Toolbox Food Webs Analysis

Léo Ledru

2024-04-18

Introduction

Ce document RMarkdown est dédié à l'analyse de réseaux trophiques, en particulier l'étude des effets indirects entre les espèces du réseau et le processus de cascade trophique.

- Le chargement et la vérification d'un réseau trophique
- La visualisation d'un réseau trophique

```
setwd("~/Documents/Post_Doc_CARRTEL/R/Toolbox")
library(rmarkdown)
library(shiny)
```

Loading food web and verifications

Le food web doit être une matrice carrée, puisqu'en lignes et en colonnes il doit y avoir les mêmes espèces, et rangées dans le même ordre. Chaque élément de la matrice doit être un numeric. En dehors de la diagonale (qui peut contenir des valeurs différentes de zéro), toute valeur négative doit avoir une valeur positive à sa symétrie (pouvant être différente néanmoins) ; en effet, il doit bien s'agir d'un food web avec uniquement des interactions trophiques, pas de compétition directe, ni commensalisme, ni mutualisme etc.

```
#load("MyFoodWeb.Rdata")
#load("MyFoodWebLake.Rdata")
load("StableFwbs_TL3.Rdata")
MyFoodWeb <- StableFwbs_TL3[[5097]][["A"]]

CheckInit <- function(MyFoodWeb){
  if (!is.matrix(MyFoodWeb)){
    stop("Erreur : la variable n'est pas une matrice")
  }
  if (!all(is.numeric(MyFoodWeb))){
    stop("Erreur : la matrice doit contenir uniquement des nombres")
  }
  if (nrow(MyFoodWeb) != ncol(MyFoodWeb)){
    stop("Erreur : la matrice n'est pas carrée")
  }
  # Diagonale must be -1, because interactions are relative to self-regulation
  if (!all(diag(MyFoodWeb)==-1)){
    stop("Erreur : toutes les valeurs diagonales doivent valoir -1")
  }
  # Vérification de la symétrie positif-négatif (uniquement liens trophiques)
  for (i in 1:nrow(MyFoodWeb)){
    for (j in 1:ncol(MyFoodWeb)){
      if (i != j){
```

```
        if ((MyFoodWeb[i, j] < 0 && MyFoodWeb[j, i] <= 0) || (MyFoodWeb[i, j] > 0 && MyFoodWeb[j, i] >=
          stop(paste("Erreur : Les valeurs négatives (positives) hors-diagonale doivent avoir une valeu
        }
      }
    }
  }
  # Vérification de l'absence d'espèce non-connectée : si existence alors suppression du réseau
  FoodWebBis <- MyFoodWeb
  diag(FoodWebBis) <- 0 # remove self-reg to test if each species is connected with at least another on
  IdxUnconnected <- which(colSums(FoodWebBis) == 0)
  MyFoodWeb <- MyFoodWeb[-IdxUnconnected,-IdxUnconnected]
  print(paste0("Species ", paste(IdxUnconnected, collapse = ", "), " are removed because unconnected"))
  return(MyFoodWeb)
}

#MyFoodWeb <- matrix(c(1,2,-3,1), nrow = 2, ncol = 2)
MyFoodWeb <- CheckInit(MyFoodWeb)
```

```
## [1] "Species 22, 23 are removed because unconnected"
```

Identification et visualisation des chaînes trophiques

Analyse de MyFoodWeb afin de référencer toutes les chaînes trophiques (TL max 3 ou 4, from top to n-2 et top to basal) en notant idx de prédateurs, consos inters et ressources. Mesure d'une cascade n-step ou non sur ces chaînes (si l'omnivorie prive la cascade n-step). Mesure omnivorie globale du réseau ainsi que collectivité, connectance.

```
library(NetIndices) # to compute trophic levels and omnivory
```

```
## Le chargement a nécessité le package : MASS
```

```
library(sparsevar) # for spectralRadius function
library(expm) # for %^% operator
```

```
## Le chargement a nécessité le package : Matrix
```

```
##
## Attachement du package : 'expm'
```

```
## L'objet suivant est masqué depuis 'package:Matrix':
##
##     expm
```

```
library(purrr) # is_empty function

# Analysis fo the food web
## outputs : a list with some food web metrics (collectivity, connectance, mean omnivory) and a datafra
source("FoodWebAnalysis.R")
#debug(FoodWebAnalysis)
FoodWebMetrics <- FoodWebAnalysis(MyFoodWeb)
```

```
library(igraph) # to show food web as a graph
```

```
##
## Attachement du package : 'igraph'
```

```
## Les objets suivants sont masqués depuis 'package:purrr':
##
```

```
##     compose, simplify

## Les objets suivants sont masqués depuis 'package:stats':
##
##     decompose, spectrum

## L'objet suivant est masqué depuis 'package:base':
##
##     union
```

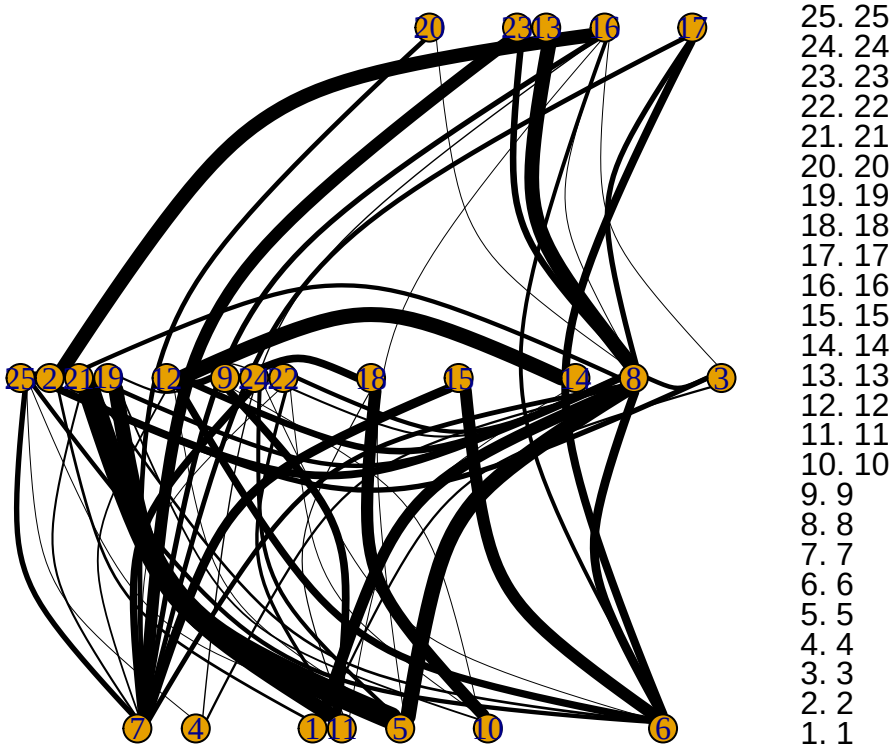
```r
library(RColorBrewer)

# Visualization
## outputs : a graph representing the food web, nodes are species and black curves are trophic links. I
source("FoodWebGraph.R")
#debug(FoodWebGraph)
Graph <- FoodWebGraph(MyFoodWeb, IdxFocusSpecies = NULL)

par(mar = c(1, 1, 1, 1)) # adjust margins
width <- 8  # Ajustez la largeur selon vos besoins
height <- 6  # Ajustez la hauteur selon vos besoins
options(repr.plot.width = width, repr.plot.height = height)

scale <- 5 # to adjust edges width
plot(Graph, vertex.size = 8,
    edge.width = as.numeric(E(Graph)$weight)*scale,
    edge.curved = 0.5, edge.color = E(Graph)$col)

# Add names of species for each node, if any
Names <- rownames(MyFoodWeb)
if (is.null(Names)){
  Names <- seq(1, nrow(MyFoodWeb))
}
text(x = max(par("usr")[1], par("usr")[2]) - 0.5,
     y = seq(min(par("usr")[3], par("usr")[4]) + 0.1,
             max(par("usr")[3], par("usr")[4]) - 0.1,
             length.out = length(Names)),
     labels = paste(seq_along(Names), Names, sep = ". "), pos = 4, col = "black", cex = 1)
```



Qui est relié avec qui, pour chaque ordre jusqu'à l'ordre maximal choisi.

```
SpNames <- paste(1:nrow(MyFoodWeb))
# Créer un objet vide avec 10 lignes et 5 colonnes
WhoWithWho <- vector("list", nrow(MyFoodWeb))
# Direct effect without self-reg
I <- diag(1, nrow = nrow(MyFoodWeb))
DirectEffects <- MyFoodWeb + I # Extract the normalized self-regulation
MaxOrder <- 5
# ComputeLinks : for each Sp find all non-null interactions for each order until MaxOrder
# Also find the cumulative orderLim from which each Sp has interacted with all others
ComputeLinks <- function(i, DirectEffects, MaxOrder, SpNames){
  order <- 1
  InnerList <- list()
  A <- DirectEffects
  OrderLim <- NULL
  IdxCumul <- c()
  cond <- TRUE
  while(order <= MaxOrder){
    Idx <- which(A[,i] != 0)
    InnerList[[paste0("order", order)]] <- Idx
    IdxCumul <- c(IdxCumul, Idx)
    if (all(SpNames %in% IdxCumul) & cond){
      OrderLim <- order
      cond <- FALSE
    }
    order <- order + 1
    A <- DirectEffects%^%order
  }
```

```
  InnerList[["orderLim"]] <- OrderLim
  return(InnerList)
}
# Each outer list is a species, which contain as much inner lists as the chosen MaxOrder with the idx o
WhoWithWho <- lapply(seq_along(WhoWithWho), function(i) ComputeLinks(i, DirectEffects, MaxOrder, SpName
SpNames <- rownames(MyFoodWeb)
names(WhoWithWho) <- SpNames

#print(WhoWithWho)
```

```
# Visualization of Inverted chain(s)
## outputs : a graph representing the food web, with links of inverted chain in colors (if any), all ot
library("scales")
```

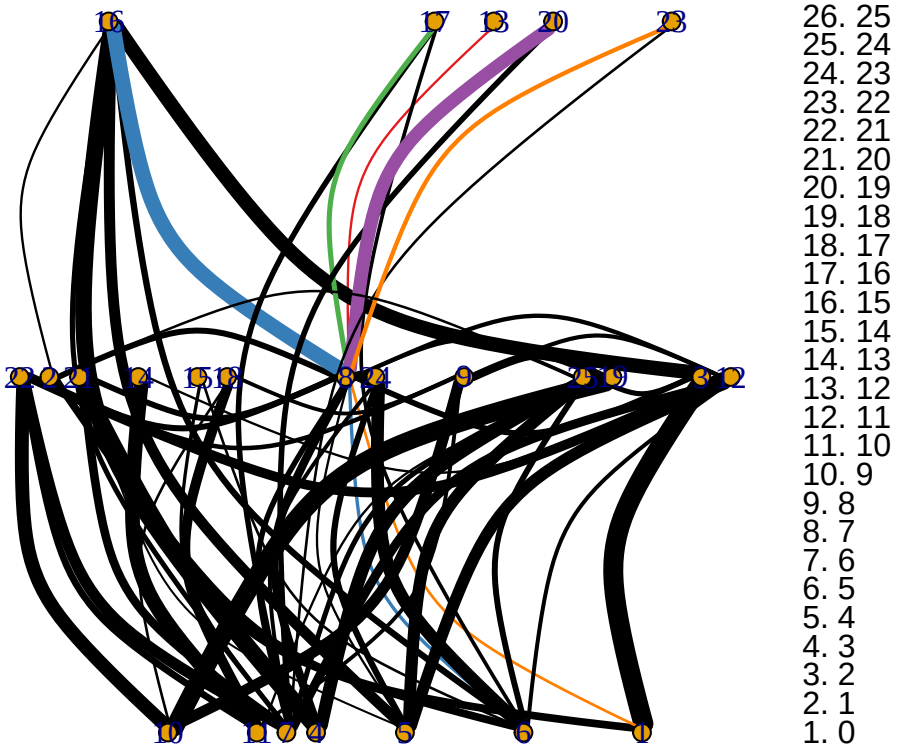
```
##
## Attachement du package : 'scales'
```

```
## L'objet suivant est masqué depuis 'package:purrr':
##
##     discard
```

```
source("InvertedChainGraph.R")
#debug(InvertedChainGraph)
Outputs <- InvertedChainGraph(MyFoodWeb, FoodWebMetrics)
Graph <- Outputs[["Graph"]]
par(mar = c(1, 1, 1, 1)) # adjust margins
width <- 8  # Ajustez la largeur selon vos besoins
height <- 6  # Ajustez la hauteur selon vos besoins
options(repr.plot.width = width, repr.plot.height = height)

scale <- 1 # to adjust edges width
plot(Graph, vertex.size = 5,
    edge.width = as.numeric(E(Graph)$weight)*scale,
    edge.curved = 0.5, edge.color = E(Graph)$col)

# Add names of species for each node, if any
Names <- rownames(MyFoodWeb)
if (is.null(Names)){
  Names <- seq(0, nrow(MyFoodWeb))
}
text(x = max(par("usr")[1], par("usr")[2]) - 0.5,
     y = seq(min(par("usr")[3], par("usr")[4]) + 0.1,
             max(par("usr")[3], par("usr")[4]) - 0.1,
             length.out = length(Names)),
     labels = paste(seq_along(Names), Names, sep = ". "), pos = 4, col = "black", cex = 1)
```



```
# Warning : si plusieurs chaînes avec inversion ont un lien en commun une couleur va se superposer à l'
print(Outputs[["InvertedChains"]])
```

```
## $Tops
## [1] 13 16 17 20 23
##
## $Middles
## [1] 8 8 8 8 8
##
## $Bottoms
## [1] 1
```

Visualisation du réseau dans l'espace des valeurs propres

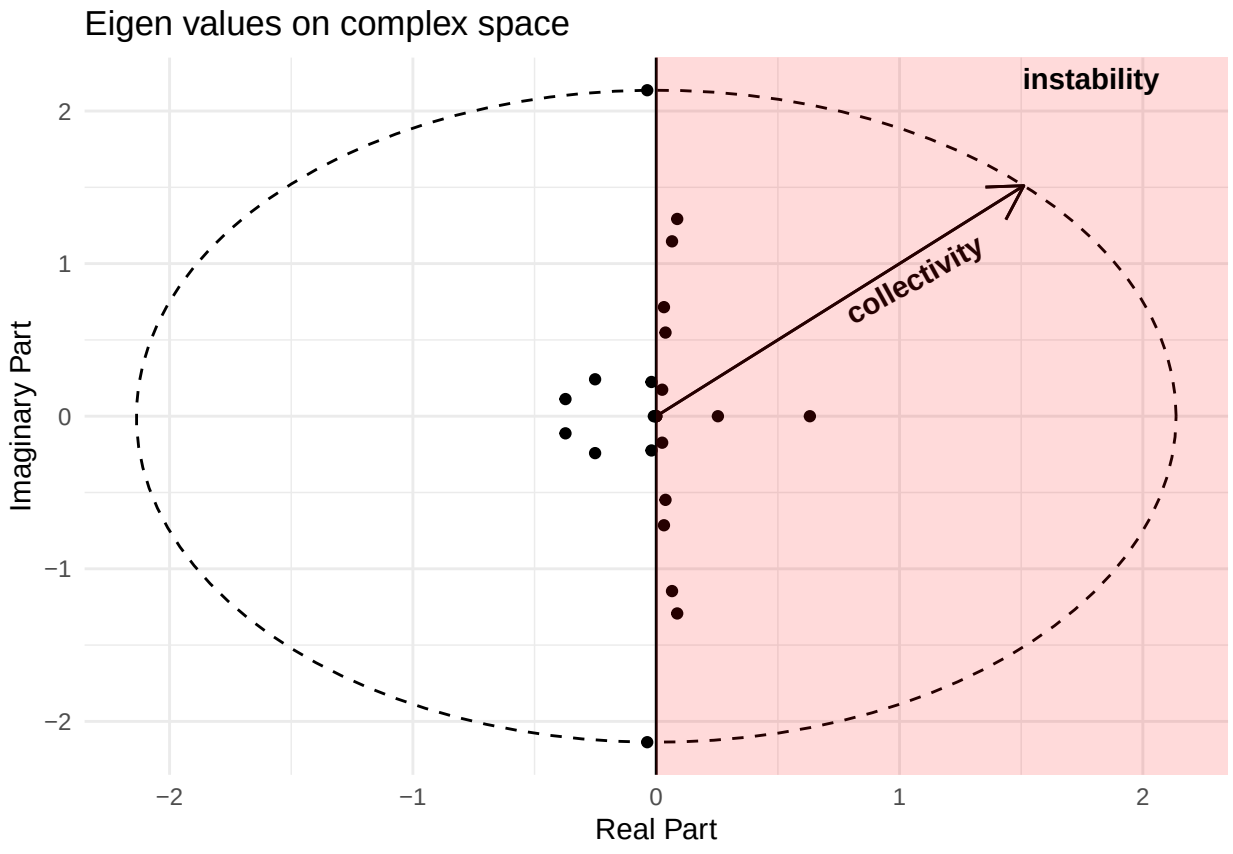
Calculer valeurs propres (avec self-reg normalisée extraite, donc diag = 0) et ploter l'espace des valeurs propres, faire apparaître la collectivité avec le cercle de rayon phi et faire apparaître la limite de stabilité sur l'axe de la partie réelle.

Également un algo pour estimer la self-reg minimale à appliquer pour "stabilité" ?

```
library(ggplot2)

# Compute eigen values, their barycentre and the collectivity
I <- diag(1, nrow = nrow(MyFoodWeb))
ValuesVectors <- eigen(MyFoodWeb + I)
Values <- ValuesVectors[[1]]
Collect <- spectralRadius(MyFoodWeb + I)
Barycentre <- sum(Values) / length(Values)
# organize data
ValuesDf <- data.frame(Re = Re(Values), Im = Im(Values))
theta <- seq(0, 2 * pi, length.out = 100)
```

```r
CollectCircle <- data.frame(Re = Re(Barycentre) + Collect * cos(theta),
                            Im = Im(Barycentre) + Collect * sin(theta))
ArrowData <- data.frame(
  x = c(Re(Barycentre), Re(Barycentre) + Collect * cos(pi/4)),
  y = c(Im(Barycentre), Im(Barycentre) + Collect * sin(pi/4))
)
# plot
ggplot(ValuesDf, aes(x = Re, y = Im)) +
  geom_point() +
  geom_vline(xintercept = 0) +
  geom_path(data = CollectCircle, aes(x = Re, y = Im), color = "black", linetype = "dashed") +
  geom_segment(data = ArrowData,
               aes(x = x[1], y = y[1], xend = x[2], yend = y[2]),
               arrow = arrow(length = unit(0.2, "inches")), color = "black") +
  annotate("text", x = Re(Barycentre) + Collect * cos(pi/4)/2,
           y = Im(Barycentre) + Collect * sin(pi/4)/2, label = "collectivity",
           angle = 29, hjust = 0, vjust = 1.3, fontface = "bold") +
  annotate("rect", xmin = 0, xmax = Inf, ymin = -Inf, ymax = Inf,
           alpha = .15, fill = "red") +
  annotate("text", x = Inf, y = Inf, label = "instability", hjust = 1.5, vjust = 1.5, fontface = "bold")
  labs(x = "Real Part", y = "Imaginary Part") +
  ggtitle("Eigen values on complex space") +
  theme_minimal()
```



Dynamique du réseau

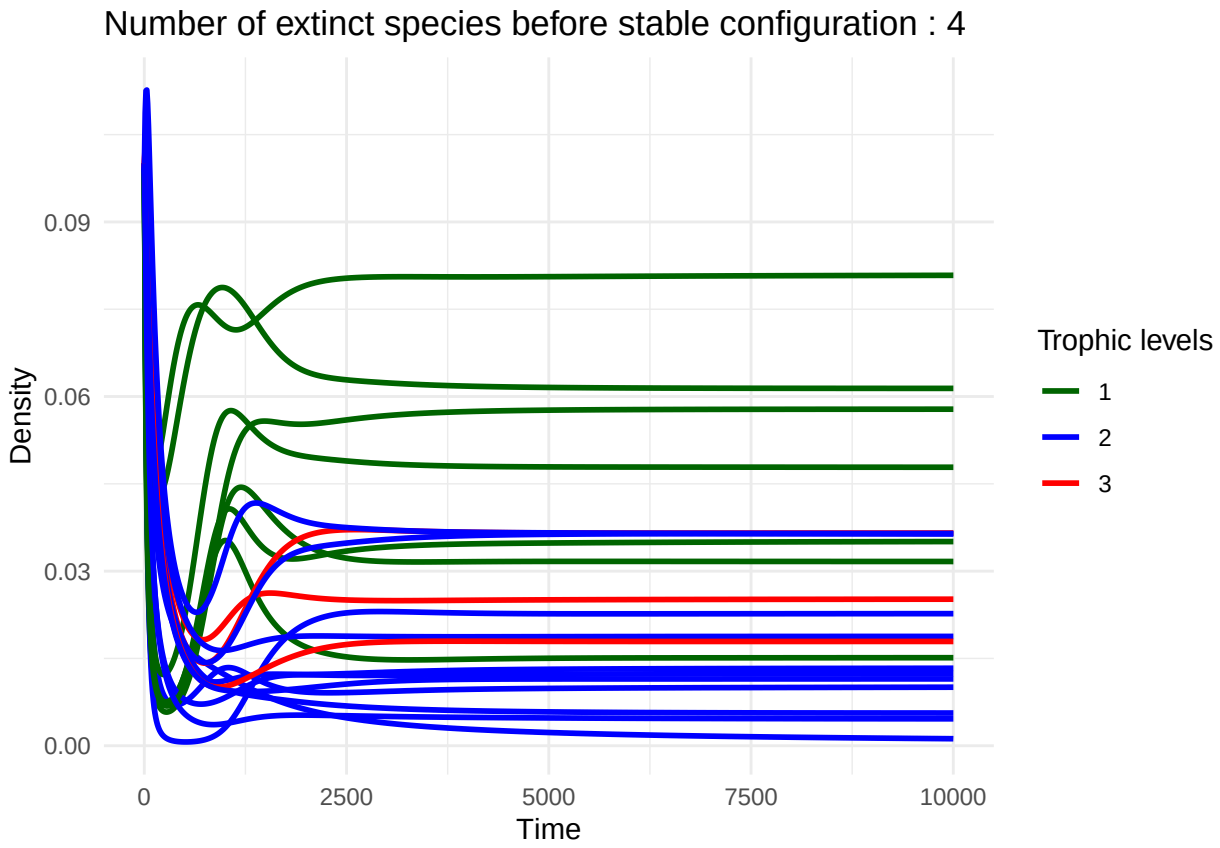
stabilité du réseau

Commence par tester si le réseau est stable : simulations successives en retirant les espèces éteintes à chaque fois tant qu'il y en a.

```
library(deSolve)

# Apply a Generalized Lotka-Volterra on the food web until a stable configuration (by removing extinct
## outputs : 1) temporal densities of the stable run after incremental removal of extinct species, if a
source("FoodWebEquilibrium.R")

# Self-regulation (normally -1 because of prior normalization)
diag(MyFoodWeb) <- -1
# Run
Tmax <- 1000
Tstep <- 0.1
GrowthRate <- 0.1
DeathRate <- 0.01
Outputs <- FoodWebEquilibrium(MyFoodWeb, Tmax, Tstep, GrowthRate, DeathRate)
# Visualization
Dynamics <- Outputs[["Dynamics"]]
if (max(Dynamics$TrophLevel) == 3){
  colors <- c("darkgreen", "blue", "red")
}else{ # maximum trophic level = 4
  colors <- c("darkgreen", "blue", "red", "black")
}
DynamicsPlot <- ggplot(Dynamics) +
                geom_line(aes(x = Time, y = Density, color = factor(TrophLevel), group = factor(IdxSpec
                labs(title = paste0("Number of extinct species before stable configuration : ", Outputs
                scale_color_manual(values = colors) +
                theme_minimal()
DynamicsPlot # show plot
```



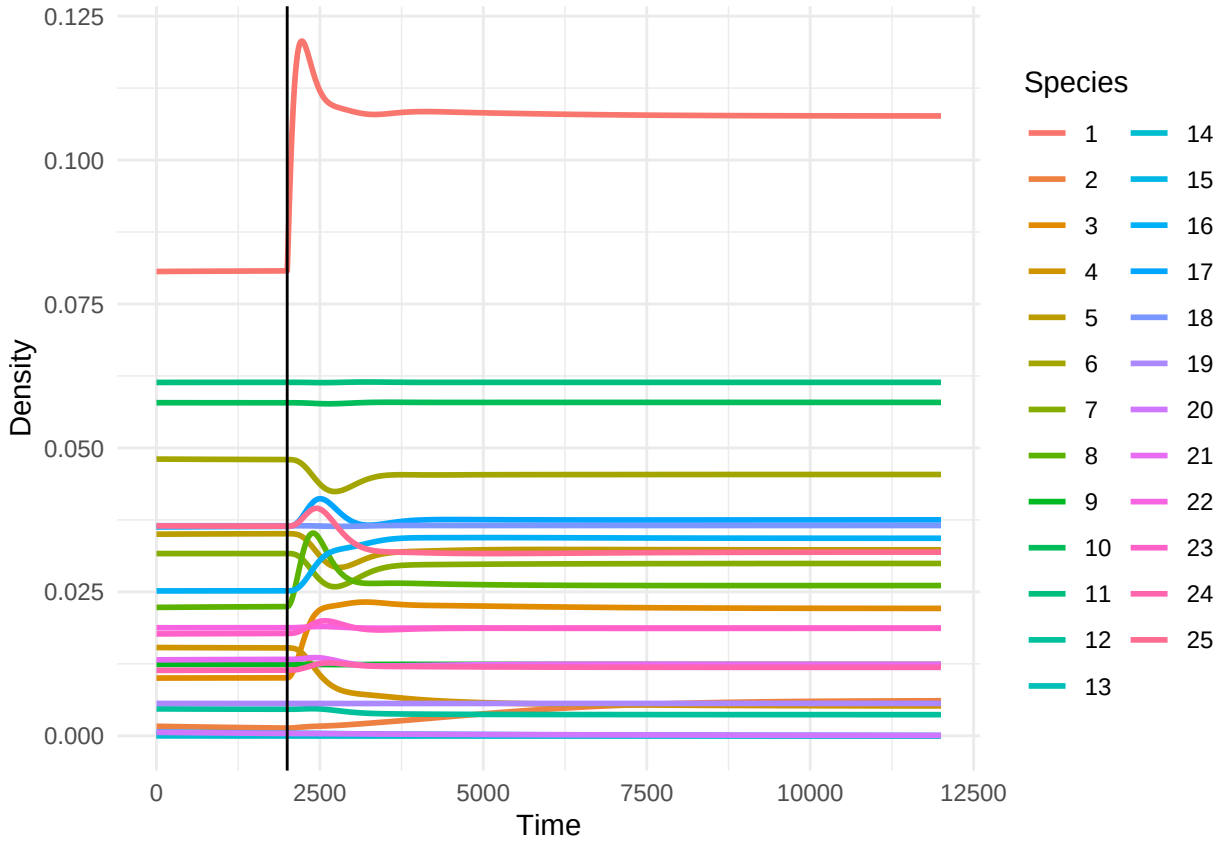
perturbation du réseau

Choisir une espèce à perturber et la valeur de perturbation ; simuler la dynamique et ploter ; différencier dans le plot l'espèce perturbé, ainsi que les espèces directement connectées à elles selon leur niveau troph, et les autres espèces en noir.

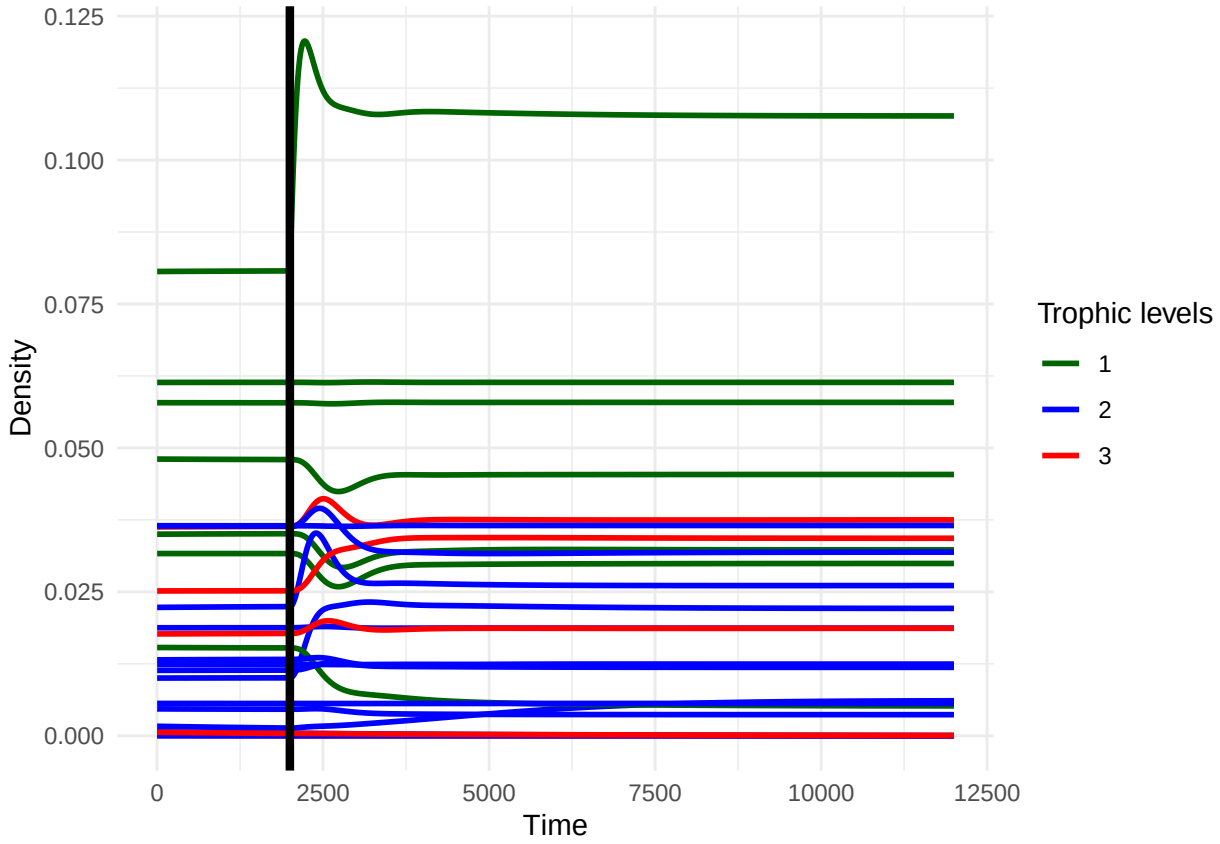
```
source("FoodWebPerturb.R")
#debug(FoodWebPerturb)

# Self-regulation (normally -1 because of prior normalization)
diag(MyFoodWeb) <- -1
# Run
Tmax <- 1000
Tstep <- 0.1
GrowthRate <- 0.1
DeathRate <- 0.01
Perturb <- 0.05
Dynamics <- FoodWebPerturb(MyFoodWeb, Tmax, Tstep, GrowthRate, DeathRate, 1, Perturb, type = "Positive")

# Version with colors = each species
DynamicsPlot <- ggplot(Dynamics) +
                geom_line(aes(x = Time, y = Density, color = factor(IdxSpecies)), linewidth = 1) +
                geom_vline(xintercept = 2000) +
                labs(x = "Time", y = "Density", color = "Species") +
                theme_minimal()
DynamicsPlot # show plot
```




```
# Version with colors = trophic levels
if (max(Dynamics$TrophLevel) == 3){
  colors <- c("darkgreen", "blue", "red")
}else{ # maximum trophic level = 4
  colors <- c("darkgreen", "blue", "red", "black")
}
DynamicsPlot <- ggplot(Dynamics) +
                geom_line(aes(x = Time, y = Density, color = factor(TrophLevel), group = factor(IdxSpec
                geom_vline(xintercept = 2000, linewidth = 1.5) +
                scale_color_manual(values = colors) +
                labs(x = "Time", y = "Density", color = "Trophic levels") +
                theme_minimal()
DynamicsPlot # show plot
```


0.125
0.100
0.075
0.050
0.025
0.000
Density
0
2500
5000
7500
10000
12500
Time
Trophic levels
1
2
3