MATLAB Tutorial – Programming in MatLab

Chienmin Chuang
School of Mathematics, University of Birmingham

May 24, 2011

INLINE FUNCTION

A handy way to define a simple function is using
FunctionName = inline('Expression','var1','var2',...,'varN')
which reads VarN,...,Var1 as inputs and returns an output by Expression.
Example:

FUNCTION HANDLE (@)

- Another alternative is using function handle of the form **FunctionName = @(var1,var2,...,varN) Expression**. Furthermore, we can use defined function handles to define a new function.
- An advantage of function handle is it can be used as an input of another function.
- A new function handles can recognize the functions defined before it (by inline or @) but a new inline function can not do so.

EXAMPLES:

FUNCTIONS FOR SINGLE-VALUE FUNCTION

- We can define a single-value function, which may be one-to-one or many-to-one, by either **inline** or **@** in MatLab.
- Once a single function is defined, we can use its **function name of inline function** or **function handle** as an input for the commands introduced later to visualize, find roots and integrate the function.
- Note that for the **built-in functions** and **m-functions** (introduced later), we need to add @ in front of function names to form function handles.

FUNCTIONS FOR SINGLE-VALUE FUNCTION

- **feval(f, var1,var2,...,varN)** provides another way to evaluate a single-value function.

FUNCTIONS FOR SINGLE-VALUE FUNCTION

- **feval(f, var1,var2,...,varN)** provides another way to evaluate a single-value function.
- **fplot(f,[a,b])** enable us visualize a **one-to-one** function f in the interval [a,b].

FUNCTIONS FOR SINGLE-VALUE FUNCTION

- **feval(f, var1,var2,...,varN)** provides another way to evaluate a single-value function.
- **fplot(f,[a,b])** enable us visualize a **one-to-one** function f in the interval [a,b].
- **fzero(f,x0)** tries to find a root of a **one-to-one** function f near x=x0 and returns a point where fun changes sign, or NaN if the root search fails.

FUNCTIONS FOR SINGLE-VALUE FUNCTION

- **feval(f, var1,var2,...,varN)** provides another way to evaluate a single-value function.
- **fplot(f,[a,b])** enable us visualize a **one-to-one** function f in the interval [a,b].
- **fzero(f,x0)** tries to find a root of a **one-to-one** function f near x=x0 and returns a point where fun changes sign, or NaN if the root search fails.
- **quad(f,l,r)** finds the approximate value of the integral of a **one-to-one** function f on the interval [l,r] (using Simpson's method).

FUNCTIONS FOR SINGLE-VALUE FUNCTION

- **feval(f, var1,var2,...,varN)** provides another way to evaluate a single-value function.
- **fplot(f,[a,b])** enable us visualize a **one-to-one** function f in the interval [a,b].
- **fzero(f,x0)** tries to find a root of a **one-to-one** function f near x=x0 and returns a point where fun changes sign, or NaN if the root search fails.
- **quad(f,l,r)** finds the approximate value of the integral of a **one-to-one** function f on the interval [l,r] (using Simpson's method).
- The counterparts of quad(f,l,r) for two and three variables are **dblquad(f,l1,r1,l2,r2)** and **triplequad(f,l1,r1,l2,r2,l3,r3)**.

EXAMPLES:

EXAMPLES:

M-FILE

The m-files are files that contain MatLab codes and have the suffix ".m". They include two sorts: the *script* files and the *function* files. Script files do not have an input and/or output. Basically, a script is a collection of several procedures. In contrast, the function files may take input arguments or return output arguments. In other words, it is another alternative to define a function, generally a more complex function or algorithm.

To generate a new m-file, go to File>New>M-File. To edit an existing one, go to File>Open and then specify the path.

M-SCRIPT

- An m-script is a collection of the statements and/or procedures that we would like to execute in order and saved as an m-file. This is extremely useful if we would like to implement computations or edit procedures repeatedly.
- In an m-file, including m-script and m-function, % refers to comments and all statement in the same line after % is to be ignored when executing the m-file.

M-SCRIPT

Examples:

M-FUNCTION

```
function [output1, ..., outputM]=FunctionName(input1, ..., inputN)
% Comments for help file of the function
            Mainbody;
end
```

- A function defined by m-file is of the form above where input1, ..., inputN are input names; output1, ..., outputM are output names; FunctionName is the function name.
- Note that the **function** is necessary to begin a m-function. The **end** is not necessary; however, it is better to add in the end to make the scope of an m-function clear.

M-FUNCTION

- The content following % in the same line will not be executed when an m-function is running. The first comments before or after the line of **function** are regarded as help file by default.
- When saving an m-function, the file name and FunctionName should be consistent.
- If an m-file contains several functions, the first one is considered the primary function and the others are its subfunctions. A subfunction can be called by other functions in the same m-file but not by a function in different m-file. However, primary functions of different m-files in the same folder can call each other.

EXAMPLE 1

Edit the PascalTriangle.m as below.

Examples:

EXAMPLE 2

Edit the bisection.m as below.

EXAMPLE 2

EXAMPLE 2

Examples:

VECTORIZATION AND TIMER

- Since MatLab is powerful in vector and matrix manipulation, code vectorization can speed up program running significantly.
- Commands **tic** and **toc** measure performance using stopwatch timer.
- Command **cputime** shows elapsed cpu time.

Examples:

Workspace

- **Workspaces** are the memory parts used to store variables by MatLab.
- The **base workspace** is the used workspace when entering commands at the Matlab prompt in the command window.
- An m-script uses the base workspace.
  - It can access any existing variables in the base workspace.
  - Variables defined in an m-script will be retained in the base workspace when the m-script finishes.
- However, m-functions do not use the base workspace; instead, each m-function owns individual workspace to store the variables defined in the m-function. Such workspaces are temporary and will vanish when the corresponding m-functions finish.

whos and clear

- The command **whos** can list all variables in the current workspace.
-
  - **clear** *var* removes the variable *var* from the current workspace.
  - **clear** removes all variables from the current workspace.
  - **clear all** remove all variables from all workspaces.

VARIABLE TYPES

There are three types of variables: local, global and persistent variables.

- If a variable is used without declaration, it is a **local variable** by default. A local variable of an m-function is unaccessible to other functions.
- A global variable is accessible to all functions and m-files who declare the variable before using. To declare a global variable, use **global** *VarName*.
  - Once declared, the initialized value is empty.
  - Conventionally, all letters of a global variable's name are in upper case to distinguish.
  - Global variables can avoid passing the value of a variable between functions.
  - However, use of global variables may cause some errors which are difficult to track down. It is suggested not to use a global variable if unnecessary.

Variable Types

- A **persistent** variable is a special *local* variable.
  - A persistent variable must be declared by **persistent** *VarName*.
  - Once declared, the initialized value is empty.
  - Different from a normal local variable, the value of a persistent variable can retain even its m-function finish. In this manner, the value can be iteratively memorized and used between repeated calls to the m-function.

EXAMPLE 1:

EXAMPLE 2:

EXAMPLE 3:

Files with suffix **.mat** is the standard format in MatLab. Besides, it can also read data with the suffixes **.dat**, **.txt** and **.xls**.

input:

- **UserEntry = input('statement')** displays statement as a prompt on the screen, waits for input from the keyboard, and returns the numerical value entered in UserEntry.
- **UserEntry = input('statement', 's')** regards the entry as a string instead of numerical value.

Examples:

save:

- **save(**'*FileName*', '*var1*', '*var2*', ...**)** or **save** *FileName var1 var2* ... save the variables — var1, var2, ... which are present in the current workspace into the file *FileName.mat*.
- **save** *FileName.dat -ascii (-double) var1 var2 ...* save the data in 8-digit (16-digit) ASCII format into the file *FileName.dat*.
- **save** *FileName.txt -ascii (-double) var1 var2 ...* save the data in 8-digit (16-digit) ASCII format into the file *FileName.txt*.
- When saving in ASCII format, all data will be converted to numbers and stacked up without separation.

load AND **whos -file**

- **load:**
  - **load** *FileName.mat* and **load** *FileName* load all variables from *FileName.mat*.
  - **load** *FileName.mat var1 var2...* only loads the specified variables *var1 var2....*
  - **load** *FileName.dat* loads the data from *FileName.dat*.
  - **load** *FileName.txt* loads the data from *FileName.txt*.
- **whos -file FileName** lists the information of all the variables in FileName.mat.

Examples:

More on -txt and -xls files:

- **textread** reads data from a txt-file with different columns and write to multiple outputs. Go to help file for more details and examples.
- **xlsread** and **xlswrite** can read and write excel files with suffix .xls. Go to help file for more details and examples.
- It is suggested to save and manipulate data in -mat format when using MatLab.

ND ARRAY

1. In addition to two-dimensional matrices, MatLab support data storage in a higher-dimensional sense. Such data type is called a **multi-dimensional array** or an **ND array**.
2. By default, the first dimension are (vertical) **columns** and the second are **rows**. The third and fourth are referred to as **pages** and **boxes**.
3. **cat(d, A1, A2, An)** combine the arrays A1, A2, ...An along the dth. dimension.

ND Array

Examples:

Cell Array

- In addition to ND array for numerical data, MatLab support high-dimensional storage for various data type. Such storage is called a **cell array**.
- **cell(n1,n2,...nd)** generates an $n1 \times n2 \times \cdots nd$ cell array with empty entries.
- Each entry of a cell array is called a **cell** which can be used to store any recognized data type by MatLab, including another cell array.
- To assign a content to a specific entry, use $\{\}$ rather than $()$ to include the desired coordinate. $\{i,j\}$ and (i,j) are called **cell indexing** and **content indexing** respectively.
- When reading entries, content indexing can work as well. However, cell indexing shows the exact formats of contents while content indexing shows them in the format of cells.

CELL ARRAY

- **cellplot(A)** enables us to see the data types of cells in the cell array A.
- **celldisp(A)** displays the exact content of all cells.

Examples:

Structure Array

- **Structure array** is a compound data type which contains several **fields**. Practically it is like a multi-functional Swiss knife.
- Each field can be a variable, a cell or even a function.
- There are two ways to define a structure:
  - **using assignment statements:**
    Simply make a name for a structure variable followed by a dot and a field name. To add one more structure variable, we can the content indexing after the structure name.
  - **using the struct function:**
    struct('field1',content1,'field2',content2, ...) generates one or more structure variables with field1 1, field2,.. equal to content1, content2, ... individually.

EXAMPLES:

EXAMPLES:

EXAMPLES:

STRUCTURE ARRAY

- Once a structure type, say *StrTyp*, is defined, we can use **rmfield(***StrTyp, 'SomeField'***)** to delete the existing field *'SomeField'*.
- To add a new field, simply create a field and assign a value to it. New fields without assigned contents is empty by default.
- **fieldnames(***StrTyp***)** show the names of all fields in *StrTyp*.

Examples:

A GLIMPSE AT CLASS

- **Class** is a more general, flexible and complicated data type than structure.
- Any object in reality, such as a car, a bird or a person, can be briefly modelled as a class in a virtual computer world.
- Such a way to define a data type like an object is called **object-oriented programming (OOP)**.
- Wiki: *OOP is a programming paradigm using "objects" data structures consisting of data fields and methods together with their interactions to design applications and computer programs. Programming techniques may include features such as data* **abstraction, encapsulation, messaging, modularity, polymorphism, and inheritance**.

A GLIMPSE AT CLASS

- A class is the realization of OOP which has vital static attributes and dynamic actions which are called **properties** and **methods** in MatLab.
- A **constructor** is a special method used to generate a variable in the form of the desired class.
- Properties, methods and a constructor are the three main parts to define a class.
- Go to help and search class for technical details.

Examples:

HOW CAN YOU BENEFIT FROM MATLAB?

- Interpolation and extrapolation for missing data
- Integration and differentiation for complex functions
- Solve a system of equations
- ANOVA and other Statistics analysis
- Regression and Time Series Analysis
- Solve ODE, BVP and PDE
- Discrete Fourier Transform
- Find the solution to an optimization problem (in Optimization toolbox)

USEFUL LINKS

- **MatLab Central**
  http://www.mathworks.com/matlabcentral/
- MatLab for Chemical Engineering
  http://faculty.kfupm.edu.sa/CHE/aljuhani/New_Folder/matlab_che.pdf
- MatLab for Chemistry
  http://www.mathworks.de/matlabcentral/fileexchange/%3Fdate%3Dsubmitted%26page%3D2%26term%3Dtag%253A%2522chemistry%2522
- MatLab for Computer Science
  http://www.mathworks.com/support/books/index_by_categorytitle.html?category=18
- MatLab for Physics and Engineering
  http://physics.gac.edu/~huber/matlab/
- MATLAB for metallurgy and material science
  http://www.engineering.com/Ask/tabid/3449/qactid/-1/qaqid/56/Default.aspx