Assignment 1 - Forest Fire Simulation - Probability and Statistics 2023

Frova Davide

Notes

- Grid
- Each cell represents a portion of the forest:
  - Unburned
  - Burning
  - Burned
- If adjacent to a burning and unburned: Can ignite with prob `p`
- If unburned can ignite spontaneously with small prob `p_start`
- Bayes:
  - Likelihood of igniting by neighbor = P (Si(t + 1) = burning|Si_neighbors (t) = burning) = p
  - Lieklihood of igniting spontaneous = P (Si(t + 1) = burning) = p_start
- A burning cell transitions into a burned cell in the next time step.
- Burned and unburned cells retain their states unless altered by the above conditions.

Design the pseudocode for the forest fire model:

A. Design the initialize_grid function

Develop pseudocode for the initialize grid function that outputs a m × n grid, where each point represents a portion of the forest. Your map should include at least one water body (Water), a few patches of dry grass (DryGrass), areas with dense trees (DenseTrees), and a few initially burning areas (Burning). Label the rest of the areas as NormalForest. This grid will represent the initial state of the forest before the fire simulation begins.

```
# Install and load the rgl package
if (!require(rgl)) install.packages("rgl")
```

```
## Caricamento del pacchetto richiesto: rgl
```

```
library(rgl)
# Install and load the animation package
if (!require(animation)) install.packages("animation")
```

```
## Caricamento del pacchetto richiesto: animation
```

```
library(animation)
```

```
cell_states <- c(
  "Water", # 1
  "DryGrass", # 2
  "DenseTrees", # 3
  "Burning", # 4
  "NormalForest", # 5
```

```
  "Burned" # 6
)

cell_states_colors <- c(
  "#0000FF", # Blue / Water
  "#ADFF2F", # GreenYellow / DryGrass
  "#006400", # DarkGreen / DenseTrees
  "#FF0000", # Red / Burning
  "#008000", # Green / NormalForest
  "#201F1F" # DarkGray / Burned
)

# Initialize grif function
initialize_grid <- function(rows, cols) {
  # Create a list to store the grid
  grid <- list(
    "status" = matrix(nrow = rows, ncol = cols),
    "prob" = matrix(nrow = rows, ncol = cols)
  )

  for (y in 1:rows) {
    for (x in 1:cols) {
      # Sample from the indexes of the cell_states dictionary (without the burned state)
      # status <- sample(seq_along(head(cell_states, -1)), 1)
      status <- 5
      grid$status[y, x] <- status
      grid$prob[y, x] <- 0.0
    }
  }

  return(grid)
}

# Parametric function that enables the user to fill the forest with a specific function "shape"
# and a specific probability function, with a desired cell_type
fill_grid <- function(forest_grid, pos_func, prob_func, cell_type, thickness = 0.5, flip_orientation = F
  max_y <- dim(forest_grid$status)[1]
  max_x <- dim(forest_grid$status)[2]

  for (y in 1:max_y) {
    for (x in 1:max_x) {
      if (pos_func(y, x, max_y, thickness, flip_orientation)) {
        forest_grid$status[y, x] <- ifelse(
          cell_type == -1,
          forest_grid$status[y, x],
          cell_type
        )
        forest_grid$prob[y, x] <- prob_func(x, y)
      }
    }
  }

  return(forest_grid)
```

```
}

# Function ultra-parametric that enables the user to fill the forest with a specific function "shape"
# y, x are the coordinates of the point to be tested,
# min_y, max_y, max_x will be used to handle the ratio/span of the function in the grid
# func is the function used that will produce the value to compare with the thickness
# thickness is the sensitivity/distance from the y,x coordinate and the actual function value
# Returns true or false if the coordinate is or not generated by the function
func_fill <- function(y, x, min_y, max_y, max_x, func, thickness) {
  # Scale y to the range of the sine function.
  # Put the 0 of the sin in the middle of my matrix
  y_scaled <- ((y - min_y) / (max_y - min_y)) * 2 - 1

  # Scale x to the range of the sine function.
  x_scaled <- (x - max_x / 2) / (max_x / 2)

  # Calculate the y value of the sine wave at position x
  y_sin <- func(x_scaled)

  # Check if the scaled y value is close to the y value of the sine wave
  return(abs(y_scaled - y_sin) < thickness)
}

rand_pos <- function(y, x, max_y, thickness, flip_orientation = FALSE) {
  if (flip_orientation) {
    # Randomly change the function orientation
    temp <- y
    y <- x
    x <- temp
  }
  return(
    func_fill(
      y = y,
      x = x,
      min_y = 0,
      max_y = max_y,
      max_x = pi,
      func = function(val) runif(1, -max_y, max_y),
      thickness = thickness # 0.2
    )
  )
}

rand_max_x <- pi * sample(10:50, 1)
rand_min_y <- sample(2:20, 1)

exp_pos <- function(y, x, max_y, thickness, flip_orientation = FALSE) {
  if (flip_orientation) {
    # Randomly change the function orientation
    temp <- y
    y <- x
    x <- temp
  }
```

```
  return(
    func_fill(
      y = y,
      x = x,
      min_y = max_y * rand_min_y,
      max_y = max_y,
      max_x = rand_max_x,
      func = exp,
      thickness = thickness # 0.3
    )
  )
}

# Wrapper function for the func_fill
# It will produce a sin_wave function (wavy river like)
sin_pos <- function(y, x, max_y, thickness, flip_orientation = FALSE) {
  if (flip_orientation) {
    # Randomly change the function orientation
    temp <- y
    y <- x
    x <- temp
  }
  return(
    func_fill(
      y = y,
      x = x,
      min_y = 0,
      max_y = max_y,
      max_x = rand_max_x,
      func = sin,
      thickness = thickness # 0.3
    )
  )
}
```

B. Design the plot_grid function

Provide pseudocode to visualize the current state of the grid using different colors for each type of area (e.g., blue for water, green for NormalForest, red for Burning areas, etc.).

```
plot_grid <- function(forest_grid, plot_title = NULL) {
  forest_grid <- forest_grid$status

  # Increase the size of the color matrix
  enlarged_forest_grid <- kronecker(forest_grid, matrix(1, nrow = 10, ncol = 10))

  # Get the number of rows and columns of the enlarged matrix
  nrow_enlarged <- dim(enlarged_forest_grid)[1]
  ncol_enlarged <- dim(enlarged_forest_grid)[2]

  # Create an empty plot
  plot(
    0, 0,
    type = "n",
```

```
    xlim = c(0, ncol_enlarged),
    ylim = c(0, nrow_enlarged),
    xlab = "", ylab = "", xaxt = "n", yaxt = "n"
  )

  # Add the title to the plot
  if (!is.null(plot_title)) {
    title(main = plot_title)
  }

  # Create a color matrix
  color_matrix <- matrix(
    cell_states_colors[enlarged_forest_grid],
    nrow = nrow_enlarged,
    ncol = ncol_enlarged,
    byrow = TRUE
  )

  # Convert the color matrix to an image
  img <- as.raster(color_matrix)

  # Draw the image
  rasterImage(img, 0, 0, ncol_enlarged, nrow_enlarged)
}

# plot_grid(initial_state)
```

```
plot_3D <- function(forest_grid) {
  flipped_grid <- apply(forest_grid$status, 2, rev)
  transposed_status <- t(flipped_grid)

  flipped_probs <- apply(forest_grid$prob, 2, rev)
  transposed_probs <- t(flipped_probs)

  # Create a 3D plot
  x <- 1:ncol(transposed_status)
  y <- 1:nrow(transposed_status)
  z <- transposed_probs
  colors <- cell_states_colors[transposed_status]
  persp3d(x, y, z, col = colors, xlab = "X", ylab = "Y", zlab = "Prob")
}

# plot_3D(initial_state)
# plot_3D(next_state)
```

C. Develop the neighbours function

Create pseudocode to identify neighboring cells of a given point.

```
# Example output:
# [[1]]
#    state coords.x coords.y
#        3        2        1
```

```
# [[2]]
#    state coords.x coords.y
#        4        1        2

# [[3]]
#    state coords.x coords.y
#        1        2        2

# forest_grid object
neighbours <- function(forest_grid, cell) {
  x <- cell[2]
  y <- cell[1]

  max_x <- dim(forest_grid$status)[2]
  max_y <- dim(forest_grid$status)[1]

  result <- list(
    c(y - 1, x - 1), # Top left
    c(y, x - 1), # Top center
    c(y + 1, x - 1), # Top right
    c(y - 1, x), # Center left
    c(y + 1, x), # Center right
    c(y - 1, x + 1), # Bottom left
    c(y, x + 1), # Bottom center
    c(y + 1, x + 1) # Bottom right
  )

  valid_result <- list()

  for (negh in result) {
    if (negh[1] > 0 && negh[2] > 0 && negh[2] <= max_x && negh[1] <= max_y) {
      negh_obj <- c(
        "state" = forest_grid$status[negh[1], negh[2]],
        "coords" = c(
          "x" = negh[2],
          "y" = negh[1]
        )
      )

      valid_result <- append(valid_result, list(negh_obj))
    }
  }

  return(valid_result)
}
```

D. Construct the propagate function

Draft pseudocode to manage fire propagation based on: - The type of area - Its current state - Neighboring cell states.

```
# Example output:
# [1] 0.3 0.0 0.0
```

```
# "Water",          # 1
# "DryGrass",       # 2
# "DenseTrees",     # 3
# "Burning",        # 4
# "NormalForest"    # 5

# Cell:
# cell <- c(
#   "state" = 1--5,
#   "coords" = c(
#     "x" = 1,
#     "y" = 1
#   )
# )
propagate <- function(cell, neighbours) {
  if (cell[["state"]] != 4) {
    return(sapply(neighbours, function(val) 0.0))
  }

  neghs_probs <- sapply(neighbours, function(neg) {
    return(
      switch(neg[["state"]],
        "1" = 0.0, # max = 0.0
        "2" = 0.125, # max = 1.0
        "3" = 0.05, # max = 0.4
        "4" = 0.0, # max = 0.0
        "5" = 0.0875, # max = 0.7
        "6" = 0.0 # max = 0.0
      )
    )
  })

  return(neghs_probs)
}
```

E. Design the update_grid function:

Develop pseudocode to manage fire dynamics and grid updates using the neighbours and propagate functions.

```
# "Water",          # 1
# "DryGrass",       # 2
# "DenseTrees",     # 3
# "Burning",        # 4
# "NormalForest"    # 5
# "Burned"          # 6

update_grid <- function(forest_grid) {
  for (y in 1:nrow(forest_grid$status)) {
    for (x in 1:ncol(forest_grid$status)) {
      # Get the current cell state
      cell_state <- forest_grid$status[y, x]
      cell_prob <- forest_grid$prob[y, x]
```

```
    # If the cell could burn, generate a random number and check with its probability
    # if so, set them to burning and reset their prob
    if (cell_state == 2 || cell_state == 3 || cell_state == 5) {
      rand_value <- runif(1)
      if (rand_value <= cell_prob) {
        forest_grid$status[y, x] <- 4
        forest_grid$prob[y, x] <- 0.0

        cell_state <- 4
        cell_prob <- 0.0
      }
    } else if (cell_state == 4) {
      # If the cell is burning, and the random value gets the prob
      # set it to burned and reset its prob, otherwise increase the prob of going out

      rand_value <- runif(1)
      if (rand_value <= cell_prob) {
        forest_grid$status[y, x] <- 6
        forest_grid$prob[y, x] <- 0.0

        cell_state <- 6
        cell_prob <- 0.0
      } else {
        # Increase the probability of going out
        forest_grid$prob[y, x] <- forest_grid$prob[y, x] + 0.3
        cell_prob <- cell_prob + 0.1
      }
    }

    # Get the neighbours of the current cell
    neghs <- neighbours(forest_grid = forest_grid["status"], cell = c(y, x))

    # Create an "object" with the data needed from the propagate function
    propagate_cell <- c(
      "state" = cell_state,
      "coords" = c(
        "x" = x,
        "y" = y
      )
    )

    # Compute the probabilities of the propagate of the neighbours cells
    probs_propagate <- propagate(propagate_cell, neghs)

    # Foraech of the neighbours for which we have the probs. of ignition
    # we sum it to the current prob. values of the forest_grid
    # (Markov Chain) t_n = t_n-1 + added_prob
    for (negIndex in 1:length(neghs)) {
      forest_grid$prob[neghs[[negIndex]][["coords.y"]], neghs[[negIndex]][["coords.x"]]] <-
        forest_grid$prob[neghs[[negIndex]][["coords.y"]], neghs[[negIndex]][["coords.x"]]] + probs_pr
    }
  }
}
```

```
  return(forest_grid)
}
```

```
plot_probs_and_states <- function(forest_grid, probs = TRUE) {
  flipped_grid <- apply(forest_grid$status, 2, rev)

  flipped_probs <- apply(forest_grid$prob, 2, rev)

  # Create a blank plot
  plot(1, 1,
    xlim = c(1, ncol(flipped_grid)), ylim = c(1, nrow(flipped_grid)),
    type = "n", xlab = "", ylab = "", xaxt = "n", yaxt = "n"
  )

  # Draw rectangles for each cell
  for (i in 1:nrow(flipped_grid)) {
    for (j in 1:ncol(flipped_grid)) {
      rect(j - 0.5, i - 0.5, j + 0.5, i + 0.5, col = cell_states_colors[flipped_grid[i, j]])
    }
  }

  # Add probabilities on top of the rectangles
  for (i in 1:nrow(flipped_probs)) {
    for (j in 1:ncol(flipped_probs)) {
      text(j, i, labels = ifelse(probs, flipped_probs[i, j], flipped_grid[i, j]), cex = 1.5)
    }
  }
}
```

F. Draft the simulate function

Provide pseudocode to orchestrate the entire simu- lation over multiple iterations using the previously developed functions.

```
rand_max_x <- pi * sample(10:50, 1)
rand_min_y <- sample(2:20, 1)

simulate <- function(t_max, N, grid_rows, grid_cols) {
  # N simulations, until t_max iterations

  for (i in 1:N) {
    paste("Simulation", i)

    # Initialize the grid full of normal forest
    forest_grid <- initialize_grid(grid_rows, grid_cols)

    # CUSTOMIZABLE PART -------------------------

    # Add random dry grass with some small self ignition probability
    forest_grid <- fill_grid(
      forest_grid = forest_grid,
      pos_func = rand_pos,
      prob_func = function(y, x) 0.0002, # Starting Probability (self-ignition prob)
      cell_type = 2, # Dry grass
```

```
  thickness = runif(1, 0, 20) # Higher value = Higher population
)

# Add random dense trees with some small self ignition probability
forest_grid <- fill_grid(
  forest_grid = forest_grid,
  pos_func = rand_pos,
  prob_func = function(y, x) 0.00002, # Starting Probability (self-ignition prob)
  cell_type = 3, # Dense trees
  thickness = runif(1, 0, 20) # Higher value = Higher population
)

#  Randomize the values of the exp_pos function
rand_max_x <- pi * sample(10:50, 1)
rand_min_y <- sample(2:20, 1)

# Adding a river with a random function plotted (exp or sin)
# I kept them separated and not param only the func_pos to be able to modify them separately
rand_value_river <- runif(1)
water_type <- 0
if (rand_value_river < 0.33) {
  # Exp function
  forest_grid <- fill_grid(forest_grid, exp_pos, function(y, x) 0.0, 1, runif(1, 0.05, 0.3), ifelse
  water_type <- 0
} else if (rand_value_river >= 0.33 && rand_value_river < 0.66) {
  # Random sin river
  forest_grid <- fill_grid(forest_grid, sin_pos, function(y, x) 0.0, 1, runif(1, 0.05, 0.3), ifelse
  water_type <- 1
} else {
  # Random rain puddles
  forest_grid <- fill_grid(forest_grid, rand_pos, function(y, x) 0.0, 1, runif(1, 0.05, 10), ifelse
  water_type <- 2
}

# Random starting point burning
forest_grid$status[sample(1:grid_rows, 1), sample(1:grid_cols, 1)] <- 4

# Define the animation function

for (t in 1:t_max) {
  # Print the current iteration
  # paste("Iteration", t)
  # Update the grid
  forest_grid <- update_grid(forest_grid)

  # Plot the grid
  plot_grid(
    forest_grid,
    paste(
      "Simulation n. ", i, " / ", N, "\nStep n. ", t, " / ", t_max,
      "\n Water type: ", ifelse(water_type == 0, "Exp River", ifelse(water_type == 1, "Sin River",
    )
  )
```

```
      print(paste("T: ", t, " / ", t_max))
    }
    print(paste(i, "| Simulation Completed"))

    # Save the animation as a GIF
    # saveGIF(ani.fun(), movie.name = paste("forest_simulation.gif"), interval = 0.2)
  }
}

# simulate(
#   t_max = 20,
#   N = 2,
#   grid_rows = 100,
#   grid_cols = 100
# )

# Gif
# saveGIF(
#   simulate(
#     t_max = 50,
#     N = 5,
#     grid_rows = 100,
#     grid_cols = 100
#   ),
#   movie.name = paste("forest_simulation.gif"),
#   interval = 0.2
# )

# Movie
# saveVideo(
#   simulate(
#     t_max = 10,
#     N = 5,
#     grid_rows = 50,
#     grid_cols = 50
#   ),
#   video.name = paste("forest_simulation.mp4"),
#   interval = 0.2
# )
```