Introduction to MATLAB

Leigh J. Little SUNY Brockport August 31, 2022

Contents

1 Introduction

MATLAB is short for Matrix Laboratory. The first versions were written in the early 1990's by Cleve Moler from the Massachusetts Institute of Technology. The original purpose was to allow for easier calculation of quantities involving vectors and matrices. In particular, it was designed to compute solutions to $n \times n$ systems of linear equations such as the 3×3 system

$$\begin{aligned} u + v + w &= 3 \\ 2u + v + w &= 0 \\ 2u + v - w &= 1. \end{aligned}$$

A set of equations such as the one above can be solved by hand, but the process is tedious and error prone. In something like a weather simulation, such systems can easily involve one million equations in one million variables.

Over time, more and more programming features were added to MATLAB. Today MATLAB has the programming capabilities consistent with most other languages. What makes it an attractive computing environment is the ability to jump right in and start computing without significant amounts of overhead.

2 The MATLAB Environment

MATLAB makes working with vectors and matrices easy, but we will start by writing programs that focus on simple calculations involving scalar quantities. We will introduce vectors and matrices later in the semester.

2.1 Scalars

Scalars are constants. 1, 2, π and e are examples of scalars. Scalars can be added, subtracted, multiplied and (most of the time) divided. Scalars can also be either real or complex. Other operations, such as the trigonometric, logarithmic and exponential functions can also be applied to scalars.

2.2 The Command Window

When you start MATLAB, most of the screen will be taken up by a pane with a double greater-than sign (`>>`). This pane is the *command window* and the `>>` is the *command prompt* or *command line* If you see a flashing cursor after the `>>`, this means that MATLAB is waiting for you to enter a command. The pane on the left side is a listing of MATLAB's current directory. The panes on the right consist of a pane containing a

list of all variables currently defined in the MATLAB workspace and a pane with a history of the commands you have entered.

From the command line, enter:

```
>> x = 6
```

The system will respond with

```
>>
x =
    6
```

If you type:

```
>> x = 6;
```

The system will respond with

```
>>
```

This is the most useful feature of MATLAB. Anytime you place a semicolon (`;`) after a command, the action is carried out, but the result of the computation is not echoed to the display. This is very useful when learning how to operate MATLAB because you can see how a calculation evolves as the series of operations is carried out. Also, it makes debugging your programs easier because you can remove the `;` from commands for which you want the results displayed during computation.

2.3 Important Features of MATLAB and the Command Window

These are some important features of MATLAB and the command window to be aware of:

a) MATLAB is case sensitive. This means that a variable named `B` is not the same as a variable named `b`.

b) MATLAB uses double precision accuracy in its calculations. There are special cases when this can be changed but we will not use them in this course.

c) By default, MATLAB uses a `short` format to display computational results. Even though it shows only the first 5 digits of a result, it is still using double precision behind the scenes. If you want MATLAB to show you all the digits in a result, you can type either `format long` or `format long e` at the command line.

d) You can type `format compact` at the command line to remove extraneous blank lines in the output.

e) You can bring up previously typed commands using the up-arrow key.

2.4 Basic Calculations

All of the usual mathematical operations are available, For example:

```
>> x = 1
x =
   1
>> y = 2
y =
   2
>> x + y
ans =
     3
>> x - y
ans =
     -1
>> x*y
```

```
ans =
     2
>> x/y
ans =
     0.5000
>> y^5
ans =
     32
>> sqrt(y)
ans =
     1.4142
```

The trigonometric functions (sin, cos, *etc.*) as well as their inverses are also available.

The basic mathematical operators are given in Table 1.

+	addition
−	subtraction
*	multiplication
/	division
^	exponentiation
=	assignment

Table 1: Mathematical Operators in MATLAB.

It is important the understand that the assignment operator (=) is not a mathematical equal sign. The purpose of this operator is to *assign* the result of the calculation on the right to the variable on the left. For example, in the statement

```
>> x = 28 * 5.6 / 23.5
x =
    6.6723
```

The value of the expression to the right of the = is computed and the resulting value is stored in the variable x. It is very common in a program to see a statement like

```
>> x = x + 1
```

This statement makes no sense mathematically; rather you should read a statement like this to mean 'the new value of x is the old value of x plus 1.'

2.5 Special Constants

There are some special constants that are pre-defined in MATLAB.

`pi` - this is a double precision approximation of π.

`i,j` - these are the imaginary units, $i, j = \sqrt{-1}$.

`Inf` - infinity (for example, the result of `1/0`).

`NaN` - stands for Not a Number (for example, the result of `0*Inf`)

2.6 Complex Numbers

We will (probably) not use complex numbers in this course much. MATLAB will automatically detect the presence of complex numbers and use complex arithmetic when needed.

```
>> x = -1
x =
   -1
>> sqrt(x)
x =
   0 + 1.0000i
>> y = 2 + 3i
y =
   2.0000 + 3.0000i
>> z = 4 - j
z =
   4.0000 - 1.0000j
>> y*z
ans =
     11.0000 + 10.0000i
```

One benefit of this is that if you are getting complex numbers in your answers and you are not expecting complex numbers, this is a good indication that you have a problem with your calculation.

2.7 Entering Formulas

Most likely, you have all used graphing calculators in the past and are familiar with the notion that mathematical formulas need to be translated into a computer friendly format. The basic rule order of operations for these are

a) Operations in parentheses are performed first.

b) Exponents are computed second.

c) Multiplications and divisions are computed third (from left to right).

d) Additions and subtractions are computed last (from left to right).

When translating a complex formula, it is often more robust to break the calculation down into several steps rather than try to type the entire formula in a single line. For example, suppose

$$z = \frac{1}{1 + \frac{1}{1+\frac{1}{x+y}}}$$

You can compute this in one line using

```
>> z = 1/(1+1/(1+(1/(x+y))))
```

However, transcription errors are less likely to occur if you use

```
>> a = 1/(x+y)
>> b = 1/(1+a)
>> z = 1/(1+b)
```

instead.

3 MATLAB Variables

To this point we have used very generic names for our variables. This is acceptable in some cases, but generally it is a much better idea to choose variable names that are reflective of the quantity the variable represents. For example, if you are computing an acceleration, then variable names like `a`, `acc`, `accel`, *etc.* make more sense than a variable named `orange`.

MATLAB variables must conform to the rules below:

- Variables must start with a letter.
- Variables can contain letters, numbers and the underscore character.
- Variables can be up to 63 characters long, but you shouldn't need anything close to this length in this class.
- Variables should not be the same as an existing MATLAB variable or built-in operation. For example, you can't use `sin` as a variable name. If you want to check if a potential variable name is already being used for something else, you can use the `exist` command

```
>> exist sin
ans =
     5
```

  If the output of this command is anything but 0, you can't use that variable name.
- Variables should not have the same name as an existing script file.

4 Math Functions

This section summarizes the most common math functions used in MATLAB. See `help command` for more information on each command.

- `abs(x)` - Computes the absolute value of x. If x is a complex number, $x = a + bi$, this computes the complex absolute value
$$|x| = \sqrt{a^2 + b^2}.$$
- `sqrt(x)` - Computes the square root of x.
- `exp(x)` - Computes e^x. Note that e is not a built in variable. If you need the value of e, you need to compute `exp(1)`. Also, do not do something like

```
>>  e = 2.718
```

  to define a value for e. This only has 4 digits of accuracy and will adversely affect your calculations.
- `log(x)` - Computes the natural logarithm of x.
- `log10(x)` - Computes the base-10 logarithm of x.
- `sin(x)`, `cos(x)`, `tan(x)` - Compute the sine, cosine and tangent functions of x. The input angle needs to be in radians.
- `asin(x)`, `acos(x)`, `atan(x)` - Compute the inverse sine, inverse cosine and inverse tangent functions of x. The angle is returned in radians. Note that because the `sin`, `cos` and `tan` functions are not one-to-one, the values returned by these inverse functions are range-restricted in the usual mathematical way:
  - `asin(x)` - returns an angle in the range $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$.
  - `acos(x)` - returns an angle in the range $[0, \pi]$.
  - `atan(x)` - returns an angle in the range $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$.
- `atan2(y,x)` - Computes the 4 quadrant inverse tangent of $\frac{y}{x}$. The angle is output in radians and will be in the range $[-\pi, \pi]$.
- `sinh(x)`, `cosh(x)`, `tanh(x)` - Compute the hyperbolic sine, cosine and tangent functions of x. These functions have these names because they obey a set of identities that are very similar to the trigonometric functions. However, the inputs to these functions are *not* angles, so you don't need to convert the inputs to radians.

- `asinh(x)`, `acosh(x)`, `atanh(x)` - Compute the inverse hyperbolic sine, cosine and tangent functions. Because these are not trigonometric functions, the outputs from these functions are not angles.
- `nthroot(x,n)` - compute the nth root of a number. For example, if you need to compute the cube root of x, you can do

```
>> y = nthroot(x,3)
```

Finally, throughout the course, all angles are assumed to be given in radians unless specifically indicated otherwise.