Matlab Vectors

Lecture 9 (Feb 21)
CS100M – Spring 2006

Announcements

- Prelim 1
  - Feb 23 at 7:30pm
  - Room assignments
    - Last names starting with A-L: Uris Auditorium
    - Last names starting with M-R: Goldwin Smith Lewis
    - Last names starting with S-Z: Goldwin Smith HEC
  - This information is also on the website
    - Follow the links to Exams and to Prelim I

Topics

- Reading: CFile 5, Section 5.1
- Plans for today
  - Matlab vectors

Definitions

- An *array* is a named collection of data values organized into rows and/or columns
- Matlab makes it easy to create & use both 1D arrays and 2D arrays
  - We start with 1D arrays, called *vectors* in Matlab
  - We identify a single data item in a vector by using an *index*
  - In Matlab, the first item of a vector is at index 1

scores	10	15	23	18	18	21
	1	2	3	4	5	6

Accessing Values in a Vector

- To access, say, the 4^{th} item in vector scores we use scores(4)
  - In words, this is often called "scores sub 4" because it corresponds to the way subscripts are used in mathematics (e.g., $scores_4$)
- In this example, scores(1) holds the value 10
  - scores(2) holds the value 15
  - scores(3) holds the value 23
  - etc.

scores	10	15	23	18	18	21
	1	2	3	4	5	6

Using Values Held in a Vector

scores	10	15	23	18	18	21
	1	2	3	4	5	6

- You can use a vector variable with its index just like any other Matlab variable
  - sum = scores(1) + scores(2) % sum is set to 25
  - scores(3) = 34 % Changes scores(3)

scores	10	15	34	18	18	21
	1	2	3	4	5	6

Creating Vectors "By Hand"

- Here's one way that we could have created the vector scores

  scores = [10 15 23 18 18 21]   % Create the vector

  - The square brackets indicate a vector
  - You can list the values separated by either spaces or commas

  scores = [10, 15, 23, 18, 18, 21]   % Same thing using commas

scores	10	15	23	18	18	21
	1	2	3	4	5	6

Programming with Vectors

- Matlab provides many built-in functions for working with vectors
- For now, we use just one built-in function, the function length()
  - This tells how many items are in the vector
  - E.g., length(scores) returns the value 6
- Example: Determine the sum of all values in the vector scores

```
sum = 0;
for k = 1:length(scores)
   sum = sum + scores(k);
end
```

Example: Average

- Goal: Write a Matlab function average(v) which returns the average of the values held in vector v
- Pseudocode:
  - Add all the items
  - Report sum/(# of items)

```
function avg = average(v)
sum = 0;
for k = 1:length(v)
   sum = sum + v(k);
end
avg = sum / length(v);
```

Example: Find the Maximum

- Goal: Write a Matlab function maximum(v) which returns the maximum value stored in vector v
- Pseudocode:
  - Initialize theMax
  - Compare theMax with each item in turn, updating as needed
  - Report theMax

```
function theMax = maximum(v)
theMax = v(1);
for k = 2:length(v)
   theMax = max(theMax, v(k));
end
```

- What happens when length(v) is 0?
  - We get an error message
- FYI: You can create an empty vector like this:

  v = [ ];   % Empty vector

Special Functions for Creating Vectors

- Some vectors are used so often that there are special functions for creating them

  zeros(1, 5) % A vector of length 5 holding all zeros
  ones(1, 7) % A vector of length 7 holding all ones
  rand(1, 4) % A vector of length 4 holding random numbers

- Why the extra function arguments?
  - Matlab (= Matrix Laboratory) uses matrices (2D arrays) as its default
  - Thus, zeros(3, 4) produces a 3-by-4 matrix of zeros
  - zeros(1, 5) produces a 1-by-5 matrix (i.e., a single row of a matrix; also called a *row vector*)
  - zeros(5, 1) produces a 5-by-1 matrix (i.e., a single column of a matrix; also called a *column vector*)

Shortcuts for Creating Vectors

- We've already seen another way to create vectors when we were using for-loops
  - We can use ":" notation

  vec = 1:7;   % [1 2 3 4 5 6 7]
  vec = 10: -2: 0   % [10 8 6 4 2 0]

- FYI
  - The for-loop actually converts the ":" notation into a vector before it executes
  - A for-loop will work with *any* vector (e.g., `for k = [2 3 5 7 11 13 17 19]`)

Appending to Vectors

- This code will create the vector xx and fill it with values:

```
xx(1) = 111;      % [111]
xx(2) = 222;      % [111 222]
xx(3) = 333;      % [111 222 333]
```

- You can even skip ahead (unspecified items are set to 0)

```
xx(6) = 666;      % [111 222 333 0 0 666]
```

- You don't even have to know the last index-value (because Matlab treats "end" as a special value in subscripts)

```
xx(end+1) = 777;  %[111 222 333 0 0 666 777]
```

Combining Vectors

- If you put a vector inside a vector then Matlab uses the values to make one new vector
  - Examples

```
[[1 2 3] [4 5 6]]            % [1 2 3 4 5 6]
[ones(1, 3), zeros(1,2)]     % [1 1 1 0 0]
[5 4 3 2 1 [1:1:5]]          % [5 4 3 2 1 1 2 3 4 5]
[1 2 zeros(1, 3)]            % [1 2 0 0 0]
```

Example: Cumulative Sum

- Write a function csum(v) that returns the cumulative sum of vector v
  - Example:

```
[2 4  6  6]   % Original vector
[2 6 12 18]   % Cumulative sum
```

  - Note that the cumulative sum of v as the same length as the original vector v
- Algorithm ideas
  - Create a vector (call it sum) to hold the sums
  - We can figure out sum(k) if we know sum(k-1) and v(k)

Example: Polynomial Evaluation

- Write a function to evaluate an n^{th} order polynomial of x:

  $a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$

- polyEval(coeff, x)
  Return the value of the polynomial represented by the vector coeff evaluated at value x
- Note that vector coeff has length n+1
- Note that coeff(1) corresponds to a_0
- Algorithm ideas
  - We need a sum and a loop
  - Each time through the loop, we add the next term to the sum
- Partial code

```
sum = 0;
for k = 1:length(coeff)
   xpow = x^(k-1);
   sum = sum + coeff(k)*xpow;
end
```

Example: Random Walk

- Write a function randomWalk(n) to perform n steps of a random walk in the plane starting from (0,0)
  - Function header: function randomWalk(n)
- At each step, possible moves are up, down, left, or right
- Display the walk
  - This part turns out to be easy
  - plot(x, y, '-') where x and y are vectors draws connecting lines from (x(0), y(0)) to (x(1), y(1)) to (x(2), y(2)) to…

Random Walk Algorithm

- To do the drawing, we need all the steps stored in two vectors: x and y
- For n steps we need vectors of length n+1
- E.g., if we use vectors of length 2, we can hold
  - The starting position (0,0)
  - And one step to either (1,0), (0,1), (-1,0), or (0,-1)
- Pseudocode

```
Load x and y with n+1 zeros
for each step k
   Choose a random direction
   Update x(k+1) and y(k+1)
Draw the result
```