MATLAB Bootcamp

Klavdia Zemlianova and Sonica Saraf

August 30, 2021

Contents

1 What is the purpose of programming languages?

If you do much work on computers, eventually you find that there's some task you'd like to automate. The goal of this course is to help you learn the basics of how to translate algorithmic or mathematical ideas you may have into instructions that your computer can understand and implement.

Programming languages provide the interface that allows you to do so and each language does so in unique ways.

While each language is different, the basic principles of programming translate across language and this is what we want you to understand.

Matlab is used by most neuroscience labs and will be what the assignments in math tools are written for.

2 Data Types

The two most important data types in matlab are numeric and characters/strings

Numeric data types are the numbers: 4, 5, 8.5, etc

A character array is a sequence of characters: 'Hello World'. This treats each character as a unit.

A string array is a container for pieces of text: "Hello World". This treats each string as a unit.

double is the default numeric data type. The function `class()` tells you the data type.

```
>> class(4)

ans =
     'double'

>> class('Hello World')

ans =
     'char'
```

```
>> class("Hello World")

 ans =
     'string'
```

You can convert between data types

```
>> num2str(4)

 ans =
     '4'

>> str2double('4')

 ans =
      4
```

3 Matlab as a Calculator

The simplest thing you can do with MATLAB, is use it as a calculator: + and – are addition, / is division, * is multiplication, ^ is an exponent.

```
>> 3+4

 ans =
     7

>> (3+4)*2

 ans =
     14
```

But be sure to be careful about the order of operations!

```
>> 3+4*2

 ans =
     11
```

You can assign values to variables and operate on variables

```
>> a = 3; b=5; a*b

 ans =
     15
```

```
% Note that above I have used ';' after defining each variable. ...
    This prevents the assignment from being print out to the screen.
```

Lastly, you can clear a variable assignment by using the command `clear`.You can clear all the variable assignments using the command `clear all`. Often, it is a good idea to start with command this so that you know you don't have previous variable assignments that could affect your new code.

MATLAB stores all of the variables you define, so you can use them later. You can use the function `whos` to check the variables you have defined and their type.

```
>> whos
```

You can also operate on strings:

```
>> a="a";b="b";a+b

ans =
    "ab"

>> a="2";b="4";a+b

ans =
    "24"
```

Matlab is a smart calculator, so it will tell you if you try something that doesn't make sense:

```
 a="2";b="4";a-b
Undefined operator '-' for input arguments of type 'string'.

>> a="4"; a*2
Undefined operator '*' for input arguments of type 'string'.
```

4 Vectors and Matrices

You can create a vector using this notation:

```
>> v=[1 2 3] % for a row vector

v =
     1     2     3

>> v=[1;2;3] % for a column vector

```

```
v =
     1
     2
     3
```

You can flip a row into a column vector by using ’

```
>> v = [1 2 3]'

v =
     1
     2
     3
```

Once you have defined vectors, it is easy to operate on them.

```
>> v*v'

ans =

    14
% BUT make sure the dimensions are correct!

>> v*v
Error using  *
Incorrect dimensions for matrix multiplication. Check that the ...
    number of
columns in the first matrix matches the number of rows in the second
matrix. To perform elementwise multiplication, use '.*'.
```

Elementwise operations are performed using the ’.operator’ notation.

```
>> v.*v

ans =

     1     4     9
>> v./v
ans =

     1     1     1
```

To create a vector of values going in even steps from one value to another value, you would use

```
>> b = 1:.5:5
b =
    1.0000    1.5000    2.0000    2.5000    3.0000    3.5000    ...
        4.0000    4.5000    5.0000
```

On the other hand, there is a command to generate linearly spaced vectors: linspace. It is similar to the colon operator (:), but gives direct control over the number of points in between. For example,

```
>> y=linspace(1,10,5)
 y =
     1.0000    3.2500    5.5000    7.7500   10.0000
```

Similarly, you can create matrices by using the ';' Just make sure the dimensions line up! (you need to have the same number of elements on each side of the ';').

```

>> m = [1 2; 3 4]

 m =
       1     2
       3     4
```

One of the strengths of MATLAB is that it makes matrix operations very easy!

```
>> m*m

 ans =
      7    10
     15    22

 % Just as in the case of vectors, elementwise operations are ...
      done using '.operator' where operator is one of ...
      \mcode{+,-,*,?,^}.
```

You can also use ' to get the transpose of a matrix

```
>> (m*m)'

 ans =
      7    15
     10    22
```

You can also make vectors and matrices using stings:

```
>> ["apple"; "banana"]

 ans =
   2 1  string array
     "apple"
     "banana"

>> ['apple'; 'banana'] % This will FAIL! WHY?
```

Lastly, sometimes it is useful to know the length of an array or the shape of a matrix, ie how many cols and rows it has. For an array, you can get the number of elements using `length()` and for a matrix, you can check for the shape using the `size()` function as below:

```
>> d = [1 2; 3 4; 5 6]; [rows,cols] = size(d)
rows =
     3

cols =
     2
```

Here are some useful functions for auto-generating elementary matrices

$eye(m,n)$	Returns an m-by-n matrix with 1 on the main diagonal
$eye(n)$	Returns an n-by-n square identity matrix
$zeros(m,n)$	Returns an m-by-n matrix of zeros
$ones(m,n)$	Returns an m-by-n matrix of ones
$diag(A)$	Extracts the diagonal of matrix A
$rand(m,n)$	Returns an m-by-n matrix of random numbers

EXERCISE: Use `linspace` to make a 15 element vector taking values from 0 to 10. Then, use the function `reshape` to convert this array into a 3x5 matrix. Lastly, multiply by a vector so the output gives an array of the row sums.

5 Logical Comparisons

```
>> 3 == 3

ans =

  logical

   1

>> 3 == 2

ans =

  logical

   0
```

You can do comparisons with variables

```
>> a=2; b=6; a==b
```

```
2
3  ans =
4
5    logical
6
7     0
```

We can also check if they are NOT equal by using ' =”

```
1   >> a=2; b=6; a≠b
2
3  ans =
4
5    logical
6
7     1
```

You can also compare with strings

```
1  >> "ab"=="ab"
2
3  ans =
4
5    logical
6
7     1
```

String comparison is case sensitive!

```
1  >>"ab"=="AB"
2
3  ans =
4
5    logical
6
7     0
8
9  % Remember that variables and strings are not the same!
10
11 >> abc ==  "abc"
12 Undefined function or variable 'abc'.
```

EXERCISE: Check how many times 2 appears anywhere in the matrix `[1 ... 0 2; 3 4 1; 2 2 5]`. HINT: the function `sum()` might be useful.

6 Basics of Conditionals

We just saw comparisons- comparisons return True or False and this can be used to tell the computer what to do. For example if 'str1' == 'str2': print("Yes they are equal!"). Let's try this...

```
>>
if "abc" == "abc"
disp('They are equal!')
end
They are equal!

% but what if they are NOT equal?
>>
if "Abc" == "abc"
disp('They are equal!')
end
```

We could compare using the NOT operator which is $\sim =$

We can also do AND and OR comparisons. The symbol for AND is `&` and the symbol for OR is |.

```
>> n = randi(5,1); % returns a random integer from 1 to 5
if n<3 | n>3 % OR comparison
   disp("n is not equal to 3!");
end
```

What if I wanted multiple conditions that were more complicated? You could use if, elseif, and else statements. The general format for these is given below:

```
if (logical expression)
    matlab command
elseif (other logical expression)
    another matlab command
else
    a matlab command
end
```

EXERCISE: Write a statement that prints "yes" if 75642 is divisible by 6 or 7 and "no" if it isn't.

7 Indexing

We saw earlier that we can define vectors and matrices:

```
>> a=2:2:10

a =
     2     4     6     8    10

>> m = [1 2; 3 4]

m =
     1     2
     3     4
```

How do I grab an element? In MATLAB indexing is done through circle brackets (). We will need to specify an integer that tells the position (or index) of the element we would like to access. MATLAB indexes starting at 1. (unlike python, which indexes at 1!)

```
>> v(3)

ans =
     3

>> v(end) % Grabs the last element

ans =
     3
>> v(end-1) % Grabs second to last element

ans =
     2

>> x=1:1:10; x(5:end) % Can access multiple elements at once

ans =
     5     6     7     8     9    10

>> m(2,1) % Similar notation for accessing entries of a matrix.

ans =
     3
```

You can also use the : to get many elements at once, such as an entire row or column.

```
>> m(:,1) % gives the first column

ans =
     1
     3

>> m(1,:) % gives the first row

ans =
     1     2
```

8 Basic Loops

One of the most useful constructions that you will learn will be a `For` loop. This will allow you to iterate through an array and perform some operation on each element. Here is a simple example:

```
for element=v
    disp(element)
end  % note the presence of 'end' to indicate the termination of ...
    what falls within the loop
```

Another type of loop is the `While` loop which will allow you to do something *while* a statement is true. Here is a simple example:

```
i=0;  % this line initializes i
while i<3
    i = i + 1;
end
disp(i) % this line prints out i after the end of the while loop
```

A word of caution on while loops, make sure there is a termination point. Otherwise, you could get stuck in the loop!

The use of `break` will allow you to exit a for or while loop entirely. Here is an example of the above while loop with a break statement:

```
i=0;
while i<5
    i = i+1;
    if i == 3
        break % allows us to exit the while loop early (when i ...
            is 3 rather than when i is 5)
    end
end
disp(i)
```

Similarly, the use of `continue` will allow you to skip ahead to the next iteration of the for or while loop.

EXERCISE: Write code for iterating through every element of an NxM matrix.

EXERCISE: First, define an array that takes integer values from 1 to 15 and then write a loop that prints out every 3rd element of this array.

9 Vectorization

MATLAB is written to deal with matrices and vectors. Because of that, it is much more efficient to solve problems solely by doing math on your vectors or portions of your vectors, rather than by ever needing to loop through particular elements in your vector and check their values. Say you wanted to evaluate a function at a set of x values, then one way you might think to do it the following:

```
x = [some vector of N elements]
for n = 1:N
    result(n) = sin(x(n));
end
```

But this could take a long time. Instead, you can avoid using loops and just perform the function on the entire vector!

```
x=0:.01:2*3.14;
y = sin(x);
plot(x,y)
```

In general, vectorizing is much faster, is easier to easier to read and produces fewer lines of code, and it more closely assembles mathematical notation.

Some useful functions when it comes to operating on arrays:

```
>> x = [4, 5, 7,1, 11, 102, 23];
>> min(x)
ans =
     1

>> [val, loc] = max(x)
val =
   102

loc =
     6

>> sum(x)
ans =
   153
>> cumsum(x)
ans =
     4     9    16    17    28   130   153

>> sort(x)
ans =
     1     4     5     7    11    23   102
>> mean(x)
ans =
   21.8571

```

```
>> unique(["a","b","a"])

ans =
  1 2  string array
    "a"    "b"

>> find([1,2,3]>2) % find(X) returns nonzero entries of the ...
    array X

ans =
     3
```

10 Functions

All matlab functions and scripts are plain text files that contain matlab commands. Matlab will treat any file that ends in .m as either a function or a script. It can find .m files you've written that are in your /matlab directory, in the directory you have cd'd into from the matlab prompt, or in a directory you've started matlab with.

A script is just a list of commands to be run in some order. Placing these commands in a file that ends in .m allows you to "run" the script by typing its name at the command line. You type the name of the script without the .m at the end.

A function is capable of taking particular variables (called arguments) and doing something specific to "return" some particular type of result.

A function needs to start with the line `function return_values = functionName(arguments)` so that matlab will recognize it as a function. Each function needs to have its own file, and the file has to have the same name as the function.

Let's create our first simple function!

```
% this is stored in a file called FtoC.m
function C_temp = FtoC(F_temp)
    C_temp =  (F_temp-32)*(5/9);
end

% now in the command, I can just type the function name and ...
    argument to execute!
>> FtoC(212)
ans =
   100
```

You can also make a function with multiple inputs and outputs:

```
function [m,s] = stat(x)
    n = length(x);
```

```
    m = sum(x)/n; % computes mean
    s = sqrt(sum((x-m).^2/n)); % computes std. dev.
end
```

Now we are going to define a function that does not take any input and has no output, but still prints content.

```
function unsolicited_comment()
    x = rand; % selects a value from the uniform random ...
        distribution on [0,1]
    above_half = x > .5;
    if above_half
        disp('you are great')
    else
        disp('try again')
    end
end
```

Remember, that the filename for this function has to be `unsolicited_comment.m`.

We can now call this function by entering the name `unsolicited_comment()` into the command window. Or, we can call it from a for loop:

```
for i=1:10
    unsolicited_comment()
end
```

EXERCISE: Write a function called dice that takes as input, n, the number of dice and outputs n dice rolls. A useful function will be `randi()`. Then use the function `any()`to check if a '3' was rolled and output the following text: "You Won!". Otherwise, output the text: "Better luck next time". You can type `help functionName` into the command window to get info about the inputs/outputs of a function.

11 Breakpoints and debugging

debugging exercise

12 Structures

From our experience, experimental data is usually loaded into MATLAB as a "struct" which holds multiple fields and data within them. Importantly, structs can data of different types – unlike matrices and vectors.

Given a structure, say `stimulus`, with fields `contrast, orientation, ... spatial_frequency, and temporal_frequency`, we can access the each field's data – e.g. `stimulus.contrast`.

```
s = struct;
s.contrast = 0.75;
s.orientation = linspace(0, 360, 16);
s.spatial_frequency = 1;
s.temporal_frequency = 6;
```

13 Multiple Condition Indexing

How to index arrays or matrices based on multiple conditions being satisfied – use .mat file.

14 Plotting

MATLAB makes plotting very easy. Here are the basic commands to make your very own simple plot:

```
x=0:.01:2*3.14;
y=sin(x);
z=cos(x);

figure;
plot(x,y,'b-o'); % plots a blue curve using 'o' as markers
xlabel('Input')
ylabel('Output')
xticks([0 3.14 6.28])
title('Plotting is so easy!' )
```

What if you now want to add another function to this plot? Use the following commands to add a curve to the existing figure. This could, for example, be useful if you want to add a curve to a plot during a loop!

```
hold on; % this line keeps the previous curve from being erased
plot(x,z,'r') % plots a new curve in red
```

Another common/useful plot is the histogram:

```
>> n=normrnd(0,1,1, 100); % generates 100 random values from a ...
      Gaussian distribution with mean 0 and std. dev.of 1.
>> hist(n) % plots histogram
```

Lastly, you can clear a figure using the `clf` command.

15 Saving and Loading Data

You can save variables to a file using the `save()` function:

```
>>save('NueralData','SpikeTimes','Voltages')
% The above line saves the variables SpikeTimes and Voltages to ...
    a file called NeuralData.mat
```

You can load data from a `.mat` file using the `load` function.

```
>> load('NeuralData') % This will load all the variables from ...
    the file NeuralData.mat.

%Alternatively, you can load a single variable using the ...
    following command:
>> load('NeuralData','Voltages')
```

16 Advanced

A cell array is a datatype that will be useful in the future. It is similar to a struct because it can hold many datatypes. The indexing operation is different, however, and similar to that of vectors because it is organized by cell number, e.g.: `c{2}` will give you the data in the 2nd cell of `c`.

Here are some useful cell array functions:

```
cellfun
num2cell
mat2cell
```

Here are some useful set functions for operations on arrays:

```
intersect
ismember
setdiff
union
unique
```