10.2 Diffusion: Overcoming Differences

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 10.2 on "Diffusion: Overcoming Differences." Complete all code development in *R*.

Initializing the System

Quick Review Question 1 The function to initialize the bar begins as follows:

```
initBar <-function(m,n,hotSites,coldSites){
```

a. Give the statement to declare *AMBIENT* to be a global variable.
b. Assign to *bar* an $m \times n$ matrix of all *AMBIENT* values.
c. The function then calls *applyHotCold* to establish the hot and cold spots:

```
bar = applyHotCold(bar, hotSites, coldSites)
```

Complete the loop in *applyHotCold* to make each of the locations indicated by *hotSites* have the value *HOT*.

```
newbar = bar
for k in 1:__________{
    bar(hotSites(_________, _________), _________) = HOT;
}
```

Heat Diffusion

Quick Review Question 2 Suppose the diffusion rate parameter is 0.1 and the temperatures in the cells are as in Figure 10.2.4. Calculate the temperature in the center cell at the next time step.

Boundary Conditions

Quick Review Question 3 Answer the following questions about Figure 10.2.4 as an extremely small entire thermal grid.
a. Give the size of the grid extended to accommodate boundary conditions.
b. Give the values in the first row of the extended matrix assuming fixed boundary conditions with fixed value 0.
c. Give the values in the first row of the extended matrix assuming reflecting boundary conditions, where we copy rows first.
d. Give the values in the first row of the extended matrix assuming periodic boundary conditions, where we copy rows first.

Quick Review Question 4 This question extends an array as in Figure 10.2.8 by attaching a copy of the first row to the beginning and a copy of the last row to the end of the original grid to form a new grid, *latNS*.

a. Give a command to return the last row of matrix *lat*.
b. Give a command to return the first row of matrix *lat*.
c. Complete the following statement to make *latNS* an extended array of *lat* as described in this question.

```
latNS = _______(lat[_______],lat,lat[_____(_______),])
```

Quick Review Question 5 This question extends an array as in Figure 10.2.9.

a. Give a command to return the last column of matrix *latNS*.
b. Give a command to return the first column of matrix *latNS*.
c. Make *extLat* a matrix containing the concatenation of the last column of *latNS*, *latNS*, and the first column of *latNS*.
d. If the original matrix *lat* 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 6 Suppose *extMat* is an extended matrix of size 97-by-62.

a. Give the size of the matrix *applyDiffusionExtended* returns.
b. When $i = 33$ and $j = 25$, give the indices of the site's neighbor to the north.
c. For this site, give the indices of its neighbor to the southwest.

Quick Review Question 7 This question develops the function *applyDiffusionExtended*, which is to have a diffusion rate parameter (*diffusionRate*) and an extended array parameter (*latExt*) and local variables *m*, *n*, *site*, *N*, *NE*, *E*, *SE*, *S*, *SW*, *W*, *NW*, *i*, and *j* and is to begin as follows:

```
applyDiffusionExtended <- function(diffusionRate,latExt){
```

a. Write the statement to assign to *m* the number of rows of the internal (non-extended) matrix.
b. Write the statement to assign to *n* the number of columns of the internal (non-extended) matrix.
c. Write a statement to assign to *newmat* an $m \times n$ matrix of zeros.
d. We can use nested *for* statements to go through every column (index *j*) and every row (index *i*) defining the values of the return matrix, *newmat*. Within the body of loops, we assign values to *site*, *N*, *NE*, *E*, *SE*, *S*, *SW*, *W*, and *NW*, and apply *diffusion* with these parameters for each internal cell site. To apply the function *diffusion* to each internal cell of *latExt*, we let *i* (and *j*) vary between two values. Give the initial value of *i* (or *j*).
e. Give the final value of *i*.
f. Give the final value of *j*.
g. Give the first line of the *for* loop with index *j* that goes through each internal column.
h. Give the first line of the *for* loop with index *i* that goes through each internal row.
i. 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 matrix *latExt*.

j. Give the code to assign to *N* the value from *latExt* corresponding to the neighbor to the north.
k. Give the code to assign to *E* the value from *latExt* corresponding to the neighbor to the east.
l. Give the complete definition of *applyDiffusionExtended*.

Simulation Program

Quick Review Question 8 Implement the loop in the *diffusionSim* function, assuming *grids* is a three-dimensional array containing in the first page the initial bar, *bar*, which Quick Review Question 1 develops.

Display Simulation

Quick Review Question 9 This question develops the function *animDiffusionGray* that produces a grayscale animation corresponding to the grids in a three-dimensional array (*grids*), where each page holds a grid for one time step of the simulation.
a. Give the function call to return the number of elements (grids) in *grids*.
b. Complete the following statement to assign to *map* levels of gray for each integer temperature from 0 to *HOT* inclusive and to make *map* a color map.

```
map =       (       :       )
```

c. Give the statement to assign to local variable *g* the transposed *k*-th element in *grids*.
d. Complete the command to produce a graphic of *g* as a rectangular grid with black representing *HOT* and white representing *COLD*.

```
image(_________, _____ = map, axes = FALSE);
```

d. Give the entire definition of *animDiffusionGray*.

Quick Review Question 10 This question refers to function *animDiffusionColor*, which produces a color animation corresponding to each simulation lattice in a list (*grids*). Complete the following segment to assign to *map* levels of gray for each integer temperature from 0 to *HOT* inclusive and to make *map* a color map.

```
map = 1:(HOT + 1)
for (i in 0:HOT) {
    amt <-
    map[i + 1] <-     (       ,       ,       )
}
                    (map);
```

Answers to Quick Review Question

1. a.
```
utils::globalVariables(c("AMBIENT"))
```
b.
```
bar <-AMBIENT*(matrix(c(rep(1,m*n)), nrow = m))
```
c.
```
newbar = bar
for(k in 1:length(coldSites[,1])) {
    newbar[coldSites[k,1],coldSites[k,2]]<-COLD
    }
```

2. $3.6 = (1 - 8 \cdot 0.1)(5) + 0.1(2 + 3 + 4 + 0 + 6 + 1 + 3 + 7)$

3. **a.** 5-by-5
b. 0, 0, 0, 0, 0
c. 2, 2, 3, 4, 4
d. 7, 1, 3, 7, 1

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

5. **a.** `latNS[,ncol(latNS)]`
b. `latNS[,1]`
c. `extLat = cbind(latNS[,1], latNS, latNS[,ncol(latNS)])`
d. 9-by-9

6. **a.** 95-by-60
b. (32, 25)
c. (34, 24)

7. **a.** `m = length(latExt[,1]) - 2`
b. `n = length(latExt[2,]) - 2`
c. `newLat = matrix(rep(0,m*n), nrow=m)`
d. `2`
e. *m* + 1
f. *n* + 1
g. `for(j in 2:(n+1)){`
h. `for(i in 2:(m+1)){`
i. `site = latExt[i, j]`
j. `N = latExt[i - 1, j]`
k. `E = latExt[i, j + 1]`
l.

```
applyDiffusionExtended <- function(diffusionRate,latExt){
# APPLYEXTENDED - Function to apply
# diffusion(diffusionRate, site, N, NE, E, SE, S, SW, W, NW)
# site of matrix latExt and to return the resulting matrix
    m = length(latExt[,1]) - 2
    n = length(latExt[2,]) - 2
    newLat = matrix(rep(0,m*n), nrow=m)
    # calculate one column at a time because R is column-oriented
    for(j in 2:(n+1)){
        for(i in 2:(m+1)){
             site = latExt[i, j]
             N = latExt[i - 1, j]
             NE = latExt[i -1, j + 1]
             E = latExt[i, j + 1]
             SE = latExt[i + 1, j + 1]
             S  = latExt[i + 1, j]
             SW = latExt[i + 1, j - 1]
             W = latExt[i, j - 1]
             NW = latExt[i - 1, j - 1]

             newLat[i - 1, j - 1] = diffusion(diffusionRate,
                    site,N, NE, E, SE, S, SW, W, NW)
        }
    }
return(newLat)
}
```

8.

```
diffusionSim = function(m,n,diffusionRate,hostSites,coldSites,t) {
#Declare global variables hot,cold and ambient.
# Initialize grid

bar = initBar(m,n,hotSites,coldSites)
bar
# Perform simulation

grids<-array(0, dim=c(m,n,t+1))
grids[,,1]<-bar

for(i in 2:(t+1)) {

#     Extend matrix
    barExtended = reflectingLat(bar)
#     Apply spread of heat function to each grid point
    bar = applyDiffusionExtended(diffusionRate,barExtended)
#     reapply hot and cold spots
    bar = applyHotCold(bar,hotSites,coldSites)
#    save new Matrix
    grids[,,i]<-bar

    }
return(grids)
}
```

9. **a.**
```
dim(graphList)[3]
```

b.
```
map = gray(0:HOT / HOT)
```

c.
```
g = t(graphList[,,k])
```

d.
```
image(HOT - g + 1, col = map, axes = FALSE)
```

d.
```
M(k) = getframe;
```

e.

```
# animate grids in gray with HOT being black, COLD more white
animDiffusionGray<- function(graphList){
    utils::globalVariables(c("HOT"))
    lengthGraphList = dim(graphList)[3]

    # set up grayscale map
    map = gray(0:HOT / HOT)

    m = dim(graphList)[1]
    n = dim(graphList)[2]

    # determine window size
    # 1.6 is approximately the golden ratio; so cell pictured as square
    fraction = n/m;
    dev.new(width = 2 * fraction, height = 2 * 1.6)

    for( k in 1:lengthGraphList) {
        dev.hold()
        g = t(graphList[,,k])  # transpose because of following comment

#"image interprets the z matrix as a table of f(x[i], y[j]) values, so
that the x axis corresponds to row number and the y axis to column
number, with column 1 at the bottom, i.e. a 90 degree counter-clockwise
rotation of the conventional printed layout of a matrix."

        # first row's image is on bottom, unlike figures in text
        image(HOT - g + 1, col = map, axes = FALSE)
        box()
```

```
            Sys.sleep(0.1)
            dev.flush()
        }
    }
```

10.

```
map <- 1:(HOT + 1)
for (i in 0:HOT) {
    amt <- i/HOT
    map[i + 1] <- rgb(amt, 0, 1-amt)  # red-green-blue amounts
 }
```