

MATLAB Tutorial

Csci 5521 Machine Learning

Using Matlab in Unix/Linux

- Add modules
 - `module load math/matlab/v.2014a`
 - `module initadd math/matlab/v.2014a`
- Matlab 2014a/b, 2015a/b is recommended
 - check your version
 - “.cshrc” in home directory (assuming the shell is tcsh)
- Availability
 - computing resources
 - <http://www.cs.umn.edu/resources/facilities/labs.php>
 - student copy (CSE Labs account is required)
 - <https://wwws.cs.umn.edu/matlab/student/>

Matlab GUI

- Command window
 - the main window where you type commands directly to the MATLAB interpreter
 - an example of Matlab command
 - `disp('Hello World!');`
- Editor window
 - a simple text editor where you can load, edit and save complete MATLAB programs
 - debug/run
 - open editor window
 - from menu (File->New->Blank M-File)
 - `edit MyProgram.m` (or any filename of your script)
- Help window
 - It also has a number of example programs and tutorials.
 - show short help in command window
 - `help sort` (or any function name)

Loading data from disk

- Supported types
 - Text
 - white-space/tab delimited
 - Spreadsheet
 - *.xls, *.xlsx, *.csv
 - MATLAB formatted data
 - *.mat
 - Other types
 - images
 - sound

Loading data from disk

- How to load data in Matlab

- use “dlmread” function

- a.txt:

- 1,2,3

- 4,5,6

- ```
>> data = dlmread('a.txt');
```

- data =

- 1 2 3

- 4 5 6

- other functions:

- load, readtable, textread, textscan, fscanf, xlsread

# Variables and Assignment

- Variable types
  - double
    - `a=6;`
    - array
      - `MyArray = [1 2 3];` (1x3 double)
  - char
    - `letter = 'A';`
    - char array (string)
      - `Name='Mark';` (1x4 char)
  - other types
    - cell, struct, class
- Display the contents of a variable
  - `disp(variable);` (e.g. `disp(MyArray);`)
  - type the name of variable and press “enter” without semicolon
- Note: MATLAB does not require you to declare the names of variables in advance of their use.

# Array operations

- Define one dimensional array
  - row vector
    - `MyArray = [1 2 3 4 5];`
    - `MyArray = zeros(1, 5);`
  - column vector
    - `MyArray = [1; 2; 3; 4; 5];` or `MyArray = [1 2 3 4 5]';`
    - `MyArray = zeros(5, 1);`
- Access/modify values
  - `a = MyArray(1);`
  - `MyArray(1)=3;`
  - `MyArray(2)=6;`
- Note1: Use `[]` to define array and use `()` to access array
- Note2: Indexes must be positive integers. The smallest index is 1.

# Array operations

- Generate arrays containing sequences with the ':' operator
  - start:stop
    - `a = 1 : 9;`  
is equivalent to `a = [1 2 3 4 5 6 7 8 9];`
  - start:increment:stop
    - `b = 1 : 2 : 9;`  
is equivalent to `b = [1 3 5 7 9];`
- Select sub-parts of the array with the ':' operator
  - `b(3:5)`  
is equivalent to `b([3 4 5])`, whose value is `[5 7 9]`
  - `b(1:2:5)`  
is equivalent to `b([1 3 5])`, whose value is `[1 5 9]`
  - `b(3:end)`  
is equivalent to `b([3 4 5])` since `b` contains 5 elements



# Matrix operations

- Define two dimensional array

- $A = [1\ 2\ 3; 4\ 5\ 6];$

A =

1 2 3

4 5 6

- Building Matrices
  - $A = \text{zeros}(2,3);$
  - $A = \text{rand}(2,5);$
  - $A = \text{eye}(6);$
  - $A = \text{ones}(5);$

# Matrix operations

- Access/modify values
  - `variable_name(row_index, column_index)`
    - `a = A(2,1);` (a will be 4)
    - `A(2,1) = 7;`

before

A =

|   |   |   |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |

after

A =

|   |   |   |
|---|---|---|
| 1 | 2 | 3 |
| 7 | 5 | 6 |

# Matrix operations

- Select sub-parts of the array with the ':' operator

A =

|    |    |    |    |    |
|----|----|----|----|----|
| 76 | 71 | 82 | 44 | 49 |
| 74 | 3  | 69 | 38 | 45 |
| 39 | 28 | 32 | 77 | 65 |
| 66 | 5  | 95 | 80 | 71 |
| 17 | 10 | 3  | 19 | 75 |

- $A(2:4, 2)$
- $A(3, 1:4)$
- $A([1\ 2], [3\ 4])$
- Q?  $A(1:2:5, \text{end})$

# Matrix operations

- Assign values to a sub-part of a matrix

- $A(2:4, 1:3) = [1\ 2\ 3; 4\ 5\ 6; 7\ 8\ 9];$

- both sides are 3x3 matrices

- A =

```
76 71 82 44 49
 1 2 3 38 45
 4 5 6 77 65
 7 8 9 80 71
17 10 3 19 75
```

- $A(2:4, 1:3) = 5;$

- the right side is a scalar

- A =

```
76 71 82 44 49
 5 5 5 38 45
 5 5 5 77 65
 5 5 5 80 71
17 10 3 19 75
```

# Matrix operations

- Matrix multiplication

- $C = A*B$

$$A = [1\ 3\ 5; 2\ 4\ 7] \quad (2 \times 3 \text{ matrix})$$

$$A = \begin{array}{ccc} 1 & 3 & 5 \\ 2 & 4 & 7 \end{array}$$

$$B = [-5\ 8\ 11; 3\ 9\ 21; 4\ 0\ 8] \quad (3 \times 3 \text{ matrix})$$

$$B = \begin{array}{ccc} -5 & 8 & 11 \\ 3 & 9 & 21 \\ 4 & 0 & 8 \end{array}$$

$$C = A*B$$

$$C = \begin{array}{ccc} 24 & 35 & 114 \\ 30 & 52 & 162 \end{array}$$

- Vector inner product

$$A = [5\ 3\ 2\ 6] \quad (1 \times 4 \text{ row vector (matrix)})$$

$$A = \begin{array}{cccc} 5 & 3 & 2 & 6 \end{array}$$

$$B = [-4\ 9\ 0\ 1]'$$

$$B = \quad (4 \times 1 \text{ col vector (matrix)})$$

$$B = \begin{array}{c} -4 \\ 9 \\ 0 \\ 1 \end{array}$$

$$A*B$$

$$\text{ans} = \begin{array}{c} 13 \end{array}$$

# Matrix operations

- Element-by-element product
  - $A.*B$
  - A and B must have the same size

A=

1 2  
3 4

B=

5 6  
7 8

$A.*B=$

5 12  
21 32

$A*B=$

19 22  
43 50

- Multiply a matrix by a scalar
    - $A*b$  or  $b*A$  (b is a scalar)
- $A*5=$
- 5 10  
15 20
- $A*b$ ,  $b*A$ ,  $A.*b$ ,  $b.*A$  are the same if b is a scalar.
- Q: How about  $A*A$ ,  $A^2$  and  $A.^2$ ?

# Control Statements

- If Statement

```
if x < 10
 disp(x); % only displays x when x < 10
end
```

- While Statement

```
p=1;
while p < 50
 p = 2 * p;
end
disp(p); % displays 64
```

- For Statement

```
for i=1:10
 disp(i);
end % displays 1 to 10
```

- Note1: They must be paired with 'end'
- Note2: Use "==" and "~=" for logical expression

# Functions

- build-in functions
  - can be called in different forms
  - e.g. max
    - $C = \max(A)$ 
      - returns the largest elements along different dimensions of an array
    - $C = \max(A,B)$ 
      - returns an array the same size as A and B with the largest elements taken from A or B
    - $[C,I] = \max(\dots)$ 
      - finds the indices of the maximum values of A, and returns them in output vector I
  - refer to the help if you are not sure about the usage
    - e.g. help max
  - what if you forget the name of the function?
    - google matlab + (the description of that function)
      - e.g. “matlab eigenvalues” or “matlab k-means”



# Functions

- Write your own function
  - e.g. calculates the mean and standard deviation of a vector

- stat.m:

```
function [mean,stdev] = stat(x)
n = length(x);
mean = sum(x)/n;
stdev = sqrt(sum((x-mean).^2/n));
```

- call the function in command window or in a script file

```
[mean stdev] = stat([12.7 45.4 98.9 26.6 53/1])
```

```
mean =
```

```
47.3200
```

```
stdev =
```

```
29.4085
```

- Note: The filename must be the same with the function name.
        - It is recommended that each function is written in separated \*.m files.

# Scripts vs. Functions

- Scripts
  - no input or output arguments
  - useful for automating series of MATLAB commands
    - computations that you have to perform repeatedly from the command line
  - analogy in C language: main function
- Functions
  - accepts input from and returns output to its caller
  - begins with a line containing the function key word
  - cannot be defined within a script file or at the MATLAB command line
  - analogy in C language: other utility functions called in main function

# Some useful command

- save
  - save workspace variables to file
  - they can be restored later by 'load' command
- who, whos
  - list variables in workspace
- clear
  - remove items from workspace, freeing up system memory
  - use it to remove unused variables when you are short of memory
- quit
  - quit Matlab
- Note: don't forget to save your source code (scripts/functions)



End!