1 I INTRODUCTION TO MATLAB

Jay Klopper MD MMed(Surgery)Cum Laude

The George Washington University

Table of Contents

Introduction

Arithmetic allows for an easy introduction to a new computer language. In this chapter we explore addition, subtraction, multiplication, division, powers, logarithms, the exponent, and trigonometric and inverse trigonometric functions. These are simple calculations that can be performed using a calculator. They do, however, serve as a gateway to all the power of MATLAB.

We also briefly consider variables and container types. The former are the name that we choose to assign an object to, such that the object (its value and type) are stored in computer memory and can be recalled by using the name or by viewing the Workspace tab. MATLAB holds data in containers, which we explore at the end of the chapter.

All notebooks are started by clearing the workspace of all variables and values. Variables persist in a MATLAB session and can cause issues when we do not keep track of the workspace. Below, we use the `clear global` command.

```
clear global
```

Addition and subtraction

Simple addition and subtraction allows us to write our first code. The usage and results are what we typically expect from a calculator. The code comments provide insights into the calculations.

```
% Add two numbers
3 + 4
```

```
ans = 7
```

```
% Addition of more than two numbers
3 + 4 + 10
```

```
ans = 17
```

```
% Subtraction of two numbers
10 - 3
```

```
ans = 7
```

```
% Subtraction of more than two numbers
10 - 3 - 4
```

```
ans = 3
```

```
% Subtraction is not commutative
4 - 3
```

```
ans = 1
```

```
3 - 4
```

```
ans = -1
```

```
% Addition and subtraction (left to right)
10 - 3 + 4
```

```
ans = 11
```

Multiplication and division

Multiplication is performed using the `*` operator and division is performed using the `/` operator. below, we view a few examples. As before, the code comments provide insights into the calculations.

```
% Multiplying two numbers
3 * 4
```

```
ans = 12
```

```
% Multiplication of more than two numbers
3 * 4 * (-1)
```

```
ans = -12
```

```
% Division of two numbers
1 / 2
```

```
ans = 0.5000
```

```
% Division of more than two numbers (left to right)
12 / 2 / 3
```

```
ans = 2
```

```
% Division is not commutative
12 / 2
```

```
ans = 6
```

```
2 / 12
```

```
ans = 0.1667
```

```
% Multiplication and division
12 * 2 / 6
```

```
ans = 4
```

Order of arithmetical operation

MATLAB obeys the usual order of arithmetical operations with parentheses before exponentiation before division and multiplication before subtraction and addition. The simple example of $(2+4) \times 10$ and $2 + 4 \times 10$ illustrates the order.

```
% Add 2 and 4 and then multiply by 10
(2 + 4) * 10
```

```
ans = 60
```

```
% Multiplication before addition
2 + 4 * 10
```

```
ans = 42
```

Powers

A power, written as a^b is achieved using the `^` symbol. Below, we calculate 2^4, 2^{3+2} and $2^3 2^2$. Note that the latter two examples are equivalent where for the integers b, c and real number a we have that $a^{b+c} = a^b a^c$.

```
% Simple power
2^4
```

```
ans = 16
```

```
% Rules of powers
2^(3 + 2)
```

```
ans = 32
```

```
2^3 * 2^2
```

```
ans = 32
```

Exponentiation and logarithms

Recall that $\log_b a = c$ means that $b^c = a$. The `log` function uses base e and is the natural logarithm. We can confirm Euler's identity $e^{i\pi} = -1$ where i is the imaginary unit by calculating the natural log of -1.

```
% Logarithm base e
log(-1)
```

```
ans = 0.0000 + 3.1416i
```

The result is indeed $i\pi$.

To use base 10 we use the `log10` function. below, we see that $\log_{10} 100 = 2$ since $10^2 = 100$.

```
% Logarithm based 10
log10(100)
```

```
ans = 2
```

The `log2` function uses base 2.

```
% Logarithm based 2
log2(16)
```

```
ans = 4
```

If e is the irrational number, we can calculate the value of e (Euler's number) by calculating e^1 using the exp function.

```
% Exponent (Euler's number)
exp(1)
```

```
ans = 2.7183
```

Trigonometric functions

MATLAB can calculate all trigonometric and inverse trigonometric functions. Note that they keyword `pi` holds a numerical approximation of the irrational number π.

```
% Ratio of circumference to diameter of a circle
pi
```

```
ans = 3.1416
```

Below, we calculate $\sin\left(\frac{\pi}{2}\right)$, $\cos\left(\frac{\pi}{2}\right)$, $\tan\left(\frac{\pi}{2}\right)$, $\csc\left(\frac{\pi}{2}\right)$, $\sec\left(\frac{\pi}{2}\right)$, and $\cot\left(\frac{\pi}{2}\right)$.

```
% Sine function
sin(pi / 2)
```

```
ans = 1
```

```
% Cosine function
cos(pi / 2)
```

```
ans = 6.1232e-17
```

```
% Tangent function
tan(pi / 4)
```

```
ans = 1.0000
```

```
% Co-secant function
csc(pi / 2)
```

```
ans = 1
```

```
% Secant function
sec(pi)
```

```
ans = -1
```

```
% Cotangent function
cot(pi)
```

```
ans = -8.1656e+15
```

Inverse trigonometric function

We also demonstrate the inverse trigonometric functions by calculating $\arcsin(1)$, $\arccos(0)$, and $\arctan(1)$.

```
% Inverse sine function
asin(1)
```

```
ans = 1.5708
```

```
% Inverse cosine function
acos(0)
```

```
ans = 1.5708
```

```
% Inverse tangent function
atan(1)
```

```
ans = 0.7854
```

Computer variables

When we create objects in a computer language, we can save this object to memory. It is our task to name this space in memory where the object resides. We have used objects in the previous chapter. Any number, either by itself, or the result of a calculation, is an object.There are many other object, some of which we will discover later.

There are rules and formats for naming variables. In this tutorial series we will use descriptive names (names describing what data is being held) and the camelCase format for variable names. The first letter is a lowercase letter from the alphabet. If the name consists of two or more word, spaces are omitted (as they are illegal characters) and the first letter of each word is in uppercase.

An object is assigned to a computer variable using the assignment operator, which is the equal symbol = in most languages. The assignment operator assigns the object to its right to the computer variable to its left. Below we assign the number 17 (an object) to the computer variable x.

```
clearvars % Clear all variables in the workspace
% Assign the number 7 (an object) to the variable x
x = 17
```

```
x = 17
```

We can now reference the name that we gave to the space in memory that is storing our object.

```
% Call the variable x
x
```

```
x = 17
```

Irrespective of the naming convention (format) it is good practice to use a memorable name. This will help us later when we revisit our code or help others that may read our code.

Container types

MATLAB has various containers for holding and representing data. An object is either empty or it holds values. In the next chapter we explore arrays in the form of vectors and matrices. Here, we list seven other container types.

1. **Cell Arrays**:

  - Description: Cell arrays are data types with indexed data containers called cells, where each cell can contain any type of data. This includes arrays of different sizes and types.

  - Usage Example: `C={1,'text',[2,3,4],magic(3)};`

```
% Create the cell array C
C = {1,'text',[2,3,4],magic(3)} % Search the documentation (upper right corner) to look up the magic function
```

C = 1×4 cell

	1	2	3	4
1	1	'text'	[2,3,4]	[8,1,6;3,5,7;…

2. **Structured Arrays** (Structs):

  - Description: Structures are MATLAB data types that group related data using data containers called fields. Each field can contain any type of data. Fields are accessed using dot notation.

  - Usage Example: `S.name = 'John'; S.age = 32; S.scores = [85, 90, 95];`

```
% Create the structure array S
S.name = 'John'
```

```
S = struct with fields:
    name: 'John'
```

```
S.age = 32
```

```
S = struct with fields:
    name: 'John'
     age: 32
```

```
S.scores = [85,90,95]
```

```
S = struct with fields:
      name: 'John'
       age: 32
    scores: [85 90 95]
```

3. **Tables**:

- Description: Tables are data types suitable for storing column-oriented or tabular data of varying types. They are similar to data frames in R or Python's pandas. Tables allow for data manipulation and access using row and column names.

- Usage Example: my_table = `table([1;2;3],{'A';'B';'C'},[10.1;11.2;12.3],'VariableNames',{'ID','Label','Value'});`

```
% Create the table my_table
my_table = table([1;2;3],{'A';'B';'C'},[10.1;11.2;12.3],'VariableNames',{'ID','Label','Value'})
```

my_table = 3×3 table

	ID	Label	Value
1	1	'A'	10.1000
2	2	'B'	11.2000
3	3	'C'	12.3000

4. **Categorical Arrays**:

- Description: Categorical arrays are used for storing data that can take on a limited and fixed number of discrete categories. This is useful for nominal or ordinal data.

- Usage Example: my_cat_array = `categorical({'red','green','blue','red','green'});`

```
% Create the categorical array my_cat_array
my_cat_array = categorical({'red','green','blue','red','green'})
```

```
my_cat_array = 1×5 categorical
red         green       blue        red         green
```

5. **Containers.Map**:

- Description: `Containers.Map` is an associative array that allows for the storage of key-value pairs, where keys are unique identifiers. This is similar to dictionaries in Python.

- Usage Example: `mapObj = containers.Map({'a', 'b', 'c'}, [1, 2, 3]);`

```
% Create the containers.Map object mapObj
mapObj = containers.Map({'a','b','c'},[1,2,3])
```

```
mapObj = 
  Map with properties:

        Count: 3
      KeyType: char
    ValueType: double
```

We use attribute notation to return the keys and the values.

```
% Call the keys
mapObj.keys
```

```
ans = 1×3 cell
'a'         'b'         'c'
```

```
% Call the values
mapObj.values
```

ans = 1×3 cell

	1	2	3
1	1	2	3

6. **Time Series**:

   - Description: Time series objects store time-stamped data, often used for financial, economic, and other time-dependent data.

   - Usage Example: `datetime(2023,1,1):days(1):datetime(2023,1,10)`

```
% Create 10 sequential dates from January 1, 2023 to January 10, 2023
datetime(2023,1,1):days(1):datetime(2023,1,10)
```

```
ans = 1×10 datetime
01-Jan-2023 02-Jan-2023 03-Jan-2023 04-Jan-2023 05-Jan-2023 06-Jan-2023 07-Jan-2023 08-Jan-
```

```
% Alternative notation using the caldays function
datetime(2023, 1, 1) + caldays(0:9)
```

```
ans = 1×10 datetime
01-Jan-2023 02-Jan-2023 03-Jan-2023 04-Jan-2023 05-Jan-2023 06-Jan-2023 07-Jan-2023 08-Jan-
```

7. **String Arrays**:

   - Description: String arrays are containers for storing pieces of text data, where each element of the array is a string.

   - Usage Example: `["hello", "world", "MATLAB"];`

```
% Create a string array
["hello","world","MATLAB"] % Must use double quotation marks
```

```
ans = 1×3 string
"hello"     "world"     "MATLAB"
```

These container types provide flexibility in handling various types of data, allowing for efficient data manipulation and organization in MATLAB.Hel