

11.2 Spreading of Fire

MATLAB Quick Review Questions

Introduction to Computational Science: Modeling and Simulation for the Sciences
Angela B. Shiflet and George W. Shiflet
Wofford College
© 2006 by Princeton University Press

This file contains system-dependent Quick Review Questions and answers in *MATLAB* for Module 11.2 on "Spreading of Fire." Complete all code development in *MATLAB*.

Initializing the System

Quick Review Question 1 Suppose the *fire* function in M-file *fire.m* begins as follows:

```
function grids = fire(n, probTree, probBurning, chanceLightning,...  
                    chanceImmune, t);
```

Part a assigns values to global variables, and the remainder of this question initializes *forest* to be an n -by- n array and *grids* to be a three-dimensional array containing *forest* in the first page. Quick Review Question 9 completes the implementation of the function *fire*.

- a. Each site is assigned a value as described in the algorithm for Cell Initialization. Complete the code to make global the variables on the first line and to assign values to them.

```
_____ EMPTY TREE BURNING probLightning probImmune  
EMPTY = 0; TREE = 1; BURNING = 2;  
probLightning = chanceLightning;  
probImmune = chanceImmune;
```

- b. 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);
```

- c. 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);
```

- d. 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;
```

- e. 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;
```

- f. 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.

- g. 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;
```

- h. Give command to establish *grids* as a three-dimensional array of n rows, n columns, and $t + 1$ pages of zeros. At the end of the simulation, each page holds one n -by- n grid.

```
grids = _____;
```

- i. Give command to place *forest* on the first page of *grids*.

```
grids_____ = forest;
```

Updating Rules

Quick Review Question 2 The following questions develop the rule for *spread*(*site*, N , E , S , W) 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:

```
function newSite = spread(site, N, E, S, W);
% 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).
%
global EMPTY TREE BURNING probLightning probImmune
```

- Select the value of *site*: *EMPTY*, *TREE*, *BURNING*, none of these
- Select the return value: *EMPTY*, *TREE*, *BURNING*, none of these
- 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:

- Select the value of *site*: *EMPTY*, *TREE*, *BURNING*, none of these
- A burning tree always burns down. Give the return value of the *spread* function for this situation.
- Complete implementation of this rule, which occurs in an *elseif* segment.

```
elseif _____
    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:

- Select the value of *site*: *EMPTY*, *TREE*, *BURNING*, none of these
- Select the meaning of the following call to *If*:

```
if (rand < probImmune)
    newSite = TREE;
else
    newSite = BURNING;
end;
```

- If a random number is less than the probability of immunity, then the tree catches fire; else it does not.
 - If a random number is less than the probability of immunity, then the tree does not catch fire; else it does.
 - If a random number is less than the probability of immunity, then the tree stays immune; else it does not.
 - If a random number is less than the probability of immunity, then the tree does not stay immune; else it does.
- 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.
 - Give the start of the *if* statement to test if one of the parameters *N*, *E*, *S*, *W* is *BURNING*.
 - 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.

```
elseif (rand < probLightning * (1 - probImmune))
    newSite = _____;
else
    newSite = _____;
end;
```

Periodic Boundary Conditions

Quick Review Question 6 This question extends an array as in Figure 11.2.6 by attaching the last row to the beginning and the first row to the end of the original array.

- Write a command to return the last row of array *mat*.
- Write a command to return the first row of array *mat*.
- Complete the following statement to make *matNS* an extended array of *mat* as described in this question.

```
extendRows = _____mat(end, :)______ mat_____ mat(1, :)______;
```

Quick Review Question 7 This question extends an array as in Figure 11.2.7.

- Write a command to return the last column of array *extendRows*.
- Write a command to return the first column of array *extendRows*.
- Complete the following statement to make *extmat* an extended array of *mat* as described in this question.

```
extmat = [extendRows(:, end)______ extendRows_____ extendRows(:, 1)];
```

- 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*.

- Complete the code to start the definition of *applyExtended*, which is to have an extended array parameter (*matExt*).

```
_____ newmat = applyExtended(matExt);
```

- Write the statement to assign to *n* the number of rows (or columns) in the internal, un-extended square array.
- Suppose *i* represents the row index and *j* the column index. To apply the function *spread* to each internal cell of *matExt*, we use nested *for* loops and let indices *i* and *j* vary between two values. Give the initial value of *i* (or *j*).
- Give the final value for *i* (or *j*).
- 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*, and *W* to each internal cell site. Figure 11.2.9 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 *matExt*.
- Give the code to assign to *N* the value from *matExt* corresponding to the neighbor to the north.
 - Give the code to assign to *E* the value from *matExt* corresponding to the neighbor to the east.
 - 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 *matExt*.

```
newmat(_____, _____) = _____(site, N, E, S, W);
```

- Give the complete definition of *applyExtended*.

Simulation Program

Quick Review Question 9 Implement the loop in the *fire* function, assuming *grids* is a three-dimensional array containing in the first page the initial forest, which Quick Review Question 1 develops.

Display Simulation

Quick Review Question 10 This question develops the function *showGraphs* that returns a movie (*M*) 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.

- The function defines *map* 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
```

- Give the statement to assign to *m* the number of grids (pages) in *graphList*.
- Give the statement to assign to local variable *g* the *k*-th page in three-dimensional array *graphList*.
- Complete the commands to produce a graphic of *g* as a rectangular grid using the color map, *map*, from Part a. The values in *g* are 0, 1, and 2. Thus, for appropriate indexing into *map* as 1, 2, and 3, respectively, values at each site should be incremented by 1. The graphic should have no axes and be square.

```
_____ (g + 1)
_____ off
_____ square
```

- e. Give the command to store this graphics as the k -th frame in movie M .
- f. Give the entire definition of *showGraphs*.
- g. Give the command to play the movie 3 times.

Answers to Quick Review Question

1.
 - a. `global EMPTY TREE BURNING probLightning probImmune`
 - b. `treesOrBurns = (rand(n) < probTree);`
 - c. `burns = treesOrBurns .* (rand(n) < probBurning);`
 - d. `trees = treesOrBurns - burns;`
 - e. `empties = ones(n) - treesOrBurns;`
 - f. C. Exactly two have values of 0, while the other is 1.
 - g. `forest = empties * EMPTY + trees * TREE + burns * BURNING;`
 - h. `grids = zeros(n, n, t + 1);`
 - i. `grids(:, :, 1) = forest;`
2.
 - a. *EMPTY*
 - b. *EMPTY*
 - c. `if site == EMPTY`
`newSite = EMPTY;`
3.
 - a. *BURNING*
 - b. *EMPTY*, which indicates an empty cell
 - c. `elseif 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 (rand < probImmune)`
`newSite = TREE;`
`else`
`newSite = BURNING;`
`end;`
5. `elseif (rand < 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;
elseif site == BURNING
    newSite = EMPTY;
```

- ```

else
 if (N == BURNING | E == BURNING | S == BURNING | W == BURNING)
 if (rand < probImmune)
 newSite = TREE;
 else
 newSite = BURNING;
 end;
 elseif (rand < probLightning * (1 - probImmune))
 newSite = BURNING;
 else
 newSite = TREE;
 end;
end;

```
6. a. `mat(end, :)`  
b. `mat(1, :)`  
c. `extendRows = [mat(end, :); mat; mat(1, :)];`
7. a. `extendRows(:, end)`  
b. `extendRows(:, 1)`  
c. Either of the following statements is correct:  
`extmat = [extendRows(:, end) extendRows extendRows(:, 1)];`  
`extmat = [extendRows(:, end), extendRows, extendRows(:, 1)];`  
d. 9-by-9
8. a. `function newmat = applyExtended(matExt);`  
b. `n = size(matExt, 1) - 2;`  
c. 2  
d.  $n + 1$   
e. `site = matExt(i, j);`  
f. `N = matExt(i - 1, j);`  
g. `E = matExt(i, j + 1);`  
h. `newmat(i - 1, j - 1) = spread(site, N, E, S, W);`  
i. `function newmat = applyExtended(matExt);`  
`% APPLYEXTENDED - Function to apply`  
`% spread(site, N, E, S, W) to every interior`  
`% site of square array matExt and to return`  
`% the resulting array`  
`n = size(matExt, 1) - 2;`  
`newmat = zeros(n);`  
`for i = 2:(n + 1)`  
 `for j = 2:(n + 1)`  
 `site = matExt(i, j);`  
 `N = matExt(i - 1, j);`  
 `E = matExt(i, j + 1);`  
 `S = matExt(i + 1, j);`  
 `W = matExt(i, j - 1);`  
 `% must use function handle for fnchandle`  
 `newmat(i - 1, j - 1) = spread(site, N, E, S, W);`  
 `end;`  
`end;`
9. `for i = 2:(t + 1)`  
 `forestExtended = extendMat1(forest);`  
 `forest = applyExtended(forestExtended);`

```
 grids(:, :, i) = forest;
end;
```

10. **a.** `colormap(map);`  
**b.** `m = size(graphList, 3);`  
**c.** `g = graphList(:, :, k);`  
**d.** `image(g + 1)`  
`axis off`  
`axis square`  
**e.** `M(k) = getframe;`  
**f.** `function M = showGraphs(graphList);`  
`% SHOWGRAPHS - Function to return movie visualization`  
`% of grids in 3-dimensional array graphList`  
`map = [1 1 0; % EMPTY -> yellow`  
 `0.1 0.75 0.2; % TREE -> forest green`  
 `0.6 0.2 0.1]; % BURNING -> burnt orange`  
`colormap(map);`  
`m = size(graphList, 3);`  
`for k = 1:m`  
 `g = graphList(:, :, k);`  
 `image(g + 1)`  
 `axis off`  
 `axis square`  
 `M(k) = getframe;`  
`end;`  
**g.** `movie(M, 3)`