A Tutorial for Bayesian Integrative Factor Models

Mavis Liang

2025-01-10

Table of contents

Welcome

This is the tutorial to guide statisticians to use Baysian integrative factor models.

```
1 + 1
```

```
[1] 2
```

1 Preliminary

1.1 Factor models and multi-study setting

This chapter introduce the theory behind factor models

See Knuth (1984) for additional discussion of literate programming.

```
1 + 1
```

```
[1] 2
```

2 Quick start

Step 1: Prepare the package and data

A small simulated data can be downloaded here:

RDS format

Step 2: Run BMSFA

Step 3: Post-processing

Step 4: Visualization

```
1 + 1
```

```
[1] 2
```

3 Package Installation

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

3.1 Stack FA, Ind FA, and BMSFA

3.2 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")
```

3.3 MOM-SS

3.4 SUFA

3.5 Tetris

3.6 Other utility packages

```
library(tidyverse)
```

4 Case study: nutrition data

4.1 Loading and previewing the data

The data used in this section is from... This data is not publicly available. Please contact the authors of the original study for access.

```
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.

4.2 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.

```
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.

```
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()
```

4.3 Model fitting

We recommend that we run the following code chunk 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 finished within half an hour. We recommend at least 5GB of memory for running the models and post-processing.

For each model, we over-specify the numbers of factors to .. [to be written].

```
# 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 = 800))
# 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,
                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. Hyperparamters $\alpha_{\mathcal{T}}$ 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)
```

4.4 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))
}
```

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])
              ))
}
```

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))
}
```

For PFA, the post-processing is a little bit tricky. According to the original paper, the common loading matrix Φ can be just the average of the estimated loadings from the posterior samples. However, we still apply OP as in BMSFA to get the common loadings, and the loadings in PFA should be the product of the estimated loadings and the square root of the variance of the factors. Therefore, we carry out multiplication for Φ for each posterior sample, and then OP the sequence. We also carry out multiplication for Q_s, Σ_Φ, Σ_{marginal}, Σ_{Λ_s}, and Ψ as in the definition of PFA, and then take the average of the results.

After that, the number of factors is determined by counting the number of columns in Φ that have all loadings less than 10^{-3}.

```
post_PFA <- function(fit){
  p <- nrow(fit$Loading[[1]])
  k <- ncol(fit$Loading[[1]])
  npost <- length(fit$Loading)
  Q_list <- fit$Pertmat
  S <- dim(Q_list[[1]])[2]
  posteriorPhis <- array(0, dim = c(p, k, npost))

  #---estimated common loadings and shared variance---
  # Element-wise multiplication
  for(i in 1:npost){
    posteriorPhis[,,i] <- fit$Loading[[i]] %*% diag(fit$Latentsigma[[i]])
  }
  # Varimax for common loadings
  est_Phi <- MSFA::sp_OP(posteriorPhis, itermax = 10, trace = FALSE)$Phi

  # Estimated shared covariance, study-specific covariance matrix, and SigmaMarginal
  sharevar <- list()
  est_SigmaLambdaList <- list()
  est_SigmaMarginal <- list()
  est_Psi <- list()
  for(s in 1:S){ # Loop over each study
  post_SigmaLambda_s <- list()
  post_SigmaMarginal_s <- list()
  Psi <- list()
```

```
  for(i in 1:npost){ # Loop over each posterior sample
    sharevar[[i]] <- fit$Loading[[i]]%*%diag(fit$Latentsigma[[i]]^2)%*%t(fit$Loading[[i]]) +
      diag(fit$Errorsigma[[i]]^2) # Get the shared variance
    Q_temp_inv <- solve(
      matrix(Q_list[[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)/length(post_SigmaMarginal_s)
  est_SigmaLambdaList[[s]] <- Reduce('+', post_SigmaLambda_s)/length(post_SigmaLambda_s)
  }
  est_Psi <- Reduce('+', Psi)/length(Psi)
  est_SigmaPhi <- Reduce('+', sharevar)/length(sharevar)
  est_Q <- Reduce('+', Q_list)/length(Q_list)
  est_Q_list <- lapply(1:S, function(s) matrix(est_Q[, s], p, p))

  # Return the results
  return(list(Phi = est_Phi, SigmaPhi = est_SigmaPhi, Psi = est_Psi, Q = est_Q_list,
              SigmaLambdaList = est_SigmaLambdaList,
              SigmaMarginal = est_SigmaMarginal))
}
```

```
# 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)}
    ))
  )
}
Phi_PFA <- readRDS("Data/Rnutrition_PFA.rds")$Phi # Load post_PFA(fit_PFA) output
K_PFA <- fun_neibour(Phi_PFA)
K_PFA
```

```
[1] 6
```

4.5 Visualization

4.6 Mean squared error (MSE)

5 Summary

In summary, this book has no content whatsoever.

```
1 + 1
```

```
[1] 2
```

References

Knuth, Donald E. 1984. “Literate Programming.” *Comput. J.* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.