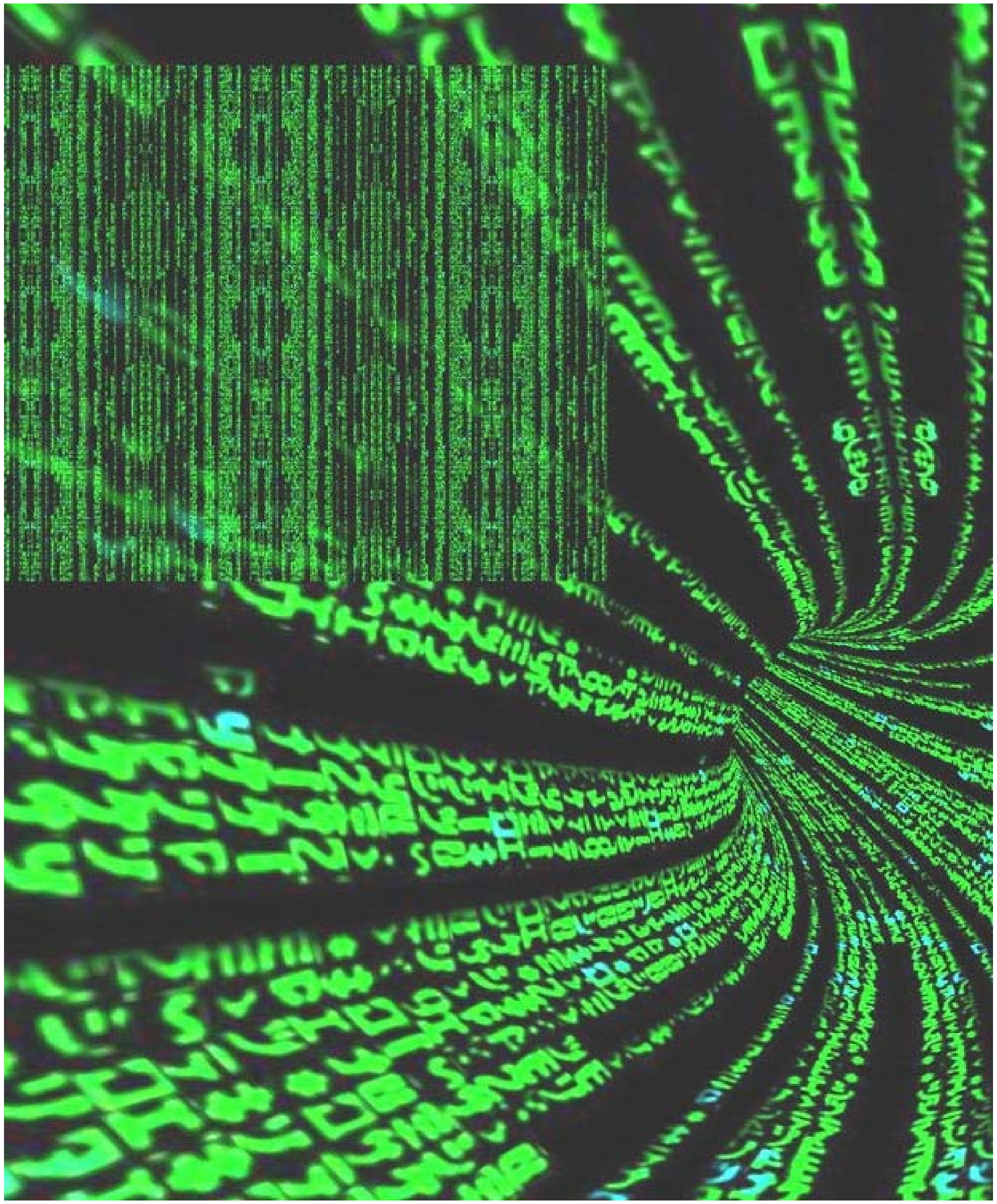


Matrices (2D Arrays)

Lecture 14 (Mar 6)
CS100M – Spring 2008

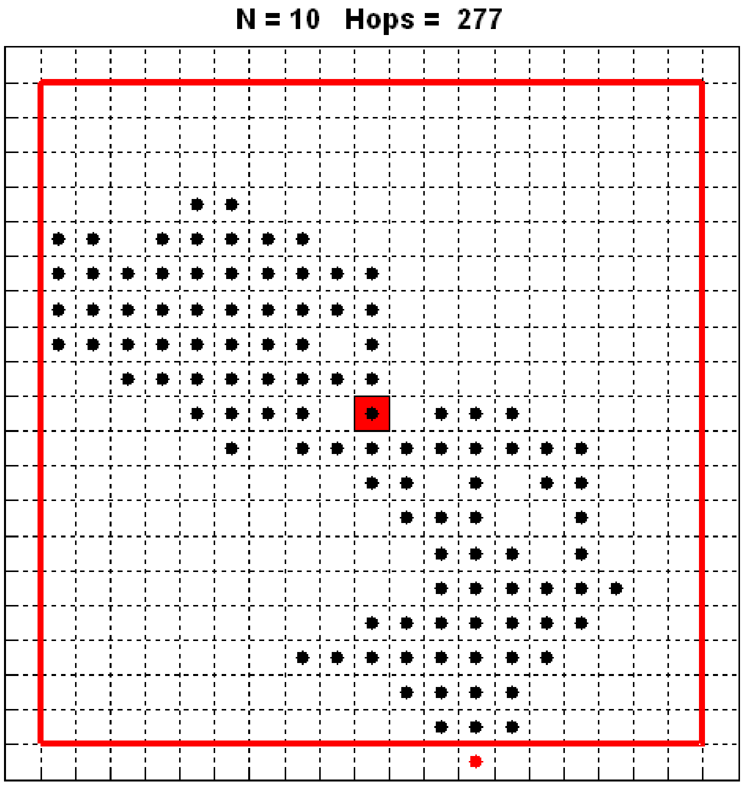
Announcements

- Prelim 2 is coming soon!
    - Date: Thursday, March 13
    - Time: 7:30-9:00 pm
    - If you have a conflict, tell us (email Kelly Patwell) immediately
        - We accommodate only university-accepted conflicts
        - Leaving early for spring break doesn't count

- Questions on current Project?

- Today's topics
    - Recall
        - Matlab vectors (1D arrays)
        - Characters & Strings
    - Plans for today
        - Matrices (2D arrays)

Random Walk Simulation



Start at the middle tile

Repeat until boundary reached:

- Pick a compass heading (N, E, S, W) at random

- Move one tile in that direction

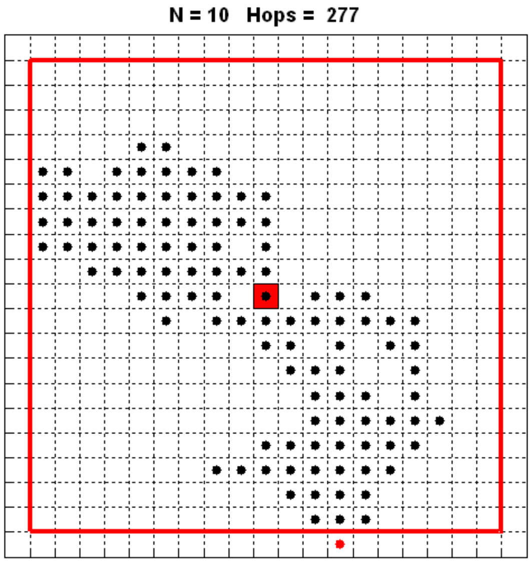
Function that Returns the Path

```
function [x y] = RandomWalk2D(N)

k = 0; xc = 0; yc = 0;

while abs(xc)<N && abs(yc)< N
        Take another hop
        Update location (xc,yc)

k = k + 1; x(k) = xc; y(k) = yc;
end
```



Choosing a Random Direction

```
if rand < .5
   if rand < .5
     xc = xc + 1;             % East
   else
     xc = xc - 1;             % West
   end
 else
   if rand < .5
     yc = yc + 1;             % North
   else
     yc = yc - 1;             % South
   end
 end
```

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*
- This example has 3 rows and 4 columns

	col 1	col 2	col 3	col 4
row 1	7	0	9	5
row 2	2	4	7	6
row 3	3	8	3	1

Creating a Matrix: “By Hand”

- Comma or space separates items in *same* row
- Semicolon “;” indicates a new row

- Example:

```
>> M = [7 0 5; 2 4 6; 3 8 1]

M =

     7     0     5
     2     4     6
     3     8     1
```

7	0	5
2	4	6
3	8	1

Creating a Matrix: Using a Function

- The vector-creating functions can also create matrices

```
>> M = zeros(4, 3)

M =

     0     0     0
     0     0     0
     0     0     0
     0     0     0
```

```
>> M = ones(3, 5)

M =

     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
```

Creating a Matrix of Dice Rolls

```
>> M = ceil(6*rand(5, 10))

M =

     2     4     6     4     3     4     5     6     6     2
     2     1     4     5     4     3     4     2     5     2
     2     5     3     4     5     5     5     6     1     4
     5     3     6     3     2     4     6     2     1     2
     2     6     5     2     6     3     4     2     6     3
```

More Matrix Creation

- Only the last row is non-zero

```
>> M = [ zeros(4, 3) ; [ 3 3 3 ] ]

M =

     0     0     0
     0     0     0
     0     0     0
     0     0     0
     3     3     3
```

- Only the first column is non-zero

```
>> M = [ [8; 2; 3] zeros(3, 4)]

M =

     8     0     0     0     0
     2     0     0     0     0
     3     0     0     0     0
```

Even More Matrix Creation

- Dimensions must match

```
>> [ones(2,4); 1:4]

ans =
    1    1    1    1
    1    1    1    1
    1    2    3    4

>> [ones(1,3); 1:4]
??? Error using ==> vertcat
All rows in the bracketed
   expression must have the
   same number of columns.
```

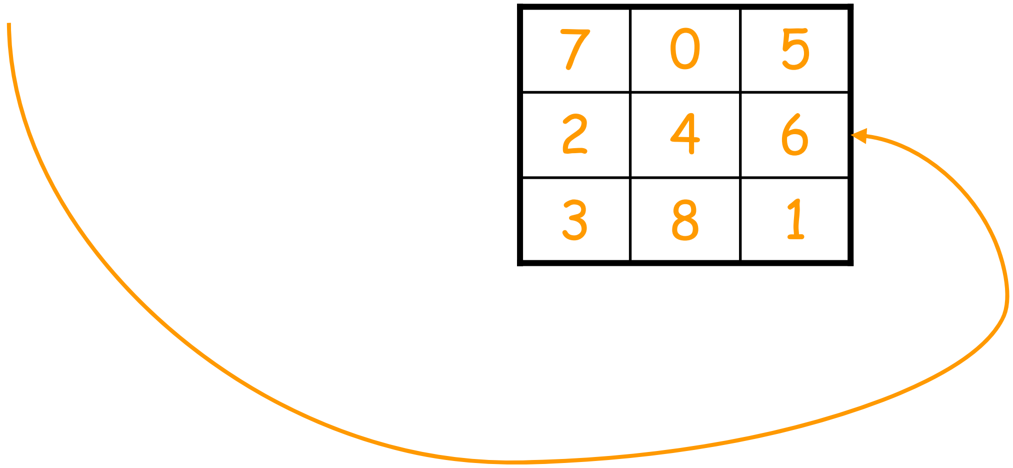
- If you start filling a matrix, Matlab will create it for you
  - Unspecified values are set to 0

```
>> B(2, 3) = 77

B =
    0    0    0
    0    0   77
```

Subscripting: Individual Entry

- 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

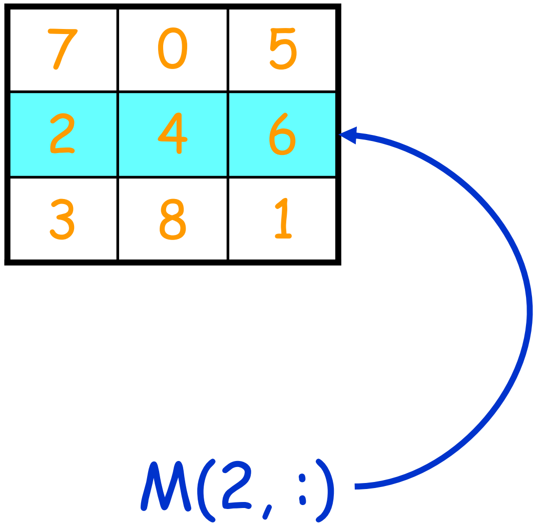
Subscripting: Entire Row

- A single colon ":" can be used to represent *all indices*

```
>> M = [7 0 5; 2 4 6; 3 8 1]
M =
     7     0     5
     2     4     6
     3     8     1

>> M(2, :)

ans =
     2     4     6
```



7	0	5
2	4	6
3	8	1

M(2, :)

Scaling a Row

M(2, :) = 10 * M(2, :)

7	0	5
2	4	6
3	8	1

Before

7	0	5
20	40	60
3	8	1

After

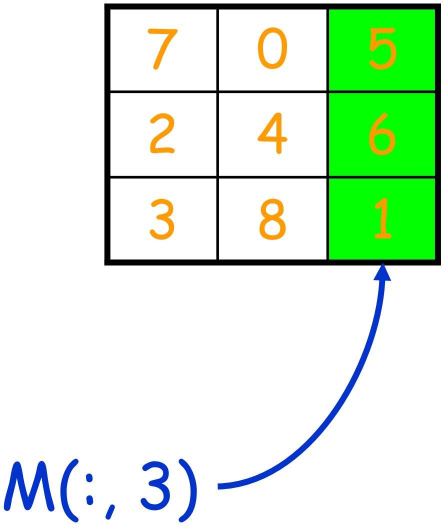
Subscripting: Entire Column

```
>> M = [7 0 5; 2 4 6; 3 8 1]

M =
     7     0     5
     2     4     6
     3     8     1

>> M(:, 3)

ans =
     5
     6
     1
```



7	0	5
2	4	6
3	8	1

M(:, 3)

Incrementing a Column

```
M(:, 3) = 1 + M(:, 3)
```

7	0	5
2	4	6
3	8	1

Before

7	0	6
2	4	7
3	8	2

After

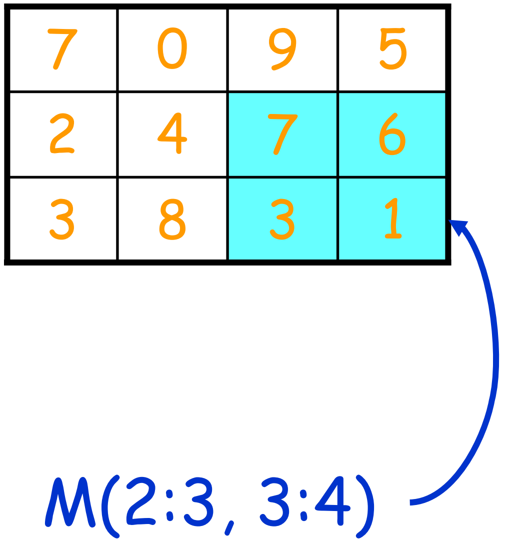
Subscripting: Subarray

```
>> M = [7 0 9 5; 2 4 7 6; 3 8 3 1]

M =
    7     0     9     5
    2     4     7     6
    3     8     3     1

>> M(2:3, 3:4)

ans =
    7     6
    3     1
```



7	0	9	5
2	4	7	6
3	8	3	1

M(2:3, 3:4)

Zeroing a Subarray

```
M(2:3, 3:4) = zeros(2, 2)
```

7	0	9	5
2	4	7	6
3	8	3	1

Before

7	0	9	5
2	4	0	0
3	8	0	0

After

Example: Create this Matrix

- Goal: Create an m-by-n matrix where every entry is of the form 10*r+c where r and c are the row and column indices, respectively

```
function A = createExample(m, n)
for r = 1:m
    for c = 1:n
        A(r, c) = 10*r + c;
    end
end
```

11	12	13	14	15
21	22	23	24	25
31	32	33	34	35

Finding the Dimensions of a Matrix

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

- Examples

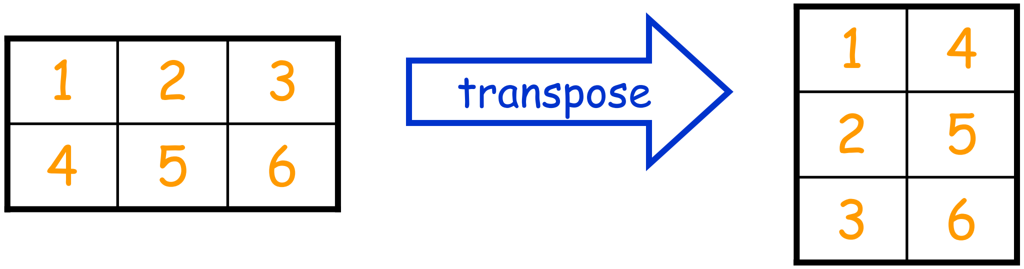
```
[nr, nc] = size(M)    % Both # of rows and # of columns
nr = size(M, 1)       % # of rows
nc = size(M, 2)       % # of columns
```

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
```

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]



1	2	3
4	5	6

transpose

1	4
2	5
3	6

What is [7 0 5]' ?

A. Error; the transpose of a vector is illegal

B. The same as [7; 0; 5]

C. [5 0 7]

What happens when this statement is executed?

```
[nr nc] = size([7 0 5])
```

A. Error; use length( ) instead of size( ) for a vector

B. nr is 3; nc is 1

C. nr is 1; nc is 3

D. nr and nc are both 3

What happens when these statements are executed?

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

A. Error in 2nd statement

B. Error in 3rd statement

C. In the end, A is a 3-by-4 matrix

D. In the end, A is a 4-by-3 matrix

E. In the end, A is a vector of length 12

What happens when the code is executed?

```
[nr nc] = size(M);
for r = 1:nr
        for c = 1:nc
                A(c,r) = M(r,c);
        end
end
```

A. A is the same as M, but with columns in reverse order

B. A is the same as M, but with rows in reverse order

C. A is the transpose of M

D. A and M are the same

What does this code do?

```
[m n] = size(M);
for g = 1:m
    for h = 1: floor(n/2)
            M(g, h) = M(g, n-h+1);
    end
end
```

A. This code reflects the right half of M onto the left half
B. This code reflects the bottom half of M onto the top half
C. This code leaves the matrix M unchanged
D. This code produces an error message

What does the following code produce?

```
M = [7 0 5; 2 4 6; 3 8 1]
W = [M(1:2, : ) ; M(2:3, 1:2)]
```

A. W is a 2-by-5 matrix

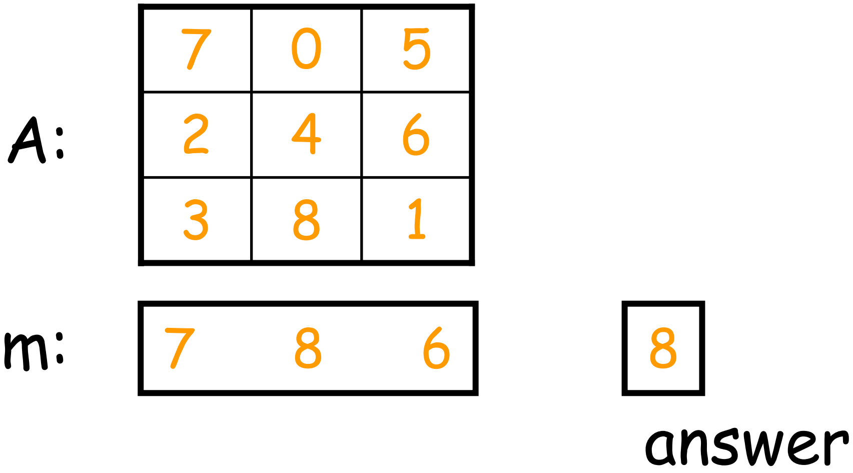
B. W is a 4-by-2 matrix

C. W is a 4-by-3 matrix

D. There is an error

Finding the Maximum Value

```
m = max(A)
answer = max(m)
```



A:

7	0	5
2	4	6
3	8	1

m:

7	8	6

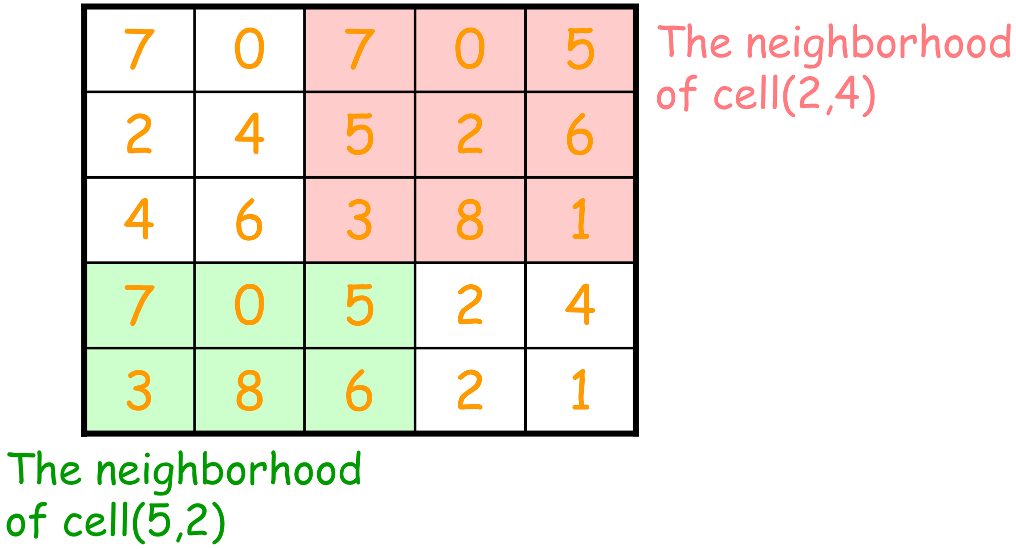
8

answer

or you can use iteration

Neighborhood of a Cell

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



7	0	7	0	5
2	4	5	2	6
4	6	3	8	1
7	0	5	2	4
3	8	6	2	1

The neighborhood of cell(2,4)

The neighborhood of cell(5,2)

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?
  - M(max(1,row-1):min(nr,row+1), max(1,col-1):min(nc,col+1))