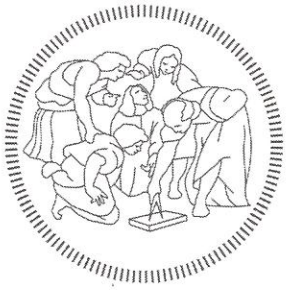

Politecnico di Milano

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

MASTER THESIS

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

Advisor
Prof. Alessandra Guglielmi

Coadvisor
Prof. Alessandro Carminati

Candidate
Federico Angelo Mor
Matr. 221429

Academic Year 2023–2024

to my cats Otto
and La Micia

However, the current formulation of the model and the implementation of the associated MCMC lock

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. To this end, the Dependent Random Partition Model (~~DRPM~~) (Page et al. ...) is one of the most relevant Bayesian models due to its explicit consideration of temporal dependence in the partitions. ~~However, the current implementation~~ lacks of the MCMC 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 model by addressing those issues trough updates on the model formulation and a brand new implementation in Julia. These advancements are then tested on synthetic and real-world datasets, including air quality data from the AgrImOnIA project in Lombardy, Italy.

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

Sommario

Vedi versione in INGLESE

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. In tal senso, il modello DRPM (Dependent Random Partition Model, modello per partizioni aleatorie dipendenti) è uno dei modelli bayesiani più rilevanti in quanto tiene conto in modo esplicito della dipendenza temporale delle partizioni. Tuttavia, l'attuale implementazione manca dell'inclusione di covariate, della gestione dei dati mancanti, e di efficienza nei tempi di esecuzione. In questo lavoro abbiamo quindi migliorato l'originale modello 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.

PAROLE CHIAVE: modello bayesiano, clustering, dati spazio-temporali, statistica computazionale

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Clustering has always been a powerful tool to identify structures and patterns in data especially in contexts where relationships between the observations are complex, e.g. when the target variable is affected by many factors simultaneously. For this reason, clustering techniques saw a continuos increase in popularity in a variety of scientific fields, including social sciences, climate and environmental analysis, economics, and healthcare. The importance of clustering becomes even more noticeable when working on spatio-temporal datasets, in which observations are collected over time and across different spatial locations, possibly concealing trends behind both information levels. This type of data, in fact, is inherently complex due to this dependence and interaction between spatial and temporal dimensions; a complexity that is further increased if covariates are also available. For that reason, an effective analysis of such data demands models that can account for this dependence while also providing efficient implementations to be possibly applied on large scale datasets, which are commonly accessible in this context.

Recently, the use of Bayesian methods to perform clustering has gained some attention, 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 developed, but one of the most relevant to this end is the Dependent Random Partition Model (DRPM), 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, written in C and available trough 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 e.g. 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 enlarging the original model, that is, preserving the primary idea of the formulation but making it richer trough the insertion of new components. We will show how this updated model can perform better than the original one, under the same testing conditions, and can also

improve the clustering accuracy and interpretation trough the inclusion of covariates. All this while also providing 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 datasets and real-world applications, with the latter involving air quality measurements from the AgrImOnIA dataset, a comprehensive record of air pollutant levels and other environmental variables measured across the Lombardy region of Italy.

Chapter 2 will briefly review the literature about bayesian clustering models and then dive deeply into the analysis and description of DRPM, and of our updated version, about their core aspects of sampling algorithm, spatial cohesions, and covariates similarities.

Chapter 3 will provide some insights about the computational aspects of the model implementation, motivating the choice of the Julia language and reporting the optimizations possibilities emerged when developing the algorithm.

Chapter 4 will be devoted to test and compare the original DRPM formulation and implementation to our updated version. We will check if they perform similarly, at a common testing level, and assess the performances of our model when we employ the new updates, i.e. the handling of missing data and the insertion of covariates at clustering and likelihood levels. An analysis about expected execution times with respect to the size of the dataset will also be provided.

Finally, in Chapter 5, we will briefly review the benefits and drawbacks that this work revealed and suggest possible further improvements or development paths.

Chapter 2

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

Before diving into the model description, we define some notation conventions. We setup in a spatio-temporal context with $i = 1, \dots, n$ and $t = 1, \dots, T$ being the indexes for units and time instants. We will denote with $\rho_t = \{S_{1t}, \dots, S_{k_t t}\}$ the partition at time t, of the n experimental units, composed by k_t cluster. Another possible representation of the partition is trough 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

$$P(\rho_1, \dots, \rho_T) = P(\rho_T|\rho_{T-1}) \cdots P(\rho_2|\rho_1)P(\rho_1) \tag{2.1}$$

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[1]

$$\gamma_{it} = \begin{cases} 1 & \text{if unit } i \text{ is } \textit{not} \text{ reallocated when moving from time } t-1 \text{ to } t \\ 0 & \text{otherwise (i.e. unit } \textit{is} \text{ reallocated)} \end{cases} \tag{2.2}$$

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}. 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

$$P(\boldsymbol{\gamma}_1, \rho_1, \dots, \boldsymbol{\gamma}_T, \rho_T) = P(\rho_T|\boldsymbol{\gamma}_T, \rho_{T-1})P(\boldsymbol{\gamma}_T) \cdots P(\rho_2|\boldsymbol{\gamma}_2, \rho_1)P(\boldsymbol{\gamma}_2)P(\rho_1) \tag{2.3}$$

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

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

The sampling algorithm requires that when we are drawing the new samples for the γ_{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

[1] a quick way to remind this convention is thinking that γ_{it} answers to the question "do I stay fixed?" asked from unit's i perspective.

$\gamma_{it}^{(d-1)} = 0$ 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 trough the model, an autoregressive AR(1) component is also added to the formulation of the model (but only optionally in the implementation), both at the data and cluster-specific parameters level. All this led the authors to the following complete model

$$
\begin{aligned}
Y_{it} | Y_{it-1}, \boldsymbol{\mu}_t^\star, \boldsymbol{\sigma}_t^{2\star}, \boldsymbol{\eta}, \boldsymbol{c}_t &\overset{\text{ind}}{\sim} \mathcal{N}(\mu^\star_{c_{it}t} + \eta_{1i} Y_{it-1}, \sigma^{2\star}_{c_{it}t}(1-\eta_{1i}^2)) \\
Y_{i1} &\overset{\text{ind}}{\sim} \mathcal{N}(\mu^\star_{c_{i1}1}, \sigma^{2\star}_{c_{i1}1}) \\
\xi_i = \text{Logit}(\tfrac{1}{2}(\eta_{1i}+1)) &\overset{\text{ind}}{\sim} \text{Laplace}(a, b) \\
(\mu^\star_{jt}, \sigma^\star_{jt}) &\overset{\text{ind}}{\sim} \mathcal{N}(\vartheta_t, \tau_t^2) \times \mathcal{U}(0, A_\sigma) \\
\vartheta_t | \vartheta_{t-1} &\overset{\text{ind}}{\sim} \mathcal{N}((1-\varphi_1)\varphi_0 + \varphi_1 \vartheta_{t-1}, \lambda^2(1-\varphi_1^2)) \\
(\vartheta_1, \tau_t) &\overset{\text{iid}}{\sim} \mathcal{N}(\varphi_0, \lambda^2) \times \mathcal{U}(0, A_\tau) \\
(\varphi_0, \varphi_1, \lambda) &\sim \mathcal{N}(m_0, s_0^2) \times \mathcal{U}(-1,1) \times \mathcal{U}(0, A_\lambda) \\
\{\boldsymbol{c}_t, \dots, \boldsymbol{c}_T\} &\sim \text{tRPM}(\boldsymbol{\alpha}, M) \text{ with } \alpha_t \overset{\text{iid}}{\sim} \text{Beta}(a_\alpha, b_\alpha)
\end{aligned}
\tag{2.4}
$$

where tRPM represents the temporal random partition model (2.3).

Moving towards our update, we decided to refine some parts of that formulation. Regarding the variances $\sigma_{jt}^{2\star}$, τ_t^2, and λ^2, we chose to model them trough an inverse gamma distribution rather than the uniform employed originally. This is indeed a more sophisticate choice, since the tuning of the parameters of an invGamma(a, b) is a bit more difficult than simply setting the bounds of a $\mathcal{U}(l, u)$, but should guarantee a better mixing in the chain. In fact, the invGamma distributions recovers conjugacy in the model, thanks to the normal law assigned to the other parameters, allowing the update step of the variances to be performed trough the analytically exact Gibbs sampler rather than the acceptance-rejection method of Metropolis algorithm. Finally, to improve the accuracy in fitting the target values, we added a regression parameter $\boldsymbol{\beta}_t$ 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.

$$
\begin{aligned}
Y_{it}|Y_{it-1}, \boldsymbol{\mu}_t^\star, \boldsymbol{\sigma}_t^{2\star}, \boldsymbol{\eta}, \boldsymbol{c}_t &\overset{\text{ind}}{\sim} \mathcal{N}(\mu^\star_{c_{itt}} + \eta_{1i} Y_{it-1} + \boldsymbol{x}_{it}^T \boldsymbol{\beta}_t, \sigma^{2\star}_{c_{itt}}(1-\eta_{1i}^2)) \\
Y_{i1} &\overset{\text{ind}}{\sim} \mathcal{N}(\mu^\star_{c_{i1}1} + \boldsymbol{x}_{i1}^T \boldsymbol{\beta}_1, \sigma^{2\star}_{c_{i1}1}) \\
\boldsymbol{\beta}_t &\overset{\text{ind}}{\sim} \mathcal{N}_p(\boldsymbol{b}, s^2 I) \\
\xi_i = \text{Logit}(\tfrac{1}{2}(\eta_{1i}+1)) &\overset{\text{ind}}{\sim} \text{Laplace}(a, b) \\
(\mu^\star_{jt}, \sigma^{2\star}_{jt}) &\overset{\text{ind}}{\sim} \mathcal{N}(\vartheta_t, \tau_t^2) \times \text{invGamma}(a_\sigma, b_\sigma) \\
\vartheta_t | \vartheta_{t-1} &\overset{\text{ind}}{\sim} \mathcal{N}((1-\varphi_1)\varphi_0 + \varphi_1 \vartheta_{t-1}, \lambda^2(1-\varphi_1^2)) \\
(\vartheta_1, \tau_t^2) &\overset{\text{iid}}{\sim} \mathcal{N}(\varphi_0, \lambda^2) \times \text{invGamma}(a_\tau, b_\tau) \\
(\varphi_0, \varphi_1, \lambda^2) &\sim \mathcal{N}(m_0, s_0^2) \times \mathcal{U}(-1,1) \times \text{invGamma}(a_\lambda, b_\lambda) \\
\{\boldsymbol{c}_t, \dots, \boldsymbol{c}_T\} &\sim \text{tRPM}(\boldsymbol{\alpha}, M) \text{ with } \alpha_t \overset{\text{iid}}{\sim} \text{Beta}(a_\alpha, b_\alpha)
\end{aligned} \tag{2.5}
$$

A visual representation of this new version of the DRPM model is also present in Figure 2.1, to more clearly appreciate the hierarchical structure and the relations among the parameters.

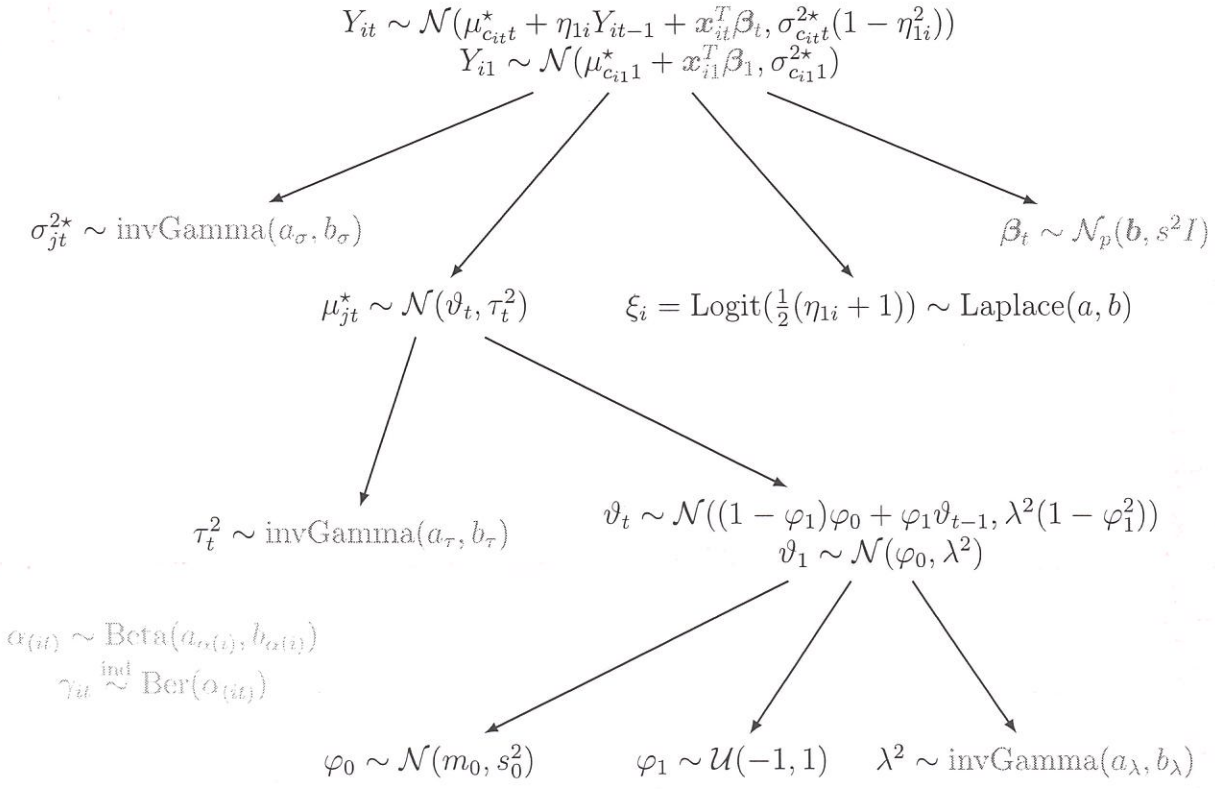

Figure 2.1: Graph visualization of the DRPM model, with highlighted in dark red the changes that we made to the original formulation and in gray the internal variables of the model.

In the course of this work, for the sake of clarity, we will refer to CDRPM for the original model formulation of [PQD22] and to JDRPM for our updated version.

We will now dive more deeply into the characteristics of the models by deriving the update rules for the parameters which will be used to implement the MCMC fit-

ting algorithm and, subsequently, inspecting the behaviours of the spatial cohesions and covariates similarities.

2.1 Update rules derivation

We briefly report the full conditionals derivation for the parameters which had a conjugacy in the model (for the full computations see Appendix A). The other variables not included here, namely η_{1i} and φ_1, involved instead the classical Metropolis update.

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

  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

  $$a_{\tau(\text{post})} = a_\sigma + \frac{|S_{jt}|}{2} \quad b_{\tau(\text{post})} = b_\sigma + \frac{1}{2}\sum_{i \in S_{jt}} (Y_{it} - \mu^\star_{jt} - \eta_{1i}Y_{it-1} - \boldsymbol{x}^T_{it}\boldsymbol{\beta}_t)^2 \tag{2.6}$$

- update $\mu^\star_{jt}$

  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

  $$\sigma^2_{\mu^\star_{jt}(\text{post})} = \frac{1}{\frac{1}{\tau^2_t} + \frac{\sum_{i \in S_{jt}} \frac{1}{1-\eta^2_{1i}}}{\sigma^{2\star}_{jt}}} \quad \mu_{\mu^\star_{jt}(\text{post})} = \sigma^2_{\mu^\star_{jt}(\text{post})}\left(\frac{\vartheta_t}{\tau^2_t} + \frac{\sum_{i \in S_{jt}} \frac{Y_{it} - \eta_{1i}Y_{i,t-1} - \boldsymbol{x}^T_{it}\boldsymbol{\beta}_t}{1-\eta^2_{1i}}}{\sigma^{2\star}_{jt}}\right) \tag{2.7}$$

- update $\boldsymbol{\beta}_t$

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

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

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

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

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

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

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

$$\boldsymbol{h}_{(\text{post})} = \left(\frac{\boldsymbol{b}}{s^2} + \sum_{i=1}^{n} \frac{(Y_{it} - \mu^{\star}_{c_{it}t} - \eta_{1i}Y_{it-1})\boldsymbol{x}_{it}}{\sigma^{2\star}_{c_{it}t}}\right) \qquad J_{(\text{post})} = \left(\frac{1}{s^2}I + \sum_{i=1}^{n} \frac{\boldsymbol{x}_{it}\boldsymbol{x}_{it}^T}{\sigma^{2\star}_{c_{it}t}}\right) \tag{2.8}$$

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 trough 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

$$f(\tau_t^2|-) \propto \text{kernel of a invGamma}(a_{\tau(\text{post})}, b_{\tau(\text{post})}) \text{ with}$$
$$a_{\tau(\text{post})} = \frac{k_t}{2} + a_\tau \qquad b_{\tau(\text{post})} = \frac{\sum_{j=1}^{k_t}(\mu^{\star}_{jt} - \vartheta_t)^2}{2} + b_\tau \tag{2.9}$$

- update ϑ_t

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

$$\sigma^2_{\vartheta_t(\text{post})} = \frac{1}{\frac{1}{\lambda^2} + \frac{\varphi_1^2}{\lambda^2(1-\varphi_1^2)} + \frac{k_t}{\tau_t^2}}$$
$$\mu_{\vartheta_t(\text{post})} = \sigma^2_{\vartheta_t(\text{post})}\left(\frac{\varphi_0}{\lambda^2} + \frac{\varphi_1(\vartheta_{t+1} - (1-\varphi_1)\varphi_0)}{\lambda^2(1-\varphi_1^2)} + \frac{\sum_{j=1}^{k_t}\mu^{\star}_{jt}}{\tau_t^2}\right) \tag{2.10}$$

- update φ_0

$$f(\varphi_0|-) \propto \text{kernel of a } \mathcal{N}(\mu_{\varphi_0(\text{post})}, \sigma^2_{\varphi_0(\text{post})}) \text{ with}$$
$$\sigma^2_{\varphi_0(\text{post})} = \frac{1}{\frac{1}{s_0^2} + \frac{1}{\lambda^2} + \frac{(T-1)(1-\varphi_1)^2}{\lambda^2(1-\varphi_1^2)}}$$
$$\mu_{\varphi_0(\text{post})} = \sigma^2_{\varphi_0(\text{post})} \left(\frac{m_0}{s_0^2} + \frac{\vartheta_1}{\lambda^2} + \frac{1-\varphi_1}{\lambda^2(1-\varphi_1^2)} \sum_{t=2}^{T} (\vartheta_t - \varphi_1 \vartheta_{t-1}) \right) \tag{2.11}$$

- update λ^2

$$f(\lambda^2|-) \propto \text{kernel of a invGamma}(a_{\lambda(\text{post})}, b_{\lambda(\text{post})}) \text{ with}$$
$$a_{\lambda(\text{post})} = \frac{T}{2} + a_\lambda$$
$$b_{\lambda(\text{post})} = \frac{(\vartheta_1 - \varphi_0)^2}{2} + \sum_{t=2}^{T} \frac{(\vartheta_t - (1-\varphi_1)\varphi_0 - \varphi_1 \vartheta_{t-1})^2}{2} + b_\lambda \tag{2.12}$$

- update α

if global α: $f(\alpha|-) \propto$ kernel of a Beta$(a_{\alpha(\text{post})}, b_{\alpha(\text{post})})$ with
$$a_{\alpha(\text{post})} = a_\alpha + \sum_{i=1}^{n}\sum_{t=1}^{T} \gamma_{it} \quad b_{\alpha(\text{post})} = b_\alpha + nT - \sum_{i=1}^{n}\sum_{t=1}^{T} \gamma_{it}$$
if time specific α: $f(\alpha_t|-) \propto$ kernel of a Beta$(a_{\alpha(\text{post})}, b_{\alpha(\text{post})})$ with
$$a_{\alpha(\text{post})} = a_\alpha + \sum_{i=1}^{n} \gamma_{it} \quad b_{\alpha(\text{post})} = b_\alpha + n - \sum_{i=1}^{n} \gamma_{it}$$
if unit specific α: $f(\alpha_i|-) \propto$ kernel of a Beta$(a_{\alpha(\text{post})}, b_{\alpha(\text{post})})$ with
$$a_{\alpha(\text{post})} = a_{\alpha i} + \sum_{t=1}^{T} \gamma_{it} \quad b_{\alpha(\text{post})} = b_{\alpha i} + T - \sum_{t=1}^{T} \gamma_{it}$$
if time and unit specific α: $f(\alpha_{it}|-) \propto$ kernel of a Beta$(a_{\alpha(\text{post})}, b_{\alpha(\text{post})})$ with
$$a_{\alpha(\text{post})} = a_{\alpha i} + \gamma_{it} \quad b_{\alpha(\text{post})} = b_{\alpha i} + 1 - \gamma_{it} \tag{2.13}$$

- update a missing observation Y_{it}

for $t = 1$: $f(Y_{it}|-) \propto$ ~~kernel of a~~ $\mathcal{N}(\mu_{Y_{it}(\text{post})}, \sigma^2_{Y_{it}(\text{post})})$ with
$$\sigma^2_{Y_{it}(\text{post})} = \frac{1}{\frac{1}{\sigma^{2\star}_{c_{it}t}} + \frac{\eta_{1i}^2}{2\sigma^{2\star}_{c_{it+1}t+1}(1-\eta_{1i}^2)}}$$
$$\mu_{Y_{it}(\text{post})} = \sigma^2_{Y_{it}(\text{post})} \left(\frac{\mu^\star_{c_{it}t} + \boldsymbol{x}_{it}^T \boldsymbol{\beta}_t}{\sigma^{2\star}_{c_{it}t}} + \frac{\eta_{1i}(Y_{it+1} - \mu^\star_{c_{it+1}t+1} - \boldsymbol{x}_{it+1}^T \boldsymbol{\beta}_{t+1})}{\sigma^{2\star}_{c_{it+1}t+1}(1-\eta_{1i}^2)} \right)$$
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
$$\sigma^2_{Y_{it}(\text{post})} = \frac{1-\eta_{1i}^2}{\frac{1}{\sigma^{2\star}_{c_{it}t}} + \frac{\eta_{1i}^2}{\sigma^{2\star}_{c_{it+1}t+1}}}$$

Esempi simulati mostrano i dati mancanti?

$$\mu_{Y_{it}(\text{post})} = \sigma^2_{Y_{it}(\text{post})} \left(\frac{\mu^\star_{c_{it}t} + \eta_{1i} Y_{it-1} + \boldsymbol{x}_{it}^T \boldsymbol{\beta}_t}{\sigma^{2\star}_{c_{it}t}(1-\eta_{1i}^2)} + \frac{\eta_{1i}(Y_{it+1} - \mu^\star_{c_{it+1}t+1} - \boldsymbol{x}_{it+1}^T \boldsymbol{\beta}_{t+1})}{\sigma^{2\star}_{c_{it+1}t+1}(1-\eta_{1i}^2)} \right)$$
$$\text{for } t = T\text{: } f(Y_{it}|-) \text{ is just the likelihood of } Y_{it} \tag{2.14}$$

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

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.

2.2 Spatial cohesions analysis

The clustering procedure revolves around the product partition model (PPM). The simplest idea is to set $P(\rho_t) \propto \prod_{j=1}^{k_t} C(S_{jt})$, with the function $C(S_{jt})$ that measures how tightly grouped the elements in A are considered to be. Then, to include spatial information, the idea is to extend the PPM from being a function of just $C(S_{jt})$ to the more informed one $C(S_{jt}, \boldsymbol{s}^\star_{jt})$, where S_{jt} is the j-th cluster at time instant t and $\boldsymbol{s}^\star_{jt}$ 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 S_h notation to indicate a general h-th cluster, rather than the more pedantic S_{jt}.

Regarding the computation of spatial cohesion, several choices are available [PQ15]. The main common idea of the following formulas is to favour few spatially connected clusters rather than a lot of singleton ones, to derive more interpretable and meaningful results. For this reason, most of the cohesions employ the $M \cdot \Gamma(|S_h|)$ term, which resembles the DP partitioning method that helps in reaching that goal.

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

All the tests of Figures 2.3 and 2.4 refer to the partition of Figure 2.2, taken as test case here from a general fit on the same spatio-temporal dataset of Chapter 4. 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

$$\mu_{Y_{it}(\text{post})} = \sigma^2_{Y_{it}(\text{post})} \left(\frac{\mu^\star_{c_{it}t} + \eta_{1i} Y_{it-1} + \boldsymbol{x}_{it}^T \boldsymbol{\beta}_t}{\sigma^{2\star}_{c_{it}t}(1-\eta^2_{1i})} + \frac{\eta_{1i}(Y_{it+1} - \mu^\star_{c_{it+1}t+1} - \boldsymbol{x}_{it+1}^T \boldsymbol{\beta}_{t+1})}{\sigma^{2\star}_{c_{it+1}t+1}(1-\eta^2_{1i})} \right)$$
$$\text{for } t = T\text{: } f(Y_{it}|-) \text{ is just the likelihood of } Y_{it} \tag{2.14}$$

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

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.

2.2 Spatial cohesions analysis

The clustering procedure revolves around the product partition model (PPM). The simplest idea is to set $P(\rho_t) \propto \prod_{j=1}^{k_t} C(S_{jt})$, with the function $C(S_{jt})$ that measures how tightly grouped the elements in A are considered to be. Then, to include spatial information, the idea is to extend the PPM from being a function of just $C(S_{jt})$ to the more informed one $C(S_{jt}, \boldsymbol{s}^\star_{jt})$, where S_{jt} is the j-th cluster at time instant t and $\boldsymbol{s}^\star_{jt}$ 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 S_h notation to indicate a general h-th cluster, rather than the more pedantic S_{jt}.

Regarding the computation of spatial cohesion, several choices are available [PQ15]. The main common idea of the following formulas is to favour few spatially connected clusters rather than a lot of singleton ones, to derive more interpretable and meaningful results. For this reason, most of the cohesions employ the $M \cdot \Gamma(|S_h|)$ term, which resembles the DP partitioning method that helps in reaching that goal.

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

All the tests of Figures 2.3 and 2.4 refer to the partition of Figure 2.2, taken as test case here from a general fit on the same spatio-temporal dataset of Chapter 4. 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.

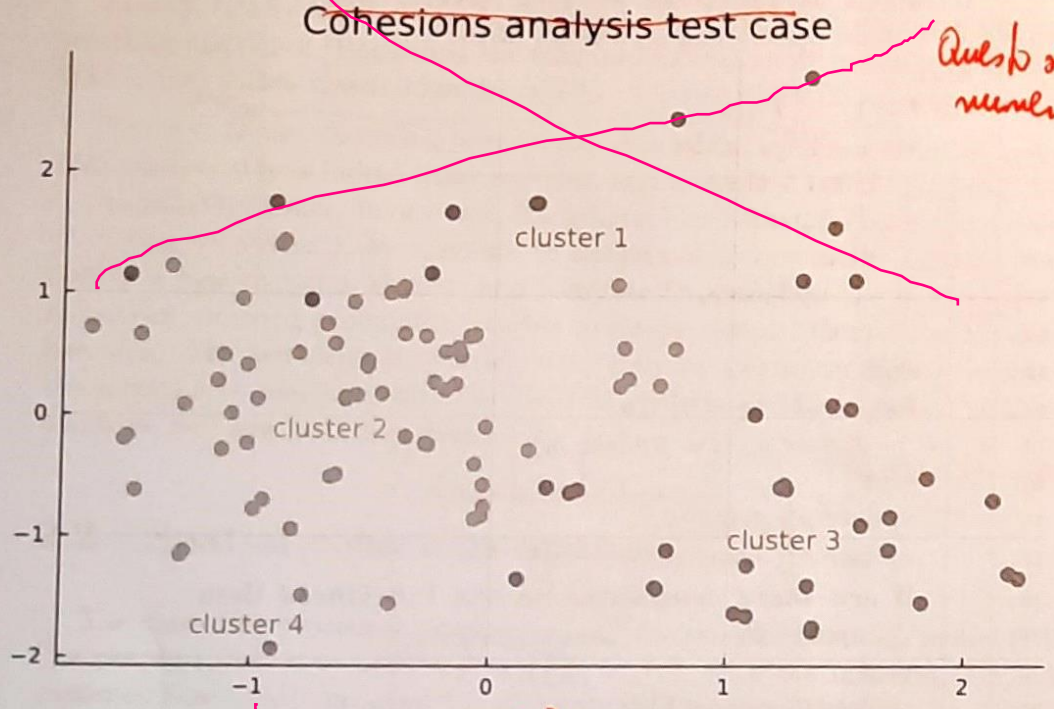

Figure 2.2: Partition considered to analyse the spatial cohesions.

The first cohesion uses a tessellation idea from [DH01] that considers $\mathcal{D}_h = \sum_{i \in S_h} \|\boldsymbol{s}_i - \bar{\boldsymbol{s}}_h\|$ 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.

$$C_1(S_h, \boldsymbol{s}_h^*) = \begin{cases} \dfrac{M \cdot \Gamma(|S_h|)}{\Gamma(\alpha \mathcal{D}_h)\mathbb{1}_{[\mathcal{D}_h \geq 1]} + \mathcal{D}_h \mathbb{1}_{[\mathcal{D}_h < 1]}} & \text{if } |S_h| > 1 \\ M & \text{if } |S_h| = 1 \end{cases} \tag{2.15}$$

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

$$C_2(S_h, \boldsymbol{s}_h^*) = M \cdot \Gamma(|S_h|) \cdot \prod_{i,j \in S_h} \mathbb{1}_{[\|\boldsymbol{s}_i - \boldsymbol{s}_j\| \leq a]} \tag{2.16}$$

According to this function, from Figure 2.3 we can see how the purple cluster is considered the one with the highest cohesion, being a singleton. The runner-up is the green cluster, because it's the first among all the non-singletons which activates cohesion 2 when we increase the parameter a. The orange and blue clusters, instead, 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 +15], is instead often desirable, for theoretical or computational purposes, and the following two cohesions are able to provide it [MQR11].

Cohesion 3, called auxiliary similarity, treats the spatial coordinates $\boldsymbol{s}^\star$ 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.

$$C_3(S_h, \boldsymbol{s}_h^\star) = M \cdot \Gamma(|S_h|) \cdot \int \prod_{i \in S_h} q(\boldsymbol{s}_i|\boldsymbol{\xi}_h) q(\boldsymbol{\xi}_h)\, d\boldsymbol{\xi}_h \tag{2.17}$$

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

$$C_4(S_h, \boldsymbol{s}_h^\star) = M \cdot \Gamma(|S_h|) \cdot \int \prod_{i \in S_h} q(\boldsymbol{s}_i|\boldsymbol{\xi}_h) q(\boldsymbol{\xi}_h|\boldsymbol{s}_h^\star)\, d\boldsymbol{\xi}_h \tag{2.18}$$

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 distance of the units from the cluster centroid, and then adjust the penalization trough the parameter φ, to control how much penalize dissimilar values.

$$C_5(S_h, \boldsymbol{s}_h^\star) = \exp\left\{-\varphi \sum_{i \in S_h} \|\boldsymbol{s}_i - \bar{\boldsymbol{s}}_h\|\right\} \tag{2.19}$$

$$C_6(S_h, \boldsymbol{s}_h^\star) = \exp\left\{-\varphi \log\left(\sum_{i \in S_h} \|\boldsymbol{s}_i - \bar{\boldsymbol{s}}_h\|\right)\right\} \tag{2.20}$$

2.3 Covariates similarities ~~analysis~~

A wide spectrum of choice is also available for covariates similarities [PQ18]. To account for them, the idea consists in extending the PPM to make it function of S_{jt},

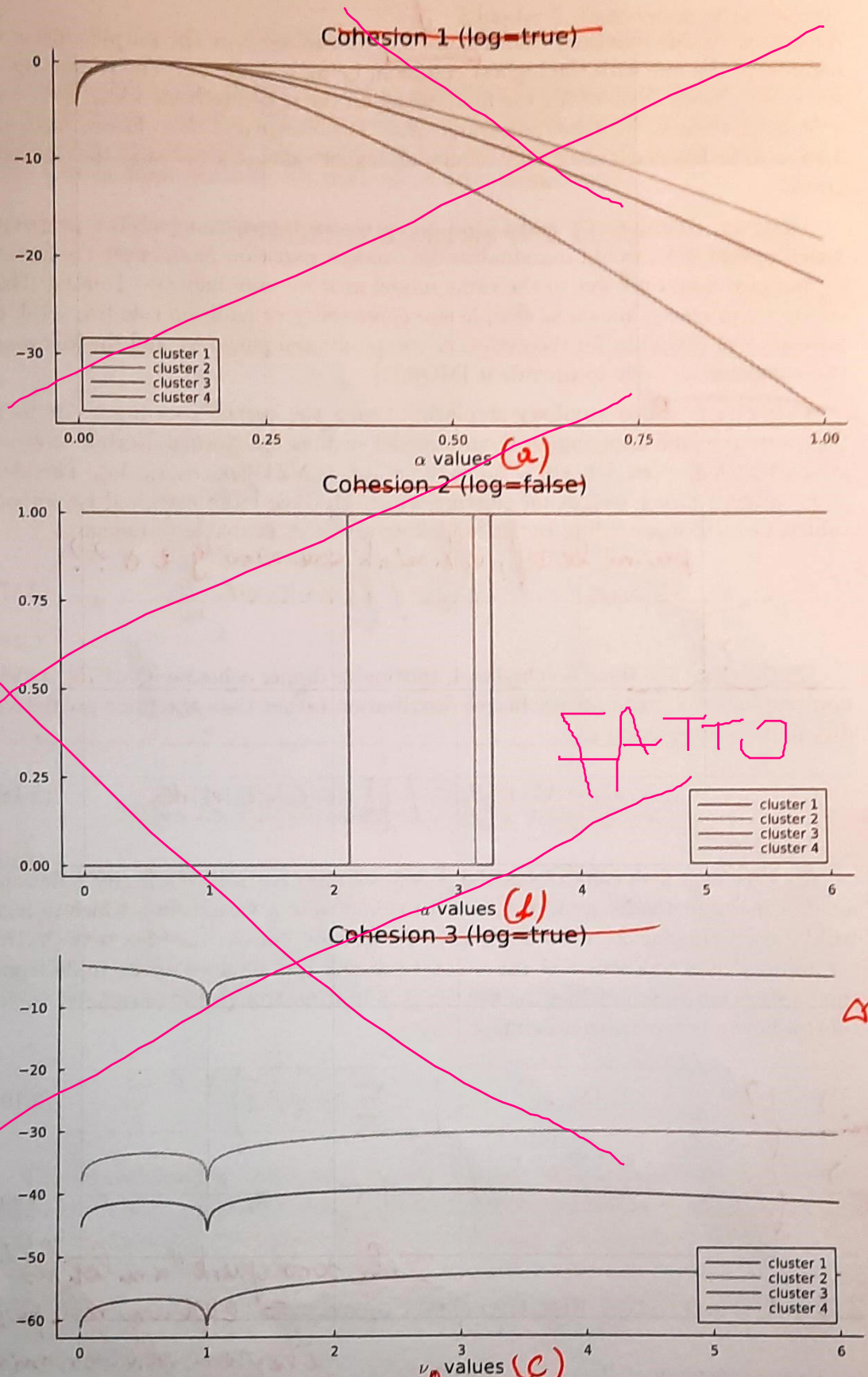

Figure 2.3: Cohesions 1, 2, and 3 computed on the test case partition, with respect to different values of their tuning parameter. Cohesion 2 is without the logarithm applied just for plotting purposes, since otherwise the values would have been $-\infty$ and 0.

$\boldsymbol{s}^\star_{jt}$, and now also of $X^\star_{jt}$, being this the $p \times |S_{jt}|$ matrix storing the covariates of the units belonging to cluster j at time t, i.e. $X^\star_{jt} = \{\boldsymbol{x}^\star_{it} = (x_{it1}, \dots, x_{itp})^T : i \in S_{jt}\}$.

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

$$P(\rho) \propto \prod_{h=1}^{k_t} C(S_{jt}, \boldsymbol{s}^\star_{jt}) \left(\prod_{r=1}^{p} g(S_{jt}, \boldsymbol{x}^\star_{jtr}) \right) \tag{2.21}$$

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

$$P(\rho) \propto \prod_{h=1}^{k_t} C(S_{jt}, \boldsymbol{s}^\star_{jt}) g(S_{jt}, X^\star_{jt}) \tag{2.22}$$

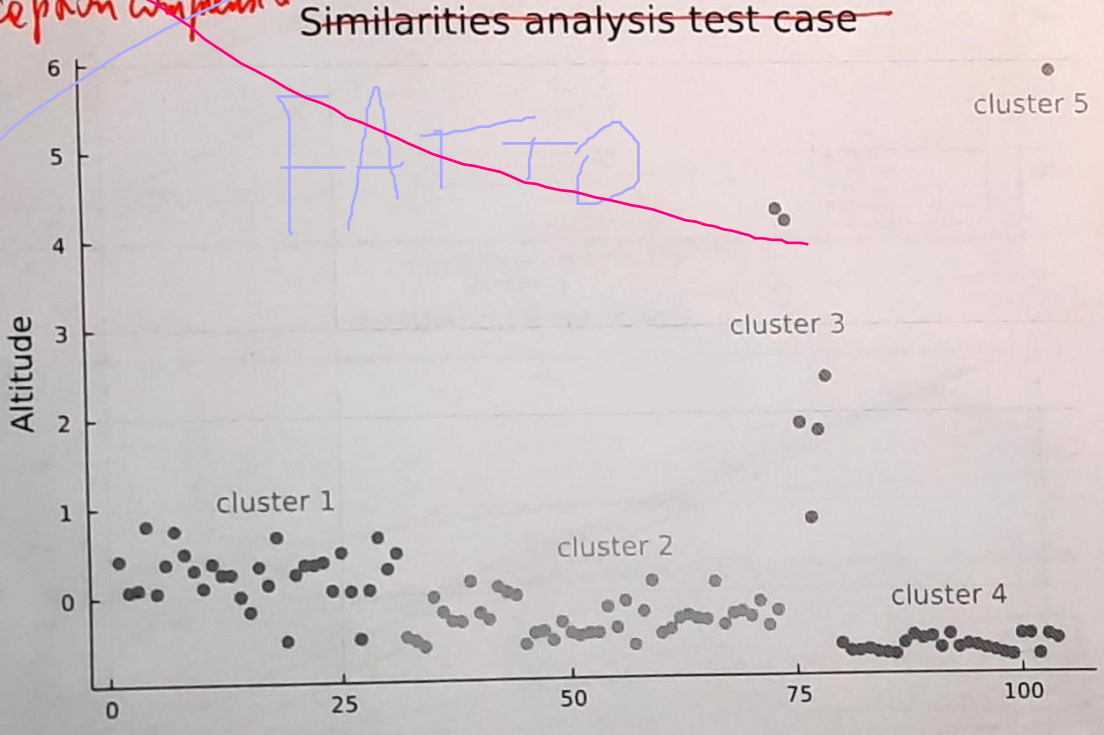

Figure 2.5: Partition considered to analyse the covariate similarities. The order of the units has been modified to plot together all units belonging to the same cluster.

As in the previous section, we will now discuss all the similarities implemented in the JDRPM model and perform tests on each of them. All the tests will be

referred to the test case partition displayed in Figure 2.5. Regarding the notation, we will again employ the simpler and general one dropping the spatio-temporal context.

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

$$g_1(S_h, \boldsymbol{x}_h^\star) = \exp\{-\varphi H(S_h, \boldsymbol{x}_h^\star)\} \tag{2.23}$$

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

$$g_2(S_h, \boldsymbol{x}_h^\star) = \exp\left\{-\alpha \sum_{i,j \in S_h, i \neq j} d(x_i, x_j)\right\} \tag{2.24}$$

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

$$g_3(S_h, \boldsymbol{x}_h^\star) = \exp\left\{-\frac{2\alpha}{|S_h|(|S_h|-1)} \sum_{i,j \in S_h, i \neq j} d(x_i, x_j)\right\} \tag{2.25}$$

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 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 trough the same modelling style of the spatial coordinates case.

$$g_4(S_h, \boldsymbol{x}_h^\star) = \int \prod_{i \in S_h} q(x_i|\boldsymbol{\xi}_h) q(\boldsymbol{\xi}_h)\, d\boldsymbol{\xi}_h \tag{2.26}$$

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 2.6 and 2.7. Intuitively, Figure 2.5 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). However, it tends to penalize a lot sparse clusters, as we saw in the previous test case. This could suggest to maybe implement other distance metrics rather than the L^2 norm in the computation of $H(S_h, \boldsymbol{x}_h^\star)$. On the other hand, similarity 4 appeared to be quite sensible to the parameters regulating the invGamma distribution for σ^2, suggesting that each covariate that we want to include in the clustering process should have her properly-tuned pair of a_0 and b_0 parameters. The JDRPM implementation will provide this flexibility in assigning separately those parameters for each clustering covariate.

Prima di parlare delle ottimizzazioni del codice, deve brevemente spiegare che il calcolo delle posterior per i modelli bayesiani si fa con metodi di simulazione, gli MCMC. Spiega in 3 frasi cosa sono, e descrive brevemente quello di Pege et al.
↳ queste parte la potrebbe mettere nel cap precedente, prima di descrivere gli algoritmi. In tal caso, qui fa riferimento a quella parte

Chapter 3

Implementation and optimizations

↳ Spiega l'algoritmo MCMC per calcolare la posterior del nostro modello è stato codificato in Julia....

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

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

As we will see in Section 4, even despite the higher complexity of the model we managed to get better performance in Julia compared to the original C implementation, at the reasonable cost of a smaller increase in the memory requirements, which nowadays should not be a big deal. Anyway, this improvement was possible trough several refinements and optimizations, which we will now discuss.

3.1 Optimizations

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

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

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

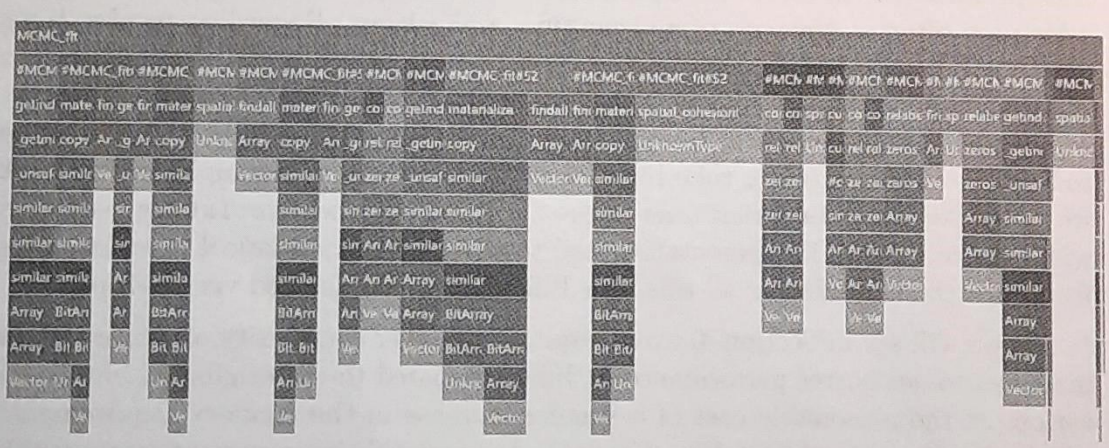

Figure 3.1: Flame graph derived from a simple test fit with $n = 20$, $T = 50$, ran for 10000 iterates. Each section, identified with a colored box, represents a function call with its stack trace below, i.e. all the other subsequent function calls generated from the initial top one. The widths are proportional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

```
aux1 = k0 * sdim; aux2 = k0 + sdim
Psi_n = Psi .+ S .+ aux1 / (aux2) .* Mtmp
```

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

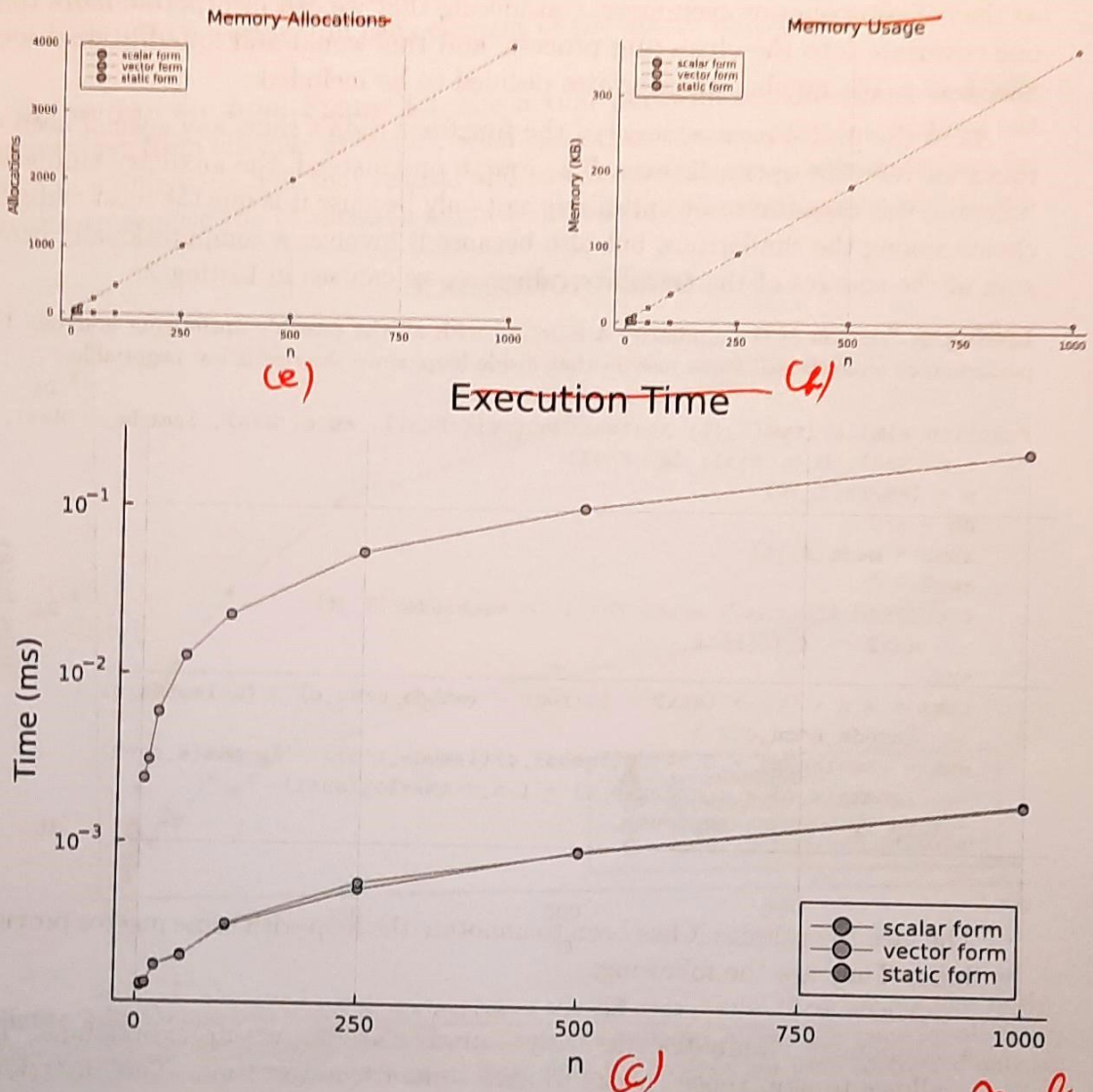

Figure 3.2: Performance comparison between the three versions of the cohesion 4 function. Tests ran through the `BenchmarkTools` package of Julia by randomly generating the spatial coordinates of the various test sizes n, with similar results standing for cohesion 3. The memory allocation and usage plots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

Figure 3.2 shows the comparison of their performances, where we can see how

the scalar and static versions indeed perform very similarly to each other, and more quickly with respect to the first version.

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

3.1.2 Optimizing covariates similarities

Another problem has been understanding how to speed up the computation of the similarity functions, since those would also be called possibly millions of times as the cohesion ones or even more, considering that we can incorporate more than one covariate into the clustering process, and this would add an additional loop based on p, the number of covariates decided to be included.

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

Listing 2: Version of the similarity 4 function with all the possible optimizing macros. The performance analysis will focus just on that inside loop, since the rest is not negotiable.

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

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

- `@inbounds` eliminates the array bounds checking within expressions. This allows to skip those checks to save some execution time. This insertion is harmless as long as we are sure that in our code design no out-of-bounds or wrong accesses will occur. Otherwise some undefined behaviour will take place. The above loop is indeed very simple and safe, so this assumption is clearly satisfied.

- `@fastmath` executes a transformed version of the expression which calls functions that may violate strict IEEE semantics[1]. For example, its use could make $(a+b)+c \neq a+(b+c)$, but just in very pathological cases. Again, this is not a problem in our function, where we are just computing $\sum X_i^2$, since it does not have an intrinsically "right order" in which it has to be done.

- `@simd` (single instruction multiple data) annotate a for loop to allow the compiler to take extra liberties to allow loop re-ordering. This is a sort of parallelism technique, but rather than distributing the computational load on more processors we just *vectorize* the loop, i.e. we enable the CPU to perform that single instruction (summing the square of the i-th component to a reduction variable) on multiple data chunks at once, using vector registers, rather than working on each element of the vector individually.

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

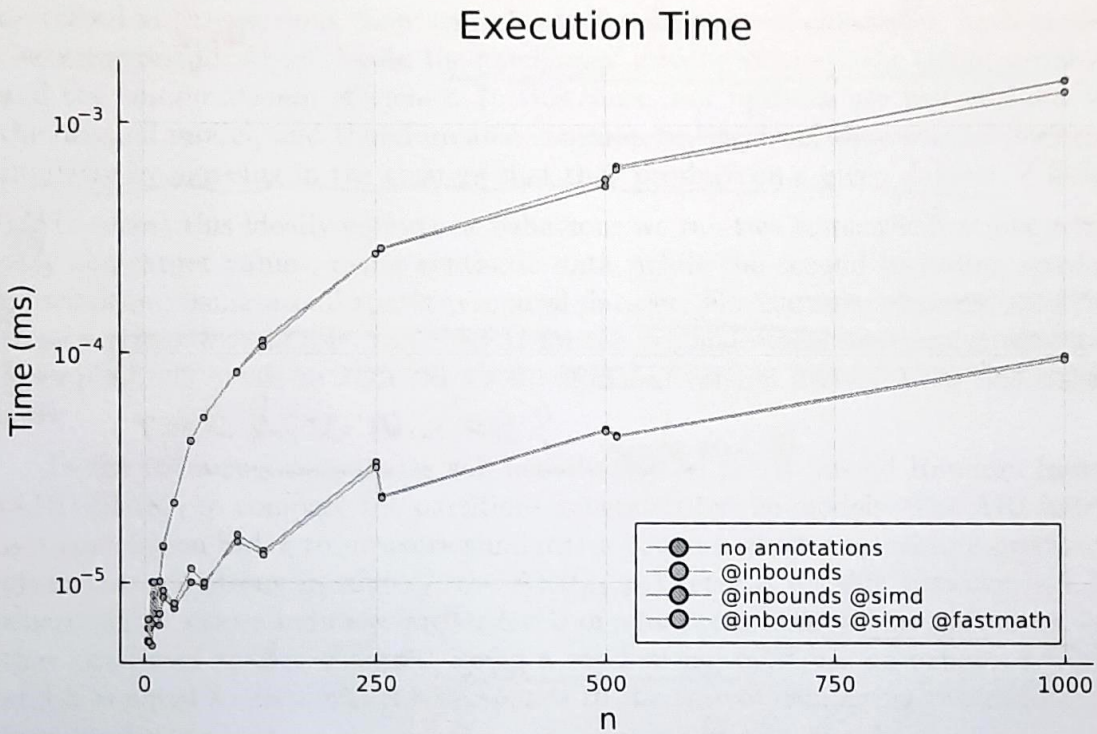

Figure 3.3: Comparison of the performances of the different possible loop annotations in the similarity 4 implementation. Their numerical output results are indeed the same for all cases. There are no memory allocation and usage plots since the analysis has been conducted only to evaluate the performances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

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

Chapter 4

Testing

4.1 Assessing the equivalence of the models

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

To assess this ideally equivalent behaviour we ran two tests: the first one with only the target values, using synthetic data, while the second including spatial information, using a real spatio-temporal dataset. For the sake of clarity, also in these sections we will refer to CDRPM for the original model and implementation from [PGD22], while to JDRPM for the updated version derived from this thesis work.

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

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

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

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

4.1.1 Target variable only

For the first test we generated a dataset of $n = 10$ units and $T = 12$ time instants, and fitted both models collecting 1000 iterates derived from 10000 total iterations, discarding the first 5000 as burnin and thinning by 5. Those parameters, as well as the generating function, were the same of [PQD22]. The generating function allowed to create data points with temporal dependence, which could be tuned trough some dedicated parameters. Both models were fitted using the full formulation, i.e. including and updating also the optional autoregression parameters η_{1i}, and φ_1, and using a time specific α.

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

	MSE mean	MSE median	LPML	WAIC	exec. time
CDRPM	1.6731	1.5861	-249.61	469.69	4.8s
JDRPM	**1.2628**	**1.2181**	**-227.83**	**415.03**	**2.5s**

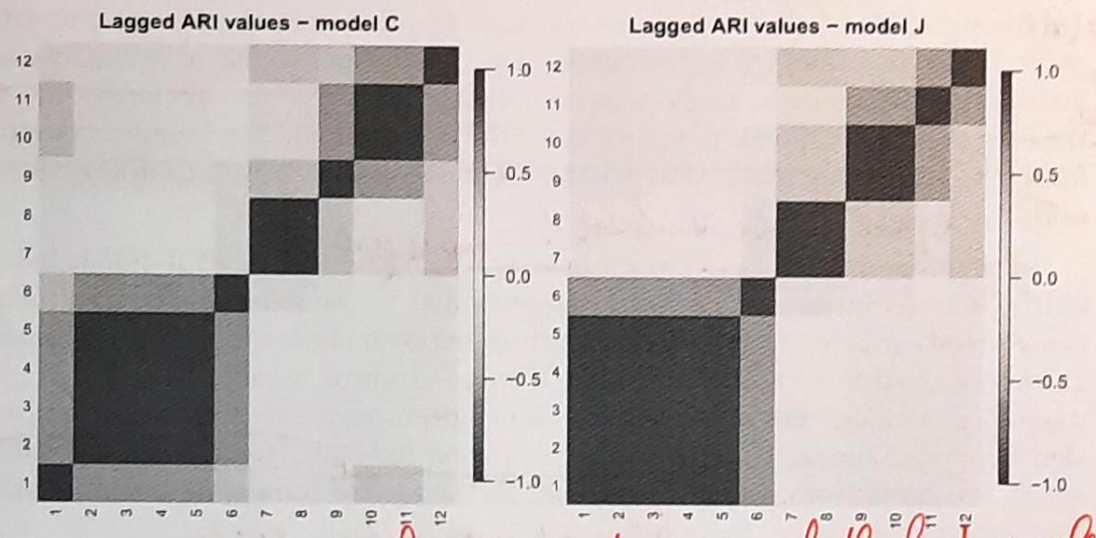

Figure 4.1: Lagged ARI values for the two models, on the fit with target only data. The partitions were estimated using the `salso` function from the corresponding R library.

At this testing stage there were just the target values from Y_{it} to dictate the clusters definition, and both models managed to provide good fits, as we can read from Table 4.1. The JDRPM model had better fit metrics (lower WAIC and higher LPML) and also faster execution time, which is however not really relevant in this small-sized fit. Regarding the fitted values, displayed in Figure 4.4 together with

4.2 Performance with missing values

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

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

4.2.1 Target variable only (NA case)

The JDRPM seems to exhibit good performance also with missing values. From Table 4.3 we can see how the performances, naturally worse than the fit in the full case, are still good. The MSE is inevitably higher since the model did not have all the data points, and so we get less precise fitted values for the corresponding missing spots in the dataset, but the fitted values of Figure 4.9 are still close the one obtained with the fit on full dataset.

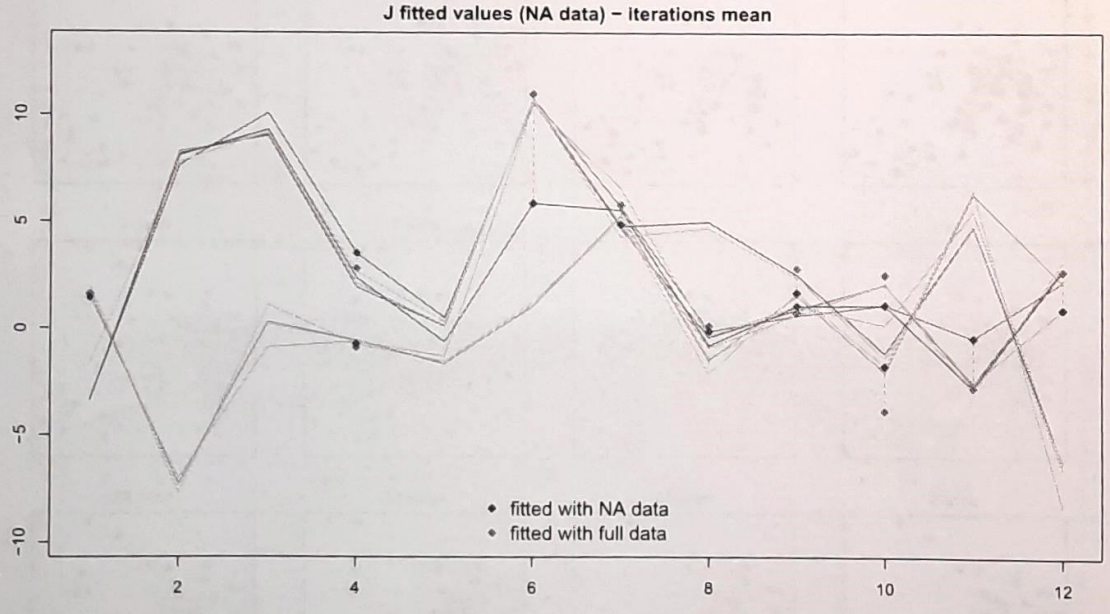

Figure 4.9: Fitted values estimate trough the mean of the 1000 iterates generated by model JDRPM. The generated target values were the same of Figure 4.4. Special point markers are used for the data points which were missing, to highlight the gaps between the fitted values on the full dataset (green squares) and the fitted ones on the NA dataset (red squares).

From Figures 4.11 and 4.10 we can see how we get almost the same temporal

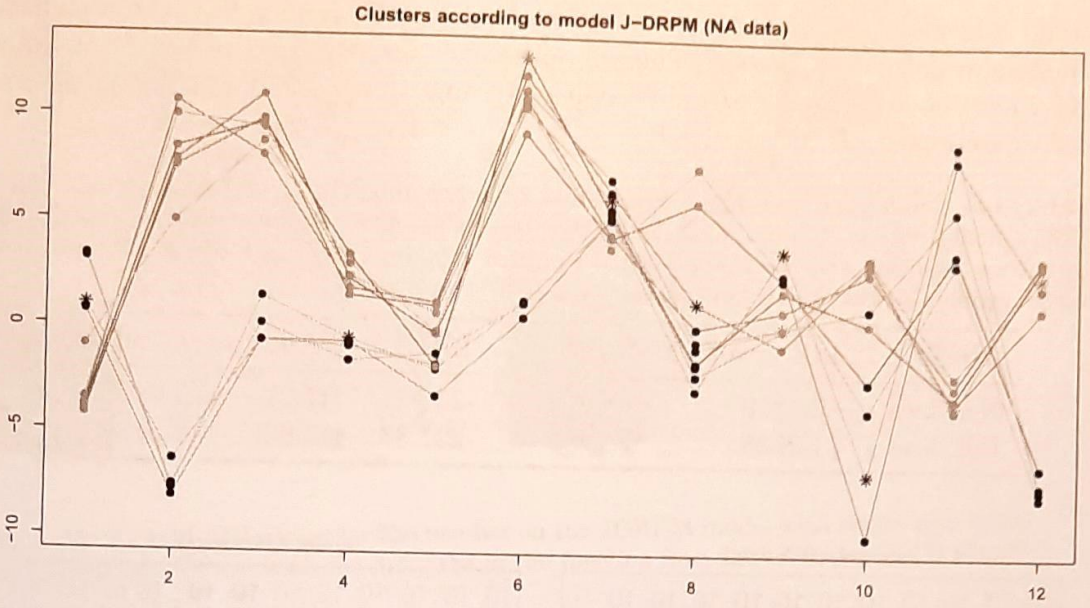

Figure 4.12: Visual representation of the clusters produced by the model, with units' labels represented as colored dots overlaid to the trend of the original generated target variables. Special point markers are used for the data points corresponding to missing values.

4.2.2 Target variable plus space (NA case)

Table 4.4: Summary of the comparison between the two Julia fits on target plus space values. The MSE columns refer to the accuracy between the fitted values generated by the models (estimated by taking the mean and the median of the returned 4000 iterates) and the true values of the target variable. Higher LPML and lower WAIC indicate a better fit.

JDRPM	MSE mean	MSE median	LPML	WAIC	exec. time
NA data	0.0160	0.0170	502.86	-1793.64	43m
full data	0.0131	0.0138	624.91	-1898.05	56m

The JDRPM model seems able to perform well also in the case of fits including spatial information. Table 4.4 shows how the accuracy reduces, which again is inevitable since we are fitting with missing data, but not drastically. In fact, the drops in LPML and WAIC metrics are just of 3.16% and 0.95% respectively. Fitted values are reported in Figure 4.14. Regarding the reported execution times, they are not completely truthful, as already said before, due to the not-full commitment of my laptop resources to the fitting task. In fact, the fit with NA data appeared to be faster than the one with full data, which is somewhat implausible since in the presence of NA data there is the additional load of updating the missing Y_{it} values. Again, more accurate results will be provided in Section 4.4.

The temporal trend managed as well to remain similar to the trend we saw in Section 4.1.2, as proved by Figure 4.13. Regarding the similarity of the produced clusters, we computed the values $\text{ARI}(\rho_{\text{JDRPM\_NA}}(t), \rho_{\text{JDRPM\_full}}(t))$ for all time instants $t = 1, \ldots, 12$, and we obtained a mean of 0.82 and a median of 0.86, denoting still a relatively high level of agreement in the partitions despite the loss

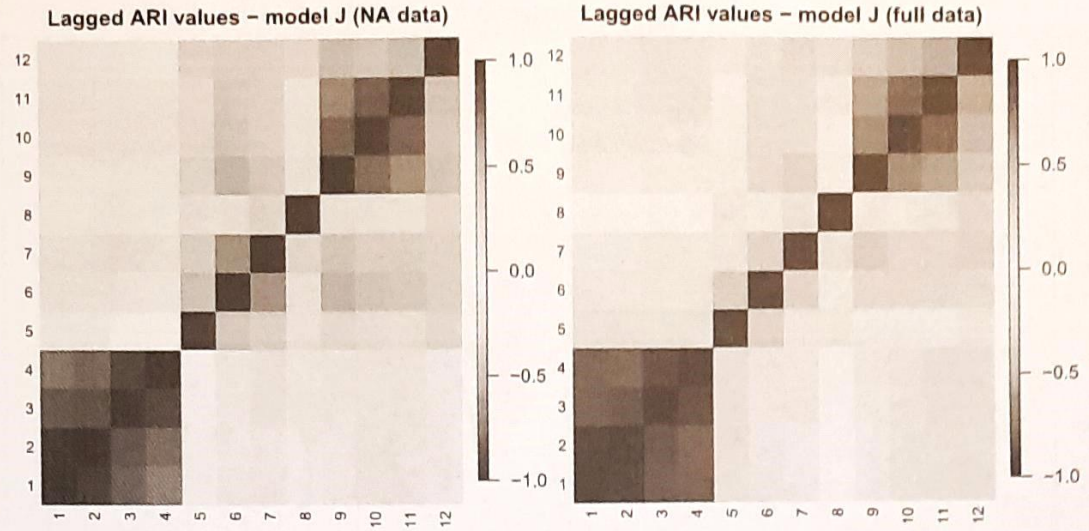

Figure 4.13: Lagged ARI values for the two fits on the JDRPM model with target plus space values. The partitions were estimated using the `salso` function from the corresponding R library.

of quite some data (121 points out of the 1260 of the full dataset).

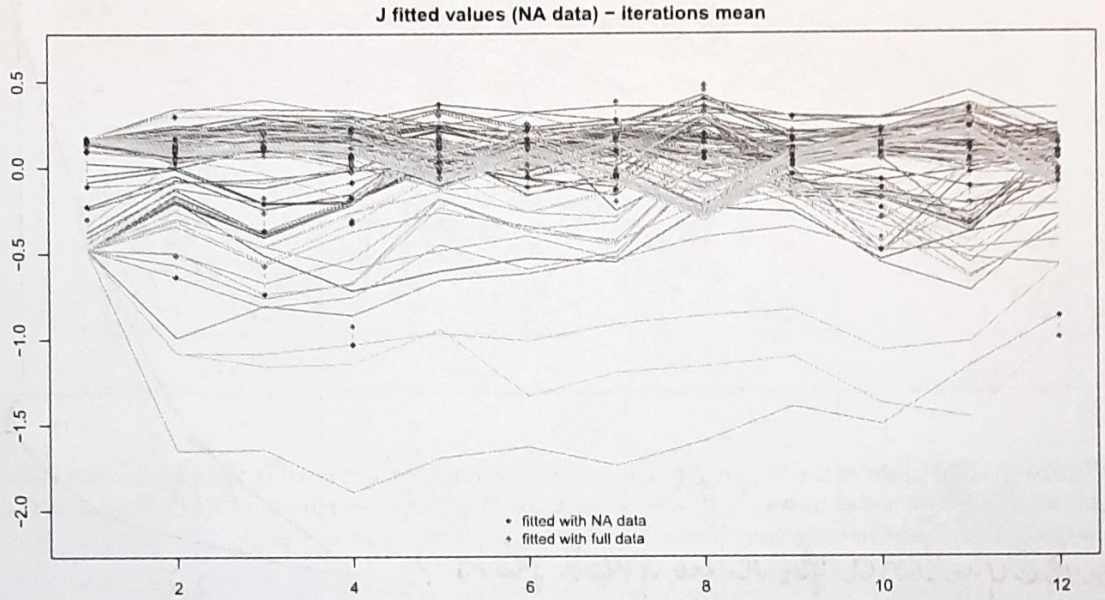

Figure 4.14: Fitted values estimate trough the mean of the 4000 iterates generated by model JDRPM. The generated target values were the same of Figure 4.7. Special point markers are used for the data points which were missing, to highlight the gaps between the fitted values on the full dataset (green squares) and the fitted ones on the NA dataset (red squares).

4.3 Effects of the covariates

To perform the fits that will now follow, all the included covariates were treated in the same way as the target value, with the time-wise centering procedure and

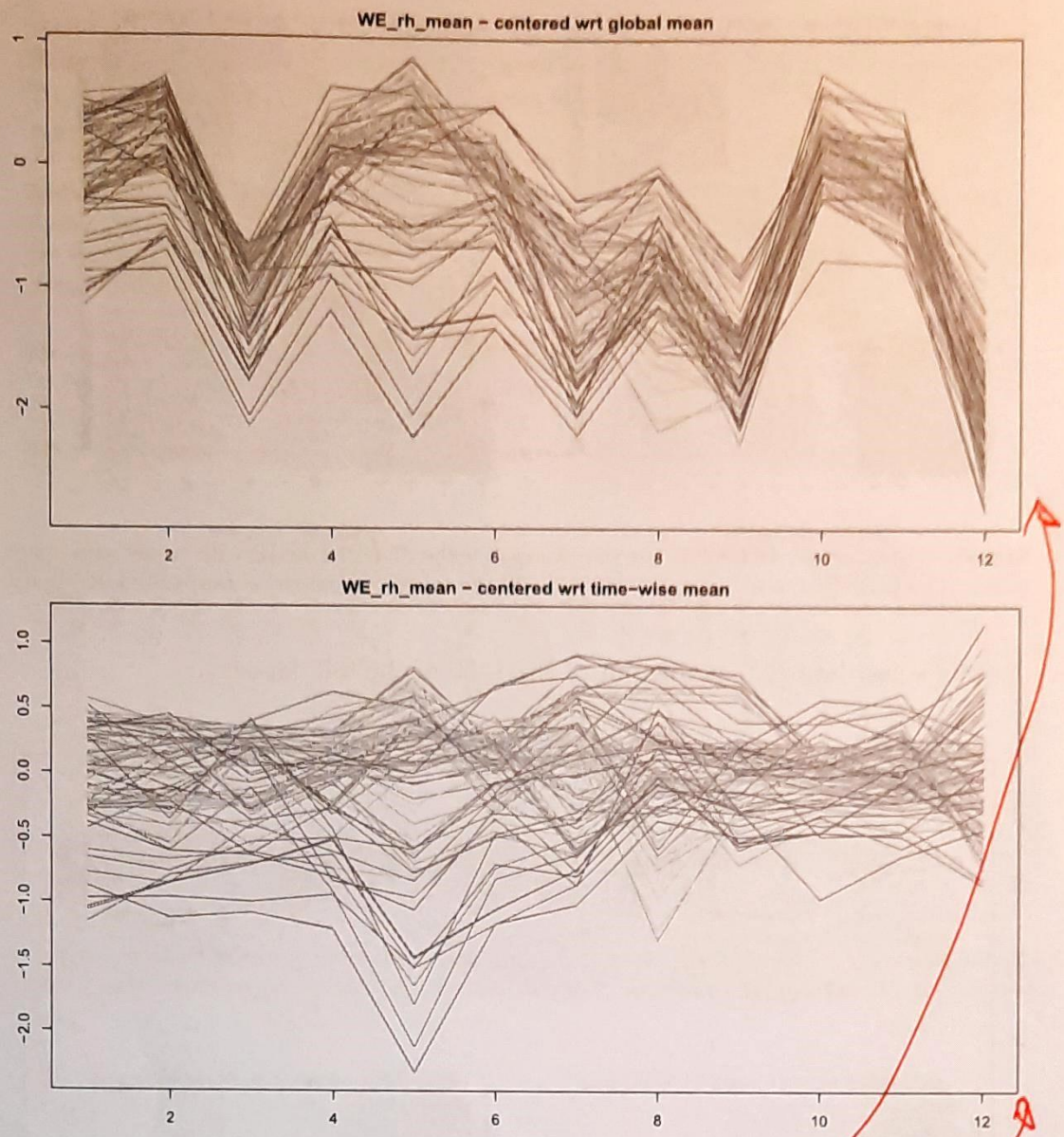

Figure 4.15: Values of the `WE_rh_meancentered` covariate corrected using the global mean (top) and using the time-wise mean (bottom). Coloring is based on the ranking of PM_{10} values of the units according to their median, from highest (red) to lowest (blue).

reasoning described in Section 4.1.2. Figure 4.15 provides an exemplification of the effect of this transformation on one of the covariates. Again, this was performed in order to remove any trend or bias on the covariates and to equalize their contribution at each time instant.

We will now see some testing fits which employ the core advancement of the JDRPM update, i.e. the inclusion of covariates. Being with radically different purposes, we will distinguish the cases of covariates in the likelihood and covariates in the clustering.

these fitted values resemble more the ones of the original target variable, especially if compared to the case of Figure 4.7, in which both fits were without any "external suggestion" of covariates in the likelihood. A further proof of this insertion is given by Figure 4.21, representing the trace plot of the fitted values for a problematic unit and time instant, with problematic in the sense that both the standard JDRPM and CDRPM fits didn't manage to provide precise estimates, which now became more accurate. In the end, this analysis denotes how the addition of this regression parameter β_t could be beneficial to recover more accuracy in the results, and also help the clustering process since the regressor contribution manifests when using variables computed from the target values inside the update steps of the parameters.

4.3.2 Covariates in the clustering

Now we consider the more theoretically effective and impactful inclusion of covariates in the clustering process. For the choice of which covariates to include, we reasoned about which where the aspects that could affect more the PM_{10} concentrations, and we ended up selecting the following three covariates:

- `WE_wind_speed_10m_max`. Wind speed plays a significant role in the concentration levels of air pollutants. Its effect is of dual: wind can disperse the pollutants away from their source, spreading them to different areas and thus locally reducing the levels, but it can also have an opposite effect, increasing the amount of contaminants in the air by resuspending the particles which were settled down on the earth surface, such as roads, soils, or buildings. This latter effect is particularly visible in dry and windy rural areas, of which Lombardy is a suitable example. The dataset offered also the wind speed at 100m, but this would be relevant for a wider analysis, where e.g. the trend of PM_{10} concentrations are followed over multiple regions or countries. Our setup, as with the time-wise mean adjustment, was instead more focused on local and particular behaviours, on the more the ground-level perspective about Lombardy cities.

- `WE_tot_precipitation`. It is well known how rainwater can improve the air quality thanks to water droplets that attract aerosol particles, taking them away from the air, and bringing them to the ground. This process, known as wet deposition, precipitation, scavenging, or washout is indeed very effective in reducing the concentrations of air pollutants.

- `WE_blh_layer_max`. The boundary layer height (BLH) is another relevant variable in terms of the dispersion of air contaminants. This layer defines the height at which air mixing occurs. Usually air gets colder when rising in the atmosphere; however, there could be some warm air regions on top of colder ones. At this boundary, air is not more able to mix, with the warm layer causing a sort of lid, a covering, which traps the lower cold air, with all the contaminants, below it. This reduces the quality of the air since the pollution builds up and accumulates since it has no way to disperse. This covariate, therefore, records the maximum height at which this problematic

Chapter 5

Conclusion

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

— Fëdor Dostoevskij, *Brothers Karamazov*

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

From a theoretical point of view, it offers the same base structure, just with the addition of a possible regression term in the likelihood, while also enhancing the quality of the sampled values of the parameters by having changed the laws of the variances from $\mathcal{U}$ to invGamma. In fact, this choice recovers conjugacy in the model and therefore improves the mixing of the chain during the fitting of the model. And, of course, the insertion of covariates in the clustering should provide even more accurate results in the generation of the partitions with respect to only having the spatio-temporal information layers.

A possible drawback, inevitable with the increased model complexity, could be the tuning of the parameters, for example in the cohesions and similarities function to find the right balance between the different contributes of the information levels. However, in this regards, an hopefully helpful analysis was conducted in Chapter 2, where the effects of the tuning parameters for all cohesions and similarities have been explored. Moreover, the Julia `MCMC_fit` function provides an optional argument `cv_weight`, defaulted to 1, which can be used to scale up (or down) the contribute of the covariates similarities, e.g. to make them closer to the interval of values to which the spatial cohesions values belong.

This higher complexity appeared also in the tuning of the invGamma(a, b) parameters, which are indeed more delicate than a simple $\mathcal{U}(l, u)$. To this end, a possible solution could be to run an initial fit with the original CDRPM model, to see where the values of the variances tend to gather, and according to that fixing the invGamma parameters. For example, in the spatio-temporal tests of Section 4.1.2 we saw, from the CDRPM fits, very low values for the variances parameters, and as such we assigned, for λ^2 and τ_t^2 parameters in the JDRPM model, an invGamma$(a =$

$1.9, b = 0.4$), which has 90% of his density in the interval $[0.109151, 1.58222]$. The more relevant $\sigma_{jl}^{2\star}$ parameter had also pretty low values, however on that we attached a more uninformative prior of invGamma($a = 0.01, b = 0.01$), which despite being not always suggested [Gel04] turned out to be fine and very precise, relatively to the sampled "reference" values of CDRPM. An alternative solution, probably less theoretically appealing but possibly effective, could be to truncate the invGamma laws to a certain region in order to avoid sampling extremely high values, which can happen when uninformative priors get not properly adjusted by the data, and to resemble more the density of a $\mathcal{U}$ random variable. This choice can be seamlessly implemented in the Julia code by e.g. swapping `rand(InverseGamma(a,b))` to `rand(truncated(InverseGamma(a,b),l,u))`, with l and u being the lower and upper bounds of the designed truncation interval.

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

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

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

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