Using sabinaHSBM for link prediction and network reconstruction using Hierarchical Stochastic Block Models

Example on how to use it to identify missing and spurious links with the full\_reconstruction method (with parallelized computation)

The *sabinaHSBM* package provides tools for link prediction and network reconstruction using hierarchical stochastic block models (HSBM). This document demonstrates a simple use for the full\_reconstruction method, showcasing how it handles missing and spurious links in an incomplete and error-prone network. To enhance computational efficiency during the prediction links step we leverage parallelized computation.

We use the example dataset dat included in the package to show key functionalities, including data preparation, link prediction, and network reconstruction.

# Loading Required Libraries

We begin by loading the required R packages. Missing packages are automatically installed.

# If the package is not installed, install it from GitHub  
# if (!requireNamespace("sabinaHSBM", quietly = TRUE)) {  
# library(remotes)  
# remotes::install\_github("h-lima/sabinaHSBM")  
# }  
  
# Load the sabinaHSBM package  
#library(sabinaHSBM)  
devtools::load\_all("/home/project\_hsbm/graph\_tool/sabinaHSBM/")

If the required packages are not already installed, install them from CRAN

list.of.packages <- c(  
 "dplyr",  
 "parallel",  
 "reshape2",  
 "reticulate",  
 "stringr",  
 "tidyr",  
 "ROCR",  
 "data.table"  
)  
  
new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()[, "Package"])]  
if (length(new.packages) > 0) {  
 install.packages(new.packages, dependencies = TRUE)  
}  
  
for (package in list.of.packages) {  
 library(package, character.only = TRUE)  
}

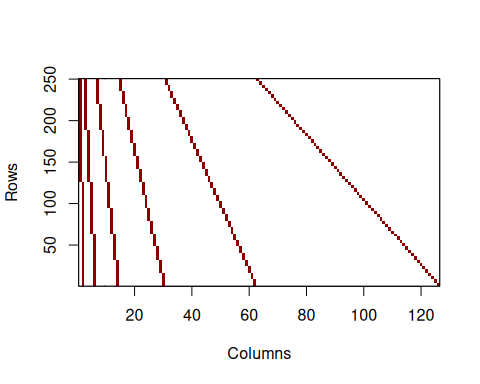
# Record starting time (optional)  
start\_time <- Sys.time()

# Load data

The dataset dat2 is a binary bipartite matrix representing a hypothetical species interactions network. Columns and rows correspond to two different types of nodes (e.g., hosts and parasites), and links (values of 1, in red) represent interaction between them while 0 (in black) represent lack of observed interaction. The distribution of these links follows a structured pattern.

# Load the dataset  
data(dat2, package = "sabinaHSBM")  
dat<- dat2

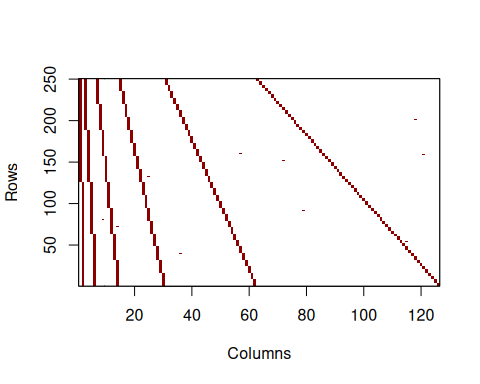
# Plot the original matrix  
plot\_interaction\_matrix(adj\_mat = dat, order\_mat = FALSE)



To simulate spurious links and address the full\_reconstruction method, we randomly add a small number of false positives (1) to the matrix. This controlled modification allows us to demonstrate how the method identifies both spurious links (observed but potentially erroneous interactions, false positives) and missing links (unobserved but likely interactions, false negatives).

# Add spurious links artificially  
set.seed(123)  
num\_spurious <- ceiling(sum(dat) \* 0.01) # Proportion of spurious links to add  
spurious\_links <- matrix(nrow = num\_spurious, ncol = 2)  
  
for (i in 1:num\_spurious) {  
 repeat {  
 random\_row <- sample(1:nrow(dat), 1)  
 random\_col <- sample(1:ncol(dat), 1)  
   
 if (dat[random\_row, random\_col] == 0) {  
 dat[random\_row, random\_col] <- 1  
 spurious\_links[i,] <- c(random\_row, random\_col)  
 break  
 }  
 }  
 }

# Plot the matrix with spurious  
plot\_interaction\_matrix(dat, order\_mat = FALSE)



# Preparing Input data for HSBM

The hsbm.input function pre-processes the dataset, creating cross-validation folds and edge lists required for modeling. Here, we use 5-fold cross-validation and the full\_reconstruction method.

# Prepare input data  
n\_folds <- 5 # Number of folds for cross-validation  
  
myInput <- hsbm.input(  
 dat, # Binary bipartite matrix of observed links  
 n\_folds = n\_folds,  
 add\_spurious = FALSE # Simulate spurious links  
)

## [1] "Actual cross-validation rate is 0.200"  
## [2] "Actual cross-validation rate is 0.200"  
## [3] "Actual cross-validation rate is 0.200"  
## [4] "Actual cross-validation rate is 0.200"  
## [5] "Actual cross-validation rate is 0.200"

Only for this example, the following code assigns the “spurious\_edge” label to the previously added spurious links. The purpose is to track these links throughout the analysis and network reconstruction process.

myInput$edgelists <- lapply(myInput$edgelists,  
 function(x){  
 x$x <- 1  
 return(x)  
 })  
  
for(i in 1:length(myInput$edgelists)){  
 el <- myInput$edgelists[[i]]  
 rows <- as.numeric(el[, 1]) + 1  
 cols <- as.numeric(el[, 2]) - nrow(myInput$data) + 1  
 rows\_cols <- paste0(rows, "\_", cols)  
 spurious\_rows\_cols <- paste0(spurious\_links[, 1], "\_", spurious\_links[, 2])  
 spurious\_loc <- which(rows\_cols %in% spurious\_rows\_cols)  
 el[spurious\_loc, ]$x <- 1  
 el[spurious\_loc, ]$edge\_type <- "spurious\_edge"  
 myInput$edgelists[[i]] <- el  
}

# Summarizes network characteristics  
summary(myInput)

## n\_rows n\_cols n\_links rows\_single\_link cols\_single\_link possible\_links  
## 1 250 126 1515 0 0 31500

# Predict link probabilities

The hsbm.predict function applies the HSBM to predict probabilities of all links (observed and unobserved links). This step is crucial to identify spurious and missing links within the data, which are often present in incomplete or error-prone networks. The function works directly with the processed input created by hsbm.input. This step can be computationally intensive when working with large datasets or numerous folds. To improve performance, we use parallelized computation, distributing tasks across multiple cores.

# Define the number of cores to use  
#nCores <- detectCores() - 1  
nCores <- 2  
  
# Generate HSBM predictions  
myPred <- hsbm.predict(  
 myInput, # Input data processed by hsbm.input()  
 iter = 10000, # Number of iterations  
 wait= 10000, # Number of iterations for MCMC equilibration  
 method = "full\_reconstruction", # Method for link prediction  
 save\_blocks = TRUE,   
 save\_pickle = FALSE,  
 n\_cores = nCores  
)

Predicted link probabilities and group assignments are stored for each fold. Below, we extract the link probabilities (p) and groups assignments of links for fold 1.

# View probabilities for fold 1  
probabilities\_fold1 <- myPred$probs[[1]]  
head(probabilities\_fold1)

## v1 v2 p v1\_names v2\_names edge\_type  
## 1 0 250 1.0000000 sp1 SPa documented  
## 2 0 252 1.0000000 sp1 SPc documented  
## 3 0 256 1.0000000 sp1 SPg documented  
## 4 0 264 1.0000000 sp1 SPo documented  
## 5 0 280 0.9927993 sp1 SPeb documented  
## 6 0 312 1.0000000 sp1 SPkc documented

The group assignments provide the hierarchical clustering structure of nodes for each fold. Let’s extract and examine the group assignments for fold 1:

# View the group/block assignments for fold 1  
groups\_fold1 <- myPred$groups[[1]]  
  
# Filter one type of nodes (e.g., hosts, in columns)  
vnames <- colnames(myPred$data)  
groups\_fold1 <- groups\_fold1 %>% filter(names %in% vnames)  
  
g\_cols <- grep("^G", names(groups\_fold1))  
groups\_fold1[g\_cols] <- lapply(groups\_fold1[g\_cols], sort)  
  
print(groups\_fold1)

## nodes G1 G2 G3 G4 G5 G6 names  
## 1 250 24 33 3 1 1 0 SPa  
## 2 251 24 160 3 1 1 0 SPb  
## 3 252 24 160 3 1 1 0 SPc  
## 4 253 24 160 3 1 1 0 SPd  
## 5 254 24 160 3 1 1 0 SPe  
## 6 255 24 160 3 1 1 0 SPf  
## 7 256 24 160 3 1 1 0 SPg  
## 8 257 24 160 3 1 1 0 SPh  
## 9 258 24 160 3 1 1 0 SPi  
## 10 259 24 160 3 1 1 0 SPj  
## 11 260 24 160 3 1 1 0 SPk  
## 12 261 24 160 3 1 1 0 SPl  
## 13 262 24 160 3 1 1 0 SPm  
## 14 263 24 160 3 1 1 0 SPn  
## 15 264 24 160 3 1 1 0 SPo  
## 16 265 118 160 3 1 1 0 SPp  
## 17 266 170 160 3 1 1 0 SPq  
## 18 267 183 160 3 1 1 0 SPr  
## 19 268 183 160 3 1 1 0 SPs  
## 20 269 183 160 3 1 1 0 SPt  
## 21 270 183 160 3 1 1 0 SPu  
## 22 271 183 160 3 1 1 0 SPv  
## 23 272 183 160 3 1 1 0 SPw  
## 24 273 183 160 3 1 1 0 SPx  
## 25 274 183 160 3 1 1 0 SPy  
## 26 275 183 160 3 1 1 0 SPz  
## 27 276 183 160 3 1 1 0 SPab  
## 28 277 183 160 3 1 1 0 SPbb  
## 29 278 183 160 3 1 1 0 SPcb  
## 30 279 183 160 3 1 1 0 SPdb  
## 31 280 183 160 3 1 1 0 SPeb  
## 32 281 183 160 3 1 1 0 SPfb  
## 33 282 194 196 3 1 1 0 SPgb  
## 34 283 194 196 3 1 1 0 SPhb  
## 35 284 194 196 3 1 1 0 SPib  
## 36 285 194 196 3 1 1 0 SPjb  
## 37 286 194 196 3 1 1 0 SPkb  
## 38 287 194 196 3 1 1 0 SPlb  
## 39 288 194 196 3 1 1 0 SPmb  
## 40 289 194 196 3 1 1 0 SPnb  
## 41 290 194 196 3 1 1 0 SPob  
## 42 291 194 196 3 1 1 0 SPpb  
## 43 292 194 196 3 1 1 0 SPqb  
## 44 293 194 196 3 1 1 0 SPrb  
## 45 294 194 196 3 1 1 0 SPsb  
## 46 295 194 196 3 1 1 0 SPtb  
## 47 296 194 196 3 1 1 0 SPub  
## 48 297 211 196 3 1 1 0 SPvb  
## 49 298 216 196 3 1 1 0 SPwb  
## 50 299 216 196 3 1 1 0 SPxb  
## 51 300 216 196 3 1 1 0 SPyb  
## 52 301 216 196 3 1 1 0 SPzb  
## 53 302 216 196 3 1 1 0 SPac  
## 54 303 216 196 3 1 1 0 SPbc  
## 55 304 216 196 3 1 1 0 SPcc  
## 56 305 216 196 3 1 1 0 SPdc  
## 57 306 216 196 3 1 1 0 SPec  
## 58 307 216 196 3 1 1 0 SPfc  
## 59 308 216 196 3 1 1 0 SPgc  
## 60 309 216 196 3 1 1 0 SPhc  
## 61 310 216 196 3 1 1 0 SPic  
## 62 311 216 196 3 1 1 0 SPjc  
## 63 312 216 196 3 1 1 0 SPkc  
## 64 313 221 286 7 5 1 0 SPlc  
## 65 314 221 330 7 5 1 0 SPmc  
## 66 315 221 330 7 5 1 0 SPnc  
## 67 316 221 330 7 5 1 0 SPoc  
## 68 317 221 330 7 5 1 0 SPpc  
## 69 318 221 330 7 5 1 0 SPqc  
## 70 319 221 330 7 5 1 0 SPrc  
## 71 320 221 330 7 5 1 0 SPsc  
## 72 321 221 330 7 5 1 0 SPtc  
## 73 322 221 330 7 5 1 0 SPuc  
## 74 323 221 330 7 5 1 0 SPvc  
## 75 324 221 330 7 5 1 0 SPwc  
## 76 325 221 330 7 5 1 0 SPxc  
## 77 326 221 330 7 5 1 0 SPyc  
## 78 327 221 330 7 5 1 0 SPzc  
## 79 328 244 330 7 5 1 0 SPad  
## 80 329 251 330 7 5 1 0 SPbd  
## 81 330 257 330 7 5 1 0 SPcd  
## 82 331 257 330 7 5 1 0 SPdd  
## 83 332 257 330 7 5 1 0 SPed  
## 84 333 257 330 7 5 1 0 SPfd  
## 85 334 257 330 7 5 1 0 SPgd  
## 86 335 257 330 7 5 1 0 SPhd  
## 87 336 257 330 7 5 1 0 SPid  
## 88 337 263 330 7 5 1 0 SPjd  
## 89 338 263 330 7 5 1 0 SPkd  
## 90 339 263 330 7 5 1 0 SPld  
## 91 340 263 330 7 5 1 0 SPmd  
## 92 341 263 330 7 5 1 0 SPnd  
## 93 342 263 330 7 5 1 0 SPod  
## 94 343 263 330 7 5 1 0 SPpd  
## 95 344 263 330 7 5 1 0 SPqd  
## 96 345 263 365 7 5 1 0 SPrd  
## 97 346 263 365 7 5 1 0 SPsd  
## 98 347 263 365 7 5 1 0 SPtd  
## 99 348 263 365 7 5 1 0 SPud  
## 100 349 263 365 7 5 1 0 SPvd  
## 101 350 263 365 7 5 1 0 SPwd  
## 102 351 263 365 7 5 1 0 SPxd  
## 103 352 268 365 7 5 1 0 SPyd  
## 104 353 268 365 7 5 1 0 SPzd  
## 105 354 268 365 7 5 1 0 SPae  
## 106 355 268 365 7 5 1 0 SPbe  
## 107 356 268 365 7 5 1 0 SPce  
## 108 357 268 365 7 5 1 0 SPde  
## 109 358 268 365 7 5 1 0 SPee  
## 110 359 286 365 7 5 1 0 SPfe  
## 111 360 286 365 7 5 1 0 SPge  
## 112 361 286 365 7 5 1 0 SPhe  
## 113 362 286 365 7 5 1 0 SPie  
## 114 363 286 365 7 5 1 0 SPje  
## 115 364 286 365 7 5 1 0 SPke  
## 116 365 286 365 7 5 1 0 SPle  
## 117 366 286 365 7 5 1 0 SPme  
## 118 367 286 365 7 5 1 0 SPne  
## 119 368 286 365 7 5 1 0 SPoe  
## 120 369 286 365 7 5 1 0 SPpe  
## 121 370 286 365 7 5 1 0 SPqe  
## 122 371 286 365 7 5 1 0 SPre  
## 123 372 286 365 7 5 1 0 SPse  
## 124 373 286 365 7 5 1 0 SPte  
## 125 374 304 365 7 5 1 0 SPue  
## 126 375 309 365 7 5 1 0 SPve

Hierarchical group assignments provide insight into how nodes (e.g., hosts) are organized across multiple levels. At the first level (G1), nodes are divided into specific groups, reflecting fine-scale patterns. Moving to higher levels (G2, G3, G4), these groups (or communities) are progressively aggregated, revealing broader patterns and relationships or communities.

Let’s now look at the links for each fold filtered by type (“documented”, “held out”, and “spurious\_edge”) with a probability lower than 0.99.

for(i in 1:5){  
 cat("Fold ", i, "\n")  
 cat("\tDocumented\n\t")  
 print(dplyr::filter(myPred$probs[[i]], edge\_type == "documented", p < 0.99))  
 cat("\tHeld out\n\t")  
 print(dplyr::filter(myPred$probs[[i]], edge\_type == "held\_out", p < 0.99))  
 cat("\tSpurious edge\n\t")  
 print(dplyr::filter(myPred$probs[[i]], edge\_type == "spurious\_edge", p < 0.99))  
}

## Fold 1   
## Documented  
## v1 v2 p v1\_names v2\_names edge\_type  
## 1 16 316 0.9512951 sp17 SPoc documented  
## 2 69 330 0.9757976 sp70 SPcd documented  
## 3 111 340 0.9798980 sp112 SPmd documented  
## 4 143 298 0.9797980 sp144 SPwb documented  
## 5 147 349 0.9827983 sp148 SPvd documented  
## 6 154 351 0.9853985 sp155 SPxd documented  
## 7 204 364 0.9887989 sp205 SPke documented  
## 8 216 307 0.9894989 sp217 SPfc documented  
## Held out  
## [1] v1 v2 p v1\_names v2\_names edge\_type  
## <0 rows> (or 0-length row.names)  
## Spurious edge  
## v1 v2 p v1\_names v2\_names edge\_type  
## 1 91 370 0.1305131 sp92 SPqe spurious\_edge  
## 2 117 274 0.4815482 sp118 SPy spurious\_edge  
## 3 136 342 0.0300030 sp137 SPod spurious\_edge  
## 4 169 258 0.8931893 sp170 SPi spurious\_edge  
## 5 178 263 0.9825983 sp179 SPn spurious\_edge  
## 6 42 350 0.0000000 sp43 SPwd spurious\_edge  
## 7 49 367 0.0000000 sp50 SPne spurious\_edge  
## 8 90 306 0.0000000 sp91 SPec spurious\_edge  
## 9 98 321 0.0000000 sp99 SPtc spurious\_edge  
## 10 210 285 0.0000000 sp211 SPjb spurious\_edge  
## Fold 2   
## Documented  
## v1 v2 p v1\_names v2\_names edge\_type  
## 1 22 282 0.9804980 sp23 SPgb documented  
## 2 55 326 0.9858986 sp56 SPyc documented  
## 3 61 287 0.9893989 sp62 SPlb documented  
## 4 108 293 0.9871987 sp109 SPrb documented  
## 5 131 345 0.9694969 sp132 SPrd documented  
## 6 225 308 0.9828983 sp226 SPgc documented  
## 7 231 309 0.9897990 sp232 SPhc documented  
## 8 236 279 0.9899990 sp237 SPdb documented  
## 9 240 310 0.9796980 sp241 SPic documented  
## Held out  
## v1 v2 p v1\_names v2\_names edge\_type  
## 1 35 321 0.9863986 sp36 SPtc held\_out  
## 2 36 321 0.9850985 sp37 SPtc held\_out  
## 3 202 364 0.9434943 sp203 SPke held\_out  
## 4 203 364 0.9386939 sp204 SPke held\_out  
## 5 213 307 0.9760976 sp214 SPfc held\_out  
## Spurious edge  
## v1 v2 p v1\_names v2\_names edge\_type  
## 1 42 350 0.5653565 sp43 SPwd spurious\_edge  
## 2 89 340 0.4190419 sp90 SPmd spurious\_edge  
## 3 91 370 0.3929393 sp92 SPqe spurious\_edge  
## 4 158 328 0.9294929 sp159 SPad spurious\_edge  
## 5 169 258 0.8361836 sp170 SPi spurious\_edge  
## 6 49 367 0.0000000 sp50 SPne spurious\_edge  
## 7 90 306 0.0000000 sp91 SPec spurious\_edge  
## 8 117 274 0.0000000 sp118 SPy spurious\_edge  
## 9 136 342 0.0000000 sp137 SPod spurious\_edge  
## 10 194 291 0.0000000 sp195 SPpb spurious\_edge  
## Fold 3   
## Documented  
## v1 v2 p v1\_names v2\_names edge\_type  
## 1 36 284 0.9896990 sp37 SPib documented  
## 2 71 330 0.8933893 sp72 SPcd documented  
## 3 118 342 0.9614961 sp119 SPod documented  
## 4 188 276 0.9887989 sp189 SPab documented  
## 5 219 368 0.9855986 sp220 SPoe documented  
## Held out  
## v1 v2 p v1\_names v2\_names edge\_type  
## 1 15 282 0.9858986 sp16 SPgb held\_out  
## 2 73 268 0.9891989 sp74 SPs held\_out  
## 3 117 295 0.9847985 sp118 SPtb held\_out  
## Spurious edge  
## v1 v2 p v1\_names v2\_names edge\_type  
## 1 42 350 0 sp43 SPwd spurious\_edge  
## 2 49 367 0 sp50 SPne spurious\_edge  
## 3 90 306 0 sp91 SPec spurious\_edge  
## 4 91 370 0 sp92 SPqe spurious\_edge  
## Fold 4   
## Documented  
## v1 v2 p v1\_names v2\_names edge\_type  
## 1 14 315 0.9898990 sp15 SPnc documented  
## 2 114 341 0.9879988 sp115 SPnd documented  
## 3 235 310 0.9814981 sp236 SPic documented  
## Held out  
## v1 v2 p v1\_names v2\_names edge\_type  
## 1 118 295 0.959796 sp119 SPtb held\_out  
## 2 212 307 0.989699 sp213 SPfc held\_out  
## Spurious edge  
## v1 v2 p v1\_names v2\_names edge\_type  
## 1 42 350 0.08480848 sp43 SPwd spurious\_edge  
## 2 49 367 0.83718372 sp50 SPne spurious\_edge  
## 3 91 370 0.61836184 sp92 SPqe spurious\_edge  
## 4 98 321 0.59975998 sp99 SPtc spurious\_edge  
## 5 136 342 0.45014501 sp137 SPod spurious\_edge  
## 6 169 258 0.92229223 sp170 SPi spurious\_edge  
## 7 210 285 0.11061106 sp211 SPjb spurious\_edge  
## Fold 5   
## Documented  
## v1 v2 p v1\_names v2\_names edge\_type  
## 1 5 313 0.9824982 sp6 SPlc documented  
## 2 63 288 0.9745975 sp64 SPmb documented  
## 3 109 340 0.9860986 sp110 SPmd documented  
## 4 114 294 0.9831983 sp115 SPsb documented  
## 5 248 311 0.9806981 sp249 SPjc documented  
## Held out  
## [1] v1 v2 p v1\_names v2\_names edge\_type  
## <0 rows> (or 0-length row.names)  
## Spurious edge  
## v1 v2 p v1\_names v2\_names edge\_type  
## 1 49 367 0.10341034 sp50 SPne spurious\_edge  
## 2 77 330 0.97669767 sp78 SPcd spurious\_edge  
## 3 90 306 0.70417042 sp91 SPec spurious\_edge  
## 4 98 321 0.46124612 sp99 SPtc spurious\_edge  
## 5 117 274 0.78677868 sp118 SPy spurious\_edge  
## 6 136 342 0.77477748 sp137 SPod spurious\_edge  
## 7 158 328 0.71037104 sp159 SPad spurious\_edge  
## 8 194 291 0.62006201 sp195 SPpb spurious\_edge  
## 9 210 285 0.02960296 sp211 SPjb spurious\_edge  
## 10 42 350 0.00000000 sp43 SPwd spurious\_edge  
## 11 91 370 0.00000000 sp92 SPqe spurious\_edge

# Network Reconstruction

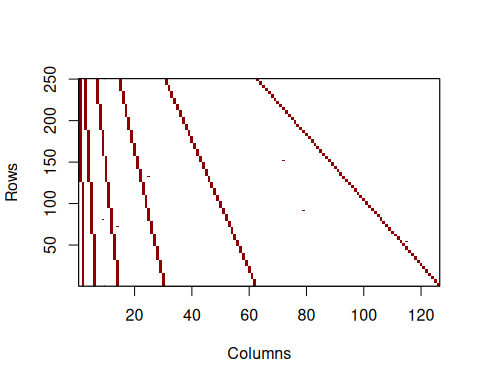
The hsbm.reconstructed function generates a reconstructed binary interaction matrix by combining predictions from all folds. The predicted matrix is binarized using a user-specified threshold.

# Network reconstruction  
myReconst <- hsbm.reconstructed(  
 myPred, # Predictions processed by hsbm.predict  
 rm\_documented=FALSE,  
 threshold = 0.5, # Binarization threshold  
 new\_matrix\_method = "average\_thresholded", # Combine fold predictions  
 spurious\_edges = TRUE  
)

## Warning in get\_hsbm\_results(hsbm\_out, input\_names = TRUE, na\_treatment =  
## na\_treatment): Predictions obtained for 13.15% of the links. Consider  
## increasing the number of iterations.

This output includes the final averaged probability matrix, the final reconstructed binary matrix, and evaluation metrics. These results highlight the predicted interactions, showcasing the method’s ability to detect spurious and missing links effectively. Let’s explore some of the key outputs.

# Visualize the reconstructed binary matrix  
plot\_interaction\_matrix(myReconst$new\_mat, order\_mat = FALSE)



# View the reconstruction summary and evaluation metrics  
summary(myReconst)

## $`Reconstructed Network Metrics`  
## obs\_links unobs\_links pred\_links kept\_links spurious\_links missing\_links  
## 1 1515 29985 1509 1509 6 0  
##   
## $`Evaluation Metrics`  
## mean\_RLRR mean\_auc mean\_aucpr mean\_yPRC mean\_prec mean\_sens mean\_spec  
## 1 0.9960396 0.9989107 1.039703 0.04809524 1 0.9968317 1  
## mean\_ACC mean\_ERR mean\_tss  
## 1 0.9998476 0.000152381 0.9968317

The summary provides the number of spurious and missing links, as well as key evaluation metrics, such as the retained link recovery rate (RLRR).

Let’s now examine the top 10 predicted links that are most likely to be spurious (false positives) by visualizing the probabilities of “documented” links.

# Visualize the top most likely spurious links  
# top\_links\_spurious <- top\_links(hsbm\_out, n = 10, edge\_type = "documented")  
# print(top\_links\_spurious)

This document demonstrates the use of the *sabinaHSBM* package for network reconstruction. By applying the full reconstruction method, we showcased how spurious and missing links can be effectively identified and addressed in complex networks.

# Computing characteristics

Show processing time and computer characteristics

end\_time <- Sys.time()  
cat("The processing time of this script took: ", end\_time - start\_time, "mins\n")

## The processing time of this script took: 2.23731 mins

This analysis was performed on a Dynabook with the following characteristics: - Processor (CPU): Intel Core i7-1165G7 @ 2.80GHz (11th Gen) - Memory (RAM): 32 GB - Operating System: Windows 11 Pro, Version 24H2 - R Version: 4.3.3