# Introduction to MATLAB

Hello everyone! Welcome to this tutorial. MATLAB is an interactive software focused on numeric calculus. The objective of this tutorial is to provide you the basic skills necessary for proficiently using this software and its main features.

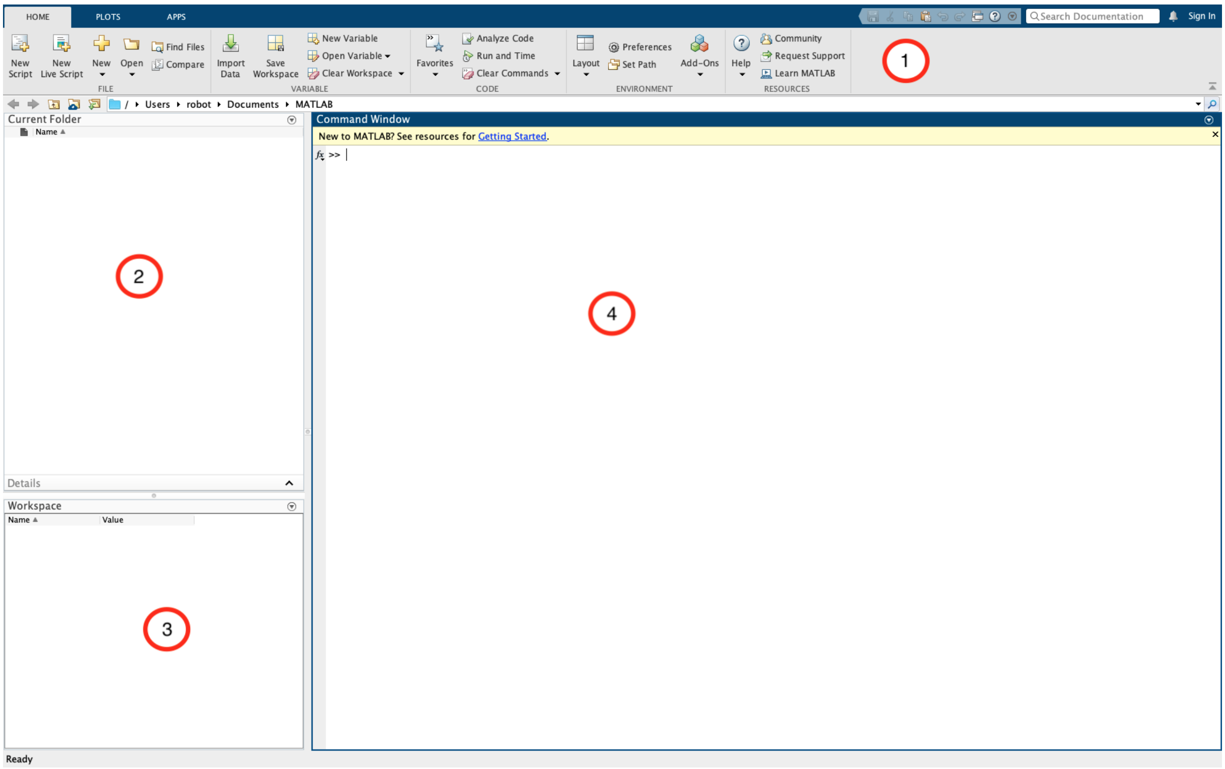
### Installation

Luckily, Instituto Superior Técnico provides a detailed tutorial on how to install it. You can find it here.

## Index

# Basics

## Default Layout



New to MATLAB? See resources for Getting Started.

```
>>
```

So this is the screen you arrive at when you first launch MATLAB. It may seem like a lot of stuff at first, but as you can see, some regions are labeled:

1. The top bar shows some basic functions like opening a file or creating a new one. You can also access settings and other features.
2. This panel indicates the directory we're in. This indicates that the current folder is part of the current scope, so any function or file inside the current directory is accessible by our code.
3. The `Workspace` shows all the variables currently in memory and a detailed view of these can be accessed by double-clicking them.
4. The `Command Window` , as the name says, is a window where you write commands. You can write any command and press `Enter` to immediately execute it.

## Command Window

Let's experiment! Write the following:

```
1+1
```

And press `Enter`. You'll see:

```
>> 1+1

ans =

     2
```

You can do anything here! **Tip:** Pressing the `Up` key you can iterate through the previous commands. For example, let's assign a variable:

```
a = 2
```

And press `Enter`:

```
>> a=2

a =

     2
```

Now you can use this variable anywhere. Let's use the variable `a` in a simple calculus:

```
10/a
```

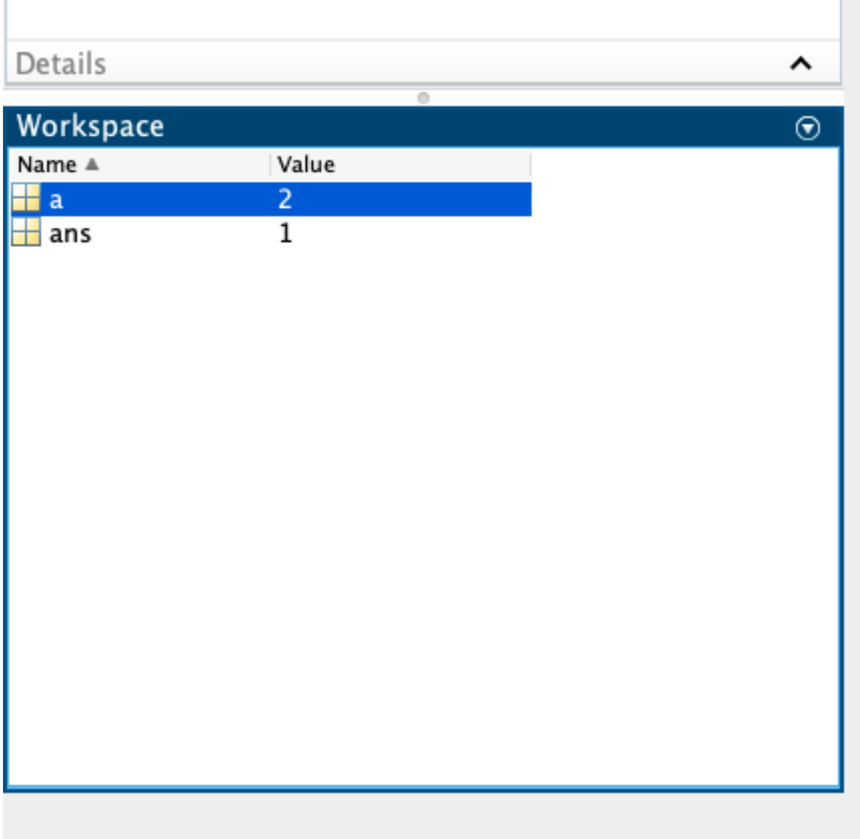
The result is:

```
>> 10/a

ans =

     5
```

**Tip:** If you press the `Tab` key while writing on the `Command Window`, MATLAB will prompt you with auto-completion.

## Workspace

If you've been paying attention, you may have noticed that the panel `Workspace` has some new information:

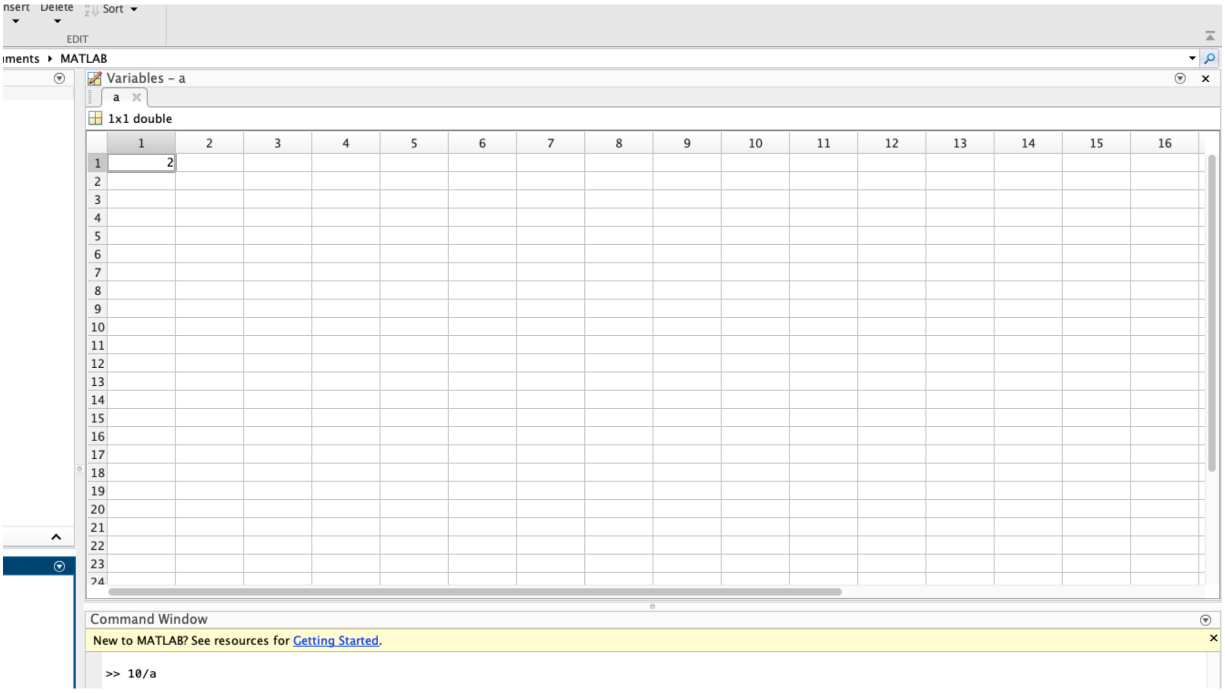


| Name ▲ | Value |
|---|---|
| a | 2 |
| ans | 1 |

Our variable `a` is listed here as well as an `ans` variable. The latter contains the output of the last command ran and can also be used:

```
>> ans-4

ans =

     1
```

If you double click on a variable on the `Workspace` panel, you can see a detailed view of it:



1x1 double

|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

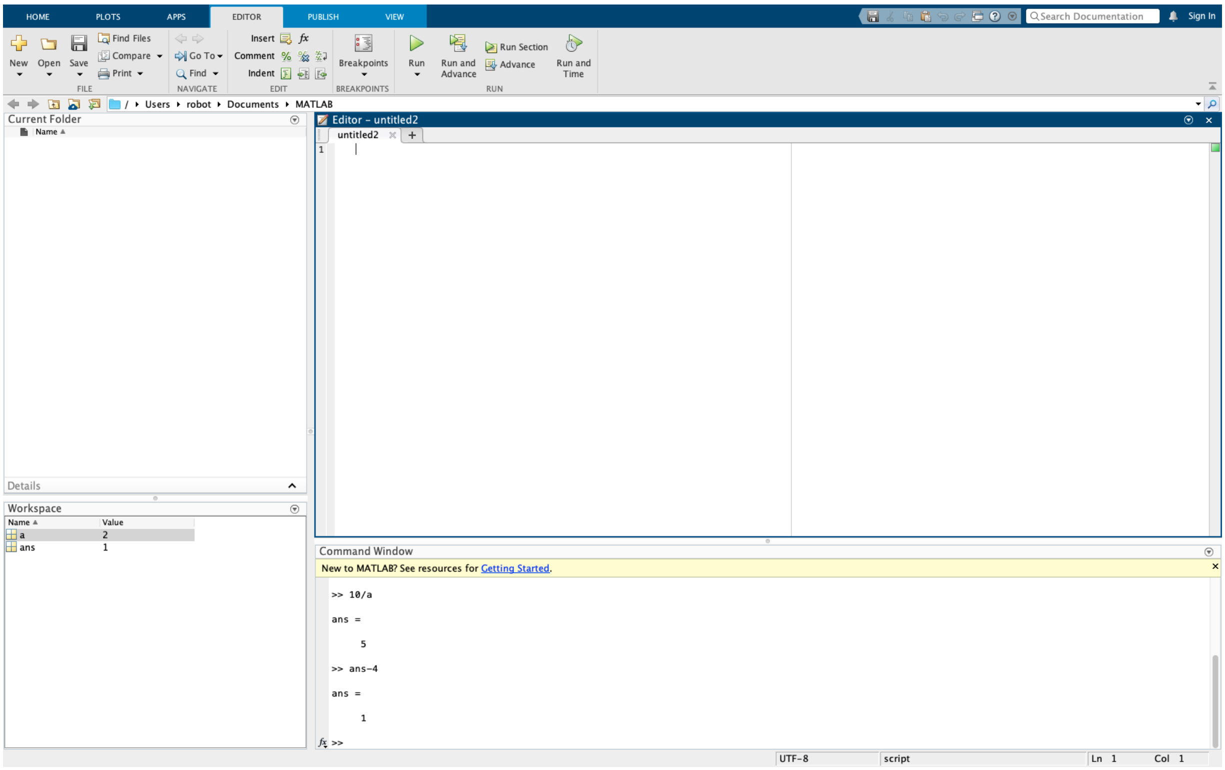
New to MATLAB? See resources for Getting Started.

```
>> 10/a
```

Here you can easily edit it.

## Script Editor

A script is a text file containing code that our program can read and execute. This very helpful if you want to write code and repeat it, To create one, make sure you're one the `HOME` tab and press the button indicating `New Script`. This will open a new window:



| Name | Value |
|---|---|
| a | 2 |
| ans | 1 |

New to MATLAB? See resources for Getting Started.

```
>> 10/a

ans =

     5

>> ans-4

ans =

     1

fx >>
```

In this new window, you can edit the newly created script and later execute it. For example, let's put everything we did on the `Command Window` on our new script:

```
1+1
a=2
10/a
ans-4
```

You should then save the script by pressing the `Save` button on the top bar, save it as `example.m`, and then press `Run` to run the script and execute all the written commands. You should see the output of the script on the `Command Window`:

```
>> example

ans =

     2


a =
     2


ans =

     5


ans =

     1
```

But sometimes you don't want the `Command Window` to display every output of the commands of your script. In MATLAB you can suppress the output of a command by appending `;` to the end of the command. For example, replace your script with the following:

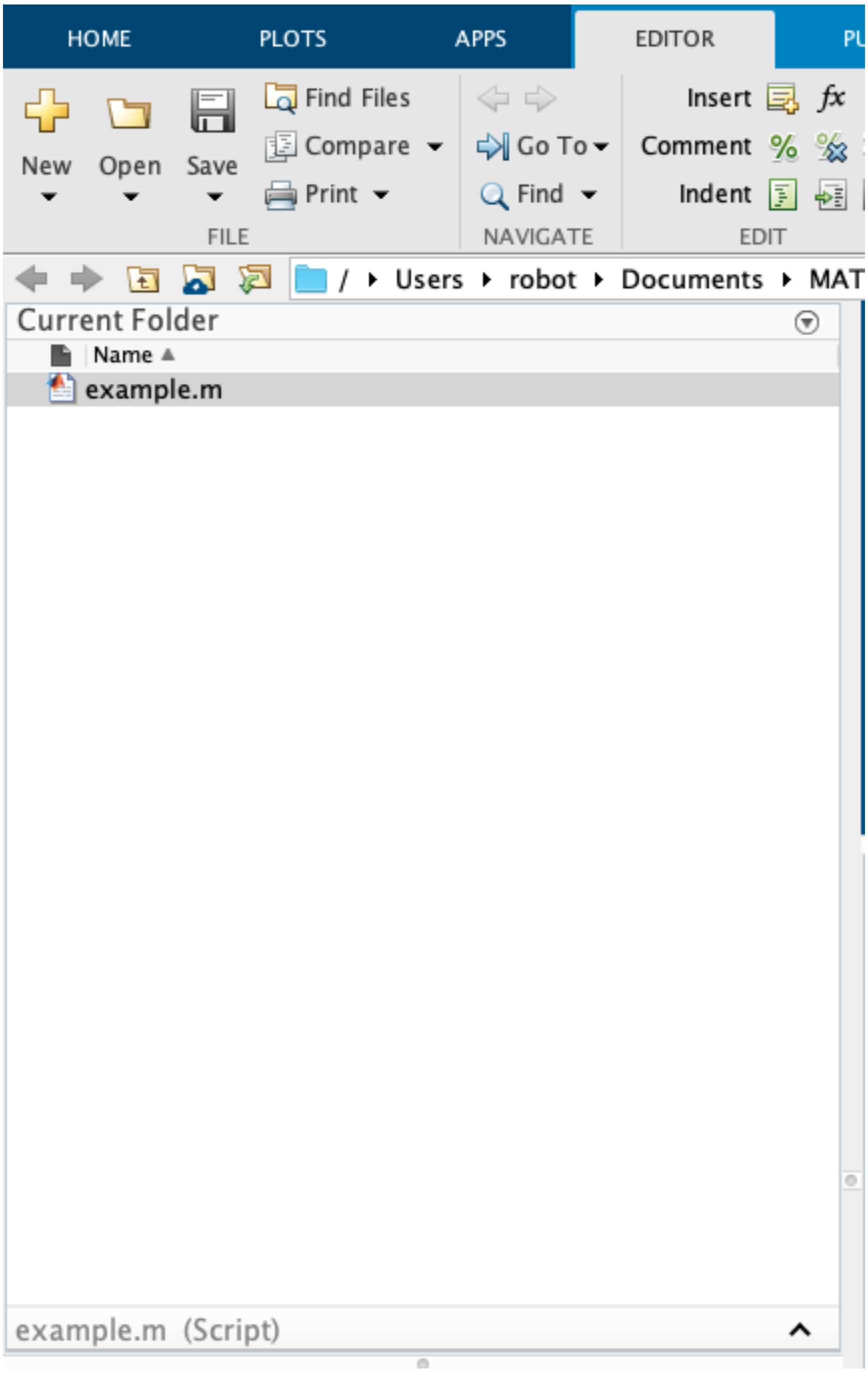
```
a=1+1;
b=a+3;
b*2
```

As you can expect, `a=2` and `b=5` , but since we suppressed their output, the only output of our script when ran is:

```
>> example

ans =

    10
```

## Current Folder

If you watch the left panel, you can see that you have a file called `example.m` which is the created script. Using this panel, you can manage the files on your current directory.



For example, try duplicating the `example.m` file by right-clicking it, selecting copy, and then pasting it. You'll create a new file called `Copy_of_example.m` . If you double click it, you'll open it on your Script Editor. You can write anything you want on this new script and you won't lose the contents of our first script. If you write the name of a script on the `Command Window` and press `Enter` you'll ask MATLAB to run that script:

```
>> Copy_of_example

ans =
     10
```

## Matrices and Arrays

MATLAB makes it easy to work with matrices. To create a matrix, you can simply write the following:

```
>> a=[1,2,3;4,5,6;7,8,9]

a =

     1     2     3
     4     5     6
     7     8     9
```

And as you can see a `3x3` matrix is created. But imagine you want to create a matrix with `100x200` elements all equal to `0`. Writing one-by-one is not efficient, luckily, MATLAB allows you to create an empty matrix by:

```
zeros(100,200);
```

If you want to create a matrix with the same number of elements, but all equal to `3`, you can do:

```
ones(100,200)*3;
```

There's also a function to generate the identity matrix:

```
>> eye(3,3)

ans =

     1     0     0
     0     1     0
     0     0     1
```

You can alter specific elements of the matrix:

```
>> a=zeros(3,3);
>> a(2,1)=3

a =

     0     0     0
     3     0     0
     0     0     0

>> a(:,2)=4

a =

     0     4     0
     3     4     0
     0     4     0
```

Note that when you use `:` you select all the elements of the specified row or column. You can check the size of the matrix using the `size` function:

```
>> [M,N]=size(a)

M =

     3


N =

     3
```

You can ommit the output of a function using a `~`:

```
>> [M,~]=size(a)

M =

     3
```

Now, if you want to generate an array with numbers from `0` to `10`, you can do it by writing:

```
>> 1:10
```

```
ans =

     1     2     3     4     5     6     7     8     9    10
```

You can also specify the step taken. If you want a step equal to `3`, you can do so:

```
>> 1:3:10

ans =

     1     4     7    10
```

Now let's start doing some simple operations using matrices and arrays. You can sum:

```
[1,2,3;4,5,6;7,8,9]+ones(3,3)

ans =

     2     3     4
     5     6     7
     8     9    10
```

And subtract, of course. A multiplication (dot product) is done by:

```
>> [1,2,3;4,5,6;7,8,9]*ones(3,3)

ans =

     6     6     6
    15    15    15
    24    24    24
```

If you want to multiply each element by the element in the same position on the other matrix, you can do so:

```
>> [1,2,3;4,5,6;7,8,9].*ones(3,3)

ans =

     1     2     3
```

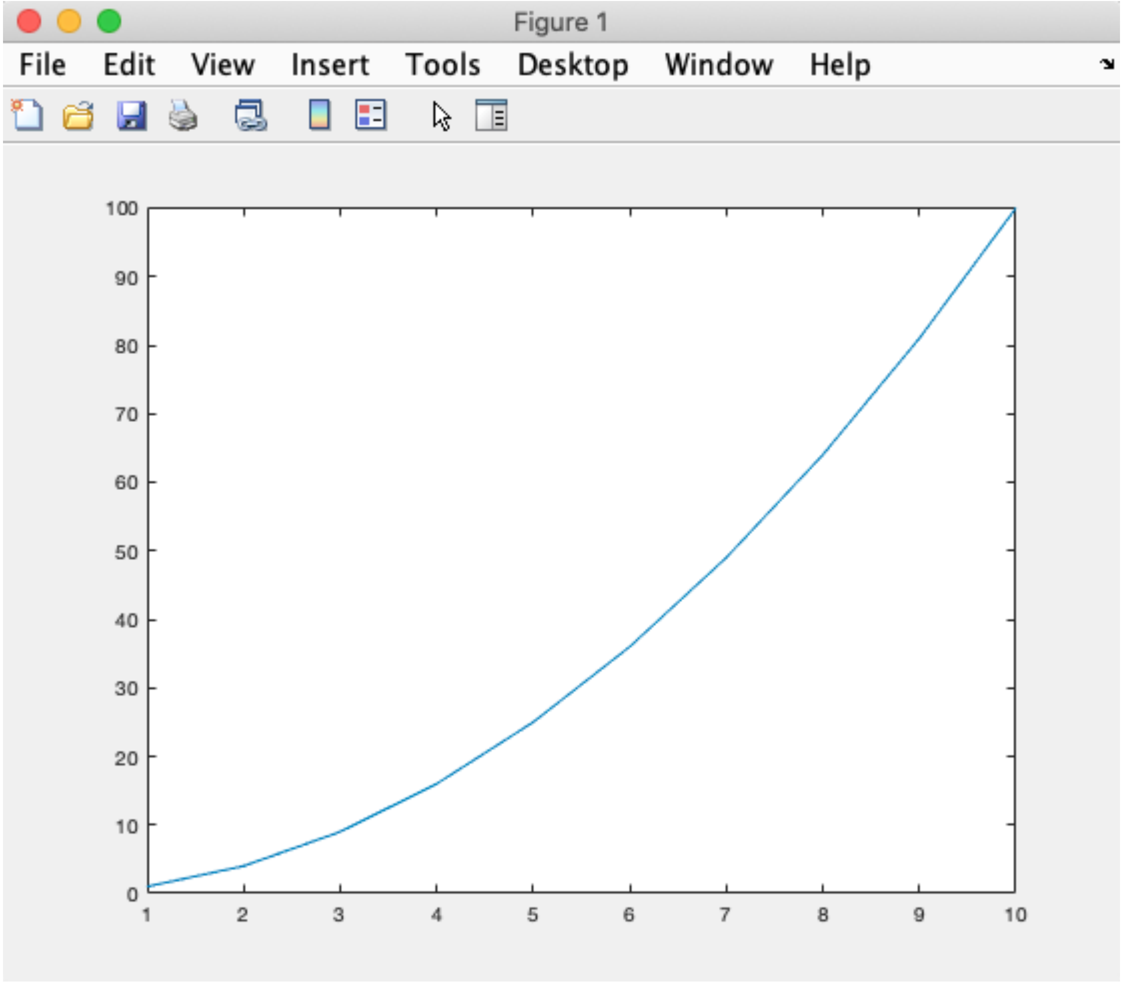
```
    4     5     6
    7     8     9
```

The same applies to division.

## Plots

MATLAB supports a wide range of graphs. You can create a simple line plot:

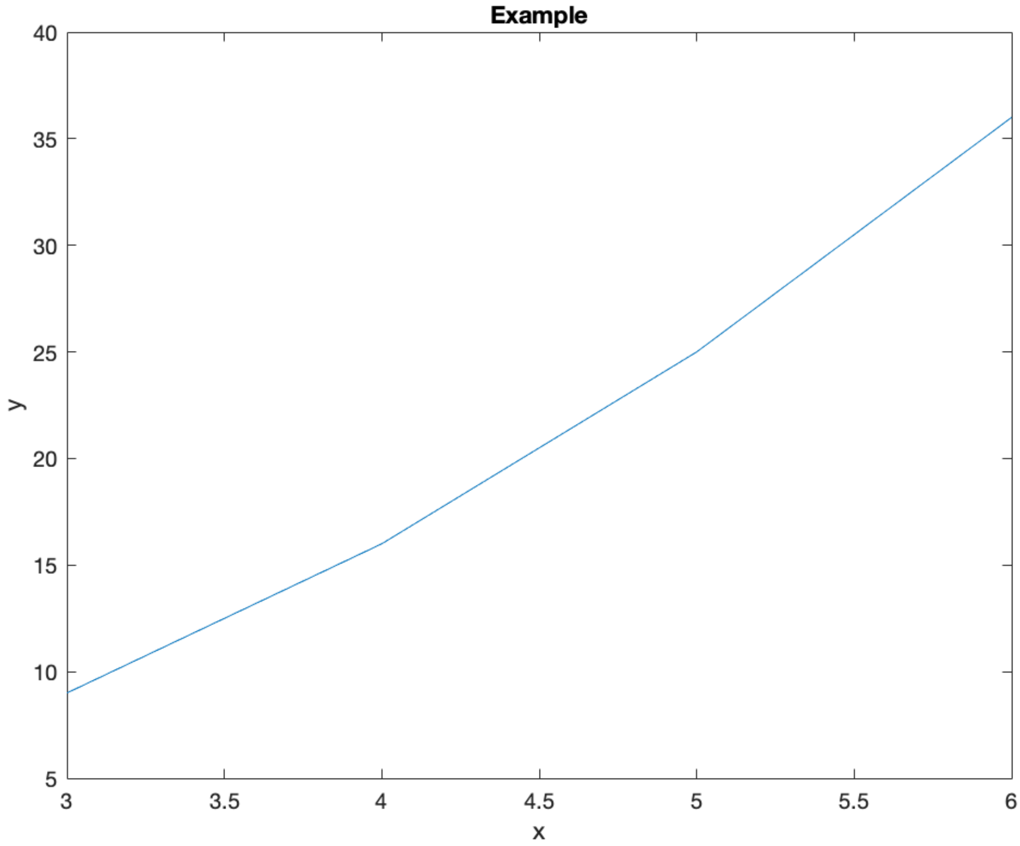
```
x=1:10;
y=x.^2;
plot(x,y)
```



It's possible to specify a title, a label for each axis and also the domain of the plot:

```
title("Example")
xlabel("x")
ylabel("y")
```

```
xlim([3,6])
```

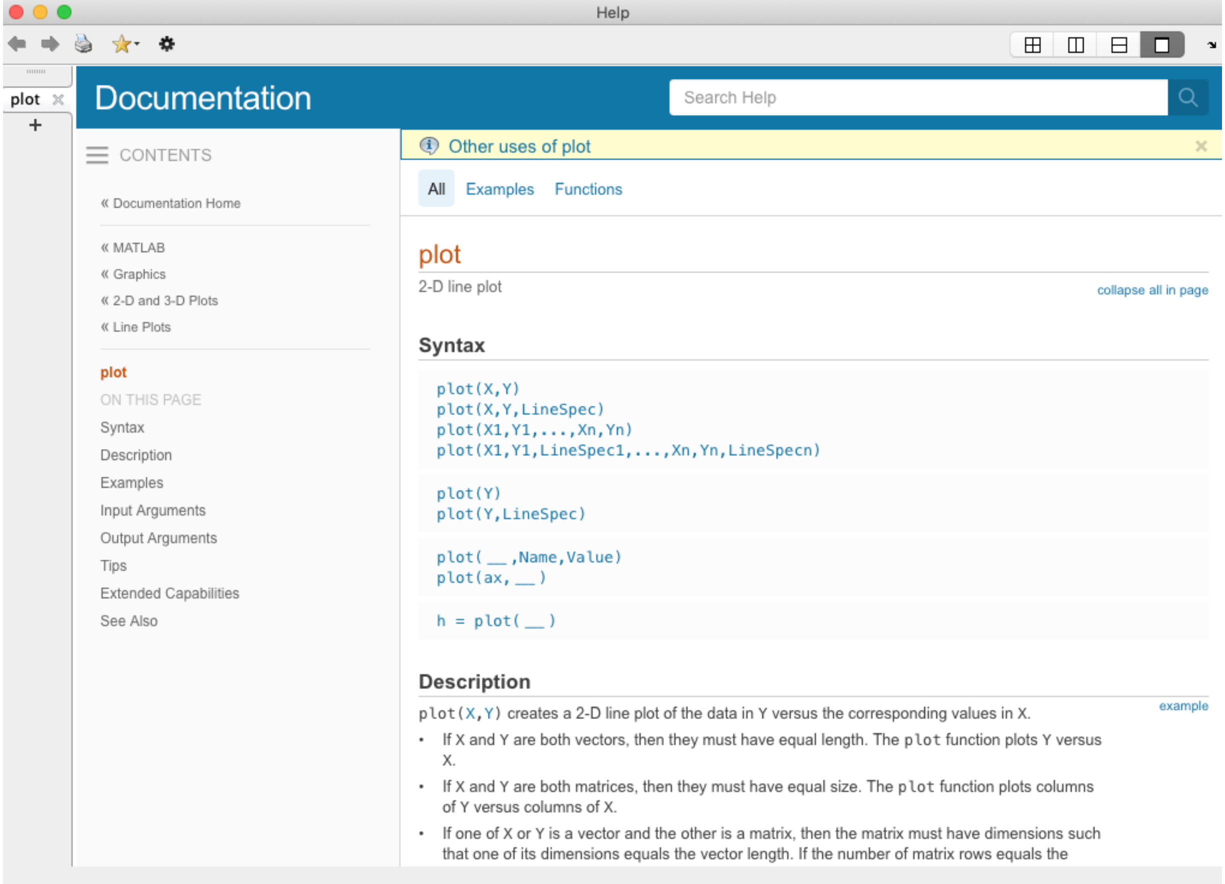


**Tip:** If you wish to further customize your plots, you can read the documentation by writing the following on the `Command Window`:

```
>> help plot
 plot   Linear plot.
    plot(X,Y) plots vector Y versus vector X. If X or Y is a matrix,
    then the vector is plotted versus the rows or columns of the matrix,
    whichever line up.  If X is a scalar and Y is a vector, disconnected
    line objects are created and plotted as discrete points vertically at
    X.

    plot(Y) plots the columns of Y versus their index.
    If Y is complex, plot(Y) is equivalent to plot(real(Y),imag(Y)).
    In all other uses of plot, the imaginary part is ignored.
 (...)
```

You can also do:

```
doc plot
```



plot

2-D line plot

Syntax

```
plot(X,Y)
plot(X,Y,LineSpec)
plot(X1,Y1,...,Xn,Yn)
plot(X1,Y1,LineSpec1,...,Xn,Yn,LineSpecn)

plot(Y)
plot(Y,LineSpec)

plot(__,Name,Value)
plot(ax,__)

h = plot(__)
```

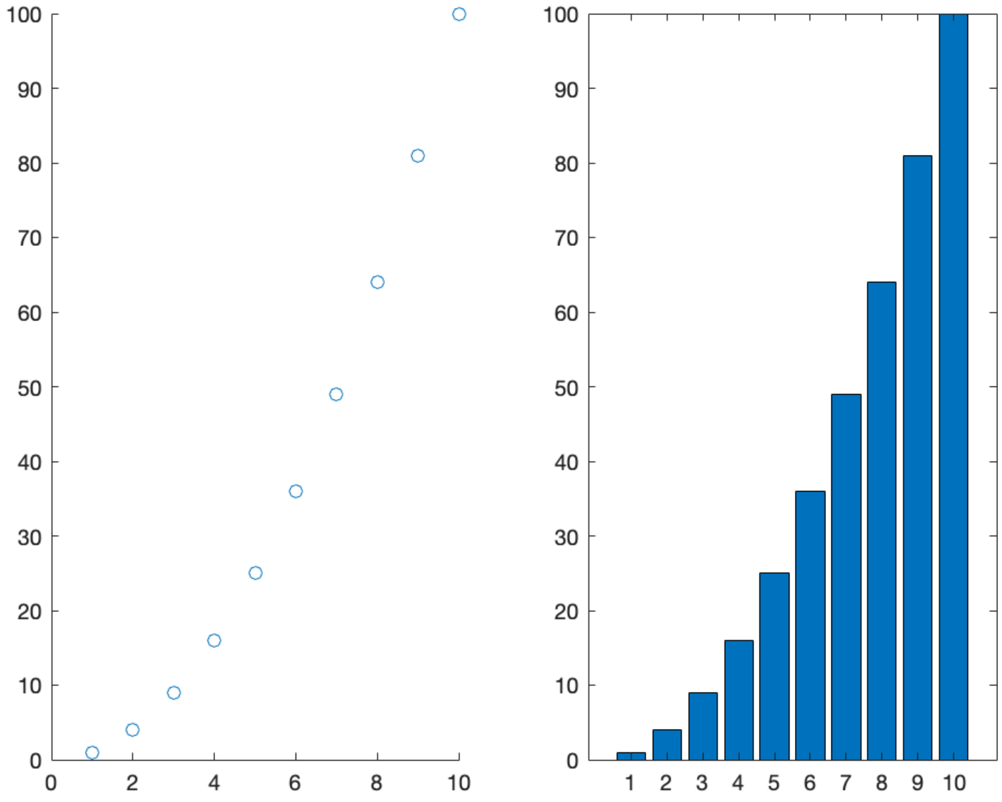
Description

`plot(X,Y)` creates a 2-D line plot of the data in Y versus the corresponding values in X.

- If X and Y are both vectors, then they must have equal length. The `plot` function plots Y versus X.
- If X and Y are both matrices, then they must have equal size. The `plot` function plots columns of Y versus columns of X.
- If one of X or Y is a vector and the other is a matrix, then the matrix must have dimensions such that one of its dimensions equals the vector length. If the number of matrix rows equals the

Besides the `plot` command, you can also use other commands that will display your data in different ways, and even display different plots in the same Figure:

```
subplot(1,2,1)
scatter(x,y)
subplot(1,2,2)
bar(x,y)
```



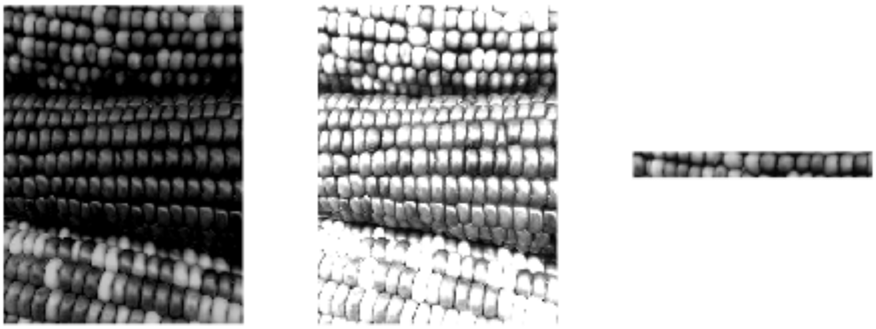
## Images

You can also visualize images and process them. An example image can be load and visualized with the following commands:

```
corn_gray = imread('corn.tif',3);
imshow(corn_gray)
```

The image is represented as a `MxN` matrix and, therefore, you can do all sorts of processing on that image, for example:

```
subplot(1,3,1)
imshow(corn_gray-50)
subplot(1,3,2)
imshow(corn_gray*2)
subplot(1,3,3)
imshow(corn_gray(32:64,:))
```



## Functions

You should organize your code in functions, which make your life a lot easier! To create a function, you have to define a name for it, the output it takes and the output it returns. Create a new script, paste the following and save it as `example_function.m` .

```
function [output1, output2] = example_function(input1, input2)
    a = input1 + 2;
    output1 = a^2;
    output2 = input2 * 3;
end
```

You can then run the script by simply typing on the `Command Window` :

```
>> example_function(32,17)

ans =

        1156
```

## Conditional Statements

```
a=10;
if a>20
    disp(1);
elseif a<10
    disp(2);
else
    disp(3);
end
```

If you run the above code, the number `3` is printed. You can also use more complex conditions:

```
a=10;
if a>20 || a<10 % or
    disp(1);
elseif a<20 && a>15 % and
    disp(2);
```

```
else
    disp(3);
end
```

## Loops

If you want to repeat some action many times, you should probably use a loop. You can use a `while` loop:

```
i=1;
while i<=10
   disp(i);
   i=i+1;
end
```

Which will print every number from 1 to 10. You can also do this with a `for` loop:

```
for i=1:10
   disp(i);
end
```

You can, of course, specify the step of the `for` loop as specified in the previous section. It's also possible to iterate over an array by doing:

```
a=[2,4,1,7];
for i=a
   disp(i);
end
```

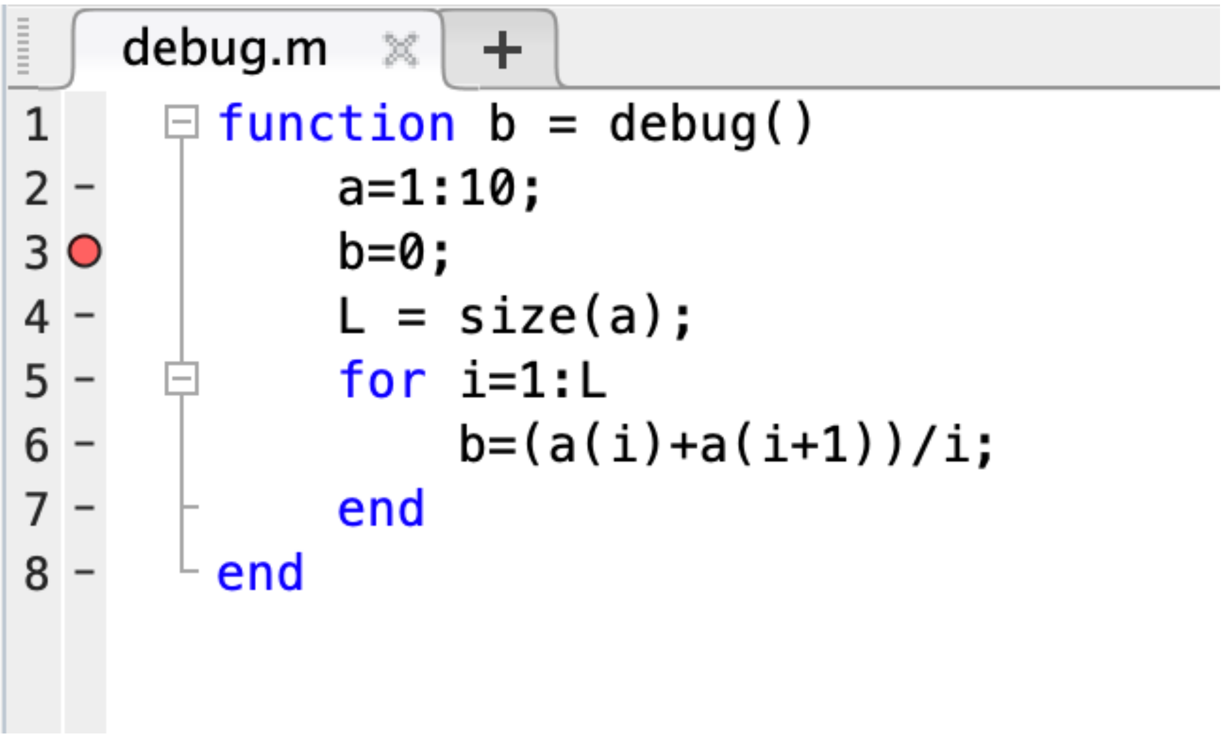
Which prints every member of the `a` array.

## Debugging

MATLAB has a life-saving feature, you can debug your code and verify the program state in each step. The debugging mode can only be used when running a script. Let's debug the following script:

```
function b = debug()
    a=1:10;
    b=0;
    L = size(a);
    for i=1:L
        b=(a(i)+a(i+1))/i;
    end
end
```

To start debugging, click on the – next to the line number of the line you which to debug (for example):



debug.m

```
function b = debug()
    a=1:10;
    b=0;
    L = size(a);
    for i=1:L
        b=(a(i)+a(i+1))/i;
    end
end
```

And then press the `Run` button. You'll notice that the program will pause its execution when it arrives at that breakpoint. You can now check each declared variable value on the `Workspace` panel and can also use the `Command Window` to execute commands over the program variables during execution:

```
K>> a*2

ans =

     2     4     6     8    10    12    14    16    18    20
```

You then can go through the code step-by-step by clicking on the `Step` button on the `Top Bar`. The `Workspace` panel will update its variables during each step. You can further explore this feature and you'll see how helpful it can be!

## First Project - PPG from Smartphone Camera

You're now ready for a simple project! Imagine you want to extract the PPG from a video. You can record a video of your finger covering your phone back camera and flash (remain very still) or you can use the example video.

## Loading the Video

The first step is loading a video.

```
video = VideoReader('ppg.mp4');
```

After loading a video you'll have a video object. The video object has many properties. We can check, for example, the number of frames:

```
>> video.NumFrames

ans =

   217
```

## Extracting Features

The next step is extracting the PPG signal from our video. We then need to iterate over each frame and do some processing on it:

```
signal = zeros(1,video.NumFrames);
i=1;
while hasFrame(video)
    frame = readFrame(video);
    % do something
    i=i+1;
end
```

To extract the PPG signal, we can take one of the color channels, for example, the red one (1st color channel - RGB), and average the pixels intensity value. This average value will vary according to blood flow.

```
while hasFrame(video)
    frame = readFrame(video);
    signal(i)=mean(frame(:,:,1),'all');
    i=i+1;
```

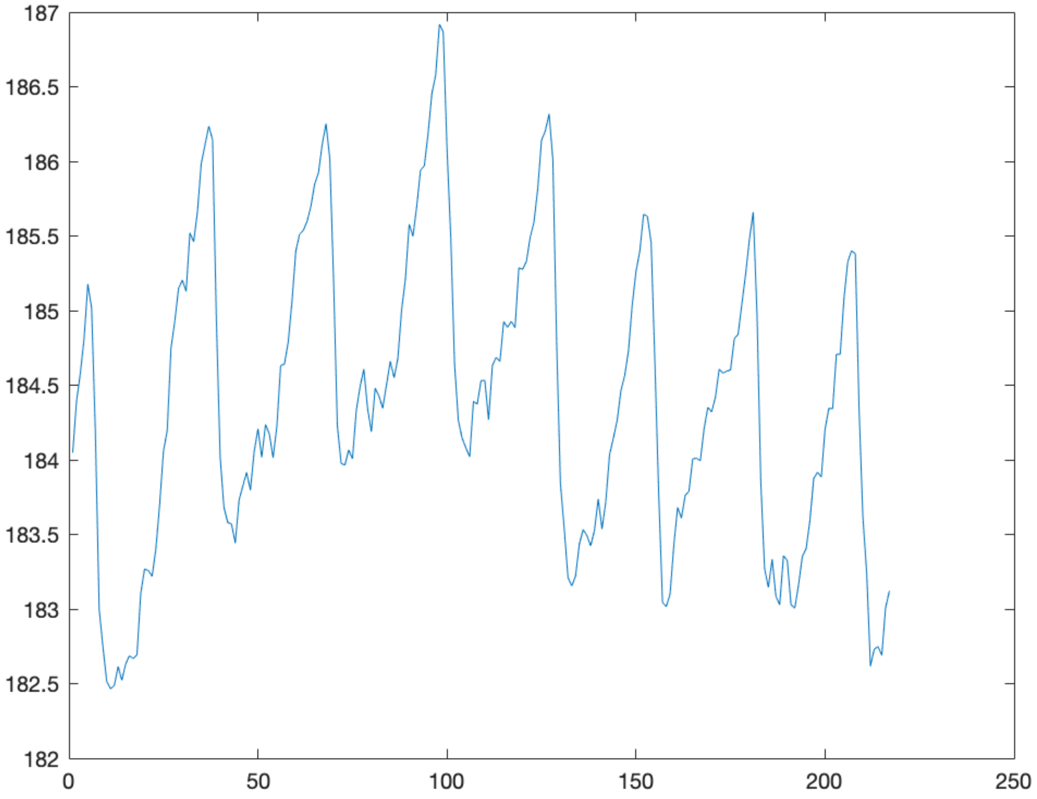
```
end
```

We now end up with a one-dimensional array that corresponds to our PPG.

## Plotting the Signal

Now the PPG signal can be simply plotted by doing:

```
plot(ppg)
```



## Final

The complete version of this code is:

```
video = VideoReader('ppg.mp4');
signal = zeros(1,video.NumFrames);
i=1;
while hasFrame(video)
    frame = readFrame(video);
    signal(i)=mean(frame(:,:,1),'all');
    i=i+1;
end
plot(signal)
```

This simple code is capable of extracting the PPG signal from a simple video.

### Challenge

Try to calculate the heart rate from the same video.

## Conclusion

Now you're capable of independently exploring MATLAB and all of its features. If you have any doubt, you can open an issue in this repository, or directly contact me at afonsocraposo@gmail.com