

11.3 Movement of Ants

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 11.3 on "Movement of Ants." Complete all code development in *MATLAB*.

Grid Initialization

Quick Review Question 1 This question refers to the initialization of a grid for ant movement. In an M-file, we define a function *gridInit* to return an initial grid, *grid*. The function begins as follows:

```
function grid = gridInit(n, maxChem, probAnt);  
% GRIDINIT - initialization of n-by-n-by-2 array for ant simulation  
% maxChem is the maximum chemical amount.  
% probAnt is the probability that a site has an ant.
```

Each grid site is an ordered pair with a chemical amount and a value indicating *EMPTY* or a direction an ant faces. Thus, *grid* can be a three-dimensional array of size *n*-by-*n*-by-2, Page 1 is an *n*-by-*n* array of chemical amounts, and Page 2 is an *n*-by-*n* array of directional values. For standard *MATLAB* arrays, the types of all elements must be the same. Thus, we define *EMPTY* and the four directions numerically, as follows:

```
global EMPTY NORTH EAST SOUTH WEST  
EMPTY = 0; NORTH = 1; EAST = 2; SOUTH = 3; WEST = 4;
```

- Initialize *grid* to be an *n*-by-*n*-by-2 array of zeros.
- Write a statement to return a random integer between 1 and 4 representing the four directions.
- Complete the code to assign values to the three-dimensional array *grid*. Leave chemical amounts, which are on the first page, as zero. With a probability of *probAnt*, a site contains an ant that faces in a random direction. Otherwise, the site does not contain an ant.

```
for i = 1:n  
    for j = 1:n  
        if _____  
            grid(_____) = floor(4 * rand + 1);  
        else  
            grid(_____) = _____;  
        end
```

```

    end
end

```

- d. Complete the loop to generate a chemical trail with no ants on the middle row of the grid, *grid*. The maximum amount of chemical, *maxChem*, occurs in column *n*, and for column *j* the amount of chemical is a fraction, j/n , expressed as an integer, of the maximum, *maxChem*. For example, if *maxChem* is 50, *j* is 10, and *n* is 17, then the amount of chemical in column 10 is the integer 29 because $(50)(10)/17 = 29.41$.

```

i = floor(n/2) + 1;
for j = 1:n
    grid(i, j, 1) = _____;
    grid(i, j, 2) = _____;
end

```

Sensing

Quick Review Question 2 This question relates to the code for the beginning of the *sense* function, which implements the first *sense* rule. Each parameter is a two-element vector with the first element being an integer amount of chemical and the second being *EMPTY* (0) or a direction (1-4).

- a. Complete the first line of the function M-file *sense.m*.

```

_____ newSite _____ sense(site, N, E, S, W)_____

```

- b. Complete the command in the function definition to indicate that *EMPTY*, *NORTH*, *EAST*, *SOUTH*, and *WEST* are not local.

```

_____ EMPTY NORTH EAST SOUTH WEST

```

- c. Complete the command to indicate that an empty site does not sense anything. Thus, its value remains the same.

```

if site_____ == _____
    newSite = _____;
end;

```

Quick Review Question 3 This question refers to the second *sense* rule that an ant turns at random in the direction of one of the neighbors with the greatest amount of chemical. We implement this rule in the *else* clause of the *if* statement that begins in Quick Review Question 2.

- a. Recall that each parameter is a two-element vector with the first element being an integer amount of chemical. Complete the assignment to *lst* of a list of chemical amounts in neighboring cells.

```

lst = [N_____, E_____, S_____, W_____];

```

- b. Complete the statement to assign the maximum level from *lst* to *mx*.

```
mx = _____;
```

- c. Complete the statement to assign to *posList* a list of indices in *lst* where the maximum level, *mx*, occurs.

```
posList = _____(lst _____ mx);
```

- d. Complete the statement to assign to *lng* the number of elements in *posList*.

```
lng = _____(posList, _____);
```

- e. Complete the statement to assign to *rndPos* a random integer between 1 and *lng*, inclusively, that is a possible index of *posList*.

```
rndPos = _____;
```

- f. Complete the statement to assign to *dirIndex* the value from *posList* with index *rndPos*.

```
dirIndex= posList_____;
```

- g. Define *dir* as follows:

```
dir = [NORTH, EAST, SOUTH, WEST];
```

Complete the assignment to *newSite* of the ordered pair indicating chemical level, which is the same as in *site*, and random direction in which to turn. Parts b-f developed the code to calculate the index of that direction, *dirIndex*.

```
newSite = [site_____, _____];
```

- h. Give this entire *sense* rule, which appears in the *else* clause.

Walking without Concern for Collision

Quick Review Question 4 Suppose the function M-file for *walk* begins as follows:

```
function newSite = walk(site, N, E, S, W, NE, SE, SW, NW, Nn, Ee, Ss, Ww);
% WALK - function to determine the value at a site after walking
global EMPTY NORTH EAST SOUTH WEST
a = site(1); % chemical amount
b = site(2); % EMPTY or direction in which ant faces
```

Complete the definition of the first *walk* rule: For a site that remains empty, the amount of chemical decrements by one but does not fall below 0. Because in an *if-elseif-else* statement *MATLAB* applies the first condition that matches, as we see shortly, this rule is not the first we define. Thus, the rule appears in an *elseif* segment.

```
if ...
    ...
```

```
elseif b == EMPTY
    newSite = [_____, _____];
```

Quick Review Question 5 Complete the definition of one form of the second *walk* bullet: If an ant wants to go north and the cell to the north is empty, the ant leaves the current cell, and the amount of chemical at that site increments by one. Similar rules apply to the east, south, and west directions. Assume $a = \text{site}(1)$ and $b = \text{site}(2)$. Because other rules are more specific and should apply first, implementation of this rule appears in an *elseif* segment.

```
elseif b == NORTH && _____ == EMPTY
    newSite = _____, EMPTY_____;
```

Quick Review Question 6 Complete the definition of the third *walk* rule: If an ant stays in a cell, the amount of chemical remains the same. Being the most general rule for a site with an ant, we place its implementation in an *else* segment.

```
else
    newSite = _____;
```

Quick Review Question 7 Complete the definition of one form of the fourth *walk* bullet: If an ant moves to a cell, the cell continues to have its same amount of chemical. In this case, the current site is initially empty and the cell to the north contains an ant that is facing south. Similar rules apply to ants in the east, south, and west directions facing the current site. Assume $a = \text{site}(1)$ and $b = \text{site}(2)$.

```
elseif _____ == EMPTY && _____ == SOUTH
    newSite = _____;
```

Walking with Concern for Collision

Quick Review Question 8

- a. Complete the definition of the *walk* rule for the situation in Figure 11.3.1 in which a collision should be avoided for the current ant that faces an empty cell to the north and a northeast ant that faces west. Assume $b = \text{site}(2)$.

```
elseif b == _____ && _____ == EMPTY && _____ == WEST
    newSite = _____;
```

- b. Give the number of similar such rules.

Quick Review Question 9

- a. Complete the definition of the *walk* rule for the situation in Figure 11.3.1 in which a collision should be avoided where the current site is empty but ants to the north and east both want to move into the site. In this situation, if the amount of chemical at the site is positive, it decrements by 1. Assume $a = \text{site}(1)$ and $b = \text{site}(2)$. Because of the specific nature of this condition, we can make it the first implemented rule, which begins the *if* statement.

```
if _____ == EMPTY && _____ == SOUTH && _____ == WEST
    newSite = _____;
```

- b. Give the number of similar such rules. If the situation in Part a is in an *if* segment, these similar rules are in *elseif* segments.

Simulation

Quick Review Question 10 Suppose the M-file for function *ants* begins as follows, where *grid* is a parameter for the initial grid and *t* is the number of time steps for the simulation:

```
function gridList = ants(grid, t);
```

- a. Write a statement to assign to *n* the number of rows (or columns) in *grid*.
 b. Write a statement to establish *gridList* as an array of zeros. The size of the array is *n*-by-*n*-by-2-by- $(t + 1)$ because *gridList* is to store the initial *grid* and a grid for each of the *t* time steps. Moreover, each grid is a three-dimensional array with Page 1 storing an *n*-by-*n* array of chemical amounts, and Page 2 having corresponding values 0-4 (for *EMPTY* and the directions to which an ant can face).
 c. Complete the statement to store *grid* in *gridList*

```
gridList(_____, _____, _____, _____) = grid;
```

- d. Implement the loop in the *ants* function. Be sure to use function handles for *sense* and *walk* in calls to *applyExtended1* and *applyExtended2*, respectively.

Visualizing the Simulation

Quick Review Question 11 Suppose, as Quick Review Question 10 describes, *gridList* is a multidimensional array of grids for the simulation at consecutive time steps.

- a. Write a statement to assign to *n* the number of rows (or columns) in a grid of *gridList*.
 b. Complete the statement to assign to *maxChem* the maximum amount of chemical in any cell of any grid in *gridList*. The chemical amounts are on Page 1 of each grid. Because three other dimensions occur, to get the over-all maximum, we take the maximum three times.

```
maxChem = _____(_____(_____(gridList(_____, _____, 1, _____))));
```

- c. We need $(maxChem + 1)$ levels of grey to represent integer chemical amounts, 0 through *maxChem*, at empty sites. Similarly, we need $(maxChem + 1)$ levels of red for sites with ants. Thus, in all, our color map must hold $(2maxChem + 2)$ rows of red, green, and blue values. Initialize *map* as such an array of zeros.

- d. Establish the gray colors in the loop below. With index i going from 1 through $maxChem + 1$, we must subtract 1 from i in the calculation of amt for corresponding chemical amounts from 0 through $maxChem$, respectively. In this implementation, we have the amount of red, green, and blue ranging from 1.0 down to 0.5. For example, $i = 1$ corresponds to $(i - 1) = (1 - 1) = 0$ amount of chemical, and the amount of each primary color is as follows:

$$amt = 1.0 - ((1 - 1) * 0.5) / maxChem = 1.0$$

At the other extreme, $i = maxChem + 1$ corresponds to $(i - 1) = maxChem$ amount of chemical, and the calculation of the amount of each color, amt , is as follows:

$$\begin{aligned} amt &= 1.0 - ((maxChem + 1 - 1) * 0.5) / maxChem \\ &= 1.0 - (maxChem * 0.5) / maxChem \\ &= 1.0 - 0.5 = 0.5 \end{aligned}$$

```
for i = 1:maxChem + 1;
    amt = 1.0 - ((i - 1) * 0.5) / maxChem;
    map(_____, _____) = [_____, _____, _____];
end;
```

- e. We have a similar loop for the shades of red. Complete the code to place these rows after the rows established in part d.

```
for i = 1:_____ ;
    amt = 1.0 - (_____ * 0.5) / maxChem;
    map(i + _____, :) = [_____, _____, _____];
end;
```

- f. Write a statement to establish *map* as a color map.
g. Initialize *gr* as an n -by- n array of zeros. Each element *gr* will store the *map* row index for the color that corresponds to the chemical amount in the corresponding *grid* site.
h. Write a statement to assign to *m* the number of grids in *gridList*.
i. A *for* loop displays a visualization of each of the m grids. Write a statement to assign to *g* the k th grid in *gridList*.
j. Recall that each cell of a grid is an ordered pair consisting of the amount of chemical and $EMPTY = 0$ or a direction ($NORTH = 1$, $EAST = 2$, $SOUTH = 3$, or $WEST = 4$) on two pages of *g*. Complete the nested loops to assign the appropriate index to each i - j element of *gr*. (See Part g.)

```
for i = 1:n
    for j = 1:n
        if g(i, j, _____) == EMPTY
            gr(i, j) = g(i, j, _____) + _____; % gray
        else
            gr(i, j) = g(i, j, _____) + _____; % shade of red
        end;
    end;
end;
```

- k. Write a segment to display an image of *gr* as square with no axes.
l. Write a statement to assign this graphics as the k -th frame in movie *M*.

Answers to Quick Review Questions

1.
 - a. `grid = zeros(n, n, 2);`
 - b. `floor(4 * rand + 1)`
 - c.


```
for i = 1:n
    for j = 1:n
        if rand < probAnt
            grid(i, j, 2) = floor(4 * rand + 1);
        else
            grid(i, j, 2) = EMPTY;
        end
    end
end
```
 - d.


```
i = floor(n/2) + 1;
for j = 1:n
    grid(i, j, 1) = _____;
    grid(i, j, 2) = _____;
end
```
2.
 - a. `function newSite = sense(site, N, E, S, W);`
 - b. `global EMPTY NORTH EAST SOUTH WEST`
 - c.


```
if site(2) == EMPTY
    newSite = site;
end;
```
3.
 - a. `lst = [N(1), E(1), S(1), W(1)];`
 - b. `mx = max(lst);`
 - c. `posList = find(lst == mx);`
 - d. `lng = size(posList, 2);`
 - e. `rndPos = floor(lng * rand + 1);`
 - f. `dirIndex= posList(rndPos);`
 - g. `newSite = [site(1), dir(dirIndex)];`
 - h.


```
else
    lst = [N(1), E(1), S(1), W(1)];
    mx = max(lst);
    posList = find(lst == mx);
    lng = size(posList, 2);
    rndPos = floor(lng * rand + 1);
    dirIndex= posList(rndPos);
    dir = [NORTH, EAST, SOUTH, WEST];
    newSite = [site(1), dir(dirIndex)];
end;
```
4.


```
a = site(1);
b = site(2);
if ...
    ...
elseif b == EMPTY
    newSite = [max(a - 1, 0), EMPTY];
```
5.


```
elseif b == NORTH && N(2) == EMPTY
    newSite = [a + 1, EMPTY];
```

6. else
 - newSite = site;
7. elseif b == EMPTY && N(2) == SOUTH
 - newSite = [a, SOUTH];
8. a. elseif b == NORTH && N(2) == EMPTY && NE(2) == WEST
 - newSite = site;
- b. 11
9. a. if b == _____ && _____ == SOUTH && _____ == WEST
 - newSite = [max(a - 1, 0), _____];
- b. 5
10. a. n = size(grid, 1);
 - b. gridList = zeros(n, n, 2, t+1);
 - c. gridList(:, :, :, 1) = grid;
 - d. for i = 1:t
 - emat1 = extendMat1(grid);
 - gridSense = applyExtended1(emat1);
 - emat2 = extendMat2(gridSense);
 - grid = applyExtended2(emat2);
 - gridList(:, :, :, i+1) = grid;
- end;
11. a. n = size(grid, 1);
 - b. maxChem = max(max(max(gridList(:, :, 1, :))));
 - c. map = zeros(2 * maxChem + 2, 3);
 - d. for i = 1:maxChem + 1
 - amt = 1.0 - ((i - 1) * 0.5) / maxChem;
 - map(i, :) = [amt, amt, amt];
 - end;
 - e. for i = 1:maxChem + 1
 - amt = 1.0 - ((i - 1) * 0.5) / maxChem;
 - map(i + maxChem + 1, :) = [amt, 0, 0];
 - end;
 - f. colormap(map);
 - g. gr = zeros(n, n);
 - h. m = size(gridList, 4);
 - i. g = gridList(:, :, :, k);
 - j. for i = 1:n
 - for j = 1:n
 - if g(i, j, 2) == EMPTY
 - gr(i, j) = g(i, j, 1) + 1;
 - else
 - gr(i, j) = g(i, j, 1) + maxChem + 2;
 - end;
 - end;
 - end;
 - k. image(gr)
 - axis off
 - axis square
 - l. M(k) = getframe;