How to use Fast Step Graph

Juan G. Colonna Marcelo Ruiz

Contents

To install the last version of this package directly from GitHub run:

```r
library(devtools)
# use "quiet = FALSE" if you want to see the outputs of this command
devtools::install_github("juancolonna/FastStepGraph", quiet = TRUE, force = TRUE)
```

Then, load with:

```r
library(FastStepGraph)
library(MASS)     # mvrnorm

# If you directly cloned the github repository,
# then you should uncomment these lines to load the functions:
# source('FastStepGraph.R')
# source('SigmaAR.R')
```

Simulate Gaussian Data with an Autoregressive (AR) Model:

```r
set.seed(1234567)
phi <- 0.4
p <- 100  # number of variables (dimension)
n <- 50 # number of samples

Sigma <- SigmaAR(p, phi)
Omega <- solve(Sigma)
Omega[abs(Omega) < 1e-5] <- 0

# Generate Data from a Gaussian distribution
X <- list()
X <- MASS::mvrnorm(n, mu=rep(0,p), Sigma)
X <- scale(X) # data normalization to zero mean and unit variance
```

Afterwards, fit the Omega matrix by calling the Fast Step Graph function, like:

```r
t0 <- Sys.time() # INITIAL TIME
G <- FastStepGraph(X, alpha_f = 0.22, alpha_b = 0.14)
difftime(Sys.time(), t0, units = "secs")
#> Time difference of 0.3409333 secs
# print(G$Omega)
```

If the `nei.max` argument is omitted, it will be 5. If you don't know the `alpha_f` and `alpha_b` values, the use cross-validation. To find the optimal $\alpha_{\mathbf{f}}$ and $\alpha_{\mathbf{b}}$ parameters for the previously generated $\mathbf{X}$ data, we can perform a cross-validation on a combination grid as follows:

```
t0 <- Sys.time() # INITIAL TIME
res <- cv.FastStepGraph(X, data_shuffle = FALSE)
difftime(Sys.time(), t0, units = "secs")
#> Time difference of 19.69518 secs

print(res$alpha_f_opt)
#> [1] 0.6935484
print(res$alpha_b_opt)
#> [1] 0.3467742
# print(res$Omega)
```

The arguments `n_folds = 5`, `alpha_f_min = 0.1`, `alpha_f_max = 0.9`, `n_alpha = 32` (size of the grid search) and `nei.max = 5`, have defaults values and can be omitted. Note that, `cv.FastStepGraph(X)` is not an exhaustive grid search over $\alpha_{\mathbf{f}}$ and $\alpha_{\mathbf{b}}$. This is a heuristic that always sets $\alpha_{\mathbf{b}} = \frac{\alpha_{\mathbf{f}}}{2}$. It is recommended to shuffle the rows of `X` before running cross-validation. The default value is `data_shuffle = TRUE`, but if you want to disable row shuffle, set it to `data_shuffle = FALSE`.

To increase time performance, you can run `cv.FastStepGraph(X, parallel = TRUE)` in parallel. Before, you'll need to install and register a parallel backend. Since Windows does not support forking, the same backend that works in a Linux or OS X environment will not work in Windows. To run on a Linux system the **doParallel** dependency must be installed `install.packages("doParallel")`. To run on a Windows system, **doSNOW** is used `install.packages("doSNOW")`. These parallel packages will also require the following dependencies: **foreach**, **iterators** and **parallel**. Make sure you satisfy them. Then, call the method setting the parameter **parallel = TRUE**, as follows:

```
t0 <- Sys.time() # INITIAL TIME
# use 'n_cores = NULL' to set the maximum number of cores on your machine minus one
res <- cv.FastStepGraph(X, parallel = TRUE, n_cores = 2)
difftime(Sys.time(), t0, units = "secs")
#> Time difference of 10.60106 secs

print(res$alpha_f_opt)
#> [1] 0.4870968
print(res$alpha_b_opt)
#> [1] 0.04870968
# print(res$Omega)
```