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`.

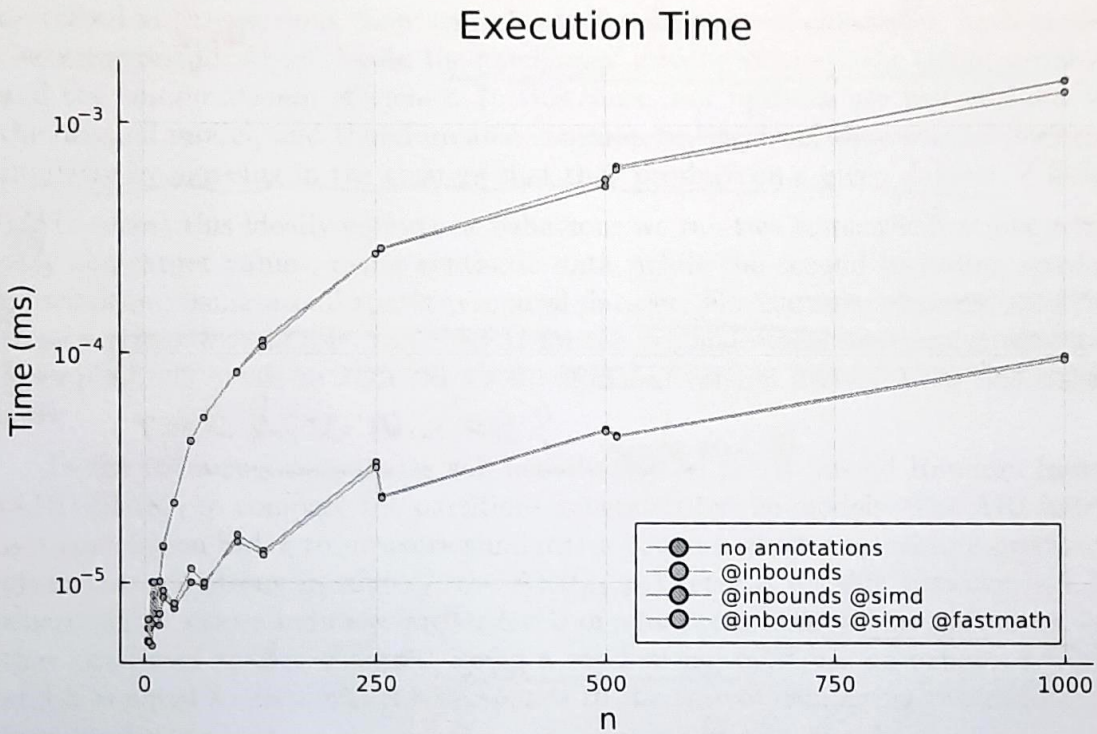

Figure 3.3: Comparison of the performances of the different possible loop annotations in the similarity 4 implementation. Their numerical output results are indeed the same for all cases. There are no memory allocation and usage plots since the analysis has been conducted only to evaluate the performances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

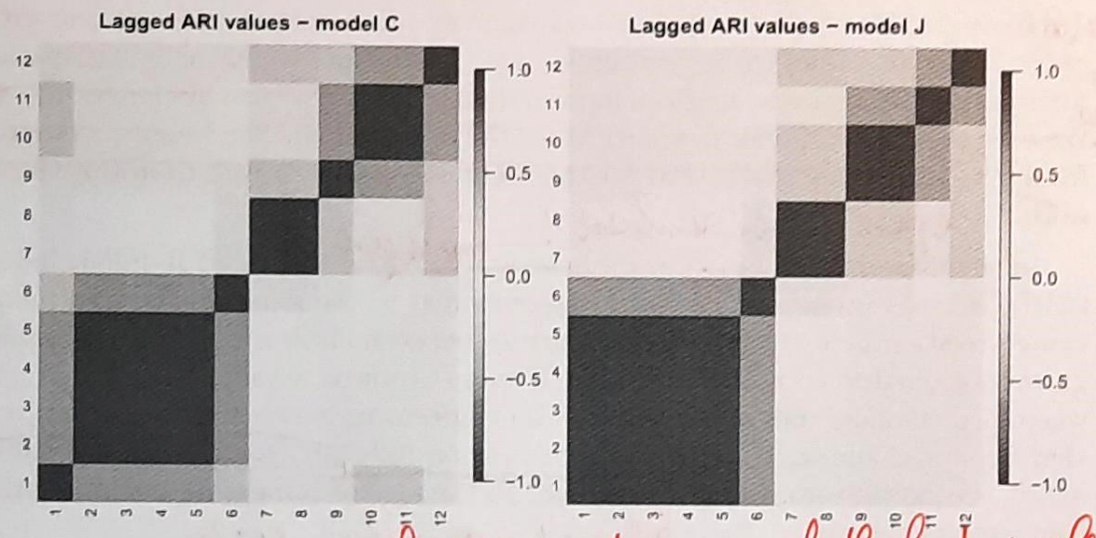

Figure 4.1: Lagged ARI values for the two models, on the fit with target only data. The partitions were estimated using the `salso` function from the corresponding R library.

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.

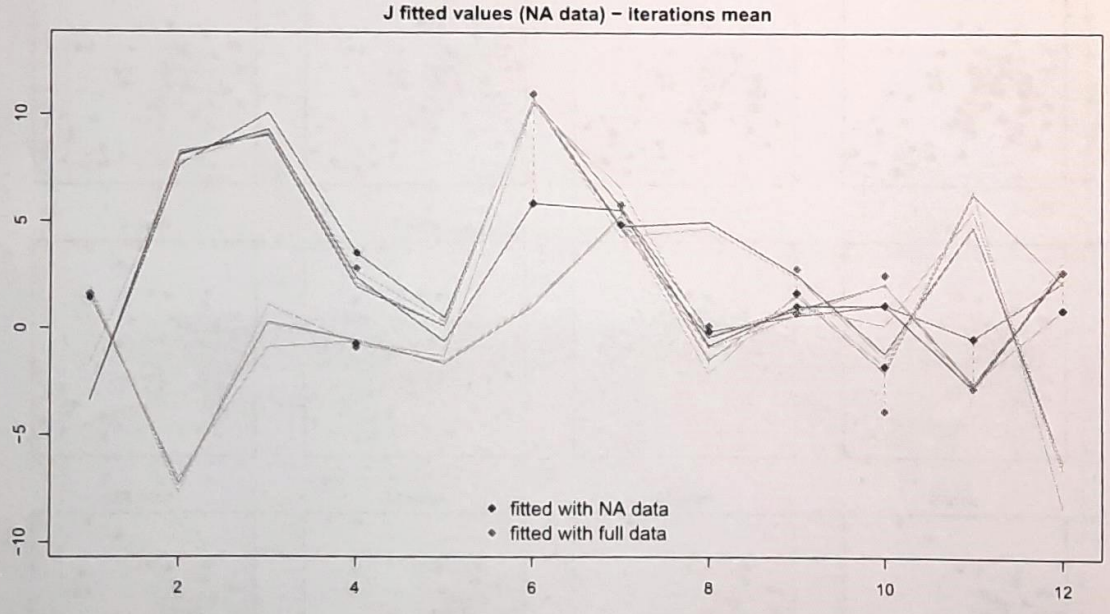

Figure 4.9: Fitted values estimate trough the mean of the 1000 iterates generated by model JDRPM. The generated target values were the same of Figure 4.4. Special point markers are used for the data points which were missing, to highlight the gaps between the fitted values on the full dataset (green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

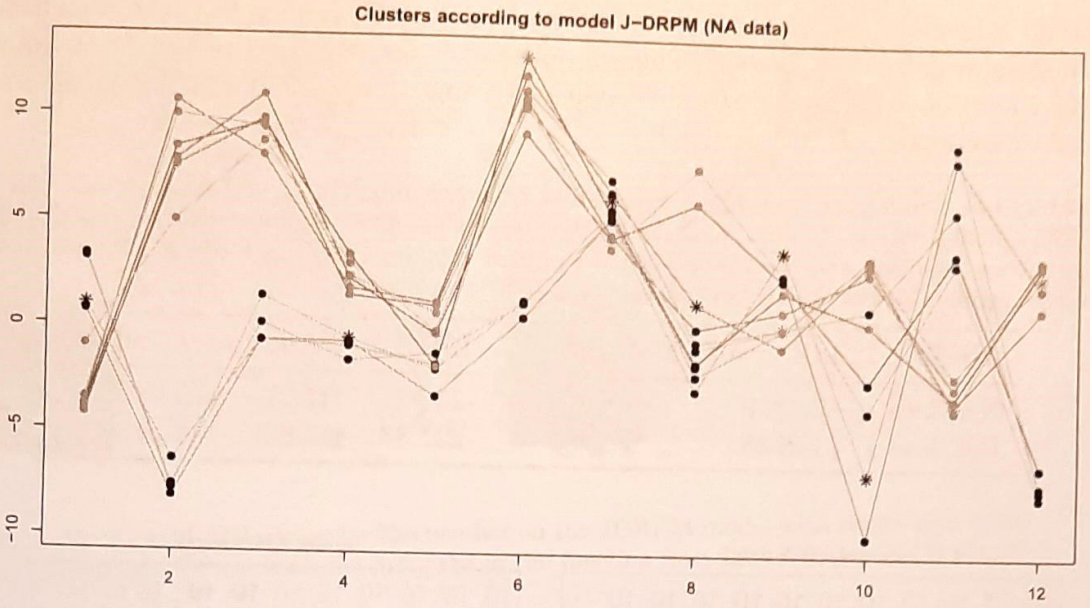

Figure 4.12: Visual representation of the clusters produced by the model, with units' labels represented as colored dots overlaid to the trend of the original generated target variables. Special point markers are used for the data points corresponding to missing values.

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

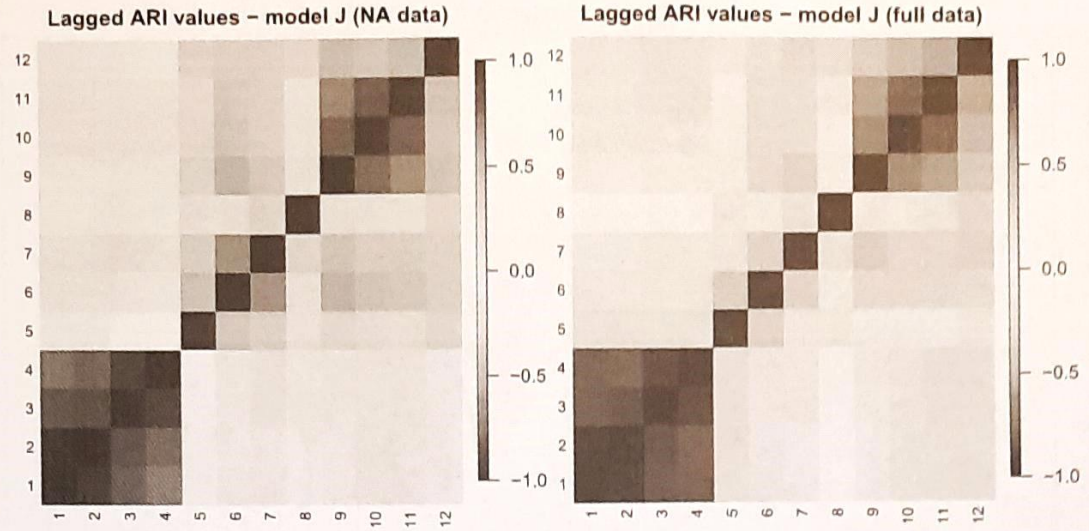

Figure 4.13: Lagged ARI values for the two fits on the JDRPM model with target plus space values. The partitions were estimated using the `salso` function from the corresponding R library.

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

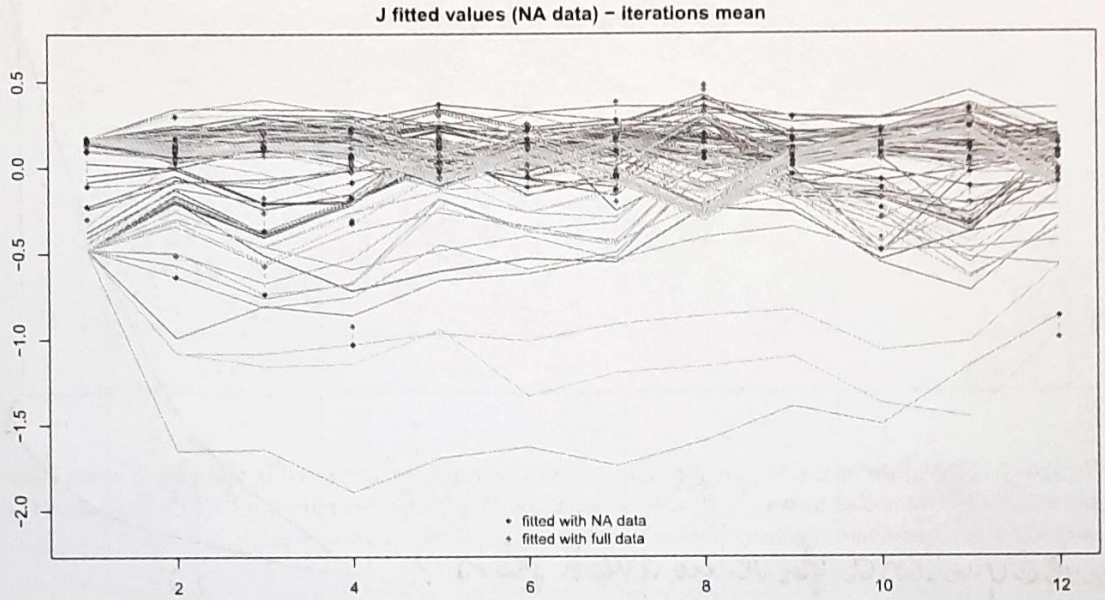

Figure 4.14: Fitted values estimate trough the mean of the 4000 iterates generated by model JDRPM. The generated target values were the same of Figure 4.7. Special point markers are used for the data points which were missing, to highlight the gaps between the fitted values on the full dataset (green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

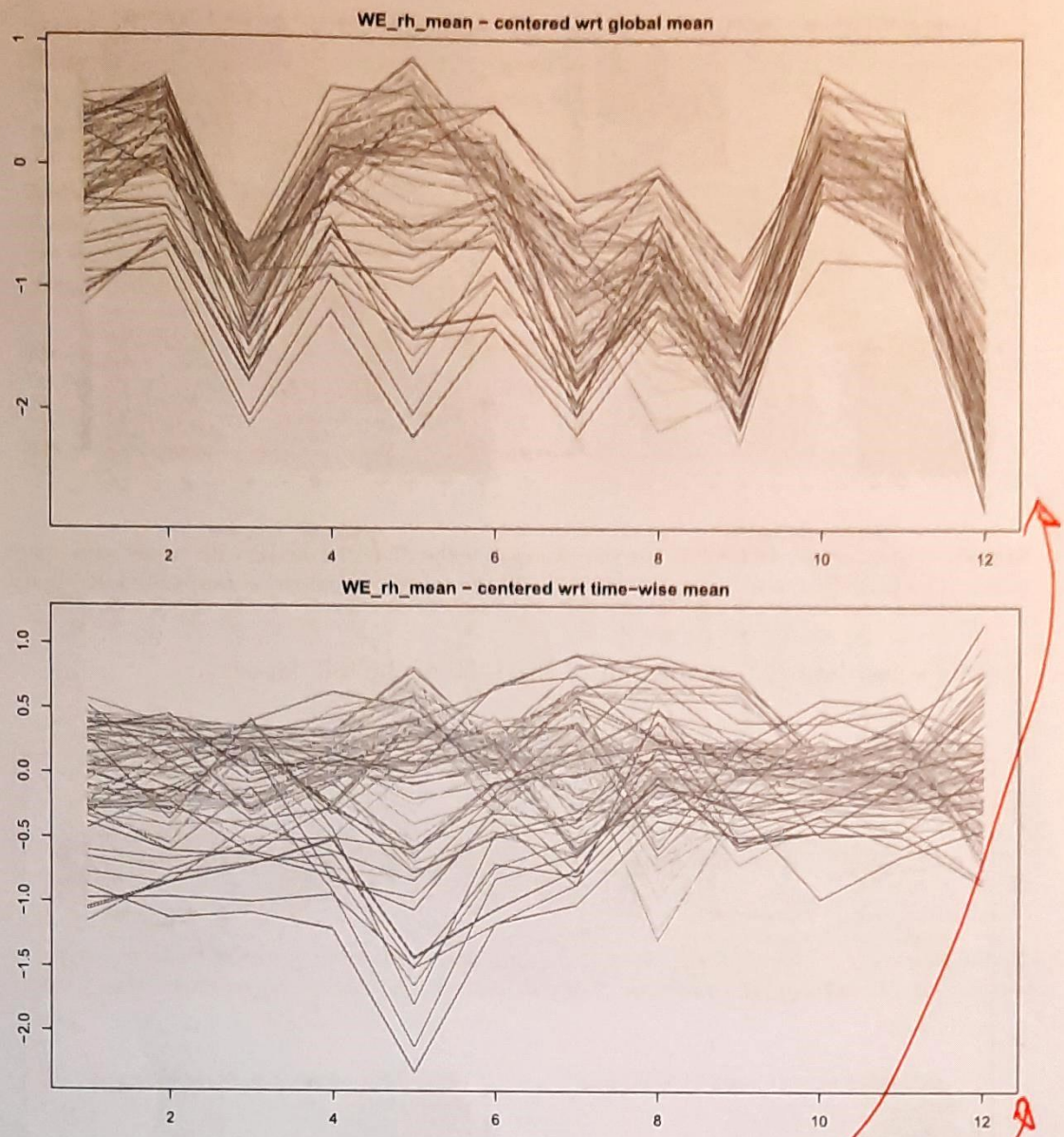

Figure 4.15: Values of the `WE_rh_meancentered` covariate corrected using the global mean (top) and using the time-wise mean (bottom). Coloring is based on the ranking of PM_{10} values of the units according to their median, from highest (red) to lowest (blue).

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$(a =$