$1.9, b = 0.4$), 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.