Assignment9

Haoran Xu

2024-11-18

```r
## Cellular Automaton (Game of Life)

#initialise
n <- 20
grid <- matrix(sample(c(0,1),n^2,replace=TRUE),nrow=n)
timeSteps <- 10

#function to calculate the next state
nextState <- function(grid) {
  n <- nrow(grid)
  newGrid <- matrix(0,nrow=n,ncol=n)
  for (i in 1:n) {
    for (j in 1:n) {
      #this line of code calculates the number of living neighbours for a
      #given cell(i,j) in the grid, considering the boundaries of the grid,
      #and stores the result in the variable aliveNeighbors.
      # grid[i,j] or grid[(i-1):(i+1), (j-1):(j+1)]
      aliveNeighbors <- sum(grid[max(i-1,1):min(i+1,n), max(j-1,1):min(j+1,n)]) - grid[i,j] # uses the

      if (grid[i,j]==1 && (aliveNeighbors<2 || aliveNeighbors>3)) {
        newGrid[i,j] <- 0
      } else if (grid[i,j]==1 || aliveNeighbors==3) {
        newGrid[i,j] <- 1
      }
    }
  }
  return(newGrid)
}

#run it
for (t in 1:timeSteps) {
  grid <- nextState(grid)
  image(1:n,1:n,z=grid,col= c("white","black"),axes=FALSE,xlab="",ylab="")
  title(paste("Time step:", t))
  Sys.sleep(0.1) #pause for a second to see the evolution
}
```

Q1. Looking at the visual output ABM, briefly describe what appears to be happening on the output image. Does the pattern appear 'random', or is there something else going on?

The distribution of black dots is first random but the changes of patterns are not "random".

At first the black (alive) dots (`grid[i,j] = 1`) are randomly scattered on the `20 x 20` matrix, as `sample()` was used to create initial settings.

Then as the simulation progresses, the patterns looked more "structured", which means there are some parts where the changes of patterns would follow a direction or a set of progressive patterns (so-called "Oscillators" and "Spaceships"), and there are also some area where the pattern stay static ("Still lifes") after few iterations (I also tried enhancing the number of iterations to `50`).

The reason behind the "structuralism" is based on the rules we set, as the rules will make the grids becoming self-organized and predictable across time.

```r
gridSize <- 10
numAgents <- 15
timeSteps <- 10

#initialize agents
colours <- c("red","blue","pink","black","grey","green","cyan","yellow")
style <- c(19,20,21,22)
agents <- data.frame(x = sample(1:gridSize,numAgents,replace = TRUE),
                     y = sample(1:gridSize,numAgents,replace = TRUE),
                     c = sample(colours,numAgents,replace=TRUE),
                     p = sample(style,numAgents,replace=TRUE))

#agent behaviour
updateAgents <- function(agents) {
  for (i in 1:nrow(agents)) {
    move <- sample(c("up","down","left","right"),1)
    if (move == "up" && agents$y[i] < gridSize) agents$y[i] <- agents$y[i] + 1
    if (move == "down" && agents$y[i] > 1) agents$y[i] <- agents$y[i] - 1
    if (move == "left" && agents$x[i] > 1) agents$x[i] <- agents$x[i] - 1
    if (move == "right" && agents$x[i] < gridSize) agents$x[i] <- agents$x[i] + 1
  }
  return(agents)
}

#simulation
for (t in 1:timeSteps) {
  plot(agents$x,agents$y,xlim=c(1, gridSize),
       ylim=c(1,gridSize),
       pch=agents$p,col=agents$c,xlab="X",ylab="Y")
  title(paste("Time Step:", t))
  Sys.sleep(0.5)#pause to visualise...
  agents <- updateAgents(agents)
}
```

Q2. Looking at the code, describe what is happening in this ABM. Try to describe the behaviour of the agents with reference to the code.

This ABM simulates the random movement of agents within a defined grid over multiple timesteps.

Initially, each agent has randomly set properties in the dataframe, such as location (`x` and `y` coordinates), colors (`c`), and styles (`p`). The grid is 10x10.

Then, the behaviors of agents are defined by function `updateAgents` as randomly moving into four direction (`up`, `down`, `left`, `right`) by 1 unit each time within the grid. They move independently without influencing each other.

Later, the simulation process loops iterations 10 times, with each iteration drawing the grid showcasing the locations of 15 agents.

The graphs showed that the agents move independently and autonomously, and always confined in the grid.

```r
smooth_matrix <- function(matrix, filter) {
  padded_matrix <- matrix(0, nrow = nrow(matrix) + 4, ncol = ncol(matrix) + 4)
  padded_matrix[3:(nrow(matrix) + 2), 3:(ncol(matrix) + 2)] <- matrix

  smoothed_matrix <- matrix(0, nrow = nrow(matrix), ncol = ncol(matrix))

  for (i in 3:(nrow(matrix) + 2)) {
    for (j in 3:(ncol(matrix) + 2)) {
      window <- padded_matrix[(i - 2):(i + 2), (j - 2):(j + 2)]
      smoothed_matrix[i - 2, j - 2] <- sum(window * filter)
    }
  }
  return(smoothed_matrix)
}

forest_size <- 50
initial_trees <- 100
growth_rate <- 0.5
sunlight <- 0.2
water <- 0.2
colors <- colorRampPalette(c("green", "darkgreen"))(100)#colour palette
soil_colours <- colorRampPalette(c("brown", "black"))(100)#colour palette

#initialize forest grid
forest <- matrix(0, nrow = forest_size, ncol = forest_size)

#smoothed soil layer
soil <- matrix(runif(forest_size*forest_size), nrow = forest_size, ncol = forest_size)
filter <- matrix(1/25, nrow = 5, ncol = 5)
soil <- smooth_matrix(soil, filter)
image(soil, col = soil_colours, axes = FALSE, main = "Soil map")
axis(1, at = seq(0, 1, length.out = forest_size), labels = 1:forest_size)
axis(2, at = seq(0, 1, length.out = forest_size), labels = 1:forest_size)
```

```r
#plant initial trees randomly
for (i in 1:initial_trees) {
  x <- sample(1:forest_size, 1)
  y <- sample(1:forest_size, 1)
  forest[x, y] <- 1
}

#function to grow trees
grow_trees <- function(forest) {
  for (i in 2:(forest_size - 1)) {
    for (j in 2:(forest_size - 1)) {
      if (forest[i, j] > 0) {
        forest[i, j] <- forest[i, j] + growth_rate * sunlight * water
      }
    }
  }
  return(forest)
}

#function to spread trees
spread_trees <- function(forest, soil) {
  new_forest <- forest # Create a copy of the forest to store new trees
  for (i in 2:(forest_size - 1)) {
    for (j in 2:(forest_size - 1)) {
      if (forest[i, j] > 1) {
        #define neighbours (3 by 3 search model)
        neighbors_i <- c(i - 1, i + 1, i, i, i - 1, i - 1, i + 1, i + 1)
        neighbors_j <- c(j, j, j - 1, j + 1, j - 1, j + 1, j - 1, j + 1)
        #check to see if cell is empty, and within neighbourhood
        valid_neighbors <- which(forest[neighbors_i, neighbors_j] == 0 &
                                 neighbors_i >= 1 & neighbors_i <= forest_size &
                                 neighbors_j >= 1 & neighbors_j <= forest_size)

        if (length(valid_neighbors) > 0) {
          selected_neighbor <- sample(valid_neighbors, 1)
          new_i <- neighbors_i[selected_neighbor]
          new_j <- neighbors_j[selected_neighbor]
          survival_chance <- soil[new_i, new_j]
          if (!is.na(survival_chance) && runif(1) < survival_chance) {
            new_forest[new_i, new_j] <- 1 # Add new tree to the new_forest
          }
        }
      }
    }
  }
  return(new_forest)
}

#simulate over time
for (t in 1:100) {
  forest <- grow_trees(forest)
  forest <- spread_trees(forest, soil)

  normalized_forest <- forest / max(forest)

  image(normalized_forest, col = colors, axes = FALSE, main = paste("Forest Simulation t=",t))
  axis(1, at = seq(0, 1, length.out = forest_size), labels = 1:forest_size)
  axis(2, at = seq(0, 1, length.out = forest_size), labels = 1:forest_size)
}
```

Q3. Read through and experiment with the code. What are the key functions that define agent behaviour, and how do they work?

The key functions that define agent behavior are `grow_trees` and `spread_trees`.

Function `grow_trees` models how trees (agents) are growing, in this case, the `height` (values in the matrix) of trees are getting bigger each time with the factors of growth rate, sunlight, and water (`forest[i, j] <- forest[i, j] + growth_rate * sunlight * water`).

Function `spread_trees` models how trees taller than `1` can spread seeds to neighboring cells to help those void cells (`forest[i, j] = 0`) to grow new trees. First it detects its "valid" neighbors where their cell values are empty (`0`). Then a random neighbor was selected and given a random number between 0 and 1. Later, this number was compared to the smoothed soil quality value (also between 0 and 1), if it is smaller than the soil quality value, it survives to be a tree and given a height of `1`.

The two functions are coined together later in the simulation process where the growing and spreading process both occur one time in each iteration, and the colors of trees (in color palettes) are painted for each graph.

```r
# install.packages("simecol")
library(simecol)
```

```
## Loading required package: deSolve
```

```r
#predator-prey model
predator_prey <- new("odeModel",
                     main = function(time, init, parms) {
                       with(as.list(c(init, parms)), {
                         dPrey=r*Prey-a*Predators*Prey
                         dPred=-s*Predators+e*a*Predators*Prey
                         list(c(dPrey,dPred))
                       })
                     },
                     parms = c(r=0.5,a=0.02,s=0.5,e=0.1),
                     #r - prey birth rate
                     #a - predation rate
                     #s - death rate of predators
                     #e - efficiency of converting food (prey) into predator births
                     times = c(from=0,to=200,by=1),
                     init = c(Prey=40,Predators=9), #starting populations
                     solver = "lsoda"
)

predator_prey <- sim(predator_prey)
# plot(predator_prey)
```

Q4. Find a way to plot all the results on a single plot instead of on two separate plots. Write out your code.

```r
results <- out(predator_prey)
plot(results$time, results$Prey, type = "l", col = "blue", lwd = 2,
     xlab = "Time", ylab = "Population", main = "Predator-Prey Model",
     ylim = range(c(results$Prey, results$Predators)))
```

```r
lines(results$time, results$Predators, col = "red", lwd = 2)
legend("topright", legend = c("Prey", "Predators"), col = c("blue", "red"), lty = 1, lwd = 2)
```

```r
#innovation diffusion model
#A is the proportion of population that has adopted the innovation
#R: The rate at which individuals adopt innovation
#B: The rate at which individuals abandon innovation
#I: The rate at which adopters influence non-adopters to adopt innovation
inn_diff <- new("odeModel",
                            main = function(time, init, parms) {
                              with(as.list(c(init, parms)), {
                                dA=R*(1-A)-B*A+I*A*(1-A)
                                list(c(dA))
                              })
                            },
                            parms = c(R=0.1,B=0.02,I=0.05),
                            times = c(from=0,to=100,by=1),
                            init = c(A=0.01),
                            solver = "lsoda"
)

#simulate
inn_diff <- sim(inn_diff)
plot(inn_diff,main="Innovation diffusion")
```

Q5. Come up with a new parameter to add to the equation predicting the rate of diffusion. Don't worry about mathematical rigour; just modify the code to add a new term and ensure that the code runs without warnings/errors. Explain the change you've made, and how it alters the behaviour of the model.

```r
inn_diff <- new("odeModel",
                            main = function(time, init, parms) {
                              with(as.list(c(init, parms)), {
                                E <- runif(1, min = 0, max = 0.005)
                                dA=R*(1-A)-B*A+I*A*(1-A) + E * (1 - A)
                                list(c(dA))
                              })
                            },
                            parms = c(R=0.1,B=0.02,I=0.05),
                            times = c(from=0,to=100,by=1),
                            init = c(A=0.01),
                            solver = "lsoda"
)

inn_diff <- sim(inn_diff)

plot(inn_diff, main = "Innovation Diffusion with Random External Influence")
```

I added a new `E` variable to represent the external influences, which I regard as random effects each time between `0` and `0.005`. By adding this external influence, the model showed unpredictable variations which may result from real-world undetectable factors. However, this newly introduced variable did not influence the mainly increasing trends of innovation diffusion.

```r
#define the SIR model
SIR_model <- new("odeModel",
                 main = function(time,init,parms) {
                   with(as.list(c(init,parms)), {
                     dS = -b * S * I
                     dI = b * S * I - g * I
                     dR = g * I
                     list(c(dS, dI, dR))
                   })
                 },
                 parms = c(b = 0.3, g = 0.1),
                 times = c(from = 0, to = 100, by = 1),
                 init = c(S = 0.99, I = 0.01, R = 0),
                 solver = "lsoda"
)

SIR_model <- sim(SIR_model)
plot(SIR_model)
```

Q6. Modify the code to account for the fact that immunity to infection is not permanent.

```r
#define the SIR model
SIR_model <- new("odeModel",
                 main = function(time,init,parms) {
                   with(as.list(c(init,parms)), {
                     dS = -b * S * I + d * R
                     dI = b * S * I - g * I
                     dR = g * I - d * R
                     list(c(dS, dI, dR))
                   })
                 },
                 parms = c(b = 0.3, g = 0.1, d = 0.05),
                 times = c(from = 0, to = 100, by = 1),
                 init = c(S = 0.99, I = 0.01, R = 0),
                 solver = "lsoda"
)

SIR_model <- sim(SIR_model)
plot(SIR_model)
```