

Introduction to MATLAB programming: Fundamentals

Shan He

School for Computational Science
University of Birmingham

Module 06-27818 and 27819: Advanced Aspects of
Nature-Inspired Search and Optimisation (Ext)

Outline of Topics

Why MATLAB?

Matrix operations in MATLAB

Programming in MATLAB

Plotting in MATLAB

What is it?

- ▶ A humble origin: an interactive matrix calculator for students.
- ▶ Now more than 1 million users
- ▶ Used in engineering, science, and economics, etc.
- ▶ "*A high-level technical computing language and interactive environment for algorithm development, data visualization, data analysis, and numerical computation.*" – Matworks

Why Matlab?

- ▶ Faster algorithm development than other languages
- ▶ Concise matrix notation and great matrix manipulation.
- ▶ Easy visualisation, debugging and code optimisation.
- ▶ Many core mathematical, engineering, and scientific functions and great community support.

Cons of Matalb?

- ▶ Commercial software, expensive
- ▶ Solution: GNU Octave, an opensource alternative to Matlab
- ▶ Download it from [here](#)

The most useful command in MATLAB

```
>> help
```

The heart of MATLAB: matrices

- ▶ MATLAB stands for “matrix laboratory”.
- ▶ The basic data type is matrix (including vectors).
- ▶ Matrix operations: create, access, modify and manipulate matrices
- ▶ Matrix operation is very fast in MATLAB – try to avoid for-loops

Creating Matrices

- ▶ Create an empty matrix:

```
>> A = []
```

- ▶ Enter data directly:

```
>> A = [1 1; 2 3; 4 5]
```

- ▶ If you know the pattern of your matrix:

```
>> A = [1:2:100]
```

- ▶ A lot of functions to create specific matrices: `zeros()`, `ones()`, `rand()`, `eye()`
- ▶ One particular useful one for 3D plot is `meshgrid`, for example, to create a 5-by-5 grids of numbers `[E,F] = meshgrid(1:5)`
- ▶ Let's try code examples (Matrix creation)

Transposing and concatenating matrices

- ▶ To transpose matrix A , use the transpose operator ' $'$ '

```
>> B = A'
```
- ▶ To concatenate matrix, enclose them inside of square brackets.

```
>> C = [A; 6 7]
```

Indexing

- ▶ To extract individual entries from a matrix, use indices inside round brackets:

```
>> A(1,2)
```

- ▶ Use the ':' operator to extract all entries along a certain dimension:

```
>> A(1,:) 
```

- ▶ Use 'end' statement to get the last index of a dimension:

```
>> A(end,1)
```

- ▶ Use logical indexing to find specific entries:

```
>> A(A>2)
```

- ▶ We can also use `find()` function to find specific entries:

```
>> A(find(A>2))
```

- ▶ Let's try code examples (Matrix operations)

Assignment and deletion

- ▶ To change entries in the matrix, using indexing to specify the entries and assign new values:

```
>> A(1,2) = 100
```

```
>> A(:,1:3:end) = 100
```

- ▶ To delete entries, assign '[]' :

```
>> A(1,:) = []
```

- ▶ For a matrix, you can only delete column(s) or row(s)
- ▶ For array (1D matrix), you can delete any entries.

```
>> A(3) = []
```

- ▶ Let's try code examples (Matrix assignment and deletion)

Sorting

- ▶ We can also sort the elements in a matrix, by default in an ascending order

- ▶ For example, to sort an array:

```
>> A = [9 0 -7 5 3 8 -10 4 2]    >> B = sort(A)
```

- ▶ For example, to sort a matrix by row, i.e., to sort each of its rows in ascending order:

```
>> A = [3 6 5; 7 -2 4; 1 0 -9]
```

```
>> B = sort(A,2)
```

- ▶ You can also get the arrangement of the elements of A into B along the sorted dimension, i.e., the index of the sorted element of the previous unsorted matrix:

```
>> A = [9 0 -7 5 3 8 -10 4 2]
```

```
>> [B, idx] = sort(A)
```

- ▶ Let's try code examples (Matrix assignment and deletion)

Matrix manipulations

- ▶ We can perform matrix addition, subtraction, multiplication, exponentiation, etc.
- ▶ For example, matrix multiplication of an m -by- n matrix and an n -by- p matrix yielding an m -by- p matrix:

```
>> C = A*B
```
- ▶ We can do element-wise matrix arithmetic by using `.'` precede the arithmetic operator

```
>> A = rand(3,3)
```

```
>> B = rand(3,3)
```

```
>> D = A.*B
```
- ▶ For element-wise arithmetic operation, both matrices must be the same size.
- ▶ Let's try code examples (Matrix manipulation)

Flow of Control

MATLAB only has the following statements:

- ▶ `if, else, elseif`
- ▶ `switch statements`
- ▶ `for loops`
- ▶ `while loops`
- ▶ `try/catch statements`

Scripts and functions

- ▶ We can use MATLAB editor to edit/save/load/exceute your programs.
- ▶ Two types of MATLAB programs: scripts and functions.
- ▶ A script is a collection of Matlab commands.
- ▶ The commands in the script are executed exactly as at the command prompt.
- ▶ However, scripts:
 - ▶ No lexical scoping, that is, the variables in scripts are global. We cannot reuse the same variable name multiple times.
 - ▶ Cannot be parameterize to be called multiple times with different inputs.
 - ▶ Difficult to read and understand.
 - ▶ Slow.

Creating functions

- ▶ Open a new file in MATLAB editor, or type: `edit filename.m` at the command prompt.
- ▶ In you m file, begin by creating the function header:
`function [output1 ,output2, output3...] = myfunction(input1, input2...)`
- ▶ We can use the inputs as local variables.
- ▶ All variables are local in a function.
- ▶ We must assign values to each of the outputs before the function terminates.
- ▶ Although optional, it is better to end the function with the “end” keyword
- ▶ Let's try code examples (MATLAB function example).

Functions: other issues

- ▶ The functions must be located in the directories of the command path, or under the current working directory.
- ▶ Use “%” for comments
- ▶ We can have multiple functions in a .m file
- ▶ We can pass functions as inputs to other functions by creating handle to the function and then pass the handle as a variable.

```
>> x = fminbnd(@humps, 0.3, 1)
```

- ▶ We can create anonymous functions without having to store your function to a file each time:

```
>> fhandle = @(arglist) expr
```

```
>> sqr = @(x) x.*x;
```

1D/2D plots

- ▶ To plot 1D and 2D data, we use `plot(y)`.
- ▶ If `y` is a vector: a piecewise linear graph of the elements of `y` versus the index of the elements of `y`
- ▶ If `y` is a matrix: `plot(y)` will automatically cycle through a predefined (but customizable) list of colors to allow discrimination among sets of data
- ▶ If we specify two vectors as arguments, `plot(x,y)` produces a graph of `y` versus `x`.
- ▶ Use `hold on` command to superimpose all the plots onto the same figure. Use `hold off` to disable.

3D plots

- ▶ To plot 3D lines: `plot3(X1,Y1,Z1)`
- ▶ To plot 3D shaded surface: `surf(X,Y,Z)`
- ▶ You can also specified shading property: shading flat/faceted/interp
- ▶ To plot 3D mesh: `mesh(X,Y,Z)`
- ▶ To plot contour lines: `contourf(X,Y,Z)`

Customise plots

- ▶ Multiple subfigures: `subplot(nr,nc,i)`
- ▶ Title: `title('you title')`
- ▶ Axis labels: `xlabel('x');` `ylabel('y');` `zlabel('z')`
- ▶ We need to get handle of a figure
 - ▶ Handle of the current figure: `gcf()`
 - ▶ Handle of the current set of axes: `gca()`
- ▶ To access specific properties: `get(handle,'property')`
- ▶ To change specific properties: `set(handle,'property1', value1, 'property2', value2, ...)`
- ▶ Let's try code examples (Plot example).

Now let's complete the exercises