

Chapter 2. MATLAB Fundamentals

Chapter Objectives

- ▶ Learning how real and complex numbers are assigned to variables.
- ▶ Learning how vectors and matrices are assigned values using simple assignment, the colon operator, and the `linspace` and `logspace` functions.
- ▶ Understanding the priority rules for constructing mathematical expressions.
- ▶ Gaining a general understanding of built-in functions and how you can learn more about them with MATLAB's Help facilities.
- ▶ Learning how to use vectors to create a simple line plot based on an equation.

The MATLAB Environment

- ▶ MATLAB uses three primary windows-
 - ▶ Command window - used to enter commands and data
 - ▶ Graphics window(s) - used to display plots and graphics
 - ▶ Edit window - used to create and edit M-files (programs)
- ▶ Depending on your computer platform and the version of MATLAB used, these windows may have different looks and feels.

Calculator Mode

- ▶ The MATLAB command window can be used as a calculator where you can type in commands line by line. Whenever a calculation is performed, MATLAB will assign the result to the built-in variable `ans`

- ▶ **Example:**

```
>> 55 - 16
```

```
ans =  
    39
```

MATLAB Variables

- ▶ While using the `ans` variable may be useful for performing quick calculations, its transient nature makes it less useful for programming.
- ▶ MATLAB allows you to assign values to variable names. This results in the storage of values to memory locations corresponding to the variable name.
- ▶ MATLAB can store individual values as well as arrays; it can store numerical data and text (which is actually stored numerically as well).
- ▶ MATLAB does not require that you pre-initialize a variable; if it does not exist, MATLAB will create it for you.

Scalars

- ▶ To assign a single value to a variable, simply type the variable name, the = sign, and the value:

```
>> a = 4
```

```
a =
```

```
4
```

- ▶ Note that variable names must start with a letter, though they can contain letters, numbers, and the underscore (_) symbol

Scalars (cont)

- ▶ You can tell MATLAB not to report the result of a calculation by appending the semi-solon (;) to the end of a line. The calculation is still performed.
- ▶ You can ask MATLAB to report the value stored in a variable by typing its name:

```
>> a
```

```
a =
```

```
4
```

Scalars (cont)

- ▶ You can use the complex variable i (or j) to represent the unit imaginary number.
- ▶ You can tell MATLAB to report the values back using several different formats using the `format` command. Note that the values are still *stored* the same way, they are just displayed on the screen differently. Some examples are:
 - ▶ `short` - scaled fixed-point format with 5 digits
 - ▶ `long` - scaled fixed-point format with 15 digits for `double` and 7 digits for `single`
 - ▶ `short eng` - engineering format with at least 5 digits and a power that is a multiple of 3 (useful for SI prefixes)

Format Examples

```
▶ >> format short; pi
ans =
    3.1416
>> format long; pi
ans =
    3.14159265358979
>> format short eng; pi
ans =
    3.1416e+000
>> pi*10000
ans =
    31.4159e+003
```

- ▶ **Note** - the format remains the same unless another `format` command is issued.

Arrays, Vectors, and Matrices

- ▶ MATLAB can automatically handle rectangular arrays of data - one-dimensional arrays are called *vectors* and two-dimensional arrays are called *matrices*.
- ▶ Arrays are set off using square brackets [and] in MATLAB
- ▶ Entries within a row are separated by spaces or commas
- ▶ Rows are separated by semicolons

Array Examples

```
▶ >> a = [1 2 3 4 5 ]
a =
     1     2     3     4     5
>> b = [2; 4; 6; 8; 10]
b =
     2
     4
     6
     8
    10
```

- ▶ Note 1 - MATLAB does not *display* the brackets
- ▶ Note 2 - if you are using a monospaced font, such as Courier, the displayed values should line up properly

Matrices

- ▶ A 2-D array, or matrix, of data is entered row by row, with spaces (or commas) separating entries within the row and semicolons separating the rows:

```
>> A = [1 2 3; 4 5 6; 7 8 9]
```

```
A =
```

```
    1    2    3
    4    5    6
    7    8    9
```

Useful Array Commands

- ▶ The transpose operator (apostrophe) can be used to flip an array over its own diagonal. For example, if b is a row vector, b' is a column vector containing the complex conjugate of b .
- ▶ The command `window` will allow you to separate rows by hitting the Enter key - script files and functions will allow you to put rows on new lines as well.
- ▶ The `who` command will report back used variable names; `whos` will also give you the size, memory, and data types for the arrays.

Accessing Array Entries

- ▶ Individual entries within a array can be both read and set using either the *index* of the location in the array or the row and column.
- ▶ The index value starts with 1 for the entry in the top left corner of an array and increases down a column - the following shows the indices for a 4 row, 3 column matrix:

1	5	9
2	6	10
3	7	11
4	8	12

Accessing Array Entries (cont)

- ▶ Assuming some matrix **C**:

```
C =  
  2      4      9  
  3      3     16  
  3      0      8  
 10     13     17
```

- ▶ `C(2)` would report 3
- ▶ `C(4)` would report 10
- ▶ `C(13)` would report an error!
- ▶ Entries can also be access using the row and column:
- ▶ `C(2,1)` would report 3
- ▶ `C(3,2)` would report 0
- ▶ `C(5,1)` would report an error!

Array Creation - Built In

- ▶ There are several built-in functions to create arrays:
 - ▶ `zeros(r, c)` will create an `r` row by `c` column matrix of zeros
 - ▶ `zeros(n)` will create an `n` by `n` matrix of zeros
 - ▶ `ones(r, c)` will create an `r` row by `c` column matrix of ones
 - ▶ `ones(n)` will create an `n` by `n` matrix one ones
- ▶ `help elmat` has, among other things, a list of the elementary matrices

Array Creation - Colon Operator

- ▶ The colon operator `:` is useful in several contexts. It can be used to create a linearly spaced array of points using the notation

```
start:difval:limit
```

where `start` is the first value in the array, `difval` is the difference between successive values in the array, and `limit` is the *boundary* for the last value (though not necessarily the last value).

```
>>1:0.6:3
```

```
ans =
```

```
1.0000    1.6000    2.2000    2.8000
```

Colon Operator - Notes

- ▶ If `diffval` is omitted, the default value is 1:

```
>>3:6
ans =
     3     4     5     6
```

- ▶ To create a decreasing series, `diffval` must be negative:

```
>> 5:-1.2:2
ans =
  5.0000  3.8000  2.6000
```

- ▶ If `start+diffval>limit` for an increasing series or `start+diffval<limit` for a decreasing series, an empty matrix is returned:

```
>>5:2
ans =
Empty matrix: 1-by-0
```

- ▶ To create a column, transpose the output of the colon operator, not the limit value; that is, `(3:6)'` not `3:6'`

Array Creation - linspace

- ▶ To create a row vector with a specific number of linearly spaced points between two numbers, use the `linspace` command.
- ▶ `linspace(x1, x2, n)` will create a linearly spaced array of `n` points between `x1` and `x2`

```
>>linspace(0, 1, 6)  
ans =  
    0    0.2000    0.4000    0.6000    0.8000  
    1.0000
```
- ▶ If `n` is omitted, 100 points are created.
- ▶ To generate a column, transpose the output of the `linspace` command.

Array Creation - logspace

- ▶ To create a row vector with a specific number of logarithmically spaced points between two numbers, use the `logspace` command.
- ▶ `logspace(x1, x2, n)` will create a logarithmically spaced array of n points between 10^{x1} and 10^{x2}

```
>>logspace(-1, 2, 4)  
ans =  
    0.1000    1.0000   10.0000  100.0000
```
- ▶ If n is omitted, 100 points are created.
- ▶ To generate a column, transpose the output of the `logspace` command.

Mathematical Operations

- ▶ Mathematical operations in MATLAB can be performed on both scalars and arrays.
- ▶ The common operators, in order of priority, are:

^	Exponentiation	$4^2 = 8$
-	Negation (unary operation)	$-8 = -8$
* /	Multiplication and Division	$2*\pi = 6.2832$ $\pi/4 = 0.7854$
\	Left Division	$6\backslash 2 = 0.3333$
+ -	Addition and Subtraction	$3+5 = 8$ $3-5 = -2$

Order of Operations

- ▶ The order of operations is set first by parentheses, then by the default order given above:
 - ▶ $y = -4 ^ 2$ gives $y = -16$
since the exponentiation happens first due to its higher default priority, but
 - ▶ $y = (-4) ^ 2$ gives $y = 16$
since the negation operation on the 4 takes place first

Complex Numbers

- ▶ All the operations above can be used with complex quantities (i.e. values containing an imaginary part entered using i or j and displayed using i)

```
x = 2+i*4; (or 2+4i, or 2+j*4, or 2+4j)
```

```
y = 16;
```

```
3 * x
```

```
ans =
```

```
6.0000 +12.0000i
```

```
x+y
```

```
ans =
```

```
18.0000 + 4.0000i
```

```
x'
```

```
ans =
```

```
2.0000 - 4.0000i
```

Vector-Matrix Calculations

- ▶ MATLAB can also perform operations on vectors and matrices.
- ▶ The * operator for matrices is defined as the *outer product* or what is commonly called “matrix multiplication.”
 - ▶ The number of columns of the first matrix must match the number of rows in the second matrix.
 - ▶ The size of the result will have as many rows as the first matrix and as many columns as the second matrix.
 - ▶ The exception to this is multiplication by a 1x1 matrix, which is actually an array operation.
- ▶ The ^ operator for matrices results in the matrix being matrix-multiplied by itself a specified number of times.
 - ▶ Note - in this case, the matrix must be square!

Element-by-Element Calculations

- ▶ At times, you will want to carry out calculations item by item in a matrix or vector. The MATLAB manual calls these *array operations*. They are also often referred to as *element-by-element* operations.
- ▶ MATLAB defines `.*` and `./` (note the dots) as the array multiplication and array division operators.
 - ▶ For array operations, both matrices must be the same size or one of the matrices must be 1x1
- ▶ Array exponentiation (raising each element to a corresponding power in another matrix) is performed with `.^`
 - ▶ Again, for array operations, both matrices must be the same size or one of the matrices must be 1x1

Built-In Functions

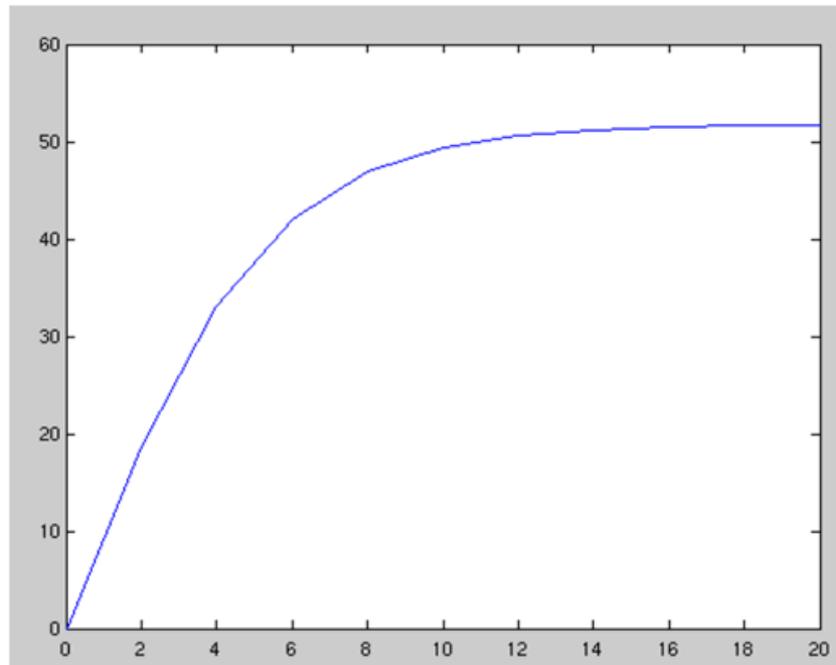
- ▶ There are several built-in functions you can use to create and manipulate data.
- ▶ The built-in help function can give you information about both what exists and how those functions are used:
 - ▶ `help elmat` will list the elementary matrix creation and manipulation functions, including functions to get information about matrices.
 - ▶ `help elfun` will list the elementary math functions, including trig, exponential, complex, rounding, and remainder functions.
- ▶ The built-in `lookfor` command will search help files for occurrences of text and can be useful if you know a function's purpose but not its name

Graphics

- ▶ MATLAB has a powerful suite of built-in graphics functions.
- ▶ Two of the primary functions are `plot` (for plotting 2-D data) and `plot3` (for plotting 3-D data).
- ▶ In addition to the plotting commands, MATLAB allows you to label and annotate your graphs using the `title`, `xlabel`, `ylabel`, and `legend` commands.

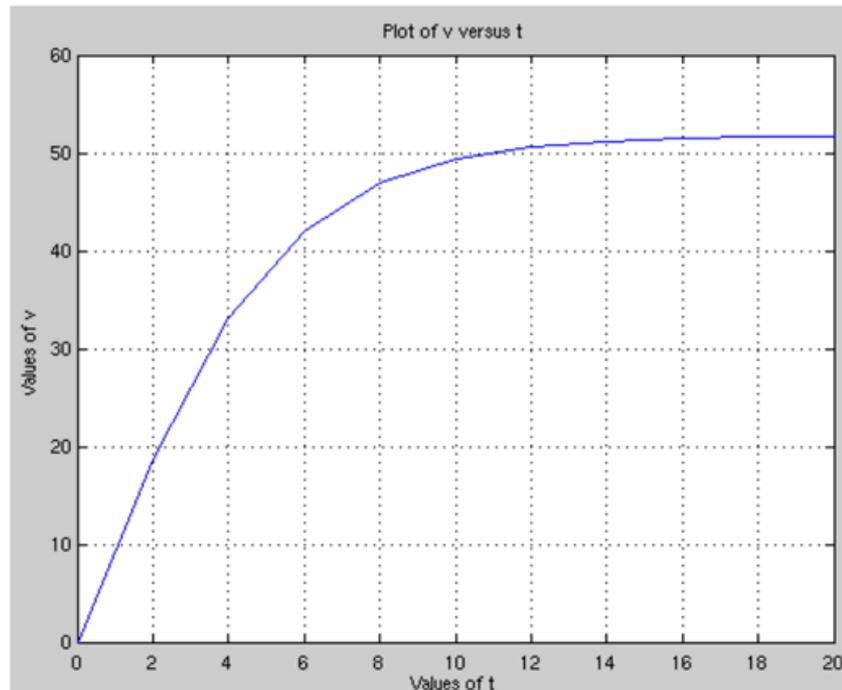
Plotting Example

```
▶ t = [0:2:20]';  
g = 9.81; m = 68.1; cd = 0.25;  
v = sqrt(g*m/cd)*tanh(sqrt(g*cd/m)*t);  
plot(t, v)
```



Plotting Annotation Example

- ▶ `title('Plot of v versus t')`
`xlabel('Values of t')`
`ylabel('Values of v')`
`grid`



Plotting Options

- ▶ When plotting data, MATLAB can use several different colors, point styles, and line styles. These are specified at the end of the `plot` command using *plot specifiers* as found in Table 2.2.
- ▶ The default case for a single data set is to create a blue line with no points. If a line style is specified with no point style, no point will be drawn at the individual points; similarly, if a point style is specified with no point style, no line will be drawn.
- ▶ Examples of plot specifiers:
 - ▶ 'ro:' - red dotted line with circles at the points
 - ▶ 'gd' - green diamonds at the points with no line
 - ▶ 'm--' - magenta dashed line with no point symbols

Other Plotting Commands

- ▶ `hold on` and `hold off`
 - ▶ `hold on` tells MATLAB to keep the current data plotted and add the results of any further plot commands to the graph. This continues until the `hold off` command, which tells MATLAB to clear the graph and start over if another plotting command is given. `hold on` should be used *after* the first plot in a series is made.
- ▶ `subplot(m, n, p)`
 - ▶ `subplot` splits the figure window into an $m \times n$ array of small axes and makes the p^{th} one active. Note - the first subplot is at the top left, then the numbering continues across the row. This is different from how elements are numbered within a matrix!