Ecological_Dynamics_-_Final___no_A_.R

jcamp_000

Fri Apr 29 14:51:14 2016

```
rm(list=ls())
library(fields)
```

```
## Warning: package 'fields' was built under R version 3.2.4

## Loading required package: spam

## Loading required package: grid

## Spam version 1.3-0 (2015-10-24) is loaded.
## Type 'help( Spam)' or 'demo( spam)' for a short introduction
## and overview of this package.
## Help for individual functions is also obtained by adding the
## suffix '.spam' to the function name, e.g. 'help( chol.spam)'.

##
## Attaching package: 'spam'

## The following objects are masked from 'package:base':
##
##     backsolve, forwardsolve

## Loading required package: maps

##
##  # ATTENTION: maps v3.0 has an updated 'world' map.        #
##  # Many country borders and names have changed since 1990. #
##  # Type '?world' or 'news(package="maps")'. See README_v3. #
```

```
library(deSolve)
```

```
##
## Attaching package: 'deSolve'

## The following object is masked from 'package:graphics':
##
##     matplot
```

```
#install.packages("hash")
library(hash)
```

```
## Warning: package 'hash' was built under R version 3.2.4

## hash-2.2.6 provided by Decision Patterns
```

```r
periodic <- function (x, dimension = 128) {
  if (x > dimension)
    x <- x - dimension
  else if (x < 1)
    x <- dimension + x
  return (x)
}
```

```r
pos.from.coords <- function (x, y, dimension = 128) {
  pos <- y + (x-1) * dimension
  return(pos)
}
```

```r
coords.from.pos <- function (pos, dimension = 128) {
  x <- (pos - 1) %/% dimension + 1
  y <- (pos - 1) %% dimension + 1
  return(list(y=y, x=x))
}
```

```r
find.neighs4 <- function (x, y, dimension = 128) {
  xplus <- periodic(x+1, dimension)
  yplus <- periodic(y+1, dimension)
  xminus <- periodic(x-1, dimension)
  yminus <- periodic(y-1, dimension)

  # Return coordinates of neighbors
  return(list(x=c(x, x, xplus, xminus),
              y=c(yplus, yminus, y, y)))
}
```

```r
find.neighs8 <- function (x, y, dimension = 128) {
  xplus <- periodic(x + 1, dimension)
  yplus <- periodic(y + 1, dimension)
  xminus <- periodic(x - 1, dimension)
  yminus <- periodic(y - 1, dimension)

  # Return coordinates of neighbors
  return(list(x=c(x, x, xplus, xminus, xminus, xminus, xplus, xplus),
              y=c(yplus, yminus, y, y, yminus, yplus, yplus, yminus)))
}
```

```r
find.pos.neighs8 <- function (x, y, dimension = 128) {
  xplus <- periodic(x + 1, dimension)
  yplus <- periodic(y + 1, dimension)
  xminus <- periodic(x - 1, dimension)
  yminus <- periodic(y - 1, dimension)

  positions <- c(pos.from.coords(x = x, y = yplus, dimension = dimension),
                 pos.from.coords(x = x, y = yminus, dimension = dimension),
                 pos.from.coords(x = xplus, y = y, dimension = dimension),
                 pos.from.coords(x = xminus, y = y, dimension = dimension),
                 pos.from.coords(x = xminus, y = yminus, dimension = dimension),
                 pos.from.coords(x = xminus, y = yplus, dimension = dimension),
                 pos.from.coords(x = xplus, y = yplus, dimension = dimension),
                 pos.from.coords(x = xplus, y = yminus, dimension = dimension))

  # Return positions of neighbors
  return(positions)
}
```

```
find.pos.neighs4 <- function (x, y, dimension = 128) {
  xplus <- periodic(x+1, dimension)
  yplus <- periodic(y+1, dimension)
  xminus <- periodic(x-1, dimension)
  yminus <- periodic(y-1, dimension)

  positions <- c(pos.from.coords(x = x, y = yplus, dimension = dimension),
                 pos.from.coords(x = x, y = yminus, dimension = dimension),
                 pos.from.coords(x = xplus, y = y, dimension = dimension),
                 pos.from.coords(x = xminus, y = y, dimension = dimension))

  # Return positions of neighbors
  return(positions)
}
```

```
set.neighbors <- function (dimension = 128, size = dimension^2, n.neighs = 8) {
  neighs <- matrix(nrow = size, ncol = n.neighs + 1)
  for (p in 1:size) {
    # Find coordinates for position p
    p.coords <- coords.from.pos(p, dimension = dimension)
    # Find neighbors for position p
    all.neighs <- find.pos.neighs8(y = p.coords$y,
                                   x = p.coords$x,
                                   dimension = dimension)
    neighs[p, ] <- c(p, all.neighs)
  }
  return (neighs)
}
```

```
rules.pred.prey <- function(lattice, parms) {
  # Empty = 1, # Prey = 2 - 101, # Pred = 102
  updated.lattice <- lattice
  with(parms, {
    rand.locs <- sample(1:size)
    for (i in rand.locs) {
      if (lattice[i] == 1) {
        # Empty
        colonizer <- strongest.colonizer(i, lattice, parms)
        if (r.vec[colonizer] > runif(1)) {
          updated.lattice[i] <- colonizer
        }
      } else if (2 <= lattice[i] & lattice[i] <= 101) {
        # Prey
        if (m > runif(1)) {
          updated.lattice[i] <- 1
        } else if (sum(lattice[neighs[i, 2:9]] == 102) > 0) {
          if (a > d.vec[lattice[i] - 1]) {
            updated.lattice[i] <- 102
          }
        }
      } else if (lattice[i] == 102) {
        # Predator
        if (m > runif(1)) {
          updated.lattice[i] <- 1
        }
      }
    }
    return(updated.lattice)
  })
}

strongest.colonizer <- function(i, lattice, parms) {
  n.freq <- hash() # Type -> Freq
  n.str <- hash() # Type -> Strength
  with(parms, {
    neigh.types <- as.character(lattice[neighs[i, 2:9]])
    # Set neighbor type frequencies
    for (n in neigh.types) {
      if (has.key(n, n.freq)) {
        .set(n.freq, keys=n, values=get(n, n.freq) + 1)
      } else {
        .set(n.freq, keys=n, values=1)
      }
    }
    # Determine strongest neighbor
    for (n in keys(n.freq)) {
      str <- r.vec[as.numeric(n)] * 0.125 * get(n, n.freq)
      .set(n.str, keys=n, values=str)
    }
    max.str <- max(values(n.str))
    n.str.inv <- invert(n.str)
    strongest <- sample(as.numeric(get(as.character(max.str), n.str.inv)), 1)
    return(strongest)
  })
}
```

```
th.log <- function(x, theta = 1) {
  return(1 - x^theta)
}
```

```
rel.abundances <- function(lattice, t, size) {
  r.a.vec <- numeric(102)
  for (p in 1:102) {
    r.a.vec[p] <- sum(lattice[, , t] == p) / size * 100
  }
  return(r.a.vec)
}
```

```r
dimension <- 64                         # Lattice dimension (1D)
size <- dimension^2                     # Lattice size (2D)
timeval <- 350                          # Total time of simulation
d.vec <- seq(0, .99, length.out = 100)  # Defensive abilities of prey
theta.vec <- c(.25, 1, 4)               # Theta to determine corresponding reproductive abilities
m <- 0.15                               # Mortality rate of prey/predator
a = 1                                   # Predation strength of predator
```

```r
set.seed(1)
neighs <- set.neighbors(dimension = dimension)
lattice.1 <- array(sample(size = size, x = 1:102, prob = c(.40, rep(.005, 100), .10), replace = TRUE),
                   dim = c(dimension, dimension, timeval))
lattice.2 <- array(sample(size = size, x = 1:102, prob = c(.40, rep(.005, 100), .10), replace = TRUE),
                   dim = c(dimension, dimension, timeval))
lattice.3 <- array(sample(size = size, x = 1:102, prob = c(.40, rep(.005, 100), .10), replace = TRUE),
                   dim = c(dimension, dimension, timeval))

rel.abun.1 <- matrix(nrow = 350, ncol = 102)
rel.abun.2 <- matrix(nrow = 350, ncol = 102)
rel.abun.3 <- matrix(nrow = 350, ncol = 102)


parms.1 <- list(d.vec = d.vec, r.vec = c(0, th.log(d.vec, theta.vec[1]), 0), m = m,
                a = a, dimension = dimension, size = size, neighs = neighs)
parms.2 <- list(d.vec = d.vec, r.vec = c(0, th.log(d.vec, theta.vec[2]), 0), m = m,
                a = a, dimension = dimension, size = size, neighs = neighs)
parms.3 <- list(d.vec = d.vec, r.vec = c(0, th.log(d.vec, theta.vec[3]), 0), m = m,
                a = a, dimension = dimension, size = size, neighs = neighs)
```

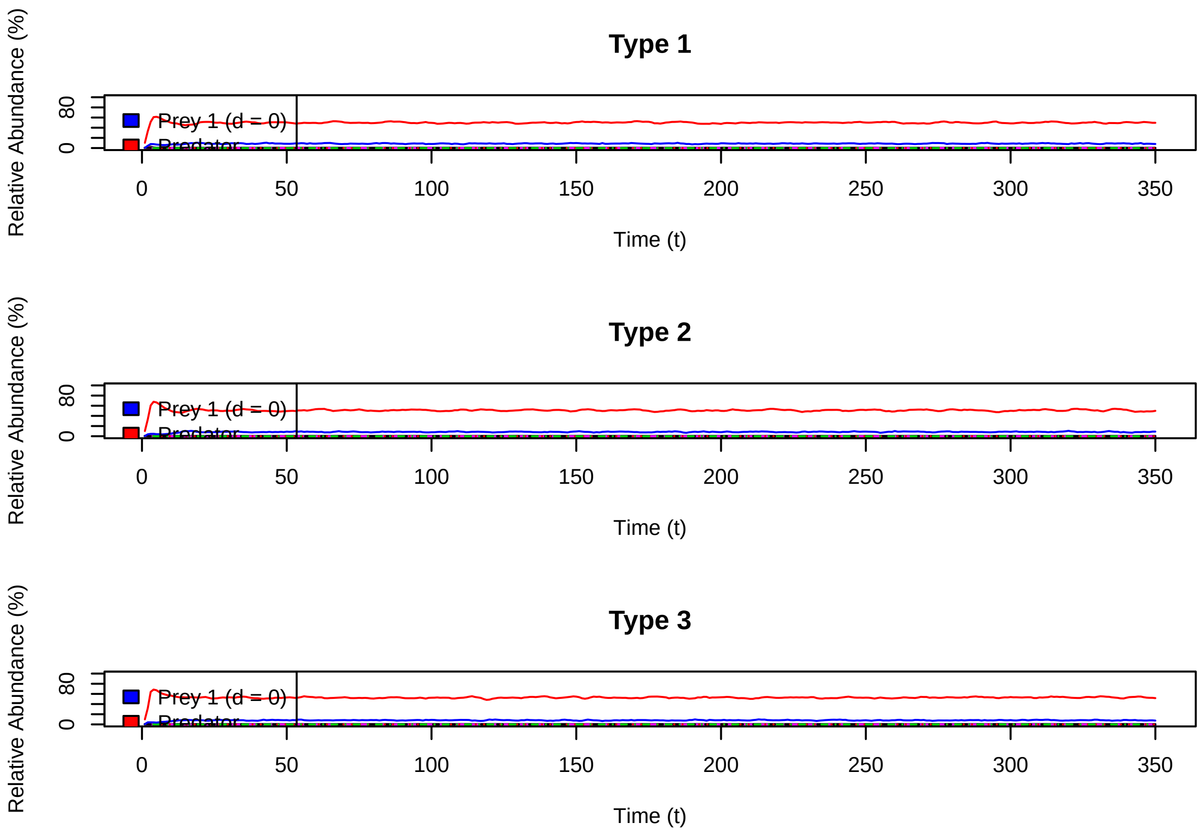
```r
start.time <- Sys.time()
print(start.time)
```

```
## [1] "2016-04-29 14:51:16 EDT"
```

```r
for (t in 2:timeval) {
  lattice.1[ , , t] <- rules.pred.prey(lattice.1[ , , t - 1], parms = parms.1)
  lattice.2[ , , t] <- rules.pred.prey(lattice.2[ , , t - 1], parms = parms.2)
  lattice.3[ , , t] <- rules.pred.prey(lattice.3[ , , t - 1], parms = parms.3)
}

for (t in 1:timeval) {
  rel.abun.1[t, ] <- rel.abundances(lattice.1, t, size)
  rel.abun.2[t, ] <- rel.abundances(lattice.2, t, size)
  rel.abun.3[t, ] <- rel.abundances(lattice.3, t, size)
}

end.time <- Sys.time()
print(end.time - start.time)
```

```
## Time difference of 30.18075 mins
```

```r
par(mfrow=c(3, 1))
matplot(1:timeval, rel.abun.1[,c(-1, -2, -102)], type = "l", pch = 2, ylim = c(0, 100),
        xlab = "Time (t)", ylab = "Relative Abundance (%)", main = "Type 1")
lines(1:timeval, rel.abun.1[,2], col = "blue", pch = 2)
lines(1:timeval, rel.abun.1[,102], col = "red", pch = 2)
legend(x = "topleft", legend = c("Prey 1 (d = 0)", "Predator"), fill = c("blue", "red"))

matplot(1:timeval, rel.abun.2[,c(-1, -2, -102)], type = "l", pch = 2, ylim = c(0, 100),
        xlab = "Time (t)", ylab = "Relative Abundance (%)", main = "Type 2")
lines(1:timeval, rel.abun.2[,2], col = "blue", pch = 2)
lines(1:timeval, rel.abun.2[,102], col = "red", pch = 2)
legend(x = "topleft", legend = c("Prey 1 (d = 0)", "Predator"), fill = c("blue", "red"))

matplot(1:timeval, rel.abun.3[,c(-1, -2, -102)], type = "l", pch = 2, ylim = c(0, 100),
        xlab = "Time (t)", ylab = "Relative Abundance (%)", main = "Type 3")
lines(1:timeval, rel.abun.3[,2], col = "blue", pch = 2)
lines(1:timeval, rel.abun.3[,102], col = "red", pch = 2)
legend(x = "topleft", legend = c("Prey 1 (d = 0)", "Predator"), fill = c("blue", "red"))
```


Type 1
Relative Abundance (%)
🟦 Prey 1 (d = 0)
🟥 Predator
Time (t)
Type 2
Relative Abundance (%)
🟦 Prey 1 (d = 0)
🟥 Predator
Time (t)
Type 3
Relative Abundance (%)
🟦 Prey 1 (d = 0)
🟥 Predator
Time (t)