Journal of Statistical Software

July 2019, Volume 90, Issue 9. doi: 10.18637/jss.v090.i09

slfm: An R Package to Evaluate Coherent Patterns in Microarray Data via Factor Analysis

João Daniel N. Duarte
UFMG-Brazil

Vinícius D. Mayrink
UFMG-Brazil

Abstract

The development of simulation-based methods, such as Markov chain Monte Carlo (MCMC), has contributed to an increased interest in the Bayesian framework as an alternative to deal with factor models. Many studies have used Bayesian factor analysis to explore gene expression data. We are particularly interested in the application of a sparse latent factor model (SLFM) based on sparsity priors (mixtures) to assess the significance of factors. The SLFM measures how strong the observed coherent expression pattern is in the data, which is an important source of information to evaluate gene activity. In the literature, this type of model has shown better results than other approaches intended for identification of patterns and metagene groups related to the underlying biology. However, a full Bayesian factor model relying on MCMC algorithms has an expensive computational cost, which makes it unattractive for general users. In this paper, we present the package **slfm** which uses C++ implementation via **Rcpp** to improve the computational performance of the SLFM within the widely used statistical tool R. We investigate real and simulated microarray data related to breast cancer.

Keywords: factor model, Bayesian inference, gene expression, sparsity prior, **slfm**, **Rcpp**.

1. Introduction

Markov chain simulation methods have been widely used to approximate the target posterior distribution in a Bayesian statistical analysis; see Gamerman and Lopes (2006) for details. These methods are general in the sense that they can be applied to explore different modeling strategies, including multilevel hierarchical structures with many parameters and latent variables. The MCMC algorithms are based on iterations where samples are drawn sequentially to build a Markov chain, the distribution of the current draws depends on the last generated values. Therefore, the computational cost can be a challenge when applying these algorithms to elaborated models. Naturally, this cost is also expensive for the analysis of high-dimensional data sets (e.g., gene expression data).

In large data set studies, most users would prefer to fit a simple model whose implementation runs faster than a robust model version proposed for the same problem. The lower the computational speed of the algorithm, the less attractive is the gain in terms of inference performance. As a possible solution for this difficulty, one could try to use computer clusters to accelerate the analysis by separating (if possible) the calculations in parts to be run independently in parallel, however, this solution may not be feasible for many reasons (e.g., no access to a cluster, limited machine memory). Another strategy is to consider programming languages that handle more efficiently the speed problem (e.g., C++).

Most programs for implementing and running C++ (e.g., **OxEdit**; Doornik and Ooms 2006) do not have the resources and popularity of R (R Core Team 2019) for a full statistical analysis. R is a well-known tool for statistical computing and inference, it runs in the main operational systems with supporting packages being constantly updated/created by users. In this paper, our main computational goal is to take advantage of the C++ speed together with the R resources through the well known package **Rcpp** (Eddelbuettel and François 2011; Francois and Eddelbuettel 2018; Eddelbuettel 2013) to implement Gibbs sampling (Gelfand and Smith 1990) for a factor model designed to study presence/absence of gene activities. This is a classification problem based on large expression data sets from Affymetrix microarrays.

In the literature, there are several studies using Bayesian factor analysis to identify patterns and underlying structures in genomic data, they often show interesting results. For example, West (2003) introduced the sparse latent factor models as a natural extension of sparse regression models. The study applies variable selection priors to test the significance of latent factors, which configures a mechanism to detect signature patterns in expression data. Lucas, Carvalho, Wang, Bild, Nevins, and West (2006) assume sparsity hierarchical priors for the detection of complex correlation patterns among genes. Carvalho, Chang, Lucas, Nevins, Wang, and West (2008) use sparsity priors to deal with dimensionality reduction in latent factor models.

We consider the sparse latent factor model structure proposed in Mayrink and Lucas (2015) to take advantage of the information about the behavior of expression values across samples (arrays). A consistent expression pattern suggests presence, whereas a random configuration indicates absence of gene activity. The paper shows a superior performance of the factor model over simpler classification methods: the usual MAS 5.0 P/A (Affymetrix 2001) and the alternative PANP approach (Warren, Taylor, Martini, Jackson, and Bienkowska 2007). Following this study, Duarte and Mayrink (2015) considered a more flexible version of the model assuming individual priors for each loading, however, the trade-off for the superior classification performance of the Bayesian model is a high computational cost, preventing its practical application.

The main contribution of this paper is to present and explore the R package **slfm** (Duarte and Mayrink 2019) created as a result of our efforts to use C++ within R via **Rcpp** to implement and thus improve the computational performance of the Bayesian factor analysis for large gene expression microarray data sets. This package includes functions for: model fitting, standard preprocessing procedures, graphical and descriptive outputs. The package is available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=slfm`. Here, we explain how to use the package and how to interpret its results.

The outline of this paper is as follows: Section 2 describes the microarray data and the detection problem motivating the factor analysis. This section also presents the sparse latent factor

model and discusses the MCMC algorithm and some inference aspects. Section 3 shows results confirming the good performance of the factor model and compares the computational cost of three implementation versions (MATLAB, The MathWorks Inc. 2018, and R with/without **Rcpp**). Section 4 explains how to use the R package **slfm** and includes some real data results. Finally, Section 5 indicates the main conclusions and final remarks.

2. SLFM model

In this section, we briefly describe the data, the motivation of the study and the chosen model.

2.1. Microarray data

We work with oligonucleotide Affymetrix GeneChip microarrays containing the expression of 22,283 genes (or probe sets) with respect to a tumor under investigation (e.g., breast cancer). In short, the data is obtained through a hybridization procedure involving complementary nucleic acids. Single RNA nucleotide strands, 25 bases long, are extracted from the tumor and labeled with fluorescent tags. These single strands are expected to connect to its complementary sequence, if found, in a specific spot within the small chip (array). The sequence in each array spot was built in a lab, i.e., they are known. Next, the array is washed to remove unmatched material and a laser is applied to activate the fluorescence. The light intensity is scanned and translated into an expression measurement, a high observed intensity may indicate many connections and thus the presence of gene activity.

In fact, a single observation in our data set is the light intensity registered for a probe (fraction of a gene) represented in the array. In this study, the term "gene" is used as a reference to a group of 11–20 probes (a probe set). In order to evaluate the presence of gene activity one should evaluate the common behavior of all intensities belonging to a probe set in a microarray (high brightness for all probes indicates presence). A combination of low and high values, within the same array, suggests noise observations originated by different reasons: cross-hybridization, dust, chip defect, etc.

We work with many arrays, i.e., we observe replications for the intensity of each probe. The microarrays are treated as independent samples of expression values produced under the same conditions, for the same type of tumor, but for different individuals. Given this configuration, the coherent expression pattern observed across the arrays is a powerful source of information to study the gene activity. We have strong evidence of gene activity, if all probes within a probe set exhibit a similar pattern of intensities across arrays. Figure 1 presents two matrices with distinct expression patterns across the columns. The left panel shows a strong coherent pattern indicating presence of gene activity. The random pattern, exhibited in the right panel, suggests absence of activity. The factor model is an appropriate tool to deal with these data structures.

2.2. Preprocessing the data

Besides the elements affecting the observations within a chip (indicated in the previous section), the intensities between chips are subject to distortions due to: the amount of RNA in the sample, camera exposure time, scanner calibration, etc. As a result, it is necessary – this is a standard procedure for microarray data – to preprocess the raw intensities to remove unwanted noise effects before fitting any model.

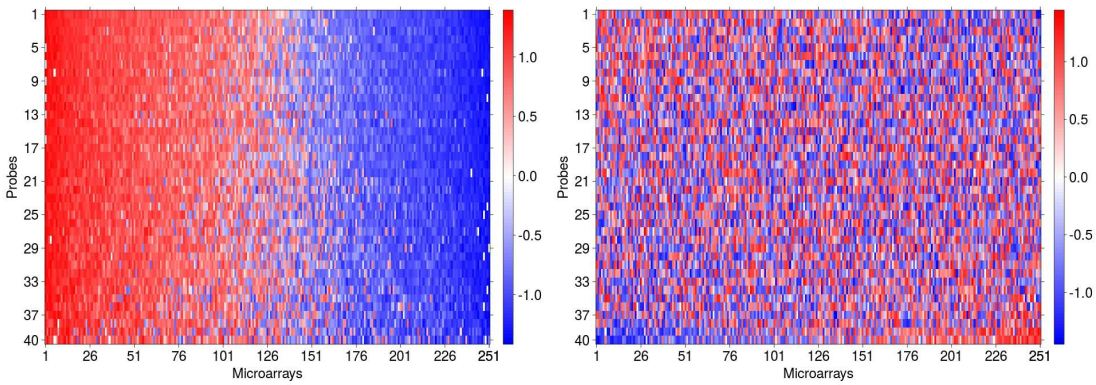

Figure 1: Image of data matrices displaying a strong (left) and random (right) patterns.

Let $Y^{(k)}$ be a $(m^{(k)} \times n)$ matrix, containing the raw data related to the probe set k, such that each row represents a probe and each column represents a microarray. Following Mayrink and Lucas (2015); Duarte and Mayrink (2015), we use the preprocessing steps: (i) organize the big matrix $Y = [Y^{(1)}, Y^{(2)}, \ldots, Y^{(K)}]^\top$ with K being the total number of probe sets, (ii) divide the probe intensities on each array by the global mean intensity of that array, (iii) apply the log transformation, (iv) standardize the rows, (v) obtain the first principal component (p_{c1}) of Y and subtract it from each row of Y, i.e., $Y_i - Y_i\, p_{c1}\, p_{c1}^\top$. As a result, consider $X^{(k)}$ as the preprocessed data matrix for the kth probe set (we will omit the "(k)" index in the further notation).

The step (v) is required to remove a systematic effect common to all probe sets. This effect could be modeled through a dependency parameter affecting all $X^{(k)}$'s, which would bring computational difficulties to fit the Bayesian model since its posterior distribution depends on the whole data set. The preprocessing step (v) allows us to model each $X^{(k)}$ separately. If K is too large (e.g., 22,283), the p_{c1} computation can be cumbersome due to lack of memory to load Y. In this case, we randomly select a subset of $Y^{(k)}$ matrices for the principal component analysis.

There are other preprocessing methods available in the literature, the most popular being the robust multi-chip average (RMA) method; see Irizarry *et al.* (2003). We are not using the RMA procedure for three main reasons: first, it includes a summarization step through median polish producing a single probe set intensity, the RMA background adjustment step relies on parametric modeling assumptions involving the exponential and normal distributions to separate signal from noise terms and, finally, RMA does not use mismatch probes, which also contain important information for the coherent pattern analysis. The adopted non-parametric preprocessing method in our study retains both mismatch and perfect-match probes.

2.3. SLFM-U model

The factor model presented here was evaluated in Duarte and Mayrink (2015). Let X be an $(m \times n)$ preprocessed data matrix for some probe set. Again, the ith row represents a probe and the jth column represents a sample (microarray). Consider the model $X = \alpha\lambda + \epsilon$, with α being the $(m \times L)$ loadings matrix, λ is the $(L \times n)$ factor scores matrix and ϵ is the $(m \times n)$

noise matrix. Assume $\epsilon_{ij} \sim N(0, \sigma_i^2)$, i.e., each row has a different variance. In terms of notation, consider that L is the number of factors in the model, $\sigma^2 = (\sigma_1^2, \sigma_2^2, \ldots, \sigma_m^2)^\top$ and $\sigma_{-i}^2 = (\sigma_1^2, \ldots, \sigma_{i-1}^2, \sigma_{i+1}^2, \ldots, \sigma_m^2)^\top$.

We want to evaluate how coherent the patterns in the rows of X are. As indicated in Mayrink and Lucas (2015); Duarte and Mayrink (2015) a model assuming $L = 1$ would be appropriate for this task, i.e., the coherent pattern in X could be explained by a non-null vector α multiplying the scores in λ (vector λ represents the underlying pattern). The single factor specification has lower computational cost, which is the main focus of this paper.

In a conjugate analysis, we assume the priors: $\sigma_i^2 \sim IG(a, b)$ (Inverse Gamma) and $\lambda_j \sim N(0, 1)$. The latter choice is a standard strategy in factor analysis to control the magnitude of λ within the product $\alpha\lambda$. We consider a univariate (U in SLFM-U) spike and slab mixture prior for each loading. These bimodal sparsity promoting priors are key elements in the structure of the model. Their formulation originated in the context of Bayesian variable selection, and it has been the subject of substantial research; see George and McCulloch (1993, 1997); Geweke (1996). There are two types of mixtures that can be used:

- A mixture with degenerated and normal components (MDN): The spike part of the mixture is a distribution degenerated at 0. Consider the formulation: $\alpha_i \sim (1 - Z_i)\delta_0(\alpha_i) + Z_i N(0, \omega_1)$, where $\omega_1 > 0$ is fixed, $\delta_0(\alpha_i)$ means $\mathsf{P}(\alpha_i = 0) = 1$.
- A mixture with two normal components (MNN): The spike part of the mixture is a normal distribution with very small variance. Consider the formulation: $\alpha_i \sim (1 - Z_i)N(0, \omega_0) + Z_i N(0, \omega_1)$, with $\omega_0 > 0$ small and ω_1 large with fixed values chosen by the researcher. As opposed to MDN, this mixture allows the classification of loadings too close to zero as not significant.

Also, $Z_i \sim \text{Bernoulli}(q_i)$ and $q_i \sim \text{Beta}(\gamma_1, \gamma_2)$. Note that q_i is a prior probability weight for the component indicating $\alpha_i \neq 0$. In contrast to this configuration, Mayrink and Lucas (2015) model the whole vector α through a multivariate mixture.

The likelihood can be expressed in two different ways. Depending on the parameters being evaluated, one version is more suitable than the other to simplify the posterior calculations.

- Likelihood 1: Denote $X_{\cdot j}$ as the jth column of X, then $(X_{\cdot j}|\alpha, \lambda_j, \sigma^2) \sim N_m[\alpha\lambda_j, D]$ with $D = \text{diag}(\sigma_1^2, \sigma_2^2, \ldots, \sigma_m^2)$. Assuming conditional independence between columns of X, we have $p(X|\alpha, \lambda, \sigma^2) = \prod_{j=1}^n p(X_{\cdot j}|\alpha, \lambda_j, \sigma^2)$.
- Likelihood 2: Let $X_{i\cdot}$ be the ith row of X, then $(X_{i\cdot}^\top|\alpha_i, \lambda, \sigma_i^2) \sim N_n[\lambda^\top\alpha_i, \sigma_i^2 I_n]$. The conditional independence between rows provides $p(X|\alpha, \lambda, \sigma^2) = \prod_{i=1}^m p(X_{i\cdot}|\alpha_i, \lambda, \sigma_i^2)$.

The full conditional posterior distributions, required in Gibbs sampling, are:

- Likelihood 1 provides
$$(\lambda_j|\alpha, \lambda_{-j}, \sigma^2, X) \sim N(M_\lambda, V_\lambda)$$
with $V_\lambda = (\alpha^\top D^{-1}\alpha + 1)^{-1}$ and $M_\lambda = V_\lambda(\alpha^\top D^{-1} X_{\cdot j})$.
- It holds
$$(\sigma_i^2|\alpha, \lambda, \sigma_{-i}^2, X) \sim IG(A, B),$$
with $A = a + (n/2)$ and $B = b + (1/2)(X_{i\cdot}X_{i\cdot}^\top - 2\alpha_i\lambda X_{i\cdot}^\top + \alpha_i\lambda\lambda^\top\alpha_i^\top)$.

- If $Z_i = 1$ (using Likelihood 2),

$$(\alpha_i|\alpha_{-i}, \lambda, \sigma^2, Z, X) \sim N(M_{[\alpha,1]}, V_{[\alpha,1]})$$

with $V_{[\alpha,1]} = [\frac{1}{\omega_1} + \frac{1}{\sigma_i^2}\sum_{j=1}^{n}\lambda_j^2]^{-1}$ and $M_{[\alpha,1]} = V_{[\alpha,1]}[\frac{1}{\sigma_i^2}\sum_{j=1}^{n}\lambda_j X_{ij}]$.

- If $Z_i = 0$,
  - with MDN,
$$(\alpha_i|\alpha_{-i}, \lambda, \sigma^2, Z, X) \sim \delta_0(\alpha_i);$$
  - with MNN and using Likelihood 2,
$$(\alpha_i|\alpha_{-i}, \lambda, \sigma^2, Z, X) \sim N(M_{[\alpha,0]}, V_{[\alpha,0]})$$
with $V_{[\alpha,0]} = [\frac{1}{\omega_0} + \frac{1}{\sigma_i^2}\sum_{j=1}^{n}\lambda_j^2]^{-1}$ and $M_{[\alpha,0]} = V_{[\alpha,0]}[\frac{1}{\sigma_i^2}\sum_{j=1}^{n}\lambda_j X_{ij}]$.

- It holds
$$(q_i|Z_i) \sim \text{Beta}(\gamma_1^*, \gamma_2^*),$$
with $\gamma_1^* = \gamma_1 + Z_i$ and $\gamma_2^* = \gamma_2 + 1 - Z_i$.

- The mixture posterior distribution for α_i has the probability weight:
  - If MDN,
$$q_i^* = p(Z_i = 1|\alpha, \lambda, \sigma^2, q_i, X) = \frac{q_i}{q_i + \frac{N[0|M_{[\alpha,1]}, V_{[\alpha,1]}]}{N[0|0,\omega_1]}(1 - q_i)}. \quad (1)$$
  - If MNN,
$$q_i^* = p(Z_i = 1|\alpha, \lambda, \sigma^2, q_i, X) = \frac{q_i}{q_i + \frac{N[0|0,\omega_0]}{N[0|M_{[\alpha,0]}, V_{[\alpha,0]}]}\frac{N[0|M_{[\alpha,1]}, V_{[\alpha,1]}]}{N[0|0,\omega_1]}(1 - q_i)}. \quad (2)$$

2.4. MCMC algorithm

We apply the Gibbs sampling algorithm to sample from the joint posterior distribution $p(\alpha, \lambda, \sigma^2|X)$. The required full conditional distributions are given in the previous section. In terms of prior specifications, **slfm** allows the user to set custom values for the hyperparameters: a, b, γ_1, γ_2, ω_0 and ω_1. The user can also select the type of mixture (MDN or MNN) to model the loadings.

The first step of the algorithm is to set initial values for all parameters. Next, conditional on the current values of other parameters, consider the updating sequence:

[1st] sample σ_i^2 from $IG(A, B)$,

[2nd] sample Z_i from Bernoulli(q_i^*) with (1) (if MDN prior) or with (2) (if MNN prior),

[3rd] sample q_i from Beta(γ_1^*, γ_2^*),

[4th] if $Z_i = 1$, sample α_i from $N(M_{[\alpha,1]}, V_{[\alpha,1]})$,

[5th] if $Z_i = 0$ and MDN, set $\alpha_i = 0$, otherwise ($Z_i = 0$ and MNN) sample α_i from $N(M_{[\alpha,0]}, V_{[\alpha,0]})$,

[6th] sample λ_j from $N(M_\lambda, V_\lambda)$.

Repeat this updating sequence until convergence is observed and the posterior sample is obtained.

A fast convergence can be observed for all parameters of the factor model proposed for the detection problem evaluated in this paper. A convergence study is presented in Section 3.1.

2.5. Estimation and interpretation

Assuming a quadratic loss function, we consider the posterior mean to estimate the parameters and thus minimize the corresponding expected loss. For σ_i^2, λ_j, Z_i and q_i^* the posterior sample includes the values generated after the burn-in period. An integer lag argument may be specified by the user to select spaced observations for the final sample, which is a well known strategy to reduce the chain's autocorrelation. If the lag is set to 1, all iterations are saved after the burn-in.

For each α_i, the generated value in the current iteration – taking into account the lag separation – is saved together with the indicator Z_i. The inference for the ith loading is developed separately for each group (samples with $Z_i = 1$ and $Z_i = 0$). This strategy prevents a deceiving inference based on a combined chain involving MCMC draws from both posterior mixture components. In order to test, whether α_i is "Significant" (S), "Inconclusive" (I) or "Not significant" (N), we evaluate the 95% highest probability density (HPD) credible interval of q_i^*. Let L_{inf} and L_{sup} be the interval's lower and upper limits. The ith loading is said: "S" if $0.5 < L_{\text{inf}}$ (take the posterior mean to estimate α_i), "I" if $L_{\text{inf}} < 0.5 < L_{\text{sup}}$, and "N" if $L_{\text{sup}} < 0.5$ (α_i is estimated as 0). Looking at all α_i's classifications, the most frequent response will indicate "Presence" (P), "Marginal" (M) or "Absence" (A) of gene activity. In this work, we consider the package **coda** (Plummer, Best, Cowles, and Vines 2006) to compute HPD intervals.

3. MCMC implementation

R can be very slow for iterative simulation-based algorithms such as Gibbs sampling and other MCMC methods. This is because R is an interpreted language and every instruction must be compiled to native machine code at runtime. Compiled languages such as C/C++ are a faster alternative for this type of algorithm.

In order to improve the computational performance and avoid MCMC iterations, one may also think about fitting the proposed factor model through the approximate Bayesian inference produced by the INLA method (Rue, Martino, and Chopin 2009). This alternative is not feasible here for two main reasons: (i) INLA only works for a subclass of structured additive regression models (latent Gaussian models) and the proposed factor model has a multiplicative structure involving α_i and λ_j, (ii) we assume a sparsity prior (mixture) for α_i which cannot be implemented in INLA.

Despite the inferior computational performance, R has attractive benefits such as functions that make it easy to read data, produce graphical results and generate summary statistics.

Implementation	Minimum	25% percentile	Median	75% percentile	Maximum
Rcpp	143.94	148.10	149.85	152.18	160.70
R	6201.09	6311.93	6345.47	6402.57	6573.78
MATLAB	4228.16	4231.26	4238.36	4249.24	4257.55

Table 1: Execution times (in milliseconds) to fit the SLFM-U MNN to 100 data matrices (20×100).

Rcpp is an R package allowing the user to embed C++ code into R, thus combining the advantages of C++ speed and the R utilities. Note that it is possible to write C code to be used in R without **Rcpp**, but this package provides a clean interface making this task easier. In addition to **Rcpp**, we consider the R package **RcppArmadillo** (Eddelbuettel and Sanderson 2014) which integrates the **Armadillo** templated linear algebra library to simplify the code in terms of matrix and vector operations.

The usual Gibbs sampling implementation of the proposed factor model in R, without **Rcpp**, is extremely simple. If working in the R environment with gene expression data, the user can take advantage of several contributed packages available on CRAN or integrated into the **Bioconductor** project (`https://www.Bioconductor.org/`), a collaborative effort providing softwares for computational biology (Gentleman *et al.* 2004). Another interesting programming language simple to implement and widely used in gene expression research is MATLAB. The **Bioinformatics Toolbox**, in MATLAB, provides functions and algorithms for microarray analysis and other bioinformatics workflows. MATLAB can be faster than R, especially in applications involving large matrices operations such as singular value and Cholesky decompositions, however, the combination R with **Rcpp** will be computationally superior for the factor model explored in this study (see next section).

3.1. Comparative studies

All studies developed in this section consider the following prior specifications: $\sigma_i^2 \sim IG(a = 2.1, b = 1.1)$ with expected value 1 and variance 10, $q_i \sim \text{Beta}(\gamma_a = 1, \gamma_b = 1) = U(0, 1)$ and the components $N(0, \omega_0 = 0.01)$ (for MNN) and $N(0, \omega_1 = 10)$ (MNN and MDN).

The first study comparing the three implementation options (R, **Rcpp** and MATLAB) involves 100 randomly generated (20×100) data matrices with $X_{ij} \sim N(0, 1)$, for $i = 1, \ldots, 20$ and $j = 1, \ldots, 100$. The focus here is to investigate the execution time required to fit the SLFM-U (MNN version) to the 100 simulated random matrices. We consider 1,000 iterations of the Gibbs sampling algorithm. Table 1 shows a summary of the results obtained using the auxiliary functions (`system.time` in R) and (`cputime` in MATLAB) to measure the computational time. All simulations were performed on the same computer with Intel Core i5 processor and 4.00 GB memory.

Table 1 indicates that the R implementation is the slowest option, whereas **Rcpp** is remarkably the fastest implementation. The MATLAB version for the SLFM-U is slightly faster than the usual R version (the median indicates 33% less time to complete the task). There is a large gap between the median execution time of the **Rcpp** version and the other two, i.e., **Rcpp** takes 97.6% and 96.5% less time than R and MATLAB, respectively.

Now, we present a convergence analysis focused on the **Rcpp** implementation of the SLFM-U. The main aim is to show that the Gibbs sampling algorithm is converging, as expected, to

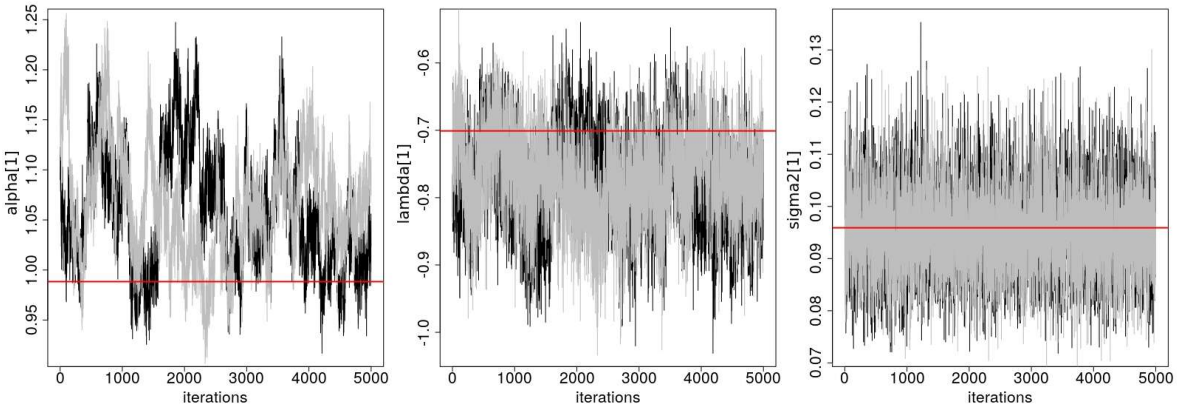

Figure 2: Trajectory of two chains (gray and black) generated for α_1, λ_1 and σ_1^2 via the factor model, with mixture MNN, applied to the simulated data matrix with strong pattern in Figure 1. The horizontal line indicates the true parameter value.

	MDN prior		MNN prior	
Param.	Strong	Weak	Strong	Weak
α	100.0% (1.0450)	97.5%* (1.0475)*	100.0% (1.1727)	97.5% (1.0292)
λ	88.0% (1.0067)	94.4% (1.0004)	98.8% (1.0880)	99.2% (1.0017)
σ^2	95.0% (1.0006)	100.0% (0.9999)	92.5% (1.0002)	90.0% (1.0002)

*Value obtained for a chain mostly constant at zero due to $\delta_0(\alpha_i)$.

Table 2: Percentages of Geweke z score statistics within the interval $(-1.96, 1.96)$. In parentheses, Gelman and Rubin potential scale reduction factor estimate based on two chains for α_1, λ_1 and σ_1^2.

the target distribution. Consider here the following MCMC setup: burn-in period with 500 iterations, lag = 1 and a posterior sample of size 5,000. The relatively short burn-in was chosen based on preliminary analyses evaluating graphs displaying the trajectory of the chains. They indicate that the convergence status is visually true after few iterations (less than 500) for all parameters. Figure 2 shows trace plots representing the trajectory of two independent chains generated for α_1, λ_1 and σ_1^2 in the scenario "MNN + Strong pattern". The behavior of the chains for λ_1 and σ_1^2 are quite similar in other mixture and pattern configurations. The chain of α_1 also behaves, in terms of autocorrelation, as shown in Figure 2 for the scenario "MDN + Strong pattern". In "MNN + Weak pattern" the mixing of α_1 resembles the Figure 2 scenario, but the trajectory is concentrated around zero, as expected for weak pattern data. In "MDN + Weak pattern", most values ($\approx$ 98% of the iterations) are 0 due to $\delta_0(\alpha_1)$, therefore, the corresponding trace plot is mostly constant at zero.

In this convergence study, we have also applied, for some parameters, the diagnostic methods proposed in Geweke (1992) and Gelman and Rubin (1992); they are available in the R package **coda**. We remind the reader that Geweke's diagnostic is based on a test for equality of the means of the first and last part of a Markov chain after a burn-in period (by default the first 10% and last 50%). If the posterior sample is drawn from the stationary distribution, the two means are equal and the test statistic asymptotically follows the $N(0, 1)$. Gelman and Rubin's diagnostic is based on the analysis of more than one chain generated for the same parameter.

Implementation	Experiment execution time (min)	Average execution time per matrix (sec)
Rcpp	4.67	0.56
R	169.02	20.28
MATLAB	159.67	19.16

Table 3: Execution times for the three SLFM-U implementations. Column 2: time (in minutes) required to fit all X matrices. Column 3: average time (in seconds) required to fit a single X matrix. These matrices may differ in number of rows (number of columns = 251).

In brief, a "potential scale reduction factor" is calculated for each chain and convergence is confirmed when this statistic is close to 1.

Table 2 presents results for both convergence diagnostic methods. For the chain of each θ_i in θ (general notation), we calculate the Geweke z scores and then evaluate the percentage of these z scores within the $N(0,1)$ symmetric 95% interval around zero. Note that the smallest percentage is 88.0% suggesting that most chains have passed the convergence test. In order to improve these results, the user can fit the model assuming a longer chain and a larger lag. Table 2 also shows (in parentheses) the Gelman and Rubin's diagnostic statistic for the chains of α_1, λ_1 and σ_1^2. As can be seen, most values are close to 1 suggesting convergence. The largest statistic (1.1727) is obtained for α_1 in the scenario "MNN + Strong pattern".

Before moving to the second comparative study, we advise the reader that the final posterior estimates may differ when fitting the proposed model to the same data matrix in two consecutive and independent runs of the MCMC. This result is expected and it is a consequence of the MCMC method, which generates different chains for the same parameter in each run. Figure 2 shows this difference and also highlights the fact that both chains converge to the same region in the parameter space. The estimates tend to be more consistent with longer runs of the Gibbs sampler.

In our second comparative study, we have simulated 500 matrices based on the data set presented in Miller *et al.* (2005), which involves 251 Affymetrix oligonucleotide microarrays representing independent samples containing information about the expression of genes in breast cancer tumors. The following steps were considered to generate the data:

1. Compute the correlation matrix displaying the linear association between rows (probes) of X for each one of the 22,283 real data matrices. This strategy gives us an insight about the pattern strength in each X.
2. Select 500 matrices, 200 with overall low correlations and 300 with overall moderate to high correlations.
3. Fit the SLFM-U to each of those selected 500 matrices and then obtain the posterior means $\{\hat{\alpha}, \hat{\lambda}, \hat{\sigma}^2\}$. The MCMC setup is as follows: burn-in = first 500 iterations, lag argument = 1, posterior sample size = 1,500.
4. Simulate 500 matrices using $X = \hat{\alpha}\hat{\lambda} + \epsilon$, with $\epsilon \sim N(\mathbf{0}, \text{diag}\{\hat{\sigma}^2\})$. Set $\alpha = \mathbf{0}$ for the 200 matrices with low correlation.

Table 3 shows again that the **Rcpp** version is the fastest option. The conclusions here are the

same as those obtained from Table 1. It is important to emphasize that the **Rcpp** implementation spends, on average, less than 1 second to run the MCMC for a single X.

In this simulation study involving 200 weak pattern and 300 strong pattern matrices, we have observed that all weak pattern cases were correctly detected as "Absent" by both models (MDN and MNN mixture). All strong pattern cases were correctly identified as "Present" by the MNN model, whereas the MDN mixture provided 98% of correct classification "Present", 0.33% of incorrect "Absent" (1 matrix out of 300) and 1.67% "Marginal". The strong pattern group involves matrices with intermediate to strong patterns. The MNN model can manage well intermediate cases, on the other hand, the MDN model has some difficulty to estimate q_i^* for a close to random pattern matrix. In this case, the 95% HPD interval of q_i^* is too wide (including 0.5) due to the $\delta_0(\alpha_i)$ component, which favors "I" calls for several loadings. A majority of rows with "S" calls is thus not observed given that some of these rows have received the "I" call. These results are similar for all three implemented versions of the factor model.

In order to evaluate the inference and MCMC performances of all implemented versions of the SLFM-U, we have considered the two (40×251) simulated matrices shown in Figure 1. These matrices were simulated using the same procedure indicated in Steps 3 and 4 of the previous analysis involving 500 matrices. Here, consider an MCMC setup with burn-in period of 2,000 iterations, lag argument = 1 and posterior sample size = 1,000. We explore results using two statistics: mean squared error (MSE) and effective sample size (n_{eff}). Their formulations are given by:

$$\text{MSE}(\theta) = \frac{1}{d}\sum_{i=1}^{d}(\theta_i - \hat{\theta}_i)^2 \quad \text{and} \quad n_{\text{eff}} = \frac{N}{1 + 2\sum_{h=1}^{H}\hat{\rho}_h}. \tag{3}$$

Denote $\theta = (\theta_1, \ldots, \theta_d)$ as the vector of the true values for d parameters of the model, and $\hat{\theta}$ is the corresponding vector of posterior estimates. We evaluate the bias of the model through the MSE (large MSE means high bias). The statistic n_{eff} in (3) summarizes the autocorrelation of the chain. Let N be the size of the posterior sample (after burn-in period) and

$$\hat{\rho}_h = \frac{\sum_{t=h+1}^{N}[\theta_i^{(t)} - \bar{\theta}_i][\theta_i^{(t-h)} - \bar{\theta}_i]}{\sum_{t=1}^{N}[\theta_i^{(t)} - \bar{\theta}_i]^2} \quad \text{with} \quad \bar{\theta}_i = \sum_{t=1}^{N}\theta_i^{(t)}$$

is the estimated autocorrelation of lag $h = 1, \ldots, H$ for a suitably large value of $H < N$. In our analysis, we set $H = 50$. Finally, consider $\theta_i^{(t)}$ as the tth value of the chain for θ_i.

An important issue in the analysis of serial data (e.g., a Markov chain) is to decide whether the observations come from a process based on independent random variables. In practice, a chain obtained via MCMC has an autocorrelation structure defining dependence between near observations. A strong autocorrelation can compromise the quality of the posterior estimates. In a random sample, autocorrelations are 0, and thus $n_{\text{eff}} = N$. In most cases, the denominator of n_{eff} is larger than 1 resulting in $n_{\text{eff}} < N$, which suggests that the actual amount of information available is lower than that of a random sample. The n_{eff} statistic, explored in this paper, is obtained from chains built with lag argument 1. The user may choose a larger lag to reduce autocorrelation and increase n_{eff}. Further details and examples regarding the n_{eff} statistic can be found in Gamerman and Lopes (2006) and Mayrink and Gamerman (2009).

Figure 3 compares the posterior means of some parameters obtained from each implemented version of the model. The graphs also include the true values and horizontal bars (from

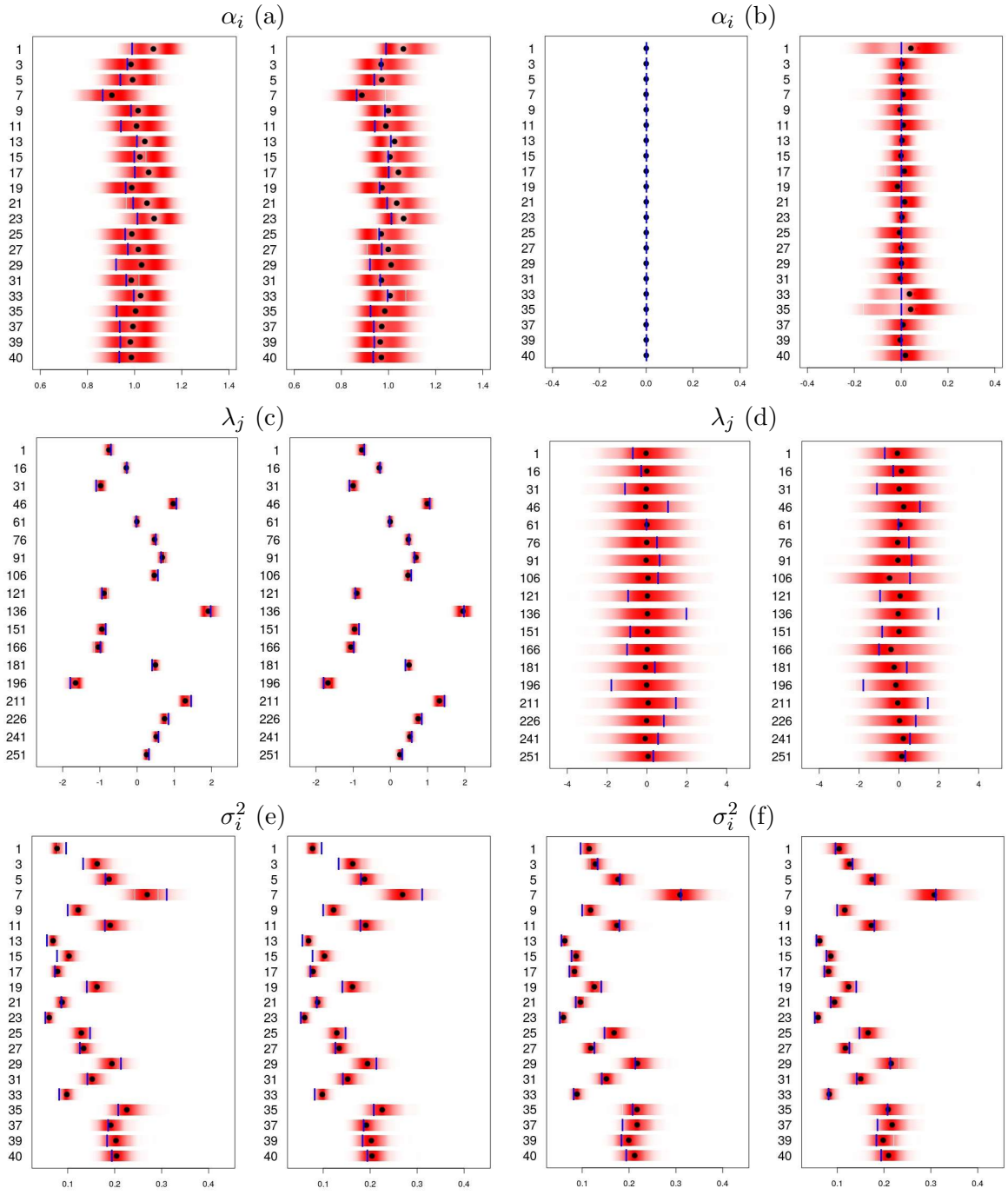

Figure 3: Posterior means (three black circles, one for each implementation), **Rcpp** posterior distribution (horizontal bars) and true values (vertical bar mark). Estimates are obtained for a matrix with strong pattern (Panels: a, c, e) and a matrix with random pattern (Panels: b, d, f). In each panel we have: MDN (left) and MNN (right).

		Median amp.			MSE			Median n_{eff} (%)		
Implemen.	Prior	α	λ	σ^2	α	λ	σ^2	α	λ	σ^2
Rcpp	MDN	0.231	0.256	0.054	0.004	0.007	0.040*	1.327	3.252	117.895
	MNN	0.223	0.257	0.056	0.002	0.006	0.040*	1.340	3.503	107.271
R	MDN	0.202	0.247	0.054	0.006	0.006	0.035*	1.467	4.640	116.404
	MNN	0.185	0.259	0.055	0.001	0.003	0.035*	1.533	5.204	104.560
MATLAB	MDN	0.174	0.235	0.054	0.006	0.006	0.035*	2.104	9.305	111.982
	MNN	0.174	0.244	0.053	0.001	0.003	0.035*	2.179	10.131	100.489

*Value should be multiplied by 10^{-2} for original scale.

Table 4: Median amplitude of the 95% HPD intervals, mean squared error (MSE) and median effective sample size relative to the posterior sample size ($100\, n_{\text{eff}}/N$ %) for the SLFM-U fitted to a simulated strong pattern matrix. Results for MATLAB were obtained with version R2010a.

		Median amp.			MSE			Median n_{eff} (%)		
Implemen.	Prior	α	λ	σ^2	α	λ	σ^2	α	λ	σ^2
Rcpp	MDN	0.000	3.862	0.051	0.899	0.735	0.023*	100	136.899	120.662
	MNN	0.197	3.342	0.054	0.878	0.556	0.024*	4.434	7.322	81.480
R	MDN	0.000	3.840	0.049	0.899	0.740	0.023*	100	106.571	123.693
	MNN	0.202	3.411	0.053	0.887	0.666	0.026*	3.394	5.925	72.402
MATLAB	MDN	0.000	3.855	0.051	0.899	0.745	0.023*	100	107.828	110.247
	MNN	0.201	3.355	0.054	0.885	0.632	0.026*	3.482	8.172	82.349

*Value should be multiplied by 10^{-2} for original scale.

Table 5: Median amplitude of the 95% HPD intervals, mean squared error (MSE) and median effective sample size relative to the posterior sample size ($100\, n_{\text{eff}}/N$ %) for the SLFM-U fitted to a simulated random pattern matrix. Results for MATLAB were obtained with version R2010a.

the **Rcpp** version) summarizing the posterior distribution. The results are similar for the parameters not included in these graphs. The posterior estimates displayed in Figure 3 for the three implemented versions are, as expected, close to each other and, in most cases, close to the targeted true value. In addition, the majority of true values are within the interval region covered by the posterior distribution. Panels (a) and (b) summarize the α_i samples from the posterior mixture component with the highest probability. As can be seen, in the strong pattern scenario (a) the model correctly generates values from the Gaussian component with large variance. Panel (b) shows the weak pattern data analysis, with posterior estimates concentrated at (MDN) or around (MNN) zero. Panel (d) shows that the range of the posterior distribution is too wide compared to those exhibited in Panel (c). This can be explained by the fact that the loadings in the weak pattern analysis are estimated as 0 (or too close to 0) and they multiply the λ_j's, therefore, the magnitude of the factor scores does not matter for the model fit, which makes these latent variables difficult to estimate in this scenario.

Tables 4 and 5 show summary statistics for all parameters, i.e., not only those exhibited in Figure 3. We evaluate the MSE in (3), the median HPD amplitude and the median percentage $100\, n_{\text{eff}}/N$%. As an example, we calculate the HPD amplitude (or the effective sample size

percentage) for each α_i, $i = 1, \ldots, m$, and take the median of these results to report in the tables. The same procedure is applied to λ and σ^2. The median is chosen for the analysis since it is less affected by outliers and this is a concern when fitting the mixture for the loadings. Note that if few elements in α are related to the degenerated component, their HPD amplitudes will be 0 (the chain is constant at 0) configuring atypical values among the amplitudes computed for other α_i's.

Looking at each parameter, Table 4 shows that the amplitude of the HPD intervals for **Rcpp**, R and MATLAB are, as expected, similar and small. The biases are also small, suggesting a good performance of the model in terms of inference (the largest MSE is 0.007). Table 5 correctly indicates MSE and median amplitude as 0 for loadings associated with the degenerated component of the MDN mixture. In the random pattern configuration, the median HPD amplitudes for λ are larger than those for other parameters. This can be explained by the product issue in $\alpha_i \lambda_j$ (with $\alpha_i \approx$ or $= 0$) as mentioned above. In both pattern scenarios, the smallest median HPD amplitude and MSE are observed for σ^2.

In some cases, not often, negative (near zero) estimates $\hat{\rho}_h$ may be obtained (for some h) leading to $-1 < 2\sum_{h=1}^{H} \hat{\rho}_h < 0$. As a consequence, we may observe n_{eff} slightly larger than N. This result does not compromise the conclusion about the autocorrelation. Here, the autocorrelation level should be regarded as low. One may consider an alternative version of (3) using $|\hat{\rho}_h|$, however, this will ignore possible negative associations between observations and determine smaller effective sample sizes.

Tables 4 and 5 also present the median effective sample size percentage for each vector of parameters, type of implementation and mixture model. Results of this statistic for α_i are based only on the iterations having values generated from the mixture component with the highest probability. Note that the autocorrelation is higher for the loadings, especially for the strong pattern scenario. Here, the largest percentage is 2.179% indicating that the level of information in the posterior sample corresponds to a small portion of the information in a completely random sample with the same size. In the MDN model fit for both data matrices, the chains autocorrelations for σ^2 are low (n_{eff} percentage near 100%). This is also true for the MNN model in the strong pattern scenario, but the percentage is smaller (72.4 to 83.3%) for the MNN model in the weak pattern case.

In Table 5, for a weak pattern matrix and assuming MDN, the median n_{eff} percentages for the loadings are 100%. In this case, most iterations are related to $\delta_0(\alpha_i)$ defining a chain constant at zero (few iterations are associated with the normal component). Given the higher probability for $\delta_0(\alpha_i)$ in this situation, we assume the number of iterations such that $\alpha_i = 0$ (i.e., $Z_i = 0$) as the effective sample size. Table 5 also shows that, assuming MNN, the effective sample sizes for α and λ (weak pattern model fit) tend to be larger than the corresponding results for a strong pattern model fit.

In summary, all versions of the model (MDN and MNN) indicate good inference results with small HPD amplitude and MSE. Again, the autocorrelation can be considered high in some chains, however, the user may choose a larger lag argument to reduce the association between consecutive observations. As expected, all implementation versions show similar results reinforcing the fact that, in this paper, the same model has been implemented in **Rcpp**, R and MATLAB.

4. Package usage

In this section, we show how to use the **slfm** package taking into account the same real data set for breast cancer (Miller *et al.* 2005) with 251 microarrays evaluated in the previous section. The latest version of **slfm** should always be available from the Comprehensive R Archive Network (CRAN) at http://CRAN.R-project.org/package=slfm. Run the following commands to install and load the package.

```
R> install.packages("slfm")
R> library("slfm")
```

The main functions are `process_matrix`, `slfm` and `slfm_list`. Both `slfm` and `slfm_list` fit the SLFM-U model, however, the first function evaluates a single preprocessed matrix and the second one evaluates a group of preprocessed matrices fitting the factor model to each one of them. The function `slfm` returns an object containing MCMC outputs that can be explored through **coda** for inference. The function `slfm_list` returns the P/M/A classification and a measure of the pattern strength.

Before fitting the SLFM-U, it is necessary to preprocess the available data matrices; see Section 2.2 for details. Recall that the preprocessing procedure takes into account the information from all probe sets and all available samples to background correct and normalize the observed light intensities, thus removing noise effects within and across the arrays. In other words, we cannot preprocess a single matrix representing a probe set, the function `process_matrix` requires a set of matrices to work properly. Each matrix should be in a tab-separated values (TSV) file inside a directory accessible to R. Those files can be distributed in different sub-directories, under the same main directory, in order to overcome the operating system limitation for the number of files under a single directory. The command to preprocess a set of matrices is given below:

```
R> process_matrix("original_folder", "processed_folder", sample_size)
```

The preprocess function will calculate the first principal component (p_{c1}) based on a random sample (size `sample_size`) of data matrices. The matrices to be selected are those available in the folder `"original_folder"`. A big matrix, used to compute the p_{c1}, is created allocating each selected matrix below the others. The random selection is a strategy to avoid problems related to the computer memory limitations, for example, Miller *et al.* (2005) investigate a data set with 251 microarrays representing 22,283 probe sets (matrices X), which would be problematic to load and manipulate with ordinary computers. The larger the value specified in `sample_size`, the more information is used to determine the p_{c1}. The preprocessed matrices are saved in the folder `"processed_folder"` with the same folder structure established in the original directory. After the preprocessing procedure, we can evaluate the patterns and get the SLFM-U P/M/A classifications using the next command:

```
R> slfm_list("processed_folder")
```

This is a short version of `slfm_list` assuming default values to configure the SLFM-U and the MCMC. A detailed description of all arguments is as follows:

- `path`: directory containing the set of preprocessed matrices for classification;

- `recursive`: Boolean indicating whether to look inside sub-directories (default = `TRUE`);
- `a`: shape parameter of the Inverse Gamma prior distribution (default = 2.1);
- `b`: scale parameter of the Inverse Gamma prior distribution (default = 1.1);
- `gamma_a`: 1st shape parameter of the Beta prior distribution (default = 1);
- `gamma_b`: 2nd shape parameter of the Beta prior distribution (default = 1);
- `omega_0`: variance of the "spike" component in the mixture prior (default = 0.01), unused if MDN;
- `omega_1`: variance of the "slab" component of the MDN or MNN prior (default = 10);
- `sample`: size of the posterior sample (default = 1,000);
- `burnin`: size of the burn-in period (default = nearest integer to `sample` $\times$ 0.25);
- `lag`: size of the lag to select observations for the posterior sample (default = 1). The total number of iterations is `burnin` + `lag` $\times$ `sample`;
- `degenerate`: Boolean indicating whether the MDN mixture is used (default = `FALSE`).

Note that the default prior specifications are the same vague priors considered in the study developed in Section 3.1. The output of the function is a '`data.frame`' containing the names of the X matrix files, their P/M/A classifications and the percentage of "Significant" loadings indicating the strength of the coherent pattern:

```
    File  Classification  Significant %
1  X_001          Absent         0.0000
2  X_002          Absent         0.0000
3  X_003          Absent         0.0000
4  X_004          Absent         0.0000
5  X_005          Absent         0.0000
6  X_006          Absent         0.0000
7  X_007        Marginal         0.1250
```

We have applied the function `slfm_list` to the data set in Miller *et al.* (2005) to generate detection calls for 22,283 probe sets. Consider the default prior specifications and MCMC configurations. Both models (MDN and MNN) were fitted to the real data and their percentage of significant loadings were investigated. Figure 4 shows an output based on artificial data resembling the behavior of the real case. It took 4.54 hours to generate all detection calls. As can be seen, Panel (b) indicates that these models tend to agree in terms of magnitude of percentages, i.e., an increasing pattern can be observed in both columns. Panel (a) shows that, for both models, the proportion of significant α_i's is larger than 50% for most probe sets (approximately 90.3% of the probe sets for MNN and MDN). Recall that the criterion to define the P/M/A calls in **slfm** is the most frequent S/I/N loading classification. This strategy provides 97.7% and 94.1% of "P" calls for MNN and MDN, respectively. The user is free to choose a more restrictive criterion, for example, one may assume a "P" call for a

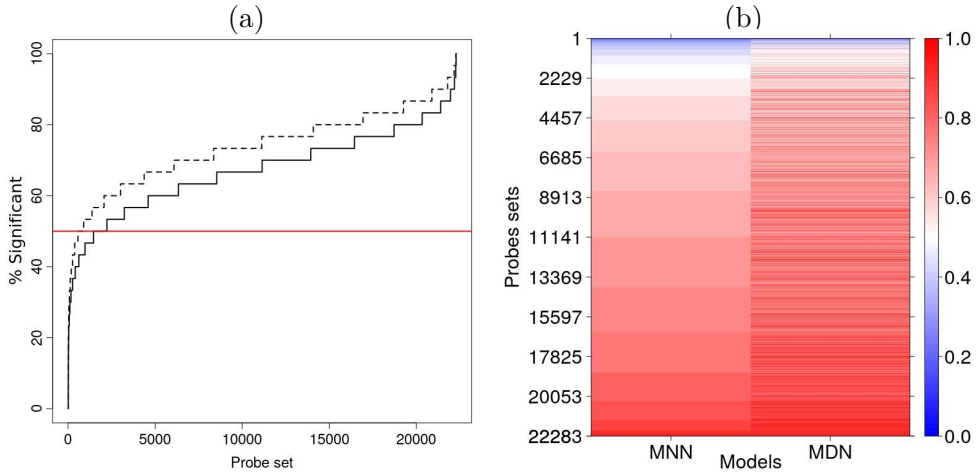

Figure 4: Percentages of significant loadings for each probe set in a simulation resembling Miller *et al.* (2005). Panel (a): percentages for the MNN model (continuous line), MDN model (dashed line) and the 50% level (horizontal line). Here, the values were sorted separately for each model. Panel (b): Image of percentages sorted with respect to the MNN model.

probe set only if the percentage of significant loadings is higher than say 90% (in this case, we have MNN = 15.4% and MDN = 37.1 % of "P" calls).

The package also provides a function `plot_matrix` to create an image graph for a visual assessment of the data pattern. The main argument of this function is the data matrix under study; see the command below. Other optional arguments, and their default values, are: `standardize.rows = TRUE` to standardize the rows of the target matrix, `reorder.rows = TRUE` and `reorder.cols = TRUE` to reorder the rows and columns of X with respect to their median values, and `high.contrast = TRUE` to improve the color contrast. The images shown in Figure 1 were obtained via `plot_matrix`.

```
R> plot_matrix(single_matrix)
```

The function `slfm` fits the SLFM-U for a single X matrix ($m \times n$) and returns a '`slfm`' object with the following elements: (i) `x` containing the data matrix X, (ii) `alpha`, `lambda` and `sigma2` displaying **coda** summary statistics for the chains of α_i, λ_j and σ_i^2, respectively ($i = 1, \ldots, m$ and $j = 1, \ldots, n$), (iii) `alpha_matrix`, `lambda_matrix`, `sigma2_matrix` and `q_star` containing the chains (after burn-in period) for α_i, λ_j, σ_i^2 and q_i^*, (iv) `z_matrix` indicating for each iteration whether α_i comes from the spike ($Z_i = 0$) or slab ($Z_i = 1$) component of the mixture and (v) `classification` showing the S/I/N statuses for each row of X. The same arguments and default values presented for `slfm_list` are also valid for `slfm`. As an example, consider the data matrix with strong pattern in Figure 1 and the MCMC setup related to the chains in Figure 2, we can apply the function `slfm` as follows:

```
R> X_matrix <- read.table(file.path("datasets", "X_matrix.txt"))
R> slfm_obj <- slfm(X_matrix)
R> slfm_obj$classification
```

```
 var1  var2  var3  var4  var5  var6  var7  var8  var9 var10
    S     S     S     S     S     S     S     S     S     S
```

```
var11 var12 var13 var14 var15 var16 var17 var18 var19 var20
    S     S     S     S     S     S     S     S     S     S
var21 var22 var23 var24 var25 var26 var27 var28 var29 var30
    S     S     S     S     S     S     S     S     S     S
var31 var32 var33 var34 var35 var36 var37 var38 var39 var40
    S     S     S     S     S     S     S     S     S     S
Levels: S
```

The output objects from **slfm** containing the MCMC chains are in fact 'mcmc' objects from **coda**, which means that we can use native functions from **coda** to explore these results. This includes the options: `autocorr`, `geweke.diag`, `geweke.plot`, `heidel.diag` and `heidel.plot` to name a few. For instance, the user can apply the Geweke diagnostic test, evaluated in Section 3.1, for the chains generated for each loading as follows:

```
R> library("coda")
R> slfm_obj <- slfm(X_matrix)
R> geweke.diag(slfm_obj$alpha_matrix)

Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

  var1   var2   var3   var4   var5   var6   var7   var8   var9  var10
0.7839 0.8263 1.0330 0.8588 0.7525 0.9893 0.9050 1.2625 0.7709 0.8064
 var11  var12  var13  var14  var15  var16  var17  var18  var19  var20
0.7764 0.7922 0.8532 0.8465 0.8191 0.8625 0.8190 0.7424 0.8602 0.6719
 var21  var22  var23  var24  var25  var26  var27  var28  var29  var30
0.7791 0.7875 0.6747 0.7557 0.7617 0.6811 0.7358 1.0044 0.8330 0.7921
 var31  var32  var33  var34  var35  var36  var37  var38  var39  var40
0.7359 0.7602 0.7948 1.1011 0.9467 0.8651 0.8195 1.0986 0.8104 0.9298
```

5. Conclusions

In this work, we have presented the R package **slfm** comprising tools for the analysis of Affymetrix GeneChip oligonucleotide microarrays. The main output are detection calls indicating whether the sequence represented by a probe set – denoted here as a gene – should be regarded as "Present", "Marginal" or "Absent". These categories indicate the status of the corresponding gene activity, with "Marginal" defining lack of evidence to determine either presence or absence of activity in the explored samples. The detection method, considered in **slfm**, takes advantage of the information available in terms of coherent patterns of probe expressions across microarrays. The probes are expected to exhibit similar (coherent) levels of expressions within the same array, if their target sequence is found in a sample.

This data structure can be well evaluated through a Bayesian factor model assuming a single factor λ and a mixture prior to test the significance of the loadings α. If α_i is significant for most probes, the estimated λ will affect most rows of the expression matrix resulting in a coherent pattern display. Looking at the credible interval for the posterior probability

of the large variance Gaussian component (for each i) we can compute the proportion of significant loadings, which measures the strength of coherent pattern in the data. According to the literature, Bayesian factor analysis has superior performance compared to other simple (fast and thus attractive) detection methods. Our study confirms the good inference results provided by the factor model through an extensive analysis involving simulated data.

A drawback, related to the proposed Bayesian model, is the use of Gibbs sampling to indirectly sample from the joint posterior distribution of the parameters, which is known only up to a normalizing constant. The main issue, in this case, is the computational cost associated with the algorithm, making the Bayesian method unattractive for a user willing to trade inference performance for computational speed. Package **slfm** circumvents this difficulty taking into account the faster performance of C++ integrated to R through the **Rcpp** package. The present paper shows that the **Rcpp** version of the factor model, using **RcppArmadillo**, is approximately 42 and 28 times faster than the usual R and MATLAB implementations, respectively. This result, together with the superior inference performance of the factor model, makes the detection method based on coherent patterns in **slfm** an interesting, effective and viable alternative.

The main reason to use **Rcpp** is its close relationship with R allowing us to easily take advantage of this software resources for a full statistical analysis. Besides working in the R environment and using the Bayesian factor model for detection calls through a fast MCMC, **slfm** also includes a nonparametric preprocessing routine for raw expression data, a graph to visualize the data and summary statistics as outputs to interpret the main inference results.

Acknowledgments

The authors would like to thank the associate editor and two anonymous referees for their review and comments to improve this work. The second author also thanks Fundação de Amparo à Pesquisa de Minas Gerais (FAPEMIG) for supporting this research.

References

Affymetrix (2001). "Statistical Algorithms Reference Guide." *Technical report*, Affymetrix Technical Report.

Carvalho C, Chang J, Lucas JE, Nevins JR, Wang Q, West M (2008). "High-Dimensional Sparse Factor Modelling: Applications in Gene Expression Genomics." *Journal of the American Statistical Association*, **103**(484), 1438–1456. doi:10.1198/016214508000000869.

Doornik JA, Ooms M (2006). *Introduction to Ox: An Object-Oriented Matrix Language*. Timberlake Consultants Press, London.

Duarte JDN, Mayrink VD (2015). "Factor Analysis with Mixture Modeling to Evaluate Coherent Patterns in Microarray Data." In A Polpo, F Louzada, L Rifo, J Stern, M Lauretto (eds.), *Interdisciplinary Bayesian Statistics*, volume 118 of *Springer Proceedings in Mathematics & Statistics*, pp. 185–195. Springer-Verlag. doi:10.1007/978-3-319-12454-4_15.

Duarte JDN, Mayrink VD (2019). ***slfm**: Tools for Fitting Sparse Latent Factor Model*. R package version 1.0.0, URL https://CRAN.R-project.org/package=slfm.

Eddelbuettel D (2013). *Seamless R and C++ Integration with* ***Rcpp***. Springer-Verlag.

Eddelbuettel D, François R (2011). "**Rcpp**: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18. doi:10.18637/jss.v040.i08.

Eddelbuettel D, Sanderson C (2014). "**RcppArmadillo**: Accelerating R with High-Performance C++ Linear Algebra." *Computational Statistics & Data Analysis*, **71**, 1054–1063. doi:10.1016/j.csda.2013.02.005.

Francois R, Eddelbuettel D (2018). ***RcppGSL**: **Rcpp** Integration for GNU **GSL** Vectors and Matrices*. R package version 0.3.6, URL https://CRAN.R-project.org/package=RcppGSL.

Gamerman D, Lopes HF (2006). *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference*, volume 68. 2nd edition. Chapman & Hall/CRC, London. doi:10.1201/9781482296426.

Gelfand AE, Smith AFM (1990). "Sampling-Based Approaches to Calculating Marginal Densities." *Journal of the American Statistical Association*, **85**(410), 398–410. doi:10.2307/2289776.

Gelman A, Rubin DB (1992). "Inference from Iterative Simulation Using Multiple Sequences." *Statistical Science*, **7**(4), 457–472.

Gentleman R, Carey V, Bates D, Bolstad B, Dettling M, Dudoit S, Ellis B, Gautier L, Ge Y, Gentry J, Hornik K, Hothorn T, Huber W, Iacus S, Irizarry R, Leisch F, Li C, Maechler M, Rossini A, Sawitzki G, Smith C, Smyth G, Tierney L, Yang J, Zhang J (2004). "Bioconductor: Open Software Development for Computational Biology and Bioinformatics." *Genome Biology*, **5**, R80. doi:10.1186/gb-2004-5-10-r80.

George EI, McCulloch E (1993). "Variable Selection via Gibbs Sampling." *Journal of the American Statistical Association*, **88**(423), 881–889. doi:10.1080/01621459.1993.10476353.

George EI, McCulloch E (1997). "Approaches for Bayesian Variable Selection." *Statistica Sinica*, **7**, 339–373.

Geweke J (1992). "Evaluating the Accuracy of Sampling-Based Approaches to the Calculation of Posterior Moments." In JM Bernardo, J Berger, AP Dawid, AFM Smith (eds.), *Bayesian Statistics*, volume 4, pp. 169–193. Oxford University Press.

Geweke J (1996). "Variable Selection and Model Comparison in Regression." In *Bayesian Statistics*, volume 5, pp. 609–620. Oxford University Press, New York.

Irizarry RA, Hobbs B, Collin F, Beazer-Barclay YD, Antonellis KJ, Scherf U, Speed TP (2003). "Exploration, Normalization, and Summaries of High Density Oligonucleotide Array Probe Level Data." *Biostatistics*, **4**(2), 249–264.

Lucas JE, Carvalho C, Wang Q, Bild A, Nevins JR, West M (2006). "Sparse Statistical Modelling in Gene Expression Genomics." In P Muller, K Do, M Vannucci (eds.), *Bayesian Inference for Gene Expression and Proteomics*, pp. 155–176. Cambridge University Press.

Mayrink VD, Gamerman D (2009). "On Computational Aspects of Bayesian Spatial Models: Influence of the Neighboring Structure in the Efficiency of MCMC Algorithms." *Computational Statistics*, **24**(4), 641–669. doi:10.1007/s00180-009-0153-0.

Mayrink VD, Lucas JE (2015). "Bayesian Factor Models for the Detection of Coherent Patterns in Gene Expression Data." *Brazilian Journal of Probability and Statistics*, **29**(1), 1–33. doi:10.1214/13-bjps226.

Miller LD, Smeds J, George J, Vega VB, Vergara L, Ploner A, Pawitan Y, Hall P, Klaar S, Liu ET, Bergh J (2005). "An Expression Signature for P53 Status in Human Breast Cancer Predicts Mutation Status, Transcriptional Effects, and Patient Survival." *Proceedings of the National Academy of Sciences of the United States of America*, **102**(38), 13550–13555. doi:10.1073/pnas.0506230102.

Plummer M, Best N, Cowles K, Vines K (2006). "**coda**: Convergence Diagnosis and Output Analysis for MCMC." *R News*, **6**(1), 7–11. URL https://CRAN.R-project.org/doc/Rnews/.

R Core Team (2019). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Rue H, Martino S, Chopin N (2009). "Approximate Bayesian Inference for Latent Gaussian Models Using Integrated Nested Laplace Approximations." *Journal of the Royal Statistical Society B*, **71**(2), 319–392. doi:10.1111/j.1467-9868.2008.00700.x.

The MathWorks Inc (2018). *MATLAB – The Language of Technical Computing, Version R2018b*. Natick. URL http://www.mathworks.com/products/matlab/.

Warren P, Taylor D, Martini PGV, Jackson J, Bienkowska J (2007). "PANP – A New Method of Gene Detection on Oligonucleotide Expression Arrays." In *Proceedings of the 7th IEEE International Conference on Bioinformatics and Bioengineering*, pp. 108–115. doi:10.1109/bibe.2007.4375552.

West M (2003). "Bayesian Factor Regression Models in the Large p, Small n Paradigm." In J Bernardo, M Bayarri, J Berger, A Dawid, D Heckerman, A Smith, M West (eds.), *Bayesian Statistics*, volume 7, pp. 723–732. Oxford University Press.

Affiliation:

João Daniel N. Duarte, Vinícius D. Mayrink
Universidade Federal de Minas Gerais

Departamento de Estatística
Instituto de Ciências Exatas
Av. Antônio Carlos, 6627
Belo Horizonte, MG, Brazil, 31270-901
E-mail: jdanielnd@gmail.com, vdm@est.ufmg.br

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
July 2019, Volume 90, Issue 9 *Submitted:* 2015-09-25
doi:10.18637/jss.v090.i09 *Accepted:* 2019-05-24