A Tutorial for Bayesian Integrative Factor Models

Mavis Liang

2025-01-10

Table of contents

Welcome

This is the tutorial to guide the use of Bayesian integrative factor models. Including naive methods Stack FA (Stacking data from all the studies and apply FA), Ind FA (applying FA to each study separatly), and advanced methods PFA (Perturbed Factor Analysis) (Roy et al. 2021), MOM-SS (Bayesian Factor Regression with the non-local spike-and-slab priors) (Avalos-Pacheco, Rossell, and Savage 2022), SUFA (Subspace Factor Analysis) (Chandra, Dunson, and Xu 2024), BMSFA (Bayesian Multi-study Factor Analysis) (De Vito et al. 2021) and Tetris (Grabski et al. 2023).

Full intro to the multi-study data and comparison for these methods can be found in the manuscript. Raw code can be found the the GitHub repository (https://github.com/Mavis-Liang/Bayesian_integrative_FA_tutorial/tree/main)

Quick start

Step 1: Prepare the package and data

```
# install.packages("remotes")
# remotes::install_github("rdevito/MSFA")
library(MSFA)
```

A small simulated data can be downloaded here:

A small simulated data (if it does not trigger instant download, you can go to the repo provided above and find it under folder `RDS`).

```
data <- readRDS("Data/sim_B_small.rds")
```

A glance at the data:

```
dim(data$Y_list[[1]])
```

```
[1] 50 20
```

For study 1, we have 50 samples and 20 features.

```
length(data$Y_list)
```

```
[1] 3
```

We have 3 studies in total.

```
sapply(data$Y_list, dim)
```

```
     [,1] [,2] [,3]
[1,]   50   50   50
[2,]   20   20   20
```

The second and third studies also have 50 samples and 20 features.

Step 2: Run BMSFA

```
fit1 <- MSFA::sp_msfa(data$Y_list, k = 5, j_s = c(4, 4, 4),
                      scaling = FALSE, centering = TRUE,
                      control = list(nrun = 5000, burn = 4000))
```

```
r= 1000
r= 2000
r= 3000
r= 4000
r= 5000
```

Step 3: Post-processing

We use `sp_OP` to apply orthogonal Procrustes rotation to the posterior samples of the common loading matrix Φ and the study-specific loading matrices Λ_s. The function `sp_OP` is from the `MSFA` package.

Common covariance and study-specific covariance matrices are calculated by the cross-product of the loading matrices.

The marginal covariance matrix is calculated by $\Sigma_s = \Phi\Phi^T + \Lambda_s\Lambda_s^T + diag(\Psi_s)$, where Ψ_s is the diagonal matrix of the residual variances for study s.

```r
library(tidyverse) # for data wrangling
post_BMSFA <- function(fit){
  # Common covariance matrix and loading
  est_Phi <- MSFA::sp_OP(fit$Phi, trace=FALSE)$Phi
  est_SigmaPhi <- tcrossprod(est_Phi)

  # Study-specific covariance matrices and loadings
  est_LambdaList <- lapply(fit$Lambda, function(x) MSFA::sp_OP(x, trace=FALSE)$Phi)
  est_SigmaLambdaList <- lapply(est_LambdaList, function(x) tcrossprod(x))

  # Marginal covariance matrices
  S <- length(est_SigmaLambdaList)
  # Get point estimate of each Psi_s
  est_PsiList <- lapply(1:S, function(s) {
    apply(fit$psi[[s]], c(1, 2), mean)
  })
  est_margin_cov <- lapply(1:S, function(s) {
    est_SigmaPhi + est_SigmaLambdaList[[s]] + diag(est_PsiList[[s]] %>% as.vector())
  })

  return(list(Phi = est_Phi, SigmaPhi = est_SigmaPhi,
         LambdaList = est_LambdaList, SigmaLambdaList = est_SigmaLambdaList,
         PsiList = est_PsiList,
         SigmaMarginal = est_margin_cov))
}
out1 <- post_BMSFA(fit1)
```

Now, determine the number of factors using eigen value decomposition (EVD).

```r
# Obtain the eigen values from the common covariance matrix
val_eigen_SigmaPhi <- eigen(out1$SigmaPhi)$values
# Proportion of variance explained by each eigen value
prop_var_SigmaPhi <- val_eigen_SigmaPhi/sum(val_eigen_SigmaPhi)
# Choose the number of factors - factors that explain more than 5% of the variance
choose_K_SigmaPhi <- length(which(prop_var_SigmaPhi > 0.05))

# Similarly, for each study-specific covariance matrix:
val_eigen_SigmaLambdaList <- lapply(out1$SigmaLambdaList, function(x) eigen(x)$values)
prop_var_SigmaLambdaList <- lapply(val_eigen_SigmaLambdaList, function(x) x/sum(x))
choose_K_SigmaLambdaList <- lapply(prop_var_SigmaLambdaList, function(x) length(which(x > 0.0
```

```r
# Print the output
choose_K_SigmaPhi
```

```
[1] 4
```

```r
for(s in 1:length(choose_K_SigmaLambdaList)){
  cat("Study", s, "has", choose_K_SigmaLambdaList[[s]], "factors.\n")
}
```

```
Study 1 has 1 factors.
Study 2 has 1 factors.
Study 3 has 2 factors.
```

Now we rerun the model with the number of factors we just determined. And again we post-process the MCMC chains with `post_BMSFA()`.

```r
fit2 <- MSFA::sp_msfa(data$Y_list, k = 4, j_s = c(1, 1, 2),
                      scaling = FALSE, centering = TRUE,
                      control = list(nrun = 5000, burn = 4000))
```

```
r= 1000
r= 2000
r= 3000
r= 4000
r= 5000
```

```r
out2 <- post_BMSFA(fit2)
```

Step 4: Check our results

```r
# For example, view the loading matrix
head(out2$Phi)
```

```
            [,1]       [,2]        [,3]      [,4]
[1,] -0.23132474 -1.1270176  0.11836711 0.3704269
[2,]  0.05108547 -1.1317790  0.14411934 0.4161137
[3,]  0.16617280 -0.8373175  0.02428551 0.4504216
[4,]  0.17184504 -0.6504157  0.10890033 0.3639910
[5,]  0.24622013 -0.4159994 -0.03999977 0.2910912
[6,]  0.34566682 -0.2943820 -0.08561154 0.3553088
```

```r
dim(out2$Phi)
```

```
[1] 20  4
```

1 Package Installation

I have this separate section to introduce the package installation because installing some packages requires extra efforts.

1.1 Stack FA, Ind FA, and BMSFA

The Stack FA, Ind FA, and BMSFA models are implemented in the `MSFA` package, which is not available on CRAN. We can install it from GitHub using the `remotes` package. We use `sp_fa()` within the `MSFA` package to fit the Stack FA and Ind FA model, and `sp_msfa()` to fit the BMSFA model.

```
install.packages("remotes")
remotes::install_github("rdevito/MSFA")
library(MSFA)
```

I edited the functions a bit so that sp_fa() and sp_msfa() can accept the `scaling` and `centering` arguments. So that we can only center the data without scaling it, or centering and scaling, or neither.

```
#devtools::install_github("Mavis-Liang/MSFA")
```

1.2 PFA

PFA does not provide any downloadable R packages and we need to download the R scripts from their GitHub repository, put them in the same directory as the main script, and source them for use.

We only need the three files: `FBPFA-PFA.R`, `FBPFA-PFA with fixed latent dim.R`, and `PFA.cpp`, which can be found in https://github.com/royarkaprava/Perturbed-factor-models. The `FBPFA-PFA.R` file contains the full Bayesian inference algorithm for the PFA model, directly set the latent dimensions equal to the dimensions or the original data. The `FBPFA-PFA with fixed latent dim.R` file contains the same algorithm that requires to set numbers of common factors K. We also notice that two version of the models are both `PFA()`, and some

functions in the `FBPFA-PFA with fixed latent dim.R` file depends on the `FBPFA-PFA.R` file. Therefore, since we want to run the dimension reduction version of the model, we must source the `FBPFA-PFA.R` file first, and then source the `FBPFA-PFA with fixed latent dim.R` file.

```
# Suppose the files are in the same directory as the main script
source("FBPFA-PFA.R")
source("FBPFA-PFA with fixed latent dim.R")
```

1.3 MOM-SS

```
BiocManager::install("sparseMatrixStats") # Dependency
install.packages("mombf")

install.packages("devtools")

devtools::install_github("AleAviP/BFR.BE")
library(BFR.BE)
```

1.4 SUFA

To install SUFA on Linux, you need to also install extra dependencies like PROJ, sqlite3 and GDAL onto PATH. On Windows, you might need to do several updates, particularly the updates for terra. We can skip building the vignettes to save time, as it contains the computation of a large dataset.

```
devtools::install_github("noirritchandra/SUFA", build_vignettes = FALSE)
library(SUFA)
```

1.5 Tetris

Similar to PFA, Tetris does not provide any downloadable R packages and we need to download the R scripts from their GitHub repository, put them in the same directory as the main script, and source them for use. The R scripts can be found in https://github.com/igrabski/tetris/tree/main.

```
# Suppose the files are in the same directory as the main script
source("Tetris.R")
```

1.6 Other utility packages

```
library(tidyverse)
library(Matrix)#for the bdiag function
```

2 Case study: nutrition data

2.1 Loading and previewing the data

The data used in this section is from a large multi-site study investigating health and diet among Hispanic/Latino adults in the United States (De Vito et al. 2022). This data is not publicly available. Please contact the authors of the original study for access.

However, you can use this simulated data instead: simulated_nutrition_data.rds (if it does not triggers instant download, you can find it in the repo site). To read in the data, use `readRDS()`. The loaded object contains data in both `Y_mat` and `Y_list` formats, as well as covariates and other information. It is generated with the Scenario 4 in the manuscript (see code). Note that instead of having 6 studies, this simulated data has 12 studies.

```
load("./Data/dataLAT_projale2.rda")
```

The resulting object is a list of 6 data frames, each corresponding to a different study. Each data frame contains information about the nutritional intake of individuals, and the columns represent different nutrients. From Study 1 to Study 6, the number of individuals (N_s) are 1364, 1517, 2210, 5184, 2478, and 959, respectively, and the number of nutrients (P) are all 42.

```
# Check how many studies in the list
length(X_s2)
```

```
[1] 6
```

```
# Dimension of each study
lapply(X_s2, dim)
```

```
[[1]]
[1] 1364   42

[[2]]
[1] 1517   42
```

```
[[3]]
[1] 2210   42

[[4]]
[1] 5184   42

[[5]]
[1] 2478   42

[[6]]
[1] 959  42
```

Let's take a look at the first few rows of the first data frame to get an idea of the data structure.

```
X_s2[[1]][1:5, 1:5]
```

```
  Animal Protein (g) Vegetable Protein (g) Cholesterol (mg)  SCSFA MCSFA
1            28.9560                14.7440          256.761 0.2665 0.939
2            33.6675                 8.9710          104.217 0.2180 0.520
3            70.0000                31.0635          207.902 0.9845 1.692
4            20.6700                13.8240          148.921 0.0625 0.239
5            15.4250                10.5550           65.060 0.0090 0.033
```

We note that the data we have available is different from the original data (cite). The original data is a collection of 12 studies, and there are known covariates for each individuals, like the one we simulated in the previous section. However, for the purpose of this case study, the data we used are collapsed into 6 studies, and only the nutritional intake data are available.

2.2 Data preprocessing

Some individuals have missing values for all nutrients, thus we will remove these individuals from the data. Also, there are some nutrition intake are less than zero, for which we will replace with 0. We then apply a log transformation to the data.

We first count how many NA values and negative values are in each study.

```r
count_na_and_negatives <- function(df) {
  # Count NA values
  na_count <- sum(is.na(df))
  # Count negative values
  negative_count <- sum(df < 0, na.rm = TRUE)

  # Print counts
  cat("Number of NAs:", na_count, "\n")
  cat("Number of negative values:", negative_count, "\n")
}
invisible(lapply(X_s2, count_na_and_negatives))
```

```
Number of NAs: 1344
Number of negative values: 0
Number of NAs: 1344
Number of negative values: 1
Number of NAs: 1344
Number of negative values: 0
Number of NAs: 1344
Number of negative values: 2
Number of NAs: 1344
Number of negative values: 1
Number of NAs: 1344
Number of negative values: 0
```

We will define a function to process the data, which removes rows where all values are NA. We also define a function that replaces negative values with 0, and applies a log transformation to the data.

```r
process_study_data <- function(df) {
  # Remove rows where all values are NA
  cleaned_df <- df[!apply(df, 1, function(row) all(is.na(row))), , drop = FALSE]
  # Count remaining rows
  remaining_rows <- nrow(cleaned_df)
  # Print results for the study
  cat("Remaining rows:", remaining_rows, "\n")
  return(cleaned_df)
}
Y_list <- lapply(X_s2, process_study_data)
```

```
Remaining rows: 1332
```

```
Remaining rows: 1485
Remaining rows: 2178
Remaining rows: 5152
Remaining rows: 2446
Remaining rows: 927
```

The numbers of individuals in each study left for analysis (N_s) are 1332, 1485, 2178, 5152, 2446, and 927, respectively.

```r
# Replace negative values with 0, then log(x+0.01) + 0.01
replace_negatives <- function(df) {
  # Replace negative values with 0
  df[df < 0] <- 0
  # Apply log transformation. Add 0.01 to avoid log(0).
  transformed_df <- log(df + 0.01)
  return(transformed_df)
}

Y_list <- lapply(Y_list, replace_negatives)
```

```r
# Check the processed data
invisible(lapply(Y_list, count_na_and_negatives))
```

```
Number of NAs: 0
Number of negative values: 11910
Number of NAs: 0
Number of negative values: 11222
Number of NAs: 0
Number of negative values: 15006
Number of NAs: 0
Number of negative values: 36230
Number of NAs: 0
Number of negative values: 19349
Number of NAs: 0
Number of negative values: 6707
```

Now we don't have any NA values or negative values in the data.

The assumptions for factor models require that each variable has a mean of 0. Therefore, for each study, we will center the data for each column. We note that for some model (Stack FA, Ind FA, BMSFA, and Tetris), this step is handeled internally, and for MOM-SS, the random intercepts are estimated, so we do not need to center the data.

```r
Y_list_scaled <- lapply(
  Y_list, function(x) scale(x, center = TRUE, scale = FALSE)
)
Y_mat_scaled <- Y_list_scaled %>% do.call(rbind, .) %>% as.matrix()
```

2.3 Model fitting

We recommend running model fitting and post-processing in a high-performance computing environment, as the model fitting process can be computationally intensive. PFA and Tetris are particularly computationally expensive, where PFA requires more than 10 hours to run, and Tetris requires more than 4 days to run. Other models can be completed in half an hour. We recommend at least 5GB of memory for running the models and post-processing.

```r
# Stack FA
Y_mat =  Y_list %>% do.call(rbind, .) %>% as.matrix()
fit_stackFA <- MSFA::sp_fa(Y_mat, k = 6, scaling = FALSE, centering = TRUE,
                           control = list(nrun = 10000, burn = 8000))
# Ind FA
fit_indFA <-
      lapply(1:6, function(s){
        j_s = c(8, 8, 8, 8, 8, 8)
        MSFA::sp_fa(Y_list[[s]], k = j_s[s], scaling = FALSE, centering = TRUE,
                    control = list(nrun = 10000, burn = 8000))
      })

# PFA
N_s <- sapply(Y_list, nrow)
fit_PFA <- PFA(Y=t(Y_mat_scaled),
                        latentdim = 6,
                        grpind = rep(1:6,
                                     times = N_s),
                Thin = 5,
                Cutoff = 0.001,
                Total_itr = 10000, burn = 8000)

# MOM-SS
Y_mat =  Y_list %>% do.call(rbind, .) %>% as.matrix()
# Construct the membership matrix
N_s <- sapply(Y_list, nrow)
M_list <- list()
```

```
   for(s in 1:6){
     M_list[[s]] <- matrix(1, nrow = N_s[s], ncol = 1)
   }
M <- as.matrix(bdiag(M_list))
fit_MOMSS <- BFR.BE::BFR.BE.EM.CV(x = Y_mat, v = NULL,
                                  b = M, q = 6, scaling = FALSE)


# SUFA
fit_SUFA <- SUFA::fit_SUFA(Y_list_scaled, qmax=6, nrun = 10000)

# BMSFA
fit_BMSFA <- MSFA::sp_msfa(Y_list, k = 6, j_s = c(2, 2, 2, 2, 2, 2),
                           outputlevel = 1, scaling = FALSE,
                           centering = TRUE,
                           control = list(nrun = 10000, burn = 8000))
```

Fitting Tetris requires a 3-step process. First, we run `tetris()` to draw posterior samples of the model parameters, including $\mathcal{T}$. Then we run `choose.A()` to choose the best $\mathcal{T}$ based on the posterior samples. Finally, we run `tetris()` again with the chosen $\mathcal{T}$ to obtain the final model. Hyperparameters $\alpha_{\mathcal{T}}$ are set to 1.25 times the number of studies.

```
# Tetris
set_alpha <- ceiling(1.25*6)
fit_Tetris <- tetris(Y_list, alpha=set_alpha, beta=1, nprint = 200,
                     nrun=10000, burn=8000)
big_T <- choose.A(fit_Tetris, alpha_IBP=set_alpha, S=6)
run_fixed <- tetris(Y_list, alpha=set_alpha, beta=1,
                    fixed=TRUE, A_fixed=big_T, nprint = 200,
                    nrun=10000, burn=8000)
```

2.4 Post processing

Post processing includes calculating the point estimates of the factor loadings and covariance matrix from the posterior samples, and determing the number of factors for each model.

For methods of MOM-SS, SUFA, and Tetris, the number of factors is determined internally in the algorithms, therefore, the output of the models are final results.

For MOM-SS, Φ is directly obtained with its post-processed common loadings in the fitted output. Th common covariance is calculated with $\Phi\Phi^\top$. The marginal covariance matrix

Σ_{marginal} is calculated by adding the estimated study-specific error covariances to the common variance. The study-specific intercepts α and the coefficients for the known covariates B are also extracted from the fitted output.

```
post_MOMSS <- function(fit, version = 2){ # version 1: M, version 2: Mpost
  est_Phi <- fit$M
  if (version==2){est_Phi <- fit$Mpost}
  est_SigmaPhi <- tcrossprod(est_Phi)

  # Marginal covariance
  S <- dim(fit$sigma)[2]
  est_PsiList <- est_SigmaMarginal <-  list()
  for(s in 1:S){
    est_PsiList[[s]] <- fit$sigma[,s]
    est_SigmaMarginal[[s]] <- est_SigmaPhi + diag(fit$sigma[,s])
  }
  # last S columns of fit$Theta are the study-specific intercepts
  est_alphas <- fit$Theta[, (dim(fit$Theta)[2]-S+1):dim(fit$Theta)[2]]
  # The rest are coeficients for the known covariates
  est_B <- fit$Theta[, 1:(dim(fit$Theta)[2]-S)]

  return(list(Phi = est_Phi, SigmaPhi = est_SigmaPhi, Psi = est_PsiList, alpha = est_alphas,
              SigmaMarginal = est_SigmaMarginal))
}
res_MOMSS <- post_MOMSS(fit_MOMSS)
saveRDS(res_MOMSS, "Data/Rnutrition_MOMSS.rds")
```

For SUFA, the shared and study-specific loading matrices, as well as the common and marginal covariance can be conveniently obtained via the `lam.est.all()`, `SUFA_shared_covmat()` and `sufa_marginal_covs()` functions. Error covariance is obtained by taking averages of the "residuals" fitted output. And the study-specific covariance matrices are calculated by subtracting the common covariance from the marginal covariance. Note that in the definition of SUFA, common covariance is $\Phi\Phi^\top + \Sigma$.

```
post_SUFA <- function(fit){
  all <- dim(fit$Lambda)[3]
  burnin <- floor(all * 0.8) # We will use the last 20% samples
  # shared and study-specific loading matrices
  loadings <- lam.est.all(fit, burn = burnin)
  # Obtain common covariance matrix and loading from fitting
  est_Phi <- loadings$Shared
  est_SigmaPhi <- SUFA_shared_covmat(fit, burn = burnin)
```

```
  est_Psi <- diag(colMeans(fit$residuals))
  # Study-specific loadings
  est_LambdaList <- loadings$Study_specific

  # Obtain study-specific covariance matrices
  S <- length(fit$A)
  marginal_cov <- sufa_marginal_covs(fit, burn = burnin)
  est_SigmaLambdaList <- list()
  for (s in 1:S) {
    est_SigmaLambdaList[[s]] <- marginal_cov[,,s] - est_SigmaPhi
  }

  return(list(SigmaPhi = est_SigmaPhi, Phi = est_Phi,
              SigmaLambdaList = est_SigmaLambdaList,
              LambdaList = est_LambdaList,
              Psi = est_Psi,
              SigmaMarginal = lapply(1:S, function(s) marginal_cov[,,s])
              ))
}
res_SUFA <- post_SUFA(fit_SUFA)
saveRDS(res_SUFA, "Data/Rnutrition_SUFA.rds")
```

For Tetris, the common loading matrix Φ can be obtained through the `getLambda()` function. The common covariance matrix is calculated as $\Phi\Phi^\top$. The study-specific loading matrices Λ_s are obtained by multiplying the common loading matrix with the study-specific matrices $T_s - P$. The study-specific covariance matrices are calculated as $\Lambda_s\Lambda_s^\top$. The marginal covariance matrix is calculated as $\Lambda T \Lambda^\top + \Psi$.

```
post_Tetris <- function(fit){
  # Estimated common covariance
  A <- fit$A[[1]]
  Lambda <- getLambda(fit,A)
  S <- dim(A)[1]
  est_Phi <- as.matrix(Lambda[,colSums(A)==S])
  est_SigmaPhi <- tcrossprod(est_Phi)
  # Estimated study-specific covariance
  P = diag((colSums(A) == S)*1)
  T_s <- list()
  est_LambdaList <- list()
  for(s in 1:S){
    T_s[[s]] <- diag(A[s,])
    Lambda_s <- Lambda %*% (T_s[[s]] - P)
```

```
    Lambda_s <- Lambda_s[,-which(colSums(Lambda_s == 0) == nrow(Lambda_s))]
    Lambda_s <- matrix(Lambda_s, nrow=nrow(Lambda))
    est_LambdaList[[s]] <- Lambda_s}
  est_SigmaLambdaList <- lapply(1:S, function(s){
    tcrossprod(est_LambdaList[[s]])})

  # Estimated marginal covariance
  Psi <- list()
  est_SigmaMarginal <- lapply(1:S, function(s){
    Psi[[s]] <- diag(Reduce("+", fit$Psi[[s]])/length(fit$Psi[[s]]))
    Sigma_s <- Lambda %*% T_s[[s]] %*% t(Lambda) + Psi[[s]]
    })

  return(list(Phi = est_Phi, SigmaPhi = est_SigmaPhi,
              LambdaList = est_LambdaList, SigmaLambdaList = est_SigmaLambdaList,
              Psi = Psi, T_s = T_s,
              SigmaMarginal = est_SigmaMarginal))
}
res_Tetris <- post_Tetris(run_fixed)
saveRDS(res_Tetris, "Data/Rnutrition_Tetris.rds")
```

The following code shows the post-processing for PFA.First, we post-process for the number of factors. We first extracts the number of factors K for each posterior sample, by calculating the number of columns in the loadings matrix. Then we finds the mode K of these values, and filter posterior samples to only those with the modal K. We continue with the downstream summary using only those aligned samples. To get the common loadings by its definition, we first multiply Φ with $V^{1/2}$ in the samples we kept. While the common loading matrix Φ can be simply calculated by the average of the posterior $\Phi V^{1/2}$, with its adjustment to address identifiability issues, we still apply OP to $\Phi V^{1/2}$. For the covariances such as the common covariance, we calculate $\Phi V \Phi^\top$ at each iterations, and then use their average as the final results. Similar procedure is applied for the study-specific covariances and marginal covariance.

```
post_PFA <- function(fit) {
  # Determine posterior dimension (number of factors per sample)
  k_vec <- sapply(fit$Loading, ncol)
  mode_k <- as.numeric(names(sort(table(k_vec), decreasing = TRUE)[1]))

  # Filter posterior samples to those with mode_k
  keep_idx <- which(k_vec == mode_k)
  fit$Loading <- fit$Loading[keep_idx]
  fit$Latentsigma <- fit$Latentsigma[keep_idx]
  fit$Errorsigma <- fit$Errorsigma[keep_idx]
```

```r
fit$Pertmat <- fit$Pertmat[keep_idx]

npost <- length(fit$Loading)
p <- nrow(fit$Loading[[1]])
k <- mode_k
S <- dim(fit$Pertmat[[1]])[2]

posteriorPhis <- array(0, dim = c(p, k, npost))
posteriorLams <- vector("list", S)

for(s in 1:S){
  posteriorLams[[s]] <- array(0, dim = c(p, k, npost))
  for(i in 1:npost){
    posteriorPhis[,,i] <- fit$Loading[[i]] %*% diag(fit$Latentsigma[[i]])
    posteriorLams[[s]][,,i] <- (solve(matrix(fit$Pertmat[[i]][, s], p, p)) - diag(p)) %*% p
  }
}

# Varimax rotation
est_Phi <- MSFA::sp_OP(posteriorPhis, itermax = 10, trace = FALSE)$Phi
est_speLoad <- lapply(posteriorLams, function(x) MSFA::sp_OP(x, itermax = 10, trace = FALSE

# Estimated covariance components
sharevar <- list()
est_SigmaLambdaList <- vector("list", S)
est_SigmaMarginal <- vector("list", S)
est_Psi_list <- list()

for(s in 1:S){
  post_SigmaLambda_s <- vector("list", npost)
  post_SigmaMarginal_s <- vector("list", npost)
  Psi <- vector("list", npost)

  for(i in 1:npost){
    sharevar[[i]] <- fit$Loading[[i]] %*% diag(fit$Latentsigma[[i]]^2) %*% t(fit$Loading[[i
      diag(fit$Errorsigma[[i]]^2)
    Q_temp_inv <- solve(matrix(fit$Pertmat[[i]][, s], p, p))
    post_SigmaMarginal_s[[i]] <- Q_temp_inv %*% sharevar[[i]] %*% t(Q_temp_inv)
    post_SigmaLambda_s[[i]] <- post_SigmaMarginal_s[[i]] - sharevar[[i]]
    Psi[[i]] <- diag(fit$Errorsigma[[i]]^2)
  }
```

```
    est_SigmaMarginal[[s]] <- Reduce('+', post_SigmaMarginal_s) / npost
    est_SigmaLambdaList[[s]] <- Reduce('+', post_SigmaLambda_s) / npost
    est_Psi_list[[s]] <- Reduce('+', Psi) / npost
  }

  est_Psi <- Reduce('+', est_Psi_list) / S
  est_SigmaPhi <- Reduce('+', sharevar) / npost
  est_Q <- Reduce('+', fit$Pertmat) / npost
  est_Q_list <- lapply(1:S, function(s) matrix(est_Q[, s], p, p))

  return(list(
    Phi = est_Phi,
    SigmaPhi = est_SigmaPhi,
    Psi = est_Psi,
    Q = est_Q_list,
    LambdaList = est_speLoad,
    SigmaLambdaList = est_SigmaLambdaList,
    SigmaMarginal = est_SigmaMarginal,
    mode_k = mode_k,
    kept_samples = length(keep_idx)
  ))
}
res_PFA <- post_PFA(fit_PFA)
saveRDS(res_PFA, "Data/Rnutrition_PFA.rds")
```

```
# columns have all loadings less than 10^-3
fun_neibour <- function(Phi, threshold = 1e-3) {
  return(
    sum(apply(Phi, 2, function(x) {
      sum(abs(x) <= threshold) < length(x)}
    ))
  )
}

# The post_PFA(fit_PFA) object
res_PFA <- readRDS("Data/Rnutrition_PFA.rds")
Phi_PFA <- res_PFA$Phi
K_PFA <- fun_neibour(Phi_PFA)
K_PFA
```

```
[1] 6
```

The estimated K for PFA is 6.

Next, we show how to process the output of Stack FA, Ind FA, and BMSFA. While point estimates for the loadings and covariances can be obtained in the similar way as in PFA, the number of factors is determined with eigen value decompositions of the covariance matrix. Once the numbers of factors are determined, we have to run the models again with the correct number of factors, and then extract the final results.

Therefore, for Stack FA, we first extract the point estimates of the Φ by applying OP to the posterior samples of the loadings. The common covariance matrix is calculated as $\Phi\Phi^\top$. The marginal covariance matrix is calculated as the average of its posterior samples.

```
post_stackFA <- function(fit, S){
  est_Phi <- MSFA::sp_OP(fit$Lambda, trace=FALSE)$Phi
  est_SigmaPhi <- tcrossprod(est_Phi)
  est_SigmaMarginal <-  lapply(1:S, function(s)
    apply(fit$Sigma, c(1, 2), mean)
  )
  Psi_chain <- list()
  for(i in 1:dim(fit$Sigma)[3]){
    Psi_chain[[i]] <- fit$Sigma[, , i] - tcrossprod(fit$Lambda[, , i])
  }
  est_Psi <- Reduce('+', Psi_chain)/length(Psi_chain)
  return(list(Phi = est_Phi, SigmaPhi = est_SigmaPhi, Psi = est_Psi,
              SigmaMarginal = est_SigmaMarginal))
}
res_stackFA <- post_stackFA(fit_stackFA, S=6)
saveRDS(res_stackFA, "Data/Rnutrition_StackFA.rds")
```

After that, we determine the number of factors by eigen value decomposition of the common covariance matrix. We then run the model again with the correct number of factors, and extract the final results.

```
fun_eigen <- function(Sig_mean) {
  val_eigen <- eigen(Sig_mean)$values
  prop_var <- val_eigen/sum(val_eigen)
  choose_K <- length(which(prop_var > 0.05))
  return(choose_K)
}
res_stackFA <- readRDS("Data/Rnutrition_StackFA.rds")
SigmaPhi_StackFA <- res_stackFA$SigmaPhi
K_StackFA <- fun_eigen(SigmaPhi_StackFA)
K_StackFA
```

```
[1] 4
```

The estimated K for Stack FA is 4.

Then we re-run the model with the correct number of factors, and extract the final results.

```
fit_stackFA_2 <- MSFA::sp_fa(Y_mat_scaled, k = K_StackFA, scaling = FALSE, centering = TRUE,
                             control = list(nrun = 10000, burn = 8000))
res_stackFA_2 <- post_stackFA(fit_stackFA_2, S=6)
saveRDS(res_stackFA_2, "Data/Rnutrition_StackFA_2.rds")
```

We repeat this process for Ind FA and BMSFA.

```
# Ind FA
post_indFA <- function(fit){
  # Estimated study-specific covariance and loading
  S <- length(fit_list)
  est_LambdaList <- lapply(1:S, function(s){
    MSFA::sp_OP(fit_list[[s]]$Lambda, trace=FALSE)$Phi
  })
  est_SigmaLambdaList <- lapply(est_LambdaList, function(x) tcrossprod(x))

  # Marginal covariance matrices
  est_SigmaMarginal <- lapply(1:S, function(s) {
    fit <- fit_list[[s]]
    apply(fit$Sigma, c(1, 2), mean)
  })

  Psi <- list()
  for(s in 1:S){
    Psi_chain <- list()
    for(i in 1:dim(fit_list[[1]]$Sigma)[3]){
      Psi_chain[[i]] <- fit_list[[s]]$Sigma[, , i] - tcrossprod(fit_list[[s]]$Lambda[, , i])
    }
    Psi[[s]] <- Reduce('+', Psi_chain)/length(Psi_chain)
  }

  return(list(LambdaList = est_LambdaList, SigmaLambdaList = est_SigmaLambdaList, Psi = Psi,
              SigmaMarginal = est_SigmaMarginal))
}

res_indFA <- post_indFA(fit_indFA)
```

```r
saveRDS(res_indFA, "Data/Rnutrition_IndFA.rds")

# BMSFA
post_BMSFA <- function(fit){
  # Common covariance matrix and loading
  est_Phi <- sp_OP(fit$Phi, trace=FALSE)$Phi
  est_SigmaPhi <- tcrossprod(est_Phi)

  # Study-specific covariance matrices and loadings
  est_LambdaList <- lapply(fit$Lambda, function(x) sp_OP(x, trace=FALSE)$Phi)
  est_SigmaLambdaList <- lapply(est_LambdaList, function(x) tcrossprod(x))

  # Marginal covariance matrices
  S <- length(est_SigmaLambdaList)
  # Get point estimate of each Psi_s
  est_PsiList <- lapply(1:S, function(s) {
    apply(fit$psi[[s]], c(1, 2), mean)
  })
  est_margin_cov <- lapply(1:S, function(s) {
    est_SigmaPhi + est_SigmaLambdaList[[s]] + diag(est_PsiList[[s]] %>% as.vector())
  })

  return(list(Phi = est_Phi, SigmaPhi = est_SigmaPhi,
         LambdaList = est_LambdaList, SigmaLambdaList = est_SigmaLambdaList,
         PsiList = est_PsiList,
         SigmaMarginal = est_margin_cov))
}
res_BMSFA <- post_BMSFA(fit_BMSFA)
saveRDS(res_BMSFA, "Data/Rnutrition_BMSFA.rds")
```

```r
SigmaLambda_IndFA <- readRDS("Data/Rnutrition_IndFA.rds")$SigmaLambda
Js_IndFA <- lapply(SigmaLambda_IndFA, fun_eigen)
Js_IndFA %>% unlist()
```

```
[1] 4 5 5 5 4 4
```

```r
SigmaPhi_BMSFA <- readRDS("Data/Rnutrition_BMSFA.rds")$SigmaPhi
K_BMSFA <- fun_eigen(SigmaPhi_BMSFA)
SigmaLambda_BMSFA <- readRDS("Data/Rnutrition_BMSFA.rds")$SigmaLambda
Js_BMSFA <- lapply(SigmaLambda_BMSFA, fun_eigen)
K_BMSFA %>% unlist()
```

```
[1] 4
```

```
Js_BMSFA %>% unlist()
```

```
[1] 2 2 2 2 2 2
```

The estimated J_s for Ind FA are 4, 5, 5, 5, 4, and 4. The estimated K for BMSFA is 4 and the estimated J_s are 2, 2, 2, 2, 2, and 2

Then we re-run the models with the correct number of factors, and extract the final results.

```
# Ind FA
fit_indFA_2 <-
  lapply(1:6, function(s){
    j_s = c(4, 5, 5, 5, 4, 4)
    MSFA::sp_fa(Y_list[[s]], k = j_s[s], scaling = FALSE, centering = TRUE,
                control = list(nrun = 10000, burn = 8000))
  })
res_indFA_2 <- post_indFA(fit_indFA_2)
saveRDS(res_indFA_2, "Data/Rnutrition_IndFA_2.rds")

# BMSFA
fit_BMSFA_2 <- MSFA::sp_msfa(Y_list, k = 4, j_s = c(2, 2, 2, 2, 2, 2),
                             outputlevel = 1, scaling = FALSE,
                             centering = TRUE,
                             control = list(nrun = 10000, burn = 8000))
res_BMSFA_2 <- post_BMSFA(fit_BMSFA_2)
saveRDS(res_BMSFA_2, "Data/Rnutrition_BMSFA_2.rds")
```

Now the final results are obtained and you can get the saved files mentioned above from the GitHub repository.

2.5 Visualization

We can make some heatmap for the loadings. Please see the figures in the paper.

2.6 Mean squared error (MSE)

We can assess the goodness-of-fit of the models by reconstructing data and calculating the reconstruction errors. With estimated loadings and error covariances, we can estimate the factor scores for a new observation, $\hat{\mathbf{f}}_{is,(new)}$ and $\hat{\mathbf{l}}_{is,(new)}$, derived by adapting the Bartlett method**?**, **?**. Then we can use the estimated factor scores, together with the estimated loadings, to reconstruct $\hat{\mathbf{y}}_{is,(new)}$ and calculate the reconstruction error which represents the fit of the models. To be specific, suppose that we have the multivariate data of a new observation from a specific study, $\mathbf{y}_{is,(new)}$, we can estimate its factor score and reconstruct its $\hat{\mathbf{y}}_{is,(new)}$ in the following ways:

- Stack FA: $\hat{\mathbf{f}}_{is,(new)} = (\widehat{\Phi}^\top\widehat{\Psi}^{-1}\widehat{\Phi})^{-1}\widehat{\Phi}^\top\widehat{\Psi}^{-1}\mathbf{y}_{is,(new)}$, and $\hat{\mathbf{y}}_{is,(new)} = \widehat{\Phi}\hat{\mathbf{f}}_{is,(new)}$.
- Ind FA: $\hat{\mathbf{l}}_{is,(new)} = (\widehat{\Lambda_s}^\top\widehat{\Psi_s}^{-1}\widehat{\Lambda_s})^{-1}\widehat{\Lambda_s}^\top\widehat{\Psi_s}^{-1}\mathbf{y}_{is,(new)}$, and $\hat{\mathbf{y}}_{is,(new)} = \widehat{\Lambda_s}\hat{\mathbf{l}}_{is,(new)}$.
- PFA: $\hat{\mathbf{f}}_{is,(new)} = (\widehat{\Phi}^\top\widehat{\Psi}^{-1}\widehat{\Phi})^{-1}\widehat{\Phi}^\top\widehat{\Psi}^{-1}\widehat{Q_s}\mathbf{y}_{is,(new)}$, and $\hat{\mathbf{y}}_{is,(new)} = \widehat{Q_s}^{-1}\widehat{\Phi}\hat{\mathbf{f}}_{is,(new)}$.
- MOM-SS: $\hat{\mathbf{f}}_{is,(new)} = (\widehat{\Phi}^\top\widehat{\Psi_s}^{-1}\widehat{\Phi})^{-1}\widehat{\Phi}^\top\widehat{\Psi_s}^{-1}(\mathbf{y}_{is,(new)} - \widehat{\mathbf{s}} - \hat{}\mathbf{x}_{is,(new)})$, and $\hat{\mathbf{y}}_{is,(new)} = \widehat{\mathbf{s}} + \hat{}\mathbf{x}_{is,(new)} + \widehat{\Phi}\hat{\mathbf{f}}_{is,(new)}$.
- SUFA: let $\Omega = [\widehat{\Phi}, \widehat{\Phi}\widehat{A_s}]$, then $\begin{bmatrix}\hat{\mathbf{f}}_{is,(new)}\\ \hat{\mathbf{l}}_{is,(new)}\end{bmatrix} = (\Omega^\top\widehat{\Psi}^{-1}\Omega)^{-1}\Omega^\top\widehat{\Psi}^{-1}\mathbf{y}_{is,(new)}$, and $\hat{\mathbf{y}}_{is,(new)} = \widehat{\Phi}\hat{\mathbf{f}}_{is,(new)} + \widehat{\Phi}\widehat{A_s}\hat{\mathbf{l}}_{is,(new)}$.
- BMSFA: let $\Omega = [\widehat{\Phi}, \widehat{\Lambda_s}]$, then $\begin{bmatrix}\hat{\mathbf{f}}_{is,(new)}\\ \hat{\mathbf{l}}_{is,(new)}\end{bmatrix} = (\Omega^\top\widehat{\Psi_s}^{-1}\Omega)^{-1}\Omega^\top\widehat{\Psi_s}^{-1}\mathbf{y}_{is,(new)}$, and $\hat{\mathbf{y}}_{is,(new)} = \widehat{\Phi}\hat{\mathbf{f}}_{is,(new)} + \widehat{\Lambda_s}\hat{\mathbf{l}}_{is,(new)}$.
- Tetris: $\hat{\mathbf{f}}_{is,(new)} = (\widehat{T_s}^\top(\widehat{\Phi^*})^\top\widehat{\Psi_s}^{-1}\widehat{\Phi^*}\widehat{T_s})^{-1}\widehat{T_s}^\top(\widehat{\Phi^*})^\top\widehat{\Psi_s}^{-1}\mathbf{y}_{is,(new)}$, and $\hat{\mathbf{y}}_{is,(new)} = \widehat{\Phi^*}\widehat{T_s}\hat{\mathbf{f}}_{is,(new)}$.

If we divide the whole data into training set and test set, we can obtain the **mean square difference (MSE)** between the true and estimated $\mathbf{y}_{is,(new)}$ in the test set for all individuals in all studies with $\frac{1}{P\sum_s^S N_s}\sum_s^S\sum_i^{N_s}\sum_p^P(\hat{y}_{isp,(new)} - y_{isp,(new)})^2$.

In the following, we divide the nutrition data into training set (70%) and test set (30%) and calculate the MSE of each model. Note that $\mathbf{y}_{new}$ should also be centered except for MOM-SS.

```r
train_ratio <- 0.7
train_list <- list()
test_list <- list()

for (s in seq_along(Y_list)) {
  N_s <- nrow(Y_list[[s]])  # Number of rows in the study
  train_indices <- sample(1:N_s, size = floor(train_ratio * N_s), replace = FALSE)

  train_list[[s]] <- Y_list[[s]][train_indices, ]
  test_list[[s]] <- Y_list[[s]][-train_indices, ]
}
# Test data has to be centered
test_list <- lapply(test_list, as.matrix)
test_list_scaled <- lapply(
  test_list, function(x) scale(x, center = TRUE, scale = FALSE)
)
```

We fit the models in the training data (on high-performance computing clusters), and then we load the fitted models and calculate the MSE on the test set for each model. The numbers of factors we input are determined by the previous results.

```r
fit_StackFA_train <- readRDS("Data/Rnutrition_stackFA_train.rds")
fit_IndFA_train <- readRDS("Data/Rnutrition_IndFA_train.rds")
fit_PFA_train <- readRDS("Data/Rnutrition_PFA_train.rds")
fit_MOMSS_train <- readRDS("Data/Rnutrition_MOMSS_train.rds")
fit_SUFA_train <- readRDS("Data/Rnutrition_SUFA_train.rds")
fit_BMSFA_train <- readRDS("Data/Rnutrition_BMSFA_train.rds")
fit_Tetris_train <- readRDS("Data/Rnutrition_Tetris_train.rds")
```

We used the derived MSE function to calculate the MSE for each model.

```r
# Stack FA
Phi <- fit_StackFA_train$Phi
Psi <- fit_StackFA_train$Psi

mse_stackFA <- 1/(42 * sum(sapply(test_list, nrow)))*sum(
  sapply(1:6, function(s){
    scores <- test_list_scaled[[s]] %*% solve(Psi) %*% Phi %*% mnormt::pd.solve(signif(t(Phi)
    norm(test_list_scaled[[s]] - scores %*% t(Phi), "F")^2
  })
)
```

```r
# Ind FA
LambdaList <- fit_IndFA_train$LambdaList
Psi <- fit_IndFA_train$Psi
mse_IndFA <- 1/(42 * sum(sapply(test_list, nrow)))*sum(
  sapply(1:6, function(s){
    scores <- test_list_scaled[[s]] %*% solve(Psi[[s]]) %*% LambdaList[[s]] %*% mnormt::pd.s

    norm(test_list_scaled[[s]] - scores %*% t(LambdaList[[s]]), "F")^2
  })
)
```

```r
# PFA
Phi <- fit_PFA_train$Phi
Psi <- fit_PFA_train$Psi
Q_list <- fit_PFA_train$Q
mse_PFA <- 1/(42 * sum(sapply(test_list, nrow)))*sum(
  sapply(1:6, function(s){
    scores <- test_list_scaled[[s]] %*% t(Q_list[[s]]) %*% Phi %*% mnormt::pd.solve(signif(t
    Y_est <- scores %*% t(Phi) %*% t(solve(Q_list[[s]]))
    norm(test_list_scaled[[s]] - Y_est, "F")^2
  })
)
```

```r
# MOM-SS
Phi <- fit_MOMSS_train$Phi
Psi <- fit_MOMSS_train$Psi %>% lapply(diag)
alpha <- lapply(1:6, function(s) {
  fit_MOMSS_train$alpha[,s]
})

mse_MOMSS <- 1/(42 * sum(sapply(test_list, nrow)))*sum(
  sapply(1:6, function(s){
    scores <- t(apply(test_list[[s]], 1, function(row) {row - alpha[[s]]})) %*% solve(Psi[[s]
    Y_est <- t(apply(scores %*% t(Phi), 1, function(row) {row + alpha[[s]]}))
    norm(test_list[[s]] - Y_est, "F")^2
  })
)
```

```r
# SUFA
Phi <- fit_SUFA_train$Phi
LambdaList <- fit_SUFA_train$LambdaList
Psi <- fit_SUFA_train$Psi
```

```
mse_SUFA <- 1/(42 * sum(sapply(test_list, nrow)))*sum(
  sapply(1:6, function(s){
    Omega <- cbind(Phi, LambdaList[[s]])
    scores <- test_list_scaled[[s]] %*% solve(Psi) %*% Omega %*% mnormt::pd.solve(signif(t(Om

    norm(test_list_scaled[[s]] - scores %*% t(Omega), "F")^2
  })
)
```

```
# BMSFA
Phi <- fit_BMSFA_train$Phi
LambdaList <- fit_BMSFA_train$LambdaList
Psi <- fit_BMSFA_train$PsiList %>%  lapply(as.vector) %>% lapply(diag)
mse_BMSFA <- 1/(42 * sum(sapply(test_list, nrow)))*sum(
  sapply(1:6, function(s){
    Omega <- cbind(Phi, LambdaList[[s]])
    scores <- test_list_scaled[[s]] %*% solve(Psi[[s]]) %*% Omega %*% mnormt::pd.solve(signi

    norm(test_list_scaled[[s]] - scores %*% t(Omega), "F")^2
  })
)
```

```
# Tetris
Phi <- fit_Tetris_train$Phi
LambdaList <- fit_Tetris_train$LambdaList
Marginal <- fit_Tetris_train$SigmaMarginal
SigmaPhi <- fit_Tetris_train$SigmaPhi
SigmaLambdaList <- fit_Tetris_train$SigmaLambdaList
Psi <- lapply(1:6, function(s) {
  Marginal[[s]] - SigmaPhi - SigmaLambdaList[[s]]
})
T_s_list <- fit_Tetris_train$T_s
mse_Tetris <- 1/(42 * sum(sapply(test_list, nrow)))*sum(
  sapply(1:6, function(s){
    Omega <- cbind(Phi, LambdaList[[s]])
    scores <- test_list_scaled[[s]] %*% solve(Psi[[s]]) %*% Omega %*% mnormt::pd.solve(signi

    norm(test_list_scaled[[s]] - scores %*% t(Omega), "F")^2
  })
)
```

We display the MSE of each model.

```r
print(paste0("Stack FA: ", mse_stackFA %>% round(3)))
print(paste0("Ind FA: ", mse_IndFA %>% round(3)))
print(paste0("PFA: ", mse_PFA %>% round(3)))
print(paste0("MOM-SS: ", mse_MOMSS %>% round(3)))
print(paste0("SUFA: ", mse_SUFA %>% round(3)))
print(paste0("BMSFA: ", mse_BMSFA %>% round(3)))
print(paste0("Tetris: ", mse_Tetris %>% round(3)))
```

```
[1] "Stack FA: 0.497"
[1] "Ind FA: 0.477"
[1] "PFA: 0.681"
[1] "MOM-SS: 0.468"
[1] "SUFA: 0.431"
[1] "BMSFA: 0.448"
[1] "Tetris: 0.281"
```

3 Case study: gene expression data

Similarly, we load utility package `tidyverse` to help with data manipulation and visualization.

```
library(tidyverse)
```

In this demonstration, we use the curate Ovarian data (Ganzfried et al. 2013) to demonstrate (1) the common gene co-expression network drawn by Σ_Φ and (2) the fit of the models via calculating MSE. This data contains the gene expression and clinical outcomes of 2970 patients collected from 23 studies. Different studies have different sequencing platforms, sample sizes, stage/subtype of the tumor, survival and censoring information.

3.1 Loading and previewing the data

We load the `curatedOvarianData` package to get the data. Description of each study can be found with `data(package="curatedOvarianData")`.

```
library(curatedOvarianData)
```

```
#data(package="curatedOvarianData")
data(GSE13876_eset)
data(GSE26712_eset)
data(GSE9891_eset)
data(PMID17290060_eset)
```

The four datasets are of similar sizes. All of them have the majority of patients in the late stage of the cancer, and histological subtypes observed in the tissue samples are mostly serous carcinoma. The datasets differs in respect sequencing platforms, which are Operonv3two-color, AffymetrixHG-U133A, AffymetrixHG-U133Plus2 and AffymetrixHG-U133A.

3.2 Data pre-processing

First we find intersection of genes that are exist in all four studies. We use `featureNames` to extract the gene names and `exprs` to extract the data matrices. More operations of the ExpressionSets can be found in (Falcon, Morgan, and Gentleman 2007).

```
inter_genes <- Reduce(intersect, list(featureNames(GSE13876_eset),
                                      featureNames(GSE26712_eset),
                                      featureNames(GSE9891_eset),
                                      featureNames(PMID17290060_eset)))

GSE13876_eset <- GSE13876_eset[inter_genes,]
GSE26712_eset <- GSE26712_eset[inter_genes,]
GSE9891_eset <- GSE9891_eset[inter_genes,]
PMID17290060_eset <- PMID17290060_eset[inter_genes,]

study1 <- t(exprs(GSE13876_eset))
study2 <- t(exprs(GSE26712_eset))
study3 <- t(exprs(GSE9891_eset))
study4 <- t(exprs(PMID17290060_eset))
```

We then filter the genes so that only high variance genes are kept for later analysis.

```
# calculate Coefficient of Variation of each gene
cv1 <- apply(study1, 2, sd) / apply(study1, 2, mean)
cv2 <- apply(study2, 2, sd) / apply(study2, 2, mean)
cv3 <- apply(study3, 2, sd) / apply(study3, 2, mean)
cv4 <- apply(study4, 2, sd) / apply(study4, 2, mean)
cv_matrix <- rbind(cv1, cv2, cv3, cv4)

# Find genes with CV >= threshold in at least one study
threshold <- 0.16
genes_to_keep <- apply(cv_matrix, 2, function(cv) any(cv >= threshold))
sum(genes_to_keep)
```

```
[1] 1060
```

```
# Filtered
study1 <- GSE13876_eset[genes_to_keep,]
study2 <- GSE26712_eset[genes_to_keep,]
study3 <- GSE9891_eset[genes_to_keep,]
study4 <- PMID17290060_eset[genes_to_keep,]
```

Next, we log-transform the data (for a better normality) and store them in 4 $N_s \times P$ matrices.

```
df1 <- study1 %>% exprs() %>% t() %>%
  log() %>% as.data.frame()
df2 <- study2 %>% exprs() %>% t() %>%
  log() %>% as.data.frame()
df3 <- study3 %>% exprs() %>% t() %>%
  log()  %>% as.data.frame()
df4 <- study4 %>% exprs() %>% t() %>%
  log()  %>% as.data.frame()
list_gene <- list(df1, df2, df3, df4)
saveRDS(list(df1, df2, df3, df4), "./Data/CuratedOvarian_processed.rds")
```

Now we can see the dimensions of each study:

```
sapply(list_gene, dim)
```

```
     [,1] [,2] [,3] [,4]
[1,]  157  195  285  117
[2,] 1060 1060 1060 1060
```

The ultimate data for analysis has $N_s = (157, 195, 285, 117)$ for $s = 1, 2, 3, 4$, and $P = 1060$.

We scale the data so that we focus on the correlations between the genes. When the data are scaled, the variance of the genes are 1 and the off-diagonal values scales up, so does the estimated covariances matrices, which facilitates a more interpretable network analysis.

```
Y_list <- readRDS("./Data/CuratedOvarian_processed.rds")
Y_list_scaled <- lapply(
 Y_list, function(x) scale(x, center = TRUE, scale = TRUE)
)
Y_mat_scaled <- Y_list_scaled %>% do.call(rbind, .) %>% as.matrix()
```

3.3 Fitting the models

For models fitted with `MSFA::sp_fa`, the function provides `scaling` and `centering` arguments. Therefore, we do not need to use the scaled data before fitting (here I use Mavis's version of MSFA).

Stack FA and Ind FA:

```
Y_mat =  Y_list %>% do.call(rbind, .) %>% as.matrix()
fit_stackFA <- MSFA::sp_fa(Y_mat, k = 6, scaling = FALSE, centering = TRUE,
                                control = list(nrun = 10000, burn = 8000))

fit_indFA <-
      lapply(1:6, function(s){
        j_s = c(8, 8, 8, 8, 8, 8)
        MSFA::sp_fa(Y_list[[s]], k = j_s[s], scaling = FALSE, centering = TRUE,
                    control = list(nrun = 10000, burn = 8000))
      })
```

MOM-SS:

```
# Construct the membership matrix
N_s <- sapply(Y_list, nrow)
M_list <- list()
   for(s in 1:4){
     M_list[[s]] <- matrix(1, nrow = N_s[s], ncol = 1)
   }
M <- as.matrix(bdiag(M_list))

fit_MOMSS <- BFR.BE::BFR.BE.EM.CV(x = Y_mat, v = NULL, b = M, q = 20, scaling = TRUE)
```

PFA (we don't recommend running it. It takes 4 days and more.)

```
N_s <- sapply(Y_list, nrow)
fit_PFA <- PFA(Y=t(Y_mat_scaled),
                        latentdim = 20,
                       grpind = rep(1:4,
                                    times = N_s),
               Thin = 5,
               Cutoff = 0.001,
               Total_itr = 10000, burn = 8000)
```

SUFA (takes about 10 hours):

```
fit_SUFA <- SUFA::fit_SUFA(Y_list_scaled, qmax=20,nrun = 10000)
```

BMSFA:

```
fit_BMSFA <- MSFA::sp_msfa(Y_list, k = 20, j_s = c(4, 4, 4, 4),
                           outputlevel = 1, scaling = TRUE, centering = TRUE,
                           control = list(nrun = 10000, burn = 8000))
```

Tetris (we don't recommend running it. It could take more than 10 days.):

```
set_alpha <- ceiling(1.25*4)
fit_Tetris <- tetris(Y_list_scaled, alpha=set_alpha, beta=1, nprint = 200, nrun=10000, burn=8
print("Start to choose big_T. It might take a long time.")
big_T <- choose.A(fit_Tetris, alpha_IBP=set_alpha, S=4)
run_fixed <- tetris(Y_list_scaled, alpha=set_alpha, beta=1,
                    fixed=TRUE, A_fixed=big_T, nprint = 200, nrun=10000, burn=8000)
```

3.4 Post-processing

We use the `post_xxx()` functions defined in the chapter of nutrition applications to post-process for point estimates of the factor loadings, common factors, and covariance matrices.

For Stack FA, Ind FA and BMSFA, we need to determine the number of common factors K and the number of latent dimensions J_s for each study after the post-processing, and we need to re-run those models.

Here we start with Stack FA, Ind FA and BMSFA:

```
# Stack FA
res_stackFA = post_stackFA(fit_stackFA, S=4)
# Ind FA
res_IndFA = post_IndFA(fit_indFA)
# BMSFA
res_BMSFA = post_BMSFA(fit_BMSFA)
```

```
# Eigenvalue decomposition
fun_eigen <- function(Sig_mean) {
  val_eigen <- eigen(Sig_mean)$values
  prop_var <- val_eigen/sum(val_eigen)
  choose_K <- length(which(prop_var > 0.05))
  return(choose_K)
}

SigmaPhi_StackFA <- res_stackFA$SigmaPhi
K_StackFA <- fun_eigen(SigmaPhi_StackFA)
```

```r
SigmaLambda_IndFA <- res_IndFA$SigmaLambda
Js_IndFA <- lapply(SigmaLambda_IndFA, fun_eigen)

SigmaPhi_BMSFA <- res_BMSFA$SigmaPhi
K_BMSFA <- fun_eigen(SigmaPhi_BMSFA)
SigmaLambda_BMSFA <- res_BMSFA$SigmaLambda
Js_BMSFA <- lapply(SigmaLambda_BMSFA, fun_eigen)

# We print the results:
print(paste0("Stack FA: K = ", K_StackFA))
```

```
[1] "Stack FA: K = 2"
```

```r
print(paste0("Ind FA: J_s = ", Js_IndFA))
```

```
[1] "Ind FA: J_s = 4" "Ind FA: J_s = 7" "Ind FA: J_s = 6" "Ind FA: J_s = 6"
```

```r
print(paste0("BMSFA: K = ", K_BMSFA))
```

```
[1] "BMSFA: K = 6"
```

```r
print(paste0("BMSFA: J_s = ", Js_BMSFA))
```

```
[1] "BMSFA: J_s = 4" "BMSFA: J_s = 4" "BMSFA: J_s = 4" "BMSFA: J_s = 4"
```

We then fit the models again with the determined number of factors and latent dimensions.

```r
# Stack FA
fit_stackFA2 <- MSFA::sp_fa(Y_mat, k = K_StackFA, scaling = FALSE, centering = TRUE,
                                control = list(nrun = 10000, burn = 8000))
# Ind FA
fit_indFA2 <- lapply(1:4, function(s){
  j_s = Js_IndFA[[s]]
  MSFA::sp_fa(Y_list[[s]], k = j_s, scaling = FALSE, centering = TRUE,
              control = list(nrun = 10000, burn = 8000))
})
# BMSFA
fit_BMSFA2 <- MSFA::sp_msfa(Y_list, k = K_BMSFA, j_s = Js_BMSFA,
```

```
                              outputlevel = 1, scaling = TRUE, centering = TRUE,
                              control = list(nrun = 10000, burn = 8000))

# Again we post-process the MCMC chains.
res_stackFA2 <- post_stackFA(fit_stackFA2, S=4)
res_IndFA2 <- post_IndFA(fit_indFA2)
res_BMSFA2 <- post_BMSFA(fit_BMSFA2)
```

For the rest of the methods, we apply the same post-processing functions as in the nutrition applications chapter.

```
# PFA
res_PFA <- post_PFA(fit_PFA)
# MOM-SS
res_MOMSS <- post_MOMSS(fit_MOMSS)
# SUFA
res_SUFA <- post_SUFA(fit_SUFA)
# Tetris
#res_Tetris <- post_Tetris(fit_Tetris)
```

3.5 Visualization

We can visualize the common gene co-expression network using the estimated common covariance matrices. We use Gephi to visualize the networks. The `SigmaPhi` matrix is the common covariance matrix, and `SigmaLambda` is the study-specific covariance matrix.

Before that, we need to filter the genes so that only genes with high correlations are kept (leaving around 200 genes in the plot). We set the threshold for each model as follows:

- Stack FA: 0.85
- MOM-SS: 0.95
- BMSFA: 0.5
- SUFA: 0.28
- PFA: 0.55

```
# Filtering genes for visualization in Gephi
genenames <- colnames(list_gene[[1]])

# StackFA
SigmaPhi_StackFA_curated <- res_stackFA2$SigmaPhi
colnames(SigmaPhi_StackFA_curated) <- rownames(SigmaPhi_StackFA_curated) <- genenames
```

```r
diag(SigmaPhi_StackFA_curated) <- NA
above_thresh <- SigmaPhi_StackFA_curated > 0.85
keep_genes <- apply(above_thresh, 1, function(x) any(x, na.rm = TRUE))
sum(keep_genes)
```

```
[1] 214
```

```r
SigmaPhi_StackFA_curated[abs(SigmaPhi_StackFA_curated) < 0.85] <- 0
SigmaPhi_StackFA_curated_filtered <- SigmaPhi_StackFA_curated[keep_genes, keep_genes]
#write.csv(SigmaPhi_StackFA_curated_filtered, "SigmaPhi_StackFA_curated_filtered.csv")

# MOM-SS
SigmaPhiMOMSS <- res_MOMSS$SigmaPhi
colnames(SigmaPhiMOMSS) <- rownames(SigmaPhiMOMSS) <- genenames
diag(SigmaPhiMOMSS) <- NA
above_thresh <- abs(SigmaPhiMOMSS) > 0.95
keep_genes <- apply(above_thresh, 1, function(x) any(x, na.rm = TRUE))
sum(keep_genes)
```

```
[1] 223
```

```r
SigmaPhiMOMSS[abs(SigmaPhiMOMSS) < 0.95] <- 0
SigmaPhiMOMSS_filtered <- SigmaPhiMOMSS[keep_genes, keep_genes]
#write.csv(SigmaPhiMOMSS_filtered, "SigmaPhiMOMSS_filtered.csv")

# BMSFA
SigmaPhiBMSFA <- res_BMSFA2$SigmaPhi
colnames(SigmaPhiBMSFA) <- rownames(SigmaPhiBMSFA) <- genenames
diag(SigmaPhiBMSFA) <- NA
above_thresh <- abs(SigmaPhiBMSFA)  > 0.5
keep_genes <- apply(above_thresh, 1, function(x) any(x, na.rm = TRUE))
sum(keep_genes)
```

```
[1] 192
```

```r
SigmaPhiBMSFA[abs(SigmaPhiBMSFA) < 0.5] <- 0
SigmaPhiBMSFA_filtered <- SigmaPhiBMSFA[keep_genes, keep_genes]
#write.csv(SigmaPhiBMSFA_filtered, "SigmaPhiBMSFA_filtered.csv")
```

```r
# SUFA
SigmaPhiSUFA <- res_SUFA$SigmaPhi
colnames(SigmaPhiSUFA) <- rownames(SigmaPhiSUFA) <- genenames
diag(SigmaPhiSUFA) <- NA
above_thresh <- abs(SigmaPhiSUFA) > 0.28
keep_genes <- apply(above_thresh, 1, function(x) any(x, na.rm = TRUE))
sum(keep_genes)
```

```
[1] 191
```

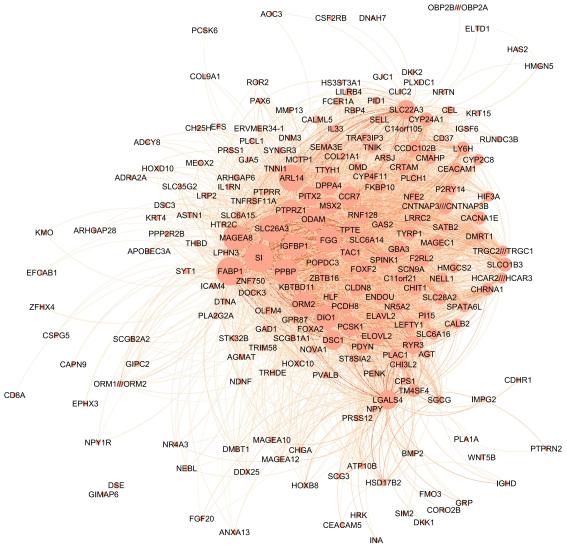
```r
SigmaPhiSUFA[abs(SigmaPhiSUFA) < 0.28] <- 0
SigmaPhiSUFA_filtered <- SigmaPhiSUFA[keep_genes, keep_genes]
#write.csv(SigmaPhiSUFA_filtered, "SigmaPhiSUFA_filtered.csv")

# PFA
SigmaPhiPFA <- res_PFA$SigmaPhi
colnames(SigmaPhiPFA) <- rownames(SigmaPhiPFA) <- genenames
diag(SigmaPhiPFA) <- NA
above_thresh <- abs(SigmaPhiPFA) > 0.55
keep_genes <- apply(above_thresh, 1, function(x) any(x, na.rm = TRUE))
sum(keep_genes)
```

```
[1] 203
```

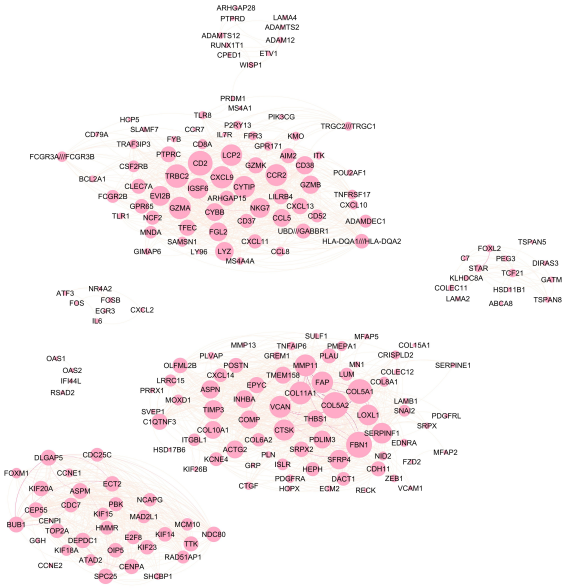
```r
SigmaPhiPFA[abs(SigmaPhiPFA) < 0.55] <- 0
SigmaPhiPFA_filtered <- SigmaPhiPFA[keep_genes, keep_genes]
#write.csv(SigmaPhiPFA_filtered, "SigmaPhiPFA_filtered.csv")
```

Attached are the final figures make with Gephi. The nodes are the genes and the edges are the correlations between the genes. The thickness of the edges indicates the strength of the correlation. The color of the nodes indicates the study they belong to. The size of the nodes indicates the number of connections they have.

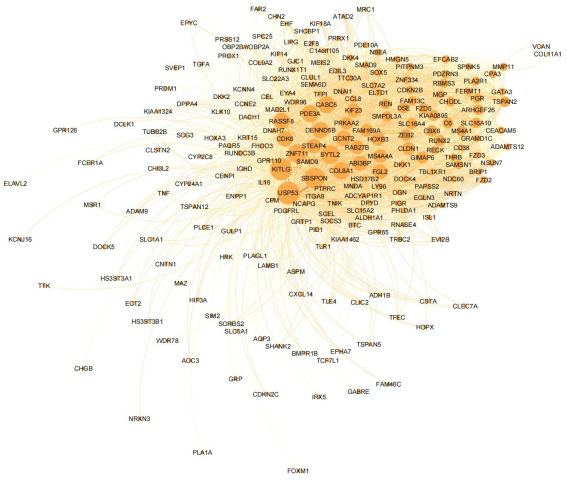
Stack FA:



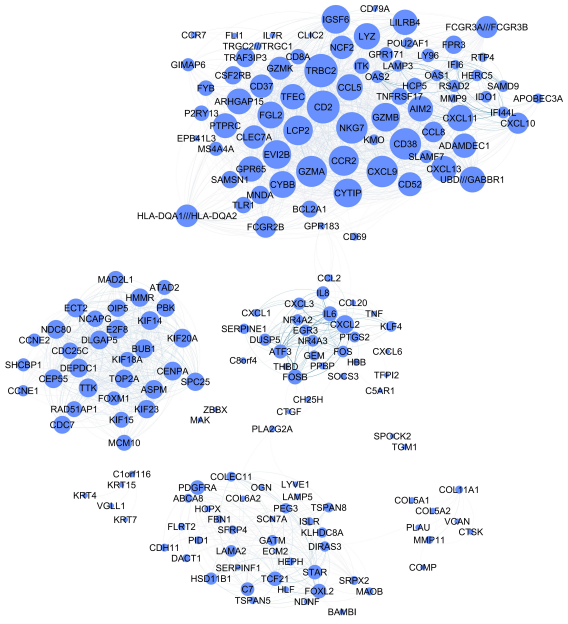
PFA:



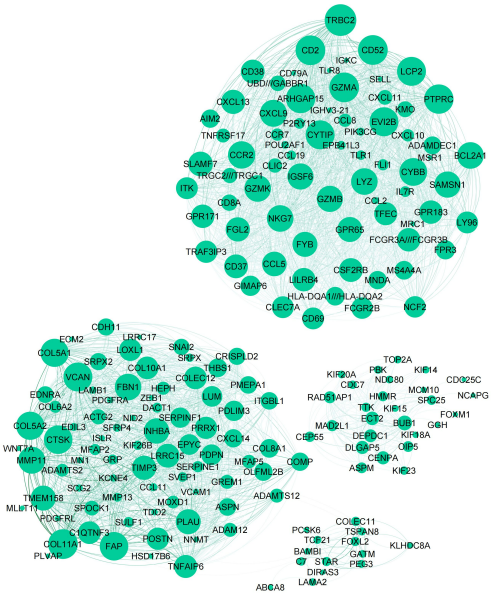
MOM-SS:



SUFA:



BMSFA:


TRBC2
CD2
CD52
IGKC
CD38
CD79A
TLR8
SIRL
LCP2
UBD
GABBR1
GZMA
CXCL13
ARHGAP15
CXCL11
PTPRC
CXCL9
IGHV3-21
KMO
AIM2
P2RY13
CCL6
EVI2B
TNFRSF17
CCR7
GYTIP
PIK3CG
CXCL10
POU2AF1
EPB41L3
ADAMDEC1
CCR2
CCL19
TLR1
MSR1
BCL2A1
SLAMF7
CLIC2
FLI1
TRGC2
TRGC1
IGSF6
LYZ
CYBB
ITK
GZMK
IL7R
SAMSN1
CD8A
GZMB
CCL2
GPR171
NKG7
TFEC
GPR183
LY96
FGL2
GPR65
MRC1
FCGR3A
FCGR3B
FYB
FPR3
TRAF3IP3
CD37
CCL5
CSF2RB
MS4A4A
GIMAP6
MNDA
LILRB4
HLA-DQA1
HLA-DQA2
CLEC7A
FCGR2B
NCF2
CD69
CDH11
EOM2
LRRC17
COL5A1
LOXL1
SNAI2
SRPX
CRISPLD2
SFRP2
TMBS1
COL10A1
VCAN
COLEC12
PMEPA1
LAMB1
FBN1
HEPH
EDNRA
PDGFRA
ZBB1
LUM
ITGBL1
COL8A2
DADT1
PDLIM3
ACTG2
NID2
SERPINE1
COL5A2
EMILIN3
SFRP4
INHBA
PRRX1
CTSK
ISLR
KIF26B
EPYC
CXCL14
WNT2A
MFAP2
LRRC15
PDPN
COL6A1
MMP11
VIM
GBP
TIMP3
SERPINE1
MFAP5
COMP
ADAMTS2
OLFML2B
KCNE4
SVEP1
GREM1
SOX2
COL11
VCAM1
TMEM158
MMP13
MXD1
ADAMTS12
MLLT11
SPOCK1
ASPN
TGFB2
PDGFRL
SULF1
PLAU
C1QTNF3
ADAM12
COL11A1
FAP
POSTN
NNMT
PLVAP
HSD17B6
TNFAIP6
KIF20A
PBK
KIF14
CDC7
NDC80
CDC25C
MCM10
HNRNPR
SPC25
NCAPG
RAD54AP1
TTK
KIF15
FOXM1
ECT2
BUB1
GGH
MAD2L1
CEP55
DEPDC1
KIF18A
DLGAP5
ORC6
CENPA
ASPM
KIF23
TOP2A
COL3C11
TSPAN8
PCSK6
TCF21
FOXL2
KLHDC8A
BAMBI
GATM
C7
STAR
PEG3
DIRAS3
LAMA2
ABCA8

References

Avalos-Pacheco, Alejandra, David Rossell, and Richard S. Savage. 2022. "Heterogeneous Large Datasets Integration Using Bayesian Factor Regression." *Bayesian Analysis* 17 (1): 33–66.

Chandra, Noirrit Kiran, David B Dunson, and Jason Xu. 2024. "Inferring Covariance Structure from Multiple Data Sources via Subspace Factor Analysis." *Journal of the American Statistical Association*, no. just-accepted: 1–25.

De Vito, Roberta, Ruggero Bellio, Lorenzo Trippa, and Giovanni Parmigiani. 2021. "Bayesian Multistudy Factor Analysis for High-Throughput Biological Data." *The Annals of Applied Statistics* 15 (4): 1723–41.

De Vito, Roberta, Briana Stephenson, Daniela Sotres-Alvarez, Anna-Maria Siega-Riz, Josiemer Mattei, Maria Parpinel, Brandilyn A Peters, et al. 2022. "Shared and Ethnic Background Site-Specific Dietary Patterns in the Hispanic Community Health Study/Study of Latinos (HCHS/SOL)." *medRxiv*, 2022–06.

Falcon, Seth, Martin Morgan, and Robert Gentleman. 2007. "An Introduction to Bioconductor's Expressionset Class."

Ganzfried, Benjamin Frederick, Markus Riester, Benjamin Haibe-Kains, Thomas Risch, Svitlana Tyekucheva, Ina Jazic, Xin Victoria Wang, et al. 2013. "curatedOvarianData: Clinically Annotated Data for the Ovarian Cancer Transcriptome." *Database* 2013.

Grabski, Isabella N, Roberta De Vito, Lorenzo Trippa, and Giovanni Parmigiani. 2023. "Bayesian Combinatorial MultiStudy Factor Analysis." *The Annals of Applied Statistics* 17 (3): 2212.

Roy, Arkaprava, Isaac Lavine, Amy H. Herring, and David B. Dunson. 2021. "Perturbed Factor Analysis: Accounting for Group Differences in Exposure Profiles." *The Annals of Applied Statistics* 15 (3): 1386–1404.