

# An Introduction to **FAD** for Exploratory Factor Analysis with High-dimensional Gaussian Data

Somak Dutta, Fan Dai, and Ranjan Maitra

Department of Statistics, Iowa State University  
somakd@iastate.edu, fd43@iastate.edu, maitra@iastate.edu

July 27, 2019

The **fad** package, which is developed via a matrix-free likelihood method for high-dimensional maximum-likelihood factor analysis on Gaussian data, provides a highly sufficient and accurate way to estimate factor model in R compared with the traditional EM algorithm. This vignette gives a brief introduction to the functions included in **fad** with simulated examples.

## 1 Factor Model

Suppose  $\mathbf{Y}_1, \dots, \mathbf{Y}_n$  are *i.i.d.*  $p$ -dimensional random vectors following the multivariate normal distribution  $\mathcal{N}_p(\mu, \Sigma)$ , a factor model decomposes the covariance into  $\Sigma = \Lambda\Lambda^\top + \Psi$ , where  $\Lambda$  is a  $p \times q$  matrix with rank  $q$  that describes the amount of variance shared among the  $p$  coordinates, and  $\Psi$  is a diagonal matrix with positive diagonal entries representing the unique variance specific to each coordinate.

Maximum likelihood (ML) estimation for covariance are derived from the log-likelihood after profiling out  $\mu$ ,

$$\ell(\Lambda, \Psi) = -0.5 \times n \{p \log(2\pi) + \log \det \Sigma + \text{Tr}\{\Sigma^{-1} \mathbf{S}\}\} \quad (1)$$

Where  $\mathbf{Y}$  denotes the  $n \times p$  data matrix and  $\bar{\mathbf{Y}}$  is sample means.  $\mathbf{S} = n^{-1}(\mathbf{Y} - \mathbf{1}\bar{\mathbf{Y}}^\top)^\top(\mathbf{Y} - \mathbf{1}\bar{\mathbf{Y}}^\top)$  represents the sample covariance matrix.

## 2 Main function in FAD

Users can call the main **fad** function to estimate a factor model,

```
> library(fad)
```

All the available options in **fad** are shown below, followed with detailed descriptions,

```
> args(fad)
```

```
function (x, factors, data = NULL, covmat = NULL, n.obs = NA,
  subset, na.action, start = NULL, scores = c("none", "regression",
    "Bartlett"), rotation = "varimax", control = NULL, lower = 0.005,
  ...)
NULL
```

**x:** A formula or a numeric matrix or an object that can be coerced to a numeric matrix.

**factor:** The number of factors to be fitted.

**data:** An optional data frame (or similar: see `model.frame`), used only if **x** is a formula. By default the variables are taken from `environment(formula)`.

**covmat:** A covariance matrix, or a covariance list as returned by `cov.wt`. Of course, correlation matrices are covariance matrices.

**n.obs:** The number of observations, used if `covmat` is a covariance matrix.

**subset:** A specification of the cases to be used, if `x` is used as a matrix or formula.

**na.action:** The `na.action` to be used if `x` is used as a formula.

**start:** NULL or a matrix of starting values, each column giving an initial set of uniquenesses.

**score:** Type of scores to produce, if any. The default is none, "regression" gives Thompson's scores, "Bartlett" given Bartlett's weighted least-squares scores. Partial matching allows these names to be abbreviated. Also note that some of the scores-types are not applicable when  $p > n$ .

**rotation:** character. "none" or the name of a function to be used to rotate the factors: it will be called with first argument the loadings matrix, and should return a list with component loadings giving the rotated loadings, or just the rotated loadings. The options included in the package are: `varimax`, `promax`, `quartimax`, `equamax` with default to `varimax`.

**control:** A list of control values:

`nstart`

The number of starting values to be tried if `start = NULL`. Default 1.

`trace`

logical. Output tracing information? Default FALSE.

`opt`

A list of control values to be passed to `optim`'s control argument.

`rotate`

a list of additional arguments for the rotation function.

**lower:** The lower bound for uniquenesses during optimization. Should be  $> 0$ . Default 0.005.

**...**: Components of control can also be supplied as named arguments to `fad`.

In `fad`, only initialization of  $\Psi$  is required and is specified through the **start** term which defaults to one minus the diagonal entries of  $\tilde{\Lambda}\tilde{\Lambda}^\top$  where  $\tilde{\Lambda}$  is computed as the first  $q$  principal components (PCs) for the scaled data matrix.

`fad` produces an object with the most critical components explained below,

**loadings:** A matrix of loadings, one column for each factor. The factors are ordered in decreasing order of sums of squares of loadings, and given the sign that will make the sum of the loadings positive. This is of class "loadings"

**uniquenesses:** The uniquenesses computed.

**criteria:** The results of the optimization: the value of the criterion (a linear function of the negative log-likelihood) and information on the iterations used.

**factors:** The argument factors.

**loglik:** The maximum log-likelihood.

**BIC:** The Bayesian Information Criteria.

Note that the ML estimates for mean and standard deviations can be easily computed from sample mean and sample standard deviations.

Here are examples for fitting a simulated  $n \times p$  data matrix and a covariance matrix with  $(n, p) = (50, 100)$  with  $q = 3$ . The diagonal entries of  $\Psi$  were randomly simulated from  $\mathcal{U}(0.2, 0.8)$  and the elements of  $\Lambda$  and  $\mu$  were from  $\mathcal{N}(0, 1)$ . We use BIC to select the optimal factors from the set of  $\{1, 2, 3, \dots, 10\}$ .

```
> set.seed(1)
```

```

> n <- 50 # number of observations
> p <- 100 # size of coordinates
> q <- 3 # true number of factors
> mu = rnorm(p)
> L = matrix(rnorm(p*q),p,q)
> D = runif(p,0.2,0.8)
> X = matrix(rnorm(n*q),n,q)
> X = tcrossprod(X,L) + matrix(rnorm(n*p),n,p)%*%diag(sqrt(D)) + rep(mu, each = n)
> BICs = rep(0,11) # store BIC values
> for(i in 1:11){
+   out = fad(x = X,factors = i-1)
+   BICs[i] = out$BIC
+ }
> BICs

[1] 5000.000000    -9.655628 -2267.381051 -5220.523088 -5013.200392 -4797.035814
[7] -4553.623101 -4307.348905 -4070.173885 -3828.008845 -3583.559953

> ind = which.min(BICs)-1# obtain the optimal factor favored by BIC
> ind

[1] 3

> BICs = rep(0,11) # store BIC values
> for(i in 1:11){
+   out = fad(covmat = cov(X), n.obs = n, factors = i-1)
+   BICs[i] = out$BIC
+ }
> BICs

[1] 5000.00000    -60.16237 -2317.88797 -5271.02988 -5063.70751 -4847.54243 -4604.12984
[8] -4357.85544 -4120.68097 -3878.51489 -3634.06701

> ind = which.min(BICs)-1
> ind

[1] 3

```

BIC selected the correct factor of 3 and we can then fit with the optimal factor model as

```

> out1 = fad(x = X,factors = 3)
> # or
> out2 = fad(covmat = cov(X), n.obs = n, factors = 3)

```