COMS 3101
Programming Languages: MATLAB

Lecture 3

Fall 2013

Instructor: Ilia Vovsha

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

Lecture Outline

- Loading and saving data
- More input/output
- Basic plotting, figures
- Data structures: cell arrays, structs, strings, handles
- Next lecture: formulating and solving problems
- Next lecture: advanced functionality, optimization

Useful Commands

- Block comments %{ … %}
- cumsum(), cumprod()
- mod(), rem()
- isempty(), isvector()
- For complicated functions reading the ‘help’ documentation is not sufficient (more details in the help browser ‘doc’)

HW #1

- Solutions are posted (average:)
- Part 2 will be discussed next week

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’

Basic Plotting

- The general function we will use is plot()
  - The function automatically generates a figure for you
- There are many options available:
  - Can specify the properties of the plot in detail
  - Can have multiple curves on the same plot
  - Can have multiple plots in one figure
  - More interesting plots are possible with bar(), pie()

Basic Plotting – Options

- Option 1: plot(X, Y)
  - Assuming X & Y are vectors, plots Y vs. X
  - A solid curve connecting the points
  - Y & X must have the same length
- Option 2: plot(Y)
  - Similar to option 1, but plots Y vs. 'index'
- Option 3: plot(X, Y, S)
  - 'S' is an optional parameter
  - A character string that controls plot appearance

Basic Plotting – Options

- Option 3: plot(X, Y, S)
  - Example: S = 'rs:' ➔ plots a red, dotted, line with a square at each data point on the curve
  - 'help plot' for all the string options
- Option 4: plot(X, Y, S, <param>, <val>)
  - Additional plot properties can be specified with different parameters and values
  - Example: plot(X,Y, S, 'LineWidth', 2) set the line-width to size 2
  - Type 'doc plot' for a list of these properties

Basic Plotting – Options

- More options:
  - Can set as many parameters as you wish (just call the function with more parameter-value pairs)
  - Can combine plots: plot(X1, Y1, X2, Y2)
  - Can 'duplicate' a plot: plot(X, Y, S1, X, Y, S2)
  - Plot combination is fine, but not the most elegant approach

Basic Plotting – Figures

- Problem:
  - plot(x,y); plot(x,z); replaces 1st plot with 2nd
- Solution: 'figure' command
  - plot(x,y); figure; plot(x,z);
  - figure(1); figure with handle #1
- Close figures:
  - Specific figure: close 1
  - All figures: close all
- MATLAB stores a handle to each figure

Basic Plotting – Figures

- Multiple plots:
  - plot(x,y); hold on; plot(x,z); hold off;
- Multiple plots, same figure:
  - subplot() command
  - figure(1); subplot(2,2,1);

Basic Plotting – Appearance

- Many options, can modify plots using the GUI
- Commands: title(), xlabel(), ylabel(), axis(), legend()
- Example:

```
1. figure(1);
2. title('test');
3. xlabel('quantity'); ylabel('price');
4. axis([1 5 1 10]);      % AXIS([XMIN XMAX YMIN YMAX])
5. grid on;               % Show grid lines
6. xlim([1 3]);           % Change x-axis limits
```

Basic Plotting – Figures

- You have created multiple figures. Which one is the 'current figure'?
- Answer: last figure you clicked on
- Better answer: use gcf() to get the handle
- Use figure(#) to make '#' the current handle
- Commands: gcf(), gca(), clf()
- Set 'object' properties using the set() command:
  - set(gca, 'XTick',[1 2 3]) % To set 'ticks' on the x-axis
- Get 'object' properties using the get() command

Saving & Loading Figures

- MATLAB has a special figure format: '.fig'
- Use openfig('name.fig') to open a saved figure
- Saving figures: use print command
  - **General Form**
  - **print** -d*format* filename
  - **Example**
  - **print –depsc 'figure.eps'**
- 'eps' is a format that stores your image in a vectorized way, which avoids quality loss after rescaling. It's particularly useful when used within LaTeX

Exercise (In Class)

- Simple plot
- Write a function that has one input parameter M, a matrix with 3 columns. Columns {1,2} of M are the {x,y} coordinates of points in the plane. Column 3 is the class to which the point belongs. The values of column 3 are one of two unknown integers
- Your function should plot all points, points in one class should be dots in red, points in the 2nd class should be squares in green. Adjust your axes properly so that the furthest points are not on the edge

Data Structures – Structs

- When a composite data structure is required, use a 'struct' (structure array)
  - Multiple fields, different data structures for each field
  - Similar in form to C++ classes
  - Use 'dot' to access fields
- Initialization:
  - S = struct([]) % Empty struct, no fields
  - S = struct ('f1', v1, 'f2', v2) % Struct with two fields
  - S.field1 = 2.5; % Create directly

Data Structures – Structs

- Functionality:
  - isstruct(S); % Check if S is a struct
  - S = setfield(S, 'field1', 5); % Set a field to a value
  - isfield(S, 'field1'); % Check if field exists
- Example:

1. S = struct('vec', [1,2,3], 'mat', rand(3));
2. my_field = 'mat';
3. isfield(S, my_field);

Structs – Note

- When multiple variables are saved in a .mat file, and then loaded into a single variable, they are saved as fields of a struct
- Example:

```
1. save file1.mat X Y Z;
2. S = load('file1');
3. isstruct(S);          % Struct with 3 fields: S.X, S.Y, S.Z
```

Data Structures – Cell Arrays

- Cell array is an 'array of matrices'.
  - Each element of a cell array can be a scalar/vector/matrix.
  - Why is it useful? Recall the 1st exercise (loading a sequence of files)
- Initialization:
  - C = cell(N) % N by N array of empty matrices
  - C = cell(N,M) % N by M array of empty matrices
  - Use brackets {} to access elements
  - Rules for regular arrays apply
  - C = {[1] [1,2,3]; rand(3), rand(3)}; % Create directly

Cell to Struct Conversion

- Commands: cell2struct(), struct2cell()
  - S = cell2struct(C, my_fields, my_dim)
  - 'my_fields' is a cell array of strings
- Example:

```
1. my_fd = {'num', 'name', 'nation'};
2. my_arr = {8, 'Iniesta', 'Spain'};
3. my_st = cell2struct(my_arr, my_fd, 2);
% You are 'folding' the dimension, size must match
```

Strings – Functions

- MATLAB is not recommended as a tool for manipulating strings. However, the functionality is available
- Check string:
  - isletter(str), isspace(str)
- Convert string:
  - lower(str), upper(str)
  - str2num(str), num2str(num)

Strings – Functions

- Operations on strings:
  - strtok(str, delim), strcmp(str1,str2), strfind(str1,pattern)
- Regular expressions (help regexp): concise and flexible means for matching strings
- Example:

1. str1 = ‘one!, no two’;
2. [bef_delim, aft_delim] = strtok(str1, ‘!’);
3. idx = strfind(str1, ‘n’);
4. res1 = strcmp(‘hi’, ‘HI’);
5. res2 = strcmp(‘hi’, lower(‘HI’);