Introduction to MATLAB

The Desktop
When you start MATLAB, the desktop appears, containing tools (graphical user interfaces) for managing files, variables, and applications associated with MATLAB. The following illustration shows the default desktop. You can customize the arrangement of tools and documents to suit your needs.

Start Button
The MATLAB Start button provides easy access to tools, demos, shortcuts, and documentation. Click the Start button to see the options.
**Command Window**

Use the Command Window to enter variables and to run MATLAB functions and scripts. MATLAB displays the results.

```
>> Magic(4)
ans =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

Press the up arrow key ↑ to recall a statement you previously typed. Edit the statement as needed, and then press **Enter** to run it.

**Command History**

Statements you enter in the Command Window are logged with a timestamp in the Command History. From the Command History, you can view and search for previously run statements, as well as copy and execute selected statements. You can also create a file from selected statements.
Basic Arithmetic

**Numbers:**
MATLAB uses conventional decimal notation, with an optional decimal point and leading plus or minus sign, for numbers.
Scientific notation uses the letter e to specify a power-of-ten scale factor.
Imaginary numbers use either i or j as a suffix.

Some examples of legal numbers are:

<table>
<thead>
<tr>
<th>Number</th>
<th>3</th>
<th>-99</th>
<th>0.0001</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.6397238</td>
<td>1.60210e-20</td>
<td>6.02252e23</td>
<td></td>
</tr>
<tr>
<td>1i</td>
<td>-3.14159j</td>
<td>3e5i</td>
<td></td>
</tr>
</tbody>
</table>

MATLAB stores all numbers internally using the long format specified by the IEEE® floating-point standard. Floating-point numbers have a finite precision of roughly 16 significant decimal digits and a finite range of roughly $10^{-308}$ to $10^{308}$.

**The format Function**
The format function controls the numeric format of the values displayed. The function affects only how numbers are displayed, *not how MATLAB software computes or saves them*. Here are the different formats, together with the resulting output produced from a vector x with components of different magnitudes.

```matlab
x = [4/3 1.2345e-6]
format short
1.3333 0.0000
format short e
1.3333e+000 1.2345e-006
format short g
1.3333 1.2345e-006
format long
1.33333333333333 0.00000123450000
format long e
1.33333333333333e+000 1.23450000000000e-006
format long g
1.33333333333333 1.2345e-006
format bank
1.33 0.00
format rat
4/3 1/810045
format hex
3ff555555555555 3eb4b6231abfd271
```
If the largest element of a matrix is larger than $10^3$ or smaller than $10^{-3}$, MATLAB applies a common scale factor for the short and long formats.
In addition to the \texttt{format} functions shown above, \texttt{format compact} suppresses many of the blank lines that appear in the output. This lets you view more information on a screen or window.

\textbf{Operators:}

\begin{itemize}
  \item \texttt{+} Addition
  \item \texttt{–} Subtraction
  \item \texttt{*} Multiplication
  \item \texttt{/} Division
  \item \texttt{\textasciicircum} Power
  \item \texttt{'} Transpose + complex conjugate (vectors and matrices)
  \item \texttt{(} Specify evaluation order
\end{itemize}

To find a value of a numerical expression, type it in, using the above operators, and hit ENTER.
For example to multiply 25.4 and 13.6, type 25.4 * 13.6 at the command prompt and press ENTER. The result looks like this:

\begin{verbatim}
>> 25.4*13.6
ans =
   3.4544e+002
\end{verbatim}

The answer is named \texttt{ans} – it is actually a \textit{variable} or symbolic name that can be called to represent the number should we need it later. Instead we can assign the result to our own variable, which we can name whatever we want, for example \texttt{x}. To do this type:

\begin{verbatim}
>> x=25.4*13.6
x =
   3.4544e+002
\end{verbatim}

Now we might want to multiply the result by something else, say by 1.35, and assign it to another variable, for example \texttt{y}:

\begin{verbatim}
>> y=x*1.35
y =
    4.6634e+002
\end{verbatim}

We still have the value of \texttt{x}. To display it, simply type \texttt{x} and ENTER:

\begin{verbatim}
>> x
x =
   3.4544e+002
\end{verbatim}

Suppose we want to try a bunch of operations with two numbers, 2 and 3. We can first assign them to variables \texttt{a} and \texttt{b}:

\begin{verbatim}
>> a=2
a =
\end{verbatim}
>> b=3
b =
3
and then do the operations with the values in a, b.

>> a*b
ans =
6
>> a/b
ans =
6.6667e-001

Note that division automatically turned integers to decimals!

>> a\b
ans =
1.5000e+000
>> a^b
ans =
8
>> a+b
ans =
5
>> a-b
ans =
-1

**Order of Precedence**
The precedence followed in mathematical operations in MATLAB is the same as in standard mathematics:

```
  exponentiation   >  multiplication and division > addition and subtraction
```

except that right division takes precedence over left division!

To override precedence (or if you’re not sure), use parentheses:

>> 2*5^2
ans =
50
>> (2*5)^2
ans =
100
The Assignment Operator

The equals sign “=” does not mean equality, it is an assignment operator. It is an instruction to assign a value to a variable. For example, the following produces error:

```plaintext
g >> x+6=90
??? x+6=90
   |
Error: The expression to the left of the equals sign is not a valid target for an assignment.
```

Assignment can be used recursively to increment or change the value of the same variable.

```plaintext
g >> x=5
   x = 5
 >> x=2*x+3
   x = 13
```

The value on the right hand side is calculated and assigned to the left hand side. If the value of the right hand side was not defined, the assignment could not be made and we get an error:

```plaintext
g >> x=5
   x = 5
 >> x=x+t
??? Undefined function or variable 't'.
```

On the other hand:

```plaintext
g >> x=5
   x = 5
 >> t=7.5
   t = 7.5000e+000
 >> x=x+t
   x = 1.2500e+001
```

Rules for Naming Variables

1) Only letters, numbers and underscore “_” are allowed.
2) No special characters, such as !, @, #, $, %, ^, &, *, <, >, -, +, ., ?, {, }, [, ], ., /, \ etc.
3) No spaces
4) Variable name must start with a letter, not number
5) Names are case sensitive
Suppressing Output

If you simply type a statement and press ENTER, MATLAB automatically displays the results on screen. However, if you end the line with a semicolon, MATLAB performs the computation but does not display any output. This is particularly useful when you generate large matrices.

```
>> x=5;
>> y=12;
>> z=32;
>> (x*y)^2+2*z
ans =
    3664
```

with semicolon you can also include multiple statements on the same line:

```
>> x=5;y=12;z=32;(x*y)^2+2*z
ans =
    3664
```

Entering Long Statements

If a statement does not fit on one line, use an ellipsis (three periods), . . ., followed by ENTER to indicate that the statement continues on the next line. For example,

```
>> s = 1 -1/2 + 1/3 -1/4 + 1/5 - 1/6 + 1/7 ...
    - 1/8 + 1/9 - 1/10 + 1/11 - 1/12;
```

Blank spaces around the =, +, and - signs are optional, but they improve readability.

Command Line Editing

Various arrow and control keys on your keyboard allow you to recall, edit, and reuse statements you have typed earlier. For example, suppose you mistakenly enter

```
rho = (1 + sqrt(5))/2
```

You have misspelled sqrt. MATLAB responds with

```
Undefined function or variable 'sqt'.
```

Instead of retyping the entire line, simply press the † key. The statement you typed is redisplayed. Use the ‰ key to move the cursor over and insert the missing x. Repeated use of the † key recalls earlier lines. Typing a few characters and then the † key finds a previous line that begins with those characters. You can also copy previously executed statements from the Command History.
Characters and Strings
Characters and strings (arrays of characters) are created using single quotes: ‘ ’. For example:

```matlab
>> letter = 'A'
letter =
A
>> name = 'string'
name =
string
```

Clearing Variables
Variables in your workspace and their values appear in the Workspace window.

To get rid of them use command `clear` plus what you want to clear. For example:

```matlab
>> clear x y
```

will clear only x and y, but keep the rest.

```matlab
>> clear all
```

will clear the entire workspace, i.e. delete all the variables. The same can be done from the menu, under Edit clicking on Clear Workspace (and confirm).
Getting Help

MATLAB has extensive documentation and help. Like in any other windows program, you can get help from the menu. If you need help on some specific command, you can type “help command” in the command window:

>> help sqrt
SQRT    Square root.
       SQRT(X) is the square root of the elements of X. Complex
       results are produced if X is not positive.

See also sqrtm, realsqrt, hypot.

Overloaded methods:
codistributed/sqrt

Reference page in Help browser
doc_sqrt

Underlined letters are hyperlings, which will take you to help pages for related commands (sqrtm, realsqrt, hypot) or more detailed documentation on sqrt (doc_sqrt)
**File Basics**

There are several ways to save your workspace into file:

1) From the menu: **File → Save Workspace As**. Specify filename (extension .mat) and hit ENTER. It will save the entire workspace.

2) In the command window type:

```matlab
>> save filename.mat
```

again, it will save the entire workspace into the file named “filename.mat”.

If you want to save only specific variables, you need to use a function form of the save command.

Functions, as we will learn more in detail later, have arguments in parentheses. In this form the filename can also be a variable.

```matlab
>> filename='test.mat';
>> a=10;
>> b=25.5;
>> save(filename,'a','b');
```

Note that all arguments are strings (or characters). To see that a, b really got saved, clear all the variables, load the file and check that a and b have the right values:

```matlab
>> clear all
>> load test.mat
>> a
a =
   10
>> b
b =
   2.5500e+001
>>
```

The **load** command gets back your saved workspace. Like save, load can also be used with arguments, for example to load only specified variables etc. You can always find more in MATLAB help. Another way to load you workspace is from the menu: **File → Open** and then selecting your .mat file.
Often it is useful to save your numbers in a format that you can read or import to other software, in other words in a text file. You can do that using the `save` command with added argument `-ascii`

```matlab
>> filename='test.prn';
>> a=10;
>> b=25.5;
>> save(filename,'a','b','-ascii')
```

Now you can open the file `test.prn` in, for example, Windows Notepad:

![Notepad window](image)

you can see that both numbers are there, but as real type: `a` is no longer an integer.

You can load the file back:

```matlab
>> load('test.prn')
```

but now any information what was `a` and what was `b` is lost. Both numbers are now in a variable (actually an array or vector, because it has two components) called `test` (the filename used as an argument of the `load` function `test.prn` minus the extension `.prn`).

```matlab
>> test
    test =
    1.0000e+001
    2.5500e+001
```

In general, when you use `load` command as `load('filename.ext')` the content will be all in a variable named `filename`. If there is more than one number, it will be a vector or a matrix. If you want it back to `a` and `b`, you need to break it up into its components:

```matlab
>> a=test(1)
a =
    10
>> b=test(2)
b =
    2.5500e+001
```

We will learn how to do manipulations of this kind later when we talk about vectors, arrays and matrices.
**Writing M-files (Scripts)**

Another important type of a file is M-file, so called because it has extension `.m`. M-file is a script that MATLAB can read and execute. Another file with extension `.m` is a function M-file, which we will talk about soon.

To write a script, select File → New → Script. It will open an editor where you can write your script.

An example of a script, which reads two numbers from the input and multiplies them is:

```matlab
%% mscript.m
% reads two numbers and divides them
clear all;
a=input('first number: ');
b=input('second number: ');
if b~=0
    a/b % if not can divide
else
    disp('cannot divide by zero!');
end
```

When you're finished writing your script, save it as `myscript.m`.

*It is important that the folder where your script is saved is defined in MATLAB Path!* You can add a folder to the Path from the menu: File → SetPath, find the folder, click Add Folder and Save.

To execute, go to command window and type `myscript` at the command prompt. It will ask you for a first number (don’t forget to hit ENTER after you type it in), then second number. Here is what happens if the second number is zero:

```matlab
>> myscript
first number:2
second number:0
cannot divide by zero!
```

and if it is not:

```matlab
>> myscript
first number:2
second number:3
ans =
   6.6667e-001
```
you get the result of dividing the two numbers.
There are a lot of things one can learn about MATLAB programming from this simple script:

1) The percent sign % denotes comments, in your MATLAB editor they appear green. Comments are ignored by MATLAB, but are very useful for adding notes, explanations etc. that help read and understand the code. You can put comments wherever you want in your program. In particular the comments right at the top (until the first “non-comment” line) show up when you call help on your script:

```matlab
>> help myscript
myscript.m
  reads two numbers and divides them
```
You can take advantage of this and put in a simple description of your script.

2) clear all

Clears all the variables that might have been assigned at the time you execute your script. It is a good practice for preventing bugs.

3) Function input()

R = input('text string') gives the prompt in the text string and then waits for input from the keyboard (until ENTER is pressed). The input have to be numerical (i.e. not character or string) but in general can be any MATLAB expression, which is evaluated, using the variables in the current workspace, and the result returned in R. If the user presses the return key without entering anything, INPUT returns an empty matrix.

If you want to input character or string, you have to use R = input('text string','s'), which, again, gives the prompt in the text string and waits for character string input. The typed input is not evaluated; the characters are simply returned as a MATLAB string.

4) if-else-end structure

This structure conditionally executes statements. The statement (or statements - there may be more than one) below if (in this case a/b) is executed if the condition right after the if keyword (b~0, meaning b is not equal zero) is TRUE. Otherwise, the statements under else are executed.

The general form is

```matlab
if expression1
  statements1
elseif expression2
  statements2
else
  statements3
end
```

The statements1 are executed if the logical expression1 is TRUE, i.e. equal to logical 1. If it is not, the expression 2 at elseif is tested etc. The elseif and else parts are optional. Zero or more elseif parts can be used as well as nested if's.
The expressions are usually of the form \( a \ rop \ b \) where \( rop \) is one of the \textit{relational operators}:

\[
\begin{align*}
== & \quad \text{(two equal signs) is equal (TRUE if } a \text{ and } b \text{ are equal)} \\
< & \quad \text{(TRUE if } a \text{ is numerically less than } b) \\
> & \quad \text{(TRUE if } a \text{ is numerically greater than } b) \\
\leq & \quad \text{(TRUE if } a \text{ is numerically less or equal to } b) \\
\geq & \quad \text{(TRUE if } a \text{ is greater or equal to } b) \\
\neq & \quad \text{(TRUE if } a \text{ is not equal to } b) \\
\end{align*}
\]

More complex logical expressions may include \textit{logical operators}:

\[
\begin{align*}
\&\& & \quad \text{logical AND} \\
\|\| & \quad \text{logical OR} \\
\sim & \quad \text{logical NOT (as above \neq \ means NOT equal)}
\end{align*}
\]

For example:

\[
\begin{align*}
\text{if } a & > b \ \&\& \ a & > c \quad \% \text{ if } a > b \text{ OR } c > d \\
& \quad \text{disp('a is greater than b and c')} \\
\text{elseif } a & > b \ \&\& \ a & < c \\
& \quad \text{disp('a is greater than b but c is greater than a')} \\
\text{elseif } a & < b \ \&\& \ a & > c \\
& \quad \text{disp('a is greater than c but b is greater than a')} \\
\text{else} \\
& \quad \text{disp('b and c are both greater than a')}
\end{align*}
\]

Be careful with complex logical expressions: relational and logical operators, just like arithmetic operators have different priorities. Use parentheses if not sure.

5) \textbf{Function } \textit{disp()}

\textit{disp(a)} \ displays the value of variable \( a \), without printing its name. \\
\textit{disp('string')} \ displays string.

With these basic commands you will be able to write scripts that can tackle most MATLAB problems. They will include more complex types of variables (arrays, vectors, matrices), functions (MATLAB standard ones and your own) programmed to do specific tasks, you will learn how to create graphs and plots etc., but the actual programming will not get much more difficult than this.