How to use Fast Step Graph

Juan G. Colonna[*]   Marcelo Ruiz[†]

To install the last version of this package directly from GitHub uncomment and 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 it:
library(FastStepGraph)
```

Simulate Gaussian Data with an Autoregressive (AR) Model:

```r
set.seed(1234567)
phi <- 0.4
p <- 50  # number of variables (dimension)
n <- 30 # number of samples

# Generate Data from a Gaussian distribution
data <- FastStepGraph::SigmaAR(n, p, phi)
X <- scale(data$X) # standardizing variables
```

To fit the Omega matrix with `FastStepGraph()` function you have to know the optimal values of $\alpha_{\mathbf{f}}$ and $\alpha_{\mathbf{b}}$. If you don't know these values, try to find them using cross-validation as follows:

```r
t0 <- Sys.time() # INITIAL TIME
res <- FastStepGraph::cv.FastStepGraph(X,
                                       alpha_f_min=0.5,
                                       alpha_f_max=0.8,
                                       data_shuffle = TRUE)
difftime(Sys.time(), t0, units = "secs")
#> Time difference of 0.4920323 secs
# print(res$alpha_f_opt)
# print(res$alpha_b_opt)
```

If your input variables are non-standardized (with zero mean and unit variance), we recommend that you set `data_scale=TURE`. Subsequently, calculate the Omega matrix by calling the `FastStepGraph()` function passing the optimal parameters $\alpha_{\mathbf{f}}$ and $\alpha_{\mathbf{b}}$ found by cross-validation to fit the final model:

```r
t0 <- Sys.time() # INITIAL TIME
G <- FastStepGraph::FastStepGraph(X, alpha_f = res$alpha_f_opt, alpha_b = res$alpha_b_opt)
difftime(Sys.time(), t0, units = "secs")
#> Time difference of 0.003271341 secs
# print(G$Omega)
```

You can also perform these two steps, the cross-validation to obtain the ideal parameters and return the fitted model, in a single line by setting the `return_model=TRUE` option as follows:

[*]Institute of Computing. Federal University of Amazonas. Brasil. juancolonna@icomp.ufam.edu.br
[†]Mathematics Department. National University of Río Cuarto. Argentina. mruiz@exa.unrc.edu.ar

```r
t0 <- Sys.time() # INITIAL TIME
res <- FastStepGraph::cv.FastStepGraph(X,
                                       alpha_f_min=0.5,
                                       alpha_f_max=0.8,
                                       return_model=TRUE,
                                       data_shuffle = TRUE)
difftime(Sys.time(), t0, units = "secs")
#> Time difference of 0.512183 secs
# print(res$alpha_f_opt)
# print(res$alpha_b_opt)
# 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 tests only a few $\alpha_{\mathbf{b}}$ values starting with the rule $\alpha_{\mathbf{b}} = \frac{\alpha_{\mathbf{f}}}{2}$. 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. To run on a Linux system the **doParallel** dependency must be installed `install.packages("doParallel")`. 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:

```r
t0 <- Sys.time() # INITIAL TIME
# use 'n_cores = NULL' to set the maximum number of cores minus one on your machine
res <- FastStepGraph::cv.FastStepGraph(X,
                                       alpha_f_min=0.5,
                                       alpha_f_max=0.8,
                                       return_model=TRUE,
                                       parallel = TRUE,
                                       n_cores = 2)
difftime(Sys.time(), t0, units = "secs")
# print(res$alpha_f_opt)
# print(res$alpha_b_opt)
# print(res$Omega)
```

Remember, you can set the `n_cores` parameter to a value equal to the number of cores you have, but be careful as this may overload your system. Setting it to `1` disables parallel processing, and setting it to a number greater than the number of available cores does not improve efficiency.