Introduction to MATLAB programming: Fundamentals

Shan He

School for Computational Science
University of Birmingham

Module 06-23836: Computational Modelling with MATLAB
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?

Key features

- "High-level language for technical computing."
- "Mathematical functions for linear algebra, statistics, Fourier analysis, filtering, optimization, and numerical integration."
- "2-D and 3-D graphics functions for visualizing data."
- "Interactive tools for iterative exploration, design, and problem solving."
Why MATLAB?

- Concise matrix notation and great matrix manipulation.
- Easy visualisation.
- Many useful “toolboxes” and great community support.
- Great tool for computational modelling and data analysis.
A tour to MATLAB
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()`

- Click here to download an example.
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:
  ```matlab
  >> A(1,2)
  ```

- Use the ‘:’ operator to extract all entries along a certain dimension:
  ```matlab
  >> A(1,:)
  ```

- Use ‘end’ statement to get the last index of a dimension:
  ```matlab
  >> A(end,1)
  ```

- Use logical indexing to find specific entries:
  ```matlab
  >> A(A>2)
  ```

- We can also use `find()` function to find specific entries:
  ```matlab
  >> A(find(A>2))
  ```
Assignment and deletion

- To change entries in the matrix, using indexing to specify the entries and assign new values:
  ```matlab
  >> A(1,2) = 100
  >> C(:, 1:2:end) = 500
  ```

- To delete entries, assign ‘[]’:
  ```matlab
  >> A(1,:) = []
  ```

- For a matrix, you can only delete column(s) or row(s)

- For array (1D matrix), you can delete any entries.
  ```matlab
  >> B(3) = []
  ```
Reshaping and Replication

- We can reshape an array of size \( m \)-by-\( n \) to size \( p \)-by-\( q \) by \texttt{reshape()}:
  ```matlab
  >> A = rand(2,3)
  >> reshape(A, 1, 6)
  ```

- We can also tile an array \( m \)-by-\( n \) times using \texttt{repmat()} function:
  ```matlab
  >> A = rand(2,3)
  >> remat(A, 1, 6)
  ```
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:
  ```matlab
  >> A = rand(3,3)
  >> B = rand(3,3)
  >> C = A*B
  ```
- We can do element-wise matrix arithmetic by using `.'` precede the arithmetic operator
  ```matlab
  >> D = A.*B
  ```
- For element-wise arithmetic operation, both matrices must be the same size.
Matrix manipulations

▶ We can also do matrix division, but be aware of **Matrix Right Divide /** and **Matrix Left Divide \**

```
>> C = A*B
>> Alsq = C/B
>> Blsq = A \ C
```

▶ Note that $C/B$ is equivalent to $(B' \ C')'$ and $A \ C$ is equivalent to $\text{inv}(A)*C$
Sparse Matrices

- If we have a large matrix but containing many zeros, it is better to convert it to a sparse matrix:
  ```matlab
  >> A = zeros(1000,1000);
  >> A = sparse(A);
  ```

- We can use sparse matrices just like ordinary matrices but slower:
  ```matlab
  >> A(10,10)=1;
  >> B = rand(1000,1000);
  >> C=A*B;
  ```

- We can always convert sparse matrices to full matrices.
  ```matlab
  >> A = full(A);
  ```
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/execute 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 your m file, begin by creating the function header:
  ```matlab
  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 see an example (download from here).
Functions: other issues

- The functions must 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.
  For example:
  ```
  >> 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)`
- Download my example at here
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 axis: gca()
- To access specific properties: get(handle, 'property')
- To change specific properties: set(handle, 'property1', value1, 'property2', value2, ...)

Introduction to MATLAB programming

Plotting in MATLAB