3 Some basics of using Matlab

To begin, you can use Matlab for simple arithmetic problems. Symbols like + (plus), − (minus), * (multiply), and / (divide) all work as you would expect. In addition, ^ is used for exponentiation.

For example, if you type

```
>> 75 - 32 * 2 + 4 / 2
```

in the Command Window (note that the >> is added by Matlab[2]) and press `<Enter>`, Matlab calculates the value of the expression and responds with

```
ans =
     13
```

There are a few things here to note. First, when you enter a command to calculate a value, Matlab performs the calculations and puts the result in a variable named `ans` (for answer). If you check the Workspace, there is now a variable listed there: `ans`. Second, by default Matlab prints the value of the variable in the Command Window in two rows. In the first row, we see the variable name, `ans` in this case, and an equality sign, and in the second, we see the value of the variable, which in this case is `13`. The value is indented by a few blank spaces.

3.1 The order of precedence

It is important to note the order in which algebraic expressions are calculated. This is called the order of precedence, or the order of operations. As you probably expect, multiplication and division are calculated before any addition or subtraction. In the expression above, this means that `32*2=64` is calculated first, `4/2=2` is calculated second, and then `75-64+2=13` is calculated last.

In general, the order of precedence is

1. parentheses
2. exponentiation
3. multiplication, division
4. addition, subtraction

If the list is not enough to determine the order in which to calculate an expression, the calculations are done from left to right. For example, if you enter `10/20/10`, Matlab first calculates `10/20=0.5` and then `0.5/10=0.05`. It does *not* start by calculating `20/10=2` and then `10/2=5`. If you want the latter, you have to make this clear by adding parentheses: `10/(20/10)=5`. If you try it in Matlab, you get

```
>> 10/20/10
ans =
    0.05
```

and

```
>> 10/(20/10)
ans =
    5
```

If you are unsure, always add parentheses. That also makes many expressions easier to read. Writing

```
>> 75 - (32 * 2) + (4 / 2)
ans =
    13
```

does not change the outcome, but you could argue that it is somewhat easier to read. As expressions become more complex, the difference becomes more pronounced.

3.2 Some algebraic functions, special characters, and tips

In Matlab, you usually have to enter all instructions for what you want the program to do as commands. This means that it is necessary to know some basic commands and syntax.

A few characters have special meaning and are used frequently.

`,`	Comma This can be used for separating commands that are written on the same line. You can, for example, type `sqrt(4),sqrt(9)` to have Matlab calculate both values at once.
`;`	Semicolon This suppresses output when commands are executed. For example, if you type `9+5;` (including the semicolon) and press `<Enter>`, the expression is evaluated and saved in the `ans` variable (which you can check in the Workspace). However, nothing is displayed in the Command Window. This is useful when you run programs and do not want intermediary calculations to be displayed. Note: If you separate commands with `;` you do not have to use `,` as well.
`%`	Comments Everything after the %-character until the end of the line is ignored. This is useful for adding comments in programs but is seldom used in the Command Window.

Basic algebraic functions include

`sqrt(9)`	Square root of 9.
`3^4`	Exponentiation 3^4; 3 to the power of 4 (i.e., 3*3*3*3).
`exp(4)`	e^4; exponentiation with base e (≈ 2.7183).
`log(5)`	The natural logarithm of 5 (note: with base e, *not* with base 10).
`log10(5)`	The logarithm of 5, using base 10.
`abs(-3)`	The absolute value of -3.
`round(1.3)`	Round to the nearest integer.
`floor(1.3)`	Round to the nearest smaller integer (i.e., towards minus infinity).
`ceil(1.3)`	Round to the nearest larger integer (i.e., towards plus infinity).
`fix(1.3)`	Round to the nearest integer towards zero (i.e., downwards for positive numbers and upwards for negative).
`sign(1)`	The sign of 1: -1 if < 0; 1 if > 0; and 0 if = 0.

Note that you do not have to use functions one at a time. Instead, you can nest them. If you, for example, want to calculate the absolute value of `-4` to the power of `3`, and then round that off to the nearest integer, you can issue

```
>> round(abs(-4)^3)
ans =
     64
```

The calculations start in the innermost pair of parentheses and move outwards.

In addition, you will probably want to use the following

`help`	We will discuss how to find help later (see Section 13). Here, it is enough to note that you can get help on any function by typing `help` and the function name. For example, `help sign` will produce a short text on how the `sign` function works in Matlab. In addition, you get clickable links to related functions and a clickable link to the corresponding reference page in the documentation.
`clc`	Clears the Command Window. Only the window is cleared, no variables or anything else is changed. This is good for getting rid of clutter.
`clear`	Deletes variables. `clear` alone deletes all variables, `clear` and one or several variable names, for example `clear abc`, deletes only the named variable(s).
↑	Pressing this arrow-key (typically located on the lower right side of the keyboard) brings back the previous command lines one-by-one. If you type one or several characters first and then press the arrow key, Matlab skips to those lines that begin with the same characters. For example, if you type `75` and press ↑, you get back the line `75 - (32*2) + (4/2)` if you entered it earlier.
`format`	Changes the way that output is displayed in the Command Window. Note: it does not change the internal calculations or the value of the variable, only how output is displayed in the Command Window. `format loose` adds extra lines between output and `format compact` suppresses extra lines. `format long g` displays numbers using 15 digits and `format short g` displays them using 5 digits. For example, `1/3` is displayed as `0.333333333333333` in the first case and as `0.33333` in the second. Try `help format` to get a few more tips on how to use this command.

3.3 The syntax of functions

Most functions in Matlab have a common form. They take one or several arguments as input and they produce one or several output arguments. The general syntax of a function is

```
[output1, output2, …] = function_name(input1, input2, …)
```

Input arguments are enclosed within parentheses, and if there are more than one argument, they must be separated by commas. If there is only one output argument, no square brackets are needed, but if there are more than one they must be added. Output arguments can be separated by commas or by blank space. Note that function names are almost always in lower case.

Some functions, such as plus and minus, have alternative syntaxes. For plus, for example, you can issue either `2+3` or `plus(2,3)` (see also sections 5.1, 5.3.2, and 5.3.3).

Furthermore, certain functions work differently depending on how many input arguments you enter or how many output arguments you request. See Section 4.1.1 for an example using the function `diag()` or Section 4.4 for an example using the function `find()`.

3.4 Variables

Earlier we calculated

```
>> 10/20/10
ans =
     0.05
```

Let us go through what happens when this command is issued, step-by-step. We enter the expression `10/20/10`. Matlab calculates that the value of this is `0.05`. Then it defines a variable with the name `ans` and sets the value of this variable equal to the answer (i.e., equal to `0.05` in this case). If a variable named `ans` already exists, the value is changed to the answer. Then it prints the variable name, `ans`, in the Command Window followed by an equality sign and, finally, the value, `0.05`, in the next row. The latter part, the printing in the Command Window, would have been omitted if we had ended the expression, `10/20/10`, with a semicolon.

This process is a special case of defining a variable that occurs when the user has not specified a variable name. If, instead, you enter `abc = 10/20/10` the process is the same, but instead of defining a variable named `ans` Matlab creates a variable named `abc`, assigns the value of the answer to this variable and prints the information in the Command Window, so that what is seen there is

```
>> abc = 10/20/10
abc =
     0.05
```

Since the equality sign, in this context, tells Matlab to *assign* a value to a variable, it is called *the assignment operator*. The command is interpreted as "set the variable…equal to…" After you have issued the command above, `abc` has been assigned a value of `0.05`. Checking the Workspace, you can confirm that there is now a variable named `abc` listed there.

When a variable has been defined, you can ask for its value by entering the name:

```
>> abc
abc =
 0.05
```

Variables can also be reassigned. The variable `abc` has the value `0.05`, but if you type `abc = 3` the old value is replaced by the new one. You can also reassign variables using other variables or even using the variable itself. So, to double the value of `abc` (which is now `3`), you can enter

```
>> abc = abc*2
abc =
     6
```

Variable names must begin with a letter and can be at most 63 characters long.[3] It is a very good idea to use names that describe what the variables represent. Multi-word names are more readable if you use capital letters or underscores to indicate new words, for example `interestRate` or `exchange_rate`. Note that Matlab is case sensitive, so `c` and `C` are two different variables, as are `interestRate` and `interestrate`.

3.4.1 Predefined variables

A few variables are predefined in Matlab.

`ans`	"answer" A variable that holds the (latest) value that has been calculated with no specified variable name. If this variable is deleted (for example by entering `clear ans`) it reappears as soon as you perform a new calculation with no specified variable name.
`pi`	The constant $\pi = 3.141592653589793\ldots$
`inf`	Infinity Rather than infinity this means "overflow". This can be the output of a calculation. For example, `1/0` gives the result `inf`. Note that you can also use it in calculations. For example, `3*inf` is `inf`.
`NaN`	Not-a-number This is used when it is not possible to record a valid numerical value, for example when you have missing observations in a data set. It can also be used in calculations or be the outcome of a calculation. `NaN*2` and `0/0` both resolve to `NaN`.

3.5 Different types of variables

Variables can be of different types. We need to distinguish between numerical, character, and logical variables. Later, we will also briefly present cell variables.

3.5. 1 Numerical and character variables

It is important to note that there are different *types* of variables. `abc` above is a *numerical* variable. However, variables can also hold one or several characters. To tell Matlab that you want it to interpret something as a string of characters, you enclose it within single quotes (`'`). Enter, for example, the text `'def'` within single quotes in the Command Window:

```
>> 'def'
ans =
def
```

Part of the text is color coded violet. This is to highlight that this part is a character string.[4] Matlab interprets your command as though you want to evaluate a string of characters. And that produces the same string of characters, `def`, which is then assigned to the variable `ans` and printed in the Command Window. If you check the Workspace, you can see that the `ans` variable is now labeled "`char`" in the class column to indicate that it is a character variable. Numerical variables are usually labeled "`double`" (referring to how numbers are stored internally).

Consequently, it is very different to enter `123` and to enter `'123'`. If we enter both, separating them with a comma, we get

```
>> 123, '123'
ans =
    123
ans =
123
```

The two answers look deceivingly similar. The main difference is that the first `123` has a few white spaces in front of it, so that it is located a bit off from the left side. The answers are, however, completely different. In the first case, `123` is interpreted as a number and evaluated as such. The result is, naturally, the number `123`. When numerical values are printed to the Command Window, a few spaces are inserted to the left. You can see this in all the preceding examples where numbers were printed.

In the second case, `'123'`, is interpreted as a string of characters. The fact that a string of `1`, `2` and `3` can be interpreted as a number is ignored. The string of characters is evaluated and the result is just the same string of characters. The result is assigned to the variable `ans` and printed to the Command Window. However, as it is not a number, no white spaces are inserted in front of it. The string is therefore printed immediately next to the window edge.

There are convenient ways to translate numerical variables to character strings and vice versa.

`str2num`	Converts a string of characters to a number, given that Matlab can interpret the string as a number. For example, `str2num('542')` is `542`.
`num2str`	Converts a number to a string of characters. For example, `num2str(542)` is the string `'542'`.

Sometimes, for instance when adding a title to a graph or when printing information to the Command Window, it is useful to mix numbers and text. In such cases, numbers must first be “translated” into character strings. This is described in Section 8.4.

Strings have several different functions in Matlab. They are frequently used to display messages and in addresses to files on the computer, for instance when importing data. Some functions also take string input arguments. You can also enter formatting commands for plots as strings.

3.5.2 Logical variables

Logical variables are a type of variable that can only take the values *true* and *false*. In Matlab, true is represented as `1` and false is represented as `0`. Superficially, logical variables therefore look like ordinary numerical variables. Logical variables are usually the outcome of relational or logical statements. Suppose we take the variable `abc` (that equals `6`) from earlier and issue the statement

```
>> abc > 5
ans =
     1
```

The answer is 1, meaning true, since it is true that abc is greater than 5. Note now that, in this case ans is not an ordinary numerical variable. Check the Workspace to see that the class of ans is "logical". Does this matter? The answer is that, sometimes it does, sometimes it does not. The variable ans can still be used in algebraic expressions. Then it is treated as an ordinary zero or one, depending on its value. If we issue

```
>> ans*1
ans =
     1
```

the variable looks unchanged (i.e., its value is still 1). If, however, we check the Workspace, we see that the class has changed to "double", indicating that it is now an ordinary numerical variable. In algebraic expressions, it does not matter whether the variable is logical or numerical. However, we will later see that it does matter when the variable is used for addressing purposes (see Section 4.2.3).

3.6 A note on interpretation and error messages

It is important to understand that Matlab *interprets* everything that you enter into the Command Window. To get the response that you want, you have to enter commands that Matlab interprets in the way that you mean them. It is not unusual that the user enters commands that seem obvious to him or her, but gets a response that seems weird or gets an error message. If you, for example, enter def (without single quotes), and have not defined this as a variable, Matlab is unable to interpret your command and responds by issuing an error message:

```
??? Undefined function or variable 'def'.
```

As you can see from the message, Matlab tried to interpret what you entered either as a function or as a variable, but was unable to find any interpretation for it as either. Error messages are, by default, color-coded red.

The error messages are often informative and you can use them to understand what it is that Matlab does not understand. However, this is not always so. If you, for example, entered def (forgetting the single quotes) and wanted Matlab to interpret this as a string of characters, the error message above is a bit perplexing. It just tells you that Matlab was not able to understand your command either as a function or as a variable. In such a case, you can often understand the message as though you performed a more fundamental error: You have entered a command that to Matlab looks like a function or like a variable. If that was not what you wanted, you have to look for a different way to enter your command. In this case, by enclosing the def within single quotes.

3.7 How Matlab "searches for meaning"

One curious feature of Matlab is that it is possible to use a function name as a variable name. For example, `sqrt()` is a function for calculating the square root:

```
>> sqrt(4)
ans =
      2
```

Suppose that you want to assign this answer, `2`, to a variable and that you think that `sqrt` is a good name.

```
>> sqrt = sqrt(4)
sqrt =
      2
```

This is perfectly ok, although not a good practice. What happens if you do this is that the variable takes precedence over the function, which, consequently, cannot be used any more. Entering the following commands sheds some light on this:

```
>> sqrt, sqrt(1), sqrt(4)

sqrt =
     2
ans =
     2
??? Index exceeds matrix dimensions.
```

In the first case, `sqrt` produces `sqrt=2` which is the value of the variable `sqrt`. In the second case, `sqrt(1)` produces `ans=2`, which is not the answer we want, if we want to calculate the square root of `1`. In the third case, `sqrt(4)` produces a counterintuitive error message.

The strange results are all consequences of the fact that `sqrt` is now a variable, not a function. In the first case, Matlab interprets the command `sqrt` as if you are asking for the value of the variable, which is `2`. In the second case, it interprets it as if you are asking for the first element of the variable `sqrt`, which is also `2`. In the next section, where we talk about matrices, we will see that a variable can hold more than one element. In the third case, Matlab interprets the command as if you are asking for the value of the fourth element of the variable `sqrt`. However, `sqrt` has only one element, so you get an error message.

To be able to use the function again, you must first delete the variable. To do that from the Command Window, use the command `clear` (see Section 3.2).

```
>> clear sqrt
```

It is also possible to clear specific variables by selecting them in the Workspace and pressing `<Delete>` or by right-clicking the variable and choosing `delete` in the context menu.

It is, of course, best to avoid using function names as variable names. If you are unsure if a certain name is already in use, just issue the command `exist` and the name. If it is not in use, Matlab will respond with `0`. Then you know that it is ok to use that name for a variable.

So, why does the variable take precedence over the function? This has to do with the order in which Matlab "searches for meaning". When a command is issued, Matlab first searches the Workspace to see if there is a variable by that name there. If there is, the program returns the value of the variable. If no variable is found, Matlab searches the Current Folder (see Section 2 and Section 6.1). If a function file is found there, it is selected and executed. If, on the other hand, nothing is found in the Current Folder either, Matlab starts searching in folders along a predefined search path (see Section 8.3) to see if a function can be found there. Again, if one is found, it is selected and executed. If nothing is found along the path either, Matlab gives up and issues an error message. This is why the first error message we encountered was that the program could not find a defined variable or function.