Ecological Dynamics
Lab 3: Competition

Background

- Competition is ubiquitous in nature and arises because of the inherent conflict between exponential biological growth and limited resources.
- Understanding how species coexist despite competing for the same limiting resources is one of the main goals of ecology.
- Both phenomenological and mechanistic models predict that competitive exclusion will occur unless species limit themselves more than each other.

Objectives

- Learn to use the `manipulate` package to determine the stability of model equilibria.
- Implicitly evaluate the eigenvalues of each equilibrium to plot stability in parameter space.
- Extend resource competition models to account for P species and non-chemostat conditions.

Instructions

- Launch RStudio, open a new script file and save it as `lab3-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).

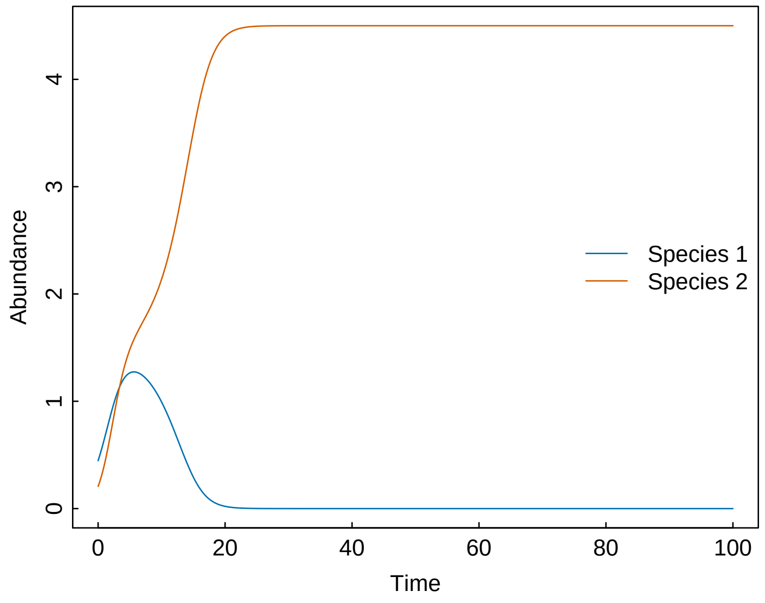
Task 1: Verifying and plotting local stability conditions

To verify the local stability results that were described in the lecture and derived analytically, we will use the `manipulate` package to plot the ZNGI as a function of the model parameters for the 2-species Lotka-Volterra model in continuous-time. We will also overlay simulations of the model for different initial conditions to determine whether the analytical results are correct. This will be a little tricky, so we will take it one step at a time.

1. Begin by writing a function that can be sent to function `ode` from package `deSolve` in order to solve the continuous-time version of the 2-species Lotka-Volterra model. Verify that your function works by plotting the dynamics for an arbitrary set of parameter values.

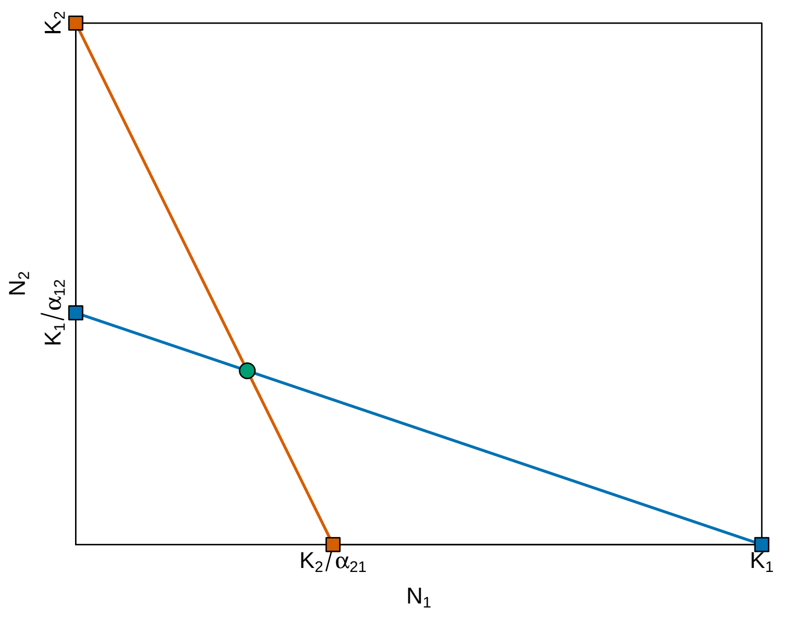
```
library(deSolve)
solve.lv <- function(t, y, parms) {
    N1 <- y[1]
    N2 <- y[2]
    with(parms, {
        dN1 <- r1 * N1 * ((K1 - N1 - a12 * N2)/K1)
        dN2 <- r2 * N2 * ((K2 - N2 - a21 * N1)/K2)
        return(list(c(dN1, dN2)))
```

```r
    })
}
r1 <- 0.5
r2 <- 1
K2 <- 4.5
K1 <- 6
a21 <- 2
a12 <- 3
parms <- list(r1 = r1, r2 = r2, K1 = K1, K2 = K2, a12 = a12, a21 = a21)
results <- ode(t = seq(from = 0, to = 100, len = 1000), parms = parms,
    y = runif(2), func = solve.lv)
plot(results[, 1], results[, 2], t = "l", col = cb["blue"], ylim = range(results[,
    2:3]), ylab = "Abundance", xlab = "Time")
lines(results[, 1], results[, 3], t = "l", col = cb["red"])
legend(x = "right", legend = c("Species 1", "Species 2"), lty = 1, lwd = 1,
    col = cb[c("blue", "red")], bty = "n")
```



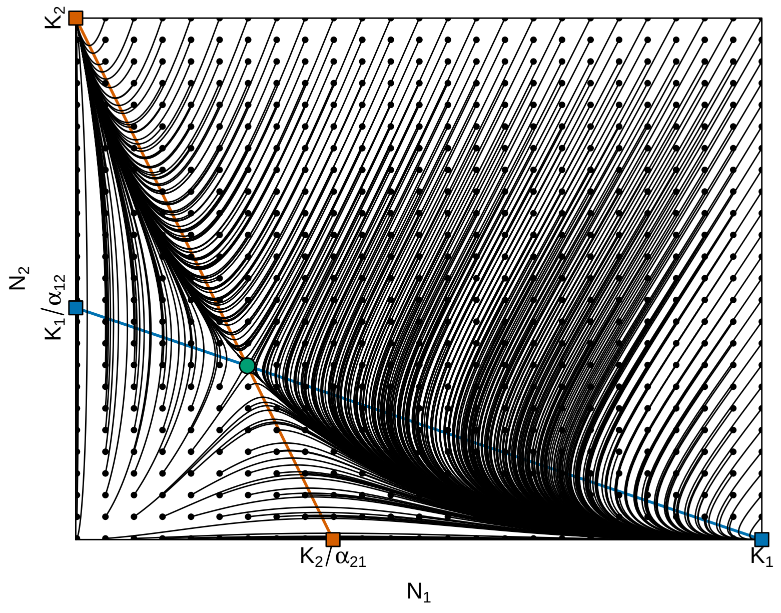
2. Using your previous code as an example, plot the ZNGI for both species using different colored lines and add all equilibria as points. Does your plot make sense in light of the simulation that you ran to answer the previous question?

```r
plot(x = 0, y = 0, t = "n", xlim = c(0, K1), ylim = c(0, K2), xlab = expression(N[1]),
    ylab = expression(N[2]), xaxt = "n", yaxt = "n", xaxs = "i", yaxs = "i")
axis(1, at = c(K1, K2/a21), lab = c(expression(K[1]), expression(K[2]/alpha[21])))
axis(2, at = c(K2, K1/a12), lab = c(expression(K[2]), expression(K[1]/alpha[12])))
lines(c(0, K2/a21), c(K2, 0), col = cb["red"], lwd = 2)
lines(c(0, K1), c(K1/a12, 0), col = cb["blue"], lwd = 2)
# Equilibria
points(K2/a21, 0, pch = 22, bg = cb["red"], xpd = NA, cex = 1.5)
points(0, K1/a12, pch = 22, bg = cb["blue"], xpd = NA, cex = 1.5)
points(K1, 0, pch = 22, bg = cb["blue"], xpd = NA, cex = 1.5)
points(0, K2, pch = 22, bg = cb["red"], xpd = NA, cex = 1.5)
points((K1 - a12 * K2)/(1 - a12 * a21), (K2 - a21 * K1)/(1 - a12 * a21),
    xpd = NA, cex = 1.5, pch = 21, bg = cb["green"])
```



3. We now need to overlay trajectories from simulations with different initial conditions. To do so, we will use function `expand.grid` to create a matrix containing all combinations of the values in 2 separate vectors. These vectors will specify the initial abundances of N_1 and N_2, with N_1 ranging from 0.01 to K_1 in 25 increments and N_2 ranging from 0.01 to K_2 in 25 increments. For each of these combinations of N_1 and N_2, you will need to simulate the dynamics of the model and then plot a line on the ZNGI plot that you generated in the previous question. Note that each simulation should be run from $t = 0$ to $t = 10$ in 100 linearly-spaced increments.

```r
plot(x = 0, y = 0, t = "n", xlim = c(0, K1), ylim = c(0, K2), xlab = expression(N[1]),
    ylab = expression(N[2]), xaxt = "n", yaxt = "n", xaxs = "i", yaxs = "i")
axis(1, at = c(K1, K2/a21), lab = c(expression(K[1]), expression(K[2]/alpha[21])))
axis(2, at = c(K2, K1/a12), lab = c(expression(K[2]), expression(K[1]/alpha[12])))
lines(c(0, K2/a21), c(K2, 0), col = cb["red"], lwd = 2)
lines(c(0, K1), c(K1/a12, 0), col = cb["blue"], lwd = 2)
grid <- expand.grid(N1.locs = seq(from = 0.01, to = K1, len = 25), N2.locs = seq(from = 0.01,
    to = K2, len = 25))
times <- seq(from = 0, to = 10, len = 100)
for (i in 1:NROW(grid)) {
    N0 <- as.numeric(grid[i, ])
    sol <- ode(t = times, parms = parms, y = N0, func = solve.lv)
    lines(sol[, 2], sol[, 3], lwd = 1)
    points(sol[1, 2], sol[1, 3], pch = 21, col = "black", bg = "black",
        cex = 0.5)
}
# Equilibria
points(K2/a21, 0, pch = 22, bg = cb["red"], xpd = NA, cex = 1.5)
points(0, K1/a12, pch = 22, bg = cb["blue"], xpd = NA, cex = 1.5)
points(K1, 0, pch = 22, bg = cb["blue"], xpd = NA, cex = 1.5)
points(0, K2, pch = 22, bg = cb["red"], xpd = NA, cex = 1.5)
points((K1 - a12 * K2)/(1 - a12 * a21), (K2 - a21 * K1)/(1 - a12 * a21),
    xpd = NA, cex = 1.5, pch = 21, bg = cb["green"])
```



4. Now that you have developed the code needed to plot the ZNGI for a specific set of parameter values, it is time to design a function to do all of this work for an arbitrary set of parameter values and place it within the `manipulate` function. Doing so will allow you to explore the behavior of the model in real-time. Your `manipulate` code should allow you to alter all model parameters along with those of the simulations (i.e., the number of time steps, maximum time, spacing between initial conditions).

```
library(manipulate)
plot.lv <- function(nsteps = 1000, parms, max.time.sims, grid.start = 0.025,
    grid.by = 1) {
    with(parms, {
        par(tck = 0.01, mar = c(4, 4, 2, 1), mgp = c(1.5, 0.2, 0))
        plot(x = 0, y = 0, t = "n", xlim = c(0, K1), ylim = c(0, K2), xlab = expression(N[1]),
            ylab = expression(N[2]), xaxt = "n", yaxt = "n", xaxs = "i",
            yaxs = "i")
        axis(1, at = c(K1, K2/a21), lab = c(expression(K[1]), expression(K[2]/alpha[21])))
        axis(2, at = c(K2, K1/a12), lab = c(expression(K[2]), expression(K[1]/alpha[12])))
        lines(c(0, K2/a21), c(K2, 0), col = cb["red"], lwd = 2)
        lines(c(0, K1), c(K1/a12, 0), col = cb["blue"], lwd = 2)
        grid <- expand.grid(N1.locs = seq(from = grid.start, to = K1, by = grid.by),
            N2.locs = seq(from = grid.start, to = K2, by = grid.by))
        times <- seq(from = 0, to = max.time.sims, len = nsteps)
        for (i in 1:NROW(grid)) {
            N0 <- as.numeric(grid[i, ])
            sol <- ode(t = times, parms = parms, y = N0, func = solve.lv)
            lines(sol[, 2], sol[, 3], lwd = 1)
            points(sol[1, 2], sol[1, 3], pch = 21, col = "black", bg = "black",
                cex = 0.5)
        }
        # Equilibria
        points(K2/a21, 0, pch = 22, bg = cb["red"], xpd = NA, cex = 1.5)
        points(0, K1/a12, pch = 22, bg = cb["blue"], xpd = NA, cex = 1.5)
        points(K1, 0, pch = 22, bg = cb["blue"], xpd = NA, cex = 1.5)
        points(0, K2, pch = 22, bg = cb["red"], xpd = NA, cex = 1.5)
```

```r
        points((K1 - a12 * K2)/(1 - a12 * a21), (K2 - a21 * K1)/(1 - a12 *
            a21), xpd = NA, cex = 1.5, pch = 21, bg = cb["green"])
    })
}
r1 <- 0.5
r2 <- 1
K2 <- 4.5
K1 <- 6
a21 <- 2
a12 <- 3
manipulate({
    r1 <- r1.val
    r2 <- r2.val
    K1 <- K1.val
    K2 <- K2.val
    a12 <- a12.val
    a21 <- a21.val
    parms <- list(r1 = r1, r2 = r2, K1 = K1, K2 = K2, a21 = a21, a12 = a12)
    plot.lv(nsteps = nsteps.val, parms, max.time.sims = time.sims.val,
        grid.start.val, grid.by.val)
}, nsteps.val = slider(min = 50, max = 1000, initial = 100, label = "Number of timesteps (t)"),
    time.sims.val = slider(min = 4, max = 100, initial = 20, label = "Max timestep (t)"),
    r1.val = slider(min = 0, max = 4, initial = 0.2, step = 0.05, label = "Growth rate (r1)"),
    r2.val = slider(min = 0, max = 4, initial = 0.1, step = 0.05, label = "Growth rate (r2)"),
    K1.val = slider(min = 1, max = 50, initial = 7, step = 1, label = "Carrying capacity (K1)"),
    K2.val = slider(min = 0.1, max = 50, initial = 5, step = 1, label = "Carrying capacity (K2)"),
    a12.val = slider(min = 0, max = 10, initial = 1, step = 0.05, label = "Competitive effect (a12)"),
    a21.val = slider(min = 0, max = 10, initial = 1, step = 0.05, label = "Competitive effect (a21)"),
    grid.start.val = slider(min = 0.025, max = 2, initial = 0.1, step = 0.05,
        label = "Grid start"), grid.by.val = slider(min = 0.3, max = 2,
        initial = 0.5, step = 0.05, label = "Grid by"))
```

5. Theoretical papers often plot the locally stable equilibrium as a function of parameters using heatmaps, with each color corresponding to a different locally stable equilibrium. To do so for the 2-species Lotka-Volterra model, you will need to download and load the following package and accessory function:

```r
# Only install the package once: install.packages('fields')
library(fields)
source("http://faraway.neu.edu/data/format3d.R")
```

Now, write a function that takes the model parameter values and computes the stability of each equilibrium by determining whether the real part of the eigenvalues are strictly negative. Note that this function will need to return the identify of the stable equilibrium in the form of a number (i.e., 1: extinction, 2: N_1 monoculture, 3: N_2 monoculture, 4: coexistence, 5: alternative stable states).

```r
# Function to compute the jacobian and return the dominant eigenvalue
max.eig <- function(parms) {
    with(parms, {
        jacobian <- rbind(c((r1/K1) * (K1 - 2 * N1 - a12 * N2), (-r1/K1) *
            a12 * N1), c((-r2/K2) * a21 * N2, (r2/K2) * (K2 - 2 * N2 -
            a21 * N1)))
        return(max(Re(eigen(jacobian)$values)))
    })
}
# Function to determine the stability of each equilibrium
determine.stability <- function(parms) {
    with(parms, {
        # Find eigenvalues
        parms$N1 <- 0
        parms$N2 <- 0
        eig.ext <- max.eig(parms)
        parms$N1 <- K1
        parms$N2 <- 0
        eig.N1 <- max.eig(parms)
        parms$N1 <- 0
        parms$N2 <- K2
        eig.N2 <- max.eig(parms)
        parms$N1 <- (K1 - a12 * K2)/(1 - a12 * a21)
        parms$N2 <- (K2 - a21 * K1)/(1 - a12 * a21)
        eig.coex <- max.eig(parms)
        result <- numeric(length = 4)
        names(result) <- c("ext", "N1", "N2", "coex")
        if (eig.ext < 0)
            result[1] <- 1
        if (eig.N1 < 0)
            result[2] <- 1
        if (eig.N2 < 0)
            result[3] <- 1
        if (eig.coex < 0 & parms$N1 > 0 & parms$N2 > 0)
            result[4] <- 1
        if (sum(result) > 1) {
            stab <- 5
        } else {
            stab <- which(result == 1)
        }
        return(stab)
    })
}
```

6. Once your function is written, use it to determine the stable equilibrium for all combinations of the following parameters: $\alpha_{12} = 0.5$, $K_1 = 3$, $r_1 = r_2 = 2$, α_{21} ranging from 0 to $5\alpha_{12}$ via 100 linearly-spaced increments, and K_2 ranging from 0.01 to $5K_1$ via 100 linearly-spaced increments. Store the results in a matrix called `results` containing 3 columns: K_2/K_1, α_{21}/α_{12}, and the stability value.

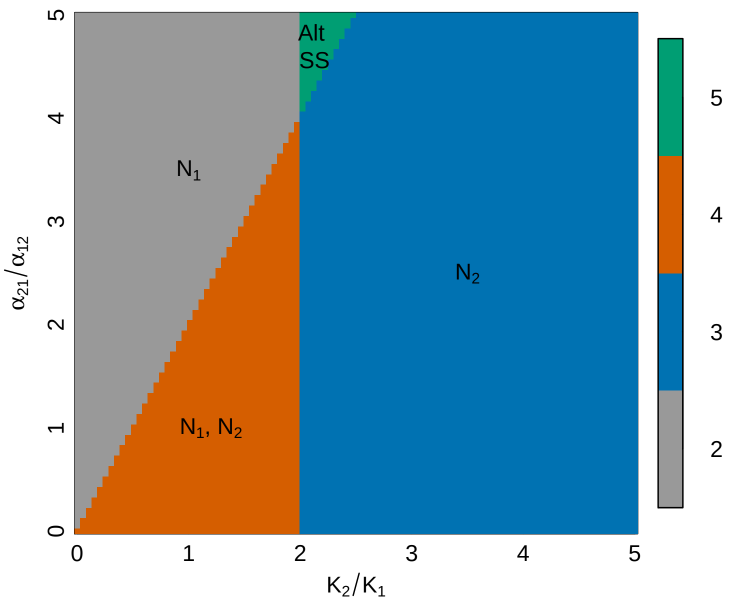
```
a12 <- 0.5
K1 <- 3
r1 <- 2
r2 <- 2
grid <- expand.grid(K2 = seq(from = 0.01, to = 5 * K1, len = 100), a21 = seq(from = 0,
    to = 5 * a12, len = 100))
results <- matrix(nrow = NROW(grid), ncol = 3, 0)
for (i in 1:NROW(grid)) {
    parms <- list(r1 = r1, r2 = r2, K1 = K1, K2 = grid[i, "K2"], a12 = a12,
        a21 = grid[i, "a21"])
    results[i, ] <- c(as.numeric(grid[i, ])/c(K1, a12), determine.stability(parms))
}
```

7. Convert your `results` matrix of parameters and corresponding stability values by using function `format3d`:

```
results.3d <- format3d(results)
```

Then plot `results.3d` using the `image.plot` function from the `fields` package. Can you interpret the figure?

```
library(fields)
source("http://faraway.neu.edu/data/format3d.R")
data.3d <- format3d(results)
image.plot(data.3d, ylab = expression(alpha[21]/alpha[12]), xlab = expression(K[2]/K[1]),
    col = cb[c("grey", "blue", "red", "green")])
# Label regions
text(x = 1.2, y = 1, label = expression(paste(N[1], ", ", N[2])))
text(x = 3.5, y = 2.5, label = expression(N[2]))
text(x = 1, y = 3.5, label = expression(N[1]))
text(x = 2.1, y = 4.7, label = paste("Alt\n", "SS"))
```



Task 2: Extending and simulating mechanistic models of competition

1. The mechanistic competition models described during the lecture were based on a simple chemostat system where the flow rate F controlled both the supply and the loss of resources R. Extend the model with two consumers N_1 and N_2 competing for a single resource R beyond chemostat scenarios. Explain your approach.

Answer:

In most systems, the inflow or supply of nutrients F does not equal their outflow or 'mortality'. To extend this modeling approach, we simply need to decouple the supply S and loss δ of resources in the model:

$$\begin{aligned}
\frac{\mathrm{d}N_1}{\mathrm{d}t} &= N_1\left(\mu_1\frac{R}{K_1+R} - M_1\right)\\
\frac{\mathrm{d}N_2}{\mathrm{d}t} &= N_2\left(\mu_2\frac{R}{K_2+R} - M_2\right)\\
\frac{\mathrm{d}R}{\mathrm{d}t} &= S - \delta R - N_1\frac{\mu_1}{Y_1}\frac{R}{K_1+R} - N_2\frac{\mu_2}{Y_2}\frac{R}{K_2+R}
\end{aligned}$$

In this case, S is the supply rate of resource R in units of concentration per time and δ is the loss rate of resource R in units of per time.

2. Does your more realistic model alter the predictions of the simpler model derived during the lecture? For instance, does the competitive exclusion principle still apply to your model extension? Explain why or why not.

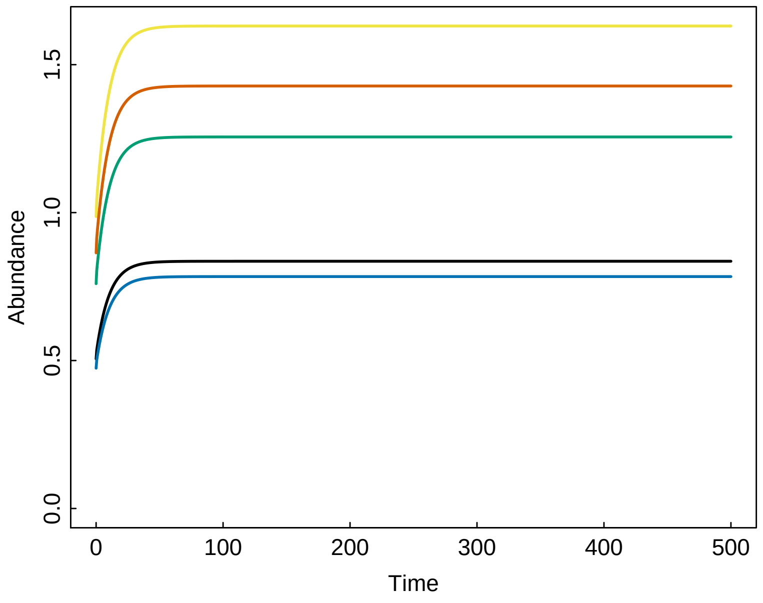
Answer:
The extended model described above yields the same fundamental insights as the simpler model because relaxing the assumption of equal inflow and outflow will only impact the equilibrium concentration of the resource R, not the conditions required for consumer coexistence.

3. Write a function to simulate the dynamics of a mechanistic competition model with P consumers N_1 through N_P and one resource R. Verify that coexistence can only occur if all consumers have the same R^* value by simulating the dynamics when one species has a lower R^* value than the rest.

Answer:
All consumers persist when they have the same R^*:

```r
library(deSolve)
solve.psp <- function(t, y, parms) {
    with(parms, {
        dy <- y[1:(length(y) - 1)] * (mu * y[length(y)]/(K + y[length(y)]) -
            M)
        dr <- F * (S - y[length(y)]) - sum(y[1:(length(y) - 1)] * ((mu/Y) *
            (y[length(y)]/(K + y[length(y)]))))
        return(list(c(dy, dr)))
    })
}
nspecies <- 5
M <- rep(0.1, nspecies)
Y <- rep(0.2, nspecies)
```

```
K <- rep(0.2, nspecies)
mu <- rep(0.7, nspecies)
S <- 3
F <- 1
parms <- list(M = M, Y = Y, K = K, mu = mu, S = S, F = F)
times <- seq(from = 0, to = 500, len = 1000)
sim <- ode(y = runif(nspecies + 1), parms = parms, times = times, func = solve.psp)
matplot(sim[, 1], sim[, 2:(NCOL(sim) - 1)], t = "l", lty = 1, col = cb[c(1,
    5:8)], lwd = 2, xlab = "Time", ylab = "Abundance", ylim = range(0,
    range(sim[, 2:(NCOL(sim) - 1)])))
```

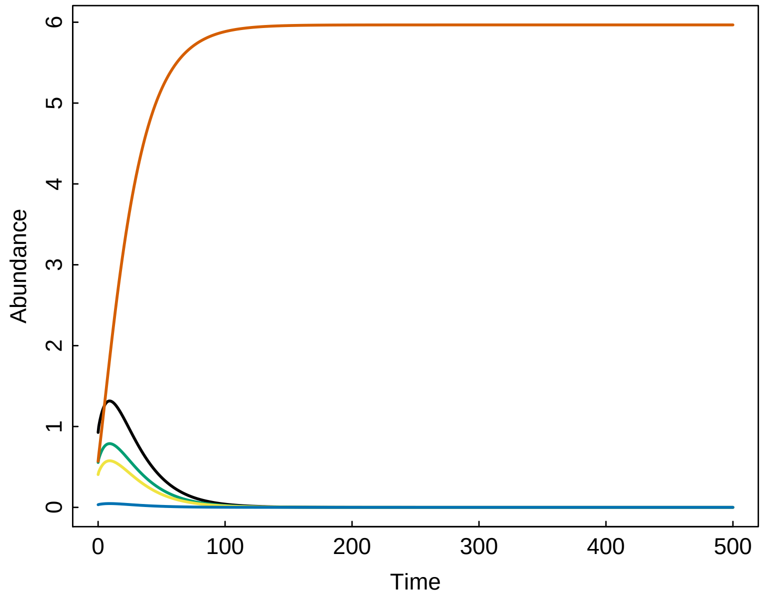


Answer:

The consumer with the lowest R^* outcompetes all others:

```
library(deSolve)
solve.psp <- function(t, y, parms) {
    with(parms, {
        dy <- y[1:(length(y) - 1)] * (mu * y[length(y)]/(K + y[length(y)]) -
            M)
        dr <- F * (S - y[length(y)]) - sum(y[1:(length(y) - 1)] * ((mu/Y) *
            (y[length(y)]/(K + y[length(y)]))))
        return(list(c(dy, dr)))
    })
}
nspecies <- 5
M <- rep(0.1, nspecies)
Y <- rep(0.2, nspecies)
K <- rep(0.2, nspecies)
K[length(K)] <- 0.1
mu <- rep(0.7, nspecies)
S <- 3
F <- 1
parms <- list(M = M, Y = Y, K = K, mu = mu, S = S, F = F)
```

```
times <- seq(from = 0, to = 500, len = 1000)
sim <- ode(y = runif(nspecies + 1), parms = parms, times = times, func = solve.psp)
matplot(sim[, 1], sim[, 2:(NCOL(sim) - 1)], t = "l", lty = 1, col = cb[c(1,
    5:8)], lwd = 2, xlab = "Time", ylab = "Abundance", ylim = range(0,
    range(sim[, 2:(NCOL(sim) - 1)])))
```


Abundance
Time