Macquarie_Island_case_study

This tutorial is derived from the document *Tutorial-2-Macquarie_Island_case_study* written in Python environment by Kristensen *et.al.* (2019) (https://github.com/nadiahpk/qualitative-modelling/tree/master/tutorials). Using the same data, this document aims to give a better illustration of the similarities and the slight differences between the original qualitative modeling code written in Python and this R version.

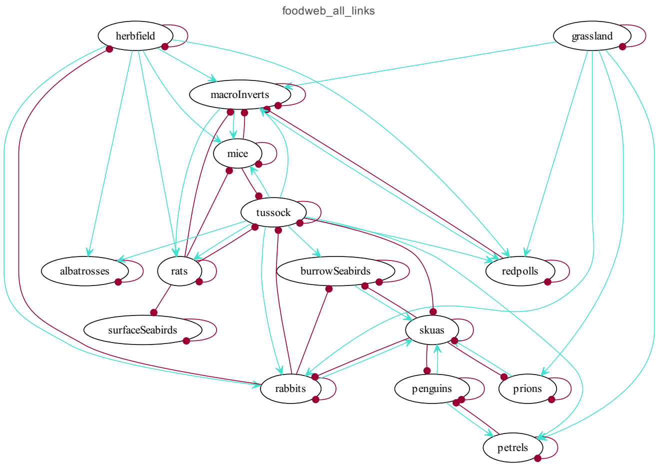
1. Define the species interaction network

Macquarie Island interaction network si defined by the species present, the interactions between them, and the signs of these interactions.

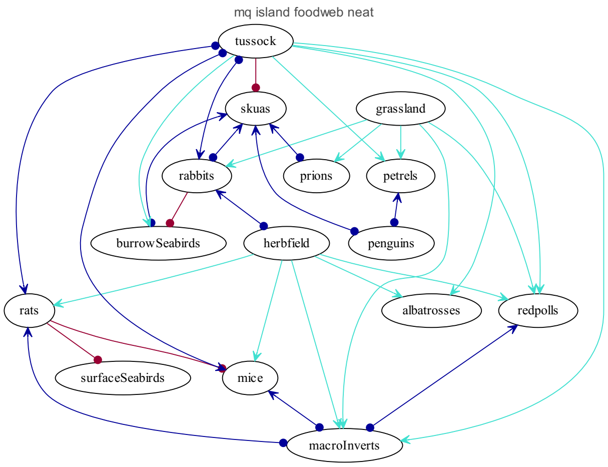
```
sppList = c(
    'albatrosses',
    'prions',
    'burrowSeabirds',
    'petrels',
    'herbfield',
    'macroInverts',
    'mice',
    'penguins',
    'rabbits',
    'rats',
    'redpolls',
    'skuas',
    'surfaceSeabirds',
    'tussock'
)

positive_edges_list = list(
    'prions'= c('grassland'),
    'skuas'= c('prions','burrowSeabirds','rabbits','penguins'),
    'petrels'= c('penguins','tussock','grassland'),
    'mice'= c('herbfield','macroInverts','tussock'),
    'rats'= c('macroInverts','herbfield','tussock'),
    'burrowSeabirds'= c('tussock'),
    'rabbits'= c('tussock','herbfield','grassland'),
    'macroInverts'= c('herbfield','grassland','tussock'),
    'albatrosses'= c('tussock','herbfield'),
    'redpolls'= c('macroInverts','tussock','herbfield','grassland')
)

negative_edges_list = list(
    'prions'= c('prions','skuas'),
    'skuas'= c('skuas','tussock'),
    'penguins'= c('penguins','skuas','petrels'),
    'petrels'= c('petrels'),
    'mice'= c('mice','rats'),
    'rats'= c('rats'),
    'burrowSeabirds'= c('burrowSeabirds','skuas','rabbits'),
    'rabbits'= c('rabbits','skuas'),
    'surfaceSeabirds'= c('surfaceSeabirds','rats'),
    'macroInverts'= c('macroInverts','rats','mice','redpolls'),
    'tussock'= c('tussock','mice','rats','rabbits'),
    'albatrosses'= c('albatrosses'),
    'herbfield'= c('herbfield','rabbits'),
    'grassland'= c('grassland'),
    'redpolls'= c('redpolls')
)
```

The function initialise_foodweb returns a DiagrammeR graph obect, storing information of the network structure (nodes and edges) as dataframes (i.e. NDF: node data frame, and EDF: edge data frame). The network can be quickly plotted using function *render_graph()* from package *DiagrammeR* for a quick check of network structures. For a neat plot, use funciton *foodweb_neat_plot()*.

```
web <- initialise_foodweb(positive_edges_list, negative_edges_list)
render_graph(web, title = "foodweb_all_links")
```



```
foodweb_neat_plot(web, title = "mq island foodweb neat")
```



Using the function *qualitative_community_matrix*, we can convert the interaction network into a qualitative community matrix (Mq), and get two named vectors *labelToIndex* and *indexToLabel* to map species labels (i.e. names) to indices of matrix and vice versa.

```
outputQM <- qualitative_community_matrix(web)
Mq <- outputQM$Mq
Mq
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
##  [1,]   -1    0    0    1    0    0    0    0    0     0     0     0     0
##  [2,]    0   -1    0    0    0    0    0    0    0    -1     0     0    -1
##  [3,]    0    0   -1    0    0    0    0    0    0     0     0     0     0
##  [4,]    0    0    0   -1    0    0    0    0    0    -1     0     0     0
##  [5,]    0    0    1    1   -1   -1    0    0    0     0    -1    -1     0
##  [6,]    0    0    0    1    1   -1    0    0    0     0    -1     0     0
##  [7,]    0    0    0    0    0    0   -1   -1    0     0     0     0    -1
##  [8,]    0    0    1    0    0    0    1   -1    0     0     0     0     0
##  [9,]    0    0    1    0    0    0    0    0   -1     0     0     0    -1
## [10,]    0    0    1    1    0    0    0    0    0    -1     0     0    -1
## [11,]    0    0    0    1    1    0    0    0    0     0    -1     0     0
## [12,]    0    0    1    1    1    0    0    0    0     0     0    -1     0
## [13,]    0    1    0    0    0    0    1    0    1     1     0     0    -1
## [14,]    0    0    0    0    0    0    0    0    0     0    -1     0     0
## [15,]    0    0    0    0    0   -1    0    0    0    -1    -1     0     0
##       [,14] [,15]
##  [1,]     0     1
##  [2,]     0     1
##  [3,]     0     0
##  [4,]     0     0
##  [5,]     0     1
##  [6,]     0     1
##  [7,]     0     0
##  [8,]     0     1
##  [9,]     0     0
## [10,]     0     1
## [11,]     0     1
## [12,]     0     1
## [13,]     0    -1
## [14,]    -1     0
## [15,]     0    -1
```

```
labelToIndex <- outputQM$labelToIndex
indexToLabel <- outputQM$indexToLabel

unname(indexToLabel[10])
```

```
## [1] "rabbits"
```

```
unname(labelToIndex["rabbits"])
```

```
## [1] 10
```

We encode the validation criteria and then use the function *get_conditions_df* to get a dataframe with three columns: species' names (label), their corresponding conditions and their corresponding indices in the qualitative community matrix. We also get the indices of the controled species and the indices of a list of species (*sppList*) that we want to monitor.

```
# response to increase in rabbits
validation <- list(
    "rabbits" = 1,
    "tussock"= -1)

condn_df <- get_condition_df(labelToIndex,validation)
condn_df
```

```
##   conditions speciesNames idx
## 1          1      rabbits  10
## 2         -1      tussock  15
```

```
control_list <- c("rabbits")
control_list_idx <- unname(labelToIndex[control_list])

sppList_idx <- unname(labelToIndex[sppList])
```

2. Probabilistic approach: perform Monte Carlo Simulation.

We run 10^3 times of simulations.

(The original tutorial does not perform the Monte Carlo simulation to this specific case study.)

```
set.seed(178)

noSim <- 1000
collectedResponses = list()

sz <- dim(Mq)
n <- length(Mq)

start_time <- Sys.time()
```

```
for (i in 1:noSim){

    valid <- FALSE

    while (!valid) {

      # find a random community matrix that is stable
      maxEig = 1

      while (maxEig > 0) {

          M = matrix(runif(n), sz[1], sz[2]) * Mq
          maxEig <- max(Re(eigen(M, symmetric=FALSE, only.values=TRUE)$values))
      }

      # Now have a valid stable matrix
      # find the sensitivity matrix
      Sq <- -solve(M)

      # check validation criteria
      valid <- all(sign(Sq[condn_df$idx, unname(labelToIndex["rabbits"])]) == condn_df$conditions)
    }

    #Now have a valid stable community matrix
    response <- vector()

    for (ps in control_list_idx) {

        resp <- ifelse(Sq[sppList_idx, ps] < 0, "neg",
                       ifelse(Sq[sppList_idx, ps] >0, "pos", "zer"))

        response <- append(response, resp)
    }

    collectedResponses[[i]] <- response

}

end_time <- Sys.time()
time_elapsed = end_time - start_time
print(time_elapsed)
```

```
## Time difference of 0.3070071 secs
```

Convert the collected list of species responses into a dataframe.

```
df_responses <- do.call(rbind, collectedResponses) %>% as.data.frame() %>% mutate_if(is.factor, as.chara

# short labels
colnames <- unlist(lapply(control_list, function(x) paste0(str_sub(x, 1, 3), "_", str_sub(sppList, 1, 3

# full labels
# colnames <- unlist(lapply(control_list, function(x) paste0(x, "_", sppList)))
```

```
colnames(df_responses) <- colnames
head(df_responses)
```

```
##   rab_alb rab_pri rab_bur rab_pet rab_her rab_mac rab_mic rab_pen rab_rab
## 1     neg     neg     neg     neg     neg     pos     pos     neg     pos
## 2     neg     pos     neg     pos     neg     pos     neg     pos     pos
## 3     neg     pos     neg     pos     neg     neg     pos     pos     pos
## 4     neg     neg     neg     neg     neg     pos     pos     pos     pos
## 5     neg     neg     neg     neg     neg     pos     pos     pos     pos
## 6     neg     pos     neg     neg     neg     pos     neg     pos     pos
##   rab_rat rab_red rab_sku rab_sur rab_tus
## 1     neg     neg     pos     pos     neg
## 2     neg     neg     neg     pos     neg
## 3     neg     neg     neg     pos     neg
## 4     neg     neg     pos     pos     neg
## 5     neg     neg     pos     pos     neg
## 6     pos     pos     neg     neg     neg
```

Aggregate simulation outcomes.

```
levels <- c('pos', 'neg')
df_responses[] <-  lapply(df_responses, factor, levels=levels)

count <- sapply(df_responses, table)
count
```

```
##     rab_alb rab_pri rab_bur rab_pet rab_her rab_mac rab_mic rab_pen
## pos       0     436       0     183       0     838     486     818
## neg    1000     564    1000     817    1000     162     514     182
##     rab_rab rab_rat rab_red rab_sku rab_sur rab_tus
## pos    1000     193     102     564     807       0
## neg       0     807     898     436     193    1000
```

3.The Boolean approach

3.1 Parameter sweep

To save time in this tutorial, we hard-coded the number of species-response combinations to be found, which is 36. If the number is not known beforehand, you can specify the number of matrices to generate using the commented-out *for* loop code.

```
collectedResponses = list()
sz <- dim(Mq)
n <- length(Mq)

start_time <- Sys.time()

while (length(collectedResponses) < 36) {

# noSim <- 1000000
# for (i in 1:noSim){

    valid <- FALSE

    # find a random community matrix that is stable
    while (!valid) {
        # find a random community matrix that is stable
        maxEig = 1

        while (maxEig > 0) {

          M = matrix(runif(n), sz[1], sz[2]) * Mq
          maxEig <- max(Re(eigen(M, symmetric=FALSE, only.values=TRUE)$values))
        }

        # Now have a valid stable matrix
        # find the sensitivity matrix
        Sq <- -solve(M)

        # check validation criteria
        valid <- all(sign(Sq[condn_df$idx, unname(labelToIndex["rabbits"])]) == condn_df$conditions)
    }

    #Now have a valid stable community matrix
    response <- vector()

    for (ps in control_list_idx) {

        resp <- ifelse(Sq[sppList_idx, ps] < 0, "neg",
                       ifelse(Sq[sppList_idx, ps] >0, "pos", "zer"))

        response <- append(response, resp)
    }

    if (!any(collectedResponses %in% list(response))) {

        collectedResponses[[length(collectedResponses)+1]] <- response
    }
}

end_time <- Sys.time()
time_elapsed = end_time - start_time
print(time_elapsed)
```

```
## Time difference of 0.231775 secs
```

```
length(collectedResponses)
```

```
## [1] 36
```

Write the parameter-sweep results (i.e. unobserved species-response combinations) into a csv file.

```
df_responses <- do.call(rbind, collectedResponses)
colnames <- unlist(lapply(control_list, function(x) paste0(x, "_", sppList))) # full labels
colnames(df_responses) <- colnames

fileNames <- unlist(lapply(control_list, function(x) paste0("uniques_web1_", x, ".csv")))
write.csv(df_responses, file = fileNames, row.names=FALSE)
```

3.2 Boolean analysis

Read in the responses from the csv file that was written previously.

```
df_mq <- read.csv("uniques_web1_rabbits.csv", head = TRUE)
# head(df_mq)

str4true = 'pos'
str4flase = 'neg'

allResponse <- colnames(df_mq);
allResponse
```

```
##  [1] "rabbits_albatrosses"     "rabbits_prions"
##  [3] "rabbits_burrowSeabirds"  "rabbits_petrels"
##  [5] "rabbits_herbfield"       "rabbits_macroInverts"
##  [7] "rabbits_mice"            "rabbits_penguins"
##  [9] "rabbits_rabbits"         "rabbits_rats"
## [11] "rabbits_redpolls"        "rabbits_skuas"
## [13] "rabbits_surfaceSeabirds" "rabbits_tussock"
```

We write a list of desired responses, *desiredResponses*. The *desiredResponses* list is then converted into a Boolean mask *desiredResponsesMask*, and passed to the function *getUnobservedInts2*.

Function *getUnobservedInts* finds the complement of the set of observed responses (i.e. unobserved response combinations), returning a list of unobserved responses as a list of integers.

```
desiredResponses = c(
    'rabbits_albatrosses',
    'rabbits_prions',
    'rabbits_burrowSeabirds',
    'rabbits_petrels',
    'rabbits_herbfield',
    'rabbits_macroInverts',
    'rabbits_mice',
    'rabbits_penguins',
    'rabbits_rats',
    'rabbits_redpolls',
    'rabbits_skuas',
    'rabbits_surfaceSeabirds')

boolLen = length(desiredResponses)

desiredResponsesMask <- which(allResponse %in% desiredResponses)

unobservedInts <- getUnobservedInts(df_mq, desiredResponsesMask, boolLen, str4true)

length(unobservedInts)
```

```
## [1] 4060
```

The function *getUnobservedBooldf* turns the list of integers *unobservedInts*, corresponding to unobserved species responses, back into a dataframe in Boolean expression (1s and 0s). The function *getUnobservedBooldf* also add an new column *unob* which is a vector of 1s (the dataframe therefore becomes a truth table) to allow funciton *logicopt()* in Package *LogicOpt* to perform Boolean minimization on the dataframe.

```
unobservedBooldf <- getUnobservedBooldf(unobservedInts, desiredResponses)
# unobservedBooldf[50:55, 7:13] # check
```

The function*logicopt()* in Package *LogicOpt* is used to perform the Boolean minimisation.

```
start_time <- Sys.time()

opt <- logicopt(unobservedBooldf, boolLen, 1, mode="espresso")
optEqn <- tt2eqn(opt[[1]], boolLen, 1)
# optEqn   # show the ON set equations for the minimized truth table

end_time <- Sys.time()
time_elapsed = end_time - start_time
print(time_elapsed)
```

```
## Time difference of 0.0538609 secs
```

The function *getPCUList* converts the optimized equations (*optEqn_c*) into a list of strings (PCUs) for further analysis.

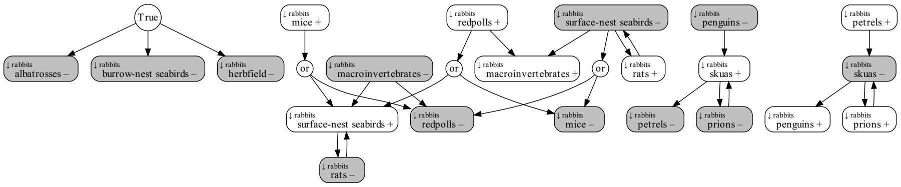
```
PCUList <- getPCUList(optEqn, str4true, str4flase, desiredResponses)
PCUList
```

```
## [[1]]
## [1] "posrabbits_albatrosses"
##
## [[2]]
## [1] "posrabbits_burrowSeabirds"
##
## [[3]]
## [1] "posrabbits_herbfield"
##
## [[4]]
## [1] "negrabbits_macroInverts"    "negrabbits_surfaceSeabirds"
##
## [[5]]
## [1] "negrabbits_penguins" "negrabbits_skuas"
##
## [[6]]
## [1] "posrabbits_petrels" "posrabbits_skuas"
##
## [[7]]
## [1] "negrabbits_prions" "negrabbits_skuas"
##
## [[8]]
## [1] "posrabbits_prions" "posrabbits_skuas"
##
## [[9]]
## [1] "negrabbits_rats"            "negrabbits_surfaceSeabirds"
##
## [[10]]
## [1] "posrabbits_rats"            "posrabbits_surfaceSeabirds"
##
## [[11]]
## [1] "negrabbits_macroInverts" "posrabbits_redpolls"
##
## [[12]]
## [1] "posrabbits_mice"            "posrabbits_redpolls"
## [3] "negrabbits_surfaceSeabirds"
```

```
niceNames = rbind(
        c('rabbits', 'rabbits'),
        c('petrels', 'petrels'),
        c('mice', 'mice'),
        c('burrowSeabirds', 'burrow-nest seabirds'),
        c('macroInverts', 'macroinvertebrates'),
        c('herbfield', 'herbfield'),
        c('redpolls', 'redpolls'),
        c('skuas', 'skuas'),
        c('rats', 'rats'),
        c('surfaceSeabirds', 'surface-nest seabirds'),
        c('penguins', 'penguins'),
        c('prions', 'prions'),
        c('albatrosses', 'albatrosses'),
        c('tussock', 'tussock')
)

niceNames <- setNames(niceNames[ ,2], niceNames[ ,1]) #create a named vector as a dictionary
```

PCULists are used as inputs to the functions *get_edgelist_singleAnte* , which get an edgelist in a single-antecedent form. Then the edgelist will be passed to *draw_implication_network* function to plot the corresponding implication network.

```
edgelist <- get_edgelist_singleAnte(PCUList)
# edgelist
draw_implication_network(edgelist, niceNames)
```


True
↓ rabbits albatrosses –
↓ rabbits burrow-nest seabirds –
↓ rabbits herbfield –
↓ rabbits mice +
or
↓ rabbits macroinvertebrates –
↓ rabbits surface-nest seabirds +
↓ rabbits rats –
↓ rabbits redpolls +
or
↓ rabbits redpolls –
↓ rabbits surface-nest seabirds –
↓ rabbits macroinvertebrates +
or
↓ rabbits rats +
↓ rabbits mice –
↓ rabbits penguins –
↓ rabbits skuas +
↓ rabbits petrels –
↓ rabbits prions –
↓ rabbits petrels +
↓ rabbits skuas –
↓ rabbits penguins +
↓ rabbits prions +