Ecological Dynamics
Lab 5: Spatial models

Background

- Most ecosystems are open and thus affected by regional processes.
- Understanding the relative importance of local vs. regional processes is a long-standing goal in ecology.
- Spatial models can provide key insights about how local and regional processes interact to affect the structure, stability and persistence of interconnected ecosystems.

Objectives

- Develop 1-D spatial models with discrete dynamics.
- Learn how develop 2-D grid models to simulate coupled map lattices.
- Use networks to represent complex spatial feedbacks between populations.

Instructions

- Launch RStudio, open a new script file and save it as `lab5-yourLastName.R`.
- Complete the activities below by adding the appropriate commands to your `R` script file.
- Embed your answers as comments in the `R` script file.
- At the end of your session, save your file onto a flash drive (files saved on lab computers are wiped when you log out).

Task 1: Harvesting in spatially-structured populations

1. To determine whether space effects the maximum sustainable yield computed for assignment 2, we will model the dynamics of a Cod metapopulation along a coastline. To do so, we will use a 1-D linear array and assume that dispersal occurs at rate d between nearest neighbors. Furthermore, we will assume that the dynamics of the species is well represented by the discrete-time logistic model with a constant *per capita* harvesting rate h. Develop the code necessary to simulate the dynamics of this model for (i) absorbing, (ii) reflecting and (iii) periodic boundary conditions.

```
library(fields)
find.neighs <- function(npops) {
    neighs.abs <- matrix(nrow = npops, ncol = 3)
    neighs.abs[, 1] <- 1:npops
    colnames(neighs.abs) <- c("pop.number", "neighb.left", "neighb.right")
    neighs.abs[2:(npops - 1), 2:3] <- c((2:(npops - 1)) - 1, (2:(npops -
        1)) + 1)
    neighs.per <- neighs.abs
    neighs.ref <- neighs.abs
    # Periodic boundaries
    neighs.per[1, 2:3] <- c(npops, 2)
```

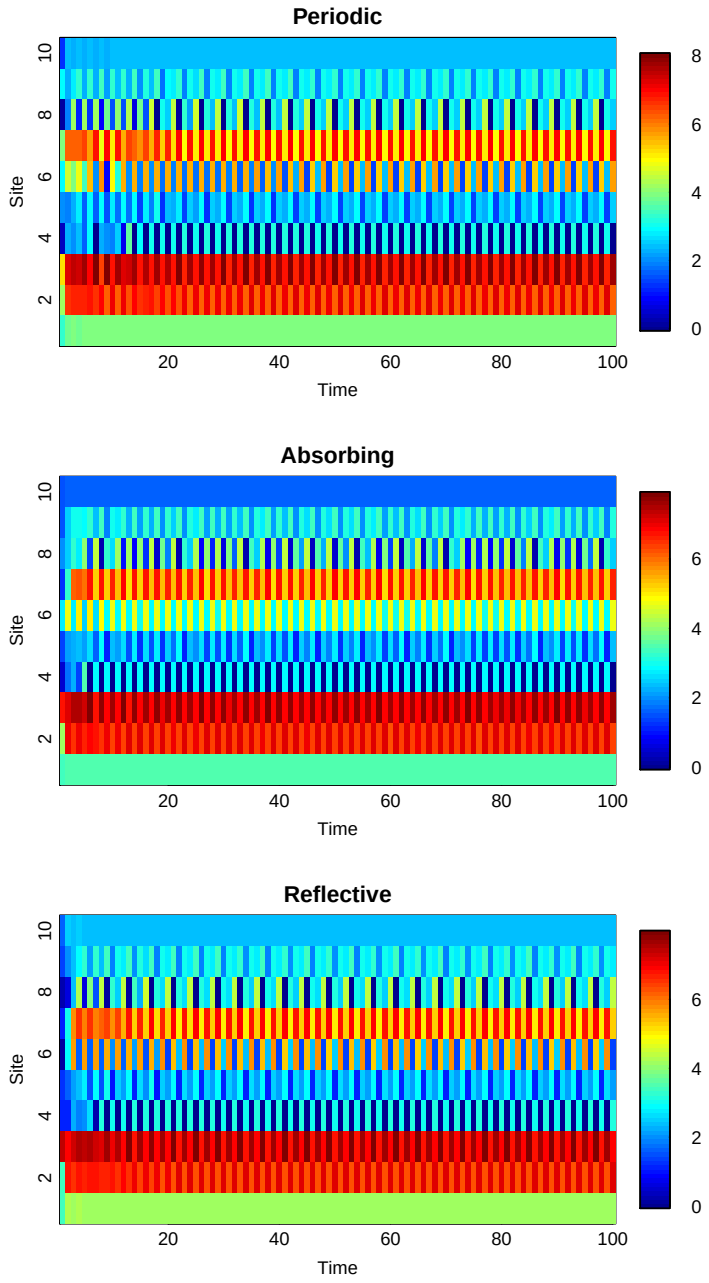
```
    neighs.per[npops, 2:3] <- c(npops - 1, 1)
    # Reflecting boundaries
    neighs.ref[1, 2:3] <- c(1, 2)
    neighs.ref[npops, 2:3] <- c(npops, 1)
    # Absorbing boundaries
    neighs.abs[1, 2:3] <- c(NA, 1)
    neighs.abs[npops, 2:3] <- c(npops, NA)
    return(list(neighs.abs = neighs.abs, neighs.per = neighs.per, neighs.ref = neighs.ref))
}
dynamics <- function(parms, neighs, max.time, npops) {
    with(parms, {
        N <- matrix(nrow = max.time, ncol = npops)
        N[1, ] <- runif(npops) * K
        for (i in 2:max.time) {
            immigration <- rowSums(0.5 * d * matrix(N[i - 1, neighs[, 2:3]],
                ncol = 2, byrow = TRUE), na.rm = TRUE)
            emigration <- d * N[i - 1, ]
            N[i, ] <- N[i - 1, ] * (1 + R * (1 - N[i - 1, ]/K) - h) - emigration +
                immigration
            N[i, ] <- ifelse(N[i, ] < 0, 0, N[i, ])
        }
        return(N)
    })
}
```

2. Simulate the model with each type of boundary condition for 10 populations, 100 total time steps, a dispersal rate of $d = 0.5$, harvesting rate $h = 0$, growth rate R randomly selected between 1 and 2, and carrying capacity K randomly selected between 0 and 10. Do this for 5 replicate simulations and average the results.

```
npops <- 10
max.time <- 100
R <- runif(npops, min = 1, max = 2)
K <- runif(npops) * 10
d <- 0.5
parms <- list(R = R, K = K, d = d, h = 0)
neighs <- find.neighs(npops)
dynamics.abs <- dynamics(parms, neighs$neighs.abs, max.time, npops)
dynamics.ref <- dynamics(parms, neighs$neighs.ref, max.time, npops)
dynamics.per <- dynamics(parms, neighs$neighs.per, max.time, npops)
```

3. To visualize the effect of using different boundary conditions, load the `fields` package and use function `image.plot` to plot the abundance as a function of time (x-axis) and space (y-axis) for each type of boundary on a 3-row by 1-column panel. Are the dynamics of the model surprising given what you know about the discrete-time logistic model? Can you spot the edge effects?

```
par(mfrow = c(3, 1))
image.plot(1:max.time, 1:npops, dynamics.per, xlab = "Time", ylab = "Site",
    main = "Periodic")
image.plot(1:max.time, 1:npops, dynamics.abs, xlab = "Time", ylab = "Site",
    main = "Absorbing")
image.plot(1:max.time, 1:npops, dynamics.ref, xlab = "Time", ylab = "Site",
    main = "Reflective")
```



Answer:

The populations seem to be fluctuating over time. This is surprising since previous analyses have shown that the model should exhibit a stable equilibrium (node) as long as $R < 2$. This suggests that dispersal is destabilizing the system.

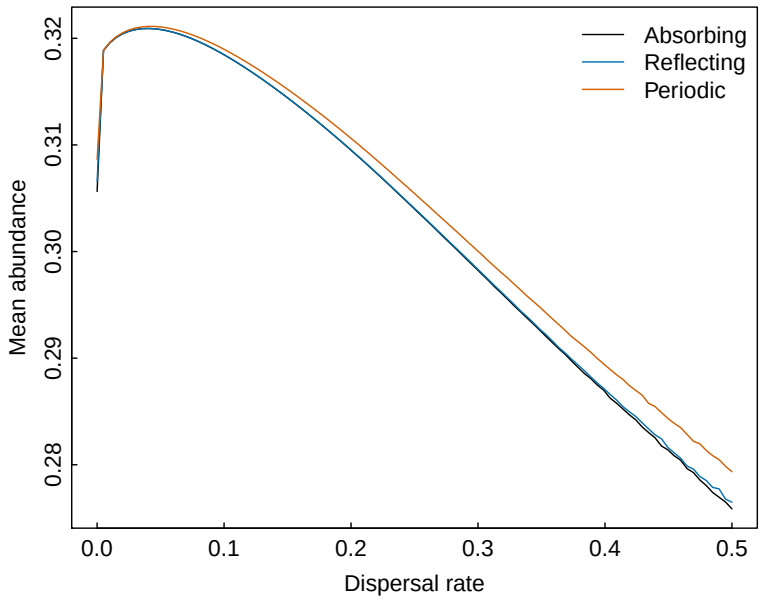
Additionally, the first and last sites or populations for the simulations with absorbing or reflective boundary conditions clearly behave differently than the other sites because of edge effects. Specifically, these sites or populations receive fewer immigrants than the other sites and thus enjoy more stable dynamics.

4. To determine the effect of dispersal on population size, run a set of simulations for each type of boundary condition with 100 population, a total of 500 time steps, R between 1 and 2, K between 0 and 10 and $h = \frac{RK}{4}$. Vary dispersal linearly between 0 and 0.5 in 100 increments and compute the mean population size across the entire metapopulation for the last 100 time steps of each simulation.

```
npops <- 100
nreps <- 5
max.time <- 500
start.time <- 100
R <- runif(npops, min = 1, max = 2)
K <- runif(npops) * 10
msy <- R * K/4
disp <- seq(from = 0, to = 0.5, length = 100)
parms <- list(R = R, K = K, h = msy)
neighs <- find.neighs(npops)
mean.abund <- matrix(nrow = length(disp), ncol = 4)
for (i in 1:length(disp)) {
    parms$d <- disp[i]
    mean.abund.tmp <- matrix(nrow = nreps, ncol = 4)
    for (j in 1:nreps) {
        dynamics.abs <- dynamics(parms, neighs$neighs.abs, max.time, npops)
        dynamics.ref <- dynamics(parms, neighs$neighs.ref, max.time, npops)
        dynamics.per <- dynamics(parms, neighs$neighs.per, max.time, npops)
        mean.abund.tmp[j, ] <- c(disp[i], mean(dynamics.abs[start.time:max.time,
            ]), mean(dynamics.ref[start.time:max.time, ]), mean(dynamics.per[start.time:max.time,
            ]))
    }
    mean.abund[i, ] <- colMeans(mean.abund.tmp)
}
```

5. Plot mean abundance as a function of dispersal rate and discuss the pattern.

```
matplot(mean.abund[, 1], mean.abund[, 2:4], lty = 1, t = "l", col = c(cb["black"],
    cb["blue"], cb["red"]), xlab = "Dispersal rate", ylab = "Mean abundance")
legend(x = "topright", legend = c("Absorbing", "Reflecting", "Periodic"),
    col = c(cb["black"], cb["blue"], cb["red"]), lty = 1, bty = "n")
```



Answer:

Mean abundance decreases with increasing dispersal. This is because the harvesting rate at each site is based on local parameter values (R, K), but increasing dispersal generates a mismatch between the local parameter values and the local population densities. Hence, some populations are being overharvested leading to a decline in the mean abundance across the entire metapopulation.

Task 2: Dispersal and stability in coupled map lattices

For this task, you will need to download a few functions to simulate coupled map lattices (i.e., coupled differential equations):

```
source("http://faraway.neu.edu/data/lattice.R")
```

You will also need to install and load packages `deSolve`, `synchrony` and `fields`.

1. The code in `lattice.R` defines a coupled map lattice version of the Rosenzweig-MacArthur predator-prey model. To run it, you'll need to set the size of the square grid or lattice along with all of the parameters of the model as follows:

```
npops <- 50
max.time <- 500
start.time <- 400
a <- 0.2  # attack rate
m <- 0.2  # mortality rate
d <- 0.01  # dispersal rate
r <- 0.5  # growth rate
K <- 40  # carrying capacity
h <- 1  # handling time
b <- 1  # efficiency
```

2. Function `get.timeseries` can be used to extract the abundances of the predator and the prey across the entire metacommunity. You will need it to compute the statistical properties of the prey population as a function of the dispersal rate. Specifically, write a function called `compute.stats` that takes in the prey time series and computes the global stability (mean across the entire metapopulation divided by the standard deviation), local stability (the average across the metapopulation of: the mean of each local population divided by the local population standard deviation), and synchrony using `kendall.w` (use `w.corrected`) for the last 100 time steps of the simulation.

```
compute.stats <- function(timeseries) {
    local.stab <- mean(apply(timeseries, 2, FUN = function(x) mean(x)/sd(x)))
    global <- rowMeans(timeseries)
    global.stab <- mean(global)/sd(global)
    return(list(local.stab = local.stab, global.stab = global.stab))
}
```

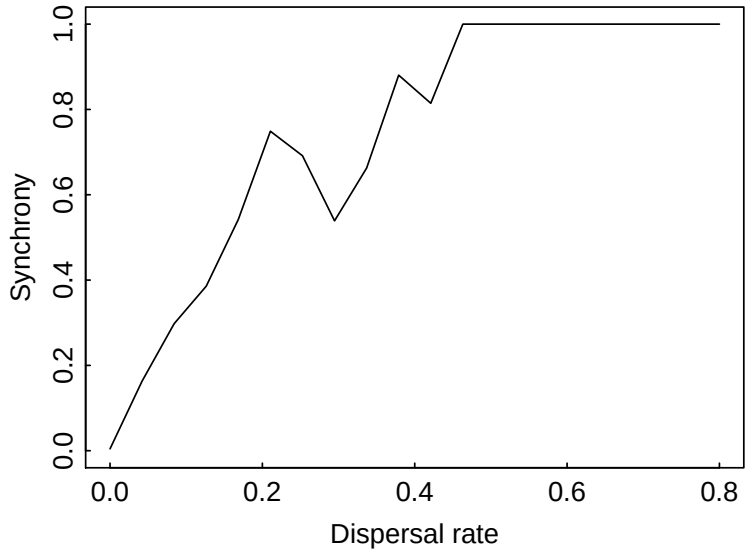
3. Once your function is written, launch a simulation for 20 linearly spaced dispersal rates ranging from 0 to 0.8. For each simulation, use the function you wrote to compute the stability and synchrony of the prey over the final 100 time steps. Do this for 10 replicate simulations and average the results.

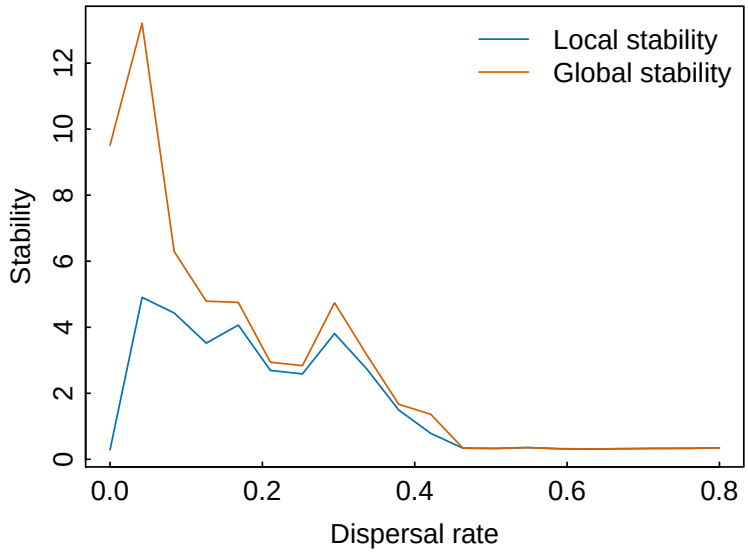
```
# Load necessary libraries
source("http://faraway.neu.edu/data/lattice.R")
library(synchrony)
library(deSolve)
library(fields)
# Parameters
dimension <- 32  # Lattice dimension (1D)
size <- dimension^2  # Lattice size (2D)
max.time <- 2000  # Total time of simulation
r <- 0.5  # Prey growth rate
K <- 15  # Prey carrying capacity
a <- 0.5  # Pred attack rate
h <- 0.2  # Pred handling time
m <- 0.1  # Pred mortality
b <- 1  # Pred efficiency
disp <- 1  # Dispersal rate
nreps <- 10
niter <- 1
neighs <- set.neighbors(dimension = dimension)
times <- 0:max.time
start.time <- max.time - 100
disp <- seq(from = 0, to = 0.8, len = 20)
results <- matrix(nrow = length(disp), ncol = 4)
parms <- list(r = r, K = K, m = m, h = h, a = a, b = b, dimension = dimension,
    size = size, neighs = neighs)
for (i in 1:length(disp)) {
    results.tmp <- matrix(nrow = nreps, ncol = 4)
    for (j in 1:nreps) {
        parms$disp <- disp[i]
        initial.densities <- runif(size * 2)
        sim <- ode(y = initial.densities, times = times, func = solve.rm,
            parms, method = "ode45")
        timeseries <- get.timeseries(sim, start.time = start.time, dimension = dimension)
        stats <- compute.stats(timeseries$prey)
        results.tmp[j, ] <- c(disp[i], stats$local.stab, stats$global.stab,
            kendall.w(timeseries$prey)$w.corrected)
```

```
        niter <- niter + 1
    }
    results[i, ] <- colMeans(results.tmp)
}
results <- as.data.frame(results)
colnames(results) <- c("dispersal", "local.stability", "global.stability",
    "synchrony")
```

4. Generate a 1-row by 2-column figure and plot local and global stability on the top panel and synchrony on the bottom panel as a function of dispersal rate. Interpret the results.

```
par(mfrow = c(2, 1))
plot(results$dispersal, results$synchrony, t = "l", ylim = c(0, 1), xlab = "Dispersal rate",
    ylab = "Synchrony")
plot(results$dispersal, results$local.stability, t = "l", col = cb["blue"],
    ylim = range(results$local.stability, results$global.stability), ylab = "Stability",
    xlab = "Dispersal rate")
lines(results$dispersal, results$global.stability, t = "l", col = cb["red"])
legend(x = "topright", legend = c("Local stability", "Global stability"),
    lty = 1, col = c(cb["blue"], cb["red"]), bty = "n")
```






```
solve.model <- function(t, y, parms) {
    with(parms, {
        y <- ifelse(y < 0, 0, y)
        N <- matrix(y, nrow = npops, ncol = 2)
        recruit_sp1 <- matrix(0, nrow = npops, ncol = 1)
        recruit_sp2 <- matrix(0, nrow = npops, ncol = 1)
        # Dispersal from source population i
        for (i in 1:npops) {
            l <- length(n[[i]])
            if (l > 1) {
                neighs <- n[[i]][2:l]
                recruit_sp1[neighs] <- recruit_sp1[neighs] + (N[i, 1] *
                  d)/(l - 1)
```

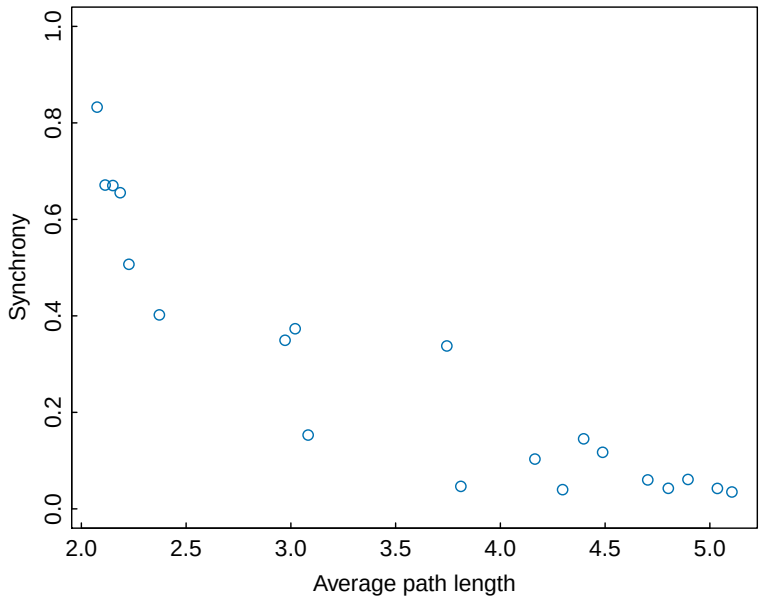
```
                recruit_sp2[neighs] <- recruit_sp2[neighs] + (N[i, 2] *
                  d)/(l - 1)
            }
        }
        # Integrate differential equations
        dy1 <- (r * N[, 1]) * (1 - N[, 1]/K) - (a * N[, 1] * N[, 2])/(1 +
            a * h * N[, 1]) - d * N[, 1] + recruit_sp1
        dy2 <- a * b * N[, 1] * N[, 2]/(1 + a * h * N[, 1]) - m * N[, 2] -
            d * N[, 2] + recruit_sp2
        return(list(c(dy1, dy2)))
    })
}
```

5. Now use the parameters below to simulate the dynamics of the model for each level of `power`. Note that for each level of `power`, you will need to compute the average path length and the synchrony of the prey over the final 100 time steps.

```
# Load required libraries
library(igraph)
require(deSolve)
require(igraph)
require(fields)
library(synchrony)
# Parameters
npops <- 50
max.time <- 500
start.time <- 400
a <- 0.2
m <- 0.2
d <- 0.01
r <- 0.5
K <- 40
h <- 1
b <- 1
# Random initial abundances
initial.abund <- cbind(runif(npops), runif(npops))
# Generate scale-free networks with different degree distros
power <- seq(from = 0, to = 2, len = 20)
mat <- matrix(nrow = length(power), ncol = 2)
for (i in 1:length(power)) {
    g <- barabasi.game(npops, directed = FALSE, power = power[i])
    n <- neighborhood(g, 2)
    parms <- list(a = a, b = b, h = h, r = r, K = K, m = m, d = d, npops = npops,
        n = n)
    results <- lsoda(y = initial.abund, func = solve.model, times = 0:max.time,
        parms = parms)
    mat[i, ] <- c(average.path.length(g), kendall.w(results[start.time:max.time,
        2:(npops + 1)])$w.corrected)
}
```

6. Plot synchrony as a function of average path length and interpret the results.

```
plot(mat, ylim = c(0, 1), xlab = "Average path length", ylab = "Synchrony",
    col = cb["blue"])
```



Answer:

The results show that synchrony decreases with increasing average path length. This means that better connected graphs (i.e., where the average shortest distance between pairs of nodes is smaller) tend to exhibit greater levels of synchrony than more poorly connected graphs.