

Introduction to MATLAB programming: Advanced topics

Shan He

School for Computational Science
University of Birmingham

Module 06-32235: Advanced Aspects of Nature-Inspired Search
and Optimisation

Outline of Topics

Character Arrays (Strings)

- ▶ Character Arrays are character matrices.

```
A = 'This is a String.'
```

```
B = A(1:5)
```

```
C = [A ; A]
```

- ▶ `char()` and `abs()`: convert from integers to the ascii equivalents and vice versa.

```
A = char(48)
```

```
B = abs('String')
```

Character Arrays (Strings)

- ▶ `num2str()` and `mat2str()`: generate string representations of numeric matrices.
- ▶ `str2num()`: parse a number from a string
- ▶ `sprintf()` and `fprintf()`: format strings.
- ▶ `strcmp()` and `strcmpi()`: Compare strings (case sensitive/insensitive)
- ▶ `strfind()`: find the occurrences of one substring inside another
- ▶ Here is an example.

Cell arrays

- ▶ A more general and power data structure
- ▶ The same cell array can hold elements of different types:
 - ▶ numeric matrices of different sizes;
 - ▶ character arrays of different sizes
 - ▶ other cells;
 - ▶ structs;
 - ▶ objects.

```
B = { [1,2,3] , 'hello' , 1; [3;5] , 'yes' , 'no' }
```

- ▶ To create a new cell array:

```
A = cell(2,4)
```

Indexing cell arrays

- ▶ One important concept: A n -by- m cell array is made up of $n \times m$, 1-by-1 cell arrays,
- ▶ Two ways to index into and assign into a cell array:
 - ▶ `()` brackets: access or assign **cells**;
`Cell = B(1,2)`
 - ▶ `{}` brackets: access or assign **the data within those cells**.
`String = B{1,2}`
- ▶ We must be very careful what kind of brackets we use. Which one is better?

```
B(1,2) = {'test'}
```

```
B{1,2} = 'test'
```

```
B{1,2} = {'test'}
```

Operating cell arrays

- ▶ We can operate cell arrays just as matrices, e.g., transpose, reshape, replicate, concatenate, and delete.
- ▶ `cellfun()`: to apply a function to the data inside every cell:

```
A = {'A', 'test', 'message', 'Which'}  
[nrows, ncols] = cellfun(@size, A)
```
- ▶ We can convert between matrices and cell arrays using `num2cell()`, `mat2cell()`, and `cell2mat()`.

Set Operations

- ▶ Matrices and cell arrays can be operated as sets or multisets.
- ▶ Set operation functions: `union()`, `intersect()`, `setdiff()`, `setxor()`, and `ismember()`.
- ▶ `unique()`: extract the unique elements of a cell array or matrix.

```
uniqueNums = unique([1,2,1,1,2,3,4,4,5,3,2,1])
```

```
uniqueNames = unique({'Bob','Fred','Bob','Ed'})
```


Putting all together: a worked example

Let's analyse William Shakespeare's *Hamlet*:

- ▶ How many unique words?
- ▶ What are the most frequent words?

Structs

- ▶ Organize data and access it by name – use it as a simple database.
- ▶ Similar to cell arrays, structs store elements of different types.
- ▶ We can also add/remove fields:

```
S = struct('name','shan','matrix',[1 1; 2 2])
```

```
S.name
```

```
S.newField = 'foo'
```

```
S = rmfield(S,'matrix')
```

- ▶ Structs can be stored in cell arrays and matrices.
- ▶ We can access fields by strings, useful in runtime:

```
fieldname = 'name'
```

```
distance = S.(fieldname)
```

Struct arrays

- ▶ Struct array: an array of structs all having the same fieldnames

```
S = struct('name',{},'Salary',{})
```

```
S(1) = struct('name','Shan','Salary',100)
```

```
S(2) = struct('name','Volka','Salary',300)
```

- ▶ Effectively can be seen as a table:
 - ▶ To access a record of fields (row): `S(1)`
 - ▶ To access a column of fields: `S.name`
 - ▶ To access a field: `S(1).name`
- ▶ We can convert between cell arrays and struct arrays: `cell2struct()` and `struct2cell()`

Hash tables: Containers.map

- ▶ Hash tables map keys to values by hash function. Two parts:
 - ▶ **Key**: a string or numeric scalar
 - ▶ **Value**: anything

```
k = {'UK', 'Italy', 'China'}
```

```
v = {'London', 'Rome', 'Beijing'}
```

```
CapitalsMap = containers.Map(k, v)
```

- ▶ To list all keys and values by keys() and values()
- ▶ To add new entry:

```
CapitalsMap('USA') = 'Washington D.C.'
```

- ▶ To retrieve values:

```
CapitalsMap('USA')
```

```
values(CapitalsMap, {'USA', 'Italy'})
```

Debugging

- ▶ `keyboard()`: add the it anywhere in your m-file to stop at that point. Type return to continue
- ▶ Use break points: step one line at a time, continue on until the next break point, or exit debug mode
- ▶ `dbstop`: Set breakpoints for debugging:
 - ▶ `dbstop if error`: stops execution at the first run-time error that occurs outside a try-catch block.
 - ▶ `dbstop if naninf`: stops if there is an infinite value (Inf) or a value that is not a number (NaN)
 - ▶ `dbstop if EXPRESSION`: stops if EXPRESSION evaluates to true

Object Oriented Programming (OOP) in MATLAB

- ▶ Q1: What is OOP?
- ▶ A1: Design of programmes using "objects".
- ▶ Q2: What is objects?
- ▶ A2: Data structures that encapsulate data fields and methods that interact with each other via the object's interface.
- ▶ Q3: When to use OOP?
- ▶ A3: When *"the number of functions becomes large, designing and managing the data passed to functions becomes difficult and error prone"*.

OOP in MATLAB: an example

- ▶ Before seeing the example, some important concepts:
 - ▶ **Class:** A kind of prototype, or specification for the construction of a objects of a certain class.
 - ▶ **Objects:** Instances of a class.
 - ▶ **Properties:** Fields that store data.
 - ▶ **Methods:** The operations we want to perform on the data.
- ▶ You can download my OOP example at [here](#).
- ▶ You can learn more from MathWorks' [Introduction to Object-Oriented Programming in MATLAB](#)

Building MATLAB Graphical User Interfaces (GUIs)

- ▶ MATLAB GUI: a figure window providing pictorial interface to a program.
- ▶ Two ways of building GUIs:
 - ▶ GUIDE (GUI Development Environment).
 - ▶ Create m-files that generate GUIs as functions or scripts
- ▶ Due to time constrains, I will show one simple example and list some useful links:
[MATLAB GUI tutorial](#)
[Youtube tutorial](#)

Code optimisation: where to optimise

- ▶ Generally MATLAB is slower than C/JAVA, but it is not always the case.
- ▶ **Optimise bottlenecks**
- ▶ To identify bottlenecks we need to profile the code: `profile on/off`
- ▶ To view the profile: `profile viewer`
- ▶ Timing your code: use `tic` before your code and `toc` afterwards

Code optimisation: techniques

- ▶ Pre-allocate memory:
- ▶ Vectorisation: making your code work on array-structured data in parallel, rather than using for-loops.
Visit MathWorks' [Code Vectorization Guide](#)
- ▶ Use built-in functions.
- ▶ Some useful functions for vectorisation:
- ▶ Finally, if you cannot vectorise your code, write it in C/C++ and call them using MEX ([See Matworks' tutorial here](#))