Instruction and an Example of BSS Algorithm

Setting Up

Before implementing the algorithm, we need the following set up:

- Install required packages and load packages.
- Call functions from .R file and .cpp files
  - "Functions_BSS.R"
  - "Functions_BSS.cpp"

```
install.packages("mvtnorm")
install.packages("lattice")
install.packages("igraph")
install.packages("pracma")
install.packages("Rcpp")
install.packages("RcppArmadillo")
```

```
library("mvtnorm")
library("lattice")
library("igraph")
```

```
##
## Attaching package: 'igraph'

## The following objects are masked from 'package:stats':
##
##     decompose, spectrum

## The following object is masked from 'package:base':
##
##     union
```

```
library("pracma")
library("Rcpp")
library("RcppArmadillo")
```

```
source("Functions_BSS.R")
#cpp code for block fused lasso and block lasso
sourceCpp("Functions_BSS.cpp")
```

Simulation Scenario Setting

In order to generating the VAR time series data, the following arguments are required:

- `T`: the number of time points
- `p`: the number of time series components
- `brk`: the true break points with $T+1$ as the last element
- `m0`: the true number of break points
- `q.t`: the true AR order
- `phi.full`: the true AR coefficient matrix

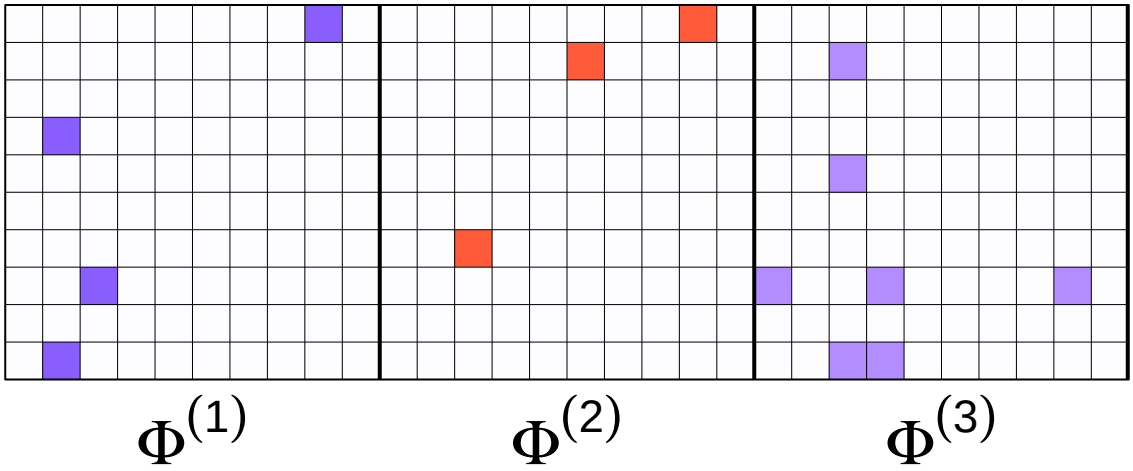
Here is an example setting of Scenario A.2 in the manuscript. In this setting, $T = 4,000$, $p = 10$, $q.t = 1$, $m0 = 2$, two break points $t_1 = \lfloor \frac{T}{3} \rfloor = 1333$, $t_2 = \lfloor \frac{2T}{3} \rfloor = 2666$ and the autoregressive coefficients are chosen to have the random structure and entries.

```
########## General Settings ######################
T <- (10^3*4); #number of time points
p <- 10; # number of time series components
brk <- c(floor(T/3),floor(2*T/3),T+1); # true break points with T+1 as the last element
m0 <- length(brk) -1; # number of break points
q.t <- 1; # the true AR order
m <- m0+1 #number of segments
sp_density <- c(0.05, 0.05, 0.05) #sparsity level (5%)
```

Data Generation

After setting the general arguments described above, we can now generate the time series data by using the function `var.sim.break()` as follows. Here, the covariance matrix of the noise process $\Sigma_\varepsilon = I_T$ (where I_T stands for the identity matrix).

```
#randomly generate transition matrix and timeseries data
try <- simu_var(method = 'sparse', nobs = T, k=p, sp_pattern= 'random', arlags=seq(1,q.t,1),
                brk=brk, sp_density = sp_density , seed = 1 )
#display the true AR coefficients matrice
print(plot.matrix(do.call("cbind",try$model_param), m ))
```




```
data <- try$series
data <- as.matrix(data)
```

Block Segmentation Scheme (BSS) Implementation

After data generataion, we now perform the block segmentation scheme (BSS) algorithm to detect the structural breaks in large scale high-dimensional non-stationary VAR models. The BSS algorithm mainly contains three steps:

- First step: initial break points selection by block fused lasso.
- Second step: local screening by minimizing a localized information criterion (LIC) to eliminate candidate break points that are located far from any true break points.
- Third step: exhaustive search by comparing SSE for each candidate point in a given cluster and keeping one break point in each cluster.

The proposed BSS algorithm in the manuscript is implemented by the function `bss()`. The only input argument is the time series data, with each column representing the time series component.

```
#run the bss method
temp <- bss(data)
```

```
## [1] "lambda.1.cv:"
##  [1] 1143.6067780  410.9909665  147.7024951   53.0815245   19.0765108
##  [6]    6.8557425    2.4638261    0.8854532    0.3182154    0.1143607
```

```
## [1] "lambda.2.cv:"
## [1] 0.239926296 0.023992630 0.002399263
## [1] "first.brk.points:"
##  [1] 1197 1323 1449 2394 2583 2709 2835 3024 3150 3465 3717
## [1] "selected lambda1:"
## [1] 19.07651
## [1] "selected lambda2:"
## [1] 0.002399263
## [1] "an:"
## [1] 63
## [1] "second.brk.points:"
## [1] 1323
## [1] "an:"
## [1] 113
## [1] "second.brk.points:"
## [1] 1323 2709
## [1] "an:"
## [1] 164
## [1] "second.brk.points:"
## [1] 1323 2709
## [1] "an:"
## [1] 215
## [1] "second.brk.points:"
## [1] 1323 2709
```

```
#display the estimated break points
print("Estimated break points:")
```

```
## [1] "Estimated break points:"
```

```
print(temp$final.selected.points)
```

```
## [1] 1333 2666
```

```
#display the true break points
print("True break points:")
```

```
## [1] "True break points:"
```

```
print(brk[-length(brk)])
```

```
## [1] 1333 2666
```

Optional Arguments and Default Values

In addition to the argument `data` which was described above, there are other optional arguments for the function `bss()` which are listed below:

- `lambda.1.cv`: the tuning parmaeter λ_1 for fused lasso. By default, `lambda.1.cv` is a vector with decreasing values constructed based on the time series data, the number of time series components `p`, and the block size `block.size`.

- `lambda.2.cv`: the tuning parmaeter λ_2 for fused lasso. By default, `lambda.2.cv` is a vector with decreasing values constructed based on the number of time points `T` and the number of time series components `p`.
- `q`: the AR order. By default, `q` = 1.
- `max.iteration`: the max number of iteration for the fused lasso. By default, `max.iteration` = 50.
- `tol`: tolerance for the fused lasso. By default, `tol` = 10^{-2}.
- `block.size`: the block size b_n. By default, $b_n = \lfloor \sqrt{T} \rfloor$.
- `blocks`: the blocks (sequence). By default, `blocks = seq(0,T,block.size)`. One could also use `blocks` to handle varying-size blocks. For example, `blocks= c(seq(0,5000,100), seq(5200,10000,200))`.
- `an.grid`: a list of values for a_n. By default it is NULL.