



Matrices (2D Arrays)

Lecture 11 (Feb 28)
CS100M - Spring 2006

Announcements

- Prelim 1
 - Will be returned at end of lecture
 - Mean = 81; standard deviation = 14.6
 - Prelim 1 solutions will be online by tomorrow
 - Don't go immediately to the online solutions
 - For problems you got wrong, take time to think about how to solve the problem (and consider talking to a staff member) *before* looking at the solution
- Project 3
 - Online later this week
 - Due: Thursday, March 9
- Section is in lab this week

Topics

- Reading: CFile 9, Section 9.1
- Recall
 - Matlab vectors (1D arrays)
 - Characters & Strings
- Plans for today
 - Matrices (2D arrays)

2D Arrays (Matrices)

- Recall: An *array* is a named collection of data values organized into rows and/or columns
- A 2D array is a table, called a matrix
- Two indices are used to identify the position of a item in a matrix
 - $M(r, c)$ refers to the item in row r , column c
 - Just like vectors, indices for matrices start at 1
 - Example: $M(2, 3)$ refers to 6

7	0	5
2	4	6
3	8	1

Creating a Matrix

- Matlab makes it easy to create a matrix
 - Use brackets
 - Comma or space separates items in *same* row
 - Semicolon ";" indicates a new row
 - Example: $M = [7\ 0\ 5; 2\ 4\ 6; 3\ 8\ 1]$ creates

7	0	5
2	4	6
3	8	1

- The vector-creating functions can also create matrices
 - `zeros(2, 3)` % 2-by-3 matrix of zeros
 - `ones(3, 2)` % 3-by-2 matrix of ones
 - `rand(3, 4)` % 3-by-4 matrix of random numbers

Creating a Matrix, Continued

- You can build a new matrix out of smaller matrices (or vectors) — as long as all the dimensions match up
 - `[ones(1,4); 1:4]` works
 - `[ones(1,3); 1:4]` doesn't
 - `[ones(2,4); 1:4]` ~~doesn't~~ works
- If you start filling a matrix, Matlab will create it for you (unspecified values are set to 0)
 - Example: `B(2, 3) = 77`

1	1	1	1
1	2	3	4

0	0	0
0	0	77

Transpose of a Matrix

- If A is a matrix then A' is the transpose of A
 - The transpose of a matrix just swaps the rows and the columns
 - An item at position (r, c) becomes an item at position (c, r)
- Example: The transpose of $[1:3; 4:6]$ is



Finding the Dimensions of a Matrix

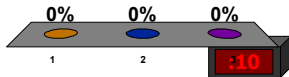
- Matlab provides a function for this: `size(M)`
- Examples

```
[nr, nc] = size(M) % Both # of rows and # of columns
nr = size(M, 1)    % # of rows
nc = size(M, 2)    % # of columns
```

What is $[7\ 0\ 5]'$?

1. Error; the transpose of a vector is illegal
- ➔ 2. The same as $[7; 0; 5]$
3. $[5\ 0\ 7]$

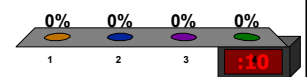
0 of 5



What happens when this statement is executed? `[nr nc] = size([7 0 5])`

1. Error; use `length()` instead of `size()` for a vector
2. nr is 3; nc is 1
- ➔ 3. nr is 1; nc is 3
4. nr and nc are both 3

0 of 5

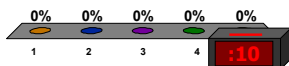


What happens when these statements are executed?

```
A = [4 4]
A = [A' ones(2,1)]
A = [1 2 3 4; A A]
```

1. Error in 2nd statement
2. Error in 3rd statement
- ➔ 3. In the end, A is a 3-by-4 matrix
4. In the end, A is a 4-by-3 matrix
5. In the end, A is a vector of length 12

0 of 5



Example: Finding Min Value in a Matrix

- Function header

```
function val = minInMatrix(M)
% Return min value in matrix M
```
- Resulting Code

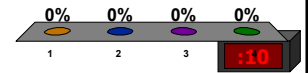
```
function val = minInMatrix(M)
% Return min value in matrix M
val = M(1,1);
[nr, nc] = size(M);
for r = 1:nr
    for c = 1:nc
        val = min(val, M(r,c));
    end
end
```
- Pseudocode:
 - Initialize val
 - Loop through all items in M
 - Update val at each item

Pattern for Traversing a Matrix M

```
[nr, nc] = size(M);
for r = 1:nr
    for c = 1:nc
        % Do something with M(r, c)
    end
end
```

```
% M is an nr-by-nc matrix
for r = 1:nr
    for c = 1:nc
        A(c,r) = M(r,c);
    end
end
```

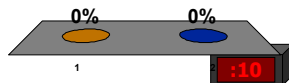
1. A is the same as M, but with columns in reverse order
2. A is the same as M, but with rows in reverse order
- 3. A is the transpose of M
4. A and M are the same



0 of 5

```
% A is an m-by-n matrix
for g = 1:m
    for h = 1: floor(n/2)
        A(g, h) = A(g, n-h+1);
    end
end
```

- 1. This code reflects the right half of A onto the left half
2. This code reflects the bottom half of A onto the top half



0 of 5

Submatrices

- Matlab colon notation can be used to easily create a *submatrix* of a matrix
- Example: Let $M = \begin{bmatrix} 7 & 0 & 5 \\ 2 & 4 & 6 \\ 3 & 8 & 1 \end{bmatrix}$

7	0	5
2	4	6
3	8	1

- $M(1:2, 1:3)$ is

7	0	5
2	4	6

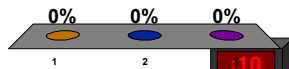
- $M(2:3, 1:2)$ is

2	4
3	8

- A single colon ":" can be used to represent "all indices"
- Thus $M(1:2, :)$ is the same as $M(1:2, 1:3)$

Let $M = \begin{bmatrix} 7 & 0 & 5; & 2 & 4 & 6; & 3 & 8 & 1 \end{bmatrix}$.
What does the following code produce?
 $[M(1:2, 2:3) \ M(2:3, 1:2)]$

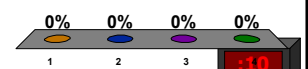
- 1. A 2-by-4 matrix
2. A 4-by-2 matrix
3. An error



0 of 5

Let $M = \begin{bmatrix} 7 & 0 & 5; & 2 & 4 & 6; & 3 & 8 & 1 \end{bmatrix}$.
What does the following code produce?
 $[M(1:2, :); \ M(2:3, 1:2)]$

1. A 2-by-5 matrix
2. A 4-by-2 matrix
3. A 4-by-3 matrix
- 4. An error

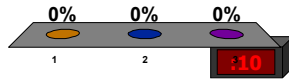


0 of 5

Let $v = [7\ 0\ 5\ 2\ 4\ 6\ 3\ 8\ 1]$.
 What does the following code produce?
`[v(1:4) ; zeros(3, 4)]`

1. A 3-by-4 matrix
- 2. A 4-by-4 matrix
3. An error

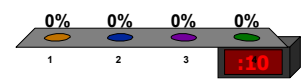
0 of 5



Let $v = [7\ 0\ 5\ 2\ 4\ 6\ 3\ 8\ 1]$.
 What does the following code produce?
`[v(1:9) ; v(9:1)]`

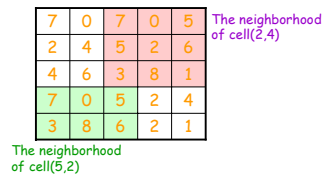
1. A 2-by-9 matrix
2. A 9-by-2 matrix
- 3. A 1-by-9 matrix
4. An error

0 of 5



Neighborhood of a Cell

- We define the *neighborhood of a cell* to be the cell itself and all adjacent cells (including diagonally adjacent)



Min of a Neighborhood

- Goal:
Write a function `minInNeighborhood(M, row, col)` that reports the minimum value in neighborhood of `cell(row, col)` in matrix `M`
- Function header
`Function val = minInNeighborhood(M, row, col)`
`% Return min in neighborhood of (row, col) in M`

Ask Yourself Questions

- Do we know how to solve a similar problem?
 - Yes, we already have code to find the min of a matrix
- Can we make a neighborhood into a matrix?
 - Yes, Matlab makes it easy to do submatrices
 - Neighborhood of `M(row, col)` is `M(row-1:row+1, col-1:col+1)`
- What happens near the edges?
 - Doesn't work near the edges: we "fall off"
- What can we do to fix up the edges?
 - We can make the code more complicated, or...
 - We can modify the matrix so we *can't* fall off
- If we add a border around `M`, what goes in the border?
 - `realmax`