`covdepGE` versus `loggle` in varying structure recovery

Contents

Problem statement

Here, we compare the performance of `loggle` to `covdepGE` though a simulation study. In this example, `X` $\in \mathbb{R}^{180\times 5}$ and `Z` $\in \mathbb{R}^{180}$. `Z` is generated by drawing 60 times each from the uniform distribution on the intervals $(-3,-1),(-1,1)$, and $(1,3)$ (without loss of generality, we sort `Z` into ascending order). Then, we generate the l-th observation of `X` by drawing once from a 5 dimensional 0 mean Gaussian distribution with precision matrix $\Omega(z_l)$ defined as:

$$\Omega(z) = \begin{cases} \Omega^{(1)}(z) & z \in (-3,-1) \\ \Omega^{(2)}(z) & z \in (-1,1) \\ \Omega^{(3)}(z) & z \in (1,3) \end{cases} \tag{1}$$

Where $\Omega_1, \Omega_2, \Omega_3 \in \mathbb{R}^{5\times 5}$ are defined as:

$$\left[\Omega^{(1)}(z)\right]_{j,k} = \begin{cases} 2 & j=k \\ 1 & (j,k) \in \{(1,2),(2,1),(2,3),(3,2)\} \\ 0 & \text{otherwise} \end{cases} \quad \left[\Omega^{(2)}(z)\right]_{j,k} = \begin{cases} 2 & j=k \\ \frac{1-z}{2} & (j,k) \in \{(1,2),(2,1)\} \\ \frac{1+z}{2} & (j,k) \in \{(1,3),(3,1)\} \\ 1 & (j,k) \in \{(2,3),(3,2)\} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$\left[\Omega^{(3)}(z)\right]_{j,k} = \begin{cases} 2 & j=k \\ 1 & (j,k) \in \{(1,3),(3,1),(2,3),(3,2)\} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

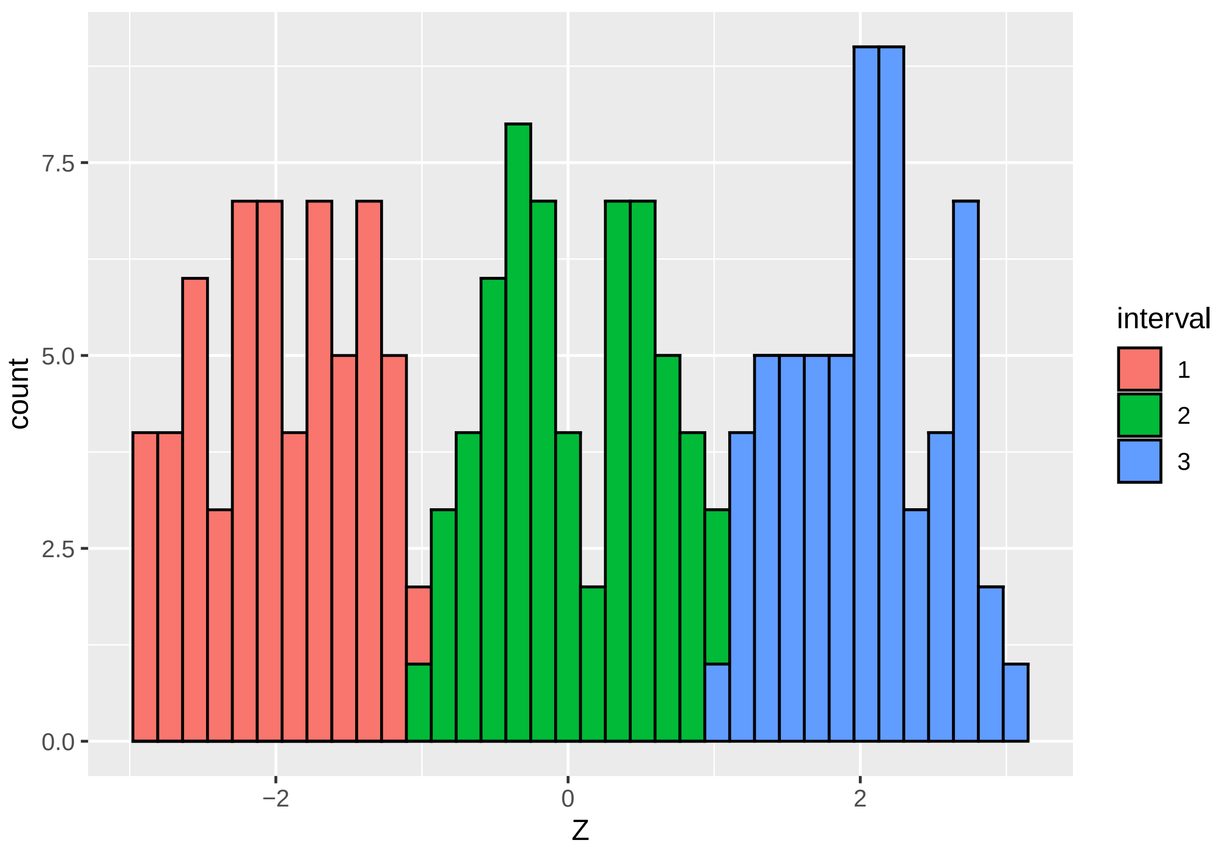
Thus, as z approaches -1 from the right, $\Omega(z)$ approaches $\Omega^{(1)}(z)$ (more formally, $\|\Omega(z) - \Omega^{(1)}(z)\|$ goes to 0). Similarly, as z approaches 1 from the left, $\Omega(z)$ goes to $\Omega^{(3)}(z)$. We visualize these precision matrices and the corresponding structures below.

Although `loggle` treats the extraneous covariate as time, since the data are sorted according to `Z`, the precision matrices can be thought of as varying in time instead of in `Z`. Note that `loggle` assumes that the data are sampled at uniformly spaced time intervals, however, `Z` is not uniformly spaced.

Data generation

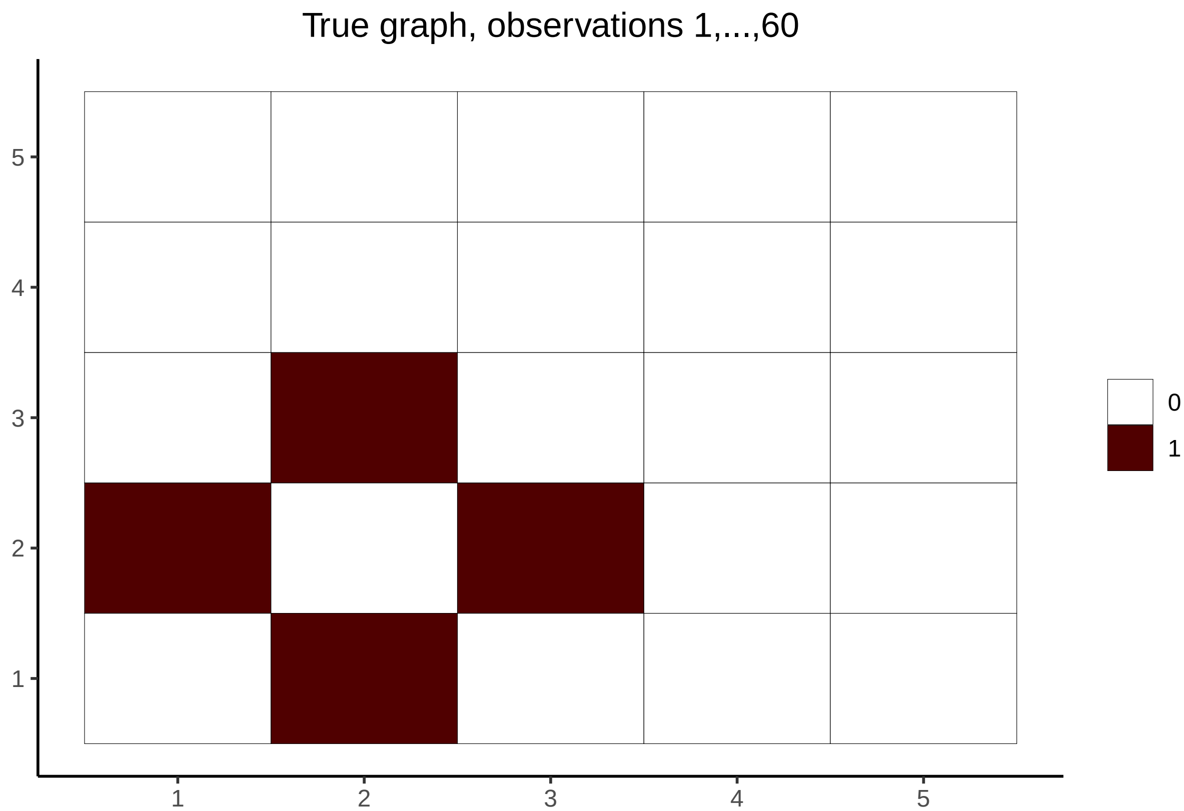
Here, we show how the data are generated.

```r
# devtools::install_github("JacobHelwig/covdepGE")
library(loggle)
library(covdepGE)
library(ggplot2)

# get the data
set.seed(1)
data <- generateData()
X <- data$X
Z <- data$Z
interval <- data$interval
prec <- data$true_precision

# get overall and within interval sample sizes
n <- nrow(X)
n1 <- sum(interval == 1)
n2 <- sum(interval == 2)
n3 <- sum(interval == 3)

# visualize the distribution of the extraneous covariate
ggplot(data.frame(Z = Z, interval = as.factor(interval))) +
  geom_histogram(aes(Z, fill = interval), color = "black", bins = n %/% 5)
```

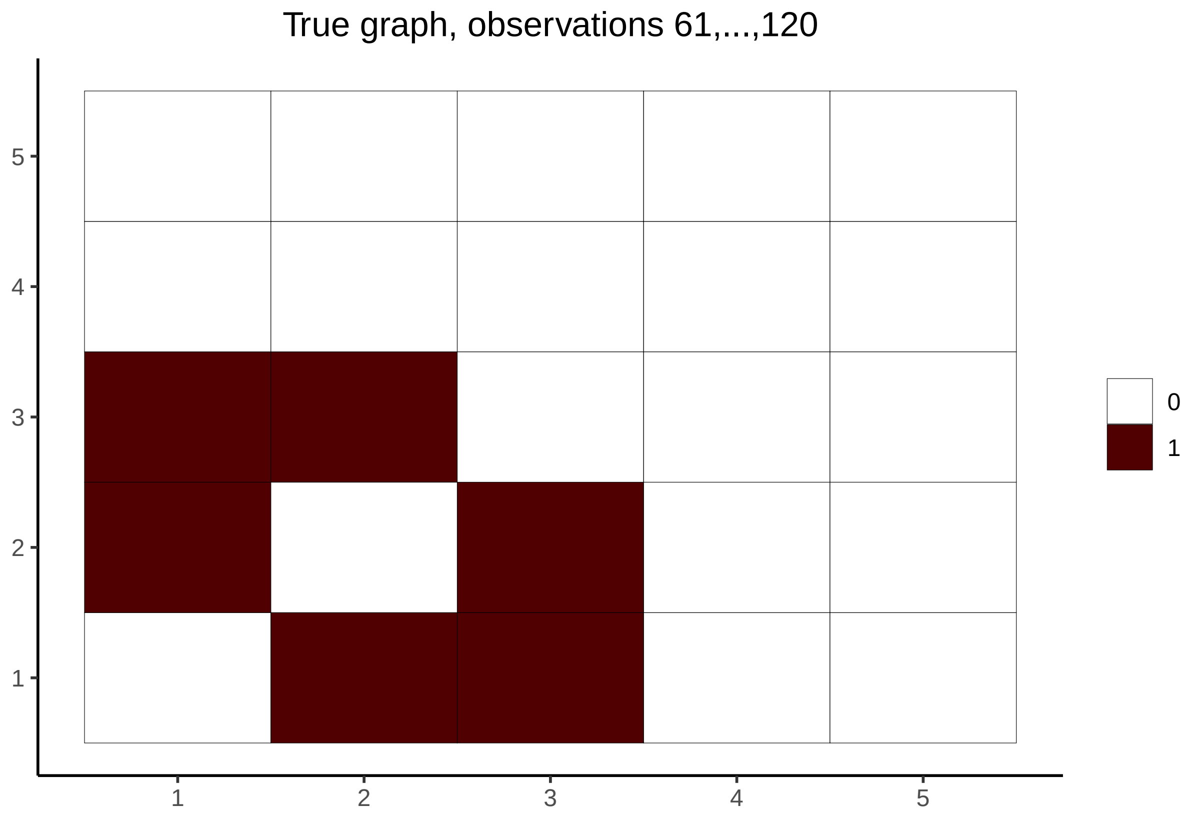


```r
# visualize the true conditional dependence structure in each of the intervals
titles <- paste0(rep("True graph, observations "), c(1, n1 + 1, n1 + n2 + 1),
                 rep(",...,", 3), c(n1, n1 + n2, n))
true_graphs <- lapply(lapply(lapply(prec, `!=`, 0), `*`, 1), `-`, diag(5))
lapply(1:3, function(j) matViz(unique(true_graphs)[[j]]) + ggtitle(titles[j]))
```

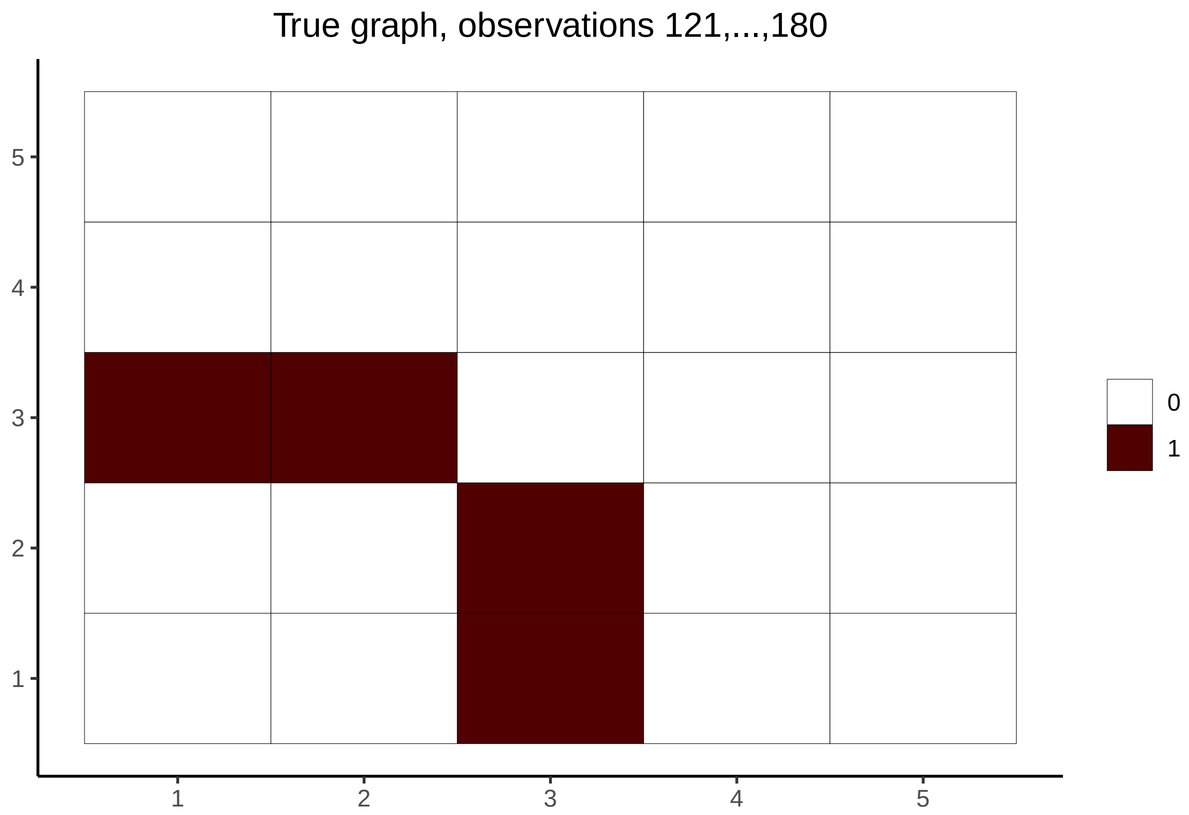
```
## [[1]]
```




```
##
## [[2]]
```


True graph, observations 61,...,120
5
4
3
2
1
1
2
3
4
5
0
1

```
##
## [[3]]
```


True graph, observations 121,...,180
5
4
3
2
1
1
2
3
4
5
0
1

Comparison

In this section, we fit each of the methods, calculate sensitivity and specificity, and visualize the resulting structures.

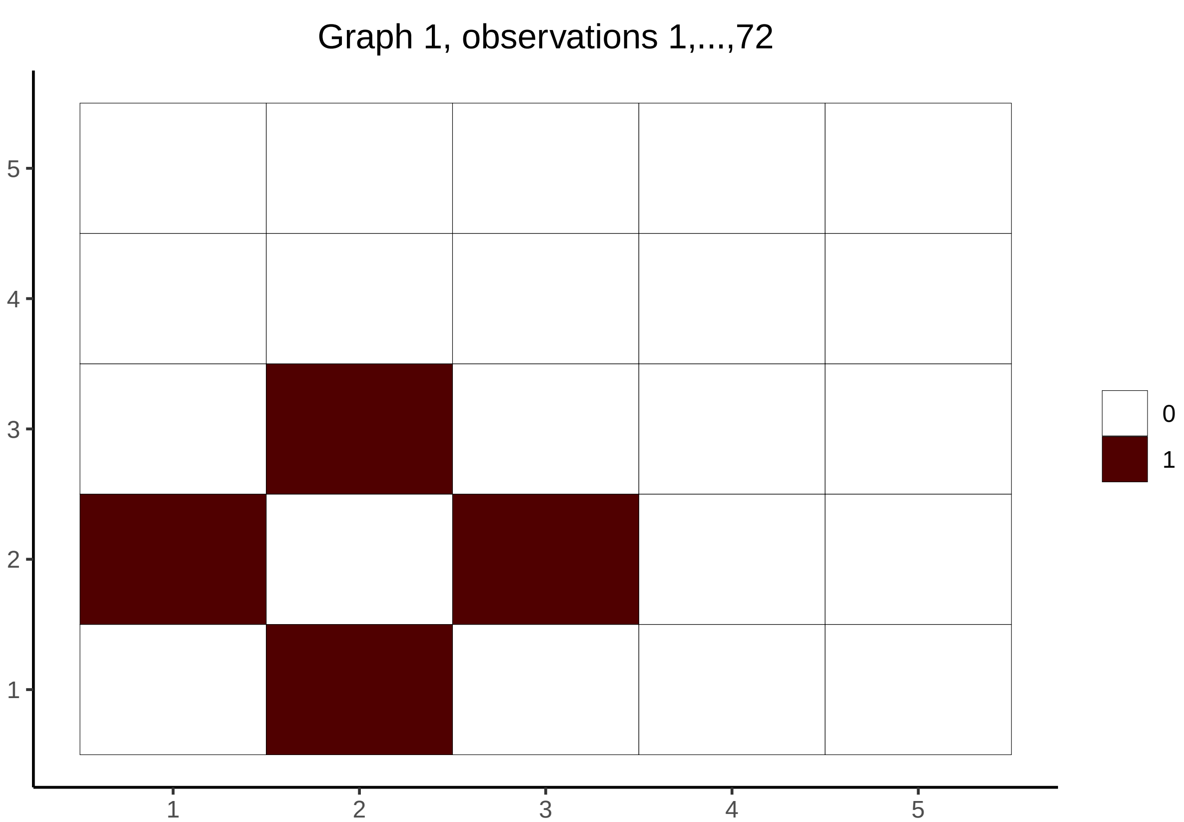
covdepGE

```r
# covdepGE; factors paralellism along p
(out_covdepGE <- covdepGE(X, Z, parallel = T, num_workers = 5))
```

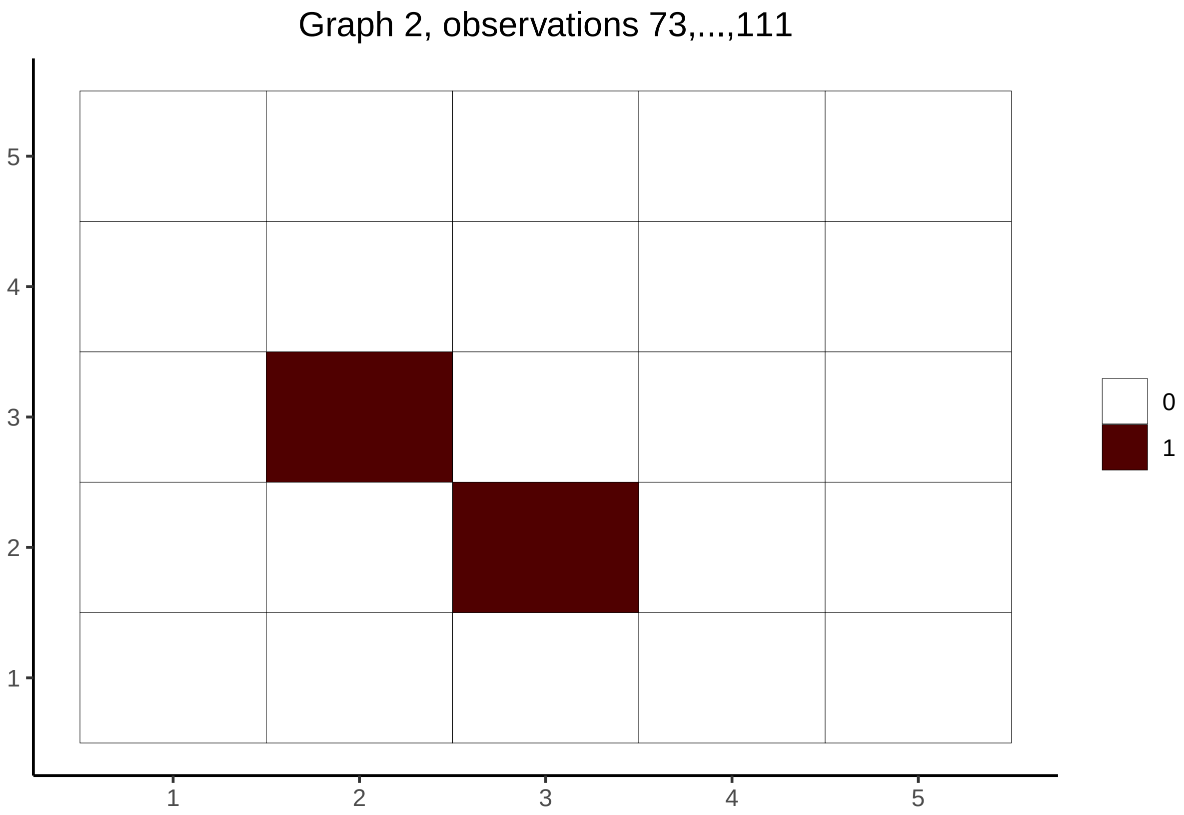
```
## Warning in covdepGE(X, Z, parallel = T, num_workers = 5): No registered workers
## detected; registering doParallel with 5 workers

##                        Covariate Dependent Graphical Model
##
## ELBO: -186296.89                                            # Unique Graphs: 3
## n: 180, variables: 5                       Hyperparameter grid size: 125 points
## Model fit completed in 2.496 secs
```

```r
plot(out_covdepGE)
```

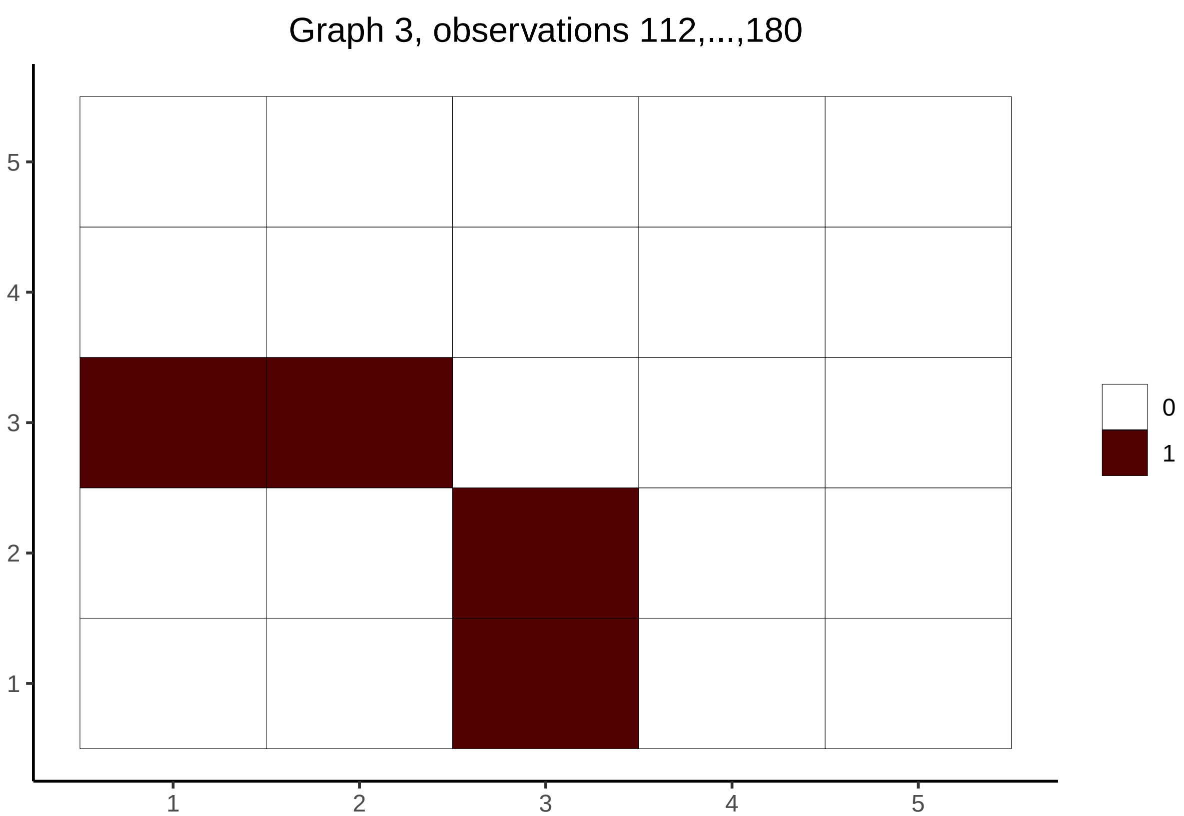
```
## [[1]]
```




```
##
## [[2]]
```


Graph 2, observations 73,...,111
5
4
3
2
1
1
2
3
4
5
0
1

```
##
## [[3]]
```


Graph 3, observations 112,...,180
5
4
3
2
1
1
2
3
4
5
0
1

```r
# calculate number of true edges and non-edges (mask out diagonal)
n <- nrow(X)
p <- ncol(X)
trueGraphs0 <- lapply(true_graphs, `+`, diag(rep(NA, p)))
trueGraphs <- array(unlist(trueGraphs0), dim = c(p, p, n))
num_true1 <- sum(trueGraphs, na.rm = T)
num_true0 <- sum(trueGraphs == 0, na.rm = T)

# calculate number of correctly detected edges
pred_graphs <- out_covdepGE$graphs$graphs
predGraphs <- array(unlist(pred_graphs), dim = c(p, p, n))
correct1 <- sum(predGraphs == trueGraphs & trueGraphs == 1, na.rm = T)

# calculate number of correctly detected non-edges
correct0 <- sum(predGraphs == trueGraphs & trueGraphs == 0, na.rm = T)

# display sensitivity and specificity
sens <- correct1 / num_true1
spec <- correct0 / num_true0
cat("\nSensitivity:", round(sens, 3))
```

```
##
## Sensitivity: 0.764
```

```r
cat("\nSpecificity:", round(spec, 3))
```

```
##
## Specificity: 1
```

```r
rm(list = c("pred_graphs", "predGraphs", "correct1", "correct0", "sens", "spec"))
```

loggle

```r
# loggle; factors parallelism along n
start <- Sys.time()
(num_workers <- parallel::detectCores())
```

```
## [1] 56
```

```r
out_loggle <- loggle.cv(t(X), num.thread = num_workers - 2, print.detail = F)
```

```
##
## Running h = 0.1 ...
## Using d.list: 0 0.001 0.01 0.025 0.05 0.075 0.1 0.15 0.2 0.25 0.3 1
## Using lambda.list: 0.15 0.17 0.19 0.21 0.23 0.25 0.27 0.29 0.31 0.33 0.35
## Detrending each variable in data matrix...
##
## Running fold 1 out of 5 folds...
```

```
## Generating sample correlation matrices for training dataset...
## Estimating graphs for training dataset...
## Calculating cross-validation scores for testing dataset...
##
## Running fold 2 out of 5 folds...
## Generating sample correlation matrices for training dataset...
## Estimating graphs for training dataset...
## Calculating cross-validation scores for testing dataset...
##
## Running fold 3 out of 5 folds...
## Generating sample correlation matrices for training dataset...
## Estimating graphs for training dataset...
## Calculating cross-validation scores for testing dataset...
##
## Running fold 4 out of 5 folds...
## Generating sample correlation matrices for training dataset...
## Estimating graphs for training dataset...
## Calculating cross-validation scores for testing dataset...
##
## Running fold 5 out of 5 folds...
## Generating sample correlation matrices for training dataset...
## Estimating graphs for training dataset...
## Calculating cross-validation scores for testing dataset...
##
## Running h = 0.15 ...
## Using d.list: 0 0.001 0.01 0.025 0.05 0.075 0.1 0.15 0.2 0.25 0.3 1
## Using lambda.list: 0.15 0.17 0.19 0.21 0.23 0.25 0.27 0.29 0.31 0.33 0.35
## Detrending each variable in data matrix...
##
## Running fold 1 out of 5 folds...
## Generating sample correlation matrices for training dataset...
## Estimating graphs for training dataset...
## Calculating cross-validation scores for testing dataset...
##
## Running fold 2 out of 5 folds...
## Generating sample correlation matrices for training dataset...
## Estimating graphs for training dataset...
## Calculating cross-validation scores for testing dataset...
##
## Running fold 3 out of 5 folds...
## Generating sample correlation matrices for training dataset...
## Estimating graphs for training dataset...
## Calculating cross-validation scores for testing dataset...
##
## Running fold 4 out of 5 folds...
## Generating sample correlation matrices for training dataset...
## Estimating graphs for training dataset...
## Calculating cross-validation scores for testing dataset...
##
## Running fold 5 out of 5 folds...
## Generating sample correlation matrices for training dataset...
## Estimating graphs for training dataset...
## Calculating cross-validation scores for testing dataset...
##
```

```
## Running h = 0.2 ...
## Using d.list: 0 0.001 0.01 0.025 0.05 0.075 0.1 0.15 0.2 0.25 0.3 1
## Using lambda.list: 0.15 0.17 0.19 0.21 0.23 0.25 0.27 0.29 0.31 0.33 0.35
## Detrending each variable in data matrix...
##
## Running fold 1 out of 5 folds...
## Generating sample correlation matrices for training dataset...
## Estimating graphs for training dataset...
## Calculating cross-validation scores for testing dataset...
##
## Running fold 2 out of 5 folds...
## Generating sample correlation matrices for training dataset...
## Estimating graphs for training dataset...
## Calculating cross-validation scores for testing dataset...
##
## Running fold 3 out of 5 folds...
## Generating sample correlation matrices for training dataset...
## Estimating graphs for training dataset...
## Calculating cross-validation scores for testing dataset...
##
## Running fold 4 out of 5 folds...
## Generating sample correlation matrices for training dataset...
## Estimating graphs for training dataset...
## Calculating cross-validation scores for testing dataset...
##
## Running fold 5 out of 5 folds...
## Generating sample correlation matrices for training dataset...
## Estimating graphs for training dataset...
## Calculating cross-validation scores for testing dataset...
##
## Running h = 0.25 ...
## Using d.list: 0 0.001 0.01 0.025 0.05 0.075 0.1 0.15 0.2 0.25 0.3 1
## Using lambda.list: 0.15 0.17 0.19 0.21 0.23 0.25 0.27 0.29 0.31 0.33 0.35
## Detrending each variable in data matrix...
##
## Running fold 1 out of 5 folds...
## Generating sample correlation matrices for training dataset...
## Estimating graphs for training dataset...
## Calculating cross-validation scores for testing dataset...
##
## Running fold 2 out of 5 folds...
## Generating sample correlation matrices for training dataset...
## Estimating graphs for training dataset...
## Calculating cross-validation scores for testing dataset...
##
## Running fold 3 out of 5 folds...
## Generating sample correlation matrices for training dataset...
## Estimating graphs for training dataset...
## Calculating cross-validation scores for testing dataset...
##
## Running fold 4 out of 5 folds...
## Generating sample correlation matrices for training dataset...
## Estimating graphs for training dataset...
## Calculating cross-validation scores for testing dataset...
```

```
##
## Running fold 5 out of 5 folds...
## Generating sample correlation matrices for training dataset...
## Estimating graphs for training dataset...
## Calculating cross-validation scores for testing dataset...
##
## Running h = 0.3 ...
## Using d.list: 0 0.001 0.01 0.025 0.05 0.075 0.1 0.15 0.2 0.25 0.3 1
## Using lambda.list: 0.15 0.17 0.19 0.21 0.23 0.25 0.27 0.29 0.31 0.33 0.35
## Detrending each variable in data matrix...
##
## Running fold 1 out of 5 folds...
## Generating sample correlation matrices for training dataset...
## Estimating graphs for training dataset...
## Calculating cross-validation scores for testing dataset...
##
## Running fold 2 out of 5 folds...
## Generating sample correlation matrices for training dataset...
## Estimating graphs for training dataset...
## Calculating cross-validation scores for testing dataset...
##
## Running fold 3 out of 5 folds...
## Generating sample correlation matrices for training dataset...
## Estimating graphs for training dataset...
## Calculating cross-validation scores for testing dataset...
##
## Running fold 4 out of 5 folds...
## Generating sample correlation matrices for training dataset...
## Estimating graphs for training dataset...
## Calculating cross-validation scores for testing dataset...
##
## Running fold 5 out of 5 folds...
## Generating sample correlation matrices for training dataset...
## Estimating graphs for training dataset...
## Calculating cross-validation scores for testing dataset...
##
## Selecting models based on 5-fold cross-validation results...
```

```r
elapsed <- round(Sys.time() - start, 3)
cat("\nTime to fit loggle:", elapsed, attr(elapsed, "units"), "\n")
```

```
##
## Time to fit loggle: 2.186 mins
```

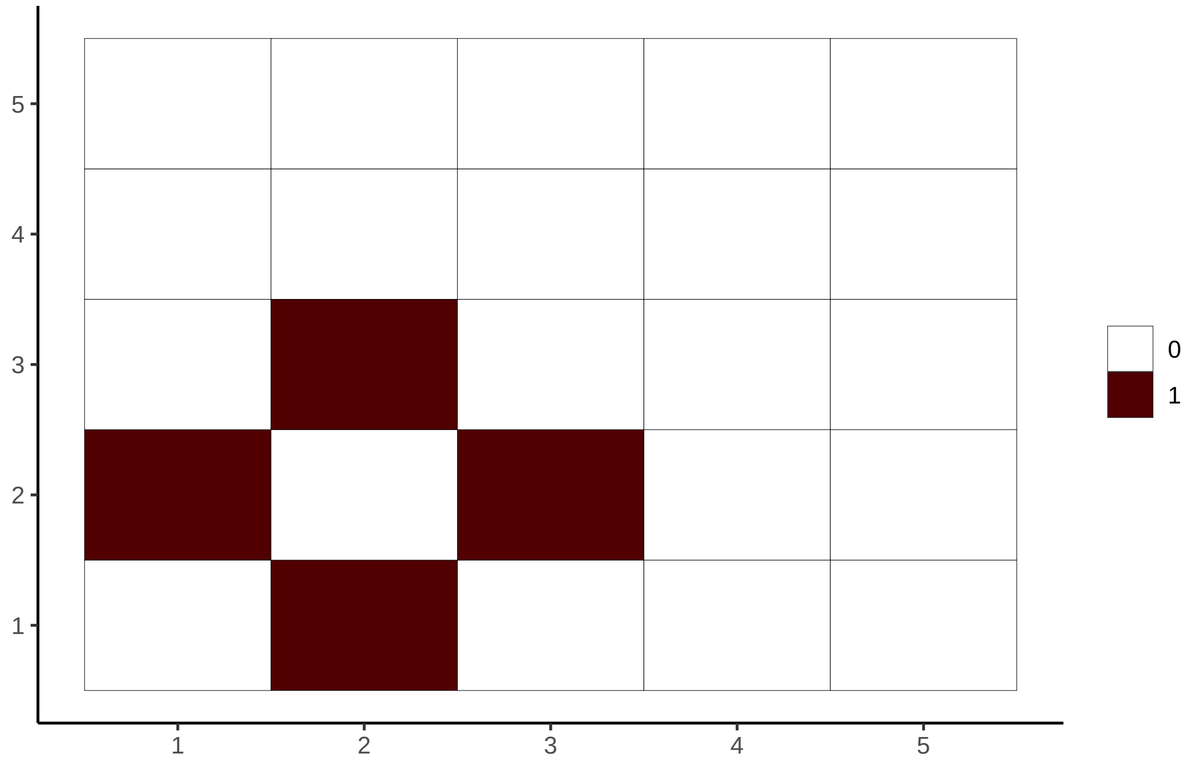
```r
out_loggle <- out_loggle$cv.select.result
gc()
```

```
##           used  (Mb) gc trigger  (Mb) max used   (Mb)
## Ncells 2103389 112.4   15826412 845.3 19783014 1056.6
## Vcells 3798297  29.0   29911516 228.3 33525121  255.8
```

```r
# get the unique graphs and find which observations they correspond to
graphs <- lapply(lapply(out_loggle$adj.mat.opt, as.matrix), `-`, diag(5))
unique_graphs <- unique(graphs)
unique_sum <- vector("list", length(unique_graphs))
names(unique_sum) <- paste0("graph", 1:length(unique_graphs))

# iterate over each of the unique graphs
for (j in 1:length(unique_graphs)){

  # fix the unique graph
  graph <- unique_graphs[[j]]

  # find indices of the observations corresponding to this graph
  graph_inds <- which(sapply(graphs, identical, graph))

  # split up the contiguous subsequences of these indices
  cont_inds <- split(sort(graph_inds), cumsum(c(1, diff(sort(graph_inds)) != 1)))

  # create a character summary for each of the contiguous sequences
  inds_sum <- sapply(cont_inds, function(idx_seq) ifelse(length(
    idx_seq) > 3, paste0(min(idx_seq), ",...,", max(idx_seq)),
    paste0(idx_seq, collapse = ",")))

  # combine the summary
  inds_sum <- paste0(inds_sum, collapse = ",")

  # add the graph, indices, and summary to the unique graphs summary list
  unique_sum[[j]] <- list(graph = graph, indices = graph_inds,
                          ind_sum = inds_sum)
}

# visualize each of the unique graphs
lapply(1:length(unique_sum), function(j) matViz(unique_sum[[j]]$graph) +
         ggtitle(paste0("Graph ", j, ", observations ", unique_sum[[j]]$ind_sum)))
```

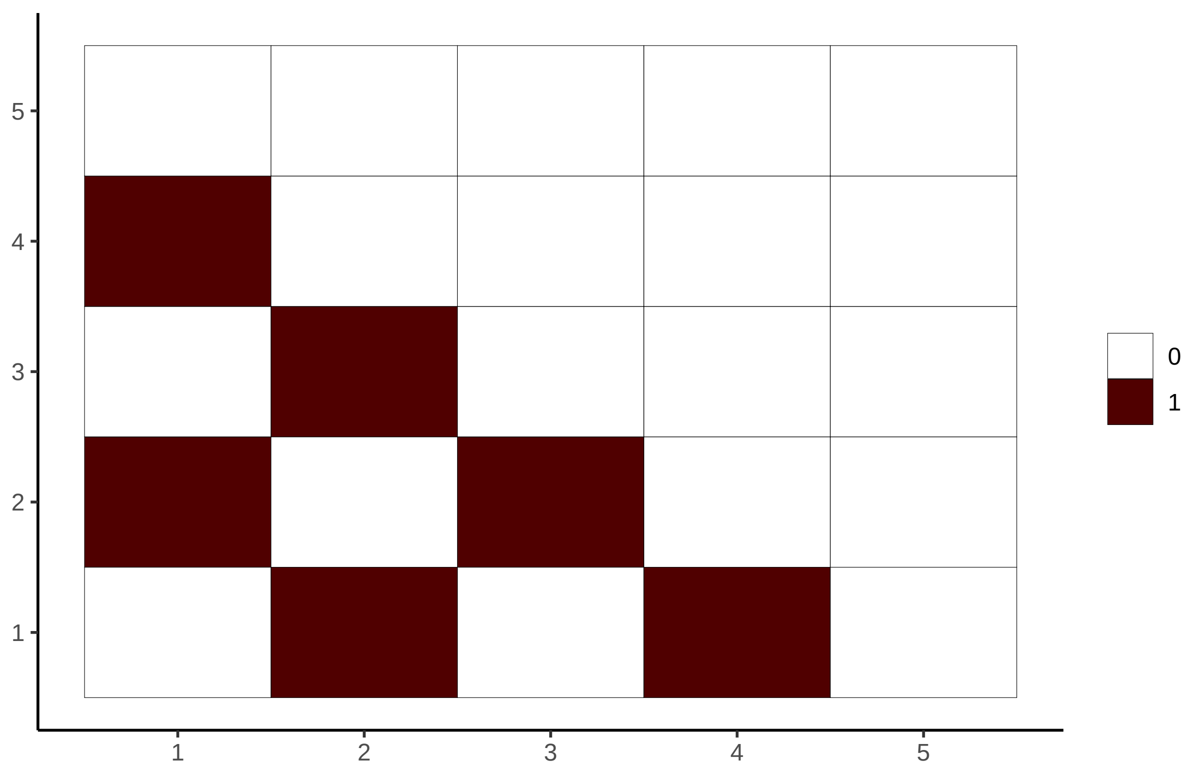
```
## [[1]]
```

Graph 1, observations 1,...,63,79

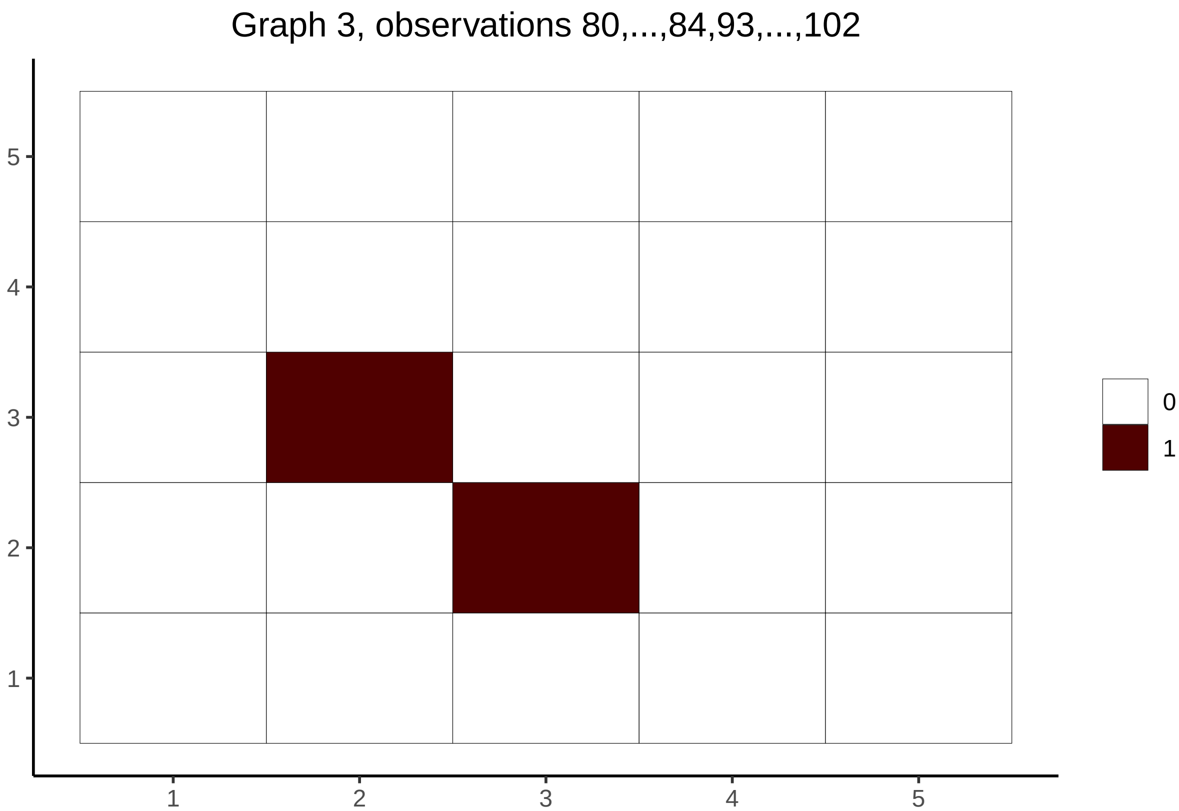



```
##
## [[2]]
```

Graph 2, observations 64,...,78

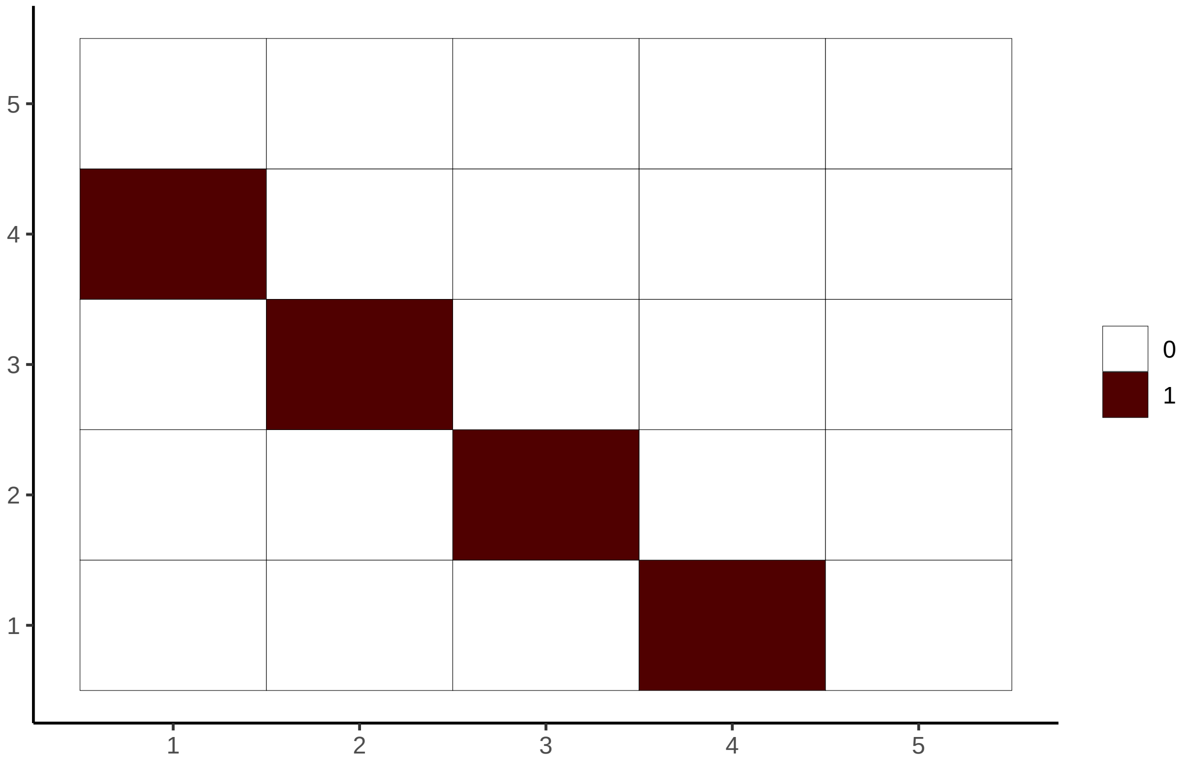


```
##
## [[3]]
```



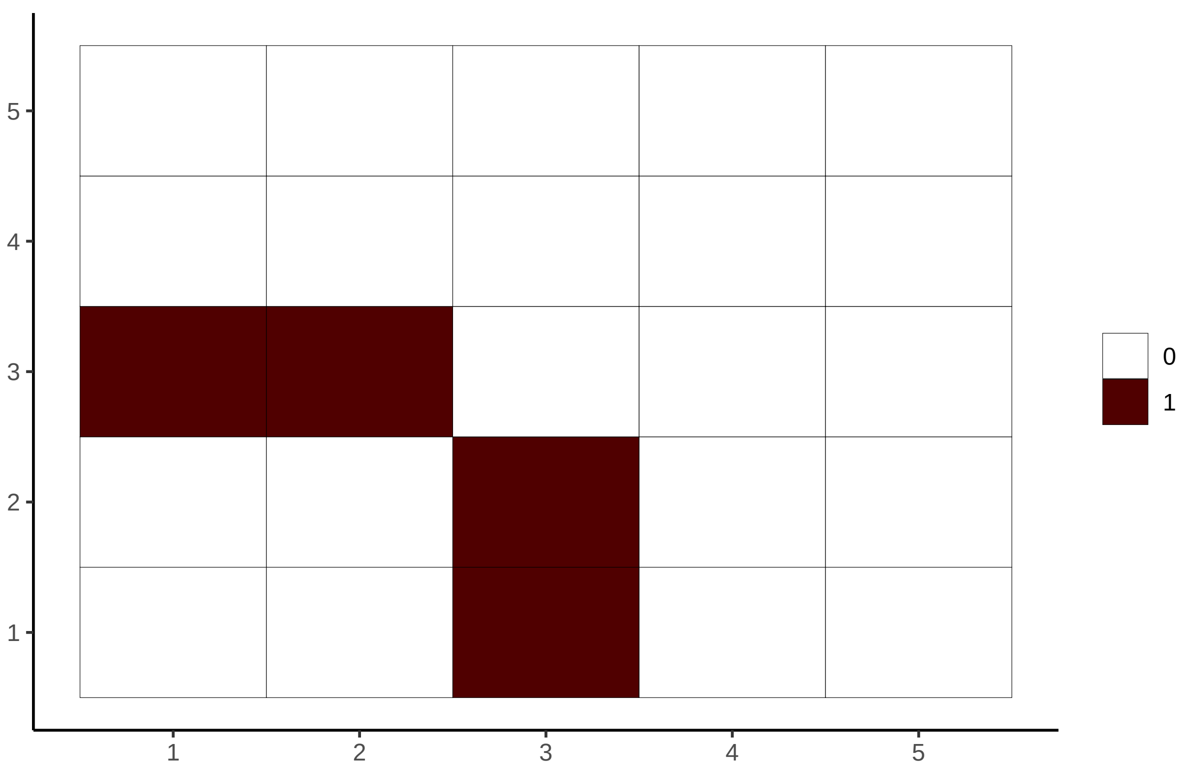

```
##
## [[4]]
```

Graph 4, observations 85,...,92

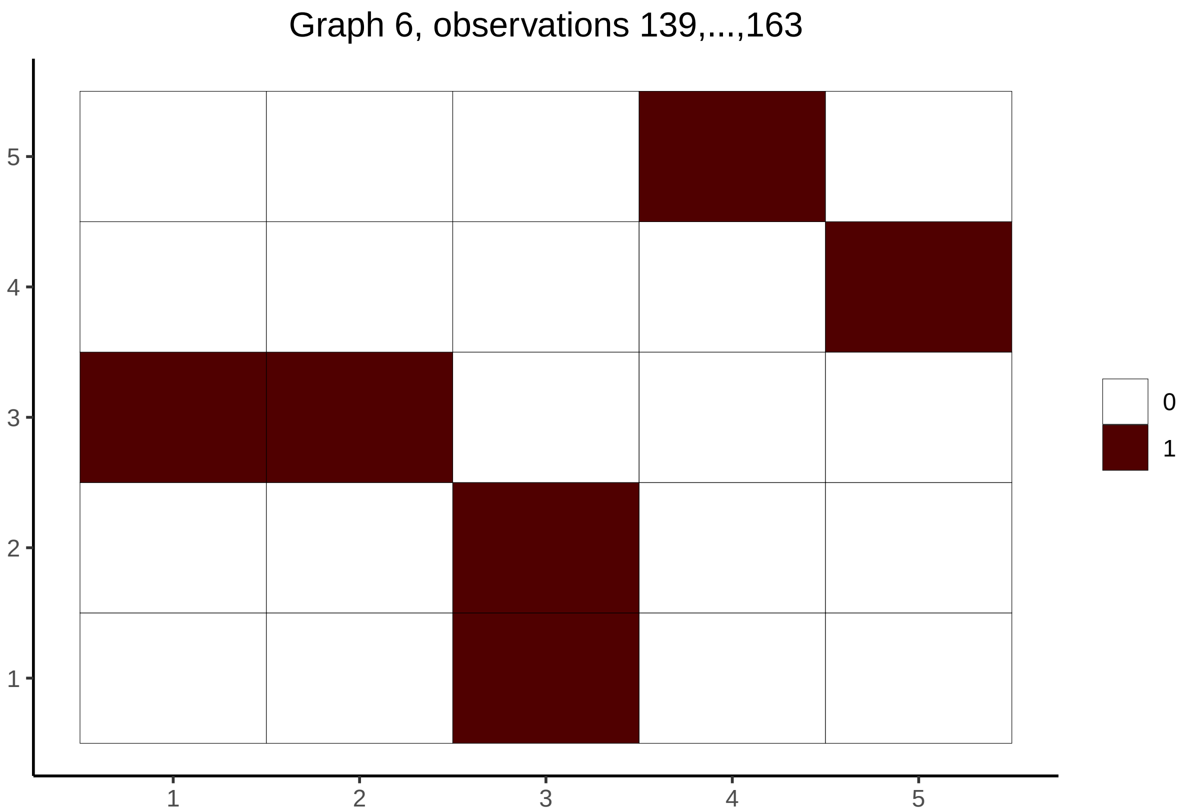



```
##
## [[5]]
```

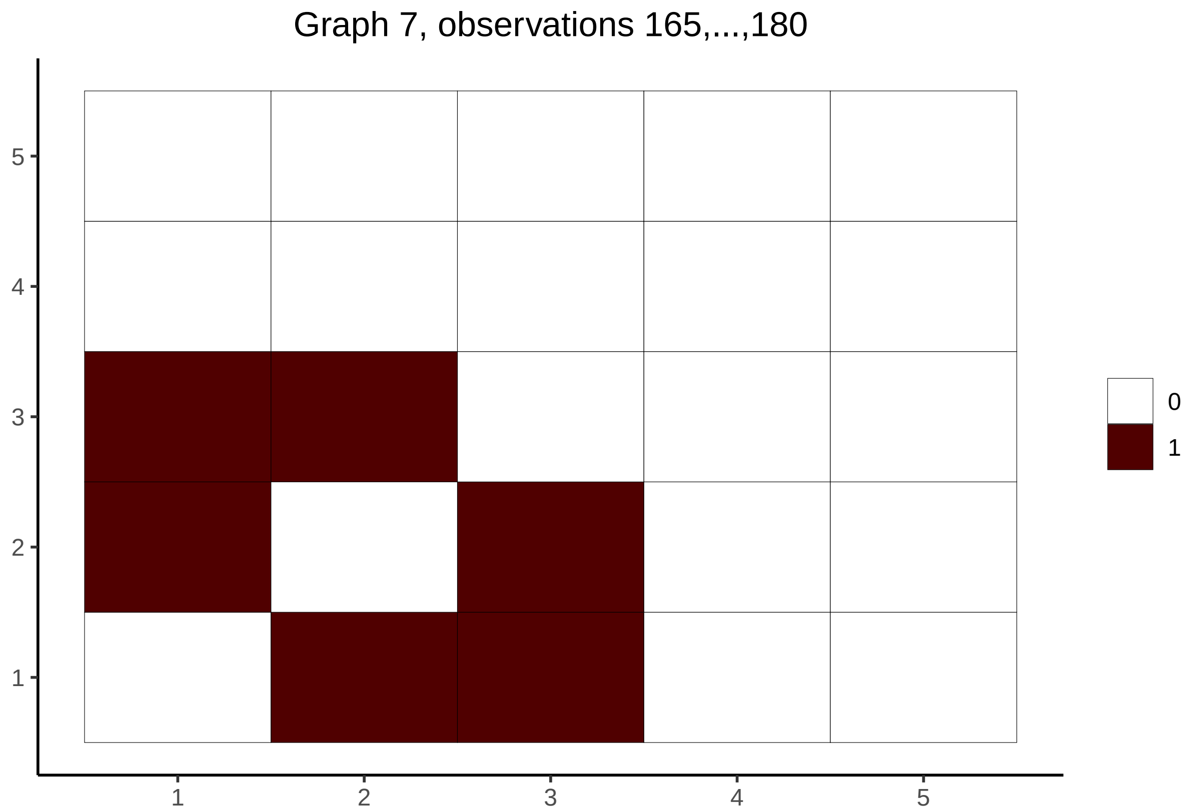
Graph 5, observations 103,...,138,164



```
##
## [[6]]
```




```
##
## [[7]]
```




```r
# calculate number of correctly detected edges
predGraphs <- array(unlist(graphs), dim = c(p, p, n))
correct1 <- sum(predGraphs == trueGraphs & trueGraphs == 1, na.rm = T)

# calculate number of correctly detected non-edges
correct0 <- sum(predGraphs == trueGraphs & trueGraphs == 0, na.rm = T)

# display sensitivity and specificity
sens <- correct1 / num_true1
spec <- correct0 / num_true0
cat("\nSensitivity:", round(sens, 3))
```

```
##
## Sensitivity: 0.802
```

```r
cat("\nSpecificity:", round(spec, 3))
```

```
##
## Specificity: 0.954
```