Models”. In: *Scandinavian Journal of Statistics* 42 (Mar. 2015). DOI: [10.1111/sjos.12151](https://doi.org/10.1111/sjos.12151) (cit. on p. 15).
- [PQ18] Garrett Page and Fernando Quintana. “Calibrating covariate informed product partition models”. In: *Statistics and Computing* 28 (Sept. 2018), pp. 1–23. DOI: [10.1007/s11222-017-9777-z](https://doi.org/10.1007/s11222-017-9777-z) (cit. on p. 18).
- [Gow71] J. C. Gower. “A General Coefficient of Similarity and Some of Its Properties”. In: *Biometrics* 27.4 (1971), pp. 857–871. ISSN: 0006341X, 15410420. URL: <http://www.jstor.org/stable/2528823> (cit. on p. 19).
- [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](https://doi.org/10.1137/141000671) (cit. on p. 23).
- [Law+79] C. L. Lawson et al. “Basic Linear Algebra Subprograms for Fortran Usage”. In: *ACM Trans. Math. Softw.* 5.3 (Sept. 1979), pp. 308–323. ISSN: 0098-3500. DOI: [10.1145/355841.355847](https://doi.org/10.1145/355841.355847) (cit. on p. 23).

- [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 (cit. on p. 23).
- [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 (cit. on p. 23).
- [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. 25).
- [HA85] Lawrence J. Hubert and Phipps Arabie. “Comparing partitions”. In: *Journal of Classification* 2 (1985), pp. 193–218. URL: <https://api.semanticscholar.org/CorpusID:189915041> (cit. on p. 31).
- [R C24] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2024. URL: <https://www.R-project.org/> (cit. on p. 31).
- [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. 31).
- [Gel04] Andrew Gelman. “Prior Distributions for Variance Parameters in Hierarchical Models”. In: *Economics and Econometrics Research Institute (EERI), EERI Research Paper Series* 1 (Jan. 2004) (cit. on p. 62).
- [GXG18] Hong Ge, Kai Xu, and Zoubin Ghahramani. “Turing: a language for flexible probabilistic inference”. In: *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain*. 2018, pp. 1682–1690. URL: <http://proceedings.mlr.press/v84/ge18b.html> (cit. on p. 90).

Acknowledgements

"This is to be my haven for many long years, my niche which I enter with such a mistrustful, such a painful sensation... And who knows? Maybe when I come to leave it many years hence I may regret it!"

— Fëdor Dostoevskij, *The House of the Dead*

I would like to thank my advisors Alessandra Guglielmi and Alessandro Carminati, primarily for having made me feel, during the entire course of the thesis, in a very calm, lovely atmosphere, in which I have been able to feel perfectly at ease (which is notoriously a NP-hard problem). The epigraphs of my beloved author Dostoevsky and the title inspired by the Star Wars films, as well as some poetic licenses and easter eggs scattered across the chapters (which they have kindly allowed me to keep) I think prove this feeling of sweet and respectful confidence.

I thank professor Alessandra Guglielmi for having guided me through the fog of uncertainty and puzzlement that often the thesis brings, or in general the final period of university. From the beginning she shared a genuine interest in the work that we would have done and, I believe, an overall confidence in my chances of really carrying it out.

I thank Alessandro Carminati for having assisted me in many of the most critical phases of the thesis. His action always precise and effective has been necessary to solve the various theoretical puddles in which I got stuck. Moreover, we share the same passion for Julia and, I believe, the delight for having overthrown the not-so-missed C of the old model implementation.

I thank my family for having helped me, in different ways, during these years of university, and my friends and fellows who have always and wisely brought to light my highest potential and resources. The passion for solving problems, inventing creative solutions, getting lost in calculations, wandering trough the magical world of the most advanced mathematics, sharing doubts and reasonings: all this has always found in you a wonderful and unforeseen companion.

I also thank all the boys and girls with whom I have had the honour to work and play in the various summer camps, for steadily reminding me of the cheerful and soft spirit with which all the various challenges can be faced.

Ringraziamenti

“Ecco il mio ponte d’approdo per molti lunghi anni, il mio angioletto, 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 rimiangerlo!”

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

Vorrei ringraziare innanzitutto i miei relatori Alessandra Guglielmi e Alessandro Carminati, principalmente per avermi fatto sentire, durante l’intero svolgimento della tesi, in un clima molto tranquillo, sereno, in cui ho avuto modo di trovarmi pienamente a mio agio. Le epigrafi del mio caro autore Dostoevskij e il titolo ispirato alla saga di Star Wars, nonché alcune licenze poetiche sparse nei vari capitoli (che loro mi hanno gentilmente concesso di tenere) credo dimostrino questa atmosfera di piacevole ma rispettosa confidenza.

Ringrazio la professoressa Alessandra Guglielmi per avermi guidato attraverso la nebbia di incertezza e confusione che spesso accompagna la tesi, o in senso lato gli ultimi mesi di università. Fin dall’inizio ha infatti condiviso un sincero interesse per il lavoro che avremmo dovuto svolgere, e una credo generale fiducia nelle mie possibilità di condurlo a termine.

Ringrazio Alessandro Carminati per avermi aiutato in molte delle fasi più critiche della tesi. Il suo intervento sempre preciso e puntuale è stato necessario per risolvere le varie pozzanghere teoriche in cui mi ero impantanato. Inoltre, condividiamo la stessa passione per Julia e, credo, la soddisfazione per aver battuto il non-così-compianto C della vecchia implementazione del modello.

Ringrazio la mia famiglia per avermi aiutato, in vari modi, durante questi anni di università, e i miei amici e compagni di studio, i quali hanno sempre sapientemente fatto emergere le mie migliori potenzialità e risorse. La passione per risolvere problemi, ideare soluzioni creative, perdersi nei calcoli, addentrarsi nel magico mondo della matematica più avanzata, condividere dubbi e ragionamenti: tutto questo ha sempre trovato in voi un’ottima e inaspettata compagnia.

Ringrazio anche tutti i bambini e ragazzi con cui ho avuto modo di lavorare e giocare nei vari campi estivi, per farmi sempre ricordare dello spirito gioioso e leggero con cui si possono approcciare tutte le varie sfide.