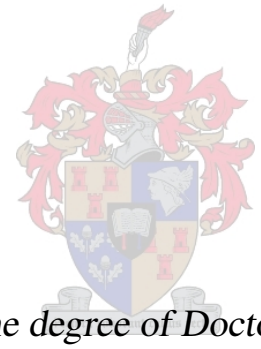
Complexity and stability of mutualistic local networks and meta-networks

by

Chinenye Assumpta Nnakenyi



Dissertation presented for the degree of Doctor of Philosophy in the Faculty of Science at Stellenbosch University

Supervisor: Prof. C. Hui

Co-supervisor: Dr. H.O. Minoarivelo

November 2020

```
    #Find the new modularity score based on node label updates.
    QbAfter = WEIGHTEDMODULARITY(BMatrix,Matsum,redlabels,bluelabels)

    #If this modularity is not as good as previous stop iterating and
    # use that previous best information

    if(QbAfter <= QbBefore) {
      redlabels = old_redlabels
      bluelabels = old_bluelabels
      TotalRedDegrees = old_TRD
      TotalBlueDegrees = old_TBD
      IterateFlag = 0
    }
  }
  Qb_now = QbAfter
  return(list(redlabels, bluelabels, Qb_now))
}
```

Appendix A1.2. A demonstration with Española Island data

```
## A DEMONSTRATION OF THE MODEL USING ESPANOLA ISLAND DATA.
## Set to the right working directory containing Appendix A1.1.R
#setwd()
## LOAD THE REQUIRED PACKAGES AND FUNCTIONS
library(bipartite)
library(deSolve)
source("Appendix S1.1.R")
## PARAMETER DEFINITIONS
# M    - Number of plants.
# N    - Number of animals.
# C    - Number of interactions
# A    - M x N binary interaction matrix.
# rP   - M x 1 matrix for plants intrinsic growth rates.
# rA   - N x 1 matrix for animals intrinsic growth rates.
# PP   - M x 1 matrix for plants density dependent coefficient.
# AA   - N x N matrix for animals density dependent coeficient.
# PA   - M x N matrix for strength of mutualistic benefits of plants from animals.
# AP   - N x M matrix for strength of mutualistic benefits of animals from plants.
# h    - Handling time, h=0.1 type II functional response.
# XP0  - M x 1 matrix for plants initial population sizes.
# XA0  - N x 1 matrix for animals initial population sizes.
# T    - Number of switching events.

## ESPANOLA ISLAND DATA DETAILS
M = 11
N = 16
C = 31

# GENERATE BINARY INTERACTION MATRIX.
AR = RndIntMx(M,N,C)
# ASSIGN VALUES TO THE PARAMETERS.
T = 5000                          # Number of switching event
sigma = 0.05                      # Standard deviation value for the normal distributions.
```

```
# GENERATE THE DENSITY DEPENDENT MATRIX
PP = matrix(runif(M,0,1), nr= M)
AA = matrix(runif(N,0,1), nr= N)

# GENERATE THE MUTUALISTIC BENEFIT MATRIX
PA <- abs(matrix(rnorm(M*N,0,sigma), nr=M))
AP <- abs(matrix(rnorm(M*N,0,sigma), nr=N))

# GENERATE INITIAL POPULATION SIZES
XP0 <- matrix(runif(M,0,1), nr= M)
XA0 <- matrix(runif(N,0,1), nr= N)

# GENERATE INTRINSIC GROWTH RATES
rP = matrix(runif(M,0,1), nr= M)
rA = matrix(runif(N,0,1), nr= N)

#### AIS MODEL, ELIMINATION ALGORITHM
h <- 0.1
A <- AR
###### AIS Model predictions from weighted interaction matrix
AIS <- elim(A,rP,rA,PP,AA,PA,AP,h,XP0,XA0,T);
AIS_nest <- matrix(unlist(AIS[2]),1,T)
AIS_mod <- matrix(unlist(AIS[3]),1,T)

#### PLOTS
# NESTEDNESS PREDICTON
tiff("Espanola_nestedness_prediction.tiff")
plot(1:T,seq(0,0.8,len=T), type = "n", xlab = "Time",ylab = "Nestedness",
     main="EspanolaIsland data")
lines(1:T, AIS_nest,lwd=2)
lines(1:T, rep(0.082,T),lwd=2,lty=2)        # observed nestedness
legend('topleft', c("AIS","Observed"), lty = c(1,2), lwd = c(2,2), cex = 0.8, bty = "n")
dev.off()
```

Supplementary Appendix S6: R scripts for Chapter 6

```
#####################################################################
# R codes for Chapter 6
# Author: Chinenye Assumpta Nnakenyi
# July 2020
#####################################################################

#### Libraries needed
library("ggplot2")
library("bipartite")
library("rootSolve")
library("deSolve")
library("grid")
library("foreach")
library("doParallel")
library("doFuture")
library("gridExtra")



###################### FUNCTIONS ################################

#####################################################################
############ Binary interaction matrix
# The function generates a binary interaction matrix for plants
# and animals interactions.

# Function inputs:
# M   = number of plants
```

```
# N   = number of animals
# C   = connectance of the matrix

# Function Output: The binary matrix

Binary_interaction <- function(M,N,C){
  S = M+N
  s = S*S
  AR = matrix(0,M,N)
  while (s>1){
    m = sample(M,1)
    n = sample(N,1)
    p = runif(1)
    if (p<C){
      AR[m,n] = 1
    }
    a = length(which(AR != 0))
    b = a/(M*N)
    if (b>C){
      s = 0
    }else{
      s = s-1
    }
  }
  return(AR)
}

#####################################################################
############ Competition matrix
# The function generates a competition matrix for the species in
# the same guild

# Function inputs:
# M   = number of species in the same guild
# C   = connectance of the off-diagonal matrix elements
# mu1  = mean of the off-diagonal matrix elements
# sg1  = standard deviation of the off-diagonal matrix elements
# m   = self-regulation term
```

```
# Function Output: The competition matrix

Competition <- function(M,C,mu1,sg1,m){
  c = C- (1/(M))
  # Number of interactions
  ITR = round(c*(S^2), digits = 0)
  s = M^4
  MM = matrix(0,S,S);   diag(MM)= -m
  while (s>1){
    x = sample(M,1)
    y = sample(M,1)
    if (x != y){
      MM[x,y] = -abs(rnorm(1,mu1,sg1))
      MM[y,x] = -abs(rnorm(1,mu1,sg1))
    }
    a = length(which(MM != 0))
    b = a-M
    if (b>=ITR){
      s = 0
    }else{
      s = s-1
    }
  }
  return(MM)
}

#####################################################################
############ Dispersal heterogeneity matrix
# The function generates a dispersal heterogeneity matrix for the
# species in the meta-network

# Function inputs:
# S   = number of species in the same guild
# n   = number of the local networks
# d   = dispersal mean value
# sgd = standard deviation of the dispersal rates

# Function Output: The dispersal heterogeneity matrix
```

```
dispersal.mat.hetero = function(S,n,d,sgd){

  dis = NULL
  for (k in 1:S){ # for each of the species
    dd = matrix(0,n,n)
    for (i in 1:n){  # for each local network
      # generate d_ikk
      w1 = abs(rnorm(1,d,sgd))

      # generate d_ilk, the proportions of the species i that is
      # moving from local network k to l
      w3 = runif(n-1)
      f = 1/sum(w3)
      w4 = f*w3

      dd[i,i] = w1
      dd[-i,i] = w1*w4
    }
    dis = rbind(dis,dd)
  }
  return(dis)

}

#####################################################################
############ Dispersal homogeneity matrix
# The function generates a dispersal homogeneity matrix for the
# species in the meta-network

# Function inputs:
# S   = number of species in the same guild
# n   = number of the local networks
# d   = dispersal mean value
# sgd = standard deviation of the dispersal rates

# Function Output: The dispersal homogeneity matrix

dispersal.mat.homo = function(S,n,d,sgd){
```

```
  dis = NULL
  for (k in 1:S){ # for each of the species
    dd = matrix(0,n,n)
    for (i in 1:n){
      w1 = abs(rnorm(1,d,sgd))
      dd[i,i] = w1
      dd[-i,i] = rep(w1/(n-1),n-1)
    }
    dis = rbind(dis,dd)
  }
  return(dis)

}


#####################################################################
### Model function

MetaNet_model <- function(M,N,c,m,sg1,sg2,mu1,mu2,n...){

  # Parameters
  AR = Binary_interaction(M,N,C)
  init.dens = matrix(runif(n*S),n*S,1)
  r = matrix(rlnorm(n*S,1,0.1),n*S,1)
  ap = NULL; aa = NULL
  bp = NULL; ba = NULL
  for (i in 1:n) {
    ap = rbind(ap, -Competition(M,C,mu1,sg1,m))
    aa = rbind(aa, -Competition(M,C,mu1,sg1,m))
    bp = rbind(bp, abs(matrix(rnorm(M*N,mu2,sg2),nr=M)))
    ba = rbind(ba, abs(matrix(rnorm(M*N,mu2,sg2),nr=N)))
  }
  h=0.1

  # saving the initial parameters
  write.csv(AR, paste0("Binary_Int_matrix.csv"), row.names = FALSE)
  write.csv(init.dens, paste0("Initial_densities.csv"), row.names = FALSE)
  write.csv(r, paste0("Growth_rate.csv"), row.names = FALSE)
  write.csv(ap, paste0("Plant_competition_matrix.csv"), row.names = FALSE)
```

```
write.csv(aa, paste0("Animal_competition_matrix.csv"), row.names = FALSE)
write.csv(bp, paste0("Plant_benefit_matrix.csv"), row.names = FALSE)
write.csv(ba, paste0("Animal_benefit_matrix.csv"), row.names = FALSE)

foreach(i=1:length(a)) %dopar% { ## for each dispersal mean value

  # Meta network model

  Lotka.LN10 = function(tt,y,parameters){
    with(as.list(c(y,parameters)),{

      y=y; AR=AR; r=r; ap=ap; aa=aa; bp=bp; ba=ba;
      h=h; d.mat=d.mat; S=S; n=n; M=M; N=N

      # Local network 1
      l=1;
      # the right indexes
      a1 = ((l-1)*S+1):((l-1)*S+M); b1 = ((l-1)*S+M+1):(l*S);
      a2 = ((l-1)*M+1):(l*M); b2 = 1:M ; a3 = ((l-1)*N+1):(l*N);
      b3 = 1:N;a4 = ((l-1)*M+1):(l*M); b4 = 1:N ; a5 = ((l-1)*N+1):(l*N); b5 = 1:M
      # plants and animals initial densities
      P1 = matrix(y[a1]); A1 = matrix(y[b1])
      # plants and animals growth rates
      rP1 = matrix(r[a1]); rA1 = matrix(r[b1])
      # plants and animals competitions
      aP1 = ap[a2,b2] ; aA1 = aa[a3,b3]
      # plants and animals benefits
      bP1 = bp[a4,b4];  bA1 = ba[a5,b5]

      # dispersal rate leaving local network k
      dP1 = as.matrix(sapply(1:M, function(j)
                     sum(d.mat[1:(M*n),l][((j-1)*n+1):(n*j)][-l])))
      # density from other other local networks
      Po1 = t(sapply(1:M, function(j) y[seq(j,n*S,by=S)][-l]))
      # dispersal rate moving into local network k from other islands
      dPo1 = t(sapply(1:M, function(j) d.mat[(j-1)*n+l,-l]))

      dA1 = as.matrix(sapply(1:N, function(j)
                     sum(d.mat[-(1:(M*n)),l][((j-1)*n+1):(n*j)][-l]) ))
```

```
Ao1 = t(sapply(1:N, function(j) y[seq((j+M),n*S,by=S)][-l] ))
dAo1 = t(sapply(1:N, function(j) d.mat[(j+M-1)*n+l,-l] ))

ddP1 = (rP1 - (aP1%*% P1) +
          ((bP1*AR) %*% A1)/(1+h*(AR)%*%A1))*P1 - dP1*P1 + rowSums(Po1*dPo1)
ddA1 = (rA1 - (aA1%*% A1) + ((bA1*t(AR)) %*% P1)/(1+h*(t(AR))%*%P1))*A1 -
          dA1*A1 + rowSums(Ao1*dAo1)

# Local network 2
l=2;
a1 = ((l-1)*S+1):((l-1)*S+M); b1 = ((l-1)*S+M+1):(l*S);
a2 = ((l-1)*M+1):(l*M); b2 = 1:M ; a3 = ((l-1)*N+1):(l*N);
b3 = 1:N;a4 = ((l-1)*M+1):(l*M); b4 = 1:N ; a5 = ((l-1)*N+1):(l*N); b5 = 1:M
P2 = matrix(y[a1]); A2 = matrix(y[b1]);
rP2 = matrix(r[a1]); rA2 = matrix(r[b1]);
aP2 = ap[a2,b2] ; aA2 = aa[a3,b3];
bP2 = bp[a4,b4];  bA2 = ba[a5,b5]
dP2 = as.matrix(sapply(1:M, function(j)
  sum(d.mat[1:(M*n),l][((j-1)*n+1):(n*j)][-l]) ))
Po2 = t(sapply(1:M, function(j) y[seq(j,n*S,by=S)][-l] ))
dPo2 = t(sapply(1:M, function(j) d.mat[(j-1)*n+l,-l] ))
dA2 = as.matrix(sapply(1:N, function(j)
  sum(d.mat[-(1:(M*n)),l][((j-1)*n+1):(n*j)][-l]) ))
Ao2 = t(sapply(1:N, function(j) y[seq((j+M),n*S,by=S)][-l] ))
dAo2 = t(sapply(1:N, function(j) d.mat[(j+M-1)*n+l,-l] ))
ddP2 = (rP2 - (aP2%*% P2) + ((bP2*AR) %*% A2)/(1+h*(AR)%*%A2))*P2 -
  dP2*P2 + rowSums(Po2*dPo2)
ddA2 = (rA2 - (aA2%*% A2) + ((bA2*t(AR)) %*% P2)/(1+h*(t(AR))%*%P2))*A2 -
  dA2*A2 + rowSums(Ao2*dAo2)

# Local network 3
l=3
a1 = ((l-1)*S+1):((l-1)*S+M); b1 = ((l-1)*S+M+1):(l*S);
a2 = ((l-1)*M+1):(l*M); b2 = 1:M ; a3 = ((l-1)*N+1):(l*N);
b3 = 1:N;a4 = ((l-1)*M+1):(l*M); b4 = 1:N ; a5 = ((l-1)*N+1):(l*N); b5 = 1:M
P3 = matrix(y[a1]); A3 = matrix(y[b1]);
rP3 = matrix(r[a1]); rA3 = matrix(r[b1])
aP3 = ap[a2,b2] ; aA3 = aa[a3,b3];
bP3 = bp[a4,b4];  bA3 = ba[a5,b5]
```

```
dP3 = as.matrix(sapply(1:M, function(j)
  sum(d.mat[1:(M*n),l][((j-1)*n+1):(n*j)][-l]) ));
Po3 = t(sapply(1:M, function(j) y[seq(j,n*S,by=S)][-l] )) ;
dPo3 = t(sapply(1:M, function(j) d.mat[(j-1)*n+l,-l] ))
dA3 = as.matrix(sapply(1:N, function(j)
  sum(d.mat[-(1:(M*n)),l][((j-1)*n+1):(n*j)][-l]) ));
Ao3 = t(sapply(1:N, function(j) y[seq((j+M),n*S,by=S)][-l] ));
dAo3 = t(sapply(1:N, function(j) d.mat[(j+M-1)*n+l,-l] ))
ddP3 = (rP3 - (aP3%*% P3) + ((bP3*AR) %*% A3)/(1+h*(AR)%*%A3))*P3 -
  dP3*P3 + rowSums(Po3*dPo3)
ddA3 = (rA3 - (aA3%*% A3) + ((bA3*t(AR)) %*% P3)/(1+h*(t(AR))%*%P3))*A3 -
  dA3*A3 + rowSums(Ao3*dAo3)

# Local network 4
l=4;
a1 = ((l-1)*S+1):((l-1)*S+M); b1 = ((l-1)*S+M+1):(l*S);
a2 = ((l-1)*M+1):(l*M); b2 = 1:M ; a3 = ((l-1)*N+1):(l*N);
b3 = 1:N;a4 = ((l-1)*M+1):(l*M); b4 = 1:N ; a5 = ((l-1)*N+1):(l*N); b5 = 1:M
P4 = matrix(y[a1]); A4 = matrix(y[b1]);
rP4 = matrix(r[a1]); rA4 = matrix(r[b1]);
aP4 = ap[a2,b2] ; aA4 = aa[a3,b3];
bP4 = bp[a4,b4];  bA4 = ba[a5,b5]
dP4 = as.matrix(sapply(1:M, function(j)
  sum(d.mat[1:(M*n),l][((j-1)*n+1):(n*j)][-l]) ));
Po4 = t(sapply(1:M, function(j) y[seq(j,n*S,by=S)][-l] )) ;
dPo4 = t(sapply(1:M, function(j) d.mat[(j-1)*n+l,-l] ))
dA4 = as.matrix(sapply(1:N, function(j)
  sum(d.mat[-(1:(M*n)),l][((j-1)*n+1):(n*j)][-l]) ));
Ao4 = t(sapply(1:N, function(j) y[seq((j+M),n*S,by=S)][-l] ));
dAo4 = t(sapply(1:N, function(j) d.mat[(j+M-1)*n+l,-l] ))
ddP4 = (rP4 - (aP4%*% P4) + ((bP4*AR) %*% A4)/(1+h*(AR)%*%A4))*P4 -
  dP4*P4 + rowSums(Po4*dPo4)
ddA4 = (rA4 - (aA4%*% A4) + ((bA4*t(AR)) %*% P4)/(1+h*(t(AR))%*%P4))*A4 -
  dA4*A4 + rowSums(Ao4*dAo4)

# Local network 5
l=5;
a1 = ((l-1)*S+1):((l-1)*S+M); b1 = ((l-1)*S+M+1):(l*S);
a2 = ((l-1)*M+1):(l*M); b2 = 1:M ; a3 = ((l-1)*N+1):(l*N);
```

```
b3 = 1:N;a4 = ((l-1)*M+1):(l*M); b4 = 1:N ; a5 = ((l-1)*N+1):(l*N); b5 = 1:M
P5 = matrix(y[a1]); A5 = matrix(y[b1]);
rP5 = matrix(r[a1]); rA5 = matrix(r[b1]);
aP5 = ap[a2,b2] ; aA5 = aa[a3,b3];
bP5 = bp[a4,b4];  bA5 = ba[a5,b5]
dP5 = as.matrix(sapply(1:M, function(j)
  sum(d.mat[1:(M*n),l][((j-1)*n+1):(n*j)][-l]) ));
Po5 = t(sapply(1:M, function(j) y[seq(j,n*S,by=S)][-l] )) ;
dPo5 = t(sapply(1:M, function(j) d.mat[(j-1)*n+l,-l] ))
dA5 = as.matrix(sapply(1:N, function(j)
  sum(d.mat[-(1:(M*n)),l][((j-1)*n+1):(n*j)][-l]) ));
Ao5 = t(sapply(1:N, function(j) y[seq((j+M),n*S,by=S)][-l] ));
dAo5 = t(sapply(1:N, function(j) d.mat[(j+M-1)*n+l,-l] ))
ddP5 = (rP5 - (aP5%*% P5) + ((bP5*AR) %*% A5)/(1+h*(AR)%*%A5))*P5 -
  dP5*P5 + rowSums(Po5*dPo5)
ddA5 = (rA5 - (aA5%*% A5) + ((bA5*t(AR)) %*% P5)/(1+h*(t(AR))%*%P5))*A5 -
  dA5*A5 + rowSums(Ao5*dAo5)

# Local network 6
l=6;
a1 = ((l-1)*S+1):((l-1)*S+M); b1 = ((l-1)*S+M+1):(l*S);
a2 = ((l-1)*M+1):(l*M); b2 = 1:M ; a3 = ((l-1)*N+1):(l*N);
b3 = 1:N;a4 = ((l-1)*M+1):(l*M); b4 = 1:N ; a5 = ((l-1)*N+1):(l*N); b5 = 1:M
P6 = matrix(y[a1]); A6 = matrix(y[b1]);
rP6 = matrix(r[a1]); rA6 = matrix(r[b1]);
aP6 = ap[a2,b2] ; aA6 = aa[a3,b3];  bP6 = bp[a4,b4];  bA6 = ba[a5,b5]
dP6 = as.matrix(sapply(1:M, function(j)
  sum(d.mat[1:(M*n),l][((j-1)*n+1):(n*j)][-l]) ));
Po6 = t(sapply(1:M, function(j) y[seq(j,n*S,by=S)][-l] )) ;
dPo6 = t(sapply(1:M, function(j) d.mat[(j-1)*n+l,-l] ))
dA6 = as.matrix(sapply(1:N, function(j)
  sum(d.mat[-(1:(M*n)),l][((j-1)*n+1):(n*j)][-l]) ));
Ao6 = t(sapply(1:N, function(j) y[seq((j+M),n*S,by=S)][-l] ));
dAo6 = t(sapply(1:N, function(j) d.mat[(j+M-1)*n+l,-l] ))
ddP6 = (rP6 - (aP6%*% P6) + ((bP6*AR) %*% A6)/(1+h*(AR)%*%A6))*P6 -
  dP6*P6 + rowSums(Po6*dPo6)
ddA6 = (rA6 - (aA6%*% A6) + ((bA6*t(AR)) %*% P6)/(1+h*(t(AR))%*%P6))*A6 -
  dA6*A6 + rowSums(Ao6*dAo6)
```

```
# Local network 7
l=7;
a1 = ((l-1)*S+1):((l-1)*S+M); b1 = ((l-1)*S+M+1):(l*S);
a2 = ((l-1)*M+1):(l*M); b2 = 1:M ; a3 = ((l-1)*N+1):(l*N);
b3 = 1:N;a4 = ((l-1)*M+1):(l*M); b4 = 1:N ; a5 = ((l-1)*N+1):(l*N); b5 = 1:M
P7 = matrix(y[a1]); A7 = matrix(y[b1]);
rP7 = matrix(r[a1]); rA7 = matrix(r[b1]);
aP7 = ap[a2,b2] ; aA7 = aa[a3,b3];  bP7 = bp[a4,b4];  bA7 = ba[a5,b5]
dP7 = as.matrix(sapply(1:M, function(j)
  sum(d.mat[1:(M*n),l][((j-1)*n+1):(n*j)][-l])));
Po7 = t(sapply(1:M, function(j) y[seq(j,n*S,by=S)][-l] )) ;
dPo7 = t(sapply(1:M, function(j) d.mat[(j-1)*n+l,-l] ))
dA7 = as.matrix(sapply(1:N, function(j)
  sum(d.mat[-(1:(M*n)),l][((j-1)*n+1):(n*j)][-l]) ));
Ao7 = t(sapply(1:N, function(j) y[seq((j+M),n*S,by=S)][-l] ));
dAo7 = t(sapply(1:N, function(j) d.mat[(j+M-1)*n+l,-l] ))
ddP7 = (rP7 - (aP7%*% P7) + ((bP7*AR) %*% A7)/(1+h*(AR)%*%A7))*P7 -
  dP7*P7 + rowSums(Po7*dPo7)
ddA7 = (rA7 - (aA7%*% A7) + ((bA7*t(AR)) %*% P7)/(1+h*(t(AR))%*%P7))*A7 -
  dA7*A7 + rowSums(Ao7*dAo7)

# Local network 8
l=8;
a1 = ((l-1)*S+1):((l-1)*S+M); b1 = ((l-1)*S+M+1):(l*S);
a2 = ((l-1)*M+1):(l*M); b2 = 1:M ; a3 = ((l-1)*N+1):(l*N);
b3 = 1:N;a4 = ((l-1)*M+1):(l*M); b4 = 1:N ; a5 = ((l-1)*N+1):(l*N); b5 = 1:M
P8 = matrix(y[a1]); A8 = matrix(y[b1]);
rP8 = matrix(r[a1]); rA8 = matrix(r[b1]);
aP8 = ap[a2,b2] ; aA8 = aa[a3,b3];
bP8 = bp[a4,b4];  bA8 = ba[a5,b5]
dP8 = as.matrix(sapply(1:M, function(j)
  sum(d.mat[1:(M*n),l][((j-1)*n+1):(n*j)][-l]) ));
Po8 = t(sapply(1:M, function(j) y[seq(j,n*S,by=S)][-l] )) ;
dPo8 = t(sapply(1:M, function(j) d.mat[(j-1)*n+l,-l] ))
dA8 = as.matrix(sapply(1:N, function(j)
  sum(d.mat[-(1:(M*n)),l][((j-1)*n+1):(n*j)][-l]) ));
Ao8 = t(sapply(1:N, function(j) y[seq((j+M),n*S,by=S)][-l] ));
dAo8 = t(sapply(1:N, function(j) d.mat[(j+M-1)*n+l,-l] ))
ddP8 = (rP8 - (aP8%*% P8) + ((bP8*AR) %*% A8)/(1+h*(AR)%*%A8))*P8 -
```

```
  dP8*P8 + rowSums(Po8*dPo8)
ddA8 = (rA8 - (aA8%*% A8) + ((bA8*t(AR)) %*% P8)/(1+h*(t(AR))%*%P8))*A8 -
  dA8*A8 + rowSums(Ao8*dAo8)

# Local network 9
l=9;
a1 = ((l-1)*S+1):((l-1)*S+M); b1 = ((l-1)*S+M+1):(l*S);
a2 = ((l-1)*M+1):(l*M); b2 = 1:M ; a3 = ((l-1)*N+1):(l*N);
b3 = 1:N;a4 = ((l-1)*M+1):(l*M); b4 = 1:N ; a5 = ((l-1)*N+1):(l*N); b5 = 1:M
P9 = matrix(y[a1]); A9 = matrix(y[b1]);
rP9 = matrix(r[a1]); rA9 = matrix(r[b1]);
aP9 = ap[a2,b2] ; aA9 = aa[a3,b3];  bP9 = bp[a4,b4];  bA9 = ba[a5,b5]
dP9 = as.matrix(sapply(1:M, function(j)
  sum(d.mat[1:(M*n),l][((j-1)*n+1):(n*j)][-l]) ));
Po9 = t(sapply(1:M, function(j) y[seq(j,n*S,by=S)][-l] )) ;
dPo9 = t(sapply(1:M, function(j) d.mat[(j-1)*n+l,-l] ))
dA9 = as.matrix(sapply(1:N, function(j)
  sum(d.mat[-(1:(M*n)),l][((j-1)*n+1):(n*j)][-l]) ));
Ao9 = t(sapply(1:N, function(j) y[seq((j+M),n*S,by=S)][-l] ));
dAo9 = t(sapply(1:N, function(j) d.mat[(j+M-1)*n+l,-l] ))
ddP9 = (rP9 - (aP9%*% P9) + ((bP9*AR) %*% A9)/(1+h*(AR)%*%A9))*P9 -
  dP9*P9 + rowSums(Po9*dPo9)
ddA9 = (rA9 - (aA9%*% A9) + ((bA9*t(AR)) %*% P9)/(1+h*(t(AR))%*%P9))*A9 -
  dA9*A9 + rowSums(Ao9*dAo9)

# Local network 10
l=10;
a1 = ((l-1)*S+1):((l-1)*S+M); b1 = ((l-1)*S+M+1):(l*S);
a2 = ((l-1)*M+1):(l*M); b2 = 1:M ; a3 = ((l-1)*N+1):(l*N);
b3 = 1:N;a4 = ((l-1)*M+1):(l*M); b4 = 1:N ; a5 = ((l-1)*N+1):(l*N); b5 = 1:M
P10 = matrix(y[a1]); A10 = matrix(y[b1]);
rP10 = matrix(r[a1]); rA10 = matrix(r[b1]);
aP10 = ap[a2,b2] ; aA10 = aa[a3,b3];  bP10 = bp[a4,b4];  bA10 = ba[a5,b5]
dP10 = as.matrix(sapply(1:M, function(j)
  sum(d.mat[1:(M*n),l][((j-1)*n+1):(n*j)][-l]) ));
Po10 = t(sapply(1:M, function(j) y[seq(j,n*S,by=S)][-l] )) ;
dPo10 = t(sapply(1:M, function(j) d.mat[(j-1)*n+l,-l] ))
dA10 = as.matrix(sapply(1:N, function(j)
  sum(d.mat[-(1:(M*n)),l][((j-1)*n+1):(n*j)][-l]) ));
```

```
        Ao10 = t(sapply(1:N, function(j) y[seq((j+M),n*S,by=S)][-l] ));
        dAo10 = t(sapply(1:N, function(j) d.mat[(j+M-1)*n+l,-l] ))
        ddP10 = (rP10 - (aP10%*% P10) + ((bP10*AR) %*% A10)/(1+h*(AR)%*%A10))*P10 -
          dP10*P10 + rowSums(Po10*dPo10)
        ddA10 = (rA10 - (aA10%*% A10) + ((bA10*t(AR)) %*% P10)/(1+h*(t(AR))%*%P10))*A10
          dA10*A10 + rowSums(Ao10*dAo10)

        dP <-  rbind(ddP1,ddA1,ddP2,ddA2,ddP3,ddA3,ddP4,ddA4,ddP5,ddA5,
                     ddP6,ddA6,ddP7,ddA7,ddP8,ddA8,ddP9,ddA9,ddP10,ddA10)
        return(list(dP))
      })
    }

    yini <-  c(init.dens[,1])
    times = seq(0,5,0.1)
    dn=a[i]
    sgdn = asd[i]
    # generate the dispersal matrix
    d.mat = dispersal.mat.homo(S,n,dn,sgdn)# for dispersal homogeneity
    #d.mat = dispersal.mat.hetero(S,n,dn,sgdn) # for dispersal heterogeneity
    parameterss = c(AR=AR, r=r,ap=ap,aa=aa,bp=bp,ba=ba,h=h,
                    d.mat=d.mat,S=S,n=n,M=M,N=N)
    out <- lsoda(y = yini, times = times, func=Lotka.LN10,parms = parameterss)
    nn = out[length(out[,1]),-1]    # the equilibrium values
    jj = jacobian.full(y=nn ,func=Lotka.LN10)    # Jacobian matrix

    ## saving the matrix results
    write.csv(jj, paste0("Jacobian_d=",a[i],".csv"), row.names = FALSE)
    write.csv(d.mat, paste0("dispersal_matrix_d=",a[i],".csv"), row.names = FALSE)
    write.csv(nn, paste0("Final_equil_densities_d=",a[i],".csv"), row.names = FALSE)
    write.csv(out, paste0("Densities_over_time_d=",a[i],".csv"), row.names = FALSE)

  }
}


################# Compositional similarity
# Morisita-Horn index:
Morisita_Horn = function(matobj){
```

```
  AA = matobj
  nn = ncol(AA)
  ## Obtain the relative abundance of the matrix
  A = matrix(0,nrow(AA),ncol(AA))
  for (i in 1:ncol(A)) {
    A[,i] = AA[,i]/sum(AA[,i])
  }

  ## Apply the Morisita-Horn index
  Denominator = (nn-1)*sum(A^2)
  val1 = c()
  for(i in 1:(nn-1)){
    for (j in i:(nn)){
      if (i != j && i<j){
        mat1 = cbind(A[,i],A[,j])
        Numerator = sum(mat1[,1]*mat1[,2])
        val1 = append(val1, Numerator)
      }
    }
  }
  C2n = (2*sum(val1))/Denominator
  mean.val1 = C2n
  se.val1 = sd(mean.val1)/sqrt(length(mean.val1))
  return(list("Mean of pairs"= mean.val1, "Std.error"=se.val1))
}



### For computing Morisita-Horn for local network
MH.values_LN <- function(n,a){
  MH.v = matrix(0,length(a),2)
  Amat = as.matrix(read.csv("Abundance_sim1.csv"))
  MH.mean.vals = c(); MH.se.vals = c()
  for (i in 1:length(a)){
    mat.A = matrix(c(Amat[,i]),(M+N),n,byrow = TRUE)
    MH = Morisita_Horn(mat.A)
    MH.mean.vals = append(MH.mean.vals, MH$`Mean of pairs`)
    MH.se.vals = append(MH.se.vals, MH$Std.error)
```

```
  }

  MH.v[,1] = MH.mean.vals
  MH.v[,2] = MH.se.vals
  return(MH.v)
}

### For computing M-H for metanetwork
MH.values_MN <- function(n,a){
  MH.v = matrix(0,length(a),2)
  Amat = as.matrix(read.csv("Abundance_sim1.csv"))

  i=1   # no dispersal
  mat.A1 = rowSums(matrix(c(Amat[,i]),(M+N),n,byrow = TRUE))

  for (i in 2:length(a)){
    mat.A2 = rowSums(matrix(c(Amat[,i]),(M+N),n,byrow = TRUE))
    mat.B = cbind(mat.A1,mat.A2)
    MH = Morisita_Horn(mat.B)
    MH.v[i,1] = MH$`Mean of pairs`
    MH.v[i,2] = MH$Std.error
  }
  return(MH.v)
}

###
Gini_index <- function(S,n,Eq){

  # For local networks
  if (length(Eq)==S){
    # Apply Gini formula
    Eqq = sort(Eq)
    E1 = NULL
    for (i in 1:length(Eqq)){
      E1 = append(E1,  ((S+1-i)*Eqq[i])/sum(Eqq) )
    }
    E2 = (1/S)*(S+1-(2*sum(E1)))
    return(E2)
  }else{
```

```
    # For meta-network
    Eq1 = matrix(0,S,n)  # to separate the densities of each local network
    for (i in 1:n){
      Eq1[,i] = Eq[((i-1)*S+1):(S*i)]
    }
    Eq2 = rowSums(Eq1)  # summed densities from n local networks

    # Apply Gini formula
    Eqq = sort(Eq2)
    E1 = NULL
    for (i in 1:length(Eqq)){
      E1 = append(E1,  ((S+1-i)*Eqq[i])/sum(Eqq) )
    }
    E2 = (1/S)*(S+1-(2*sum(E1)))
    return(E2)
  }
}

#### Functions of the networks metrices

# Computing the network metrices such as leading eigenvalues,
# total abundance, gini,  nestedness and modularity from the saved matrices

Comput_network_metrices <- function(n,a,S...){
  X = matrix(0,n*(M+N),length(a))      # Abundance
  Eigenvalues = matrix(0,length(a),n+1)  # Eigenvalues
  Gini = matrix(0,length(a),n+1)
  Nestedness = matrix(0,length(a),n+1)
  Modularity = matrix(0,length(a),n+1)

  for (i in 1:length(a)){ # for loop over dispersal

    v1 = i

    X[,v1] <- as.matrix(read.csv(paste0("Final_equil_densities_d=",a[i],".csv")))
    jj = as.matrix(read.csv(paste0("Jacobian_d=",a[i],".csv")))
    AR = as.matrix(read.csv("Binary_Int_matrix.csv"))
```

```
  # For local networks

  plant.mat = matrix(0,M,n)
  animal.mat = matrix(0,N,n)
  for (k in 1:n){
    Abund.k = X[,v1][(((k-1)*S) + 1):(k*S)]
    P_abund = as.matrix(Abund.k[1:M])
    A_Abund = as.matrix(Abund.k[-(1:M)])

    w.mat = AR * (P_abund %*% t(A_Abund))  # the weighted matrix A*Pi*Aj
    Nestedness[v1,k+1] = nested(w.mat, method = "weighted NODF",
                                rescale=FALSE, normalised=TRUE)/100
    Modularity[v1,k+1] = LPA_wb_plus(w.mat)$modularity

    plant.mat[,k] = P_abund
    animal.mat[,k] = A_Abund

    a1 = ((k-1)*S+1):(k*S); b1 = ((k-1)*S+1):(k*S);
    Gini[v1,k+1] = Gini_index(S,n,X[,v1][a1])
    Eigenvalues[v1,k+1]= max(Re(eigen(jj[a1,b1])$values))
  }

  # For Meta-network
  Eigenvalues[v1,1] =  max(Re(eigen(jj)$values))
  Gini[v1,1] = Gini_index(S,n,X[,v1])

  wm.mat = AR * (plant.mat) %*% t((animal.mat))
  Nestedness[v1,1] = nested(wm.mat, method = "weighted NODF",
                            rescale=FALSE, normalised=TRUE)/100
  Modularity[v1,1] = LPA_wb_plus(wm.mat)$modularity

}

c.name = c("MNT", paste0("LN",1:n))
colnames(Eigenvalues)= c.name;
colnames(Nestedness)= c.name
colnames(Modularity)= c.name
colnames(Gini)= c.name;
```

```
  return(list("Abundance"=X, "Eigenvalues"=Eigenvalues, "Nestedness"=Nestedness,
              "Gini"=Gini, "Modularity"=Modularity,"dispersal"=a))
}

#############################################################################

############### Implementation:

# for parallel computing
no_cores = detectCores()
registerDoParallel(makeCluster(no_cores))
registerDoFuture()

set.seed(123)

# set a working directory
setwd("C:/Users/assumpta/Desktop/example")

M=30; N=20; C=c=0.2; m=1; sg1=0.05;sg2=0.05;
mu1=0; mu2=0; S=M+N;n=10;sgd=0.01

M=2; N=3; C=c=0.5; m=1; sg1=0.05;sg2=0.05;mu1=0; mu2=0; S=M+N;n=10;sgd=0.01

# dispersal mean values
a = c(0, exp(seq(-4,3,by=0.25)))

# standard deviation values for each dispersal mean
asd = c(0,rep(sgd,times=(length(a)-1)))

# Run the model
system.time(
  MetaNet_model(M,N,c,m,sg1,sg2,mu1,mu2,n...)
)
stopImplicitCluster()



# compute network metrices
system.time({
  LN = Comput_network_metrices(n,a,S...)
```

```
})
```