Introduction to MATLAB®

Spring 2019

What is MATLAB®?

MATLAB®

- Short for Matrix Laboratory $\longrightarrow$ *matrix data structures are at the heart of programming in MATLAB*
- We will consider **arrays** in *n*-dimensions where

$$1\text{D} \Longrightarrow \text{vectors and } 2\text{D} \Longrightarrow \text{matrices, e.t.c.}$$

- In addition MATLAB is also a **programming language**
  - It is an interpreted language (i.e no need to compile)
  - Commands are executed line by line

Why MATLAB®?

1. MATLAB is used by engineers and scientists world wide to analyze and design products and systems
2. MATLAB is a powerful programming tool for Mathematics and Statistics because it has built-in functions for performing a wide variety of mathematical operations, plotting and visualization of data.
3. MATLAB has additional packages that can be installed for specific subject areas, e.g Symbolic Math Toolbox ™ for performing calculus on functions, Deep Learning Toolbox™ for exploring AI algorithms.
4. The MATLAB programming environment allows one to customize programs to fit specifications.

Objectives

- Familiarize yourself with the MATLAB programming environment

  At the most basic level we can think of MATLAB as a very powerful graphing calculator! (with may built-in functions)

- Learn how to use MATLAB to:
  1. execute simple commands.
  2. create variables and display their values

MATLAB WINDOWS

- MATLAB has 8 main windows, the first 4 are on the main screen by default

Command window	Enter commands and variables, run programs
Command history window	History of commands entered in window
Workspace window	Information about current variables
Current directory window	Shows files in the current folder/directory
Figure window	Output from graphic commands
Editor window	Create and debug scripts and functions
Help window	Get help on functions and scripts
Launch pad window	Access to tools, demos and documentation

- Some commands
  - `clc` - Clear command window
  - `clf` - Clear figure
  - `exit` or `quit` - Quit MATLAB
  - `disp('text')` - Displays `text`

Variables and Expressions

Variable - A value that can change depending on conditions or information passed to the program. In contrast, a program may have data values that are fixed constants.

Expressions - These are created using values, variables previously created, mathematical operators and built in functions e.g.

```
>> 2*cos(1) + sqrt(2)
 ans =
     2.4948
```

Adding a semi-colon `(;)` at the end of the expression suppresses screen output

Expressions and formatting

- The default in MATLAB is to display numbers that have decimal points with 4 decimal places.
- There are many other options
  1. **format** `long` will result in 15 decimal places

```
>> format long
>> 2*cos(1) + sqrt(2)
ans =
   2.494818174109374
```

  2. **format** `shortEng` will result in 4 decimal places engineering format.

```
>> format shortEng
>> 2*cos(1) + sqrt(2)
ans =
2.4948e+000
```

Basic formats displaying π

`format short` (**default**)	`3.1416`
`format long`	`3.141592653589793`
`format rat`	`355/113`
`format bank`	`3.14`
`format short e`	`3.1416e+00`
`format long e`	`3.141592653589793e+00`
`format compact`	(no blank lines)

Creating Variables

We create variables in MATLAB using an assignment statement

```
>> variablename = expression
```

REMARKS

1. The '=' is an assignment operator (unlike the mathematics equal sign, does not mean equality)
2. The `expression` is evaluated at the result is stored in `variablename`

Use variable names that make sense, e.g. if a variable to store the radius the name `radius` makes sense, `y` does not!

Rules for variable names

1. Must begin with a letter of the alphabet, after that it may contain letters,digits, and the underscore but NO SPACES! e.g. `interest_rate`,`my_num_1`, e.t.c.
2. The limit to the number of characters in a variable name is 63 stored in `namelengthmax`.
3. MATLAB is case sensitive so `INTEREST_RATE` is different from `Interest_rate`, e.t.c
4. Certain words are reserved so they should not be used as variable names.
5. Names of built in functions cannot be used as variable names.

Some reserved variable names

- You should not these variables in your code

`ans:` The result of a previous calculation

`computer:` The type of computer you are on

`eps:` The smallest positive number ϵ that satisfies $1 + \epsilon > 1$

`i,j:` The imaginary unit ($\sqrt{-1}$)

`realmax,realmin:` The largest and smallest real numbers that can be represented on this machine.

Commands related to Variables

1. **who** shows variables that have been defined
2. **whos** shows more information about the type of variables
3. **clear** clears out all variables. You can also specify a specific variable to clear.
4. **save** saves all your variables in a file `matlab.mat`

Mathematical Operators

*	multiplication
+	addition
-	negation, subtraction
/	division (division by, e.g. $10/5 = 2$)
\	division (division into, e.g, $5 \backslash 10 = 2$)
^	exponentiation (e.g. $5^2 = 25$)

Operator precedence rules

- MATLAB follows mathematical rules for order of operations:

$$\{\,()\,\} \longrightarrow \{\,\hat{}\,\} \longrightarrow \{\,*\ /,\ \backslash\,\} \longrightarrow \{\,+,\ -\,\}$$

- You can use parenthesis to change the precedence of an expression.

MATLAB built-in functions and constants

- Some basic functions and constants:

`sqrt(x)`	square root
`exp(x)`	Exponential (e^x)
`abs(x)`	Absolute value
`log(x)`	Natural logarithm
`sin(x)`	Sine of x
`cos(x)`	Cosine of x
`tan(x)`	Tangent of x
`pi`	π
`i`	$\sqrt{-1}$
`inf`	∞
`NaN`	"not a number", such as the result of $0/0$.

Help and lookfor commands

There are many built-in functions in MATLAB:

- Use **lookfor** can be used to find a specific command to perform a task.
- Use **help** to find out information about a known command.

`diary` function

- You can keep a record of your MATLAB sessions using the `diary` function.
- Calling

```
>> diary filename
```

  will save the commands your enter into the command window along with the output in a plain text file.
- `diary off` and `diary on` will pause and restart the recording, resp.

MATLAB `script files` a.k.a `m-files`

- MATLAB `script files` (also known as `m-files`) are useful for running commands multiple types without having to type them into the command line.
- Script files are named with an extension `.m`
- You can type in your commands into the `m-file`, edit, save them and execute them as many times as you want.
- The MATLAB editor is the best way to create `m-files`.
- A typical script file will contain MATLAB commands and comments
- Comments describe the purpose of the script file and any useful information about the commands executed
- Comments within `m-files` begin with (`%`)

Vectors and Matrices

- **Vectors** and **matrices** are used to store values of the same type
- A **vector** can be either *column vector* or a *row vector*.
- **Matrices** can be visualized as a table of values with dimensions $r \times c$ (r is the number of rows and c is the number of columns).

Creating row vectors

Place the values that you want in the vector in square brackets separated by either spaces or commas. e.g

```
>> vec=[1 2 3 4 5]
vec =
     1     2     3     4     5

>> vec=[1,2,3,4,5]
vec=
     1     2     3     4     5
```

Creating row vectors - colon operator

If the values of the vectors are regularly spaced, the colon operator can be used to iterate through these values. e.g.

```
>> vec = 1:5
vec =
     1     2     3     4     5
```

A **step value** can also be specified with another colon in the form (first:step:last)

```
>>vec = 1:2:9
vec =
     1     3     5     7     9
```

Creating row vectors - colon operator

Example

In using (`first:step:last`)
What happens if adding the step value would go beyond the range specified by last? e.g:

```
>>v = 1:2:6
```

Answer

```
>> v= 1:2:6
v =
     1     3     5
```

Creating row vectors - `colon operator`

Example

How can we use (`first:step:last`) to generate the vector `v1 = [9 7 5 3 1 ]`?

Answer

```
>> v1= 9:-2:1
v1 =
     9     7     5     3     1
```

Creating row vectors - `linspace` function

linspace

```
>>linspace(x,y,n)
```

creates a row vector with n values in the inclusive range from x to y.

Example

```
>>v2 = linspace(3,15,5)
  v2 =
       3     6     9    12    15
```

Vector concatenation

- We can use existing vectors to create new ones

```
>> v1= 9:-2:1
v1 =
     9     7   5   3   1
>> v2 = linspace(3,15,5)
v2 =
     3     6   9   12  15
>> newvec = [v1 v2]
newvec =
     9   7   5   3   1   3   6   9   12  15
```

Accessing elements of a vector

newvec

```
newvec =
     9     7     5     3     1     3     6     9    12    15
```

- 5^{th} element

```
>> newvec(5)
ans =
     1
```

- Elements 4 through 6

```
>> newvec(4:6)
ans =
     3     1     3
```

Accessing elements of a vector

newvec

```
newvec =
     9    7    5    3    1    3    6    9    12   15
```

- Elements 2, 3, 7

```
>>newvec([2 3 7])
ans =
     7    5    6
```

- To set the first entry to 10

```
>> newvec(1)=10
newvec =
10   7    5    3    1    3    6    9    12   5
```

Modifying vector entries

- Create a vector of size 10 with all zero entries

```
>> vec=zeros(1,10)
vec =
     0     0     0     0     0     0     0     0     0     0
```

- Set entries from 2 to 4 to 1

```
>> vec(2:4)=1
vec =
     0     1     1     1     0     0     0     0     0     0
```

- Set entries from 2 to 4 to 2 to 4!

```
>> vec(2:4)=2:4
vec =
     0     2     3     4     0     0     0     0     0     0
```

What about column vectors?

- Explicitly

```
>> col_vec = [2;4;6;8]
col_vec =
     2
     4
     6
     8
```

- No direct way using `colon` operator or `linspace` function...BUT we can fix this!

What about column vectors?

- Simply take the transpose of your row vector!

```
>> row_vec =2:2:8
row_vec =
       2    4    6    8
>> col_vec = row_vec'
col_vec =
       2
       4
       6
       8
```

Matrix variables

- We generalize the creation of row and column vectors, e.g.

```
>> mat1 = [3 4 5; 2 5 6]
mat =
     3     4     5
     2     5     6
```

- Or use the `colon` operator within rows

```
>> mat2=[1:3; 4:6]
mat2 =
     1     2     3
     4     5     6
```

Caution

- NOTE: There must always be the same number of values in each row
- If you attempt to create a matrix with different number of values in the rows MATLAB will complain:

```
>> mat=[1 2 4; 3 4]
Error using vertcat
Dimensions of matrices being concatenated
```

Matrices of random numbers

- `rand(n)` - returns an $n \times n$ matrix of random values on $(0, 1)$.
- `rand(m,n)` - returns an $m \times n$ matrix.
- A 4×4 matrix of random entries on $(0, 1)$.

```
>> rand(3)
ans =
    0.9649    0.9572    0.1419
    0.1576    0.4854    0.4218
    0.9706    0.8003    0.9157
```

- A (4×3) matrix of random entries on $(0, 1)$.

```
>> rand(2,3)
ans =
    0.7922    0.6557    0.8491
    0.9595    0.0357    0.9340
```

Referring to and modifying matrix elements

- Given the following 3×4 matrix

```
>> mat=[1:4;5:8;9:12]
mat =
     1     2     3     4
     5     6     7     8
     9    10    11    12
```

- We can extract element $mat_{2,3}$ as follows:

```
>> mat(2,3)
ans =
     7
```

- The submatrix consisting of rows $1, 2$ and columns $2, 3$:

```
>> mat(1:2,2:3)
 ans =
     2     3
     6     7
```

Referring to and modifying matrix elements

- Given

```
>> mat=[1:4;5:8;9:12]
mat =
     1     2     3     4
     5     6     7     8
     9    10    11    12
```

- Extract the first row of `mat`

```
>> mat(1,:)
ans =
     1     2     3     4
```

- Extract the first column of `mat`

```
>> mat(:,1)
ans =
     1
     5
     9
```

Referring to and modifying matrix elements

- Given

```
>> mat=[1:4;5:8;9:12]
mat =
     1     2     3     4
     5     6     7     8
     9    10    11    12
```

- Set the first column of `mat` to `[5;6;7]`

```
>> mat(:,1) = 5:7
mat =
     5     2     3     4
     6     6     7     8
     7    10    11    12
```

Deleting vector entries

- Given a vector with 8 random integers between 1 and 10

```
>> vec=round((rand(1,8))*10)
vec =
     7     2     1     5    10     3     6
```

- We can delete the 3*rd* entry using the *empty vectors* `[]` as

```
>> vec(3)=[]
vec =
     7     2     5    10     3     6     2
```

- To delete multiple entries, simply pass in an `index array`, e.g. to delete all even entries:

```
>> vec(2:2:length(vec)) =[]
vec =
     7     1    10     6
```

- Note: Here, we can also use `end` as `>>vec(2:2:end) = []`

Deleting matrix entries

- Given a 5×5 matrix

```
>> M=magic(5) %look up the magic function!
M =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
```

- Delete the 3*rd* column

```
>> M(:,3)=[]
M =
    17    24     8    15
    23     5    14    16
     4     6    20    22
    10    12    21     3
    11    18     2     9
```

Adding entries

- Given a 5×5 matrix

```
>> M=magic(5)
M =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
```

- Add a 6th row

```
>> [M,[1:5]']
ans =
    17    24     1     8    15     1
    23     5     7    14    16     2
     4     6    13    20    22     3
    10    12    19    21     3     4
    11    18    25     2     9     5
```

Some useful array commands

Find out what each command does before starting **Homework 1**.

1. `length`
2. `size`
3. `numel`
4. `max`
5. `min`
6. `sort`
7. `zeros`
8. `ones`
9. `reshape`
10. `eye`
11. `spy`