Contents

VARIABLES

Variables are used to store data (numbers, letters, etc) in MATLAB. There are a few rules that must be followed when creating variables in MATLAB:

- Variable names must begin with a letter and can be followed with any combination of letters, numbers, or underscores only – no special characters are allowed in variable names.
- Variable names must not exceed 63 characters in length.
- Variable names are case sensitive. This means that the variable `j` is not the same as the variable `J` and the variable `Apple` is not the same as the variable `apple`.

Examples of acceptable variables: `a`, `b`, `c`, `apple`, `Apple`, `ApPLE89_3`, `ILIKECHEESE`
Examples of unacceptable variables: `89`, `8fish`, `Fis h`, `pizza!`

You should choose variable names that are descriptive and provide some indication of the value they are storing. For example, use the variable `numapple` to store information about the number of apples, or something like `Velocity_x` or `V_x` to store information about velocity in the x-direction.

You can check if a variable name is allowed with the `isvarname` command. This is just one of many built-in commands in MATLAB. Note that `1 = Yes` and `0 = No` in this case. (The number `1` actually corresponds to "true" in certain situations in MATLAB, and `0` corresponds to "false". This will be covered in detail later in the course when discussing logical operators and `if` statements.)

```
>> isvarname pizza

ans =

     1
```

```
>> isvarname pizza!

ans =

     0
```

In MATLAB, unlike other languages such as FORTRAN, we do not need to declare the data type of variables (integer, real, character, etc). All variables in MATLAB are assumed to store floating point data (e.g. real, decimal numbers) unless otherwise stated (there will be times where we will want to store letters instead of numbers, or force a variable to be an integer). In fact, you do not even need to declare variables before assigning them a value for the first time. They are automatically declared and created when you assign them a value. In MATLAB, the equals sign (=) means "assign the value on the right to the variable on the left" not "equals" – this is a very important concept to understand in programming and allows us to perform unique operations (such as recursive relationships, discussed later).

Storing Numerical Data

```
>> b = 2

b =

     2
```

The value on the right side of the equals sign is assigned to the variable `b`. Think of `b` as a box (in your computer's memory) that can hold values (numbers, characters, etc). Once you have defined `b`, it is now stored in memory (see the *Workspace* on the right hand side of your MATLAB environment). You can now use the stored value of `b` to perform calculations.

```
>> b + 5

ans =

     7

>> b * 5

ans =

    10
```

Once it is stored in memory, you can use the value of `b` to calculate the value of another variable, `c`.

```
>> c = b + 5

c =

     7
```

Here, `2 + 5` is evaluated (the right hand side is evaluated first), and the resulting value is assigned to `c` (the variable on the left hand side of the equals sign).

You can use the value of multiple variables to get the value of another variable, `cow`.

```
>> cow = b * c

cow =

    14
```

In this example, `2 * 7` is evaluated, and the answer is assigned to `cow`. I emphasize that the right side of the equals sign is evaluated first, then the result is assigned to the variable on the left side. This allows us to use what is called a recursive relationship.

```
>> cow = cow + 1

cow =

    15
```

This equation may appear wrong algebraically, but computationally it is fine. The right side (`14+1`) is evaluated first (`14+1` is `15`), then assigned to `cow`. Note that if you re-use a variable name and assign it a new value using the equals sign, you have replaced the old value the variable had previously. Thus, the variable `cow` stores the value `15` after the above operation has been executed.

You can check out what variables have been created with the `who` command (and also by looking in the *Workspace*). This built-in MATLAB command lists all the variables currently stored in memory (which can be used or accessed for additional calculations).

```
>> who

Your variables are:

ans  b    c    cow
```

Note that `ans` is a reserved variable in MATLAB – it is assigned a value if you perform a calculation but do not define a variable on the left hand side of the equals sign. You can get more detailed information about the variables with the `whos` command. This shows the variables in your workspace, what type (class) they are, how much memory they are taking, their size, and other attributes.

```
>> whos
  Name         Size                Bytes  Class      Attributes

  ans          1x1                     8  double
  b            1x1                     8  double
  c            1x1                     8  double
  cow          1x1                     8  double
```

There is a lot of information here. We will learn more about the additional details as the course goes on. The concept of variable scope will become important when we learn about functions. For now, just notice how all the variables are considered 1x1 arrays that are storing double precision data (that is, all variables contain floating-point numbers).

You can suppress output to the *command window* by ending any expression with a semi-colon "`;`". Note that this just prevents the result from being displayed into the *command window* – the calculation is still performed and the result is still stored in memory.

```
>> d = 2 * c

d =

    14

>> e = 2 * c;
```

Note that the variable `e` is still stored in memory. We can access it and check its value by just typing its variable name in the command line.

```
>> e

e =

    14
```

What if you try to use a variable that has not been defined yet?

```
>> y * 2
Undefined function or variable 'y'.
```

MATLAB cannot perform the calculation because the variable `y` does not yet exist. Therefore, MATLAB stops and gives an error message (and sound if enabled). We can assign `y` a value and perform the same calculation without an error.

```
>> y = 9

y =

     9

>> y * 2

ans =

    18
```

In MATLAB **variable names are case sensitive**. This means that an uppercase letter is distinct from a lowercase letter.

```
>> t = 0.5

t =

    0.5000

>> z = 2 * T
Undefined function or variable 'T'.

Did you mean:
>> z = 2 * t

z =

     1
```

Note that using `T` returned an error since the variable `t` is different from the variable `T`. MATLAB even suggested a correction for the error, thinking you may have inadvertently made a typo.

Variables will exist in memory until you exit MATLAB or erase them with the `clear` command.

```
>> who

Your variables are:

ans  b    c    cow  d    e    t    y    z
```

```
>> whos
  Name       Size                Bytes  Class     Attributes

  ans        1x1                     8  double
  b          1x1                     8  double
  c          1x1                     8  double
  cow        1x1                     8  double
  d          1x1                     8  double
  e          1x1                     8  double
  t          1x1                     8  double
  y          1x1                     8  double
  z          1x1                     8  double

>> clear
>> who
>> whos
```

Nothing appears since all the variables are gone after the `clear` command – the workspace is now empty. There is no way to recover variables that have been cleared from memory.

If your screen gets too messy or cluttered with output, you can use the `clc` command to clear the screen (*command window*). Note that this does **not** clear (erase) variables from memory.

Limits on Numerical Data

There is a limit to the size of the numbers you can store in memory in MATLAB. There are built-in commands to output these limits:

```
>> intmax

ans =

  2147483647

>> intmin

ans =

 -2147483648

>> realmax

ans =

  1.7977e+308
```

```
>> realmin

ans =

  2.2251e-308


>> eps

ans =

   2.2204e-16
```

Largest double-precision floating-point number: `1.7977e+308`
Smallest double-precision floating-point number: `2.2251e-308`
Largest integer: `2147483647`
Smallest integer: `-2147483648`
Smallest step size between double-precision floating-point numbers: `2.2204e-16`

Sometimes you run into problems when you do mathematical operations with very large and very small numbers. For example:

```
>> x = 1e200

x =

  1.0000e+200

>> y = 1e-200

y =

  1.0000e-200

>> z = x / y

z =

   Inf                    (The true answer is 1.0000e+400)

>> a = y / x

a =

     0                    (The true answer is 1.0000e-400)
```

```
>> b = (y / x) * x
```
(The true answer is `1.0000e-200`)

```
b =

     0
```

These numbers go beyond the limits of what MATLAB can handle as defined previously.

Storing Character Strings

Besides storing numbers (e.g., integers and decimals), you can store character data in variables too. In both cases, you still use the equals sign to assign the value on the right hand side of the equals sign to the variable on the left hand side of the equals sign. Character data can be letters, special characters, and even numbers. In MATLAB you enclose any text that you want to store as a character variable in single quotes. You **must** put single quotes around the text so that MATLAB recognizes it as a character string. This is very important – if you do not put quotes around a character string, MATLAB will search for a variable with that name, rather than recognizing it as a string of characters (e.g. a word, name, etc.) that you want to assign to a variable.

Let's look at some examples of how we can assign character strings to variables.

```
>> e = 'banana'

e =

banana
```
(The variable `e` stores the word "`banana`")

```
>> f = 'chimp'

f =

chimp
```
(The variable `f` stores the word "`chimp`")

```
>> g = chimp
Undefined function or variable 'chimp'.
```
(There is no variable called `chimp`)

Notice in the third example above, we did not enclose the word `chimp` in single quotes. Therefore, MATLAB did not recognize it as a character string. Instead, MATLAB attempted to search for a variable named `chimp`, which did not exist. Therefore, an error message was produced.

```
>> whos
  Name       Size            Bytes  Class     Attributes

  e          1x6                12  char
  f          1x5                10  char
```

Notice that `e` and `f` are character variables, a different class of variable than those storing numerical data. The variable arrays are also now bigger than `1x1`. Each letter is stored in a separate space in memory and takes one element of the array – since `banana` has six letters, `e` is a `1x6` character array. We will discuss this in more detail in the future.

Logical Variables

We will learn about this type of variable later in the context of relational and logical operators and `if` statements. For now, just be aware that it exists and is different from floating point numbers and character arrays.

MATLAB'S BUILT-IN VARIABLES AND FUNCTIONS

MATLAB contains a large library of commonly used variables and functions. For example, the variable name `pi` is reserved for π (`3.141592654`…). Note that it is possible for you to over-write a built-in variable and replace it with a value that you assign, but this is not a good idea. Using `clear` to clear memory will revert these variable back to their default, built-in values. We can access or use these built-in variables without having to define them or assign values to them.

```
>> pi

ans =

    3.1416
```

(many more digits are stored, but only four digits right of the decimal are shown)

```
>> z = 3 * pi

z =

    9.4248

>> degrees = 90

degrees =

    90
```

```
>> radians = degrees * (pi / 180)

radians =

    1.5708
```

There are many different types of functions in MATLAB. Almost all functions are given one or more arguments (the input sent into the function to evaluate the output). For example, the trigonometric function `sin(x)` has one argument, `x`, which must store numerical data. The arguments of trigonometric functions in MATLAB are assumed to be in radians – be sure to convert from degrees to radians before using `sin()`, `cos()`, etc.

```
>> sin(0)

ans =

     0

>> cos(0)

ans =

     1

>> sin(pi)

ans =

   1.2246e-16

>> cos(pi)

ans =

    -1
```

Note that in this example, `sin(pi)` does not exactly equal to zero. This brings up a very important limitation of computers – they cannot store an infinite amount of information. That is, `pi` is an irrational number and has an infinite number of digits after the decimal. Since only a finite amount of digits are stored in MATLAB, the sine of `pi's` value that is stored in MATLAB is not *exactly* zero. Please see the notes about numerical (rounding) errors for additional information.

Some other commonly used built-in MATLAB functions:

`sqrt(x)`	calculates √x
`exp(x)`	calculates e^x
`abs(x)`	calculates |x|

See MATLAB's help documentation or the textbook for a longer list of built-in functions.

MATLAB has many functions that allow you to do common tasks with a single command. Other languages, such as FORTRAN and C, may not have these built-in functions and are less user friendly in some respects. (However, the price you pay for this user-friendly behavior in MATLAB is that it typically executes calculations slower than equivalent FORTRAN or C codes.) For example, in MATLAB, `mean(x)` finds the average value of one or more numbers stored in `x`. However, in this class, we will not be using many of these functions because it is important that you understand the thinking and calculations behind the functions and not just how to plug in numbers.

It is possible to use built-in function or built-in variable names as names for your variables. This is a **bad idea** because it may confuse the user, and it essentially "breaks" the function until you clear that variable from memory.

```
>> pi

ans =

    3.1416

>> 3*pi

ans =

    9.4248

>> pi = 7

pi =

     7

>> 3 * pi

ans =

    21
```

```
>> sin(4)

ans =

   -0.7568

>> sin = 2

sin =

     2

>> sin(4)
Index exceeds matrix dimensions.
```

In these examples, we overwrote the variable `pi` with a value of `7`, and assigned a numerical value to the variable name `sin`, which prevented the `sin()` function from working in the next command. Once again you should never use built-in variable or function names to name your variables.

There are also certain keywords that are not allowed to be used for variable names. Use the command `iskeyword` to see the keywords reserved in MATLAB.

```
>> iskeyword

ans =

    'break'
    'case'
    'catch'
    'classdef'
    'continue'
    'else'
    'elseif'
    'end'
    'for'
    'function'
    'global'
    'if'
    'otherwise'
    'parfor'
    'persistent'
    'return'
    'spmd'
    'switch'
    'try'
    'while'
```

MATLAB will not allow you to create a variable with any of the above names.

GETTING HELP IN MATLAB

MATLAB has a lot of documentation built-in that can help you if you need more information. For example, if you need help with the `sin()` function, you can use the `help` command in the *command window*.

```
>> help sin
 sin    Sine of argument in radians.
    sin(X) is the sine of the elements of X.

    See also asin, sind.

    Reference page for sin
    Other functions named sin
```

There is **a lot** of information in the *search documentation* area in the upper right hand corner and on the official MathWorks website. There are countless tutorials online (videos, walkthroughs, etc.) as well. Google is also commonly used when searching for additional MATLAB help on specific commands or when looking for examples from other sources.

CREATING M-FILES

So far we have only been executing simple commands at the command line inside the *command window*. What if you want to make a small change to a code that has a lot of commands without having to type those commands over again every time you want to execute your code?

You can create an "M-File" – a simple ASCII text file that ends with a .m extension. You can create or edit M-Files using external text editors like Notepad++ and Emacs, or within MATLAB's built-in text editor in the *Editor* window. You can create or open a text file at the command line using the `edit` command. (For this class, please use the MATLAB's built-in editor for creating and modifying your M-Files. Your files **MUST** have a .m extension when submitting your homework.)

```
>> edit myfile.m
```

You will be prompted to create the file if it does not exist, and you should see the *Editor* window open above the *command window* inside of MATLAB. Commands are executed sequentially from line 1, line 2, line 3, etc., in the same exact way as if you typed them in the *command window*. For example, a basic M-File file might look like this:

In `myfile.m`:

```
clear;
x = 5;
y = 7
c = x*y
```

Be sure to save your M-File after making any changes (which you created with the name `myfile`, with the .m extension). To run your M-File, type the filename without the .m extension at the command line, or click the *Run* button above the *Editor* window (the green arrow).

Sample Output:

```
>> myfile  (notice we don't include the .m extension here although the file has it)
           (x = 5 is suppressed due to the use of a semi-colon)
y =

     7


c =

    35
```

You may open and edit your M-File later if needed by either using the `edit` command again or simply double clicking the file from the *Current Folder* window or your computers file browser. You can have multiple M-Files open at once in the *Editor* and they will each have their own tab (similar to a web browser).

It is a good idea to use the `clear` command at the beginning of your M-File. This will remove all variables from memory and prevent unwanted interactions and/or odd behavior due to variables left in memory from other codes executed previously.

COMMENTS IN MATLAB

The percent symbol (`%`) is used to create commentary in MATLAB. Comments are completely ignored by MATLAB, and can contain any characters you like (letters, words, numbers, symbols, etc.).

There are two main reasons you should heavily comment your codes:

1. Another programmer may need to use/edit your code and will need to understand what is going on. Comments allow you to leave notes to yourself and/or other users.
2. When your code becomes long or you work on it over a long period of time, it becomes increasingly difficult to remember exactly what you were thinking when you wrote a certain line. However, you still need to remember exactly what each line of code is doing, what data a

given variable is storing, what units are being used, etc. Anything after the `%` will be ignored by MATLAB.

It is very important that you develop the habit of adding descriptive comments into your code early on. Let's look at the same sample code as before but with some comments added:

In `myfile.m`:

```
%My Code - Practicing MATLAB          (nothing will happen from this line)
clear;     % clear old variables from memory
x = 5;     % x-position in [m]
y = 7      % y-position in [m]
c = x * y
```

Sample output:

```
>> myfile

y =

     7


c =

    35
```

Here, the variable `x` is assigned a value but it is not printed due to the semicolon suppressing output to the *command window*. The text `%My Code - Practicing MATLAB` is ignored by the program, as are all the other comments preceded by the `%` sign. Note that the output is the same as before we added the comments – the comments are only there to help the programmer.

It is always a good idea to add lots of comments to you code to document the variables you used (e.g., units for numbers) and the logic behind the operations. For the purposes of this class, comments can also be used as a header and should include your name and ID#, the homework and problem number, a brief problem statement, etc.

Comments can also be used when testing your code or trying to find which line of code is causing an error ("debugging"). For example, if you want to disable a certain line of code, simply add a `%` sign as the first character in that line. This will turn the line into a comment, which will be ignored by MATLAB. When used to debug your code (find errors), comments can be used in the following way. Starting at the bottom (last line) of your code, you can start commenting out lines one or two at a time. After you comment out a line or two, try and run the code again. If it runs without error, one of the lines you just commented out was the source of the error. You can then examine that line in detail to find the source of the error. (When an error occurs, MATLAB attempts to pinpoint the exact line number that causes the error. However, it is not always accurate and the error may originate from an earlier line of code.)