Ecole Nationale des Statistiques et de l'Administration Economique

Report

Decoupling Shrinkage and Selection in Gaussian Linear Factor Analysis

CAMADINI Louisa & FARAH Beatriz
ENSAE 3ème année
Année scolaire 2022-2023

Contents

1 Introduction

The DSSFA[1] Method consists of separating the estimation of the factor loadings matrix (which consists of the description of the relationship between the variables and the factors) from the estimation of the number of factors.

The DSS method has the goal of identifying sparse structures in high-dimensional data, which is useful in the very common context of data analysis where the number of features in a problem is much larger than the number of data observations. This method consists of two main steps, which are the following:

- **Shrinkage:** This process consists of imposing regularization to the factor loadings matrix in order to decrease the variance of the estimated factors.
- **Selection:** In this step, the previously estimated covariance matrix is used to perform feature selection, in order to find the features that are more strongly associated with the response variable.

The DSSFA method delivers more accurate and stabilized estimations of the factor loadings matrix, being useful in numerous scenarios, such as:

- **Dimensionality reduction:** As the method performs the shrinkage and selection steps, it allows for the identification of a smaller set of factors responsible for the variation in a larger set of observed variables.
- **Feature Selection:** By performing the selection step, this method can be used to identify the most important features in a given scenario.
- **Exploratory analysis:** By identifying a set of factors that explain well the variation and structure of a dataset, this method can be used to understand the existing structure or patterns in the data.

Factor model [2] The matrix called factor loadings will be denoted by B and of size $p \times k$. A random sample of n observations $y = (y_1, ..., y_n)^T$ contains the p-dimensional vector $Y_i = (y_{i1}, ..., y_{ip})^T$ for $i = 1, ..., n$, linked to a vector of common latent factors f_i of dimension $k \leq p$, such that :

$$y_i = Bf_i + \epsilon_i \tag{1}$$

where ϵ_i is the p-dimensional error vector.

The assumptions we make about the distribution laws are the following :

- ϵ_i is distributed as $\mathcal{N}_p(0, \Sigma)$ and $\Sigma = \text{diag}(\sigma_1^2, ..., \sigma_p^2)$ is called the "unique variance" ;
- f_i is distributed as $\mathcal{N}_k(0, I_k)$ where I denotes the identity matrix ;
- f_j and ϵ_i are independent for all $i \neq j$;
- This implies that y_i is also distributed as a $\mathcal{N}_p(0, \Omega)$, where $\Omega = BB^T + \Sigma$ is the covariance matrix.

Challenges We want to highlight the multivariate dependency structure between the variables. For this purpose, factor analysis method is widely used. However, the DSSFA Method we present here is intended to assist in determining the number of factors – *ie.* determine the value of k – and obtain sparse representation of the loadings. In other words, we want B to be a sparse matrix, a matrix in which most of the elements are zero. Conceptually, sparsity corresponds to systems with few pairwise interactions.

As explained, this information will be summarized in the multivariate posterior, and obtained by sparse point estimates generated from samples of the posterior distribution. Moreover, this information is supposed to highlight a trade-off between performance and sparsity of the model.

Method To do so, we will start by generating a sample from the a posteriori of the factor loadings B and the uniqueness Σ, without restriction on the choice of the prior (as long as the posterior samples are valid) – which is a robustness property of this method.

Then, we will minimize an expected predictive loss function and obtain a series of sparse point estimates, and a decreasing number of factor dimensions k. This depends only on the posterior covariance matrix.

In this way, we will obtain the a posteriori summary that we are looking for, which we will highlight with a graph.

[1] Decoupling Shrinkage and Selection in Gaussian Linear Factor Analysis

[2] DSS Method in case of linear models, introduced by Hahn, R. P. and C. M. Carvalho (2015).

2 Methodology

The main goal of the framework proposed by the reference paper is to obtain an informative summary contained in the multivariate posterior to highlight the final estimates, in order to explicit the trade-off between sparcity and the predictive performance of the model. The proposed method consists of three main steps:

- **1) Model fitting**: Firstly, the posteriors of factor loadings and uniqueness are sampled. This model imposes no restrictions on prior choices, provided that posterior samples are available. In this step, it is used a conservative value for the factor dimension in order to aggregate information to the posterior.
- **2) Minimization of the predicted loss function:** In the second step, the expected loss function, that depends only on the posterior covariance matrix, is minimized using a penalized version of the expectation-maximization (EM) algorithm.
- **3) Posterior summary plots:** Lastly, we perform a graphical summary in order to highlight the final estimates.

One of the advantages of the proposed method is its flexibility and robustness, given that it imposes no restrictions on prior choices for the factor model. In the following subsections, we will describe with detail the different steps of the model.

2.1 Loss Function

In the study, a loss function is defined by evaluating the negative log-likelihood of the multivariate normal distribution and penalizing the model complexity. This loss is defined in Equation 2, where $\tilde{\mathbf{\Omega}}$ is a $p \times p$ positive semi-definite symmetric matrix, $\mathbf{\Omega}$ is a $p \times p$ covariance matrix, $P(.)$ is the complexity penalty and $\text{tr}(A)$ is the trace of matrix A.

$$\mathcal{L}_\lambda(\mathbf{\Omega}, \tilde{\mathbf{\Omega}}) = \log|\tilde{\mathbf{\Omega}}| + \text{tr}\left(\tilde{\mathbf{\Omega}}^{-1}\mathbf{\Omega}\right) + \lambda P(\tilde{\mathbf{\Omega}}) \tag{2}$$

We may then proceed with the decision analysis and obtain optimal point estimates. We have that the optimal point estimate is given by:

$$\begin{aligned} \hat{\mathbf{\Omega}}_\lambda &\equiv \underset{\tilde{\Omega}}{\text{argmin}} E_{\theta|y}\left[\mathcal{L}_\lambda(\mathbf{\Omega}, \tilde{\mathbf{\Omega}})\right] \\ &= \log|\tilde{\mathbf{\Omega}}| + \text{tr}\left(\tilde{\mathbf{\Omega}}^{-1}\overline{\mathbf{\Omega}}\right) + \lambda P(\tilde{\mathbf{\Omega}}) \end{aligned} \tag{3}$$

As our interest lies in optimal decisions for the factor analysis model, in particular in the structure of the matrix B, in our decision framework we assume $\tilde{\mathbf{\Omega}} = \tilde{B}\tilde{B}^T + \tilde{\Sigma}$, where $\tilde{B}$ is a $p \times \tilde{k}$ matrix, $\tilde{k} \in \{1, 2, ...\}$ are elements of the decision analysis. $\tilde{k}$ is a choice of dimension summary on the resulting loading estimates and $\tilde{\Sigma}$ is a $p \times p$ positive and diagonal matrix, where $\tilde{B}, \tilde{\Sigma}$ and $\tilde{k}$ are actions in the decision analysis.

The point-estimates of the factor loadings and uniqueness are obtained by solving, subject to $\tilde{\Sigma} = \tilde{B} + \tilde{B}^T + \tilde{\Sigma}$ and different values of $\tilde{k}$:

$$\left(\hat{\boldsymbol{B}}_{\tilde{k},\lambda}, \hat{\mathbf{\Sigma}}_{\tilde{k},\lambda}\right) \equiv \underset{\tilde{\boldsymbol{B}}, \tilde{\mathbf{\Sigma}}}{\text{argmin}} \left\{\log|\tilde{\mathbf{\Omega}}| + \text{tr}\left(\tilde{\mathbf{\Omega}}^{-1}\overline{\mathbf{\Omega}}\right) + \lambda\|\tilde{\boldsymbol{B}}\|_1\right\} \tag{4}$$

After this step, we will have a sequence $\left(\hat{\boldsymbol{B}}_{\tilde{k},\lambda}, \hat{\mathbf{\Sigma}}_{\tilde{k},\lambda}\right)$, where $\tilde{k} = 1, ..., k_{max}$; $\lambda = \lambda_0, \lambda_1, ...\lambda_l$, where $\lambda_0 = 0$ and λ_l is determined by the optimization method used to solve 4.

2.2 Posterior summary

The last step of the framework developed in the paper is a proposed posterior summary that is useful to understand the trade-off between factor dimension, sparse loadings and loss in the model fit. The main idea in this step is to summarize how changing $\tilde{k}$ and λ affects model fit in different dimensions and complexities. This is done by generating a sequence of loss functions $\mathcal{L}_{\tilde{k},\lambda}\left(\mathbf{\Omega}, \hat{\mathbf{\Omega}}_{\tilde{k},\lambda}\right)$, where $\hat{\mathbf{\Omega}}_{\tilde{k},\lambda} = \hat{\boldsymbol{B}}_{\tilde{k},\lambda}\hat{\boldsymbol{B}}^T_{\tilde{k},\lambda} + \hat{\mathbf{\Sigma}}_{\tilde{k},\lambda}$ is obtained via the covariance matrices generated by the point estimates from the optimization procedure (seen in Equation 4), and $\tilde{k} = 1, ..., k_{max}$; $\lambda = \lambda_0, \lambda_1, ...\lambda_l$.

From the generated sequence, it is then possible to select a quantile for the loss function of the full model and construct a posterior summary plot, visualising $E_{\boldsymbol{\theta}|\boldsymbol{y}}\left[\mathcal{L}_{\tilde{k},\lambda}\left(\boldsymbol{\Omega}, \hat{\boldsymbol{\Omega}}_{\tilde{k},\lambda}\right)\right]$ in relation to $\tilde{k}, \lambda$.

3 Simulation

3.1 Real Personality data

In this section we will illustrate the DSSFA Method by applying it to a real dataset. Following Bolfarine et al., 2022 and Legramanti et al., 2020, we performed our simulations on a subset of the Big Five Inventory (BFI). The data measures each individual on the big five factors of personality, which are [3]:

- Extroversion, agreeableness, conscientiousness, neuroticism, and openness.

Each item was answered as a grade from 1 to 6, where 1 means *Very Inaccurate* and 6, *Very Accurate*. Available on the R package *psych* [4], this dataset is largely used in the context of factor anaysis. Following Bolfarine et al., 2022, we used a subset of $p = 25$ personality variables from $n = 126$ individuals with ages above 50.

3.2 Code

Our code was developed in R and inspired by a couple of functions for some of the steps of the DSSFA method and visualisation written by the authors of the reference paper available on GitHub [5]. We produced our own functions to perform the method BFI dataset, which are available on the corresponding codes files attached to this report.

The functions from the authors' GitHub repository were partly written in C++, which makes the use of R libraries such as *Rcpp* and *RcppArmadillo* necessary in order to run the C++ extracts in R. Unfortunately, we encountered compatibility problems when calling these libraries in order to run the authors' methods. Therefore we wrote new functions from scratch, using only the R programming language. Our code consists of 4 main functions:

1. **Gibbs sampler**

We implemented an adaptive Gibbs sampler for Gaussian factorial models, following Legramanti et al., 2020 and Bolfarine et al., 2022.
We ran the Gibbs sampler using the bfa package to generate the posterior samples with 10000 iterations, in which 5000 were discarded as burn-in. The Gibbs sampler function takes several inputs:

- Y : the matrix of observed data,
- *seed* : the seed for the random number generator,
- *N_sample* : the number of iterations for the Gibbs sampler,
- *etc.*

The function stores the results of the Gibbs sampling process: the number of factors, the factor loading matrix, the factor score matrix, the column-specific loading accuracies, the break weights, the augmented data, and the error variances. It then runs the Gibbs sampler for *N_sample* iterations, updating these variables at each iteration. Finally, it returns the final estimates of the variables.

Then, we create an object (a list), named *obj*, which contained all the informations obtained :

- Number of variables in the sample : 25,
- Sample size: 2000,
- Posterior sample covariance matrix,
- Some prior informations,
- Posterior sample size : 2000.

[3]More information on: http://www.uoregon.edu/~sanjay/bigfive.html#where
[4]R documentation available on: https://www.rdocumentation.org/packages/psych/versions/2.2.9/topics/bfi
[5]Codes available on: https://github.com/hbolfarine/dssfa/tree/master/R_codes

2. **DSSFA function**

Here we apply the DSSFA method to our object. The maximum number of factors, k_{max}, is a parameter to be specified by the user. It represents the number of underlying dimensions that the model should take into account. It should be chosen according to the specific research question or data structure. Overall, $k_{max} \leq p$, where p is the dimension of our model. But if we have good reason to believe that the data have a certain number of underlying dimensions, we may set k_{max} to this value.

With this data set $p = 25$. However, we have chosen to be more ambitious in terms of size reduction, and have thus set $k_{max} = 15$. The function takes several inputs:

- obj : the previous object that contains information about the data and prior information for the factor analysis,
- $length.lambda = 2$: the number of values of the lambda shrinkage parameter to consider,
- $k.max = 15$: the maximum number of factors to consider,
- $tol.em = 0.00001$: tolerance for the EM algorithm.

The function makes a list to store the results and initializes several variables such as the prior information, the maximum number of factors, the sample size, the number of variables and the lambda length. It then runs the DSSFA algorithm for each value of $\tilde{k}$ (number of factors) from 1 to $k.max$, using the *fanc()* function to perform the factor analysis. This package computes the solution path of the penalized maximum likelihood estimates. Finally, the function stores the results and returns a list.

3. **Summary plots**

As already said in the Section 2.2, the generated sequence now allows to select a quantile for the loss function of the full model and construct a posterior summary plot, visualising $E_{\boldsymbol{\theta}|\boldsymbol{y}}\left[\mathcal{L}_{\tilde{k},\lambda}\left(\boldsymbol{\Omega}, \hat{\boldsymbol{\Omega}}_{\tilde{k},\lambda}\right)\right]$ in relation to $\tilde{k}, \lambda$.
This is exactly what is done in this function.

Indeed, it creates a summary graph for the Bayesian factor analysis model. The function uses a 95% confidence interval. It then uses the fit values of the BFA model to select the best number of factors and the best lambda value. Then it uses the selected loadings and uniquenesses to calculate the Omega matrices. The function generates a graph of the BFA model summary using the created object, see here for the results 1.

4. **Model selection**

This function selects the "best" model from a list of models based on the 95% confidence interval (CI). It checks if a k-value (number of factors) has been provided, otherwise it uses all k-values between 1 and the k_{max} provided in the input list.

Then, it calculates the quantiles of the credible interval using the fit values of the models. The function then loops over the selected k-values and checks whether the fit value of each model is within the CI. If it is, the function stores the corresponding model parameters, otherwise it stores a null value.

This function also records any additional information such as uniquenesses and correlation matrices if they have been provided in the input list. The function returns 3 which is, in our case, the selected model, *ie.* the selected number of factors.

3.3 Results

Following the steps described in 2 and Bolfarine et al., 2022, we applied the DSSFA method to the data described in 3.1. In this case, we have $p = 25$, and the number of factors was initialized as $k_{max} = 15$, as explained previously. We used the DSSFA method with $\tilde{k} = 1, 2, ..., k_{max}$ factors, and let it select the factor dimension under the 95%-quantile of the loss function, under no regularization.

The posterior summary plots can be found in 1. This image illustrates, for each number of factors $\tilde{k}$ (here plotted from 1 to 15 factors), the density of the loss functions $\left(\mathcal{L}_{\tilde{k},\lambda}\left(\boldsymbol{\Omega}, \widehat{\boldsymbol{\Omega}_{\tilde{k},\lambda}}\right)\right)$, which correspond to the violin plots, and their respective posterior means $\left(E_{\theta|y}\left[\mathcal{L}_{\tilde{k},\lambda}\left(\boldsymbol{\Omega}, \hat{\Omega}_{\tilde{k},\lambda}\right)\right]\right)$, which correspond to the dots in the image. In the Figure 1, we also display the 95%-quantile of the loss function of the model generated with $k_{max} = 15$, which is plotted as the dashed lines.

We have that the selected model is the first one whose expected value lies within the 95%-CI of the loss the full model. In this context, as we can see in 1, the corresponding selected model is a factor model of size 3. In Bolfarine et al., 2022 and in Legramanti et al., 2020 the authors also identified three main factors in a similar setting, where they used the same method and data, but set $k_{max} = 24$. It is important to note that our result delivering the size of the factor model as being equal to 3 doesn't necessarily imply that this is the "real" number of underlying factors in the data. Indeed, this solution reflects the optimal result found by the DSS method given its criteria, therefore it is not the unique solution for this problem and it could be compared with results found by other methods that hold different criteria and hypothesis for the data.

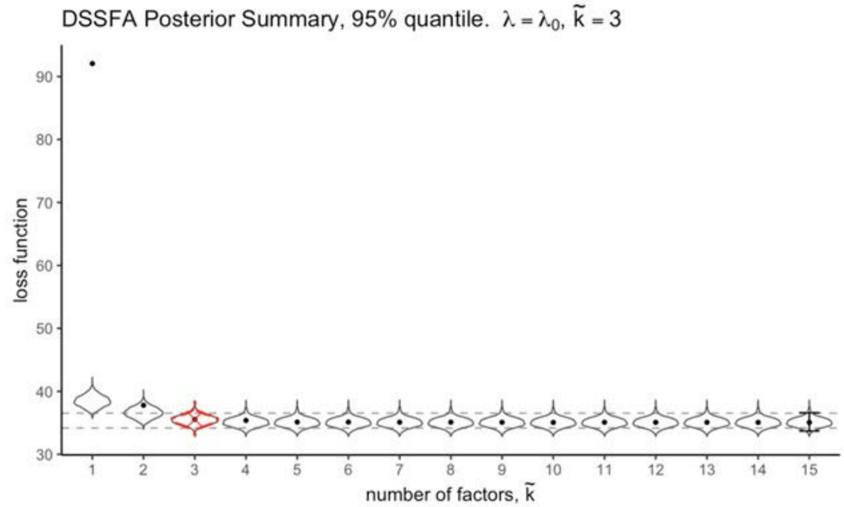


Figure 1: Simulation result for the DSSFA Posterior Summary

In Figure 2, we can see the resulting estimates of the loadings matrix without penalty $\hat{B}_{3,\lambda}$, selected with the DSSFA method. The percentage of zeroed loadings in the regularized model is approximately 20%.

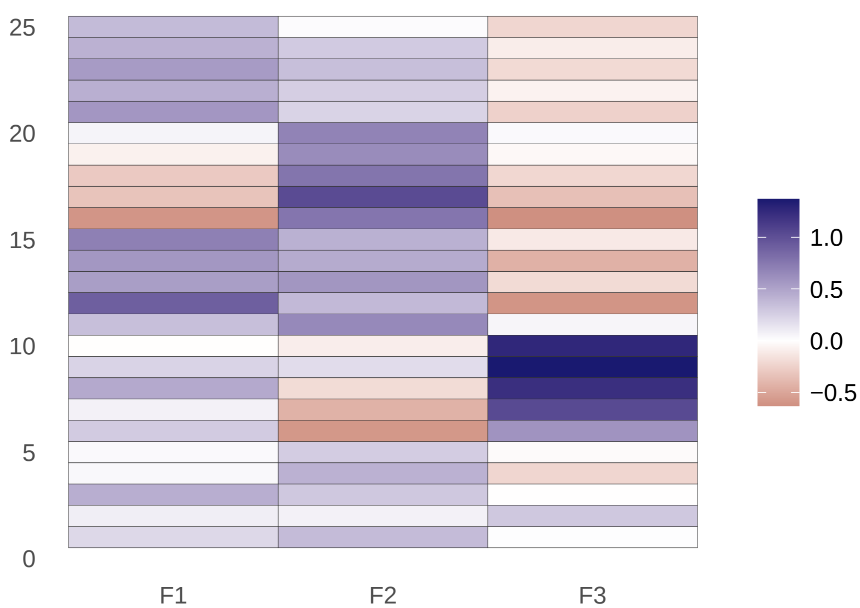


Figure 2: Loadings sparse matrix (size $p \times k = 25 \times 3$, the number of suggested factors)

4 Conclusion

In this project, we followed the DSSFA method described in Bolfarine et al., 2022, where its main goal is to provide estimates for the factor loadings obtained from the decision analysis, and also propose a summary of the information gathered by the posteriori.

We implemented the described method and applied it in the context of real personality traits data.
Following the DSSFA steps, we applied the method to the set of data considering $\tilde{k} = 1, 2, ...k_{max}$ factors, where we set $k_{max} = 15$. Using no regularization, under a 95%-quantile of the loss function of the full model, our algorithm selected a factor model of size 3, which is the same result obtained in Bolfarine et al., 2022 for $k_{max} = 24$.

This procedure provided useful posterior summary and interpretable factor loadings, and has the advantage of selecting simultaneously the model and the factor loadings. It could be further used in scenarios such as dimensionality reduction and feature selection problems, as well as a tool of exploratory analysis of a dataset.

5 References

Bolfarine, H., Carvalho, C. M., Lopes, H. F., & Murray, J. S. (2022). Decoupling shrinkage and selection in gaussian linear factor analysis. *Bayesian Analysis*, *1*(1), 1–23.

Legramanti, S., Durante, D., & Dunson, D. B. (2020). Bayesian cumulative shrinkage for infinite factorizations. *Biometrika*, *107*(3), 745–752.

Simoni, A. (2022). Course notes and slides.

DSSFA Project

Louisa Camadini & Beatriz Farah

12/01/2023

Upload data from 'psych'

This dataset contains 25 self-report personality items from the International Personality Item Pool (ipip.ori.org). It was developed as part of the Personality Assessment Project. The data contains the responses of 2800 people and can be used for demonstration purposes, as well as for the factor analysis that we will use here. We try to reduce the dimension to at least 25-1 = 24 factors, but we can even be more ambitious and propose directly k_max = 15 for the speed of implementation of the following algorithms. More information here.

```r
data(bfi)
bfi = na.omit(bfi)
bfi = bfi[which(bfi$age>50),1:25]
```

```
##       A1 A2 A3 A4 A5 C1 C2 C3 C4 C5 E1 E2 E3 E4 E5 N1 N2 N3 N4 N5 O1 O2 O3 O4
## 61661 -1  5  6  5  6  4  3  2 -4 -5 -2 -1  2  5  2  2  2  2  2  2  6 -1  5  5
## 61669 -2  4  4  4  3  6  5  6 -1 -1 -2 -4  4  2  6  3  3  5  3  2  5 -2  6  6
## 61782 -1  2  2  4  2  2  4  2 -5 -1 -2 -2  4  2  2  4  2  2  2  4  2 -3  4  2
## 61813 -1  5  5  5  5  5  2  4 -1 -2 -4 -1  5  5  5  1  1  1  4  1  6 -1  5  5
## 62013 -3  5  4  4  4  3  3  4 -4 -4 -5 -4  3  2  4  2  3  2  3  2  5 -4  4  3
## 62197 -1  5  5  5  5  5  4  5 -1 -1 -2 -2  3  5  5  1  2  2  2  2  2 -2  5  4
## 62260 -1  5  5  5  4  4  4  4 -2 -2 -4 -4  4  2  5  1  2  1  1  2  4 -2  5  6
## 62267 -1  6  5  6  5  5  6  6 -2 -1 -5 -2  5  1  6  5  4  5  4  2  6 -5  3  6
## 62333 -1  6  6  6  6  6  6  6 -1 -1 -5 -1  6  6  6  1  6  1  1  1  6 -1  3  6
## 62345 -2  4  4  6  5  4  4  4 -3 -4 -1 -1  5  5  6  2  4  3  3  1  6 -1  6  4
## 62368 -1  6  1  6  5  4  1  2 -5 -6 -1 -1  5  5  6  2  4  6  6  5  6 -1  6  6
## 62423 -2  6  5  6  3  5  6  5 -2 -3 -3 -2  5  5  5  3  5  5  1  4  4 -2  4  6
## 62448 -1  6  2  6  5  5  5  5 -4 -2 -2 -6  1  4  5  4  2  1  2  3  6 -5  3  6
## 62498 -1  5  6  6  6  1  5  6 -4 -1 -6 -6  3  2  5  4  1  2  6  6  6 -5  2  6
## 62518 -2  6  6  6  6  6  6  5 -1 -1 -1 -1  6  5  6  1  1  2  2  1  6 -1  6  5
## 62577 -2  5  5  5  5  2  1  2 -5 -4 -5 -5  5  4  2  5  5  5  6  4  5 -6  4  6
## 62597 -2  5  5  5  4  6  2  4 -2 -5 -1 -2  5  5  5  4  5  5  5  4  6 -2  5  6
## 62599 -1  5  5  6  6  6  1  5 -3 -1 -6 -6  1  5  1  4  5  5  6  1  6 -5  1  6
## 62617 -2  6  6  6  6  6  6  6 -1 -1 -3 -6  6  1  6  4  4  1  1  3  6 -1  6  2
## 62622 -1  6  6  6  6  6  6  5 -1 -4 -1 -1  6  6  6  4  6  5  2  4  6 -5  6  6
## 62638 -2  6  6  2  6  6  3  5 -1 -2 -1 -1  5  5  6  1  2  2  1  1  5 -1  6  4
## 62664 -1  5  6  5  6  6  5  5 -1 -2 -2 -1  5  1  5  4  5  4  5  4  5 -4  4  6
## 62688 -1  6  5  6  4  6  6  5 -1 -2 -1 -5  2  5  6  5  6  4  4  3  4 -5  5  5
## 62712 -1  6  5  5  6  6  5  5 -1 -3 -1 -1  5  6  6  1  2  1  1  1  6 -1  6  6
## 62719 -1  4  6  5  6  5  2  6 -1 -1 -1 -3  5  1  5  6  6  6  6  6  4 -1  5  6
## 62750 -2  4  4  6  5  5  2  5 -1 -2 -2 -2  3  5  4  2  5  2  2  1  4 -3  4  4
## 62874 -1  6  6  6  6  5  6  6 -1 -1 -1 -1  6  6  6  1  1  1  1  2  5 -2  3  4
```

```
## 66965 -2
## 67026 -5
## 67029 -5
## 67055 -5
## 67098 -1
## 67232 -2
## 67342 -4
## 67388 -3
## 67406 -2
## 67453 -1
```

These data are preprocessed by us. Here, we hide a part of the code if it is not directly related to the subject, for a better readability.

Note : The algorithms called with "source()" are presented in the appendix, to ease the reading.

Gibbs Sampler

```
source("GIBBS.R")
gibbs = gibbs_sampler(y,my_seed=6784,N_sampl=15000,
                           alpha=5,a_sig=1,b_sig=0.3,a_theta=2,b_theta=2,theta_inf=0.05,
                           start_adapt=500,Hmax=dim(y)[2]+1,alpha0=-1,alpha1=-5*10^(-4))
```

```
burn_in = 5000
```

Burn-in for the sampler

```
p = dim(gibbs$y)[2]
```

Number of variables : 25

```
thin = 5
thinning = seq(burn_in+1,gibbs$N_sampl,by=thin)
```

Thinning

```
Omega_post = array(NA,c(p,p,length(thinning)))
```

Posterior sample of the covaraince matrix

Object to build

```
obj = list()
```

```
obj$P = p
```

Number of variables in the sample : 25

```
obj$samp.size = length(thinning)
```

Sample size : 2000

```
obj$post.cov = Omega_post
```

Posterior sample covariance matrix

```
obj$nsim = 2000
obj$thin = 1
```

Posterior sample size : 2000

DSSFA Method applied to our object (k_max = 15) :

```
source("DSSFA.R")
obj.bfa.list.CUSP = DSSFA(obj, length.lambda = 2, k.max = 15, print.out = T, tol.em = 0.00001)
```

```
## Factor =  1
## Factor =  2
## Factor =  3
## Factor =  4
## Factor =  5
## Factor =  6
## Factor =  7
## Factor =  8
## Factor =  9
## Factor =  10
## Factor =  11
## Factor =  12
## Factor =  13
## Factor =  14
## Factor =  15
```

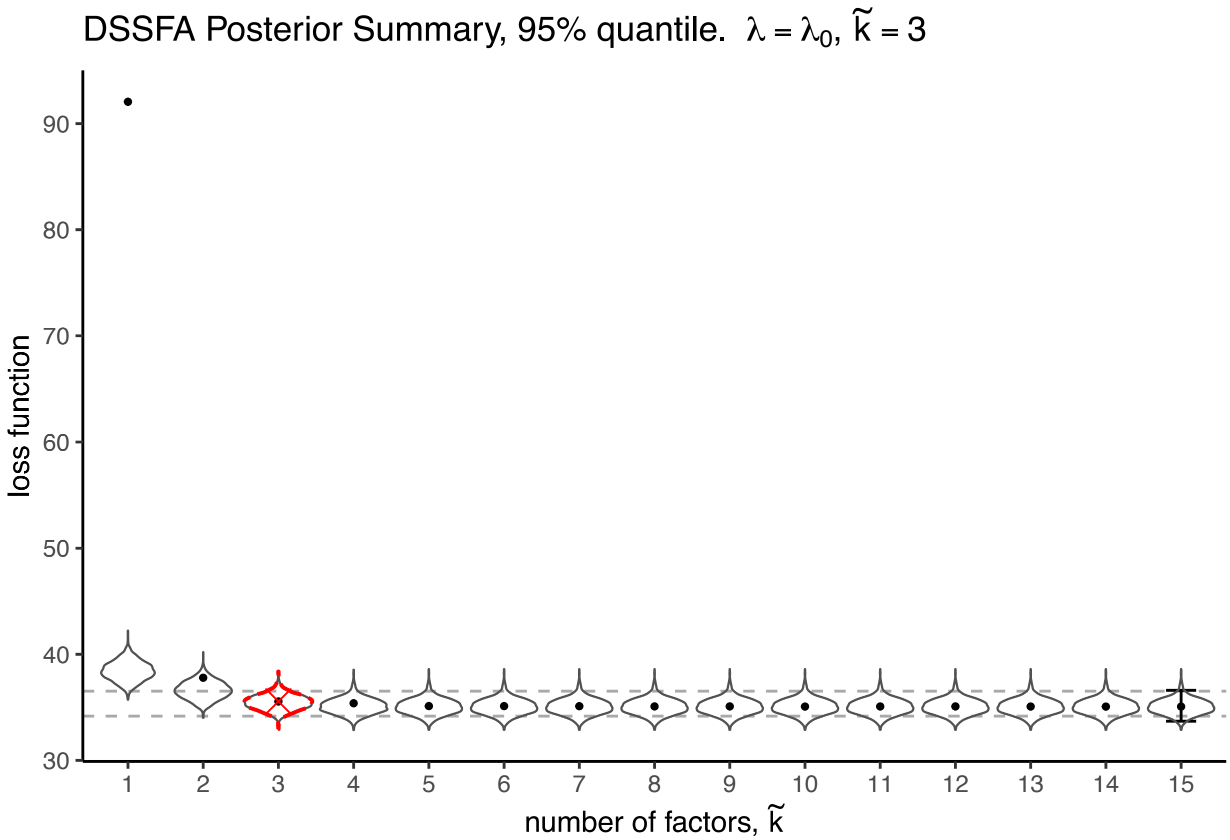
Creation of the 95% confidence interval :

```
CI = list()
CI$CI.85 = list(c("85"),c(0.075,0.925))
CI$CI.90 = list(c("90"),c(0.05,0.95))
CI$CI.95 = list(c("95"),c(0.025,0.975))
CI$CI.99 = list(c("99"),c(0.005,0.995))
```

Plots that summarize the DSSFA method :

```
source("summary_plot.R")
summary.plot(obj.bfa.list.CUSP,CI[[3]],cov.dssfa = T, text_main_prior = "CUSP prior,", col.lambda = 1,l
```

```
## Warning: Using ‘size‘ aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use ‘linewidth‘ instead.
```



“Summary.plot” creates a summary graph for the Bayesian factor analysis model, as above. This function uses a default CI of 95%. It uses the fit values of the model to select the best number of factors and the best lambda value of the loss. Then the function uses the selected loadings and uniquenesses to calculate the Omega matrices. In this case, we can see the selected number of factors : k=3.

Select the right number of factors :

```
source("DSSFA_sel_mat.R")
mat.sel = sel.model.dssfa(obj.bfa.list.CUSP,CI[[3]])
sel = min(which(lapply(mat.sel$mat.sel.ksel,is.null) == F))
cat('The right number of factors is : k =', sel)
```

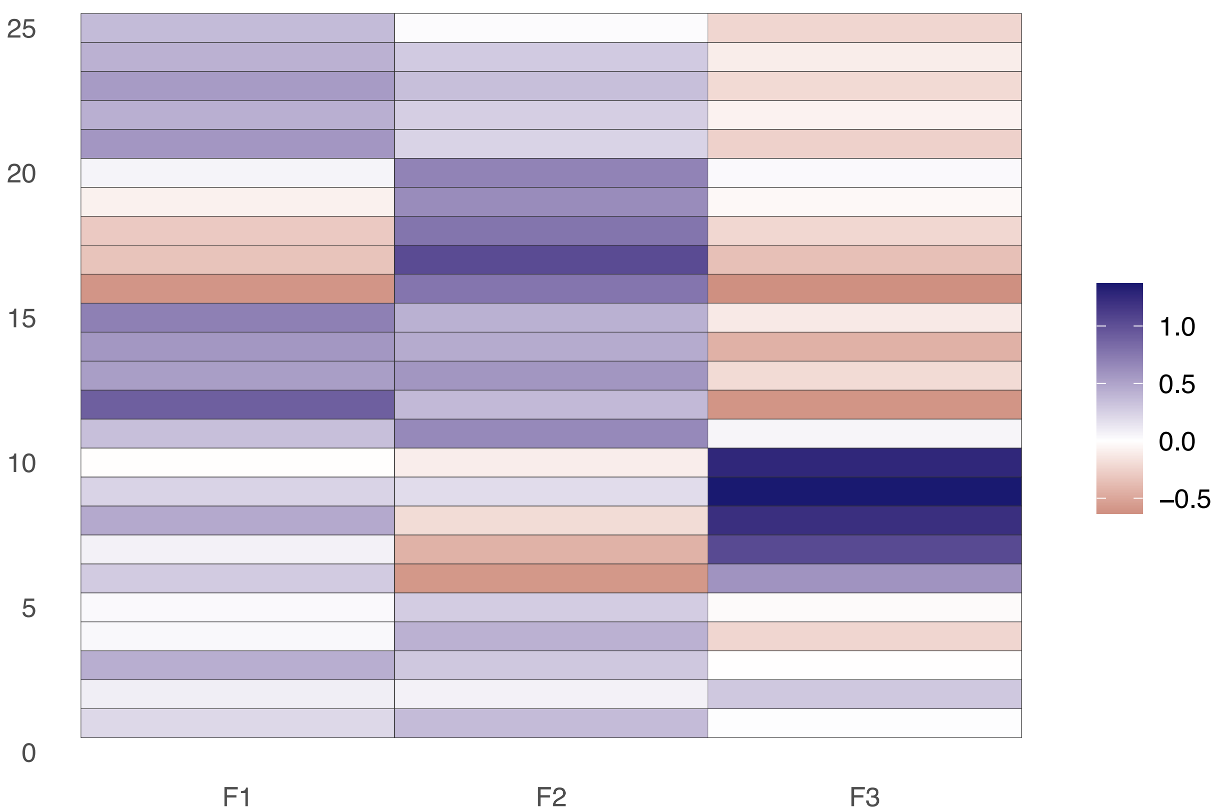
```
## The right number of factors is : k = 3
```

"Sel.model.dssfa" selects the "best" model from a list of models based on the confidence interval. The function checks if a k-value has been provided (number of factors), otherwise it uses all k-values between 1 and the maximum k-value provided in the input list, here k_max=15. Then, it calculates the quantiles of the credible interval using the fit values of the models and loops over the selected k-values and checks whether the fit value of each model is within the credible interval. If this is the case, the function saves the parameters of the corresponding model, otherwise it saves a null value. It also records uniquenesses and correlation matrices if they have been provided in the input list. Here, the function returns "3" : the selected model (ie. the selected number of factors).

Loadings Matrix :

We finally obtain the loadings matrix from the DSSFA method, for k=3 number of factors (F1, F2, and F3). A clear and interpretable pattern is observed in the loadings from the DSSFA method, despite the reduction of the factor dimension.

```
mat = mat.sel$mat.sel.ksel[[sel]]
mat = mat[,c(3,1,2)]
make.plot.mat(mat,fact.name = c("F1","F2","F3"))
```



Appendix

You will find the functions useful for the realization of our code.

Gibbs sampler

```
gibbs_sampler <- function(y,my_seed,N_sampl,alpha,a_sig,b_sig,a_theta,b_theta,
                              theta_inf,start_adapt,Hmax,alpha0,alpha1){
  set.seed(my_seed)
  n<-dim(y)[1]
  p<-dim(y)[2]
  u<-runif(N_sampl)

  H<-rep(NA,N_sampl)
  Hstar<-rep(NA,N_sampl)
  Lambda_post<-vector("list",N_sampl)
  eta_post<-vector("list",N_sampl)
  theta_inv_post<-vector("list",N_sampl)
  w_post<-vector("list",N_sampl)
  z_post<-vector("list",N_sampl)
  inv_sigma_sq_post<-matrix(NA,p,N_sampl)

  H[1]<-p+1
  Hstar[1]<-p
  Lambda_post[[1]]<-matrix(rnorm(p*H[1]),p,H[1])
  eta_post[[1]]<-matrix(rnorm(n*H[1]),n,H[1])
  theta_inv_post[[1]]<-rep(1,H[1])
  w_post[[1]]<-rep(1/H[1],H[1])
  z_post[[1]]<-rep(H[1],H[1])
  inv_sigma_sq_post[,1]<-1

  t0 <- proc.time()
  for (t in 2:N_sampl){
    Lambda_post[[t]]<-matrix(NA,p,H[t-1])
    eta_post[[t]]<-matrix(NA,n,H[t-1])
    theta_inv_post[[t]]<-rep(NA,H[t-1])
    w_post[[t]]<-rep(NA,H[t-1])
    z_post[[t]]<-rep(NA,H[t-1])

    for (j in 1:p){
      V_j<-chol2inv(chol(diag(theta_inv_post[[t-1]],nrow=H[t-1])+inv_sigma_sq_post[j,t-1]*(t(eta_post[[t
      mu_j<-V_j%*%t(eta_post[[t-1]])%*%(y[,j])*inv_sigma_sq_post[j,t-1]
      Lambda_post[[t]][j,]<-mvrnorm(1,mu_j,V_j)
    }

    diff_y<-apply((y-(eta_post[[t-1]]%*%t(Lambda_post[[t]])))^2,2,sum)
    for (j in 1:p){
      inv_sigma_sq_post[j,t]<-rgamma(n=1,shape=a_sig+0.5*n,rate=b_sig+0.5*diff_y[j])
    }

    V_eta<-chol2inv(chol(diag(H[t-1])+t(Lambda_post[[t]])%*%diag(inv_sigma_sq_post[,t])%*%Lambda_post[[t
    for (i in 1:n){
```

```
    mu_eta_i<-V_eta%*%t(Lambda_post[[t]])%*%diag(inv_sigma_sq_post[,t])%*%y[i,]
    eta_post[[t]][i,]<-mvrnorm(1,mu_eta_i,V_eta)
  }

  lhd_spike<-rep(0,H[t-1])
  lhd_slab<-rep(0,H[t-1])
  for (h in 1:H[t-1]){
    lhd_spike[h]<-exp(sum(log(dnorm(Lambda_post[[t]][,h], mean = 0, sd = theta_inf^(1/2), log = FALSE
    lhd_slab[h]<-dmvt(x=Lambda_post[[t]][,h], mu=rep(0,p), S=(b_theta/a_theta)*diag(p), df=2*a_theta)
    prob_h<-w_post[[t-1]]*c(rep(lhd_spike[h],h),rep(lhd_slab[h],H[t-1]-h))
    if (sum(prob_h)==0){
      prob_h<-c(rep(0,H[t-1]-1),1)
    }
    else{
      prob_h<-prob_h/sum(prob_h)
    }
    z_post[[t]][h]<-c(1:H[t-1])%*%rmultinom(n=1, size=1, prob=prob_h)
  }

  v<-rep(NA,H[t-1])
  for (h in 1:(H[t-1]-1)){
    v[h]<-rbeta(1, shape1 = 1+sum(z_post[[t]]==h), shape2 = alpha+sum(z_post[[t]]>h))
  }
  v[H[t-1]]<-1
  w_post[[t]][1]<-v[1]
  for (h in 2:H[t-1]){
    w_post[[t]][h]<-v[h]*prod(1-v[1:(h-1)])
  }

  for (h in 1:H[t-1]){
    if (z_post[[t]][h]<=h){
      theta_inv_post[[t]][h]<-theta_inf^(-1)
    }
    else{
      theta_inv_post[[t]][h]<-rgamma(n=1,shape=a_theta+0.5*p,rate=b_theta+0.5*t(Lambda_post[[t]][,h])
    }
  }

  active <- which(z_post[[t]]>c(1:H[t-1]))
  Hstar[t] <- length(active)
  H[t]<-H[t-1]

  if (t>=start_adapt & u[t]<=exp(alpha0+alpha1*t)){
    if (Hstar[t]<H[t-1]-1){
      H[t]<-Hstar[t]+1
      eta_post[[t]]<-cbind(eta_post[[t]][,active],rnorm(n))
      theta_inv_post[[t]]<-c(theta_inv_post[[t]][active],theta_inf^(-1))
      w_post[[t]]<-c(w_post[[t]][active],1-sum(w_post[[t]][active]))
      Lambda_post[[t]]<-cbind(Lambda_post[[t]][,active],rnorm(p,mean=0,sd=sqrt(theta_inf)))
    } else if (H[t-1]<Hmax) {
      H[t]<-H[t-1]+1
      eta_post[[t]]<-cbind(eta_post[[t]],rnorm(n))
      v[H[t-1]]<-rbeta(1,shape1=1,shape2=alpha)
```

```
        v<-c(v,1)
        w_post[[t]]<-rep(NA,H[t])
        w_post[[t]][1]<-v[1]
        for (h in 2:H[t]){
          w_post[[t]][h]<-v[h]*prod(1-v[1:(h-1)])
        }
        theta_inv_post[[t]]<-c(theta_inv_post[[t]],theta_inf^(-1))
        Lambda_post[[t]]<-cbind(Lambda_post[[t]],rnorm(p,mean=0,sd=sqrt(theta_inf)))
      }
    }
  }
  runtime <- proc.time()-t0
  output<-list("y"=y,"my_seed"=my_seed,"N_sampl"=N_sampl,
               "alpha"=alpha,"a_sig"=a_sig,"b_sig"=b_sig,"a_theta"=a_theta,"b_theta"=b_theta,
               "theta_inf"=theta_inf,"start_adapt"=start_adapt,"Hmax"=Hmax,
               "alpha0"=alpha0,"alpha1"=alpha1,
               "H"=H,"Hstar"=Hstar,"Lambda_post"=Lambda_post,"eta_post"=eta_post,
               "theta_inv_post"=theta_inv_post,"w_post"=w_post,"z_post"=z_post,
               "inv_sigma_sq_post"=inv_sigma_sq_post,"runtime"=runtime)
  return(output)
}
```

DSSFA function

Here are some intermediate functions that we had to set up to use our objects.

```
require(fanc)
fitcpp.post.fanc.cov <- function(Mat1, Mat2) {
  OmegaSamp <- array(Mat1, dim = c(dim(Mat1)[-3], dim(Mat1)[3]))
  OmegaBar <- as.matrix(Mat2)

  M <- dim(OmegaSamp)[2]
  fit <- numeric(M)

  for (i in 1:M) {
    fit[i] <- log(det(OmegaBar)) + sum(solve(OmegaBar) * OmegaSamp[,,i])
  }
  return(fit)
}

fitcpp.fanc <- function(Mat1, Mat2, Mat3) {
  c1 <- array(Mat1, dim = c(dim(Mat1)[-3], dim(Mat1)[3]))
  m2 <- as.matrix(Mat2)
  OmegaBar <- as.matrix(Mat3)

  lambda <- dim(c1)[3]
  fit <- numeric(lambda)
  for (i in 1:lambda) {
    if(all(dim(c1[,,i]) == dim(m2[i,]))){
      fit[i] <- log(det(c1[,,i] + diag(m2[i,]))) + sum(OmegaBar %*% solve(c1[,,i] + diag(m2[i,])))
    } else {
      stop("Input matrices have different dimensions.")
    }
```

```
  }
  return(fit)
}

fit.fanc.post.cov = function(list.bfa){

  list.bfa$fit.post.fanc.cov = fitcpp.post.fanc.cov(list.bfa$post.cov,list.bfa$Omega.kmax)
  return(list.bfa)
}

fit.fanc = function(list.bfa){

  list.bfa$fit.fanc = fitcpp.fanc(list.bfa$betas.prod.fanc,list.bfa$fanc.sigma,list.bfa$Omega.bar)
  return(list.bfa)
}
```

And the main function (from here, partly modified)

```
DSSFA = function(obj,length.lambda = 1, k.max = 1, print.out = F, plot.summary = F, cor.factor = F, tol
  if(cor.factor){
    cat("Oblique - Sparse")
    cat("\n")
  }
  list.bfa = list()
  list.bfa$prior.bfa = obj$prior.bfa
  list.bfa$k.max = k.max
  list.bfa$M = obj$samp.size
  list.bfa$p = obj$P
  list.bfa$length.lambda = length.lambda
  list.bfa$cor.factor = cor.factor
  list.bfa$post.cov = obj$post.cov
  list.bfa$Omega.bar = apply(obj$post.cov,c(1,2),mean)

  fit.fanc.mat  = matrix(0,length.lambda,list.bfa$k.max)
  num.zero.mat  = matrix(0,length.lambda,list.bfa$k.max)
  lambda.mat    = matrix(0,length.lambda,list.bfa$k.max)
  fit.sel       = vector("list",list.bfa$k.max)
  num.zero.list = vector("list",list.bfa$k.max)
  list.sel      = vector("list",list.bfa$k.max)
  list.full     = vector("list",list.bfa$k.max)
  uniq.full     = list()

  if(list.bfa$cor.factor == T){
    phi.full      = list()
    phi.list = vector("list",list.bfa$k.max)
  }
  for(k.temp in 1:list.bfa$k.max){
    if(length.lambda >= 2){
      fanc.bfa = fanc(covmat = list.bfa$Omega.bar, factors = k.temp,
                      gamma = Inf, control = list(start = "warm", min.rhozero = T, cor.factor = cor.fac
    }else{
      fanc.bfa = fanc(covmat = list.bfa$Omega.bar, factors = k.temp, rho = 0,
                      gamma = Inf, control = list(start = "warm", min.rhozero = T, cor.factor = cor.fac
```

```r
    }
    list.bfa$lambda = fanc.bfa$rho
    lambda.mat[,k.temp] = fanc.bfa$rho
    list.bfa$loadings.fanc.bfa = as.array(fanc.bfa$loadings$gamma1)
    list.bfa$fanc.sigma.bfa = fanc.bfa$uniquenesses[,,1]

    if(length.lambda < 2){
      list.bfa$fanc.sigma.bfa = matrix(list.bfa$fanc.sigma.bfa,nrow = 1)
    }
    fanc.matrix.bfa = matrix(0,length(list.bfa$lambda),list.bfa$p*k.temp)
    for(i in 1:length(list.bfa$lambda)){
      fanc.matrix.bfa[i,] = as.numeric(list.bfa$loadings.fanc.bfa[[i]])
    }
    list.bfa$fanc.matrix.bfa = fanc.matrix.bfa
    betas.bfa.fanc.full = replicate(length(list.bfa$lambda),matrix(0,list.bfa$p,k.temp))
    for(i in 1:length(list.bfa$lambda)){
      betas.bfa.fanc.full[,,i] = as.matrix(list.bfa$loadings.fanc.bfa[[i]])
    }
    list.bfa$fanc.matrix.sel = betas.bfa.fanc.full
    mat_dif_zero = which(apply(betas.bfa.fanc.full,3,function(x) sum(colSums(x) == 0)) != 0)

    if(length(mat_dif_zero) == length.lambda){
      cat("Zeroed Columns")
      fanc.bfa  = fanc(covmat = list.bfa$Omega.bar, factors = k.temp, gamma = Inf, control = list(start
      list.bfa$lambda = fanc.bfa$rho
      lambda.mat[,k.temp] = fanc.bfa$rho
      list.bfa$loadings.fanc.bfa = as.array(fanc.bfa$loadings$gamma1)
      list.bfa$fanc.sigma.bfa = t(as.matrix(fanc.bfa$uniquenesses[,,1]))
      if(length.lambda < 2){
        list.bfa$fanc.sigma.bfa = matrix(list.bfa$fanc.sigma.bfa,nrow = 1)
      }
      fanc.matrix.bfa = matrix(0,length(list.bfa$lambda),list.bfa$p*k.temp)
      for(i in 1:length(list.bfa$lambda)){
        fanc.matrix.bfa[i,] = as.numeric(list.bfa$loadings.fanc.bfa[[i]])
      }
      list.bfa$fanc.matrix.bfa = fanc.matrix.bfa
      betas.bfa.fanc.full = replicate(length(list.bfa$lambda),matrix(0,list.bfa$p,k.temp))
      for(i in 1:length(list.bfa$lambda)){
        betas.bfa.fanc.full[,,i] = as.matrix(list.bfa$loadings.fanc.bfa[[i]])
      }
      list.bfa$fanc.matrix.sel = betas.bfa.fanc.full
      if(apply(betas.bfa.fanc.full,3,function(x) sum(colSums(x) == 0)) == 0) cat(" - FULL Matrix") else
      mat_dif_zero = length.lambda-1
    }
    num.zero.mat[,k.temp] = apply(fanc.matrix.bfa, 1, function(x) sum(x != 0))
    list.sel[[k.temp]]  = betas.bfa.fanc.full
    list.full[[k.temp]] = betas.bfa.fanc.full

    if(list.bfa$cor.factor){
      phi.full[[k.temp]] = fanc.bfa$Phi[,,,1]
    }
    betas.prod.bfa.fanc = replicate(length(list.bfa$lambda),matrix(0,list.bfa$p,list.bfa$p))
    if(cor.factor == T & k.temp > 1){
```

```
      for(i in 1:length(list.bfa$lambda)){
        betas.prod.bfa.fanc[,,i] = betas.bfa.fanc.full[,,i]%*%fanc.bfa$Phi[,,i,1]%*%t(betas.bfa.fanc.fu
      }
      list.bfa$betas.prod.fanc = betas.prod.bfa.fanc
    }else{
      for(i in 1:length(list.bfa$lambda)){
        betas.prod.bfa.fanc[,,i] = betas.bfa.fanc.full[,,i]%*%t(betas.bfa.fanc.full[,,i])
      }
      list.bfa$betas.prod.fanc = betas.prod.bfa.fanc
    }
    if(k.temp == list.bfa$k.max){
      Omega.kmax = list.bfa$betas.prod.fanc[,,length(list.bfa$lambda)]+diag(fanc.bfa$uniquenesses[length
      list.bfa$Omega.kmax = Omega.kmax
      list.bfa = fit.fanc.post.cov(list.bfa)
    }
    list.bfa = fit.fanc(list.bfa)
    fit.fanc.mat[,k.temp] = list.bfa$fit.fanc
    if(length(mat_dif_zero) == 0){
      fit.sel[[k.temp]] = list.bfa$fit.fanc
      num.zero.list[[k.temp]] = apply(fanc.matrix.bfa, 1, function(x) sum(x == 0))
      uniq.full[[k.temp]] = fanc.bfa$uniquenesses[,,1]
      if(length.lambda < 2){
        uniq.full[[k.temp]] = matrix(fanc.bfa$uniquenesses[,,1],nrow = 1)
      }
      if(list.bfa$cor.factor == T){
        phi.list[[k.temp]] = fanc.bfa$Phi[,,,1]
      }
    }else{
      fit.sel[[k.temp]] = list.bfa$fit.fanc[-mat_dif_zero]
      num.zero.list[[k.temp]] = apply(fanc.matrix.bfa, 1, function(x) sum(x == 0))[-mat_dif_zero]
      if(k.temp == 1){
        dim.array = dim(list.sel[[k.temp]][,,-mat_dif_zero])[2]
        list.sel[[k.temp]] = array(list.sel[[k.temp]][,,-mat_dif_zero],dim = c(list.bfa$p,k.temp,dim.ar
      }else{
        list.sel[[k.temp]] = list.sel[[k.temp]][,,-mat_dif_zero]
      }
      uniq.full[[k.temp]] = fanc.bfa$uniquenesses[-mat_dif_zero,,1]
      if(length.lambda < 2){
        uniq.full[[k.temp]] = matrix(fanc.bfa$uniquenesses[,,1],nrow = 1)
      }
      if(list.bfa$cor.factor == T){
        phi.list[[k.temp]] = fanc.bfa$Phi[,,-mat_dif_zero,1]
      }
    }
    if(print.out == T){
      cat("Factor = ",k.temp)
      cat("\n")
    }
  }

  if(list.bfa$cor.factor){
    list.bfa$phi.list = phi.list
    list.bfa$phi.full = phi.full
```

```
  }
  list.bfa$uniq.full = uniq.full
  list.bfa$fit.sel = fit.sel
  list.bfa$num.zero.list = num.zero.list
  list.bfa$list.sel = list.sel
  list.bfa$list.full = list.full
  list.bfa$rho.mat = lambda.mat

  return(list.bfa)
}
```

Summary plot function

Intermediate function :

```
fitcpp.post.plot.cov <- function(Mat1, Mat2) {
  Omegacov <- array(Mat1, dim = c(dim(Mat1)[-3], dim(Mat1)[3]))
  Omegakmax <- array(Mat2, dim = c(dim(Mat2)[-3], dim(Mat2)[3]))

  M <- dim(Omegacov)[3]
  kmax <- dim(Omegakmax)[3]
  Omega_lamb <- matrix(nrow = M, ncol = kmax)

  for (i in 1:kmax) {
    val <- log(det(Omegakmax[,,i]))
    for (j in 1:M) {
      Omega_lamb[j,i] <- val+sum(solve(Omegakmax[,,i]) * Omegacov[,,j])
    }
  }
  return(Omega_lamb)
}
```

Main function (from here, partly modified) :

```
summary.plot = function(list.bfa,CI = NULL, cov.dssfa = F, reg = T, text_main_prior = "", col.lambda = 

  if(is.null(CI)){
    ic.1 = 0.025
    ic.2 = 0.975
    CI.text = "95"
  }else{
    ic.1 = CI[[2]][1]
    ic.2 = CI[[2]][2]
    CI.text = CI[[1]]
  }
  q1 = quantile(list.bfa$fit.post.fanc,probs = c(ic.1,ic.2))[1]
  q2 = quantile(list.bfa$fit.post.fanc,probs = c(ic.1,ic.2))[2]

  k.sel = min(which(lapply(list.bfa$fit.sel, function(x) length(which(x < q2))) != 0))
  lamb.sel = lapply(list.bfa$fit.sel, function(x) list.bfa$length.lambda-length(which(x < q2))+1)
  lamb.sel = lamb.sel[[k.sel]]
```

```r
fit.list.plot = list()
for(i in 1:list.bfa$k.max){
  if(length(dim(list.bfa$list.sel[[i]])) == 3){
    dim.size = dim(list.bfa$list.sel[[i]])[3]
    Omega.temp = array(0,dim = c(list.bfa$p,list.bfa$p,dim.size))
    for(j in 1:dim.size){
      beta.max = as.matrix(list.bfa$list.sel[[i]][,,j])
      uniq.sel = list.bfa$uniq.full[[i]][j,]
      Omega.temp[,,j] = beta.max%*%t(beta.max)+diag(uniq.sel)
    }
  }else if(length(dim(list.bfa$list.sel[[i]])) != 3){
    dim.size = 1
    Omega.temp = array(0,dim = c(list.bfa$p,list.bfa$p,1))
    beta.max = as.matrix(list.bfa$list.sel[[i]])
    uniq.sel = list.bfa$uniq.full[[i]]
    Omega.temp[,,1] = beta.max%*%t(beta.max)+diag(uniq.sel)
  }
  if(cov.dssfa == T){
    fit.list.plot[[i]] = as.matrix(fitcpp.post.plot.cov(list.bfa$post.cov,Omega.temp))
  }
  colnames(fit.list.plot[[i]]) = ((list.bfa$length.lambda-dim.size)+1):list.bfa$length.lambda
}

temp.melt = lapply(fit.list.plot, melt)

for(i in 1:list.bfa$k.max){
  temp.melt[[i]]$kmax = as.numeric(i)
}

temp = bind_rows(temp.melt)
temp = data.frame(temp)

max.lamb = max(temp$Var2)
temp$Var2 = factor(temp$Var2,levels = 1:max.lamb,ordered = T)

temp.melt2 = lapply(list.bfa$fit.sel,melt)
for(i in 1:list.bfa$k.max){
  temp.melt2[[i]]$kmax = as.numeric(i)
  dim.size = dim(temp.melt2[[i]])[1]
  temp.melt2[[i]]$lamb = ((list.bfa$length.lambda-dim.size)+1):list.bfa$length.lambda
}

temp2 = bind_rows(temp.melt2)

mygreys = colorRampPalette(brewer.pal(9,"Greys"))(max.lamb + 7)
mygreys = mygreys[-((max.lamb+6):(max.lamb+7))]
mygreys = mygreys[-(1:5)]

min = min(temp2$lamb)

g = ggplot() + geom_point(data = temp2, aes(factor(kmax),value, col = factor(lamb, labels = (max.lamb
  scale_color_manual(values = mygreys, aesthetics = c("col")) + theme(legend.position = legend.pos) +
  guides(colour = guide_legend(ncol = col.lambda))
```

```
  temp.data.reg = temp[which(temp$kmax == k.sel & temp$Var2 == lamb.sel),]
  temp.point.reg = temp2[which(temp2$kmax == k.sel & temp2$lamb == lamb.sel),]

  g = g + geom_hline(yintercept = q2, lty = "dashed", col = "darkgrey") + geom_hline(yintercept = q1, lt

  for(i in 1:max.lamb){

    temp.data = temp[which(temp$Var2 == i),]
    temp.point = temp2[which(temp2$lamb ==  i),]

    if(i == max.lamb){
      df.summary <- temp.data %>% group_by(kmax) %>%
        summarize(mid = mean(value),
                  lo = quantile(value, ic.1),
                  hi = quantile(value, ic.2))

      g = g + geom_errorbar(data = df.summary[list.bfa$k.max,],aes(factor(max(kmax)),ymin = lo, ymax = h
    }

    g = g + geom_violin(data  = temp.data,aes(factor(kmax), value), alpha = 0, size = 0.4, col = mygreys

    if(i == max.lamb){
      g = g + geom_point(data = temp.point, aes(kmax,value,col = factor(lamb)), col = "#000000", size =
    }else{
      g = g + geom_point(data = temp.point, aes(kmax,value,col = factor(lamb)), col = mygreys[i], size =
    }

    if(reg == T & i == lamb.sel){
      g = g + geom_violin(data = temp.data.reg, aes(factor(kmax),value),col = "red", lty = "twodash", al
    }
  }
  g = g + geom_point(data = temp.point.reg, aes(kmax,value), col = "red", size = 4, shape = 4)

  if(reg == T){
    lamb.sel = max.lamb-lamb.sel
    text.lambda = paste("$\\lambda = \\lambda_{",lamb.sel,"}$", sep = "")
    text.main = paste("DSSFA Posterior Summary, ", CI.text, "% quantile.  ",text.lambda,", $\\tilde{k} =
  }else{
    text.main = paste("DSSFA Posterior Summary, ", CI.text, "% quantile.  ",sep = "")
  }
  g = g  + labs(title = TeX(text.main), x = TeX("number of factors, $\\tilde{k}$"), y = TeX("loss functi

  g
}
```

Function that selects the right model

This one is from this repo.

```
sel.model.dssfa = function(list.bfa,CI = NULL, k.sel = NULL, fast = F){

  if(length(CI) == 0){
```

```r
  ic.1 = 0.025
  ic.2 = 0.975
}else{
  ic.1 = CI[[2]][1]
  ic.2 = CI[[2]][2]
}
if(is.null(k.sel)){
  k.sel = 1:list.bfa$k.max
}
q1 = quantile(list.bfa$fit.post.fanc,probs = c(ic.1,ic.2))[1]
q2 = quantile(list.bfa$fit.post.fanc,probs = c(ic.1,ic.2))[2]

sel.model     = list()
mat.sel.ksel  = vector("list",length(k.sel))
uniq.sel.ksel = vector("list",length(k.sel))
phi.sel.ksel  = vector("list",length(k.sel))
mat.sel.CI    = vector("list",list.bfa$k.max)
uniq.sel.CI   = vector("list",list.bfa$k.max)
phi.sel.CI    = vector("list",list.bfa$k.max)

l = 1 # count the number of select factors
if(fast){
  for(i in k.sel){
    if(sum(which(list.bfa$fit.sel[[i]] < q2)) == 0){
      # cat("aqui11")
      mat.sel.ksel[[l]]  = NULL
      uniq.sel.ksel[[l]] = NULL
      if(list.bfa$cor.factor){
        phi.sel.ksel[[l]] = NULL
      }
      l = l+1
    }else{
      # cat("aqui12")
      min.sel = min(which(list.bfa$fit.sel[[i]] < q2))
      mat.sel.ksel[[l]]  = list.bfa$betas.full[[i]]
      uniq.sel.ksel[[l]] = list.bfa$list.uniqueness[[i]]
      if(list.bfa$cor.factor){
        phi.sel.ksel[[l]] = list.bfa$phi.full[[i]]
      }
      l = l+1
    }
  }
}else{
  for(i in k.sel){
    if(sum(which(list.bfa$fit.sel[[i]] < q2)) == 0){
      # cat("aqui11")
      mat.sel.ksel[[l]]  = NULL
      uniq.sel.ksel[[l]] = NULL
      if(list.bfa$cor.factor){
        phi.sel.ksel[[l]] = NULL
      }
      l = l+1
    }else{
```

```
        if(length(dim(list.bfa$list.sel[[i]])) != 3){
          # cat("aqui12")
          min.sel = min(which(list.bfa$fit.sel[[i]] < q2))
          mat.sel.ksel[[l]]  = list.bfa$list.sel[[i]]
          uniq.sel.ksel[[l]] = list.bfa$uniq.full[[i]]
          if(list.bfa$cor.factor){
            phi.sel.ksel[[l]] = list.bfa$list.bfa$phi.list[[i]]
          }
          l = l+1
        }else{
          # cat("aqui13")
          min.sel = min(which(list.bfa$fit.sel[[i]] < q2))
          mat.sel.ksel[[l]]  = list.bfa$list.sel[[i]][,,min.sel]
          if(list.bfa$length.lambda < 2){
            uniq.sel.ksel[[l]] = list.bfa$uniq.full[[i]]
          }else{
            uniq.sel.ksel[[l]] = list.bfa$uniq.full[[i]][min.sel,]
          }
          if(list.bfa$cor.factor){
            if(i == 1){
              phi.sel.ksel[[l]] = 1
            }else{
              phi.sel.ksel[[l]] = list.bfa$phi.list[[i]][,,min.sel]
            }
          }
          l = l+1
        }
      }
    }
  }
  sel.model$mat.sel.ksel  = mat.sel.ksel
  sel.model$uniq.sel.ksel = uniq.sel.ksel
  if(list.bfa$cor.factor){
    sel.model$phi.sel.ksel = phi.sel.ksel
  }
  return(sel.model)
}
make.plot.mat = function(mat,col.names,fact.name){
  rownames(mat) = col.names
  colnames(mat) = fact.name
  mat = apply(mat, 2, rev)
  longmat = melt(mat)

  names(longmat)[2] = "factors"
  names(longmat)[1] = "variable"

  g1 = ggplot(longmat, aes(x = factors, y = variable)) + geom_tile(aes(fill = value),colour = "grey20")

  g2 = g1 + scale_fill_gradient2(low = "#800000", high = "#191970", mid = "white")+ theme(
    axis.title.x = element_blank(),
    axis.title.y = element_blank(),
    # panel.grid.major = element_blank(),
    text = element_text(size = 12),
```

```r
    panel.border = element_blank(),
    panel.background = element_blank(),
    axis.ticks = element_blank(),
    # axis.text = element_blank(),
    legend.title = element_text(),
    plot.title = element_text(hjust = 0.5)) + labs(fill = " ")
  g2
}
```