# COMS 3101
# Programming Languages: MATLAB

## Lecture 2

Fall 2013

Instructor: Ilia Vovsha

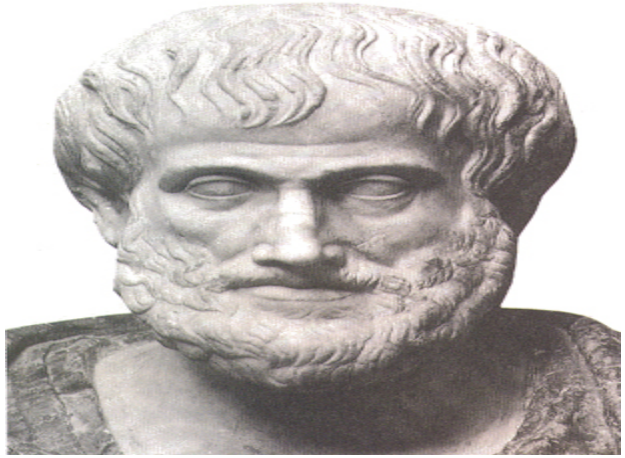
http://www.cs.columbia.edu/~vovsha/coms3101/matlab

# Lecture Outline

- Quick review of array manipulation
- Control flow
- Simple scripts and functions
- Loading and saving data
- Generating simple plots and curves
- Solve toy problems in class

# Useful Commands

- why
- fliplr()
- reshape(), repmat()



- Can have multiple statements on the same line:
  - x = 5; y= 7;
- 'help stats' for statistics toolbox

# Array Manipulation

- Special functions for arrays:
  - sum(), prod(), mean(), max(), min()
  - First argument we pass to each function is the array
  - What if array is a matrix, what does sum(X) mean?
- Useful routines:
  - sort(), find(), sortrows(), any()
  - Same issue as above, what does sort a matrix mean?

# Array Manipulation

- Special functions for arrays:
  - sum(), prod(), mean(), max(), min()
- Suppose M is a matrix, then:
  - TScol = sum(M); TScol is a row vector, each element is the column sum
  - TSrow = sum(M,2); TSrow is a column vector, each element is the row sum
  - TS = sum(sum(M)); TS is the total sum of all elements in the matrix
  - Can also obtain the location of the max/min value (if argument is a vector): [val, loc] = min(M);

# Array Manipulation

- Useful routines:
  - sort(), find(), sortrows(), any()
- Function sort() sorts elements in *ascending* (default) order. Can specify the order explicitly:
  - sort_list = sort(M, 'descend');
  - If M is a matrix, sorts each column independently
  - As above, can sort across columns: sort(M,2,'ascend')
  - If consistency between rows is required, use sortrows()

# Array Manipulation

- Function find() returns indices of elements which satisfy some condition:
  - find(M) Return indices of nonzero elements of M (default)
  - find(M == 1) Return indices of elements of M equal to one
  - find(M >= 2 & M < 6) Return indices that satisfy both conditions
- If M is a matrix, the returned index is a single value which is a bit annoying. Solution: use ind2sub(), sub2ind() to convert between indices
- [r,c] = ind2sub(size(M),index)

# Scripts

- MATLAB specific script files (M-files)
- Files have the extension .m (example: test.m)
- To run your script just type ‘test’ in the command line
- A script is a list of commands that should be executed
- To pass arguments to the script we will use functions (more on that later)

# Comments

- Comments make the code more manageable
- Can add a one-line comment by using the '%' symbol:
  - **% this is a comment**
- Anything that follows a % is ignored by the compiler (not executed)
- When you type help <func_name> you get the comments at the top of the <func_name> script
- Can also use block comments: %{ …. %}

# Debugging

- Debugging is the proper way of tracking program execution and fixing errors
- Type 'help debug' or use 'debug' from the workspace menu
- Simple alternative: pause() function
  - pause(): halt program until user strikes any key
  - pause(n): halt program for n seconds
- Approach: add output statements followed by pause() to follow progress

# Control Flow

- Without flow, the scripts have no life
- In general: a control flow statement begins with some keyword and ends with the keyword end
- General Form (if):

```
if (conditional statement)
        body
end
```

- Example:

```
if  x == 1
    y = 5;
end
```

# Control Flow – Cond. Statement

- General Form (if):

```
if (conditional statement)
    body
end
```

- Conditional statements can be with or without ()
- Typically use logical and relational operators in the condition:
  - | (OR), & (and), ~ (NOT)
- Good idea to use () when the statement is a long one
- Alternative: define a variable in the previous line
  - T = (x == 2 || y == 3) & x > 5;
  - if (T) …

# Control Flow

- General Form (if - else):

```
if (conditional statement)
    body1
else
    body2
end
```

- Example:

```
if  x == 1
    y = 5;
else
    y = 2;
end
```

# Control Flow

- General Form (if - elseif):

```
if (cond. statement1)
    body1
elseif (cond. statement 2)
    body2
else
    body3
end
```

- Example:

```
if  x == 1
    y = 5;
elseif x == 2
    y = 3;
else
    y = 2;
end
```

# Control Flow – Loops

- Used for repetition
- General Form (for):

```
for (iteration variable)
    body1
end
```

- Notice how the iteration variable is iterated over a row vector
- Example:

```
for  i = 1:5
    disp(i+1);
end
```

# Control Flow – Loops

- General Form (while):

```
while (cond. statement)
    body1
end
```

- Example:

```
x = 0;
while  x < 5
    x = x + 1;
    disp(x);
end
```

# Control Flow – Loops

- Example (continue):

```
y = 0;
for  x = 1:10
   if (x == 3);
     continue;
   end
   y = y +1;
end
```

- What is y = ?

- Example (break):

```
y = 0;
for  x = 1:10
   if (x == 3);
     break;
   end
   y = y +1;
end
```

- What is y = ?

# Control Flow – Caveat

- Loops are extremely inefficient in MATLAB, avoid them like the plague!
- Alternatives?
  - Built-in functions (find, sort, rand)
  - Pre-allocate memory (initialize array before entering loop)

# Exercise (In Class)

- Control flow & array manipulation
- Write a script that takes two (hard-coded) vectors {A,B}. For every element of A, print the element of B at the corresponding index (index = element of A). At the end of the procedure, print the sum and product of the printed elements

# Functions – Definition

- The keyword is function. There are multiple ways of defining functions:
  - function < name > ( <var>)
  - function < name > ( <var1>, <var2>)
  - function < out_var > = <name> ( <var>)
  - function [ <out_var1>, <out_var2>] = <name> ( <var1>,<var2>)
- The body of a function is just a set of statements (no different from a script)
- You can use the keyword end to terminate a function, but this is not required

# Functions – Definition

- It is possible to have a 'variable' number of (input or output) arguments:
  - function < name > ( <var>, varargin)
  - function [ <out_var1>, varargout] = <name> ( <var1>,<var2>)
- varargin, varargout are cell arrays (we discuss these later).
- You can check the number of declared input arguments for the function using nargin, nargout:
  - Example: nargin('mean')

# Functions – Example

- function wakeup(N)

```
if N > 10
   pause(N);
   disp('Alarm');
end
```

- function s = total(A, B)

```
if length(A) == length(B)
   s = A + B;
else
   s = A(1) + B(1);
end
```

# Functions – Subfunctions

- Typically one function for one file (file name = function name)
- Also possible to include multiple functions in one file:
  - Main function and sub-functions
  - Sub-functions are only “visible” to other functions in the same file
  - Useful when you have many minor and very specific routines
  - Type ‘help function’ to see an example

# Exercise (In Class)

- Functions & matrix manipulation
- Suppose M is a {0,1} matrix that represents a directed graph. If M(i,j) = 1, then there exists a directed edge from i to j. Write a function that takes M as input, and (for every edge) prints the word ‘edge’ followed by the vertices of the edge. The function should return the total number of edges in the graph in the variable T

# Input / Output – sprintf()

- sprintf() converts (formats) data into a string:
  - Example:  S = sprintf('myfile%d', 2);
  - The character following % specifies the format into which the supplied parameter is converted
  - Can specify different formats and supply multiple parameters
  - Example:  S = sprintf('myfile%d%s', 2, 'test');
- Common formats: %d (integer) %s (string) %g (float)
- Example:

1. prefix = 'home_dir/mydata';
2. file_num = 2;
3. filename = sprintf('%s%d.txt', prefix, file_num);

# I/O – Formats

- format <type> :
  - Controls output formats, and has no effect on computation
  - For better appearance of numbers with many decimal digits use ‘format short g’
  - To remove annoying spaces (line-feeds) use ‘format compact’
  - Possible to specify multiple formats as long as they don’t contradict each other

# I/O – User Input

- input() is a command that prompts the user to enter some input
- To record the input, you need to supply a variable
- The input stream is closed when the user hits enter
- By default, the expected input is a MATLAB expression
- Examples:
  - color = input('Enter your choice');
  - color = input('Enter your choice', 's');
  - color = input('Enter your choice\n', 's');

# I/O – MATLAB Data

- MATLAB has its own file extension ".mat"
- A .mat file is stored in binary format (unlike say text files it is pointless to view a .mat file)
- Storing data in a .mat file is convenient:
  - Can load and save data using built-in commands
  - Can load/save specific variables to/from the workspace
- Examples (load):
  - load file1;  % Automatically checks for file1.mat and loads all vars
  - v1 = load('file1.mat', X);  % Loads the variable X from file1.mat

# I/O – MATLAB Data

- Examples (save):
  - save file1.mat X;  % Save variable X to file1.mat
  - save(filename, 'X');  % Save X to a file whose name is stored in the variable 'filename'
  - save file1.mat X Y Z;  % Save X, Y, Z to file1.mat

# I/O – Text Files

- Commands: fopen(), fclose(), fgetl(), fseek(), fprintf()
- More complex than loading .mat files. First need to open a file fopen(), then read/write the data fgetl(), fprintf(), then close the file fclose()
- To open a file, we create a 'file identifier':
  - FID = fopen('myfile.text, 'r');  % 2nd argument specifies operation
  - 'r' (read) , 'w' (write), 'a' (append)
  - In case of failure, FID = -1
  - Always check whether file was opened

# I/O – Text Files

- To close a file is easy:
  - FID = fopen('myfile.text, 'r');
  - SOPEN = fclose(fid);  % returns -1 if failed to close
- Read a line from a file:
  - myline  = fgetl(FID);  % Read a single line
- Rewind file:
  - fseek(fid,0,-1);  % Often necessary to traverse a file multiple times
- Write to a file:
  - fprintf() writes formatted data to file
  - fprintf(fid, FORMAT, ARRAY);  % Format is a string

# I/O – Text Files

- Typically files have a specific format: numeric data separated by some delimiter (a character)
- MATLAB has built-in functions: dlmread(), dlmwrite()
- Examples:
  - M = dlmread('file1.txt', '\t')  % Read tab delimited data into M
  - Choose any character(s) as your delimiter. Be wary of spaces or dots as delimiters
  - M = dlmread('file1.txt', ';' , [R1 C1 R2 C2] )  % 3rd parameter is the range (zero-based) from which the data is read
  - dlmwrite('file1.txt', M, '\t')  % Write M to a tab-delimited file

# I/O – Images

- Image are just matrices. Color images have an additional dimension (B,G,R values). Grayscale images just have pixel values
- Built-in commands: imread(), imwrite()
- Examples:
  - M = imread('myphoto.jpg'); % Read an image
  - imwrite(M, 'myphoto,jpg'); % Infers the format from the ext.
  - imwrite(M, 'myphoto', 'JPEG');

# Exercise (In Class)

- Loading and saving a sequence of files.
- Suppose you have a directory ‘home/mydata’ which contains 100 files all named ‘dataset_#.mat’. Each files stores a matrix M. Write a function that takes an input parameter N, and opens each file (from 1 to N), sorts the matrix M in row order, and save the sorted matrix to a new file named ‘newdata_#.mat’