Politecnico di Milano

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

MASTER THESIS

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

Academic Year 2023–2024

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 (G. L. Page et al., 2022) 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 through 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 (Fassò et al., 2023) 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 (G. L. Page et al., 2022) 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 (Fassò et al., 2023).

Parole chiave: modellistica bayesiana, clustering, dati spazio-temporali, MCMC, Julia

Contents

List of Figures

List of Tables

Introduction

Clustering is a fundamental technique of unsupervised learning where a set of data points has to be divided into homogenous groups of units which exhibit a similar behaviour relative to a target variable. It has long been a powerful tool for identifying structures and patterns in data, especially in contexts where relationships between observations are complex, such as when the target variable is affected by multiple factors simultaneously. For this reason, clustering methods have become increasingly popular across a variety of scientific fields, including social sciences, climate and environmental analysis, economics, and healthcare.

Clustering approaches are generally divided into two main categories: algorithmic and model-based methods.

Algorithmic methods such as hierarchical, partition-based, or density-based methods, address the clustering problem as an optimization problem where a certain metric has to be minimised (or maximised). For instance, partition-based methods like k-means (J. A. Hartigan et al., 1979) generate 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 their assigned cluster centroid. However, these methods require to specify the desired number k of clusters in advance and are limited to numerical data. Hierarchical clustering methods, on the other hand, build a tree-like structure of clustering solutions, represented as a dendrogram, which captures the relationships among potential clusters (Jain et al., 1988) (Kaufman et al., 1990). This is done through either an agglomerative (bottom-up) strategy, where each unit starts in her own cluster that gets iteratively combined with other units or clusters, or in a divisive (top-down) strategy, where units start in a single cluster that is iteratively subdivided. These methods do not necessitate a predefined number of clusters, but they are highly sensitive to the choice of the distance metric, which drives all merging and splitting decisions. Density based methods, such as DBSCAN (Ester et al., 1996), define clusters through a density metric which can produce more flexible structures with clusters of varying shapes. However, these methods remain sensitive to the choice of the density parameters and may yield less interpretable and irregular clusters.

In summary, these algorithmic approaches are largely heuristic (Gormley et al., 2022), performing well only with well-separated clusters or standard geometric forms, but often failing in more complex scenarios. Furthermore, lacking a solid statistical foundation, these methods can lead to unsatisfactory results and provide no assessment about clustering uncertainty.

An alternative and more flexible approach is therefore proposed by model-based methods, which assume a probabilistic modelling of the data. This is generally done through a mixture model (Bouveyron et al., 2019) (Grün, 2018) (Böhning, 2007), where each mixture component corresponds to a cluster with its specific cluster parameters (Grazian, 2023). 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

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

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 $\sum_{j=1}^{J} \pi_j = 1$, $\boldsymbol{\vartheta}_j^\star$ are the cluster-specific parameters, and $f_j(\cdot, |\vartheta_j^\star)$ 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 $\boldsymbol{\vartheta_j^\star} = (\mu_j^\star, \sigma_j^{2\star})$, or $\boldsymbol{\vartheta_j^\star} = (\boldsymbol{\mu}_j^\star, \Sigma_j^\star)$ 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 (Franzén, 2008). 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 $\pi_1, \dots, \pi_J$. The estimation step is often performed through the Expectation-Maximization (EM) algorithm (McLachlan et al., 2008) (Ng et al., 2004), 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 through the Bayes rule.

This approach of mixture model can be naturally moved in a Bayesian framework. As opposed to MLE, Bayesian mixture models incorporate prior information on the parameters, allowing to assess uncertainty in the clustering structure (Wade, 2023). Each parameter is therefore treated as a random variable with a corresponding prior distribution. This leads (1) to be reformulated into

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

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 $\boldsymbol{\vartheta}_j^\star$ 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 (Grazian, 2023) (i.e. that does not require a predetermined

number J of components) can be introduced. This leads to a model formulation in the form

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

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 $\boldsymbol{\vartheta}_1^\star,\dots,\boldsymbol{\vartheta}_K^\star$ the unique values of $\boldsymbol{\vartheta}_1,\dots,\boldsymbol{\vartheta}_n$, the generic h-cluster can be defined as $S_h = \{i \in \{1,\dots,n\} : \boldsymbol{\vartheta}_i = \boldsymbol{\vartheta}_h^\star\}$. This infinite mixture model often relies on the Dirichlet Process (DP) (Ferguson, 1973) for the algorithm implementation. The Dirichlet Process plays a significant role in general in Bayesian nonparametrics (Grazian, 2023), but especially in clustering, thanks to its computationally manageable implementation through the stick-breaking representation (Teh, 2010), the Pólya urn representation (Blackwell et al., 1973), and the Chinese restaurant process (CRP) (Aldous, 1985).

However, implementing Bayesian methods often requires the design of Markov Chain Monte Carlo (MCMC) algorithms, which can often be complex and computationally intensive. MCMC techniques, such as the Metropolis-Hastings and Gibbs sampling algorithms, are essential for approximating the posterior distributions of the model parameters and to allow inference on the generated clusters (A. Gelman et al., 2003) (Robert et al., 2000). These algorithms work by constructing a Markov chain that eventually generates samples from the target distribution; a process that requires a certain tuning of the hyperparameters and final convergence diagnostics to ensure reliability of the results. The iterative nature of MCMC also implies that considerable computational resources may be needed, particularly for models with an high-dimensional parameter space or simply when working with large datasets. Consequently, while Bayesian methods provide a robust framework for modelling uncertainty and incorporating prior knowledge, they also demand significant computational effort and expertise in algorithm design to effectively extract their full potential.

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 (Mozdzen et al., 2022), 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, producing clusters which evolve in time in a more gentle and interpretable way, 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 (G. L. Page et al., 2022) 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 (Fassò et al., 2023), 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 (Nieto-Barajas et al., 2012) (Antoniano Villalobos et al., 2015) (Gutiérrez et al., 2016) (Jo et al., 2017) (Kalli et al., 2018) (De Iorio and Kottas, 2018) (De Iorio, Favaro, et al., 2019) or through the Pólya urn scheme (Caron et al., 2017). 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) (G. L. Page et al., 2022), 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 $\rho_t = \{S_{1t}, \dots, S_{k_t t}\}$ 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 $\boldsymbol{c}_t = \{c_{1t}, \dots, c_{nt}\}$, 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 $(\rho_1, \dots, \rho_T)$, 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

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

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

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

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 $\gamma_{it} \overset{\text{ind}}{\sim} \text{Ber}(\alpha_t)$ 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

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

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 γ_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_{it}a, 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 $\gamma_{it}^{(d-1)} = 0 \to \gamma_{it}^{(d)} = 1$ 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 through 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

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

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 $\sigma^{2\star}_{jt}$, τ_t^2, and λ^2, we chose to model them through 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 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 through 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 $\boldsymbol{\beta}_t$ 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.

$$
\begin{aligned}
Y_{it}|Y_{it-1}, \boldsymbol{\mu}^\star_t, \boldsymbol{\sigma}^{2\star}_t, \boldsymbol{\eta}, \boldsymbol{c}_t &\overset{\text{ind}}{\sim} \mathcal{N}(\mu^\star_{c_{it}t} + \eta_{1i} Y_{it-1} + \boldsymbol{x}_{it}^T \boldsymbol{\beta}_t, \sigma^{2\star}_{c_{it}t}(1-\eta^2_{1i})) \\
Y_{i1} &\overset{\text{ind}}{\sim} \mathcal{N}(\mu^\star_{c_{i1}1} + \boldsymbol{x}_{i1}^T \boldsymbol{\beta}_1, \sigma^{2\star}_{c_{i1}1}) \\
\boldsymbol{\beta}_t &\overset{\text{ind}}{\sim} \mathcal{N}_p(\boldsymbol{b}, s^2 I) \\
\xi_i = \text{Logit}(\tfrac{1}{2}(\eta_{1i}+1)) &\overset{\text{ind}}{\sim} \text{Laplace}(a, b) \\
(\mu^\star_{jt}, \sigma^{2\star}_{jt}) &\overset{\text{ind}}{\sim} \mathcal{N}(\vartheta_t, \tau^2_t) \times \text{invGamma}(a_\sigma, b_\sigma) \\
\vartheta_t|\vartheta_{t-1} &\overset{\text{ind}}{\sim} \mathcal{N}((1-\varphi_1)\varphi_0 + \varphi_1 \vartheta_{t-1}, \lambda^2(1-\varphi_1^2)) \\
(\vartheta_1, \tau^2_t) &\overset{\text{iid}}{\sim} \mathcal{N}(\varphi_0, \lambda^2) \times \text{invGamma}(a_\tau, b_\tau) \\
(\varphi_0, \varphi_1, \lambda^2) &\sim \mathcal{N}(m_0, s_0^2) \times \mathcal{U}(-1, 1) \times \text{invGamma}(a_\lambda, b_\lambda) \\
\{\boldsymbol{c}_t, \dots, \boldsymbol{c}_T\} \sim \text{tRPM}(\boldsymbol{\alpha}, M) &\text{ with } \alpha_t \overset{\text{iid}}{\sim} \text{Beta}(a_\alpha, b_\alpha)
\end{aligned}
\tag{1.5}
$$

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.

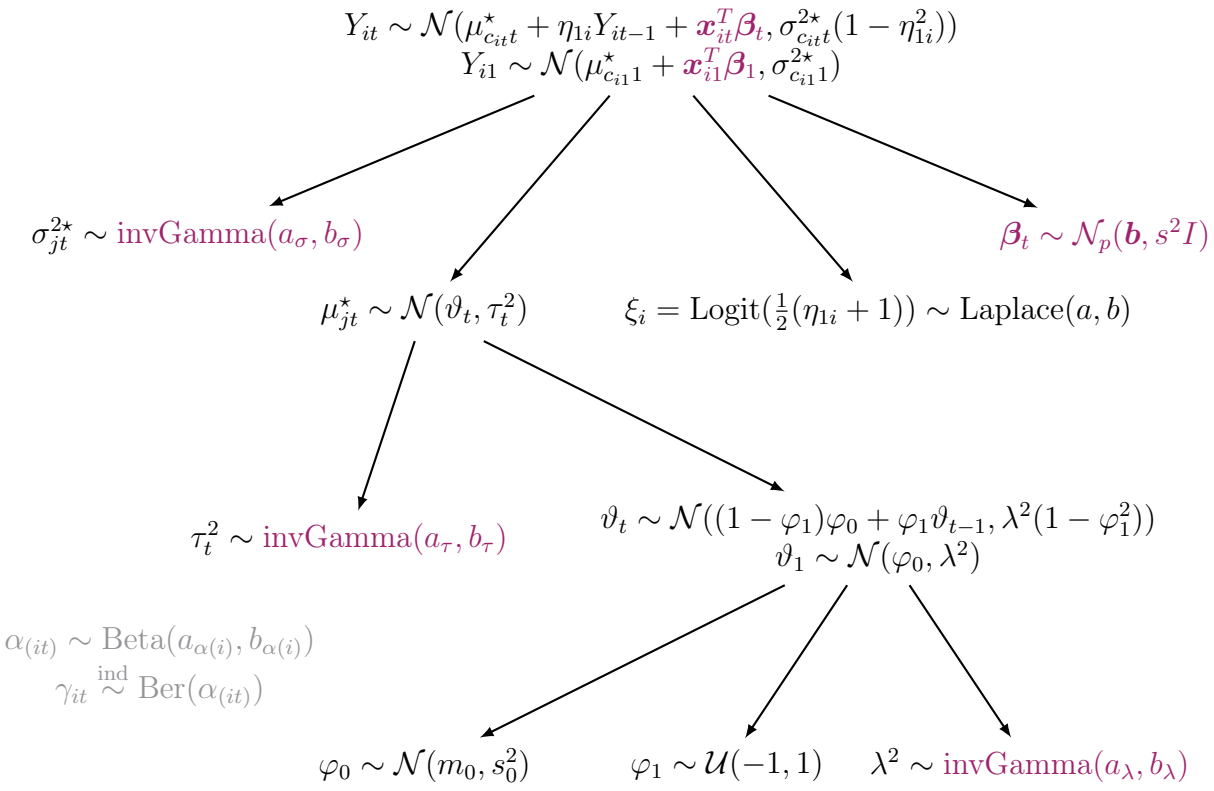


Figure 1.1: Graph visualization of the DRPM model, with highlighted in dark red the changes that we made to the original formulation and in gray the internal variables of the model.

In the course of this work, for the sake of clarity, we will refer to CDRPM for the original model formulation of (G. L. Page et al., 2022), 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 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 *posterior* $\propto$ *likelihood* $\cdot$ *prior*, but followed useful suggestions and tricks from (Duncan, 2016). The other variables not included here, namely η_{1i} and φ_1, involved instead the classical Metropolis update.

- update $\sigma^{2\star}_{jt}$. 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.

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

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

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

  $$a_{\tau(\text{post})} = a_\sigma + \frac{|S_{jt}|}{2} \quad b_{\tau(\text{post})} = b_\sigma + \frac{1}{2}\sum_{i\in S_{jt}} (Y_{it} - \mu^\star_{jt} - \eta_{1i}Y_{it-1} - \boldsymbol{x}^T_{it}\boldsymbol{\beta}_t)^2$$

  (1.6)

- update $\mu^\star_{jt}$. This update rule is the same for both JDRPM and CDRPM.

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

  $$\sigma^2_{\mu^\star_{jt}(\text{post})} = \frac{1}{\frac{1}{\tau^2_t} + \frac{|S_{jt}|}{\sigma^{2\star}_{jt}}} \quad \mu_{\mu^\star_{jt}(\text{post})} = \sigma^2_{\mu^\star_{jt}(\text{post})}\left(\frac{\vartheta_t}{\tau^2_t} + \frac{\sum_{i\in S_{jt}}(Y_{i1} - \boldsymbol{x}^T_{it}\boldsymbol{\beta}_t)}{\sigma^{2\star}_{jt}}\right)$$

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

  $$\sigma^2_{\mu^\star_{jt}(\text{post})} = \frac{1}{\frac{1}{\tau^2_t} + \frac{\sum_{i\in S_{jt}}\frac{1}{1-\eta^2_{1i}}}{\sigma^{2\star}_{jt}}} \quad \mu_{\mu^\star_{jt}(\text{post})} = \sigma^2_{\mu^\star_{jt}(\text{post})}\left(\frac{\vartheta_t}{\tau^2_t} + \frac{\sum_{i\in S_{jt}}\frac{Y_{it} - \eta_{1i}Y_{i,t-1} - \boldsymbol{x}^T_{it}\boldsymbol{\beta}_t}{1-\eta^2_{1i}}}{\sigma^{2\star}_{jt}}\right)$$

  (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.

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

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

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

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

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

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

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

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

  Here $\mathcal{N}\text{Canon}(\boldsymbol{h}, J)$ is the canonical formulation of the $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$, with $\boldsymbol{h} = \Sigma^{-1}\boldsymbol{\mu}$ 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.

  $$\begin{aligned} &f(\tau_t^2|-) \propto \text{kernel of a invGamma}(a_{\tau(\text{post})}, b_{\tau(\text{post})}) \text{ with} \\ &a_{\tau(\text{post})} = \frac{k_t}{2} + a_\tau \quad b_{\tau(\text{post})} = \frac{\sum_{j=1}^{k_t}(\mu^{\star}_{jt} - \vartheta_t)^2}{2} + b_\tau \end{aligned} \tag{1.9}$$

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

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

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

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

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

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

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

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

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

$$\mu_{\vartheta_t(\text{post})} = \sigma^2_{\vartheta_t(\text{post})} \left(\frac{\varphi_0}{\lambda^2} + \frac{\varphi_1(\vartheta_{t+1} - (1-\varphi_1)\varphi_0)}{\lambda^2(1-\varphi_1^2)} + \frac{\sum_{j=1}^{k_t} \mu^\star_{jt}}{\tau_t^2} \right) \tag{1.10}$$

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

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

$$\sigma^2_{\varphi_0(\text{post})} = \frac{1}{\frac{1}{s_0^2} + \frac{1}{\lambda^2} + \frac{(T-1)(1-\varphi_1)^2}{\lambda^2(1-\varphi_1^2)}}$$

$$\mu_{\varphi_0(\text{post})} = \sigma^2_{\varphi_0(\text{post})} \left(\frac{m_0}{s_0^2} + \frac{\vartheta_1}{\lambda^2} + \frac{1-\varphi_1}{\lambda^2(1-\varphi_1^2)} \sum_{t=2}^{T} (\vartheta_t - \varphi_1 \vartheta_{t-1}) \right) \tag{1.11}$$

- 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.

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

$$a_{\lambda(\text{post})} = \frac{T}{2} + a_\lambda$$

$$b_{\lambda(\text{post})} = \frac{(\vartheta_1 - \varphi_0)^2}{2} + \sum_{t=2}^{T} \frac{(\vartheta_t - (1-\varphi_1)\varphi_0 - \varphi_1 \vartheta_{t-1})^2}{2} + b_\lambda \tag{1.12}$$

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

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

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

- 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 $\mathcal{N}(\mu_{Y_{it}(\text{post})}, \sigma^2_{Y_{it}(\text{post})})$ with

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

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

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

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

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

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 (Christensen et al., 2010) and (Andrew Gelman et al., 2013).

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 relies on the Product Partition Model (PPM) (J. Hartigan, 1990) (Barry et al., 1993) (Crowley, 1997). At its core, this model assigns a prior for ρ_t in

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

for $d = 1, \ldots,$ *draws* **do**
  if *target variable has missing values* **then**
    update the missing Y_{it}'s using (1.14)
  end
  for $t = 1, \ldots, T$ **do**
    for $i = 1, \ldots, n$ **do**
      if $t = 1$ **then**
        $\gamma_{it} = 0$
      else
        update γ_{it}
      end
    end
    for $i \in \{l : \gamma_{lt} = 0\}$ **do**
      update c_{it} (i.e. update ρ_t)
    end
    update $\mu^{\star}_{jt}$ using (1.7)
    update $\sigma^{2\star}_{jt}$ using (1.6)
    if *are there covariates in the likelihood* **then**
      update $\boldsymbol{\beta}_t$
    end
    update ϑ_t using (1.10)
    update τ^2_t using (1.9)
  end
  if *update_eta = true* **then**
    update η_{1i} using Metropolis sampling
  end
  if *update_alpha = true* **then**
    update α using (1.13)
  end
  update φ_0 using (1.11)
  if *update_phi1 = true* **then**
    update φ_1 using Metropolis sampling
  end
  update λ^2 using (1.12)
  if $d >$ *burnin and* $d \,\%\, thin = 0$ **then**
    save MCMC current iterate
  end
end
compute LPML and WAIC metrics
if *perform diagnostics* **then**
  print diagnostics (ESS and $\hat{R}$) on parameters λ^2, φ_0, τ^2_t, ϑ_t, η_{1i}, α
end

the form $P(\rho_t) \propto \prod_{j=1}^{k_t} C(S_{jt})$, with the function $C(S_{jt})$ that measures how likely elements of S_{jt} would be grouped in the same cluster a priori. To enhance this model by incorporating spatial information, the PPM can be extend from being a function of just $C(S_{jt})$ to the more informed one $C(S_{jt}, \boldsymbol{s}^\star_{jt})$, where $\boldsymbol{s}^\star_{jt}$ 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 notation S_h to indicate a general h-th cluster, rather than the more precise S_{jt}. For the same reason, the temporal dependence that linked ρ_t with on $\boldsymbol{\gamma}_t$ and ρ_{t-1}

Regarding the computation of spatial cohesion, several choices are available (G. Page et al., 2015). The main common idea of the following formulas is to favour few spatially connected clusters rather than a lot of singleton ones, to derive more interpretable and meaningful results.

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 **??** and **??** 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.

The first cohesion uses a tessellation idea from (Denison et al., 2001) that considers $\mathcal{D}_h = \sum_{i \in S_h} \|\boldsymbol{s}_i - \bar{\boldsymbol{s}}_h\|$ as the total distance from the units to the cluster centroid $\bar{\boldsymbol{s}}_h$. 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.

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

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 through the parameter a.

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

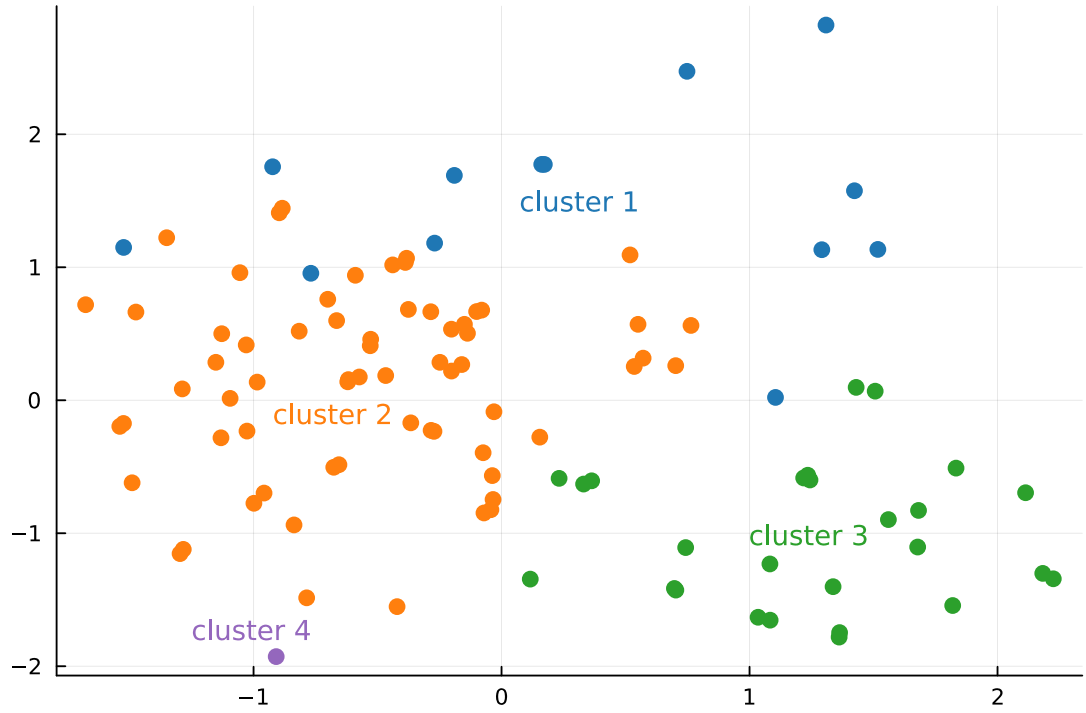


Figure 1.2: Partition and data considered to analyse the spatial cohesions. The points refer to the spatial coordinates of the dataset that will be used in Chapter 3.

According to this function, from Figure **??** 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, cohesions 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 Blasi et al., 2015), is instead often desirable, for theoretical or computational purposes, and the following two cohesions are able to provide it (Müller et al., 2011).

Cohesion 3, called auxiliary similarity, treats the spatial coordinates $\boldsymbol{s}$ as if they were random, applying on them a model such as the Normal/Normal-Inverse-Wishart with $\boldsymbol{\xi} = (\boldsymbol{m}, V)$, $\boldsymbol{s}|\boldsymbol{\xi} \sim \mathcal{N}(\boldsymbol{m}, V)$ 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.

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

On the same line there is cohesion 4, the double dipper cohesion (F. Quintana et al., 2015), which now employs the posterior predictive distribution rather than

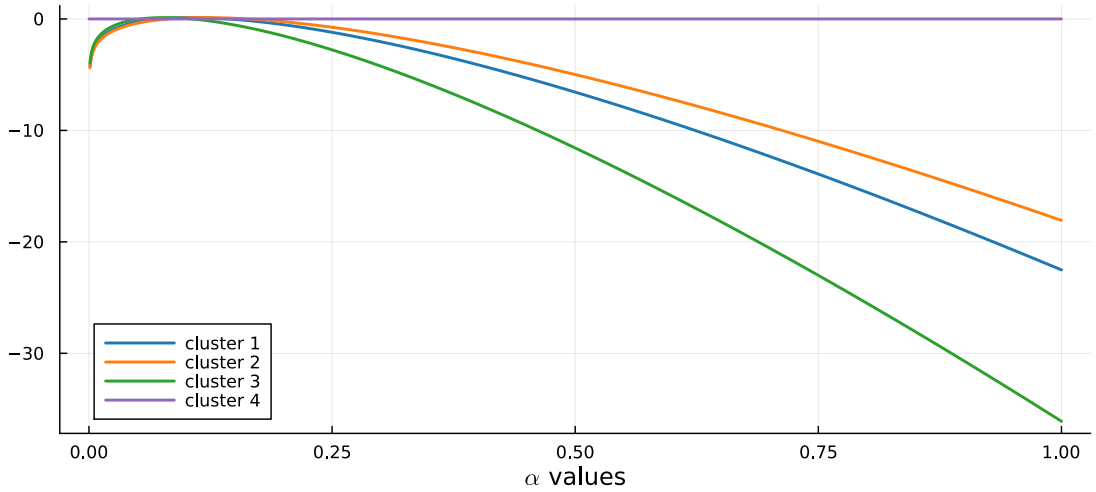


Figure 1.3: Cohesions 1 log-transformed values computed on the test case partition of Figure 1.2, with respect to different values of its tuning parameter α.

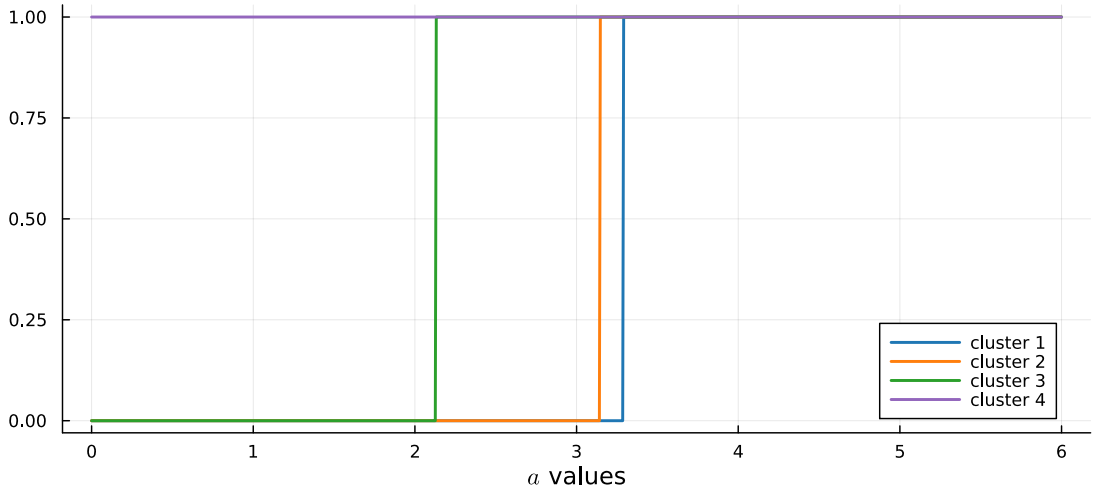


Figure 1.4: Cohesions 2 computed on the test case partition of Figure 1.2, with respect to different values of its tuning parameter a. The values here are without not log-transformed for plotting purposes since otherwise the values would have been $-\infty$ and 0.

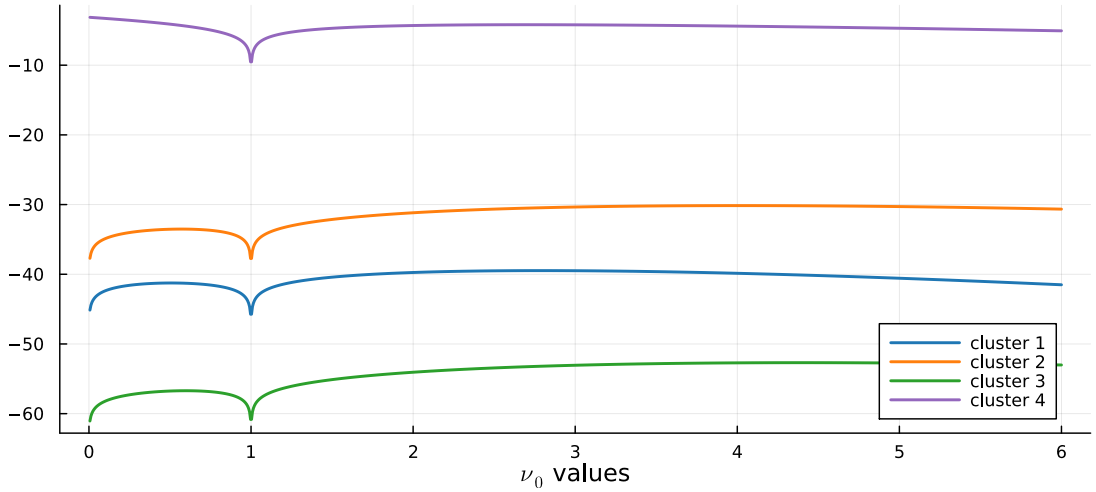


Figure 1.5: Cohesions 3 log-transformed values computed on the test case partition of Figure 1.2, with respect to different values of its tuning parameter ν_0.

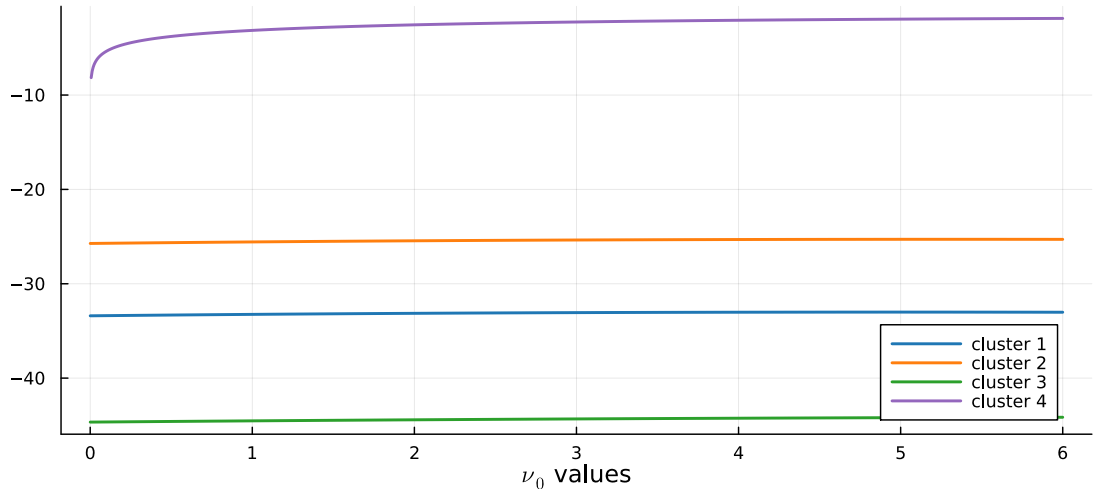


Figure 1.6: Cohesions 4 log-transformed values computed on the test case partition of Figure 1.2, with respect to different values of its tuning parameter ν_0.

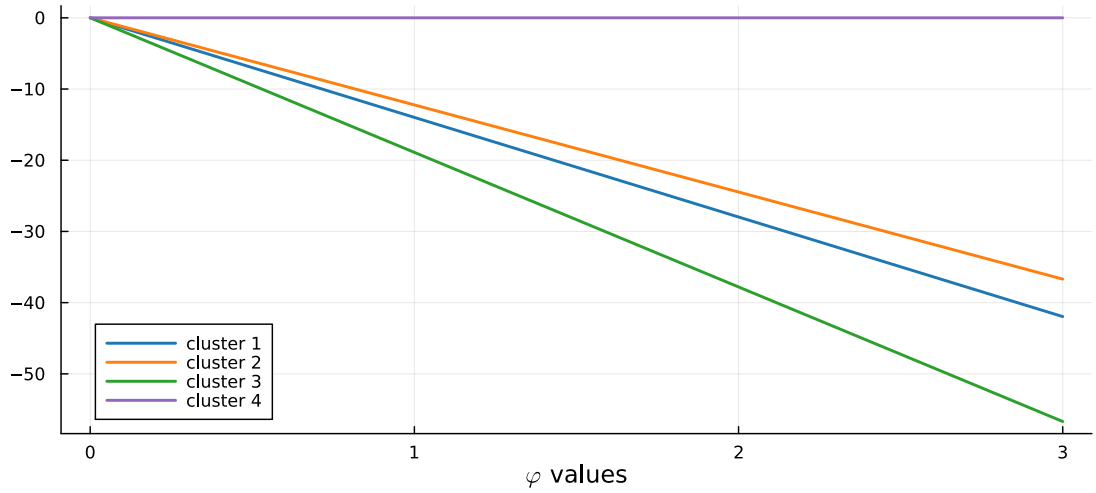


Figure 1.7: Cohesions 5 log-transformed values computed on the test case partition of Figure 1.2, with respect to different values of its tuning parameter φ.

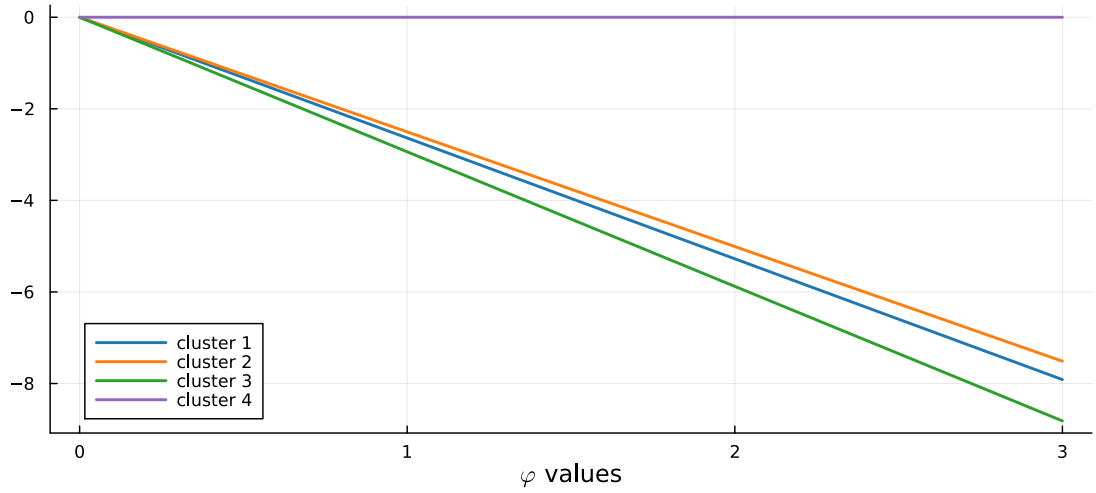


Figure 1.8: Cohesions 3 log-transformed values computed on the test case partition of Figure 1.2, with respect to different values of its tuning parameter φ.

the prior predictive distribution of cohesion C_3.

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

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 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 (G. Page et al., 2018).

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

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

1.3 Covariates similarities analysis

A wide spectrum of choice is also available for covariates similarities (G. Page et al., 2018). To account for them, the idea consists in extending the PPM to make it function of S_{jt}, $\boldsymbol{s}_{jt}^\star$, and now also of $X_{jt}^\star$, being this the $p \times |S_{jt}|$ matrix storing the covariates of the units belonging to cluster j at time t, i.e. $X_{jt}^\star = \{\boldsymbol{x}_{it}^\star = (x_{it1}, \ldots, x_{itp})^T : i \in S_{jt}\}$.

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 $\boldsymbol{x}_{jt1}^\star, \ldots, \boldsymbol{x}_{jtp}^\star$, of which all contributions will be considered independently one to each other. In this way, the final PPM form will be

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

where $\boldsymbol{x}_{jtr}^\star$ represents the vector recording the r-th covariate values for all the units inside cluster S_{jt}, i.e. row r of matrix $X_{jt}^\star$. 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

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

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

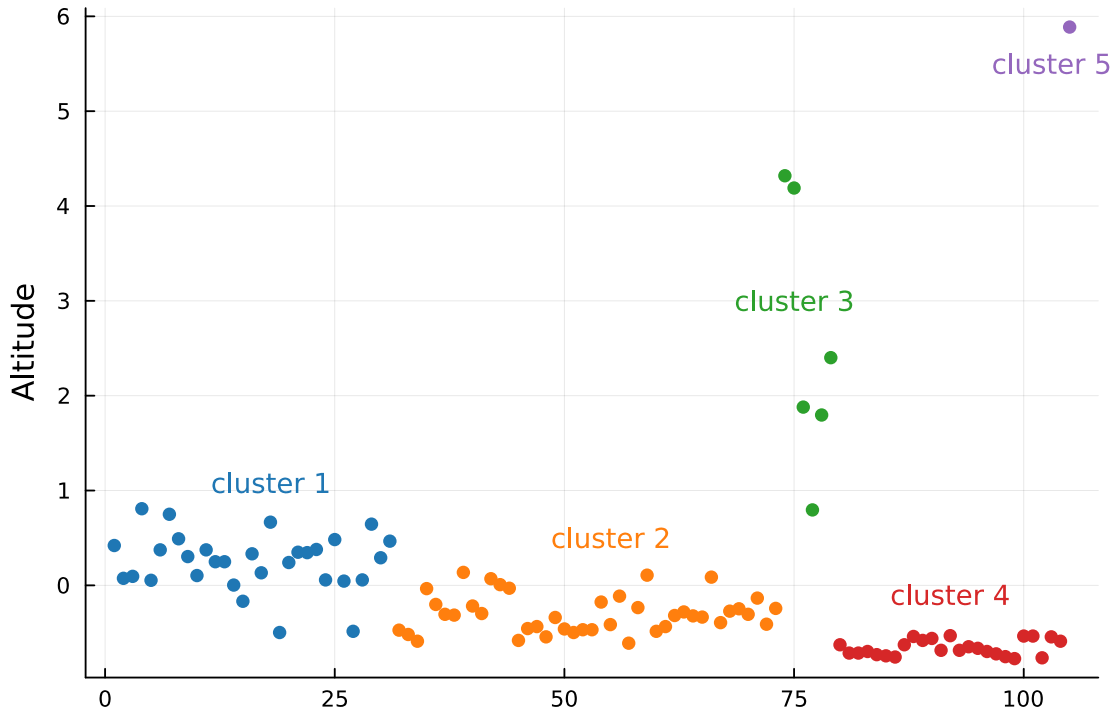


Figure 1.9: Partition considered to analyse the covariate similarities. The order of the units has been modified to plot together all units belonging to the same cluster.

referred to the test case partition displayed in Figure 1.9 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 (G. Page et al., 2018), which as the name suggest can work with both numerical and categorical variables. The general form is

$$g_1(S_h, \boldsymbol{x}_h^\star) = \exp\left\{-\varphi H(S_h, \boldsymbol{x}_h^\star)\right\} \tag{1.23}$$

with $H(S_h, \boldsymbol{x}_h^\star) = \sum_{i \in S_h} (x_i - \bar{x}_h)^2$ for numerical covariates, being $\bar{x}_h$ the mean value of the $\boldsymbol{x}_h^\star$ vector, and $H(S_h, \boldsymbol{x}_h^\star) = -\sum_{c=1}^{C} \hat{p}_c \log(\hat{p}_c)$ 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 $H(S_h, X_h^\star) = \sum_{r=1}^{p} \|\boldsymbol{x}_r - \bar{\boldsymbol{x}}_h\|$.

Another popular choice is the Gower similarity function (Gower, 1971), 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.

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

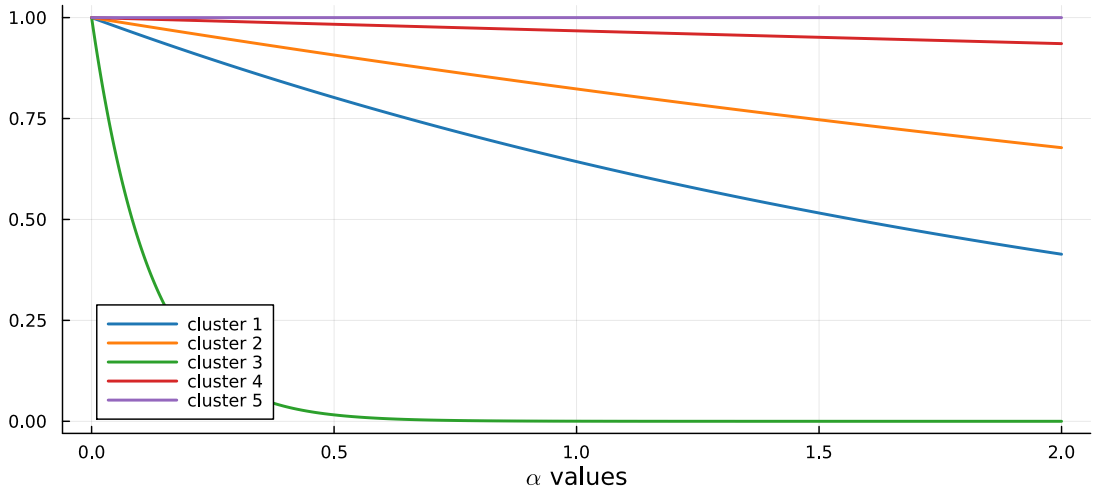


Figure 1.10: Similarity 1 values, computed on the test case partition of Figure 1.9, with respect to different values of its tuning parameter α. The values here are without not log-transformed to better highlight the difference of the computed values among the clusters.

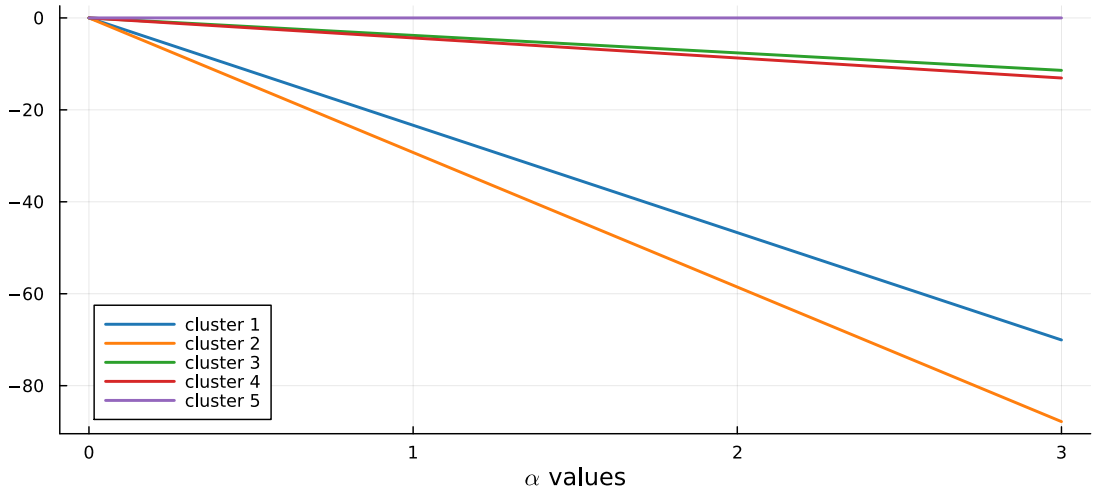


Figure 1.11: Similarity 5 log-transformed values computed on the test case partition of Figure 1.9, with respect to different values of its tuning parameter α.

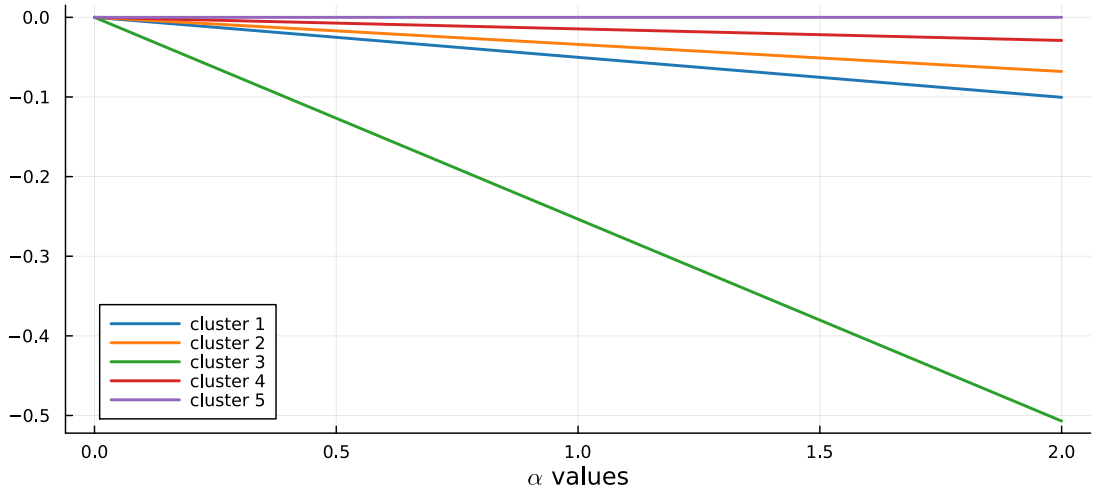


Figure 1.12: Similarity 3 log-transformed values computed on the test case partition of Figure 1.9, with respect to different values of its tuning parameter α.

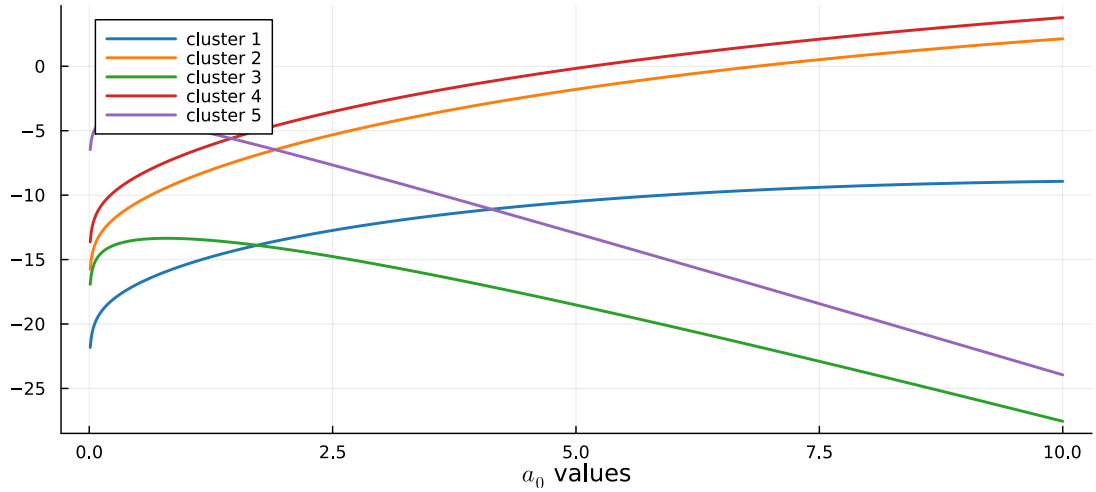


Figure 1.13: Similarity 4 log-transformed values values, computed on the test case partition of Figure 1.9, with respect to different values of its tuning parameter a_0. The other parameters has been set to $\mu_0 = 0$, $\lambda_0 = 1$, $b_0 = 1$.

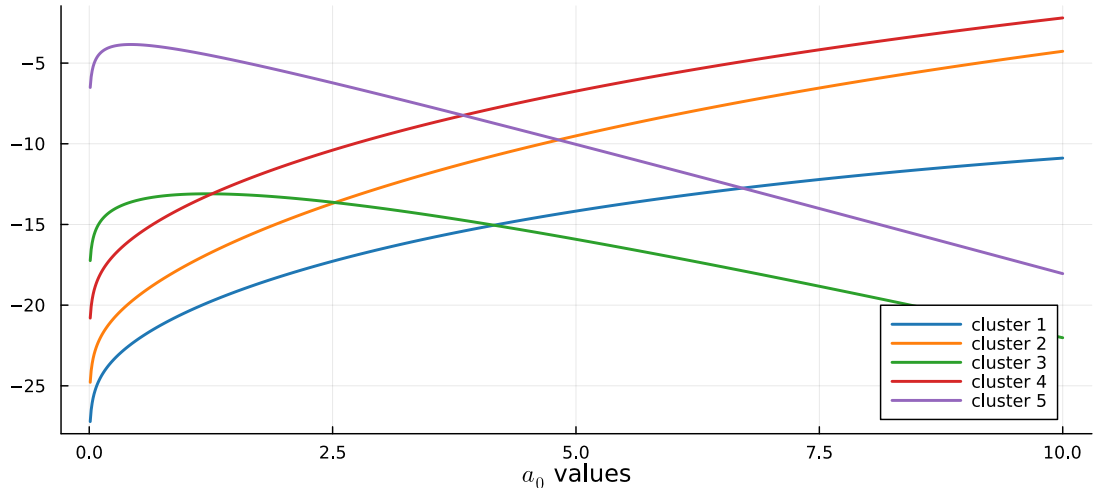


Figure 1.14: Similarity 4 log-transformed values computed on the test case partition of Figure 1.9, with respect to different values of its tuning parameter a_0. The other parameters has been set to $\mu_0 = 0$, $\lambda_0 = 1$, $b_0 = 2$.

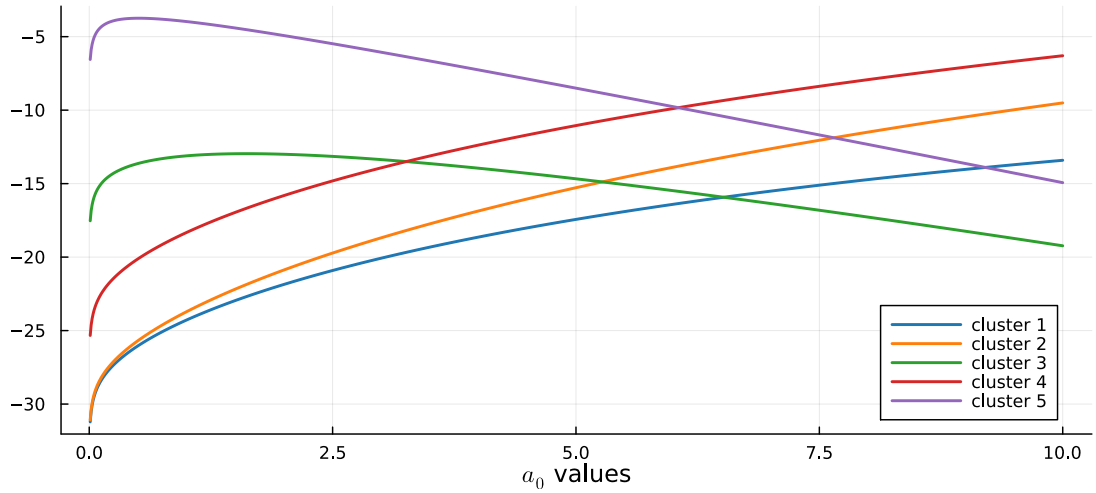


Figure 1.15: Similarity 4 log-transformed values computed on the test case partition of Figure 1.9, with respect to different values of its tuning parameter a_0. The other parameters has been set to $\mu_0 = 0$, $\lambda_0 = 1$, $b_0 = 3$.

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.

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

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 $R = \max(\boldsymbol{x}) - \min(\boldsymbol{x})$ 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 $d(\boldsymbol{x}_i, \boldsymbol{x}_j)$ function.

The last similarity (G. Page et al., 2015) 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.

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

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 **??** and **??**. Intuitively, Figure 1.9 seems to confirm this ranking, by visually reasoning about the sparsity of each cluster. Similarities g_4 and g_2 seems 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 that maybe other distance metrics rather than the L^2 norm could be used in the computation of $H(S_h, \boldsymbol{x}_h^\star)$. On the other hand, similarity g_4 appeared to be quite sensitive to the parameters regulating the invGamma distribution for σ^2. This suggests that covariates should be standardized prior to their usage, in order to simplify the research and uniform the choice of the most appropriate set of parameters. In any case, the JDRPM implementation can provide some flexibility in this regard by possibly assigning separately the pair of a_0 and b_0 for each covariate that we wish to include in the clustering process.

Chapter 2

Implementation and optimizations

"You see, I've brought you my Nellie," I said, going in.
— Fëdor Dostoevskij, *Humiliated and Insulted*

The MCMC algorithm to compute the posterior samples of our updated model, described in Section 1.1, has been implemented in Julia (Bezanson et al., 2017).

Julia is a relatively new programming language that combines the ease and expressiveness of high-level languages with the efficiency and performance characteristic of low-level languages. This balance is primarily achieved through just-in-time (JIT) compilation, using the LLVM framework, along with features such as dynamic multiple dispatch and extensive code specialization against multiple run-time types. Such design enables Julia to be used interactively, in the same fashion to the R, MATLAB, or Python consoles, while also supporting the traditional execution style of statically compiled languages like C, C++, and Fortran. This flexibility facilitates faster development phases, since code sections can be easily evaluated and tested, even line by line, while still guaranteeing efficient implementations through the compilation process. Performance is further enhanced by the native integration of optimized BLAS (Lawson et al., 1979) and LAPACK libraries for linear algebra operations, which are essential for many scientific applications. Moreover, Julia features an extensive ecosystem currently comprising over ten thousand packages that span nearly all branches of science and engineering. Most of these packages are well-tested and highly optimized, thus significantly reducing the implementation time required to users. For instance, in this work, we employed the `Distributions` (Besançon et al., 2021) (Lin et al., 2019) and `Statistics` packages, whereas the original C implementation required developing all statistical functionalities from scratch. Given these benefits, choosing Julia was a natural decision.

As we will discuss in Chapter 3, we obtained improved performance in Julia, with respect to the original C implementation, despite the increased complexity of the model and the associated MCMC algorithm. This enhancement came at the reasonable cost of a modest increase in memory requirements, which nowadays is generally manageable given the current available technologies. This improvement was made possible through various refinements and optimizations, which we will

now briefly outline.

2.1 Optimizations

One of the primary challenges encountered during the implementation of the MCMC algorithm in Julia was managing the amount of memory and allocations that some functions, structures, or algorithms would require. Initially, during the development and testing phases, where the correctness of the algorithm was the top priority, we observed that a significant portion of execution time was actually consumed by Julia's garbage collector, which had the burden of tracking all the allocated memory and reclaim the unused one in order to make it available again for new computations.

In addition to minimizing unnecessary allocations and managing memory more effectively, another critical aspect for improving performance is ensuring type stability of the code. Fortunately, Julia provides several tools to inspect and address both issues.

Regarding type stability, there are various tools available, such as the `Cthulhu` package or the simple `@code_warntype` macro, which help to verify that a function is type-stable. This means that the types of all variables can be correctly predicted by the compiler and remain consistent throughout execution. Type stability is essential for performance as it allows the compiler to generate optimized machine code, eliminating the overhead associated with run-time type checks. Julia, in fact, is dynamically-typed, meaning that variables do not need to be explicitly declared with their types, unlike fully statically-typed languages such as C. In Julia, for instance, a variable initialized as an integer could later become a float or even a string, during execution. However, for performance reasons, such dynamisms should be avoided. Through the aforementioned tools we successfully reduced unnecessary type instabilities, however a partial amount was retained in order to preserve a certain degree of flexibility, e.g. allowing to select at runtime which cohesion and similarity functions to use.

Regarding memory issues, we deeply inspected the code performance using the `ProfileCanvas` package. This profiler generates a flame graph, illustrated in Figure 2.1, which represents different sections of code with regions whose size is proportional to specific metrics, e.g. execution time or, in this case, the amount of memory allocations required for each section. The flame graph is read from left to right regarding the order of execution of the different sections, and from top to bottom according to the stack trace, meaning the order of subsequent function calls originating from the initial top one. This visualization enables to detect whether execution time is spent on productive operations or rather on less efficient activities such as garbage collection and runtime evaluations, thereby highlighting the portions of code that could possibly be optimized. All the modelling and working variables were preallocated; however, the key strategy has been to refactor the code to operate more in-place. This involved passing as arguments the variables that would be modified by a function, and applying those changes directly

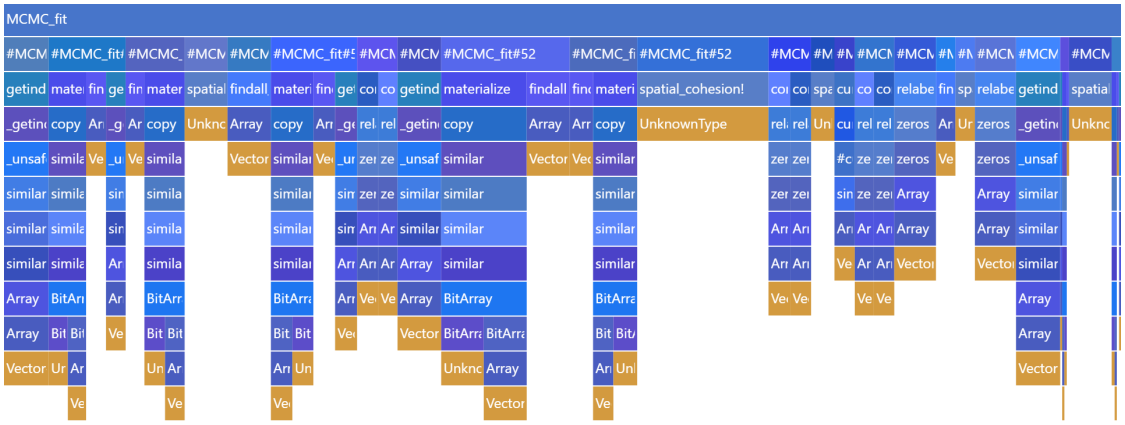


Figure 2.1: Flame graph derived from an example JDRPM fit with $n = 20$ and $T = 50$, ran for 10000 iterates.

within the function, rather than returning values and subsequently using them to update the original variables. Additionally, we implemented other straightforward improvements, thanks to the Julia language functionalities, such as the `@view` macro. This macro allows to eliminate unnecessary copies of vectors and matrices by passing references to the specific slices needed for a certain computation, rather than passing the entire structures. For example, when only the first row of a matrix `M` needs to be passed to a function, using `@view M[1,:]` instead of `M[1,:]` prevents a memory allocation.

Another valuable package to conduct performance analyses is `BenchmarkTools` (Chen et al., 2016), which allows to compare entire functions as well as specific portions of code. Regarding the latter, this package facilitates straightforward testing of different versions of equivalent instructions to determine which is the most efficient, as in this case, where we are computing the current number of clusters by counting the non-zero entries of the `nh_tmp` variable

```
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, where we are retrieving the indexes of the units assigned to cluster k

```
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)
```

The `$` symbol is employed for interpolating a variable, ensuring to treat the that variable as a local variable within the benchmark; an approach that eliminates any performance bias that might arise from accessing a global variable.

During the final stages of code refinement, we also exploited low-level tools such as the `--track-allocation` option, which asks Julia to execute the code while also annotating the source file, line by line, to indicate where allocations occurred and of which amount.

These tools collectively contributed to achieving an optimal level of performance. Table 2.1 provides a final reference for the effectiveness of these optimizations.

Table 2.1: Performance analysis of JDRPM's MCMC algorithm implementations, comparing an early stage code to the final version. The metrics are derived from an example fit ran for 1000 iterations, using a dataset with $n = 50$ and $T = 20$. The "% gc time" metric represents the percentage of execution time spent in garbage collection operations.

	execution time	memory allocated	% gc time
early stage version	17.314s	18.233 GiB	9.22%
final version	**4.515s**	**2.236 GiB**	**4.64%**

2.1.1 Optimizing spatial cohesions

The issue of memory allocation was particularly pronounced in the computation of spatial cohesions. This computation occurs in both the update steps of γ_{it} and ρ_t, which are nested within outer loops on draws, time, and units, and also involve additional loops over clusters. Consequently, these cohesion computations would potentially be executed millions of times during each fitting process. A simple inspection of the MCMC algorithm suggests that the computational complexity lies between dTn and dTn^2, where d represents the number of iterations, T the time horizon, and n the number of units. A more precise estimate is not possible due to the variability and randomness inherent in the inner loops which depend on the distribution of the clusters. Given this context, optimizing the performance of the cohesion functions was crucial to provide fast execution times.

The optimization efforts focused on carefully designing the implementations of the cohesion functions. Cohesions 1, 2, 5, and 6 did not present any complications or need for further optimization: the natural conversion into code from their mathematical models was already optimally performing. The main challenges emerged with cohesions 3 and 4, the auxiliary and double dippery, which inherently involve linear algebra operations with vectors and matrices, thereby increasing computational demands. The initial implementation of these cohesions was notably slow due to the overhead associated with allocating and freeing the memory of vectors and matrices at each call. As a result, a significant portion of execution time was consumed by the garbage collector rather than by useful computations. A preliminary solution involved resorting to a scalar implementation, which processed each component individually rather than operating on entire vectors and matrices. This approach effectively eliminated the overhead associated with the more complex memory structures. Ultimately, we succeeded in combining the readability of the first method with the efficiency of the second into a final, refined solution. Listing 1 showcases the logic behind the three different solutions.

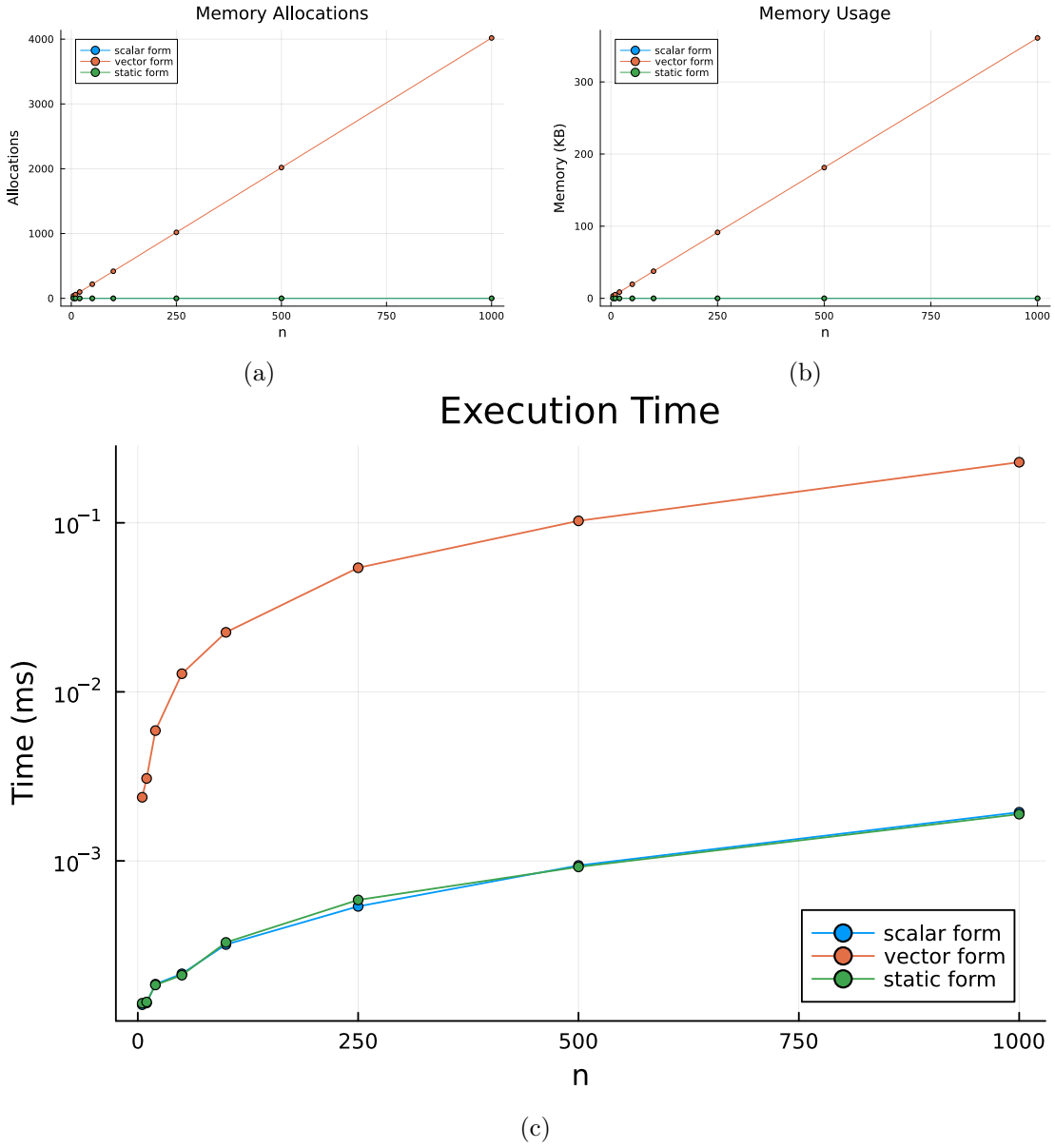
Listing 1: Snippets of Julia code for the three implementation solutions of the spatial cohesions 3 and 4. The first is the naive vector implementation, the second is its scalar conversion, and the third is the static vector solution.

```
# original vector version
sbar = [mean(s1), mean(s2)] # this is a standard vector
# all the following matrices and vector will be standard structures
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)) # this is statically-sized vector
# all the following matrices and vector will be statically-sized
vtmp = sbar .- mu_0
Mtmp = vtmp * vtmp'
aux1 = k0 * sdim; aux2 = k0 + sdim
Psi_n = Psi .+ S .+ aux1 / (aux2) .* Mtmp
```

This final version employs the `StaticArrays` package of Julia, which enables more efficient use of vectors and matrices when their sizes are known at compile time. This is suited for the spatial cohesion computations as we consistently work with 2×1 vectors and 2×2 matrices, due to the context of planar spatial coordinates. The benefits of this final implementation include preserving the natural mathematical form of the first solution, thus enhancing code clarity, while also capitalizing on the performance improvements seen in the second solution. In fact, with static structures, the compiler is able to optimize all memory allocations related to vectors and matrices just as it does with simple scalar variables.

Figure 2.2 provides a performance comparison of the three solutions. Panel 2.2c proves that the scalar and static versions exhibited similar performance, and both significantly outpaced the initial vector implementation, and both are significantly faster than the initial vector implementation. Additionally, panels 2.2a and 2.2b highlight the substantial reduction in memory requirements compared to the first solution. This comparison was conducted by running the three implementations against multiple sets of spatial coordinates with a varying number of units n.

Notably, the CDRPM implementation of the corresponding MCMC algorithm was not required to consider these optimizations, since C does not natively support



(a) (b)

(c)

Figure 2.2: Performance comparison among the three versions of the cohesion 4 function. Similar results stand for cohesion 3. Panels (a) and (b) are constant at zero for both the scalar and static cases.

vectors and matrices and was therefore forced to the scalar implementation without much deliberation.

2.1.2 Optimizing covariates similarities

Another significant challenge that we faced consisted in optimizing the computation of the similarity functions. As the cohesion functions, the similarities would potentially be called millions of times, if not more, considering that multiple covariates can be incorporated in the prior and thus introducing an additional loop based on p, the number of included covariates. As in the case of the previous

analysis, many of the similarity functions did not exhibit a significant need for optimization. However, the fourth function, the auxiliary similarity function, required optimization due to the computational load associated with calculating the sum of the squares of the covariate values, as illustrated in Listing 2.

Listing 2: Similarity 4 function implementation, with all optimizing annotations. The performance analysis will just focus on that inside loop.

```
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 optimization strategy involved annotating the loop with several macros provided by Julia. They were the following:

- `@inbounds` eliminates array bounds checking within expressions. This allows the compiler to bypass these checks, thus saving execution time. This annotation is safe to use as long as we can guarantee that the code will not access elements outside the array bounds; otherwise undefined behavior may occur. In our case, the loop structure is simple and safe, so this assumption holds true.

- `@fastmath` executes a modified version of the expression that may invoke functions violating strict IEEE semantics[1]. For instance, using this macro could result in $(a + b) + c \neq a + (b + c)$, but only in highly pathological cases. Again, this is not an issue for our loop, which computes $\sum X_i^2$, as there is no intrinsic "correct order" for performing this operation.

- `@simd` (Single Instruction Multiple Data) allows the compiler to apply further optimizations to the loop. This technique is akin to parallelism; however, rather than distributing the computational load across multiple processors, `@simd` vectorizes the loop. This means that the CPU can execute the same instruction (summing the square of the i-th component into a reduction variable) on multiple data chunks simultaneously using vector registers. As a result, this approach accelerates computation by eliminating the need to process each element of the vector individually.

[1]Institute of Electrical and Electronics Engineers. The IEEE-754 standard specifies floating-point formats, which dictate how real numbers are represented in hardware, along with the expected behavior of arithmetic operations on them, including precision, rounding, and the handling of special values (e.g., NaN (Not a Number) and infinity).

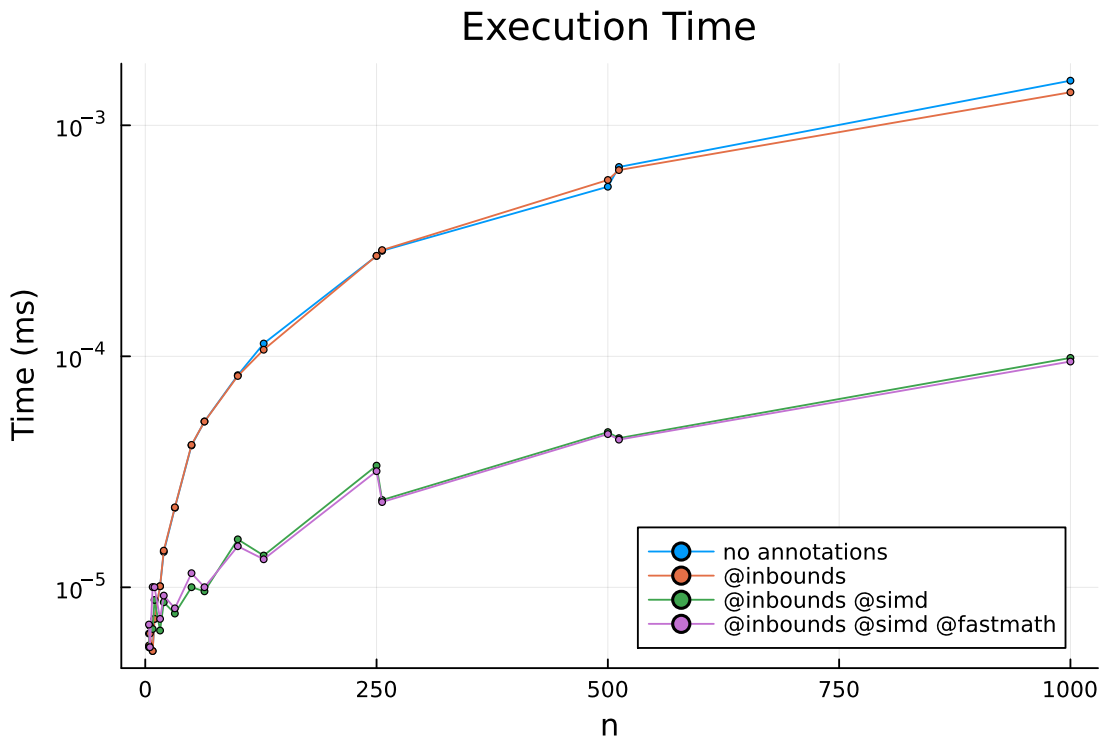


Figure 2.3: Performance comparison of the different loop annotations in the similarity 4 implementation.

As illustrated in Figure 2.3, the observed performance difference primarily arises from the use of `@simd`, while the other two annotations have minimal impact. Consequently, to address concerns from pure mathematicians, we opted to remove the `@fastmath` annotation, retaining only `@inbounds` and `@simd`. Memory allocation and usage panels are not reported since the analysis focused solely on evaluating the performance of the inner loop, which does not present any memory-related issues.

Moreover, we observe that experiments employing the `@simd` annotation tend to run faster when n is a power of two compared to n being its nearest rounded integers (for instance, 256 versus 250 or 512 versus 500), even though this configuration involves processing a relatively larger dataset. This pattern underscores the effectiveness of the SIMD approach: nding on the architectures, CPUs can support various register sizes (e.g. 64, 128, 256, or 512 bits). When the total memory occupied by the data points aligns perfectly with these register sizes, i.e. when the number of elements is a power of two, the data can be efficiently loaded into registers without any waste. Conversely, if the data size does not match, there will be "leftover chunks" that still need to be processed, which can introduce some overhead due to the inefficiencies associated with this imperfect fit.

Chapter 3

Analysis of CDRPM and JDRPM

In the following analyses, we will make use of the Adjusted Rand Index (ARI) (Hubert et al., 1985) to compare the partitions generated by the models. The ARI index serves as a correlation metric that quantifies the similarity between two clusterings. Specifically, for two partitions ρ_1 and ρ_2, the function $\text{ARI}(\rho_1, \rho_2)$ produces a value within the range $[-1, 1]$ where higher values indicating greater agreement between the partitions. A perfect match $\rho_1 = \rho_2$ is represented with the limit case $\text{ARI}(\rho_1, \rho_2) = 1$.

We will employ this index to analyse the temporal evolution of the partitions, examining whether ρ_{t+k} correlates with ρ_t, and to evaluate the level of agreement between the two models, by comparing clusters estimates generated by CDRPM and JDRPM. These cluster estimates will be computed using the `salso` function, with the binder loss, using the associated `salso` library (David B. Dahl et al., 2022) on R.

All analyses of this Chapter were conducted on a laptop equipped with 8 GB of RAM and a 1.80 GHz CPU base clock speed. The software used was R (R Core Team, 2024), interfaced with Julia through the `JuliaConnectoR` library (Lenz et al., 2022). This library handled all the communication between R and Julia, where the JDRPM algorithm is implemented. The CDRPM model is also accessible from R via a dedicated package, `drpm`, which similarly employs a wrapper to invoke the C code where the algorithm is implemented.

3.1 Assessing the equivalence of the models

Our model, along with its corresponding Julia implementation, represents an enhancement over the original DRPM and its associated C implementation. The improvements, as outlined in previous chapters, include the ability to incorporate covariates into both the prior and likelihood levels of the Bayesian model, the possibility of allowing for missing data in the response variable, and the guarantee of greater computational efficiency. In this regard, our updates serve as extensions to the original model. Therefore, when tested under equivalent hyperparameters and MCMC configurations, both models are expected to perform similarly and

produce comparable clusters estimates for a given dataset.

To evaluate the numerical performance of both algorithms, we will analyse posterior samples and cluster estimates in two scenarios: firstly using a synthetic dataset that includes only the response variable, and secondly employing a real-world spatio-temporal dataset. The latter also provides covariates; however, their effects will be extensively examined in the dedicated Section 3.3.

3.1.1 On a synthetic dataset

For the initial comparison we generated a dataset of $n = 10$ units and $T = 12$ time instants. The data generating function was the same employed in (G. L. Page et al., 2022) for their analyses and it allows for the creation of data points with temporal dependence, which could be adjusted through specific parameters. Regarding the MCMC setup, both algorithms were executed deriving 2000 iterates from a total of 50000 iterations, by discarding the first 40000 as burnin and then thinning by 5. Both models were fitted using a time specific α and using their full formulations, i.e. including and updating also the optional autoregression parameters η_{1i} and φ_1.

Table 3.1: Comparison between CDRPM and JDRPM fits and their associated algorithms, in the simulated data scenario.

	MSE mean	MSE median	execution time
CDRPM	1.6221	1.5823	19s
JDRPM	**1.2634**	**1.2034**	**13s**

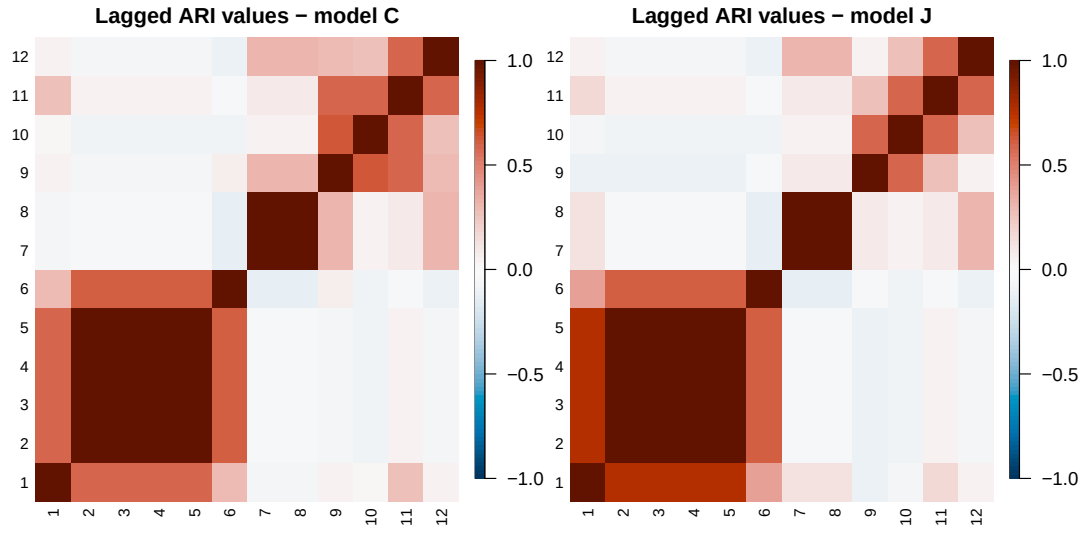


Figure 3.1: Lagged ARI values of CDRPM and JDRPM fits, in the simulated data scenario.

During this testing phase, the clusters were defined solely by the target values from Y_{it}, and both models achieved satisfactory results. Fitted values are presented in Figure 3.4 alongside the original data. From Table 3.1 we can see how JDRPM

exhibited greater accuracy, as proven by the lower mean squared error (MSE), and a faster execution time. In the table, the MSEs were computed by comparing the fitted values generated by the models, estimated by taking the the mean and median of the 2000 iterations, with the true values of the target variable. We do not report fitting metrics for WAIC and LPML since both models were conceptually equivalent and tested under identical hyperparameters and MCMC configurations. Consequently, any observed differences in these metrics would be likely attributable to chance.

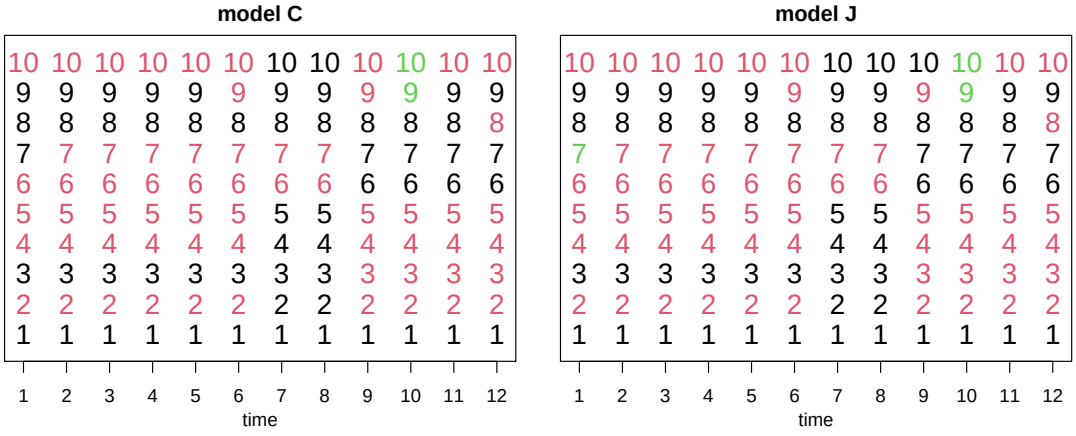


Figure 3.2: Clusters estimates produced by CDRPM and JDRPM fits, in the simulated data scenario. Time instants are annotated on the x axis, units are indicated vertically by their number, and colors represent the cluster label.

The resulting partitions were remarkably similar. As illustrated in Figure 3.1, both models effectively captured the temporal dynamics of the response variable. Moreover, Figure 3.2 confirms the agreement in the cluster estimates; however, it also reveals two minor differences which we will now discuss.

A closer examination of the clusters presented in Figure 3.3 reveals interesting distinctions at times $t = 1$ and $t = 9$. At $t = 1$, JDRPM classified unit 7 as a singleton within the green cluster, whereas CDRPM assigned unit 7 to the black cluster. Both classifications are reasonable: the data point is indeed closer to the black cluster, but it later aligns more distinctly to the red cluster; suggesting the classification given by JDRPM as a detection of its initial anomalous behaviour. At $t = 10$, both models successfully identified a small, temporary third cluster resulting from the transition of units 9 and 10. The JDRPM interprets this transition as a label swap: between times $t = 9$ and $t = 11$, unit 9 shifts from the red cluster to the black cluster, while unit 10 from the black cluster to red cluster. In contrast, at $t = 9$ CDRPM assigns both units to the red cluster, potentially reflecting a misclassification considering the temporal trend of unit 10 which, until $t = 10$, indicated a closer alignment with the black cluster.

3.1.2 On spatio-temporal data

In this section we examine a real-world application by fitting CDRPM and JDRPM on a spatio-temporal dataset. Specifically, we used the AgrImOnIA dataset

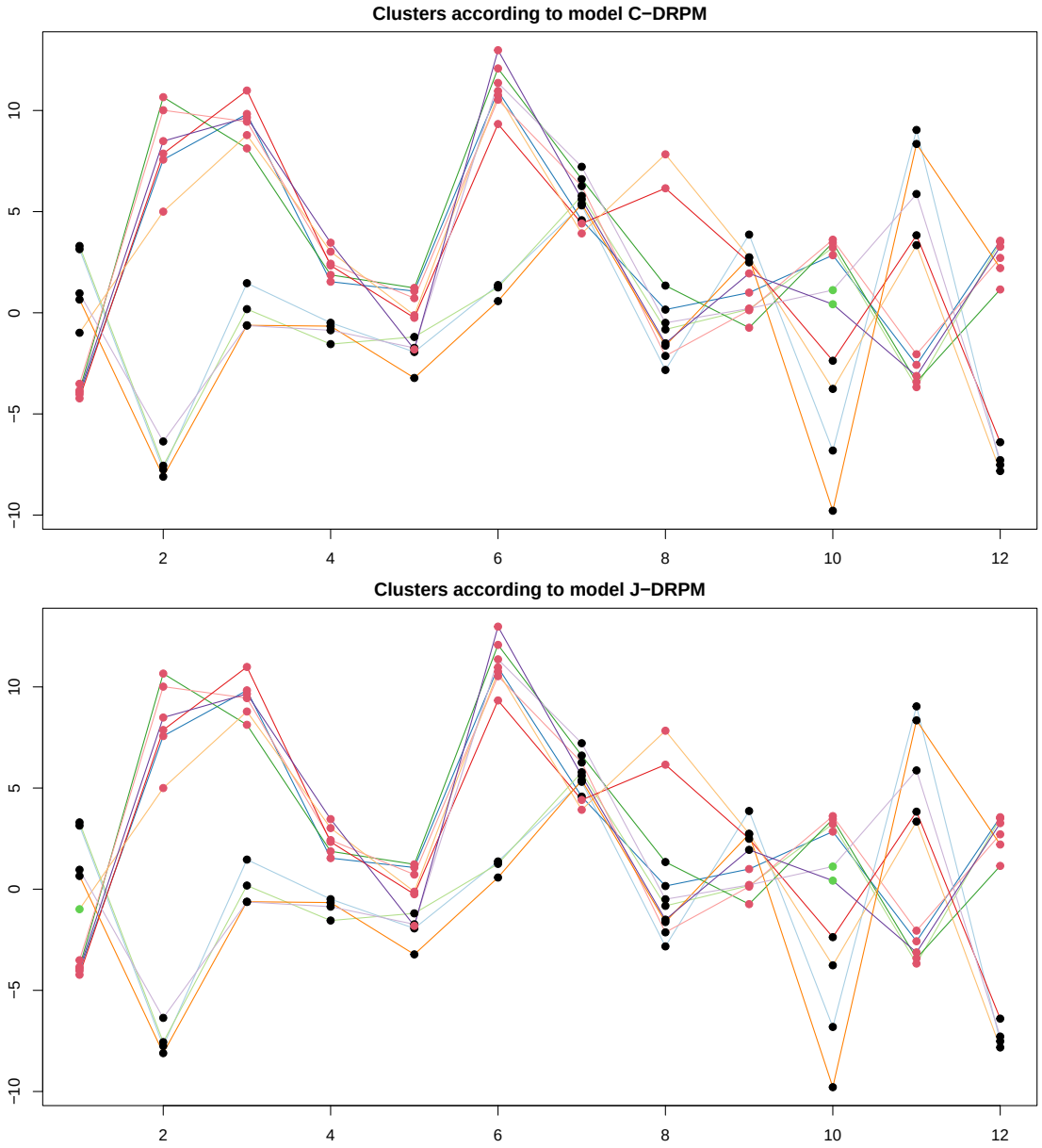


Figure 3.3: Visual representation of the clusters estimates produced by CDRPM and JDRPM fits, in the simulated data scenario. Cluster labels are represented as colored dots overlaid to the trend of the target variable.

(Fassò et al., 2023) which encompasses measurements of air pollutants, together with many other environmental variables, in the Lombardy region of Italy from 2016 to 2021.

For our subsequent analyses, we employed a dataset comprising weekly averages from the year 2018. The target variable chosen for these studies was PM_{10}, which represents the concentration of particulate matter with a diameter of less than 10 µm. This variable underwent log-transformation, to recover a normal distribution, and was centered with respect to time-wise means. More precisely, each observation Y_{it} was adjusted by subtracting the mean $\bar{Y}_t$ of all observations at that time instant t. This approach, consistent with the methodology employed by (G. L. Page et al.,

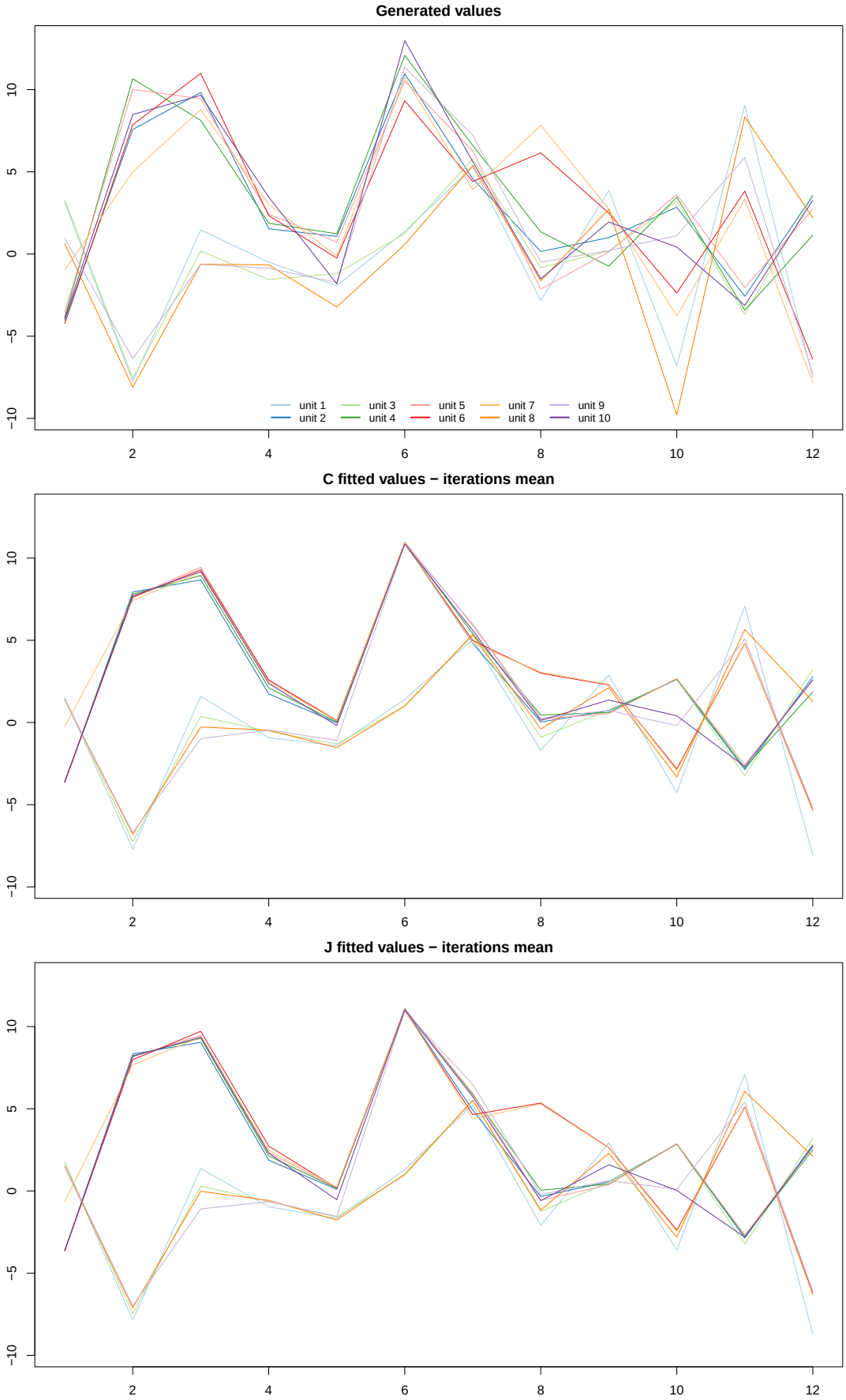


Figure 3.4: Fitted values of CDRPM (middle) and JDRPM (bottom) fits, in the simulated data scenario, alongside the generated target values (top).

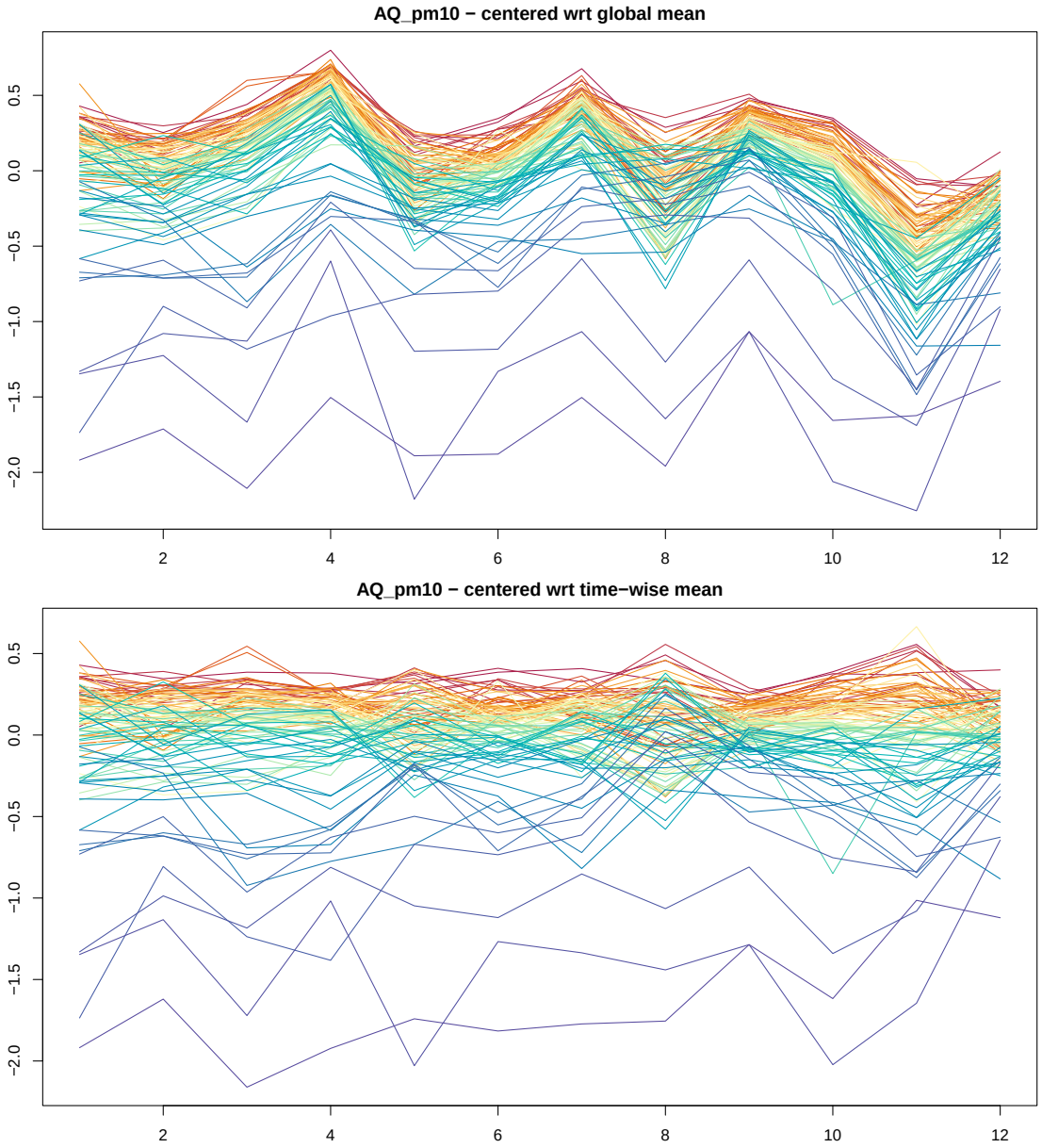


Figure 3.5: Values of the target variable `AQ_pm10` adjusted using the global mean (top) and the time-wise mean (bottom). Coloring is based on the ranking of PM_{10} values of the units according to their median, from highest (red) to lowest (blue).

2022) during their analyses, helps to emphasize variations within each time point rather than across the entire dataset. This approach is particularly beneficial for understanding how individual units deviate from their typical behavior at specific times, thereby highlighting temporal trends and anomalies. Moreover, this method effectively mitigates any temporal bias that may arise from periods in which all target values are disproportionately high or low due to external anomalous factors. In contrast, the traditional approach of centering the data around their global mean $\bar{Y}$ would primarily facilitate the detection of overall trends, without providing an elaborate examination of each time step. For instance, preliminary analyses using global centering revealed only three clusters across all time points. While this

result is indeed valid in light of the trend illustrated in Figure 3.5, our alternative approach produced multiple clusters, thereby enhancing the specialization of the clusters and also their interpretability.

To perform the fit, we used all the available stations in the dataset, amounting to $n = 105$, but restricted the time horizon to $T = 12$, corresponding to a three-month monitoring period. This limitation was implemented solely to reduce the computational time required to conduct the analyses. As in the previous section, both CDRPM and JDRPM algorithms were fitted with a time-specific parameter α and using their complete model formulations including the optional parameters η_{1i} and φ_1. Regarding the spatial cohesion, we selected C_3, the auxiliary similarity function. To ensure convergence, we inspected the trace plots of both model and, in the case of JDRPM, we checked numerical diagnostics as the Effective Sample Size (ESS) and $\hat{R}$. In fact, JDRPM is able to provide such statistical assessments directly from the fitting function. Regarding the MCMC setup, we derived 4000 iterates from 110000 total iterations, by discarding the first 90000 as burnin and then thinning by 5.

Table 3.2: Comparison between CDRPM and JDRPM fits and their associated algorithms, in the real-world scenario.

	MSE mean	MSE median	execution time
CDRPM	0.0142	0.0149	1h38m
JDRPM	**0.0131**	**0.0138**	**48m**

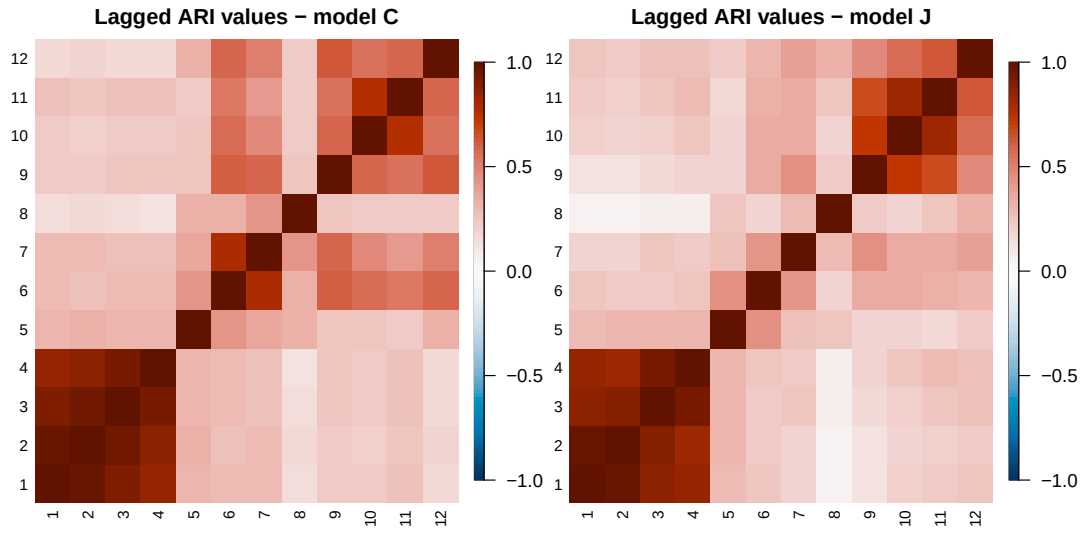


Figure 3.6: Lagged ARI values of CDRPM and JDRPM fits, in the real-world scenario.

Table 3.2 demonstrates that both models achieved remarkable accuracy in their fitted values, with JDRPM leading over CDRPM in terms of MSEs. As previously mentioned, we have chosen not to report the fitting metrics for WAIC and LPML due to the theoretical equivalence of the evaluated models and their corresponding MCMC algorithms. While execution times are included, they may not fully reflect

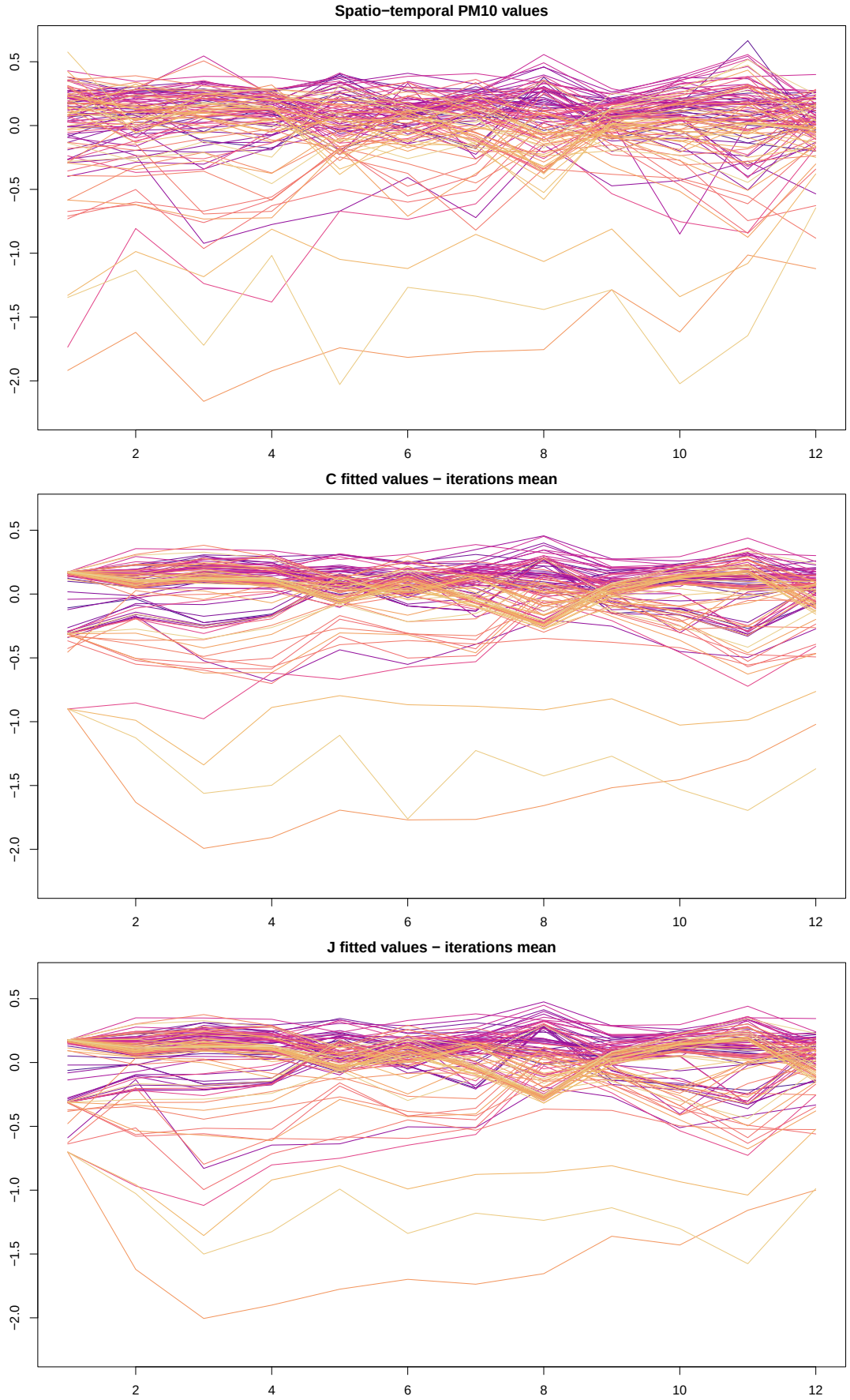


Figure 3.7: Fitted values of CDRPM (middle) and JDRPM (bottom) fits, in the real-world scenario, alongside the generated target values (top).

the actual performance, as I was concurrently engaged in writing this thesis, which limited my laptop's computational resources during the fitting process. A more precise assessment of performance timings will be presented in Section 3.4.

Figure 3.6 reveals a similar temporal trend, further validating the correct implementation of JDRPM's MCMC algorithm. In terms of clusters similarity, a partition plot as in Figure 3.2 would appear more congested due to the increased number of units. To effectively convey this information, we computed the adjusted Rand index $\text{ARI}(\rho_{\text{JDRPM}}(t), \rho_{\text{CDRPM}}(t))$ for all time points $t = 1, \dots, 12$. The results yielded a mean of 0.80 and a median of 0.86, signifying a strong agreement between the clusters estimates generated by the two models.

A visual representation of the clusters estimates generated by the two models is provided in Figures C.1 and C.2. Nevertheless, a more comprehensive analysis and discussion will take place in Section 3.3.2, where we will also examine fits with the effects of covariates in the prior and likelihood levels.

3.2 Performance with missing values

In this section, we replicate the analyses and evaluations from Section 3.1, this time focusing on scenarios involving missing values. Our objective is to investigate how the JDRPM performs in the absence of complete datasets and to determine whether it maintains effective performance under such conditions.

Given the extent of missing values in the AgrImOnIA dataset (Fassò et al., 2023), which was used for the spatio-temporal analysis, we opted to set 10% of the values as missing (NAs). To implement this, we randomly selected $nT/10$ indexes from the sets $[1, \dots, n]$ and $[1, \dots, T]$ to identify all the pairs (i, t) that would be designated as missing in the target variable Y_{it}. The ability to handle missing data is an enhancement introduced by the JDRPM; therefore these studies cannot be repeated with the original CDRPM model, which does not accept incomplete datasets.

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

3.2.1 No spatial information

The JDRPM model demonstrates remarkable performance even in the presence of missing values. As shown in Table 3.3, the model's performance is understandably lower compared to the full dataset scenario; however, it clearly remains satisfactory. The MSE has naturally increased due to the partial absence of data points which led to less accurate fitted values for the corresponding missing entries. Nevertheless, the fitted values, illustrated in Figure 3.8, remain closely aligned with the ones derived from the full dataset analysis.

From Figures 3.10, we observe a nearly identical temporal trend to that seen

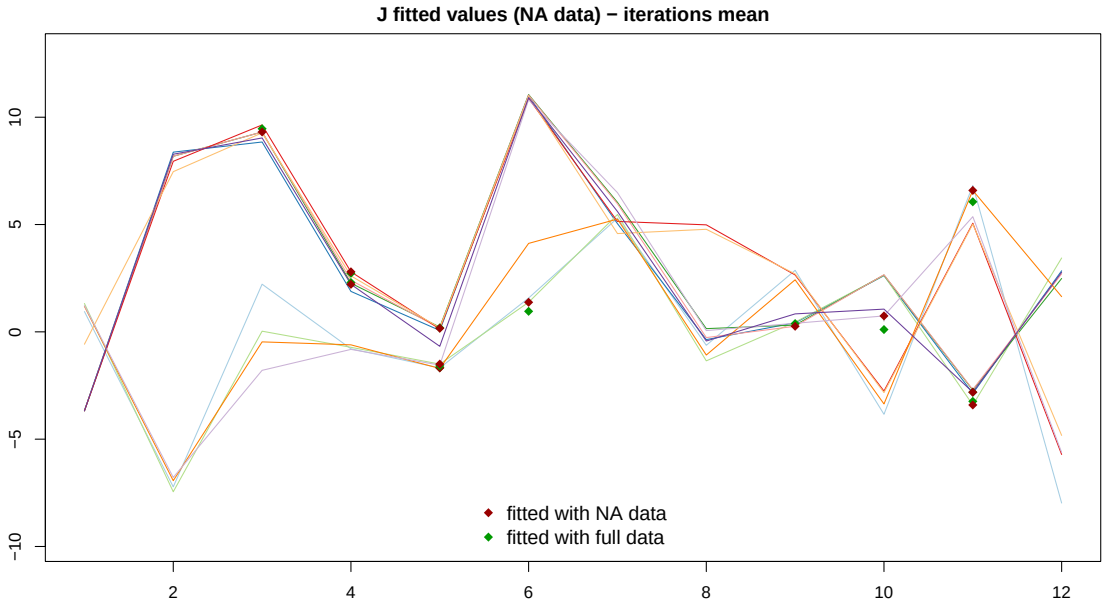


Figure 3.8: Fitted values of JDRPM fit, in the simulated data scenario, with missing values in the target variable. Special square markers are devoted to the data points which were missing, highlighting the gaps between the fitted values on the full dataset (green) and the fitted values on the dataset with missing values (red).

Table 3.3: Comparison of JDRPM fits, in the simulated data scenario, on a complete dataset and on a dataset with missing values.

	MSE mean	MSE median	LPML	WAIC	exec. time
NA data	1.4721	1.2101	-236.93	401.44	13s
full data	1.2634	1.2034	-223.36	393.97	13s

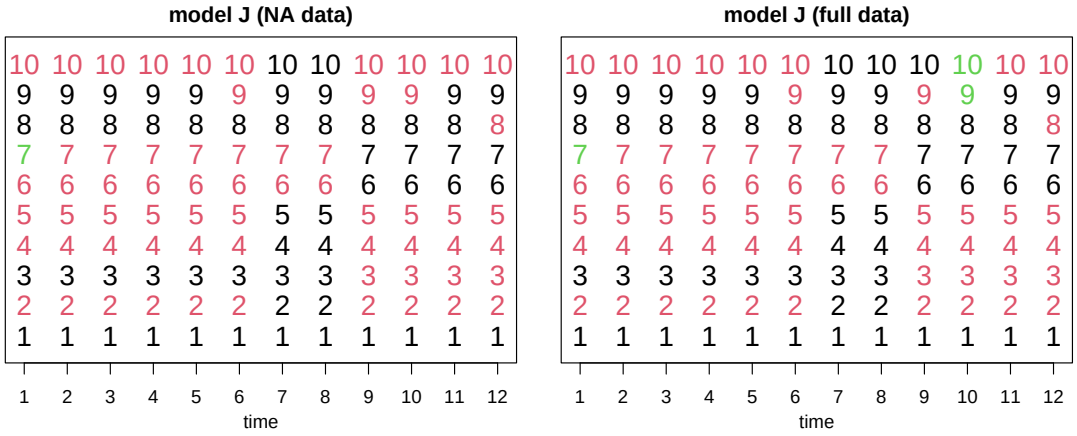


Figure 3.9: Clusters estimates produced by JDRPM fits, in the simulated data scenario, on a dataset with missing values. Time instants are annotated on the x axis, units are indicated vertically by their number, and colors represent the cluster label.

in the absence of missing data, indicating that JDRPM effectively captured a robust temporal dependency structure even when some data points were missing. Additionally, the clusters generated, as shown in Figure 3.11, appear remarkably

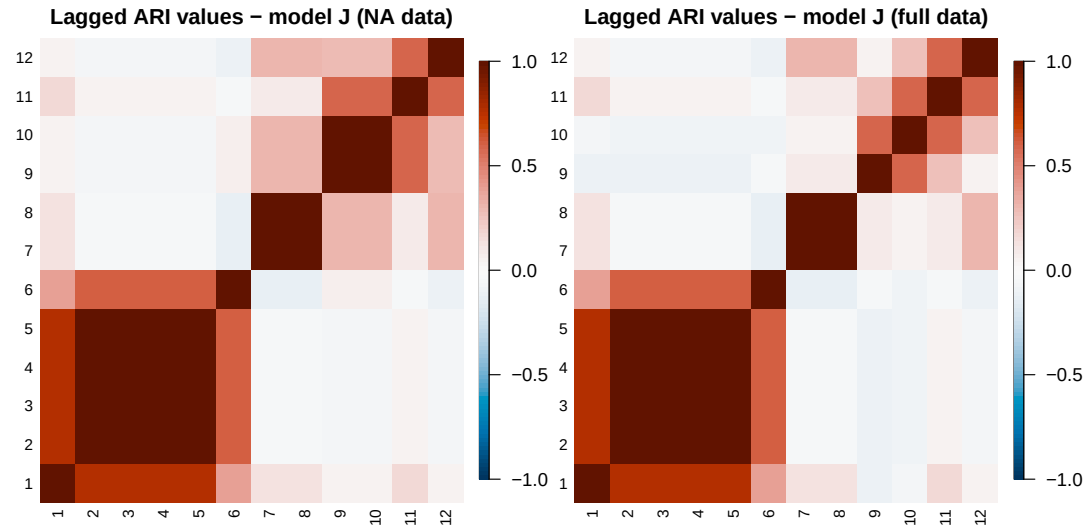


Figure 3.10: Lagged ARI values of JDRPM fits, in the simulated data scenario, on a dataset with missing values.

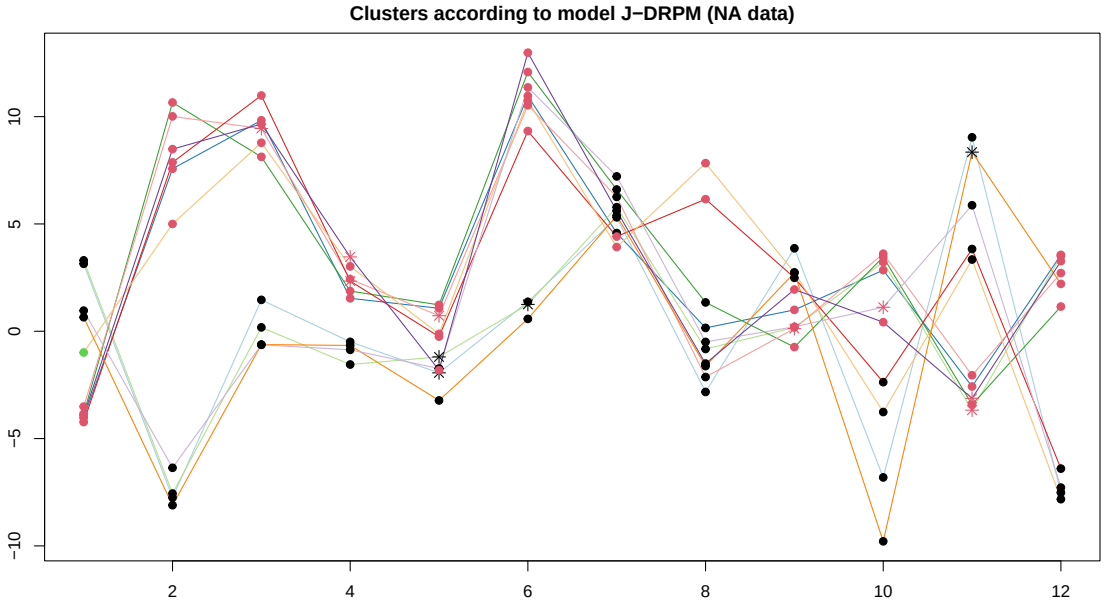


Figure 3.11: Visual representation of the clusters estimates produced by JDRPM fit, in the simulated data scenario, on a dataset with missing values. Cluster labels are represented as colored dots overlaid to the trend of the target variable, with special point markers devoted to data points corresponding to missing values.

similar. The only notable difference occurred around time $t = 10$, where the model now failed to identify the green cluster that characterized the final transition phase of units 9 and 10. This discrepancy may be attributed to the NA assignment of unit 9 at time 10, which likely removed critical information necessary for identifying that specific third cluster.

Finally, Figure 3.12 presents the 95% credible intervals for the fitted values corresponding to the missing data points. In nearly all cases, except for one, the

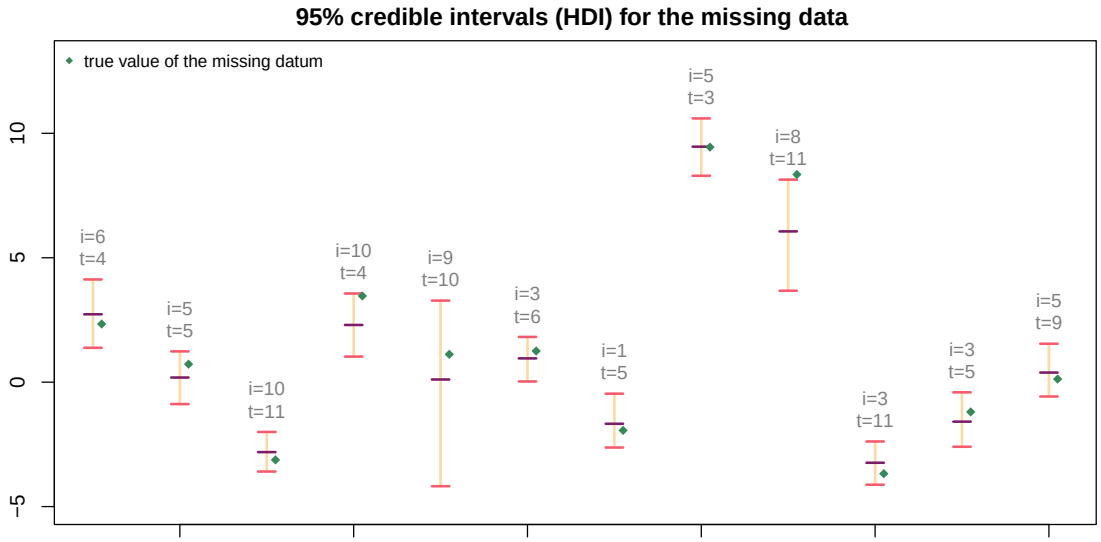


Figure 3.12: Credible intervals, computed with the highest posterior density (HPD) method at a 95% confidence, for the fitted values of the missing units in the JDRPM fit, in the simulated data scenario, on a dataset with missing values. In gray are reported the indexes of units i and time instants t which where missing, while green dots refer to the true values of the missing data.

true value falls within the credible interval, demonstrating the accuracy of JDRPM in providing reliable estimates of the target values, even in situations where no additional information was available from either space or covariates.

3.2.2 With spatial information

Table 3.4: Comparison of JDRPM fits, in the real-world scenario, on a complete dataset and on a dataset with missing values.

	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 demonstrates exhibits robust performance also in real-world scenario involving a dataset with missing values. Table 3.4 indicates a slight reduction in accuracy, which is expected due to the presence of missing data; however, the decline is not substantial. The fitted values are displayed in Figure 3.13, revealing how the estimation of the missing data becomes more challenging for the points that lie farther away from the main trajectory of the distribution. It is worth noting that the reported execution times may not fully reflect reality, as previously mentioned, due to my laptop's resources not being entirely devoted to the fitting process. In fact, the JDRPM fit with missing data ran faster than the JDRPM fit with complete data, which raises some eyebrows since we should typically expect additional computational demands when dealing with missing Y_{it} values due to their sampling in the MCMC algorithm. Again, for more accurate performance assessments we refer to Section 3.4.

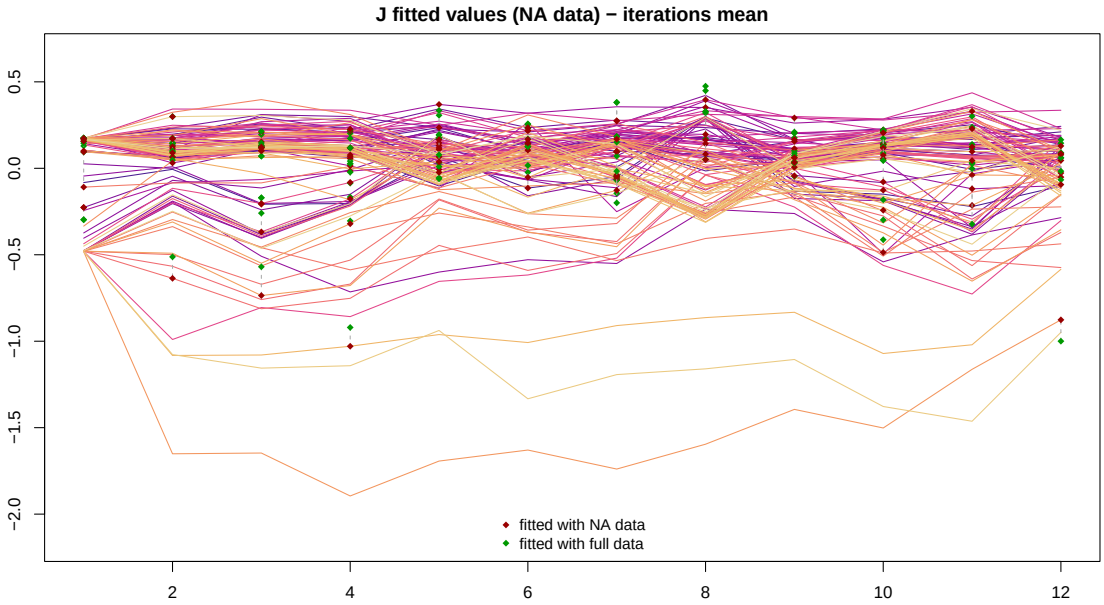


Figure 3.13: Fitted values of JDRPM fit, in the real-world scenario, on a dataset with missing values. Special square markers are devoted to the data points which were missing, highlighting the gaps between the fitted values on the full dataset (green) and the fitted values on the dataset with missing values (red).

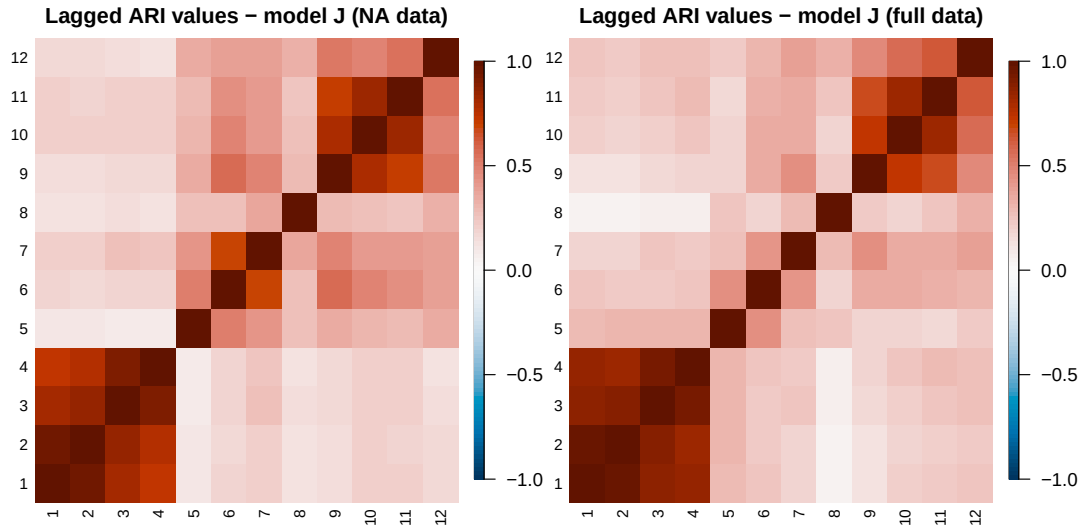


Figure 3.14: Lagged ARI values of JDRPM fits, in the real-world scenario, on a dataset with missing values.

The temporal trend remained consistent with what we observed in Section 3.1.2, as evidenced by Figure 3.14. Interestingly, the ARI plot for the JDRPM fit with missing data closely mirrors the original trend displayed by the CDRPM fit on the complete dataset; a similar pattern was also noted in the previous section regarding fits in the simulated data scenario. To precisely evaluate clusters similarity, we computed the adjusted Rand index $\mathrm{ARI}(\rho_{\mathrm{JDRPM\_NA}}(t), \rho_{\mathrm{JDRPM\_full}}(t))$ for each time point $t = 1, \ldots, 12$. The results yielded a mean of 0.82 and a median of 0.86, indicating a strong level of agreement in the partitions despite the loss of a

significant amount of data (121 points out of 1260 from the full dataset).

3.3 Effects of the covariates

We now conduct several experiments to explore the key advancement introduced by the JDRPM: the inclusion of covariates. Given their distinctly different purposes, we study separately the effects of including covariates in the likelihood and including covariates in the prior. Nevertheless, an analysis of the combined use of both information levels will be provided in Section 3.3.3. For the upcoming fits, all included covariates underwent the same time-wise correction applied to target variable, as outlined in Section 3.1.2.

3.3.1 Covariates in the likelihood

Following the discussion of the previous section, we can explore whether the inclusion of covariates in the likelihood can enhance accuracy in fits that deal with missing data. In fact, the main purpose of introducing the regression parameter $\boldsymbol{\beta}t$ was to improve the precision of the model's estimates for the target values Y_{it}, without significantly influencing the clusters generation. Therefore, the most natural application of this parameter arises when fitting models with missing values.

Table 3.5: Comparison of JDRPM fits, in the real-world scenario, with and without the inclusion of covariates in the likelihood, on a complete dataset and on a dataset with missing values.

	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

Indeed, after including multiple covariates in the likelihood and repeating the fit on the spatio-temporal dataset with missing values, we observed notable improvements. To effectively compare these enhanced results, we also performed the corresponding fit with covariates in the likelihood on the complete dataset, with the resulting cluster estimates displayed in Figure C.3. The results are summarized in Table 3.5, which also includes reference results from previous fits of Sections 3.1.2 and 3.2.2, which did not include the regressor component.

The inclusion of covariates in the likelihood clearly enhanced all fitting metrics. This demonstrates that incorporating covariates in the likelihood can effectively recover accuracy in the presence of missing values or to generally improve the estimates of the target variable Y_{it}. The regression vectors $\boldsymbol{\beta}_t$ exhibited favourable trace plots, as shown in Figures 3.16 and 3.17, confirming the correctness and effectiveness of the implementation. Additionally, the values of the regressors appear reasonable and interpretable. For instance, it is well established that

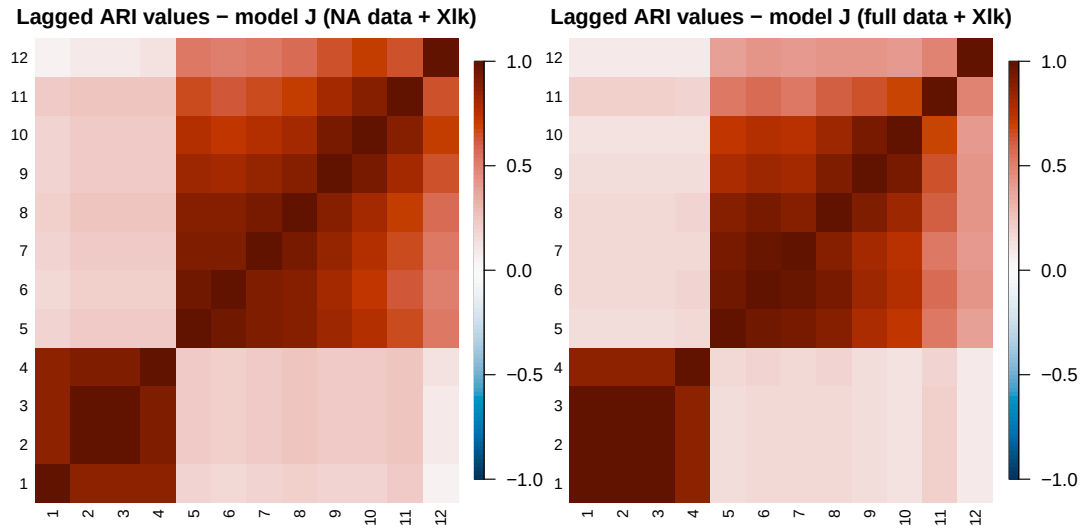


Figure 3.15: Lagged ARI values of JDRPM fits, in the real-world scenario, with covariates in the likelihood, on a dataset with missing values

higher altitudes correlate with lower air pollution levels, due to the scarcity of emission sources like industries and vehicles, stronger winds, and lower atmospheric pressure that facilitates air mixing. Consequently, the $\boldsymbol{\beta}_t$ component associated with `Altitude` hovers around negative values, indicating a reduction in PM_{10} concentrations. Conversely, the `LI_bovine` covariate, which represents the density of bovines per km^2 near the measuring station, tends to remain positive, reflecting the contribution of livestock industries to air pollutant emissions.

The generated partitions remained mostly unchanged. The spatio-temporal trend was similarly preserved, as shown in Figure 3.15, although a generally higher dependence was noted in the latter part of the time interval, after the correctly identified change point at $t = 4$.

The function which implements JDRPM's MCMC algorithm includes a parameter `beta_update_threshold`, defaulted to zero, that specifies the iteration after which the algorithm begins updating the $\boldsymbol{\beta}_t$ parameter. This addition, both harmless and straightforward, allows the model to prioritize the updating and refinement of more critical clustering parameters, such as $\mu_{jt}^{\star}$ and $\sigma_{jt}^{2\star}$, before turning its attention to updating the $\boldsymbol{\beta}_t$ parameter. Otherwise, without this adjustment, the early development of model parameters and cluster assignments could be skewed by inaccurate samples from the likelihood regressor, potentially compromising the overall performance of the model.

Figure 3.18 illustrates the fitted values from the JDRPM applied to the spatio-temporal dataset with missing values and incorporating covariates in likelihood. Visually, and as demonstrated by the improved MSEs, these fitted values more closely align with those of the real target variable. This is particularly evident when compared to the results in Figure 3.7, where both fits were performed without any "external suggestions" from covariates. Further evidence supporting the effectiveness of including covariates in the likelihood is provided in Figure 3.19, which displays

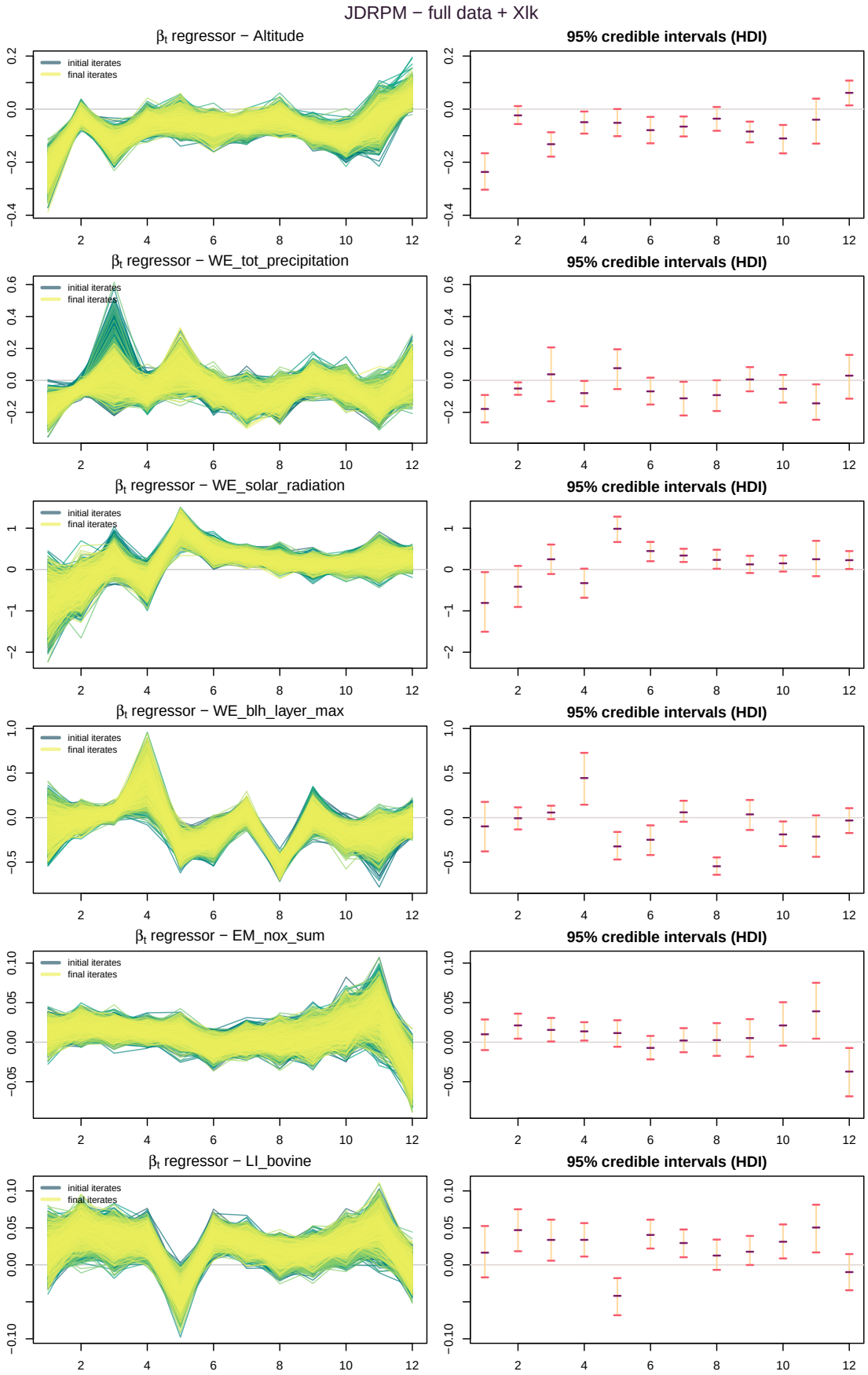


Figure 3.16: Regression vector $\boldsymbol{\beta}_t$ of JDRPM fit, in the real-world scenario, for the $p = 6$ covariates inserted in the likelihood, on the full spatio-temporal dataset, with trace plots (left) and 95% credible intervals (right) computed with the highest density interval (HDI) method.

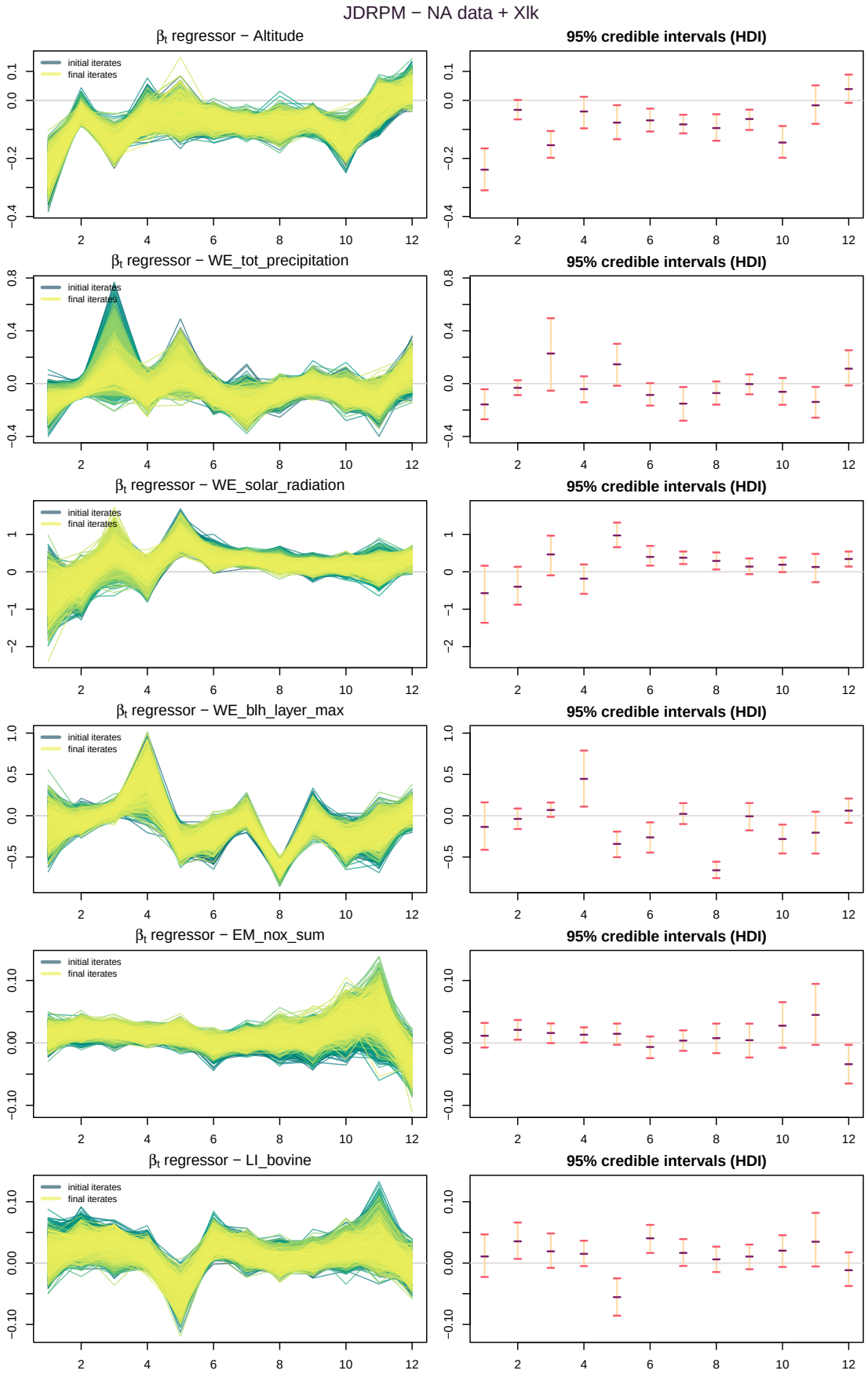


Figure 3.17: Regression vector $\boldsymbol{\beta}_t$ of JDRPM fit, in the real-world scenario, for the $p = 6$ covariates inserted in the likelihood, on the spatio-temporal dataset with missing values, with trace plots (left) and 95% credible intervals (right) computed with the highest density interval (HDI) method.

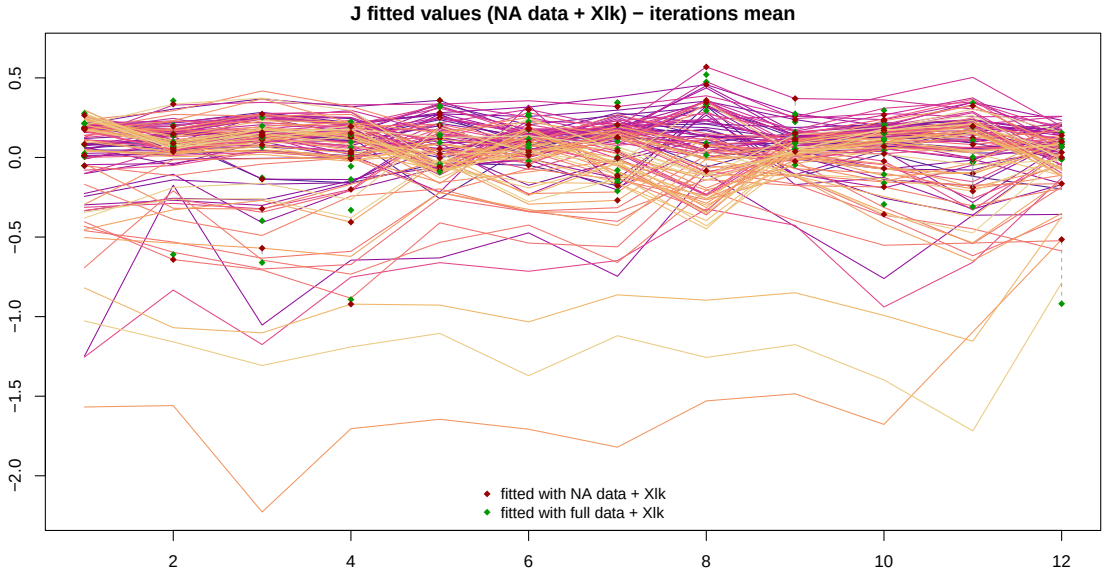


Figure 3.18: Target and fitted values of the JDRPM fits with target plus space values, on the NA and full dataset, to see the effects of the insertion of covariates in the likelihood.

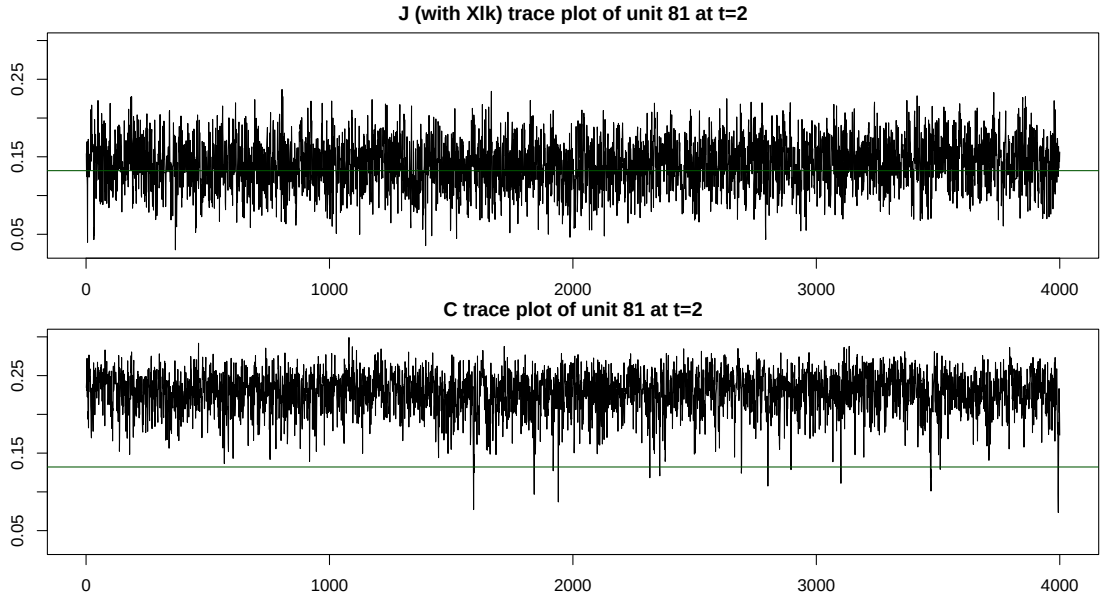


Figure 3.19: Trace plot of the fitted values of a specific unit i and time instant t, comparing the JDRPM fit with covariates in the likelihood to the standard spatially-informed CDRPM fit. The green line represents the true value of Y_{it}.

the trace plot of the fitted values for a problematic unit and time in which both the standard JDRPM and CDRPM fits struggled to deliver accurate estimates. However, with the inclusion of covariates, these estimates have become more precise and are now closer to the actual associated Y_{it} value.

Overall, this analysis highlights how the addition of the regression parameter $\boldsymbol{\beta}_t$ can improve the accuracy of the results and partially support for the clustering process, as the contribution from the covariates in improving the Y_{it} estimates becomes evident in refining the update steps for the clustering parameters. In fact,

as we saw in (1.6) and (1.7), the update rules of $\sigma_{jt}^{2\star}$ and $\mu_{jt}^{\star}$ are also influenced by a term involving $\boldsymbol{\beta}_t$.

3.3.2 Covariates in the prior

We now turn our attention to the inclusion of covariates in the prior. In deciding which covariates to incorporate, we focused on the factors that would most significantly influence PM_{10} concentrations. Ultimately, we selected the following three covariates:

- `WE_wind_speed_10m_max`. Wind speed is a key factor affecting air pollutant concentration levels. Its impact is twofold: on one hand, wind can disperse pollutants away from their sources, thereby reducing local concentrations; on the other hand, it can increase airborne contaminants by resuspending settled particles from surfaces like roads, soil, and buildings. This latter effect is particularly evident in dry and windy rural areas, making Lombardy region a relevant case study. The dataset also included wind speed at 100 m, a variable that however would be more suitable for broader analyses that track PM_{10} concentration trends across multiple regions or countries. In contrast, our approach focused on local dynamics and ground-level perspectives to cities in Lombardy, thus making the 10 m measurement more appropriate for our objectives.

- `WE_tot_precipitation`. Rain is well-known for its ability to enhance air quality by capturing aerosol particles and bringing them down to the ground. This process, often referred to as wet deposition, precipitation scavenging, or washout, is highly effective at reducing air pollutant concentrations. Consequently, this variable was considered highly relevant for inclusion in our analysis.

- `WE_blh_layer_max`. The boundary layer height (BLH) is another key factor influencing the dispersion of air contaminants. This variable defines the maximum altitude at which air mixing takes place. Typically, air cools as it rises in the atmosphere; however, there can be instances where warm air exists above colder air. At this boundary, mixing becomes inhibited, as the warm layer acts like a lid trapping the cooler air, and its associated pollutants, beneath it. This phenomenon leads to a deterioration in air quality, as pollution accumulates without a means of dispersal. Consequently, this covariate measures the maximum height of this problematic layer, with lower values indicating a higher expected concentration of pollutants due to restricted vertical mixing.

To conduct the JDRPM fit with covariates in the prior, we retained the same hyperparameters and MCMC configuration used in our earlier spatio-temporal experiments described in Sections 3.1.2 and 3.2.2. For the similarity function we opted for g_4, the auxiliary similarity function, setting its parameters to $\mu_0 = 0$, $\lambda_0 = 1$, $a_0 = 7.5$, and $b_0 = 2$. This selection was based on experimental adjustments

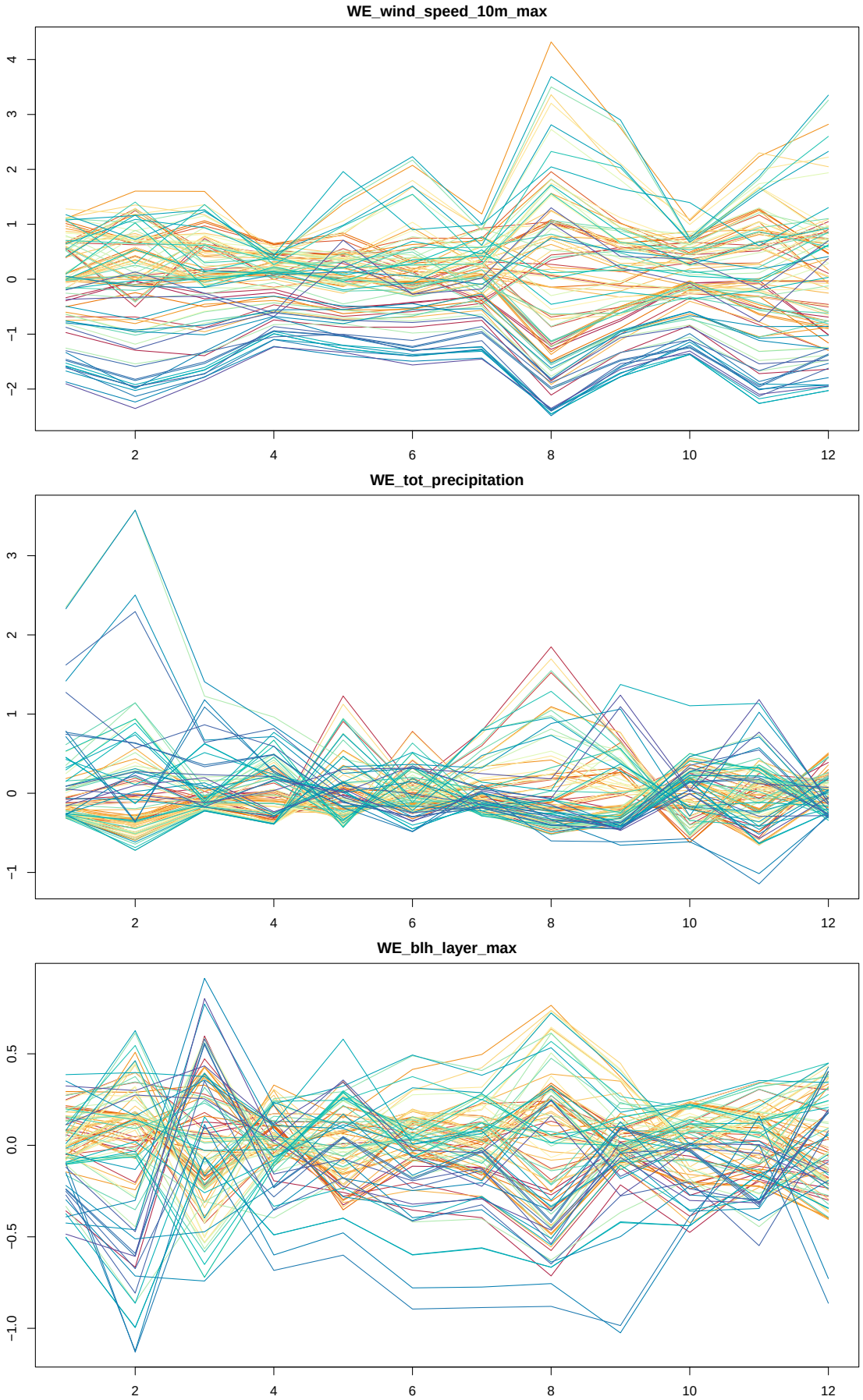


Figure 3.20: Covariates selected to be included in the prior. Coloring is based on the ranking of PM_{10} values of the units according to their median, from highest (red) to lowest (blue).

and especially on the standardization and centering process that was applied to covariates. In fact, as effectively illustrated in Figure 3.20, the trends of the covariates generally remained within the $[-1, 1]$ interval.

Table 3.6: Comparison between CDRPM and JDRPM fits and their associated algorithms, in the real-world scenario, with and without covariates in the prior.

	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

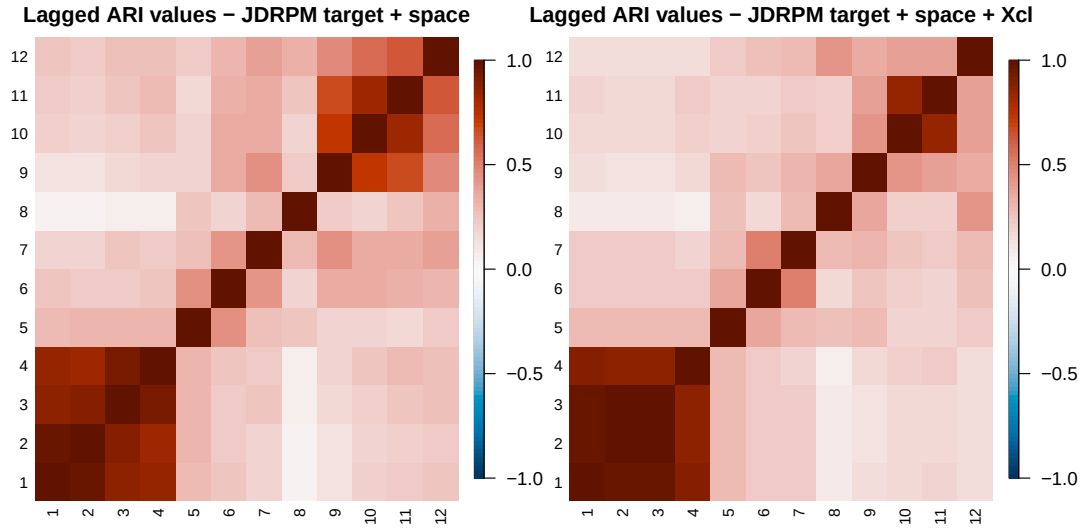
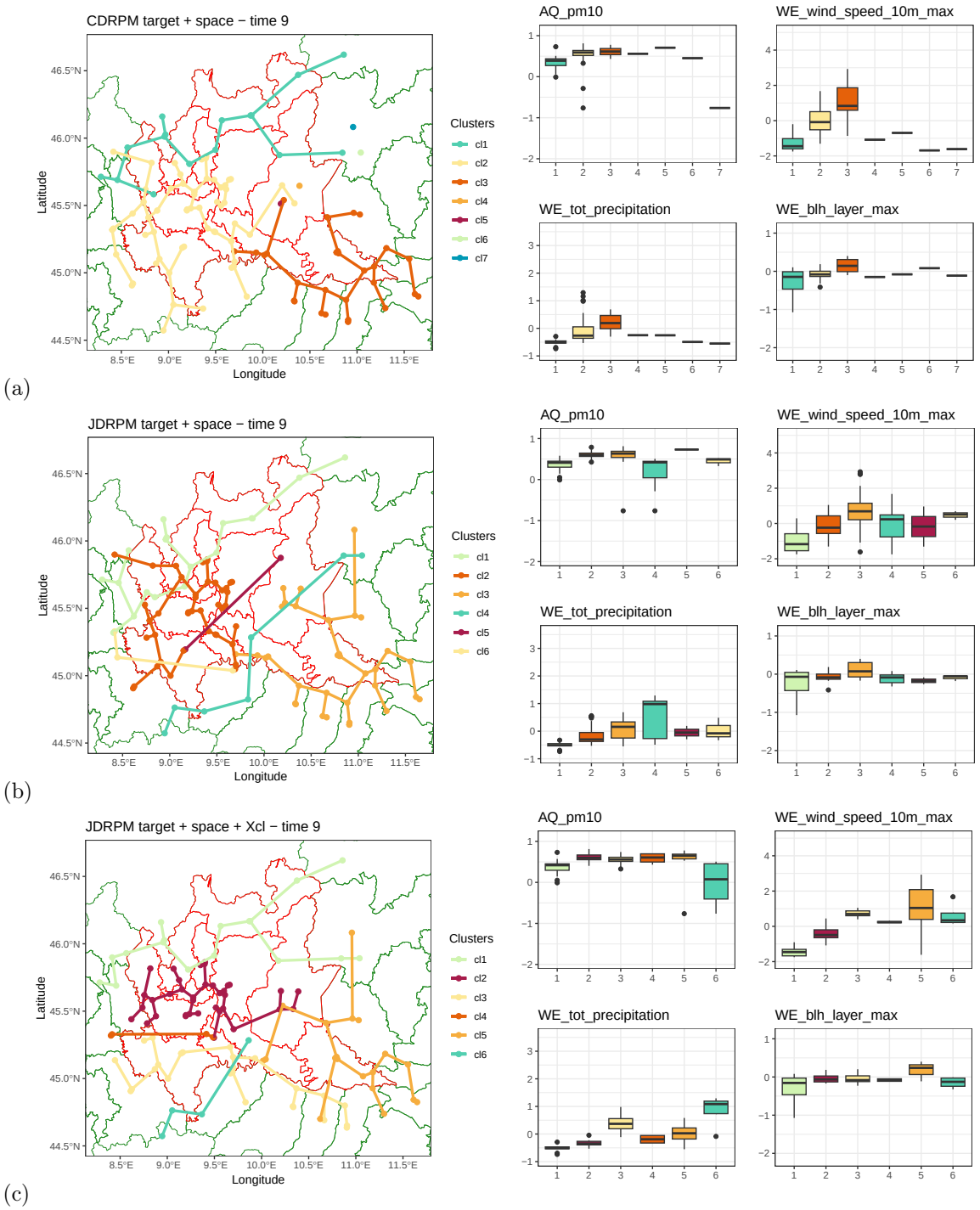


Figure 3.21: Lagged ARI values of CDRPM and JDRPM fits, in the real-world scenario, with covariates in the prior.

The results of this analysis are quite promising, as highlighted in Table 3.6, where we observe substantial improvements across all metrics when covariates are incorporated into the prior. The temporal trend of this latter fit is proposed in Figure 3.21. Although incorporating multiple covariates introduces the possibility of divergent "clustering suggestions," their contributions can still be effectively assessed. This is illustrated by complementing the partition representation with cluster-specific boxplots for the target variable PM_{10} and the three covariates considered in the analysis. Figure 3.22 provides a comparative overview of the fits reported in Table 3.6. This figure demonstrates how the JDRPM fit with covariates in the prior captured a more refined and coherent clustering structure compared to its two competitors.

The JDRPM fit with covariates in the prior, illustrated in panel (c), is the only model that demonstrates a clear separation among the covariates, particularly regarding wind speed and total precipitation. A subtler separation also emerges for the BLH covariate, although somewhat obscured by the overall proximity of values across all the units, but evidenced by the disappearance of outliers present



(a)

(b)

(c)

Figure 3.22: Visual representation of the clusters estimates produced by CDRPM and JDRPM fits, in the real-world scenario, exemplified by the case $t = 9$. The three panels refer to the standard spatially-informed fits of CDRPM (a) and JDRPM (b), along with the JDRPM with covariates in the prior (c).

in cluster 2 in panels (a) and (b). These distinctions, supported by the improved metrics, appear to contribute to a more accurate estimation of the clusters.

For instance, all models successfully identified the cluster extending over the mountain arch in the northern part of the map. This cluster is characterized by

relatively low levels of PM_{10} and low values for the wind speed covariate. This is clearly depicted in panel (c), where cluster 1 is positioned at the bottom of the plot and is well-separated from the other clusters. In panels (a) and (b), the associated wind speed boxplots also place cluster 1 at the lower end of the scale; however, its values intersect with those of other clusters. This suggests that some units correctly assigned to cluster 1 by model (c) may have been misclassified by the other models.

Another notable improvement is observed in the identification of the most polluted cluster, depicted in dark red across all panels. While both spatially-informed models yielded smaller clusters, a singleton in panel (a) and a couple in panel (b), model (c) provided a more comprehensive analysis by identifying a larger and more spatially connected region of highly polluted units. This cluster is also characterized by the second lowest levels of wind speed and total precipitation, suggesting that these factors likely hindered pollutant dispersion, leading to these cities becoming the most polluted. Again, this insight was facilitated by the covariate contributions of model (c), which remained partially concealed in the purely spatially-informed fits presented in panels (a) and (b).

The impact of incorporating covariates in the prior is further illustrated in Figures 3.23 and 3.24, which compare the distribution of clusters with respect to the values of the `WE_wind_speed_10m_max` covariate. In these figures, colors represent cluster labels, so that units are ordered not by their conventional indexing $i = 1, \dots, 105$ but rather by the clusters sets $\{i : i \in S_{1t}\}, \dots, \{i : i \in S_{k_t t}\}$. While all models employed PM_{10} as the target variable, suggesting that the highest degree of separation should be observed there, we can expect that the inclusion of covariates would also reveal a clustering pattern within the auxiliary covariates included. Indeed, the distribution of clusters in the JDRPM fit with covariates exhibits a clearer distinction regarding the wind speed variable, while this separation appears more ambiguous in the CDRPM fit. The effects of including covariates are particularly evident at time points 8, 9, and 12. For instance, at $t = 12$, the red cluster identified by CDRPM in Figure 3.23 splits into red, green, and blue clusters that are more distinctly separated in the JDRPM informed with covariates shown in Figure 3.24. Additionally, when comparing this latter JDRPM fit with covariates to the standard spatially-informed JDRPM fit of Figure 3.25, we also observe a slight improvement.

It is important to note that the enhancement resulting from the inclusion of covariates could potentially have been more pronounced had we set a higher value for the parameter `cv_weight` in the Julia function implementing the MCMC algorithm. In this analysis, in fact, the covariate-informed fit was obtained with this parameter set to 0.2, to balance the total contribution of the covariates with that of the spatial information. Therefore, increasing the value of `cv_weight` would likely yield even clearer distinctions in this covariate separation analysis.

3.3.3 Inference on a new location

We now conduct a final analysis that consolidates all the enhancements introduced by the JDRPM. In this analysis, we consider a spatio-temporal scenario

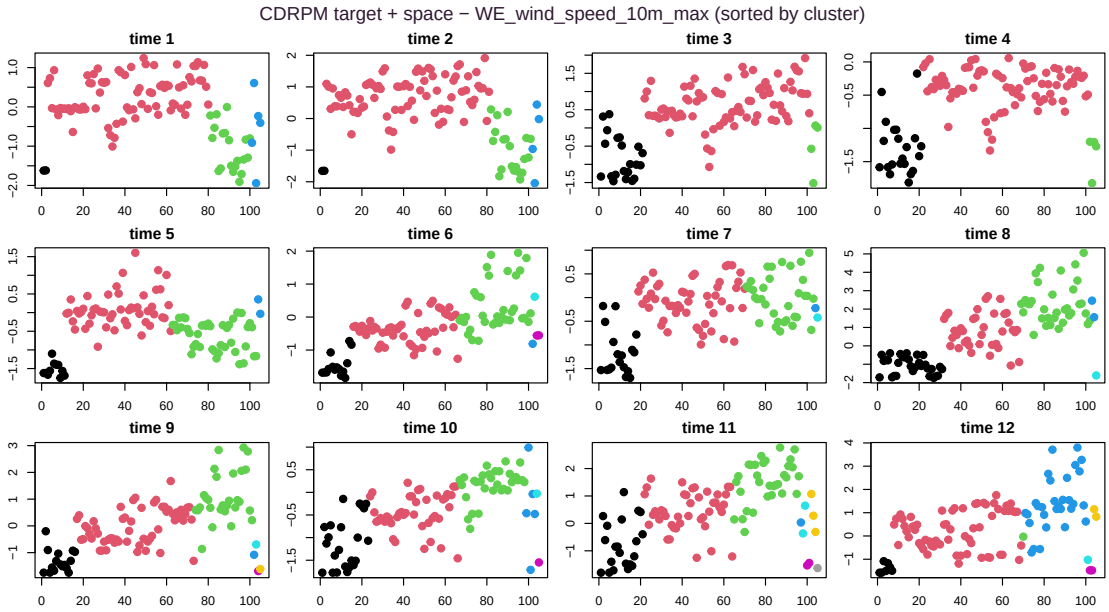


Figure 3.23: Distributions of clusters estimates with respect to the wind speed covariate, in the CDRPM spatially-informed fit.

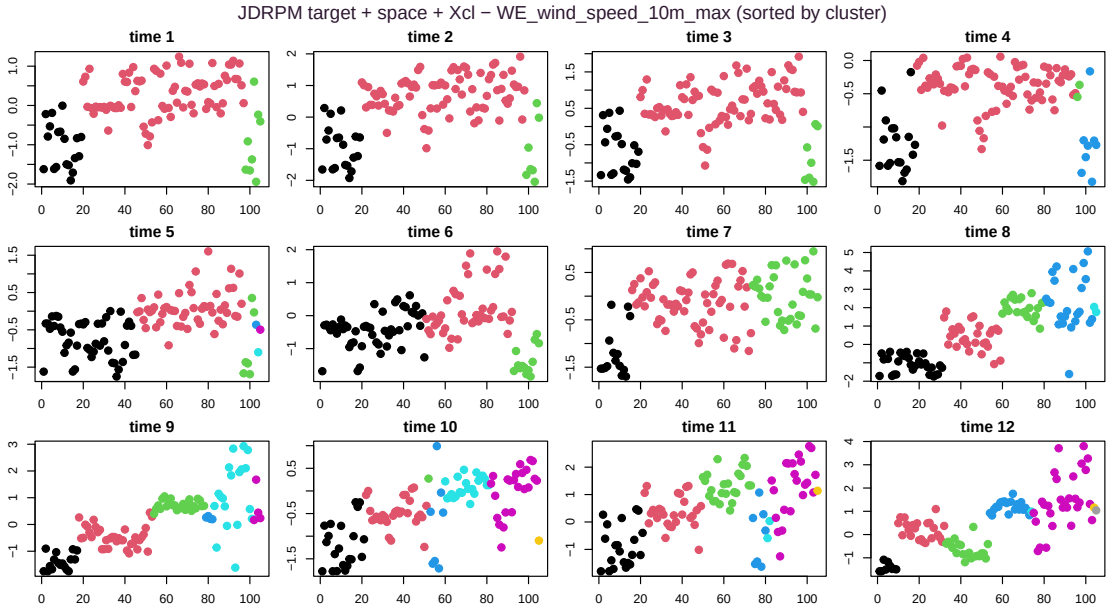


Figure 3.24: Distributions of clusters estimates with respect to the wind speed covariate, in the JDRPM spatially-informed fit with covariates in the prior.

in which a new unit is added at a new location, with the objective of inferring the values of its target variable time series. Alternatively, this scenario can be interpreted as removing all data entries from an existing unit within the dataset. In this context of kriging (Krige, 1951), we aim to evaluate the JDRPM's accuracy in predicting the behavior of a unit for which sensors may be absent or inactive, with the expectation that the estimation accuracy will improve as model complexity increases. To assert this, we conducted an experiment in which all data entries of the response variable were to NA for three randomly selected units, represented

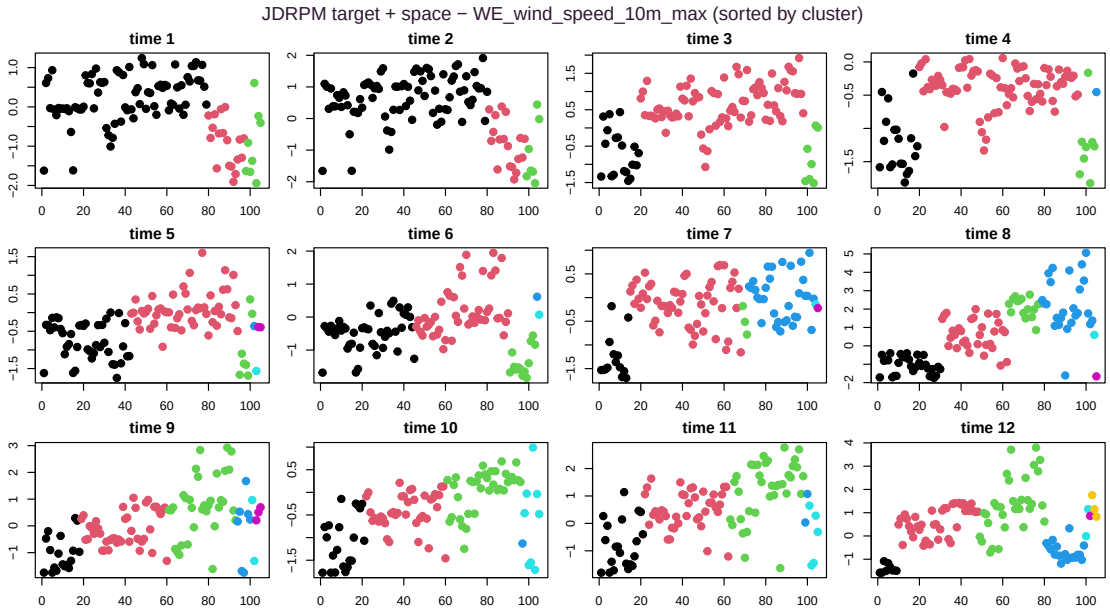


Figure 3.25: Distributions of clusters estimates with respect to the wind speed covariate, in the JDRPM spatially-informed fit.

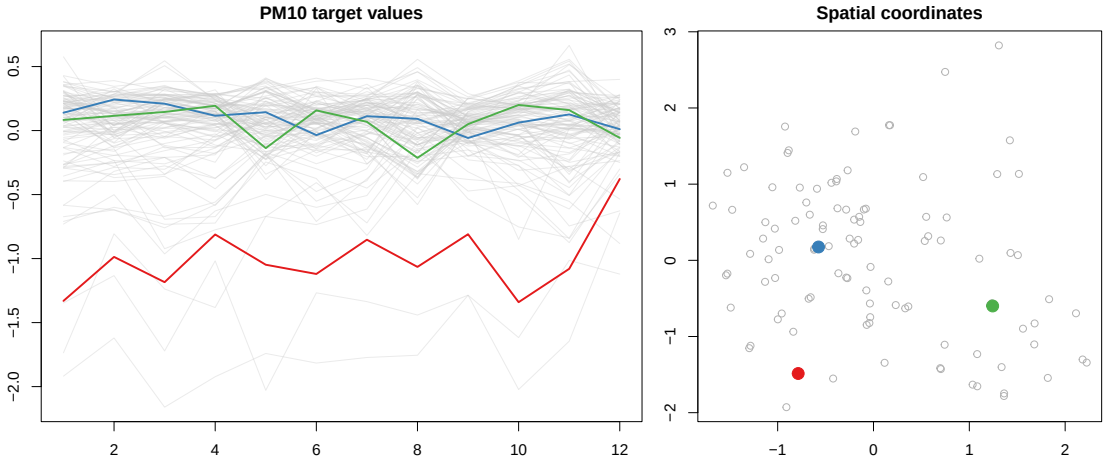


Figure 3.26: Representation of the three units selected for the kriging analysis, with their associated target variable time series (left) and spatial coordinates (right).

in Figure 3.26. We performed multiple JDRPM fits with incremental levels of information: firstly with only the spatial information, then incorporating covariates in the likelihood, and finally including covariates also in the prior. We do not report the results from tests that included only covariates in the prior, as such an approach would be more aligned for clustering, rather than kriging. We used the same set of six covariates employed in Section 3.3.1 for the likelihood component, and the same set of three covariates used in Section 3.3.2 for the prior component.

Table 3.7 illustrates that the inclusion of covariates substantially improved the estimates for the missing units. The MSEs were calculated by taking both the mean and median of the fitted values from each model and comparing them to the true values of the missing units. Overall, the fit that incorporated covariates solely in

the likelihood achieved the highest performance level. Notably, the base spatially-informed fit provided greater estimation accuracy for the fitted values associated with the green unit. In contrast, for the other units, the covariate-informed fits proved to be substantially more accurate. Figures 3.27 to 3.29 illustrate the 95% credible intervals of the estimated target values for the kriging units, derived from the three models examined.

Table 3.7: Comparison of JDRPM fits and their associated algorithms, in the kriging scenario.

		space	space+Xlk	space+Xlk+Xcl
unit 92 (red)	MSE mean	0.112452	**0.042037**	0.044957
	MSE median	0.111573	**0.041676**	0.045216
unit 61 (blue)	MSE mean	0.004117	**0.002449**	0.002527
	MSE median	0.004711	0.002547	**0.002534**
unit 44 (green)	MSE mean	**0.003919**	0.006368	0.005945
	MSE median	**0.003997**	0.006419	0.005950
	execution time	45m	**40m**	1h15m
	LPML	620.98	758.11	**791.86**
	WAIC	-1842.48	**-2041.95**	-1976.73

According to (F. A. Quintana et al., 2015), the sole inclusion of covariates in the likelihood accounts only for their linear effects, as represented by the regressor term $\boldsymbol{\beta}_t$, resulting in a minimal impact on clusters estimates. In contrast, incorporating covariates solely in the prior captures all effects of the covariates, often leading to a greater number of smaller, more specific clusters, without necessarily improving the fitted estimates. A balanced approach can be achieved by incorporating covariates at both levels of information. This strategy allows for the linear effects to be considered in the likelihood, while also accommodating nonlinear effects in the clustering process. As a result, this dual approach can yield improved outcomes for both fitted values and cluster estimates.

3.4 Scaling performances

To rigorously assess whether the objective of faster execution times was achieved, we designed a series of experiments to compare the two models across different dataset sizes and varying levels of information.

Regarding information levels, CDRPM allowed clustering based solely on the target variable or with additional spatial information, while JDRPM expanded these options enabling the inclusion of covariates at both the likelihood and prior levels. We recall that, aside from the target variable which is of course always required, all other information levels are optional and independent, allowing flexible configurations such as fitting with covariates but without spatial information. However, with an additive perspective in mind, we conducted incremental experiments where we progressively inserted new information layers on top of each other. Consequently,

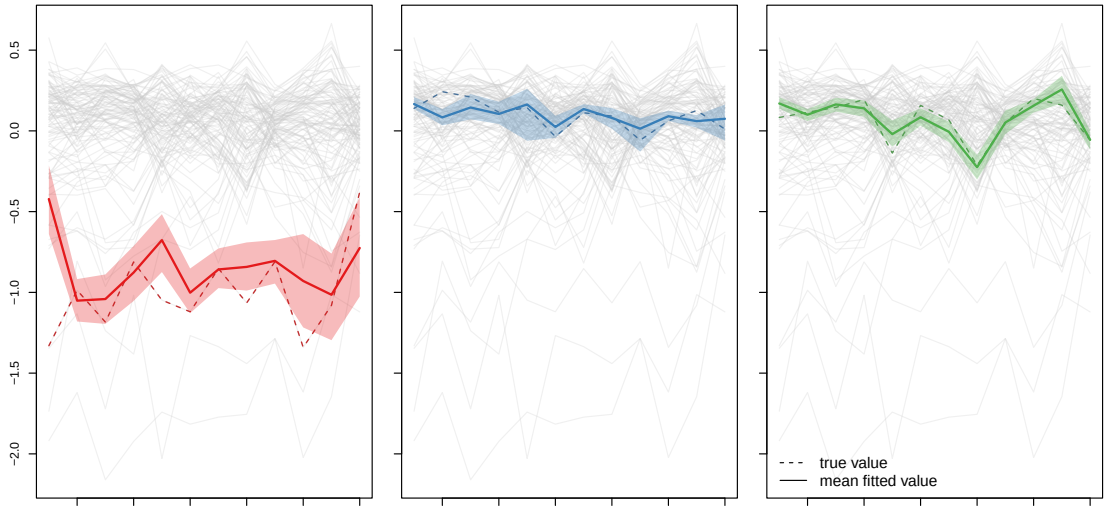


Figure 3.27: Kriging performances of JDRPM spatially-informed fit.

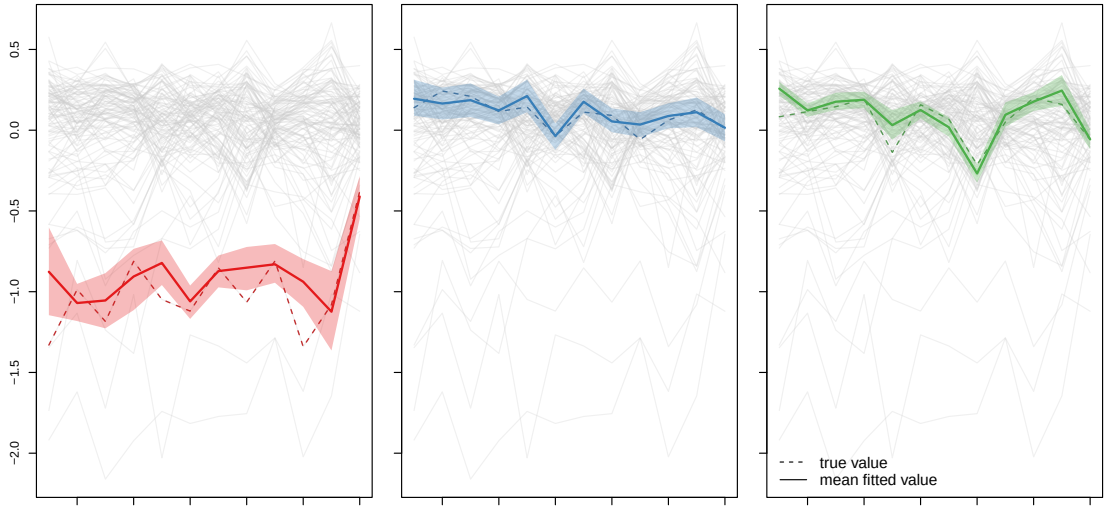


Figure 3.28: Kriging performances of JDRPM spatially-informed fit with covariates in the likelihood.

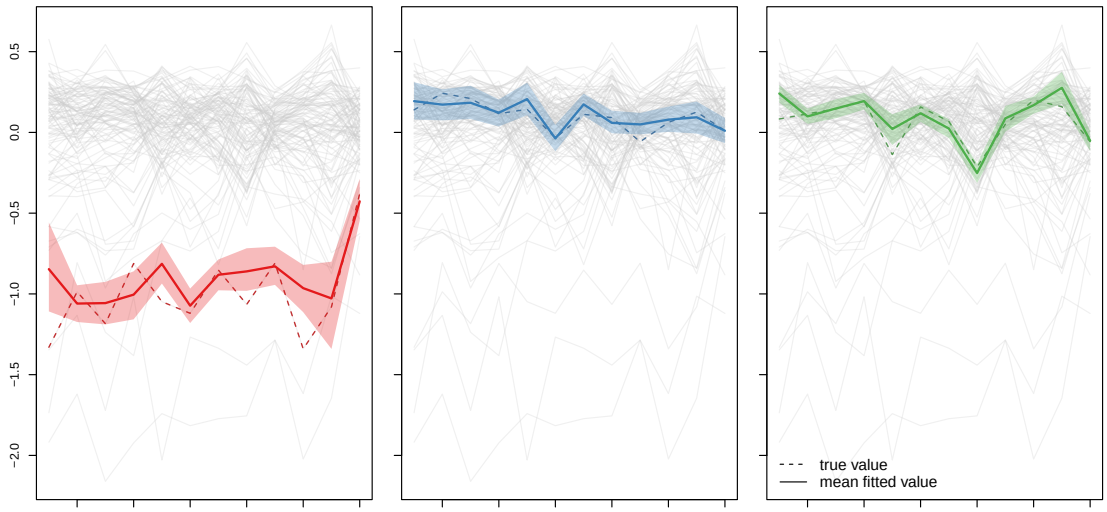


Figure 3.29: Kriging performances of JDRPM spatially-informed fit with covariates in the likelihood and in the prior.

we will compare fits starting with the target variable, followed by the addition of spatial information, the inclusion of covariates in the prior, and finally the inclusion of covariates also in the likelihood.

The different information levels affect the model complexity and therefore reflect into the computational load, however also the size of the testing dataset plays a significant role. To systematically explore this , we conducted the experiments across a "mesh" of dataset sizes, with both the number of units n and the time horizons T ranging trough the set $\{10, 50, 100, 250\}$. As discussed in Section 2, the Julia implementation exhibited a slightly higher memory demand compared to the C implementation. As a consequence, when running the algorithms on the larger datasets, my system possibly ran out of RAM, requiring some data to be stored in the slower swap space on the disk and therefore leading to a drop in performance. As such, the extreme-sized experiments with n or T equal to 250 should be taken with a pinch of salt as they may reflect system hardware limitations rather than intrinsic model performance. For all the other experiments, on the other hand, the results should be highly accurate and reliably proving the JDRPM's improved performance.

In conducting the comparisons, we generated synthetic target data Y_{it} and spatial coordinates $\boldsymbol{s}_i$ according to the values of n and T. To measure the average execution time per iteration of each fit we defined the number of iterations to be inversely proportional to the size of the dataset, e.g. 10000 iterations in the case $(n, T) = (10, 10)$ and 16 iterations in the case $(n, T) = (250, 250)$. This ensured that each run lasted approximately the same amount of time. Moreover, each fit was repeated multiple times to record the minimum execution time observed. This practice is common in benchmarking and helps to eliminate bias attributable to the system computational demands and fluctuations to simulate the "ideal" testing environment in which all computational resources are devoted exclusively to the model fitting task.

As shown in Figure 3.30, the basic fit using only target values achieves significantly faster execution times in Julia, with speedups peaking around a 2x improvement. Similar performance gains are observed in the fits that incorporate spatial information, as illustrated in Figure 3.31. Therefore, particularly when examining the more reliable intermediate-sized tests, we can confidently conclude that the JDRPM algorithm implementation outperforms the CDRPM algorithm implementation with respect to execution speed.

In the analysis of fits involving covariates, we generated them by randomly creating matrices of dimensions $n \times p \times T$. For these experiments, we maintained a consistent value for p, the number of covariates, to avoid complications. In fact, the previous Figures 3.30 and 3.31 represented projections of three-dimensional data: n, T, and the execution time. If we extended the mesh construction to allow for varying p we would have dealt with four-dimensional data, or even five-dimensional if we considered separately covariates in the prior and in the likelihood. Therefore, to preserve clarity and comprehensibility in our presentation, we fixed $p = 5$ for both covariates information levels. Nonetheless, a test with varying p for both prior and likelihood covariates, but fixed n and T, will be proposed in Figure 3.33.

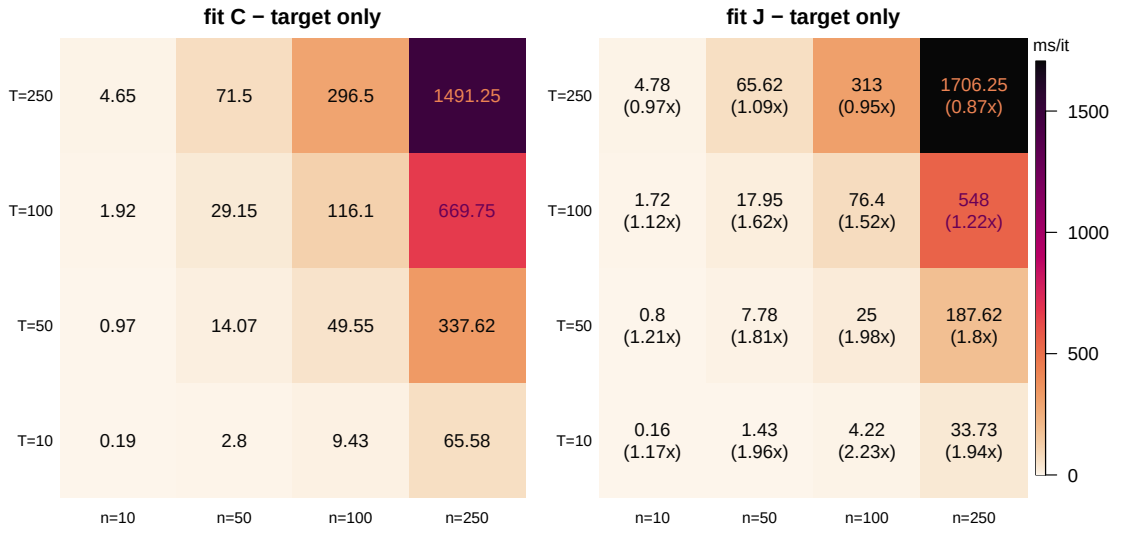


Figure 3.30: Execution times, measured in milliseconds per iteration, when fitting CDRPM and JDRPM in a simulated data scenario. In the JDRPM plot (right), in brackets, are reported the speedups relative to the CDRPM timings (left), where higher values indicate better performance.

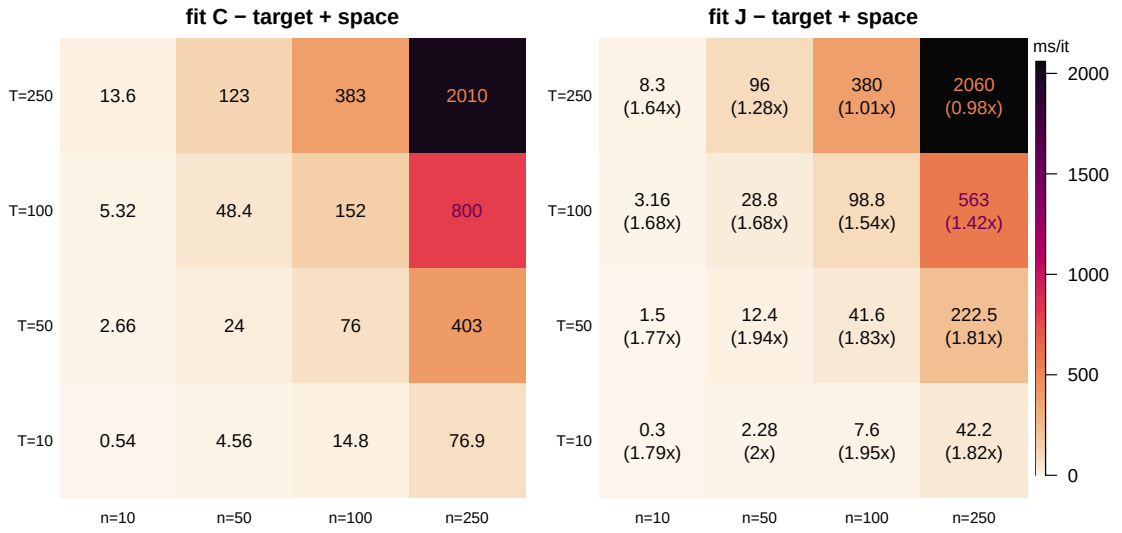


Figure 3.31: Execution times, measured in milliseconds per iteration, when fitting CDRPM and JDRPM in a real-world scenario. In the JDRPM plot (right), in brackets, are reported the speedups relative to the CDRPM timings (left), where higher values indicate better performance.

Figure 3.32 illustrates that fits incorporating covariates experienced a decrease in performance, which was expected due to the additional computational demands introduced by the new information layers. However, they maintained a satisfactory level of performance. As shown in Figure 3.34, some fits that included all information levels in Julia were surprisingly faster than a standard spatially-informed fit implemented in C.

Building on the encouraging results presented, we further investigated the model complexity at which the JDRPM implementation would reach the execution times provided by CDRPM. To conduct this analysis, we selected a moderately-sized

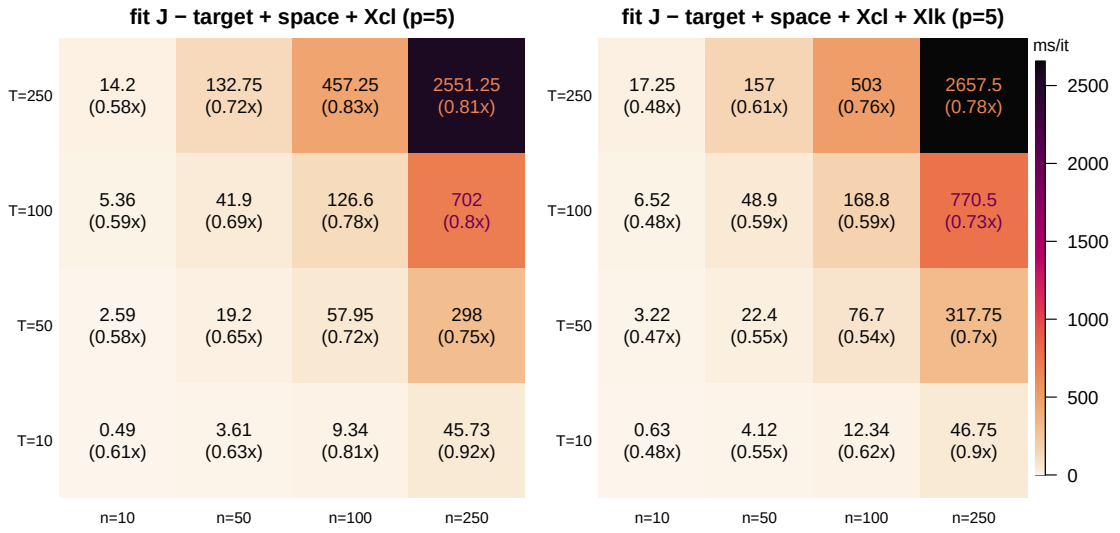


Figure 3.32: Execution times, measured in milliseconds per iteration, when fitting JDRPM in a real-world scenario, with a fixed number $p = 5$ of covariates in the prior (left) and in both the prior and the likelihood (right). In brackets are reported the speedups relative to the JDRPM timings of the fits with spatial information, with higher values still indicating better performance.

dataset with $n = 50$ and $T = 50$, and used the performance metric from the CDRPM fit with spatial information as a reference; a value of 24 ms/iteration as retrieved by Figure 3.31. We then assessed the performance of JDRPM fits that included covariates at both likelihood and clustering levels, allowing p to vary independently in each information level to identify the degree of complexity at which the new implementation would align with the original one. The results of this analysis are summarized in Figure 3.33. Our experiments suggest that until that until we include $p_{\text{cl}} = 5$ covariates in the prior, we can expect JDRPM to outclass the CDRPM performance reference. This indicates that within the same amount of time in which CDRPM executes a spatially-informed fit, JDRPM can perform a fit including up to five covariates in the prior. Additionally, there is potential to include an indeterminate number of covariates in the likelihood, since the main performance drop appeared to associated with the increasing of p_{cl}, rather than of p_{lk}.

As in the previous tests of this section, this analysis was conducted with the intention of dedicating all available computational resources to the fitting task; therefore, the results should be regarded as accurate. Notably, even in the noisier environment of the real-world experiments, a considerable speedup was observed, further proving the performance improvements achieved. The real-world experiments were carried out on a dataset comprising $n = 105$ units and $T = 12$ time instants, with a summary of their results presented in Table 3.6. From this table, we note that the spatially-informed CDRPM fit required 1 hour and 38 minutes, while both JDRPM fits, with equivalent setup and with covariates in the prior, exhibited faster execution times. Remarkably, the equivalent JDRPM fit reduced the execution time by more than half, aligning with the measured speedup factor of 1.95x reported in Figure 3.31 when considering the closest corresponding case of $n = 100$ and $T = 10$.

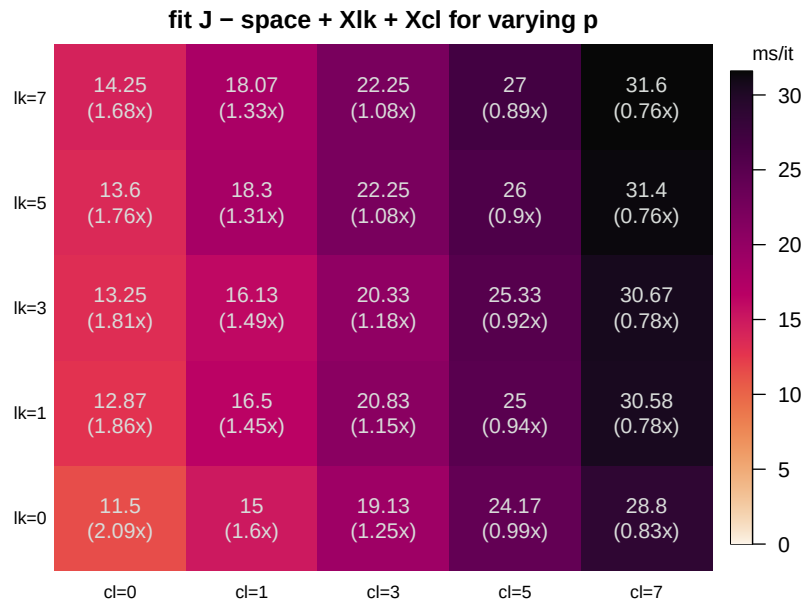


Figure 3.33: Execution times, measured in milliseconds per iteration, when fitting JDRPM in a real-world scenario, with a varying number of covariates in the likelihood (symbol lk on the y axis) and in the prior (symbol cl on the x axis). In brackets are reported the speedups relative to the CDRPM timing of the spatially-informed fit on the same $n = 50$, $T = 50$ dataset size, with higher values indicating better performance.

Additionally, the fit that included covariates in the prior demonstrated the expected speedup suggested by Figure 3.34, which we now discuss in detail.

Figure 3.34 encapsulates all the performance analyses conducted so far. It also offers an additional insight: the bottom right panel, corresponding to experiments on $n = 250$ units, reveals a convergence pattern in the speedup factors across all fits, denoting how all models tend to exhibit similar execution times regardless of the information levels included. This suggests that when applied to large-scale datasets, the performance bottleneck of JDRPM but also CDRPM shifts from the algorithmic complexity, that is, from the selection of desired information levels, to the limitations of available hardware resources.

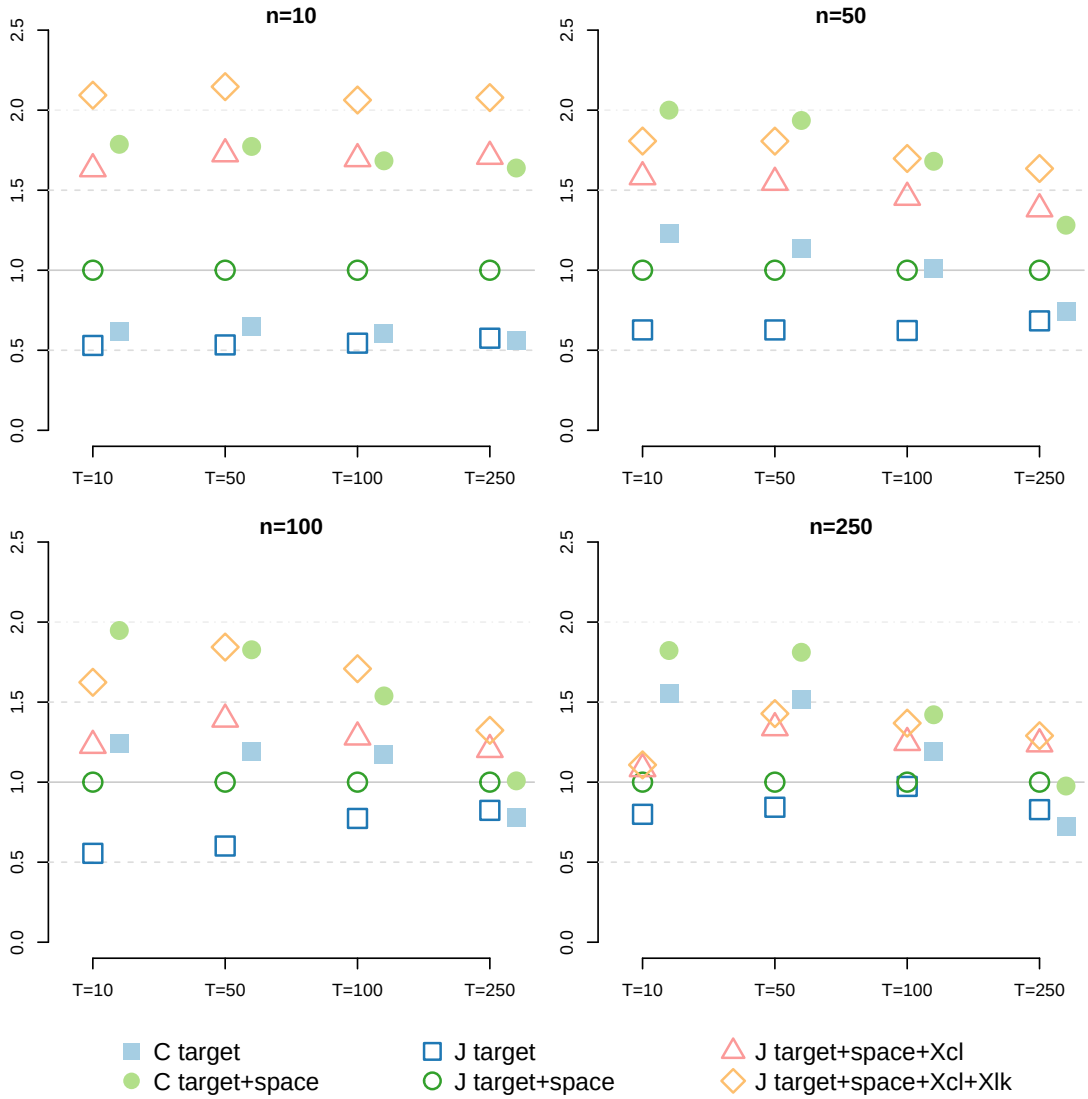


Figure 3.34: Visual representation of the performance of all the fits, for all the n and T cases, relatively to the JDRPM fit using target values and spatial information. Namely, the execution time per iteration metric of that fit has been taken as a reference, to which then all other fits have been 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 conclusion, the JDRPM represents a significant enhancement over the original CDRPM. From a theoretical perspective, JDRPM retains the foundational structure of its predecessor while introducing the covariates in the prior and likelihood levels. Together with the reduction in the execution times, this is the core upgrade since it should help to yield more accurate and informed results in the partitions. Moreover, the modification on the variances, from a uniform law to an inverse gamma, possibly could enhance the quality of the posterior samples. This choice, in fact, restores conjugacy within the model, thereby improving the mixing properties of the Markov chain during the fitting process.

Despite these improvements, it is important to acknowledge potential drawbacks associated with the increased complexity of the model. The robustness of results may diminish as the intricacies of parameter selection—particularly concerning cohesion and similarity functions—can significantly influence cluster estimates. Striking an appropriate balance between these two sources of information may require empirical testing. In this context, Chapter 1 provides a comprehensive analysis of how tuning parameters for cohesion and similarity impact model performance.

Despite these improvements, it is important to acknowledge potential drawbacks associated with the increased complexity of the model. The robustness of the fits may diminish as the intricacies of parameters selection, particularly concerning the ones regulating cohesion and similarity functions, can significantly influence the cluster estimates. To this end, Chapter 1 provided a comprehensive analysis about how the possible choices on the parameters for cohesions and similarities reflect on their generated values. Moreover, in fits including both space and covariates information, reaching an appropriate balance between these two sources of information may require empirical testing. To address this balance, the Julia function `MCMC_fit` includes an optional argument, `cv_weight`, defaulted to 1, that allows to adjust the influence of covariate similarities.

This complexity is especially evident when defining the prior for the inverse gamma distribution, inherently more delicate than a simpler uniform, of the variance parameters. Other than conducting multiple experiments and studying the trace plots to asses a correct behaviour, a pragmatic approach to address this challenge could be in conducting an initial fit using the original CDRPM to see the expected range in which the variance samples tend to settle, and tune accordingly the inverse gamma parameters. For instance, in the spatio-temporal tests detailed in Section 3.1.2, we observed low variance estimates from the CDRPM fits. Consequently, we assigned an invGamma($a = 1.9, b = 0.4$) for λ^2 and τ_t^2, which has 90% of his density in the interval $[0.109151, 1.58222]$. For the more critical parameter $\sigma_{jt}^{2\star}$ we adopted a less informative prior invGamma($a = 0.01, b = 0.01$) which, despite being not always recommended (Andrew Gelman, 2004), proved to be effective and precise relative to sampled reference values from CDRPM. An alternative strategy, albeit less theoretically appealing, could involve truncating the inverse gamma distributions to mitigate the risk of sampling excessively high values when uninformative priors are inadequately adjusted by data. This adjustment can be seamlessly integrated into the Julia code by modifying `rand(InverseGamma(a,b))` to `rand(truncated(InverseGamma(a,b),l,u))`, where l and u define truncation bounds.

From a computational perspective, we successfully reduced execution times by up to 50% compared to the original implementation. Although this occurred with a slight increase in memory requirements, it is a trade-off that is surely manageable with the modern computing resources.

Looking ahead, there remains a wide opportunity for further enhancements, particularly in the usability front given the actual complexity of the model. While the current JDRPM implementation features basic logging capabilities, that allow for stepwise computation tracking, there is potential for more sophisticated profiling and monitoring tools that leverage Julia's flexibility. Possible suggestions could include heuristics about the initialization of the hyperparameters, based on the specific datasets at hand, or visualization tools, using Julia's robust plotting ecosystem, to provide real-time monitoring of the sampled parameters distributions and trace plots directly during execution. Moreover, implementing parallel processing with multiple chains, which is a common in technique implemented in many less heavy Bayesian models, could further enhance performance. Exploring GPU integration through packages within the `JuliaGPU` collection may also yield significant computational efficiencies.

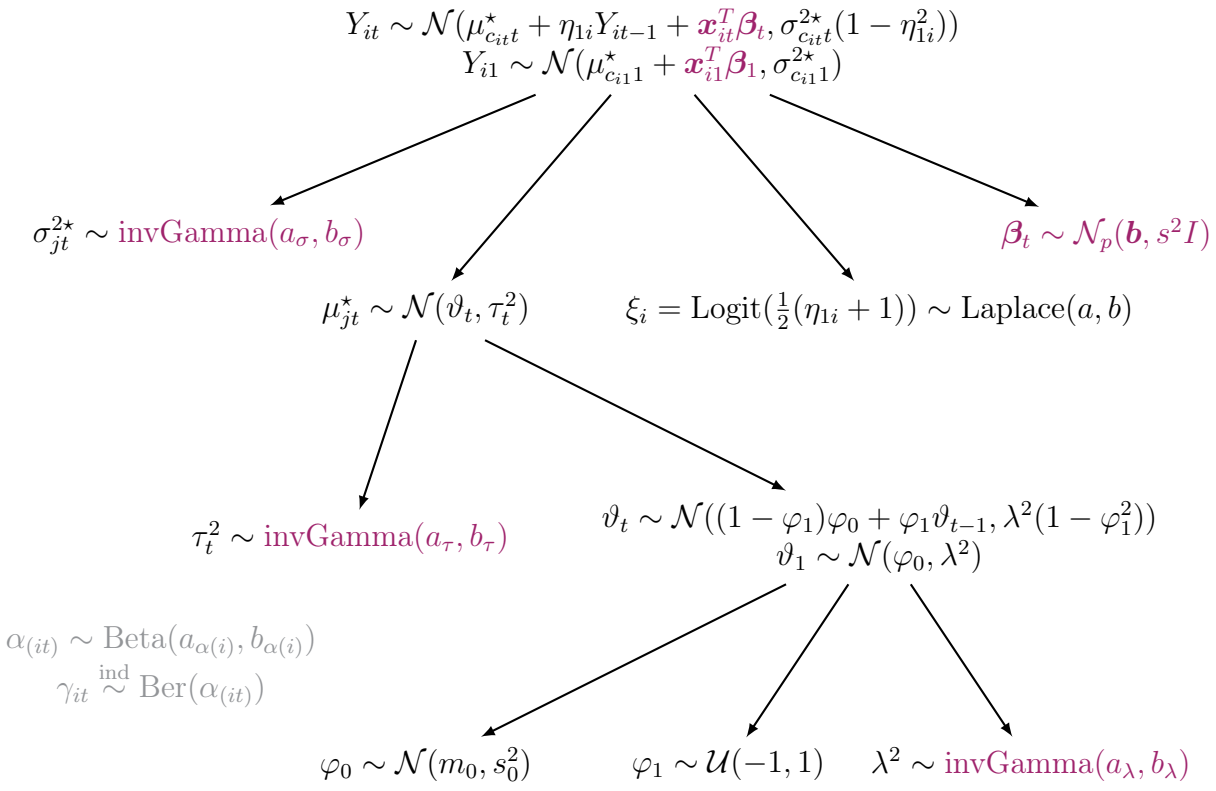
In short, while the JDRPM's complexity may present certain challenges, it also lays the groundwork for significant methodological advancements and practical enhancements. These developments are expected to enhance the model's applicability and ease of use, for more effective research outcomes.

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 of JDRPM to make it quickly accessible as a reference for the laws involved in the following computations.



- update $\sigma^{2\star}_{jt}$. 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.

for $t = 1$:

$$
\begin{aligned}
f(\sigma_{jt}^{2\star}|-) &\propto f(\sigma_{jt}^{2\star}) f(\{Y_{it} : c_{it} = j\}|\sigma_{jt}^{2\star}, -) \\
&= \mathcal{L}_{\text{invGamma}(a_\sigma, b_\sigma)}(\sigma_{jt}^{2\star}) \prod_{i \in S_{jt}} \mathcal{L}_{\mathcal{N}(\mu^\star_{c_{it}t} + \boldsymbol{x}_{it}^T \boldsymbol{\beta}_t, \sigma^{2\star}_{c_{it}1})}(Y_{it}) \\
&\propto \left[\left(\frac{1}{\sigma_{jt}^{2\star}} \right)^{a_\sigma + 1} \exp\left\{ -\frac{1}{\sigma_{jt}^{2\star}} b \right\} \right] \\
&\cdot \left[\prod_{i \in S_{jt}} \left(\frac{1}{\sigma_{jt}^{2\star}} \right)^{1/2} \exp\left\{ -\frac{1}{2\sigma_{jt}^{2\star}} (Y_{it} - \mu_{jt}^\star - \boldsymbol{x}_{it}^T \boldsymbol{\beta}_t)^2 \right\} \right] \\
&\propto \left(\frac{1}{\sigma_{jt}^{2\star}} \right)^{(a_\sigma + |S_{jt}|/2) + 1} \exp\left\{ -\frac{1}{\sigma_{jt}^{2\star}} \left(b_\sigma + \frac{1}{2} \sum_{i \in S_{jt}} (Y_{it} - \mu_{jt}^\star - \boldsymbol{x}_{it}^T \boldsymbol{\beta}_t)^2 \right) \right\}
\end{aligned}
$$

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

$$
\begin{aligned}
a_{\tau(\text{post})} &= a_\sigma + \frac{|S_{jt}|}{2} \\
b_{\tau(\text{post})} &= b_\sigma + \frac{1}{2} \sum_{i \in S_{jt}} (Y_{it} - \mu_{jt}^\star - \boldsymbol{x}_{it}^T \boldsymbol{\beta}_t)^2
\end{aligned} \tag{A.1}
$$

for $t > 1$:

$$
\begin{aligned}
f(\sigma_{jt}^{2\star}|-) &\propto f(\sigma_{jt}^{2\star}) f(\{Y_{it} : c_{it} = j\}|\sigma_{jt}^{2\star}, -) \\
&= \mathcal{L}_{\text{invGamma}(a_\sigma, b_\sigma)}(\sigma_{jt}^{2\star}) \prod_{i \in S_{jt}} \mathcal{L}_{\mathcal{N}(\mu^\star_{c_{it}t} + \eta_{1i} Y_{it-1} + \boldsymbol{x}_{it}^T \boldsymbol{\beta}_t, \sigma^{2\star}_{c_{it}1})}(Y_{it}) \\
&\propto \left[\left(\frac{1}{\sigma_{jt}^{2\star}} \right)^{a_\sigma + 1} \exp\left\{ -\frac{1}{\sigma_{jt}^{2\star}} b \right\} \right] \\
&\cdot \left[\prod_{i \in S_{jt}} \left(\frac{1}{\sigma_{jt}^{2\star}} \right)^{1/2} \exp\left\{ -\frac{1}{2\sigma_{jt}^{2\star}} (Y_{it} - \mu_{jt}^\star - \eta_{1i} Y_{it-1} - \boldsymbol{x}_{it}^T \boldsymbol{\beta}_t)^2 \right\} \right] \\
&\propto \left(\frac{1}{\sigma_{jt}^{2\star}} \right)^{(a_\sigma + |S_{jt}|/2) + 1} \\
&\cdot \exp\left\{ -\frac{1}{\sigma_{jt}^{2\star}} \left(b_\sigma + \frac{1}{2} \sum_{i \in S_{jt}} (Y_{it} - \eta_{1i} Y_{it-1} - \mu_{jt}^\star - \boldsymbol{x}_{it}^T \boldsymbol{\beta}_t)^2 \right) \right\}
\end{aligned}
$$

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

$$
\begin{aligned}
a_{\tau(\text{post})} &= a_\sigma + \frac{|S_{jt}|}{2} \\
b_{\tau(\text{post})} &= b_\sigma + \frac{1}{2} \sum_{i \in S_{jt}} (Y_{it} - \mu_{jt}^\star - \eta_{1i} Y_{it-1} - \boldsymbol{x}_{it}^T \boldsymbol{\beta}_t)^2
\end{aligned} \tag{A.2}
$$

- update $\mu^\star_{jt}$. This update rule is the same for both JDRPM and CDRPM.

  for $t = 1$:

$$
\begin{aligned}
f(\mu^\star_{jt}|-) &\propto f(\mu^\star_{jt}) f(\{Y_{it} : c_{it} = j\}|\mu^\star_{jt}, -) \\
&= \mathcal{L}_{\mathcal{N}(\vartheta_t, \tau_t^2)}(\mu^\star_{jt}) \prod_{i \in S_{jt}} \mathcal{L}_{\mathcal{N}(\mu^\star_{c_{it}t} + \boldsymbol{x}_{it}^T \boldsymbol{\beta}_t, \sigma^{2\star}_{c_{it}1})}(Y_{it}) \\
&\propto \exp\left\{ -\frac{1}{2\tau_t^2} (\mu^\star_{jt} - \vartheta_t)^2 \right\} \exp\left\{ -\frac{1}{2\sigma^{2\star}_{jt}} \left(\sum_{i \in S_{jt}} (\mu^\star_{jt} - (Y_{i1} - \boldsymbol{x}_{it}^T \boldsymbol{\beta}_t))^2 \right) \right\} \\
&\propto \exp\left\{ -\frac{1}{2\tau_t^2} (\mu^\star_{jt} - \vartheta_t)^2 \right\} \exp\left\{ -\frac{|S_{jt}|}{2\sigma^{2\star}_{jt}} \left(\mu^\star_{jt} - \frac{\text{SUM}_y}{|S_{jt}|} \right)^2 \right\} \\
&\text{where } \text{SUM}_y = \sum_{i \in S_{jt}} Y_{i1} - \boldsymbol{x}_{it}^T \boldsymbol{\beta}_t
\end{aligned}
$$

  $\implies f(\mu^\star_{jt}|-) \propto$ kernel of a $\mathcal{N}(\mu_{\mu^\star_{jt}(\text{post})}, \sigma^2_{\mu^\star_{jt}(\text{post})})$ with

$$
\begin{aligned}
\sigma^2_{\mu^\star_{jt}(\text{post})} &= \frac{1}{\frac{1}{\tau_t^2} + \frac{|S_{jt}|}{\sigma^{2\star}_{jt}}} \\
\mu_{\mu^\star_{jt}(\text{post})} &= \sigma^2_{\mu^\star_{jt}(\text{post})} \left(\frac{\vartheta_t}{\tau_t^2} + \frac{\text{SUM}_y}{\sigma^{2\star}_{jt}} \right)
\end{aligned} \tag{A.3}
$$

  for $t > 1$:

$$
\begin{aligned}
f(\mu^\star_{jt}|-) &\propto f(\mu^\star_{jt}) f(\{Y_{it} : c_{it} = j\}|\mu^\star_{jt}, -) \\
&= \mathcal{L}_{\mathcal{N}(\vartheta_1, \tau_t^2)}(\mu^\star_{jt}) \prod_{i \in S_{jt}} \mathcal{L}_{\mathcal{N}(\mu^\star_{c_{it}t} + \eta_{1i} Y_{it-1} + \boldsymbol{x}_{it}^T \boldsymbol{\beta}_t, \sigma^{2\star}_{c_{it}1})}(Y_{it}) \\
&\propto \exp\left\{ -\frac{1}{2\tau_t^2} (\mu^\star_{jt} - \vartheta_t)^2 \right\} \\
&\cdot \exp\left\{ -\frac{1}{2\sigma^{2\star}_{jt}} \left(\sum_{i \in S_{jt}} \frac{1}{1 - \eta_{1i}^2} \left(\mu^\star_{jt} - (Y_{it} - \eta_{1i} Y_{i,t-1} - \boldsymbol{x}_{it}^T \boldsymbol{\beta}_t) \right)^2 \right) \right\} \\
&\propto \exp\left\{ -\frac{1}{2\tau_t^2} (\mu^\star_{jt} - \vartheta_t)^2 \right\} \exp\left\{ -\frac{\text{SUM}_{e2}}{2\sigma^{2\star}_{jt}} \left(\mu^\star_{jt} - \frac{\text{SUM}_y}{\text{SUM}_{e2}} \right)^2 \right\} \\
&\text{where } \text{SUM}_y = \sum_{i \in S_{jt}} \frac{Y_{it} - \eta_{1i} Y_{i,t-1} - \boldsymbol{x}_{it}^T \boldsymbol{\beta}_t}{1 - \eta_{1i}^2}, \ \text{SUM}_{e2} = \sum_{i \in S_{jt}} \frac{1}{1 - \eta_{1i}^2}
\end{aligned}
$$

  $\implies f(\mu^\star_{jt}|-) \propto$ kernel of a $\mathcal{N}(\mu_{\mu^\star_{jt}(\text{post})}, \sigma^2_{\mu^\star_{jt}(\text{post})})$ with

$$
\begin{aligned}
\sigma^2_{\mu^\star_{jt}(\text{post})} &= \frac{1}{\frac{1}{\tau_t^2} + \frac{\text{SUM}_{e2}}{\sigma^{2\star}_{jt}}} \\
\mu_{\mu^\star_{jt}(\text{post})} &= \sigma^2_{\mu^\star_{jt}(\text{post})} \left(\frac{\vartheta_t}{\tau_t^2} + \frac{\text{SUM}_y}{\sigma^{2\star}_{jt}} \right)
\end{aligned} \tag{A.4}
$$

- 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.

for $t = 1$:

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

for $t > 1$:

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

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

- 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.

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

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

$$
\begin{aligned}
&\text{for } t = T\text{:} \\
&\quad f(\vartheta_t|-) \propto f(\vartheta_t) f((\mu^{\star}_{1t}, \ldots, \mu^{\star}_{k_t t})|\vartheta_t, -) \\
&\qquad = \mathcal{L}_{\mathcal{N}((1-\varphi_1)\varphi_0 + \varphi_1 \vartheta_{t-1}, \lambda^2(1-\varphi_1^2))}(\vartheta_t) \prod_{j=1}^{k_t} \mathcal{L}_{\mathcal{N}(\vartheta_t, \tau_t^2)}(\mu^{\star}_{jt}) \\
&\qquad \propto \exp\left\{ -\frac{1}{2(\lambda^2(1-\varphi_1^2))} \Big(\vartheta_t - ((1-\varphi_1)\varphi_0 + \varphi_1 \vartheta_{t-1}) \Big)^2 \right\} \\
&\qquad \cdot \exp\left\{ -\frac{k_t}{2\tau_t^2} \left(\vartheta_t - \frac{\sum_{j=1}^{k_t} \mu^{\star}_{jt}}{k_t} \right) \right\}
\end{aligned}
$$

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

for $1 < t < T$:

$$\begin{aligned}
f(\vartheta_t|-) &\propto \underbrace{f(\vartheta_t) f((\mu^\star_{1t}, \dots, \mu^\star_{k_t t})|\vartheta_t, -)}_{\text{as in the case } t=T} f(\vartheta_{t+1}|\vartheta_t, -) \\
&= \mathcal{L}_{\mathcal{N}(\mu_{\vartheta_t(\text{post})}, \sigma^2_{\vartheta_t(\text{post})})}(\vartheta_t) \mathcal{L}_{\mathcal{N}((1-\varphi_1)\varphi_0 + \varphi_1 \vartheta_t, \lambda^2(1-\varphi_1^2))}(\vartheta_{t+1}) \\
&\propto \exp\left\{ -\frac{1}{2\sigma^2_{\vartheta_t(\text{post})}} \left(\vartheta_t - \mu_{\vartheta_t(\text{post})}\right)^2 \right\} \\
&\quad \cdot \exp\left\{ -\frac{1}{2\frac{\lambda^2(1-\varphi_1^2)}{\varphi_1^2}} \left(\vartheta_t - \frac{\vartheta_{t+1} - (1-\varphi_1)\varphi_0}{\varphi_1} \right)^2 \right\}
\end{aligned}$$

$$\begin{aligned}
&\implies f(\vartheta_t|-) \propto \text{kernel of a } \mathcal{N}(\mu_{\vartheta_t(\text{post})}, \sigma^2_{\vartheta_t(\text{post})}) \text{ with} \\
&\quad \sigma^2_{\vartheta_t(\text{post})} = \frac{1}{\frac{1+\varphi_1^2}{\lambda^2(1-\varphi_1^2)} + \frac{k_t}{\tau_t^2}} \\
&\quad \mu_{\vartheta_t(\text{post})} = \sigma^2_{\vartheta_t(\text{post})} \left(\frac{\sum_{j=1}^{k_t} \mu^\star_{jt}}{\tau_t^2} + \frac{\varphi_1(\vartheta_{t-1} + \vartheta_{t+1}) + \varphi_0(1-\varphi_1)^2}{\lambda^2(1-\varphi_1^2)} \right)
\end{aligned} \tag{A.9}$$

for $t = 1$:

$$\begin{aligned}
f(\vartheta_t|-) &\propto f(\vartheta_t) f(\vartheta_{t+1}|\vartheta_t, -) f(\boldsymbol{\mu}^\star_t|\vartheta_t, -) \\
&= \mathcal{L}_{\mathcal{N}(\varphi_0, \lambda^2)}(\vartheta_t) \mathcal{L}_{\mathcal{N}((1-\varphi_1)\varphi_0 + \varphi_1 \vartheta_{t-1}, \lambda^2(1-\varphi_1^2))}(\vartheta_{t+1}) \prod_{j=1}^{k_t} \mathcal{L}_{\mathcal{N}(\vartheta_t, \tau_t^2)}(\mu^\star_{jt}) \\
&\propto \exp\left\{ -\frac{1}{2\lambda^2} (\vartheta_t - \varphi_0)^2 \right\} \\
&\quad \cdot \exp\left\{ -\frac{1}{2\frac{\lambda^2(1-\varphi_1^2)}{\varphi_1^2}} \left(\vartheta_t - \frac{\vartheta_{t+1} - (1-\varphi_1)\varphi_0}{\varphi_1} \right)^2 \right\} \\
&\quad \cdot \exp\left\{ -\frac{k_t}{2\tau_t^2} \left(\vartheta_t - \frac{\sum_{j=1}^{k_t} \mu^\star_{jt}}{k_t} \right) \right\}
\end{aligned}$$

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

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

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

- 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.

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

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

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

$$\begin{aligned} f(\alpha|-) &\propto f(\alpha) f((\gamma_{11}, \ldots, \gamma_{1T}, \ldots, \gamma_{n1}, \ldots, \gamma_{nT})|\alpha) \\ &\propto \alpha^{a_\alpha - 1}(1-\alpha)^{b_\alpha - 1} \prod_{i=1}^{n}\prod_{t=1}^{T} \alpha^{\gamma_{it}}(1-\alpha)^{1-\gamma_{it}} \\ &= \alpha^{(a_\alpha + \sum_{i=1}^{n}\sum_{t=1}^{T}\gamma_{it}) - 1}(1-\alpha)^{(b_\alpha + nT - \sum_{i=1}^{n}\sum_{t=1}^{T}\gamma_{it}) - 1} \\ \implies f(\alpha|-) &\propto \text{kernel of a Beta}(a_{\alpha(\text{post})}, b_{\alpha(\text{post})}) \text{ with} \\ a_{\alpha(\text{post})} &= a_\alpha + \sum_{i=1}^{n}\sum_{t=1}^{T}\gamma_{it} \quad b_{\alpha(\text{post})} = b_\alpha + nT - \sum_{i=1}^{n}\sum_{t=1}^{T}\gamma_{it} \end{aligned} \tag{A.13}$$

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

$$\begin{aligned} f(\alpha_t|-) &\propto f(\alpha_t) f((\gamma_{1t}, \ldots, \gamma_{nt})|\alpha_t) \\ &\propto \alpha_t^{a_\alpha - 1}(1-\alpha_t)^{b_\alpha - 1} \prod_{i=1}^{n} \alpha_t^{\gamma_{it}}(1-\alpha_t)^{1-\gamma_{it}} \\ &= \alpha_t^{(a_\alpha + \sum_{i=1}^{n}\gamma_{it}) - 1}(1-\alpha_t)^{(b_\alpha + n - \sum_{i=1}^{n}\gamma_{it}) - 1} \\ \implies f(\alpha_t|-) &\propto \text{kernel of a Beta}(a_{\alpha(\text{post})}, b_{\alpha(\text{post})}) \text{ with} \\ a_{\alpha(\text{post})} &= a_\alpha + \sum_{i=1}^{n}\gamma_{it} \quad b_{\alpha(\text{post})} = b_\alpha + n - \sum_{i=1}^{n}\gamma_{it} \end{aligned} \tag{A.14}$$

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

$$\begin{aligned} f(\alpha_i|-) &\propto f(\alpha_i) f((\gamma_{i1}, \ldots, \gamma_{iT})|\alpha_i) \\ &\propto \alpha_i^{a_{\alpha i} - 1}(1-\alpha_i)^{b_{\alpha i} - 1} \prod_{t=1}^{T} \alpha_i^{\gamma_{it}}(1-\alpha_i)^{1-\gamma_{it}} \\ &= \alpha_{it}^{(a_{\alpha i} + \sum_{t=1}^{T}\gamma_{it}) - 1}(1-\alpha_{it})^{(b_{\alpha i} + T - \sum_{t=1}^{T}\gamma_{it}) - 1} \\ \implies f(\alpha_i|-) &\propto \text{kernel of a Beta}(a_{\alpha(\text{post})}, b_{\alpha(\text{post})}) \text{ with} \\ a_{\alpha(\text{post})} &= a_{\alpha i} + \sum_{t=1}^{T}\gamma_{it} \quad b_{\alpha(\text{post})} = b_{\alpha i} + T - \sum_{t=1}^{T}\gamma_{it} \end{aligned} \tag{A.15}$$

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

$$\begin{aligned} f(\alpha_{it}|-) &\propto f(\alpha_{it}) f(\gamma_{it}|\alpha_{it}) \\ &\propto \alpha_{it}^{a-1}(1-\alpha_{it})^{b-1}\alpha_{it}^{\gamma_{it}}(1-\alpha_{it})^{1-\gamma_{it}} \\ &= \alpha_i^{(a+\gamma_{it})-1}(1-\alpha_i)^{(b+1-\gamma_{it})-1} \\ \implies f(\alpha_{it}|-) &\propto \text{kernel of a Beta}(a_{\alpha(\text{post})}, b_{\alpha(\text{post})}) \text{ with} \\ a_{\alpha(\text{post})} &= a_{\alpha i} + \gamma_{it} \quad b_{\alpha(\text{post})} = b_{\alpha i} + 1 - \gamma_{it} \end{aligned} \tag{A.16}$$

- 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 $t = 1$:

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

$$
\implies f(Y_{it}|-) \propto \text{kernel of a } \mathcal{N}(\mu_{Y_{it}(\text{post})}, \sigma^2_{Y_{it}(\text{post})}) \text{ with}
$$

$$
\begin{aligned}
\sigma^2_{Y_{it}(\text{post})} &= \frac{1}{\frac{1}{\sigma^{2\star}_{c_{it}t}} + \frac{\eta_{1i}^2}{2\sigma^{2\star}_{c_{it+1}t+1}(1-\eta_{1i}^2)}} \\
\mu_{Y_{it}(\text{post})} &= \sigma^2_{Y_{it}(\text{post})} \left(\frac{\mu^{\star}_{c_{it}t} + \boldsymbol{x}_{it}^T \boldsymbol{\beta}_t}{\sigma^{2\star}_{c_{it}t}} + \frac{\eta_{1i}(Y_{it+1} - \mu^{\star}_{c_{it+1}t+1} - \boldsymbol{x}_{it+1}^T \boldsymbol{\beta}_{t+1})}{\sigma^{2\star}_{c_{it+1}t+1}(1-\eta_{1i}^2)} \right)
\end{aligned}
\tag{A.17}
$$

for $1 < t < T$:

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

$$
\implies f(Y_{it}|-) \propto \text{kernel of a } \mathcal{N}(\mu_{Y_{it}(\text{post})}, \sigma^2_{Y_{it}(\text{post})}) \text{ with}
$$

$$\sigma^2_{Y_{it}(\text{post})} = \frac{1-\eta_{1i}^2}{\frac{1}{\sigma^{2\star}_{c_{it}t}} + \frac{\eta_{1i}^2}{\sigma^{2\star}_{c_{it+1}t+1}}}$$
$$\mu_{Y_{it}(\text{post})} = \sigma^2_{Y_{it}(\text{post})} \left(\frac{\mu^\star_{c_{it}t} + \eta_{1i} Y_{it-1} + \boldsymbol{x}_{it}^T \boldsymbol{\beta}_t}{\sigma^{2\star}_{c_{it}t}(1-\eta_{1i}^2)} + \frac{\eta_{1i}(Y_{it+1} - \mu^\star_{c_{it+1}t+1} - \boldsymbol{x}_{it+1}^T \boldsymbol{\beta}_{t+1})}{\sigma^{2\star}_{c_{it+1}t+1}(1-\eta_{1i}^2)} \right) \tag{A.18}$$

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

Appendix B

Computational details

B.1 Implementation of the MCMC algorithm

We now present the code from the `MCMC_fit` function which implements the JDRPM algorithm. We report exclusively the functional part of the implementation, omitting setup lines related to function definitions, variable preallocations, and input argument checks. This inclusion allows readers to appreciate the ease, clarity, and elegance of the Julia language in translating the mathematical formulations into code. As such, we hope that Julia will emerge as the natural choice in the statistical and scientific computing fields, offering them a refreshing approach.

In addition to the points discussed in Chapter 2, we note that the productivity granted by the Julia language is not only derived from its extensive ecosystem of packages and documentation, but also from an active online forum (`https://discourse.julialang.org`), where I personally posed some questions and received valuable answers during the development of this thesis.

Listing 3: Julia code that implements JDRPM's MCMC algorithm.

```
############# 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)
   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,lC)
            spatial_cohesion!(spatial_cohesion_idx, s1n, s2n,
            ↪  sp_params_struct, true, M_dp, S,2,false,lC)
        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,lC)
    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
            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 subract 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)

      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
            lPP[1] += log_Mdp + lgamma(nh_tmp[kk])
         end

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

         # 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 subract 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
```

```
############# 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 MvNormal and the inversion required
      # Am1_star = inv(A_star)
      # beta_iter[t] = rand(MvNormal(inv(Symmetric(A_star))*b_star,
      ↪  inv(Symmetric(A_star))))
   else
      sum_Y = zeros(p_lk)
      A_star = I(p_lk)/s2_beta
      for j in 1:n
         X_jt = @view Xlk_covariates[j,:,t]
         sum_Y += (Y[j,t] - muh_iter[Si_iter[j,t],t] -
         ↪  eta1_iter[j]*Y[j,t-1]) * X_jt / sig2h_iter[Si_iter[j,t],t]
         A_star += (X_jt * X_jt') / sig2h_iter[Si_iter[j,t],t]
      end
      b_star = beta0/s2_beta + sum_Y
      A_star = Symmetric(A_star)
      beta_iter[t] = rand(MvNormalCanon(b_star, A_star))
   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)))

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

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

else # 1<t<T
```

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

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

    ############# update tau2 #############
    kt = nclus_iter[t]
    aux1 = 0.0
    for k in 1:kt
        aux1 += (muh_iter[k,t] - theta_iter[t])^2
    end
    a_star_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,:,t), beta_iter[t])
            ↪  : 0.0

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

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

      i_out += 1
   end

next!(progresso)

end # for i in 1:draws

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

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

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

```
WAIC *= -2
println("WAIC: ", WAIC, " (the lower the better)")

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

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

We now provide some technical details regarding the overall implementation design. The fitting algorithm was written in Julia, but its primary intended use is within the R programming environment. This choice reflects R's current status as the leading language for statistical analysis.

To achieve this integration, we relied on the `JuliaConnectoR` library (Lenz et al., 2022) in R, which enables interaction between the two languages. This library allows to load the Julia project `JDRPM`, which contains the functionalities, including code and package dependencies, required for the implementation of the JDRPM algorithm. Through `JuliaConnectoR` we can pass data and parameters from R to Julia, run the fitting algorithm, and convert the output back into R structures. The design of this workflow is illustrated in Listing 4.

Listing 4: Overview of the R and Julia integration process for running the JDRPM algorithm.

```
############# Requirements #############
# install the required pacakge
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 depdendencies
juliaEval("Pkg.instantiate()")
# now, as a check, this should print the list of packages that JDRPM uses,
# such as Distributions, Statistics, LinearAlgebra, SpecialFunctions, etc.
juliaEval("Pkg.status()")

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

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

# 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
```

In terms of the user interface and feedback provided by the function, we aimed to enhance the friendliness and informativeness compared to the original C implementation. Notable improvements include the ability to perform convergence

diagnostics, on a subset of the sampled parameters, and the real-time progress monitoring, which updates every second, to display the estimated remaining time for completing the fit. These features further exemplify the ease of use inherent in Julia: they were implemented with just a few additional lines of code, leveraging the Julia packages `MCMCChains` (Ge et al., 2018) 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(μ=0.0, σ=10.0)
lambda2 ∼ invGamma(1.9, 0.4)
alpha ∼ Beta(2.0, 2.0)

- using seed 111.0 -
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)
[✓] with covariates in the likelihood? true (p=6)
[✓] with covariates in the clustering process? true (p=3, similarity 4.0)
[✓] are there missing data in Y? true

2024-10-30 13:50:23
Starting MCMC algorithm
Progress 100% Time: 1:15:45 (41.33 ms/it)

done!
Elapsed time 1 hour, 15 minutes, 45 seconds, 920 milliseconds
LPML: 791.8603234076745 (the higher the better)
WAIC: -1976.7279705951883 (the lower the better)
acceptance ratio eta1: 52.18%
acceptance ratio phi1: 35.04%

# 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
```

All codes and insights related to JDRPM's MCMC algorithm implementation, along with the experiments conducted throughout this work, are available at `https://github.com/federicomor/Tesi/tree/main/src/JDRPM`.

Appendix C

Further plots

We now present the visualization of the clusters estimates for the fits analysed in the experiments of Chapter 3.

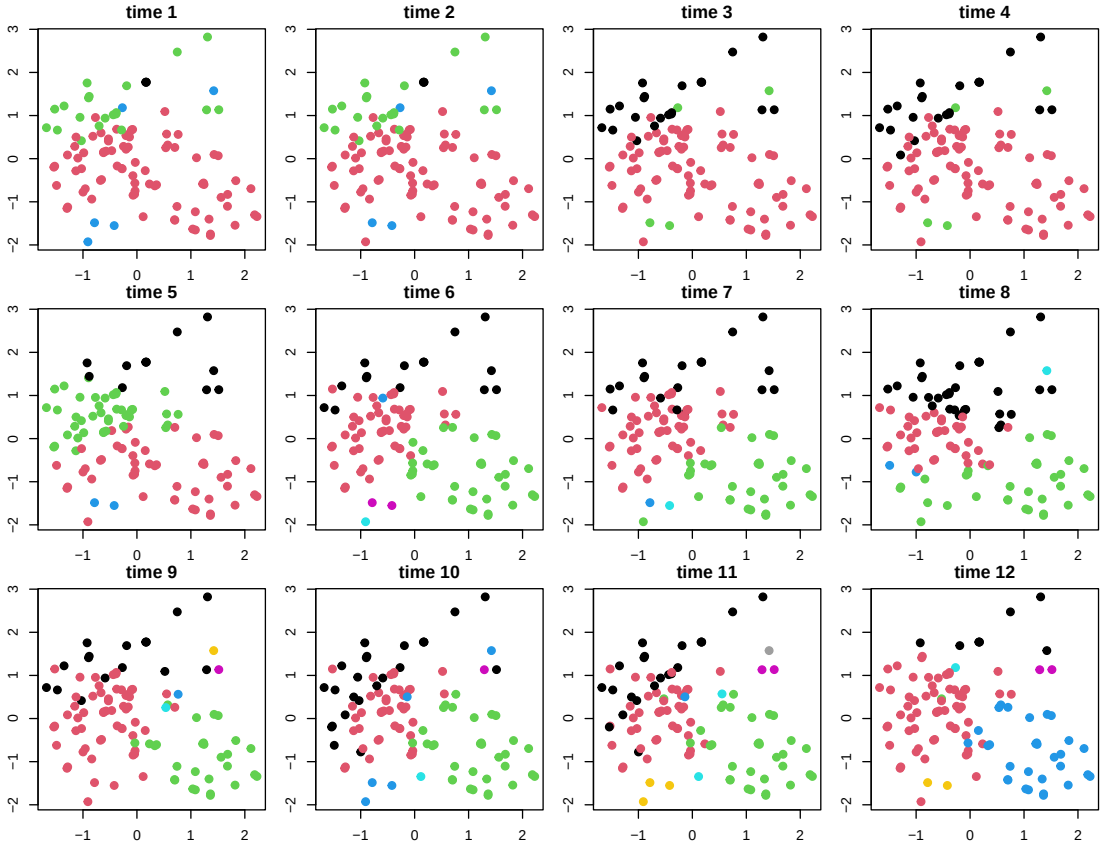


Figure C.1: Clusters estimates produced by CDRPM fit, in the real-world scenario.

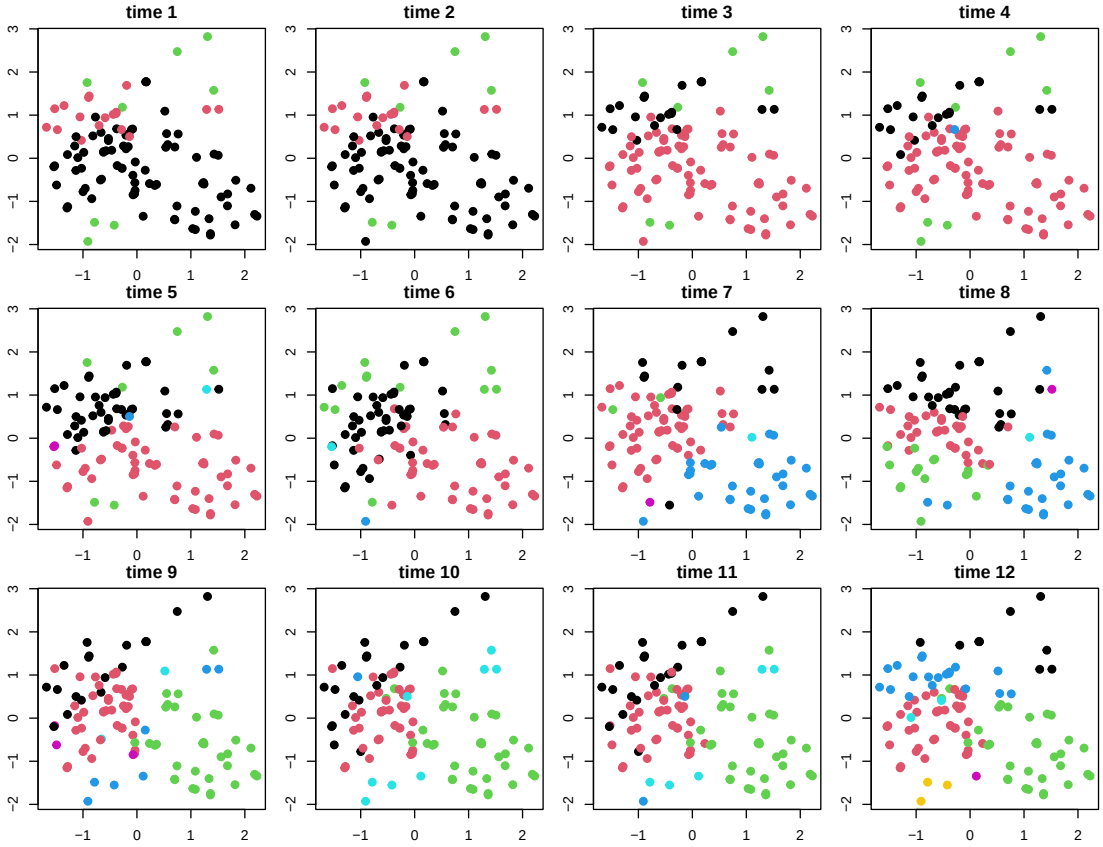


Figure C.2: Clusters estimates produced by JDRPM fit, in the real-world scenario.

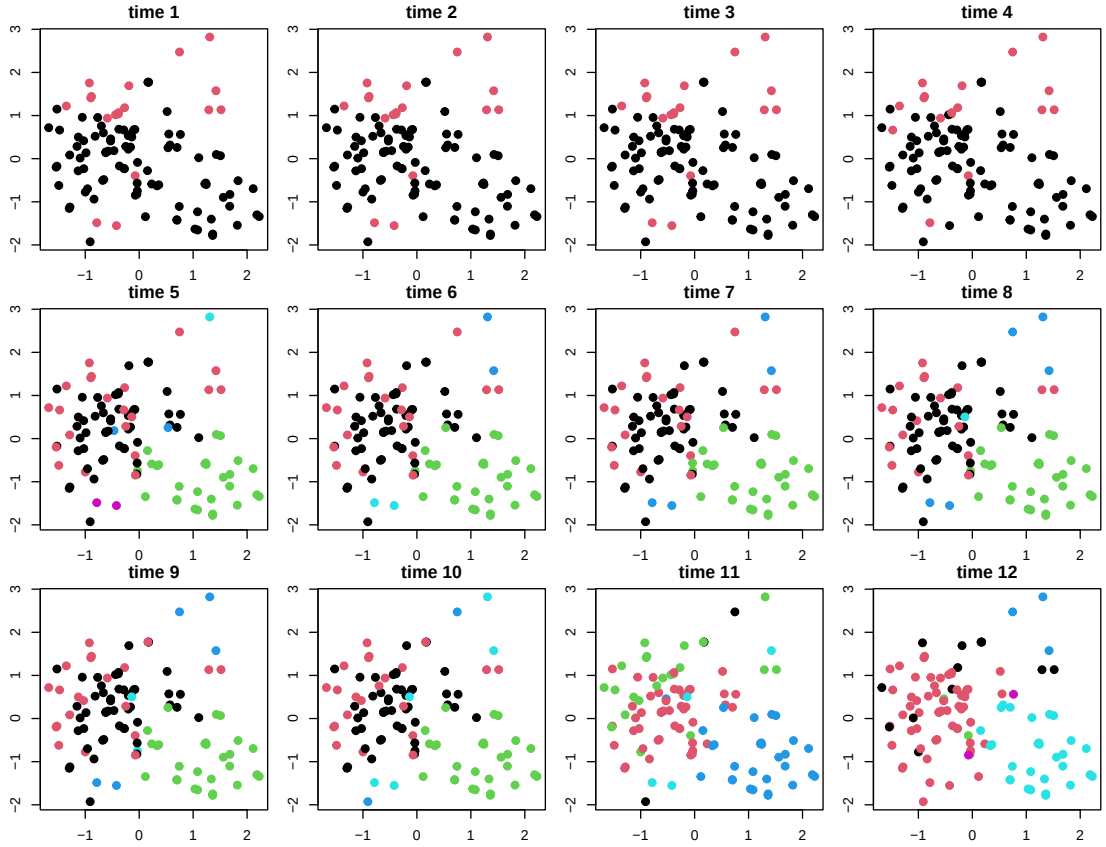


Figure C.3: Clusters estimates produced by JDRPM fit, in the real-world scenario, with covariates in the likelihood.

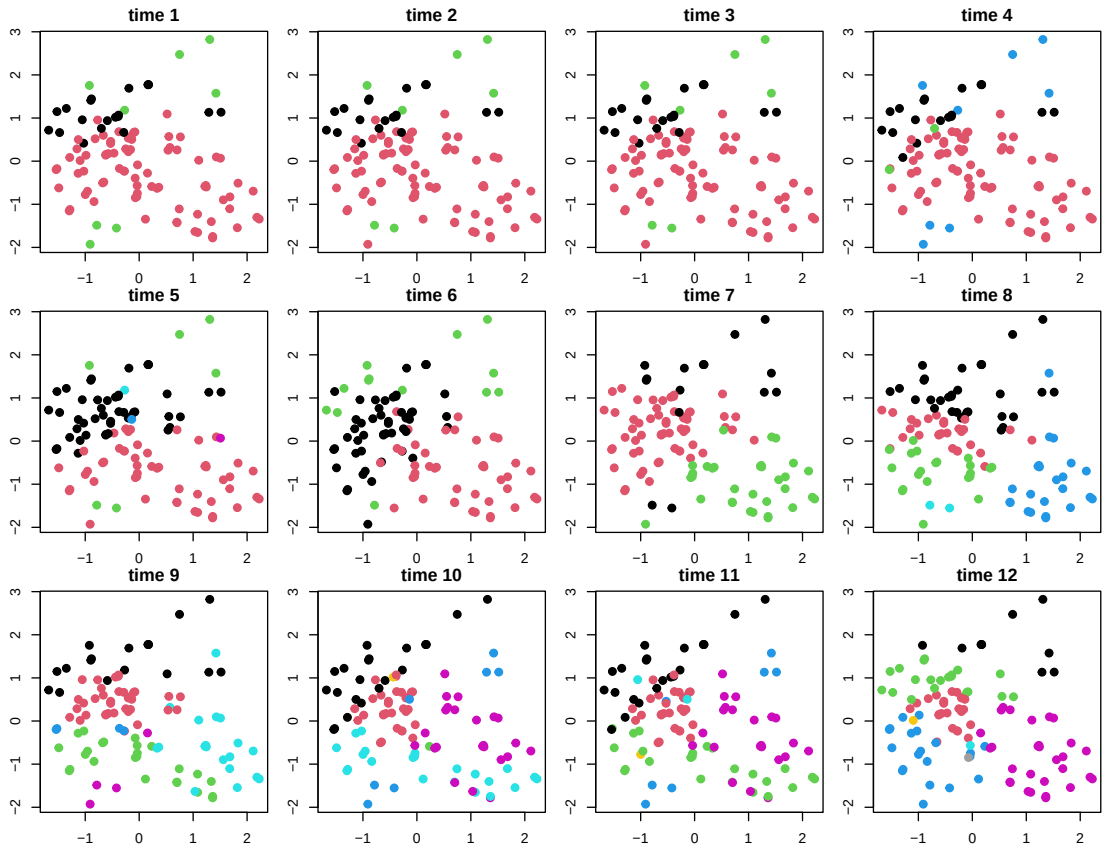


Figure C.4: Clusters estimates produced by JDRPM fit, in the real-world scenario, with covariates in the prior.

Bibliography

Aldous, David J. (1985). "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, pp. 1–198. ISBN: 978-3-540-39316-0 (cit. on p. 3).

Antoniano Villalobos, Isadora and Stephen Walker (Aug. 2015). "A Nonparametric Model for Stationary Time Series". In: *Journal of Time Series Analysis* 63. DOI: 10.1111/jtsa.12146 (cit. on p. 5).

Barry, Daniel and J. A. Hartigan (1993). "A Bayesian Analysis for Change Point Problems". In: *Journal of the American Statistical Association* 88.421, pp. 309–319. ISSN: 01621459, 1537274X. URL: http://www.jstor.org/stable/2290726 (cit. on p. 12).

Besançon, Mathieu, Theodore Papamarkou, David Anthoff, Alex Arslan, Simon Byrne, Dahua Lin, and John Pearson (2021). "Distributions.jl: Definition and Modeling of Probability Distributions in the JuliaStats Ecosystem". In: *Journal of Statistical Software* 98.16, pp. 1–30. ISSN: 1548-7660. DOI: 10.18637/jss.v098.i16 (cit. on p. 23).

Bezanson, Jeff, Alan Edelman, Stefan Karpinski, and Viral B Shah (2017). "Julia: A fresh approach to numerical computing". In: *SIAM Review* 59.1, pp. 65–98. DOI: 10.1137/141000671 (cit. on p. 23).

Blackwell, David and James B. MacQueen (1973). "Ferguson Distributions Via Polya Urn Schemes". In: *The Annals of Statistics* 1.2, pp. 353–355. DOI: 10.1214/aos/1176342372 (cit. on p. 3).

Böhning, Dankmar (Aug. 2007). "Finite Mixture and Markov Switching Models by S. Frühwirth-Schnatter". In: *Biometrics* 63.3, pp. 971–972. ISSN: 0006-341X. DOI: 10.1111/j.1541-0420.2007.00856_6.x. eprint: https://academic.oup.com/biometrics/article-pdf/63/3/971/52454335/biometrics_63_3_971.pdf (cit. on p. 2).

Bouveyron, Charles, Gilles Celeux, T. Brendan Murphy, and Adrian E. Raftery (2019). *Model-Based Clustering and Classification for Data Science: With Applications in R*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press (cit. on p. 2).

Caron, François, Willie Neiswanger, Frank Wood, Arnaud Doucet, and Manuel Davy (Jan. 2017). "Generalized Pólya urn for time-varying Pitman-Yor processes". In: *Journal of Machine Learning Research* 18.1, pp. 836–867. ISSN: 1532-4435 (cit. on p. 5).

Chen, Jiahao and Jarrett Revels (Aug. 2016). "Robust benchmarking in noisy environments". In: *arXiv e-prints*, arXiv:1608.04295. arXiv: 1608.04295 [cs.PF] (cit. on p. 25).

Christensen, Ronald, Wesley Johnson, Adam Branscum, and Timothy Hanson (2010). *Bayesian ideas and data analysis. An introduction for scientists and statisticians*. CRC Press. DOI: 10.1201/9781439894798 (cit. on p. 12).

Crowley, Evelyn M. (1997). "Product Partition Models for Normal Means". In: *Journal of the American Statistical Association* 92.437, pp. 192–198. ISSN: 01621459. URL: http://www.jstor.org/stable/2291463 (cit. on p. 12).

David B. Dahl, Devin J. Johnson and Peter Müller (2022). "Search Algorithms and Loss Functions for Bayesian Clustering". In: *Journal of Computational and Graphical Statistics* 31.4, pp. 1189–1201. DOI: 10.1080/10618600.2022.2069779 (cit. on p. 31).

De Blasi, Pierpaolo, Stefano Favaro, Antonio Lijoi, Ramses H. Mena, Igor Prunster, and Matteo Ruggiero (Feb. 2015). "Are Gibbs-Type Priors the Most Natural Generalization of the Dirichlet Process?" In: *IEEE Trans. Pattern Anal. Mach. Intell.* 37.2, pp. 212–229. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2013.217 (cit. on p. 15).

De Iorio, Maria, Stefano Favaro, Alessandra Guglielmi, and Lifeng Ye (Oct. 2019). *Bayesian nonparametric temporal dynamic clustering via autoregressive Dirichlet priors*. DOI: 10.48550/arXiv.1910.10443 (cit. on p. 5).

De Iorio, Maria and Athanasios Kottas (2018). "Modeling for Dynamic Ordinal Regression Relationships: An Application to Estimating Maturity of Rockfish in California". In: *Journal of the American Statistical Association* 113.521, pp. 68–80. DOI: 10.1080/01621459.2017.1328357 (cit. on p. 5).

Denison, D. and C Holmes (Apr. 2001). "Bayesian Partitioning for Estimating Disease Risk". In: *Biometrics* 57, pp. 143–9. DOI: 10.1111/j.0006-341X.2001.00143.x (cit. on p. 14).

Duncan, Earl (2016). *Deriving the Full Conditionals*. BRAG: Bayesian Research & Application Group. URL: https://bragqut.wordpress.com/wp-content/uploads/2018/04/deriving-the-full-conditionals.pdf (cit. on p. 9).

Ester, Martin, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu (1996). "A density-based algorithm for discovering clusters in large spatial databases with noise". In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD'96. Portland, Oregon: AAAI Press, pp. 226–231 (cit. on p. 1).

Fassò, A., J. Rodeschini, A. Fusta Moro, Q. Shaboviq, P. Maranzano, M. Cameletti, F. Finazzi, N. Golini, R. Ignaccolo, and P. Otto (2023). *AgrImOnIA: Open Access dataset correlating livestock and air quality in the Lombardy region, Italy (3.0.0)*. DOI: https://doi.org/10.5281/zenodo.7956006 (cit. on pp. v, vii, 4, 34, 39).

Ferguson, Thomas S. (1973). "A Bayesian Analysis of Some Nonparametric Problems". In: *The Annals of Statistics* 1.2, pp. 209–230. DOI: 10.1214/aos/1176342360 (cit. on p. 3).

Franzén, Jessica (2008). "Bayesian Cluster Analysis : Some Extensions to Non-standard Situations". In: URL: https://api.semanticscholar.org/CorpusID:12397258 (cit. on p. 2).

Ge, Hong, Kai Xu, and Zoubin Ghahramani (2018). "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*, pp. 1682–1690. URL: http://proceedings.mlr.press/v84/ge18b.html (cit. on p. 92).

Gelman, A., J.B. Carlin, H.S. Stern, and D.B. Rubin (2003). *Bayesian Data Analysis, Second Edition*. Chapman & Hall/CRC Texts in Statistical Science. Taylor & Francis. ISBN: 9781420057294. DOI: 10.1201/9780429258480 (cit. on p. 3).

Gelman, Andrew (Jan. 2004). "Prior Distributions for Variance Parameters in Hierarchical Models". In: *Economics and Econometrics Research Institute (EERI), EERI Research Paper Series* 1 (cit. on p. 64).

Gelman, Andrew, Jessica Hwang, and Aki Vehtari (July 2013). "Understanding predictive information criteria for Bayesian models". In: *Statistics and Computing* 24. DOI: 10.1007/s11222-013-9416-2 (cit. on p. 12).

Gormley, Isobel, Thomas Murphy, and Adrian Raftery (Oct. 2022). "Model-Based Clustering". In: *Annual Review of Statistics and Its Application* 10. DOI: 10.1146/annurev-statistics-033121-115326 (cit. on p. 1).

Gower, J. C. (1971). "A General Coefficient of Similarity and Some of Its Properties". In: *Biometrics* 27.4, pp. 857–871. ISSN: 0006341X, 15410420. URL: http://www.jstor.org/stable/2528823 (cit. on p. 19).

Grazian, Clara (Mar. 2023). "A review on Bayesian model-based clustering". In: DOI: 10.48550/arXiv.2303.17182 (cit. on pp. 2, 3).

Grün, Bettina (July 2018). *Model-based Clustering*. DOI: 10.48550/arXiv.1807.01987 (cit. on p. 2).

Gutiérrez, Luis, Ramsés H. Mena, and Matteo Ruggiero (2016). "A time dependent Bayesian nonparametric model for air quality analysisa". In: *Computational Statistics & Data Analysis* 95.C, 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).

Hartigan, J. A. and M. A. Wong (1979). "Algorithm AS 136: A K-Means Clustering Algorithm". In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28.1, pp. 100–108. ISSN: 00359254, 14679876. URL: http://www.jstor.org/stable/2346830 (visited on 11/04/2024) (cit. on p. 1).

Hartigan, J.A. (1990). "Partition models". In: *Communications in Statistics - Theory and Methods* 19.8, pp. 2745–2756. DOI: 10.1080/03610929008830345 (cit. on p. 12).

Hubert, Lawrence J. and Phipps Arabie (1985). "Comparing partitions". In: *Journal of Classification* 2, pp. 193–218. URL: https://api.semanticscholar.org/CorpusID:189915041 (cit. on p. 31).

Jain, Anil K. and Richard C. Dubes (1988). *Algorithms for clustering data*. Prentice-Hall, Inc. URL: http://portal.acm.org/citation.cfm?id=46712 (cit. on p. 1).

Jo, Seongil, Jaeyong Lee, Peter Müller, Fernando A. Quintana, and Lorenzo Trippa (2017). "Dependent Species Sampling Models for Spatial Density Estimation". In: *Bayesian Analysis* 12, pp. 379–406. URL: https://api.semanticscholar.org/CorpusID:52028880 (cit. on p. 5).

Kalli, Maria and Jim Griffin (Jan. 2018). "Bayesian nonparametric vector autoregressive models". In: *Journal of Econometrics* 203. DOI: 10.1016/j.jeconom.2017.11.009 (cit. on p. 5).

Kaufman, Leonard and Peter Rousseeuw (Jan. 1990). *Finding Groups in Data: An Introduction To Cluster Analysis*. ISBN: 0-471-87876-6. DOI: 10.2307/2532178 (cit. on p. 1).

Krige, Daniel G (1951). "A statistical approach to some basic mine valuation problems on the Witwatersrand". In: *Journal of the Southern African Institute of Mining and Metallurgy* 52.6, pp. 119–139 (cit. on p. 54).

Lawson, C. L., R. J. Hanson, D. R. Kincaid, and F. T. Krogh (Sept. 1979). "Basic Linear Algebra Subprograms for Fortran Usage". In: *ACM Trans. Math. Softw.* 5.3, pp. 308–323. ISSN: 0098-3500. DOI: 10.1145/355841.355847 (cit. on p. 23).

Lenz, Stefan, Maren Hackenberg, and Harald Binder (2022). "The JuliaConnectoR: A Functionally-Oriented Interface for Integrating Julia in R". In: *Journal of Statistical Software* 101.6, pp. 1–24. DOI: 10.18637/jss.v101.i06 (cit. on pp. 31, 90).

Lin, Dahua, John Myles White, Simon Byrne, Douglas Bates, Andreas Noack, John Pearson, Alex Arslan, Kevin Squire, David Anthoff, Theodore Papamarkou, Mathieu Besançon, Jan Drugowitsch, Moritz Schauer, and other contributors (July 2019). *JuliaStats/Distributions.jl: a Julia package for probability distributions and associated functions*. DOI: 10.5281/zenodo.2647458 (cit. on p. 23).

McLachlan, Geoffrey J and Thriyambakam Krishnan (2008). *The EM algorithm and extensions*. John Wiley & Sons. DOI: 10.1002/9780470191613 (cit. on p. 2).

Mozdzen, Alexander, Andrea Cremaschi, Annalisa Cadonna, Alessandra Guglielmi, and Gregor Kastner (2022). "Bayesian modeling and clustering for spatio-temporal areal data: An application to Italian unemployment". In: *Spatial Statistics* 52, p. 100715. ISSN: 2211-6753. DOI: https://doi.org/10.1016/j.spasta.2022.100715. URL: https://www.sciencedirect.com/science/article/pii/S2211675322000768 (cit. on p. 3).

Müller, Peter, Fernando Quintana, and Gary Rosner (Mar. 2011). "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, pp. 260–278. DOI: 10.1198/jcgs.2011.09066 (cit. on p. 15).

Ng, See, Thriyambakam Krishnan, and G. Mclachlan (Jan. 2004). "The EM algorithm". In: *Handbook of Computational Statistics: Concepts and Methods*. DOI: 10.1007/978-3-642-21551-3_6 (cit. on p. 2).

Nieto-Barajas, Luis E., Peter Müller, Yuan Ji, Yiling Lu, and Gordon B. Mills (Jan. 2012). "A Time-Series DDP for Functional Proteomics Profiles". In: *Biometrics* 68.3, pp. 859–868. ISSN: 0006-341X. DOI: 10.1111/j.1541-0420.2011.01724.x. eprint: https://academic.oup.com/biometrics/article-pdf/68/3/859/53245075/biometrics_68_3_859.pdf (cit. on p. 5).

Page, Garrit L., Fernando A. Quintana, and David B. Dahl (2022). "Dependent Modeling of Temporal Sequences of Random Partitions". In: *Journal of Computational and Graphical Statistics* 31.2, pp. 614–627. DOI: 10.1080/10618600.2021.1987255 (cit. on pp. v, vii, 4, 5, 9, 32, 34).

Page, Garritt and Fernando Quintana (Sept. 2018). "Calibrating covariate informed product partition models". In: *Statistics and Computing* 28, pp. 1–23. DOI: 10.1007/s11222-017-9777-z (cit. on pp. 18, 19).

— (Apr. 2015). "Spatial Product Partition Models". In: *Bayesian Analysis* 11. DOI: 10.1214/15-BA971 (cit. on pp. 14, 22).

Quintana, Fernando, Peter Müller, and Ana Luisa Papoila (Mar. 2015). "Cluster-Specific Variable Selection for Product Partition Models". In: *Scandinavian Journal of Statistics* 42. DOI: 10.1111/sjos.12151 (cit. on p. 15).

Quintana, Fernando A., Peter Müller, and Ana Luisa Papoila (2015). "Cluster-Specific Variable Selection for Product Partition Models". In: *Scandinavian Journal of Statistics* 42.4, pp. 1065–1077. DOI: https://doi.org/10.1111/sjos.12151 (cit. on p. 56).

R Core Team (2024). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. URL: https://www.R-project.org/ (cit. on p. 31).

Robert, Christian and George Casella (Nov. 2000). "Monte Carlo Statistical Method". In: *Technometrics* 42. DOI: 10.2307/1270959 (cit. on p. 3).

Teh, Yee Whye (2010). "Dirichlet Process". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, pp. 280–287. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_219 (cit. on p. 3).

Wade, Sara (Mar. 2023). "Bayesian cluster analysis". In: *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences* 381, p. 20220149. DOI: 10.1098/rsta.2022.0149 (cit. on p. 2).

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 a 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 and supported me 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 through 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 I had the delight of working and playing with in the various summer camps, for consistently reminding me of the cheerful and mild spirit with which all the various challenges can be faced.

And finally, I thank all the cats and kittens that I met, also quite literally, during my journey, which through a tender meow or a kind rub have always managed to enlighten the various days.

Ringraziamenti

> *"Ecco il mio ponte d'approdo per molti lunghi anni, il mio angoletto, nel quale faccio il mio ingresso con una sensazione così diffidente, così morbosa... Ma chi lo sa? Forse, quando tra molti anni mi toccherà abbandonarlo, magari potrei anche rimpiangerlo!"*
>
> — Fëdor Dostoevskij, *Memorie da una casa di morti*

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 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 e supportato 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.

Infine, ringrazio tutti i gatti e micini che ho incontrato, anche letteralmente, durante il mio percorso, che con un tenero miagolio o un'affettuosa strusciatina sono sempre riusciti a migliorare le varie giornate.