10.3 Spreading of Fire

R Quick Review Questions

Introduction to Computational Science:
Modeling and Simulation for the Sciences, 2nd Edition
Angela B. Shiflet and George W. Shiflet
Wofford College

This file contains system-dependent Quick Review Questions and answers in *R* for Module 10.3 on "Spreading of Fire." Complete all code development in *R*.

Initializing the System

Quick Review Question 1 Suppose the *fire* function initializes global variables *EMPTY*, *TREE*, and *BURNING*.

a. A site has a tree with a probability of *probTree*. Assign to *treesOrBurns* an *n*-by-*n* array of zeros and ones, where 1 occurs in an element with a probability of *probTree*, that is, a uniformly distributed random number between 0 and 1 is less than *probTree* for that element.

```
treesOrBurns = __________(__________ ______ probTree,__________=n)
```

b. A tree, which has a value of 1 in *treesOrBurns*, is burning tree with a probability of *probBurning*. Assign to *burns* an *n*-by-*n* array of zeros and ones, where 1 occurs in an element with a probability of *probTree*. Thus, if an element is 1 in *treesOrBurns* and a uniformly distributed random number between 0 and 1 is less than *probBurning*, then the corresponding element in *burns* is 1. We obtain *burns* by taking the array product of *treesOrBurns* and an appropriate array of zeros and ones.

```
burns = treesOrBurns ______ ______(_______ ______ probBurning,______=n)
```

c. Assign to *trees* an *n*-by-*n* array of zeros and ones, where 1 occurs where the cell has a non-burning tree. Thus, if an element is 1 in *treesOrBurns* and 0 in *burns*, the corresponding element is 1 in *trees*. If 1 occurs in corresponding elements of *treesOrBurns* and *burns*, that element is 0 in *trees*.

```
trees = treesOrBurns ______ burns
```

d. Assign to *empties* an *n*-by-*n* array of zeros and ones, with 1 indicating an empty site. Thus, if 0 occurs in *treesOrBurns*, the corresponding element in *empties* is 1. If 1 is at a site in *treesOrBurns*, that element is 0 in *empties*.

```
empties = ____________ ______ treesOrBurns
```

e. Which of the following is (are) true about corresponding elements of *empties*, *trees*, and *burns*.

A. All three have values of 1.
B. Exactly two have values of 1, while the other is 0.
C. Exactly two have values of 0, while the other is 1.
D. All three have values of 0.

E. It is impossible to know.

f. Assign to *forest* an *n*-by-*n* array where an element is *EMPTY* if the corresponding element in *empties* is 1, is *TREE* if the corresponding element in *trees* is 1, and is *BURNING* if the corresponding element in *burns* is 1

```
forest =  empties ____ EMPTY + trees ____ TREE + burns ____ BURNING
```

Updating Rules

Quick Review Question 2 The following questions develop the rule for *spread*(*site*, *N*, *E*, *S*, *W*, *probLightning*, *probImmune*) that applies to the situation where a site does not contain a tree at this or any time step. Suppose the function M-file *spread.m* begins as follows:

```
spread = function(site, N, E, S, W, probLightning, probImmune) {
# SPREAD - Function to return the value of a site
# at the next time step
# An empty cell remains empty.
# A burning cell becomes empty.
# If a neighbor to the north, east, south, or west of
# a tree is burning, then the tree does not burn with a
# probability of probImmune.
# If a tree has no burning neighbors, it is hit by lightning
# and burns with a probability of probLightning * (1 - probImmune).

   utils::globalVariables(c("EMPTY", "TREE", "BURNING"))
```

a. Select the value of *site: EMPTY*, *TREE*, *BURNING*, none of these

b. Select the return value: *EMPTY*, *TREE*, *BURNING*, none of these

c. Complete the implementation for this rule.

```
if (___________________)
    newSite = ________________
```

Quick Review Question 3 The following questions develop the rule for *spread* that applies to the situation where a site contains a burning tree:

a. Select the value of *site: EMPTY*, *TREE*, *BURNING*, none of these

b. A burning tree always burns down. Give the return value of the *spread* function for this situation.

c. Complete implementation of this rule, which occurs in an *elseif* segment.

```
else if (_________________)
    newSite = ________________
```

Quick Review Question 4 The following questions develop the rule for *spread* that applies to the situation where a site contains a non-burning tree that may catch fire because a neighboring site contains a burning tree:

a. Select the value of *site: EMPTY*, *TREE*, *BURNING*, none of these

b. Select the meaning of the following call to *If:*

```
if (runif(1) < probImmune)
    newSite = TREE
else
    newSite = BURNING
end
```

A. If a random number is less than the probability of immunity, then the tree catches fire; else it does not.
B. If a random number is less than the probability of immunity, then the tree does not catch fire; else it does.
C. If a random number is less than the probability of immunity, then the tree stays immune; else it does not.
D. If a random number is less than the probability of immunity, then the tree does not stay immune; else it does.

c. For the tree to have a chance of burning due to fire at a neighboring site, give the value that at least one of *N*, *E*, *S*, *W* must have.
d. Give the start of the *if* statement to test if one of the parameters *N*, *E*, *S*, *W* is *BURNING*.
e. Give an implementation of this rule.

Quick Review Question 5 Complete implementation of the rule for *spread* that applies to the situation where a site contains a non-burning tree that may be hit by lightning and burn. This code provides an alternative to the situation in the previous Quick Review Question, where a tree has a burning neighbor.

```
else if (runif(1) < probLightning * (1 - probImmune))
    newSite =
else
    newSite =
end
```

Periodic Boundary Conditions

Quick Review Question 6 This question extends an array as in Figure 10.3.1 by attaching the last row to the beginning and the first row to the end of the original array.
a. Write a command to return the last row of array *lat*.
b. Write a command to return the first row of array *lat*.
c. Complete the following statement to make *latNS* an extended array of *mat* as described in this question.

```
extendRows = _____lat[nrow(lat),]_____ lat_____ lat[1,])
```

Quick Review Question 7 This question extends an array as in Figure 10.3.2.
a. Write a command to return the last column of array *extendRows*.
b. Write a command to return the first column of array *extendRows*.
c. Complete the following statement to make *extlat* an extended array of *lat* as described in this question.

```
extlat = [extendRows(,ncol(extendRows))_____ extendRows_____ extendRows(, 1)]
```

d. If the original array *mat* is of size 7-by-7, after extending the matrix as in this and the previous Quick Review Question, give the size of the extended matrix.

Applying a Function to Each Grid Point

Quick Review Question 8 This question develops the function *applyExtended*.
a. Complete the code to start the definition of *applyExtended*, which is to have an extended array parameter (*latExt*).

```
applyExtended = _______(latExt , probLightning, probImmune)
```

b. Write the statement to assign to *n* the number of rows (or columns) in the internal, un-extended square array.

c. Suppose *i* represents the row index and *j* the column index. To apply the function *spread* to each internal cell of *latExt*, we use nested *for* loops and let indices *i* and *j* vary between two values. Give the initial value of *i* (or *j*).

d. Give the final value for *i* (or *j*).

e. Within the body of inner *for* loop, we assign values to *site*, *N*, *E*, *S*, and *W*. Then, we apply the function *spread* with parameters *site*, *N*, *E*, *S*, *W*, *probLightning*, and *probImmune* to each internal cell site. Figure 10.2.11 gives the coordinates of a site and its neighbors. Give the code to assign to *site* the value of the (*i*, *j*)-element of two-dimensional array *latExt*.

f. Give the code to assign to *N* the value from *latExt* corresponding to the neighbor to the north.

g. Give the code to assign to *E* the value from *latExt* corresponding to the neighbor to the east.

h. Complete the assignment to the appropriate *newmat* element of the evaluation of the function *spread* with parameters *site*, *N*, *E*, *S*, and *W*. Because *newmat* has the size of the original un-extended array, the *newmat* element is on a row and column, where each index (*i* or *j*) is one less than the corresponding index of *site* in *latExt*.

```
newmat[______, ______] = ______(site, N, E, S, W , probLightning, probImmune)
```

i. Give the complete definition of *applyExtended*.

Simulation Program

Quick Review Question 9 Complete the implementation of the *fire* function, assuming *grids* is a three-dimensional array containing in the first page the initial forest and the forest at all other time steps.

```
fire = function(n, probTree, probBurning, probLightning, probImmune, t) {
# FIRE simulation
   utils::globalVariables(c("EMPTY", "TREE", "BURNING"))
   EMPTY = 0
   TREE = 1
   BURNING = 2
   grids = array(____=c(n,n,t+1))
   grids[_____, ______, ______] = forest
   for (i in 2:(t + 1)) {
     _________
     _________
     _________
     }
   return(grids)
}
```

Display Simulation

Quick Review Question 10 This question develops the function *showGraphs* that of graphics corresponding to the grids in a three-dimensional array (*graphList*), where each page holds a grid for one time step of the simulation.

a. The function calls another function, *pointsForGrid*, which has parameters of a two-dimensional matrix, *grid*, and a value, *val*, such as *TREE* or *BURNING*. The function *pointsForGrid* returns a list of two vectors, *xcoords* and *ycoords*, which contain the *x*-coordinates and *y*-coordinates where the *grid* values are *val*. Write a statement to assign to *xcoords* an empty vector.

b. In the following nested loop, complete the *if* statement possibly to place new values in *xcoords* and *ycoords*. To match the matrix values in the eventual visualization, we reverse the rows.

```
for (row in 1:________(grid)) {
        for (col in 1:___________ (grid)) {
            if (__________== val) {
                xcoords[__________(xcoords)+1] = col
                ycoords[__________(ycoords)+1] = nrow(grid) - row
            }
        }
    }
```

as below so that a visualization pictures a site value of *EMPTY* (0) as yellow, *TREE* (1) as forest green, and *BURNING* (2) as burnt orange. The values on a row represent the amounts of red, green, and blue for the corresponding color. Give the command to make *map* the color map for the graphics.

```
map = [1 1 0;          % EMPTY   -> yellow
    0.1 0.75 0.2;      % TREE    -> forest green
    0.6 0.2 0.1];      % BURNING -> burnt orange
```

c. Returning to the development of *showGraphs*, give the statement to assign to *m* the number of grids (pages) in *graphList*.

d. Give the statement to assign to local variable *g* the *k*-th page in three-dimensional array *graphList*.

e. Complete the commands to produce a graphic of *g* as a rectangular grid were trees are green and burning trees are red. Sleep 0.2 s between frames.

```
trees = pointsForGrid(g,TREE)
burnings = pointsForGrid(g,BURNING)
plot(trees[[_______]],trees[[_____]],pch=19,col=_____,
           xlim=c(0,n+1),ylim=c(0,n+1))
points(burnings[[______]],burnings[[______]],col=________, pch=23,
           bg="orange")
Sys.sleep(0.2)
```

Answers to Quick Review Question

1.
a. `treesOrBurns = matrix(runif(n^2) < probTree,nrow=n)`
b. `burns = treesOrBurns * matrix(runif(n^2) < probBurning,nrow=n)`
c. `trees = treesOrBurns - burns`
d. `empties = 1 - treesOrBurns`
e. C. Exactly two have values of 0, while the other is 1.
f. `forest =  empties * EMPTY + trees * TREE + burns * BURNING`

2. **a.** *EMPTY*
b. *EMPTY*
c.
```
if (site == EMPTY)
     newSite = EMPTY
```

3. **a.** *BURNING*
b. *EMPTY*, which indicates an empty cell
c.
```
else if (site == BURNING)
     newSite = EMPTY
```

4. **a.** *TREE*
b. B. If a random number is less than the probability of immunity, then the tree does not catch fire; else it does.
c. *BURNING*
d. `if (N==BURNING || E ==BURNING || S == BURNING || W == BURNING)`
e.
```
if (N==BURNING || E==BURNING || S==BURNING || W==BURNING) {
    if (runif(1) < probImmune)
       newSite = TREE
    else
       newSite = BURNING
    }
```

5.
```
else if (runif(1) < probLightning * (1 - probImmune))
     newSite = BURNING
else
     newSite = TREE
end
```

The following segment contains all the updating rules for the function *spread:*

```
if (site == EMPTY){
    newSite = EMPTY
    }
else if (site == BURNING){
     newSite = EMPTY
     }
else if (site == TREE) {
     if (N == BURNING || E == BURNING || S == BURNING || W == BURNING) {
         if (runif(1) < probImmune)
              newSite = TREE
         else
              newSite = BURNING
         }
     else if (runif(1) < probLightning * (1 - probImmune))
         newSite = BURNING
     else
         newSite = TREE
     }
```

6. **a.** `lat[nrow(lat), ]`
b. `lat[1, ]`
c. `extendRows = rbind(lat[nrow(lat), ], lat, lat[1, ])`

7. **a.** `extendRows[,ncol(extendRows)]`
b. `extendRows[, 1]`
c.
```
extlat = cbind(extendRows[,ncol(extendRows)],extendRows,extendRows[,1])
```
d. 9-by-9

8. a.

```
applyExtended = function(latExtended, probLightning, probImmune){
```

b.
```
n = nrow(latExtended) - 2
```

c. 2

d. $n + 1$

e
```
site = latExt[i, j]
```

f.
```
N = latExt[i - 1, j]
```

g.
```
E = latExt[i, j + 1]
```

h.
```
newmat[i - 1, j – 1] = spread(site, N, E, S, W ,
     probLightning, probImmune)
```

i.

```
applyExtended = function(latExt, probLightning, probImmune) {
# APPLYEXTENDED - Function to apply
# spread(site, N, E, S, W, probLightning, probImmune) to every interior
# site of square array latExt and to return the resulting array
    n = nrow(latExt) - 2
    newmat = matrix(c(rep(0,n*n)), nrow = n)
    for (j in 2:(n + 1)) {
        for (i in 2:(n + 1)) {
            site = latExt[i, j]
            N = latExt[i - 1, j]
            E = latExt[i, j + 1]
            S = latExt[i + 1, j]
            W = latExt[i, j - 1]
            newmat[i - 1, j - 1] = spread(site, N, E, S, W,
               probLightning, probImmune)
        }
    }
    return(newmat)
}
```

9.

```
fire = function(n, probTree, probBurning, probLightning, probImmune, t)
{
# FIRE simulation
    utils::globalVariables(c("EMPTY", "TREE", "BURNING"))
    EMPTY = 0
    TREE = 1
    BURNING = 2

    forest  = initForest( n, probTree, probBurning )
    grids = array(dim=c(n,n,t+1))

    grids[,,1] = forest

    for (i in 2:(t+1)) {
        forestExtended = periodicLat(forest)
        forest = applyExtended(forestExtended, probLightning,
               probImmune)
        grids[,,i] = forest
    }

    return(grids)
}
```

10. a.
```
xcoords = vector()
```

b.

```
    for (row in 1:nrow(grid)) {
        for (col in 1:ncol(grid)) {
            if (grid[row,col] == val) {
                xcoords[length(xcoords)+1] = col
                ycoords[length(ycoords)+1] = nrow(grid) - row
```

```
            }
        }
    }
```

c.
```
m = dim(graphList)[3]
```

d.
```
g = graphList[,, k]
```

e.
```
trees = pointsForGrid(g,TREE)
burnings = pointsForGrid(g,BURNING)
plot(trees[[1]],trees[[2]],pch=19,col="green",
    xlim=c(0,n+1),ylim=c(0,n+1))
points(burnings[[1]],burnings[[2]],col="red",pch=23,bg="orange")
Sys.sleep(0.2)
```