

# APPENDIX A: MATLAB Introduction

---

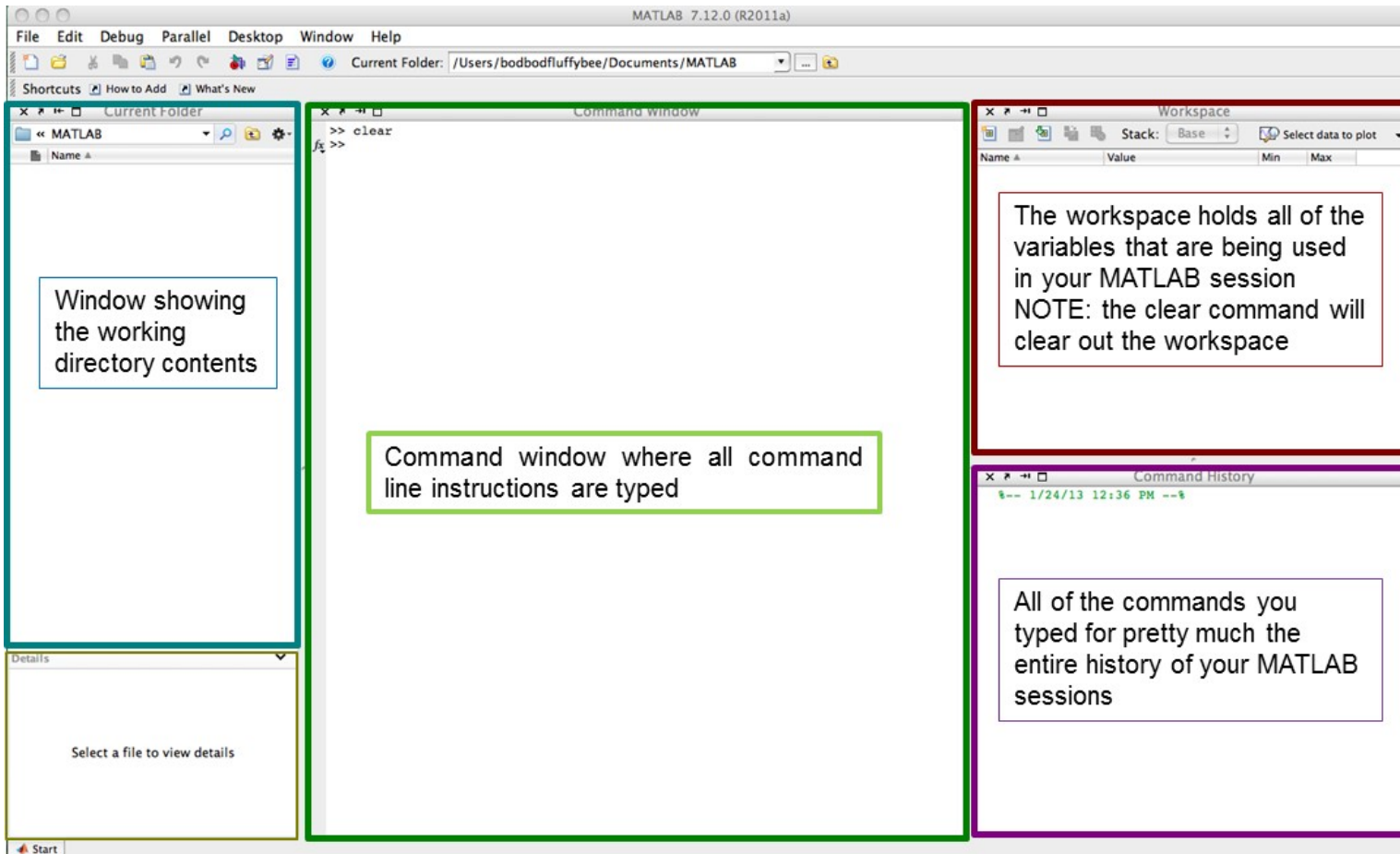
## Introduction

MathWorks describes their software package MATLAB -

*“MATLAB® is the high-level language and interactive environment used by millions of engineers and scientists worldwide. It lets you explore and visualize ideas and collaborate across disciplines including signal and image processing, communications, control systems, and computational finance.”*

During the lab you will be using MATLAB to create plots, control and read data from the bench equipment, simple computations, and for the brave as an interface with the Arduino. Let's start by using this powerful computational resource as a spreadsheet.

Notes:



**The MATLAB development environment** – depending on your computer and operating system the windows may be in different places – this is how mine looks on my Mac. The look and placement of the windows is customizable and all are detachable.

# Creating Plots in MATLAB

---

## Introduction

MATLAB is an extremely powerful (if inelegant) computational and educational environment as you will come to understand as you use it more. For the ECE 110 lab you will primarily be using it to create easy to interpret graphs, as an automated interface to your bench equipment making tasks like finding IV curves much faster, and as an interface to the Arduino microcontroller. This appendix introduces step-by-step how to generate graphs using two of the simpler methods.

The first method is to just use the working environment to input data by hand and use MATLAB in the same way you would use Excel – as a spreadsheet. The second method is more commonly used - graphing is usually done using MATLAB functions and variables either by entering them using the command line interface or by entering all of the commands into a script file so you only need to write the commands once.

# Using MATLAB as a spreadsheet

## Entering data using the workspace window

Step 1: create a new variable

The image shows three sequential screenshots of the MATLAB Workspace window, illustrating the steps to create a new variable:

- Screenshot 1:** The Workspace window is empty. A blue arrow points to the 'Add Variable' icon (a grid with a plus sign) in the toolbar. A blue text label 'Adds a new variable' is positioned to the left of the arrow.
- Screenshot 2:** A new variable named 'unnamed' has been added to the workspace. The 'Value' column shows '0', and the 'Min' and 'Max' columns also show '0'. A blue arrow points to the 'unnamed' text in the Name column. A blue text label 'Name the variable IVcurve and hit return' is positioned to the left of the arrow.
- Screenshot 3:** The variable is now named 'IVcurve'. The 'Value' column shows '0', and the 'Min' and 'Max' columns also show '0'.

## Entering data using the workspace window

Step 2: enter data

Double click on the icon to bring up the variable editor

Workspace

Name	Value
IVcurve	0

Variable Editor - IVcurve

1	2	3
1	0	
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		
21		
22		
23		
24		
25		
26		

Variable Editor - IVcurve

1	2	3	4	5	6	7	8	9	10
1	1	27.1828							
2	1.2000	33.2012							
3	1.4000	40.5520							
4	1.6000	49.5303							
5	1.8000	60.4965							
6	2	73.8906							
7	2.2000	90.2501							
8	2.4000	110.2318							
9	2.6000	134.6374							
10	2.8000	164.4465							
11	3	200.8554							
12									
13									
14									
15									
16									
17									
18									
19									
20									
21									
22									
23									
24									
25									
26									

Enter voltage in col. 1  
Enter current in col. 2

voltage  
current

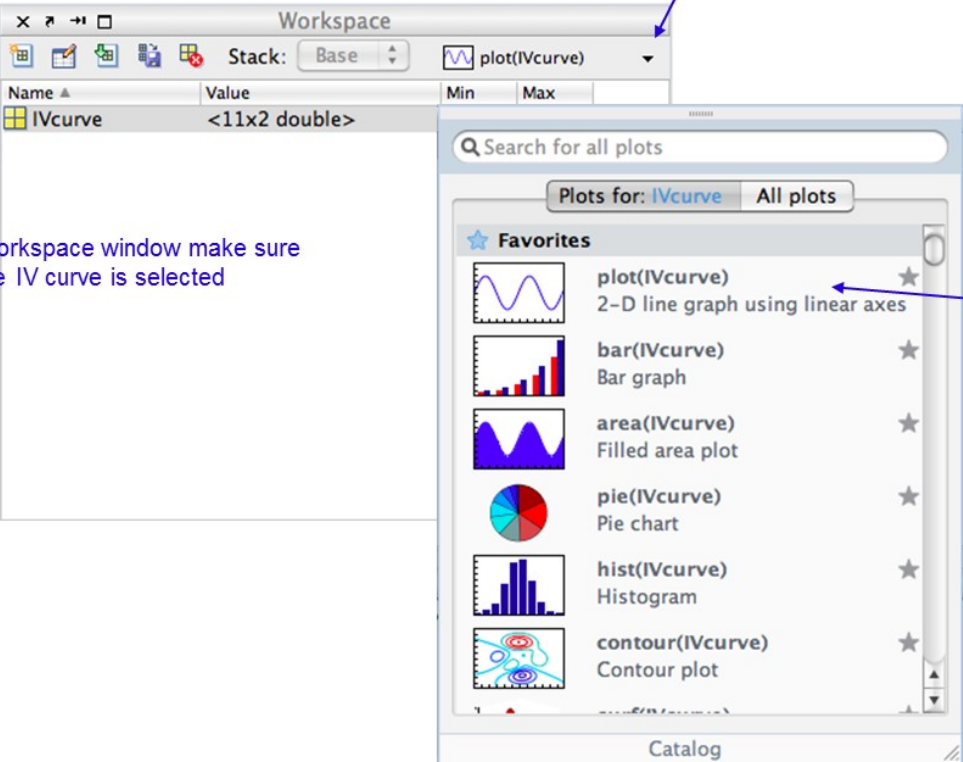
## Data plotting using the workspace window

Step 1: choose plot type

1. In the workspace window make sure the variable IV curve is selected

2. click the arrow beside the menu option plot(IVcurve) to bring down plot menu

3. choose the 2-D line graph option



The screenshot shows a software interface with a 'Workspace' window and a 'Catalog' window. The 'Workspace' window has a table with the following data:

Name	Value	Min	Max
IVcurve	<11x2 double>		

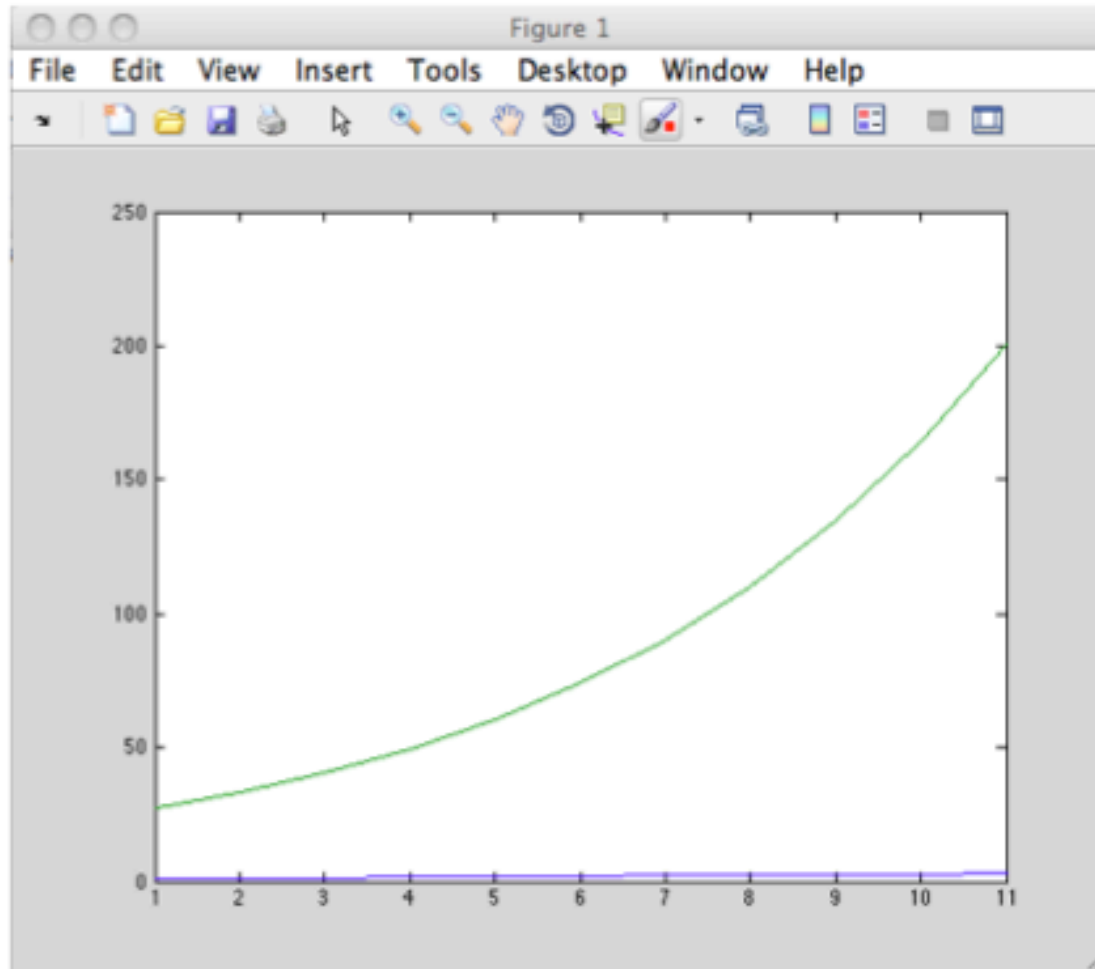
The 'Catalog' window displays a search bar and a list of plot types for the variable 'IVcurve':

- plot(IVcurve) 2-D line graph using linear axes
- bar(IVcurve) Bar graph
- area(IVcurve) Filled area plot
- pie(IVcurve) Pie chart
- hist(IVcurve) Histogram
- contour(IVcurve) Contour plot

Arrows indicate the following steps: 1. An arrow points to the 'IVcurve' variable in the workspace table. 2. An arrow points to the dropdown arrow next to 'plot(IVcurve)' in the workspace window. 3. An arrow points to the 'plot(IVcurve) 2-D line graph using linear axes' option in the catalog window.

Notes:

**Data plotting** using the workspace window (cont.) - futzing with the look of the graph



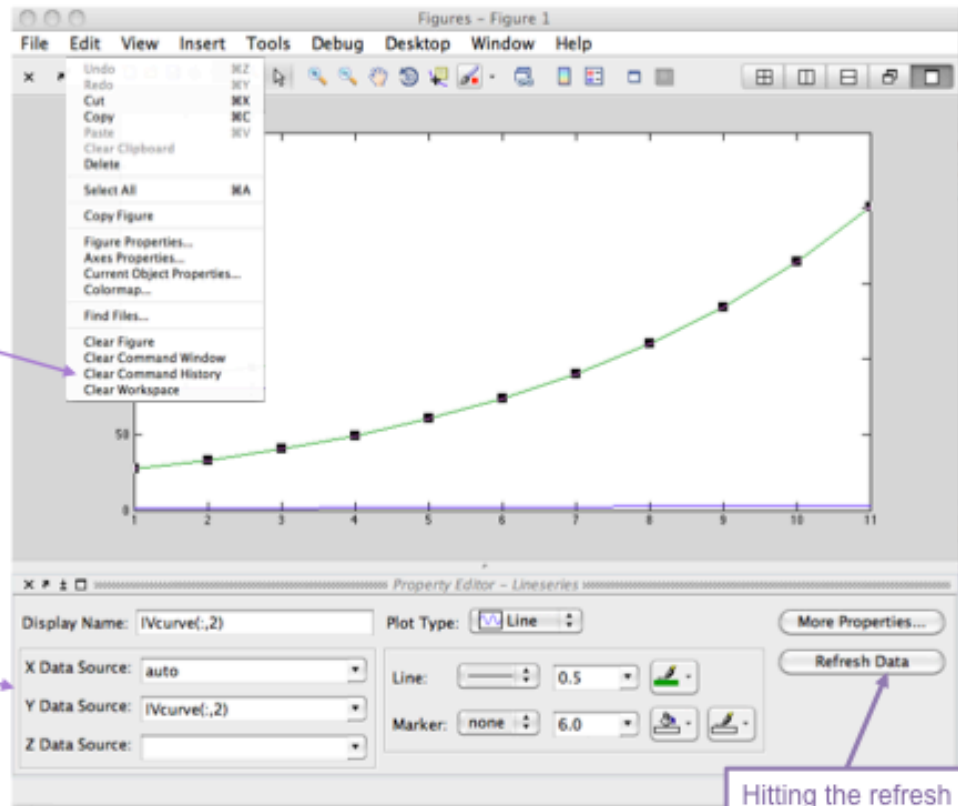
A new window should pop-up with the plot of data in column 1 against the row number (blue line on this graph) and the data in column 2 also against the row number.

This is NOT the graph we want. We want the data in col. 1 to be the x-values and the data in col. 2 to be the y-values so we need to change how MATLAB is plotting the data.

Data plotting using the workspace window (cont.) – more futzing

In the "Edit" pull-down menu select "Current Object Properties"...

...and a window opens up underneath the plot.



Hitting the refresh data button re-plots the graph

Change x data source so that the voltage values are on x-axis - IVcurve(:,1) - and the current values are on the y-axis.

NOTE: IVcurve(:,1) indicates the values in the first column of the array IVCurve



### Data plotting using the workspace window (cont.)

Notes:

Now click on the other line and push the delete button...

...to get the I-V plot

Property Editor - Lineseries

Display Name: |Vcurve(:,2) Plot Type: |V Line

X Data Source: |Vcurve(:,1) Line: 0.5

Y Data Source: |Vcurve(:,2) Marker: none 6.0

Z Data Source:

Property Editor - Axes

Title:

Colors: Y Label: Ticks...

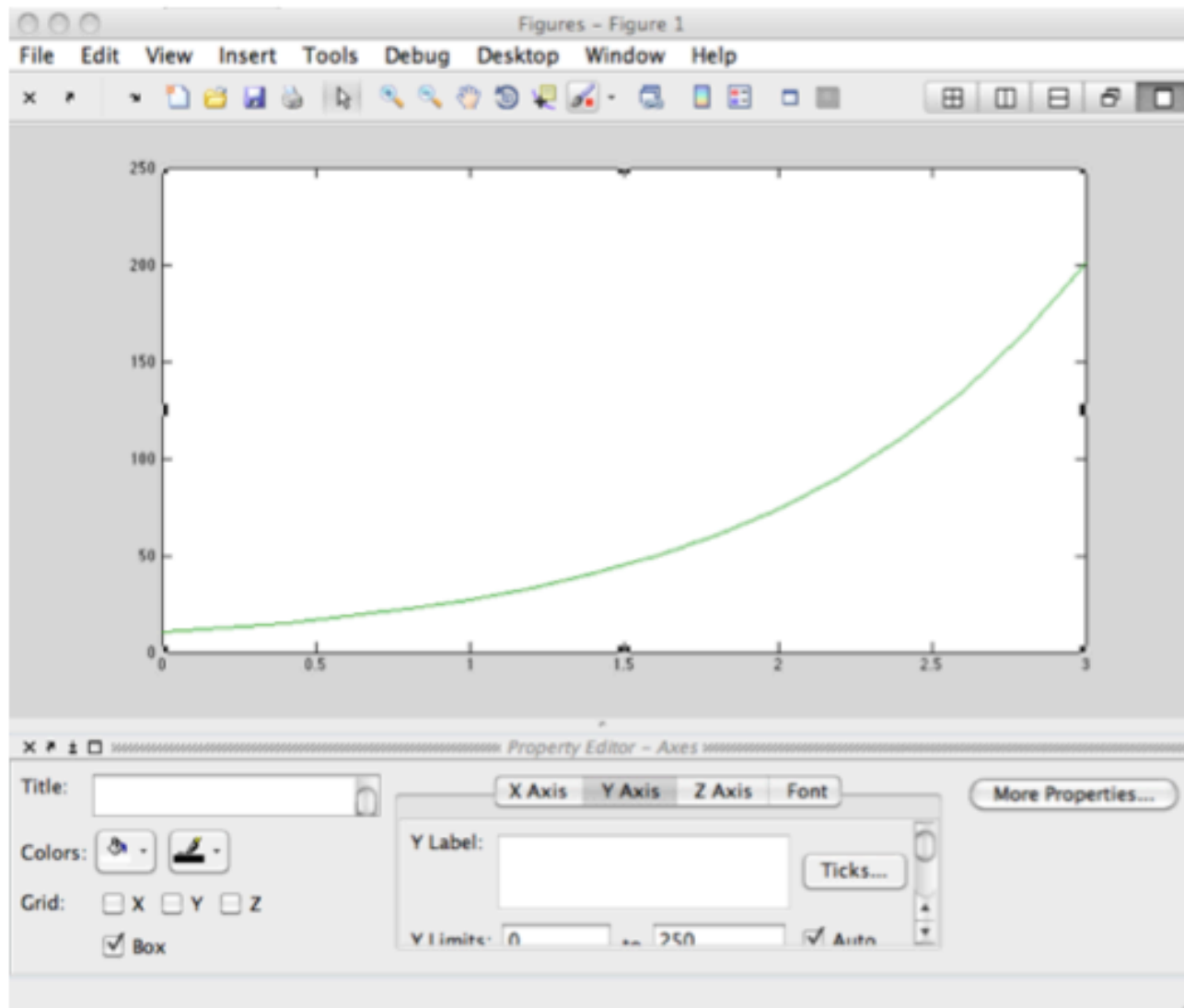
Grid:  X  Y  Z  Box

Y limits: 0 to 250  Auto

Notes:

### Data plotting using the workspace window

Step 4: Add annotations by investigating editing functions. Give the x- and y-axis labels and add a title. Find out how to change the color of the line. Add some gridlines.



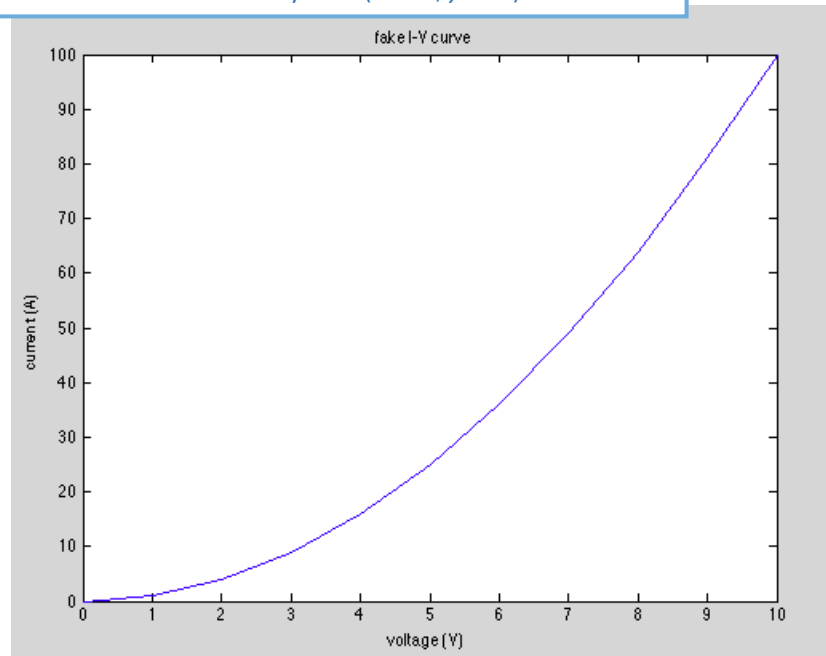
# Plotting Using MATLAB's Command Line Interface

As some of you know who have used MATLAB before, there is a much simpler way to create the same graphs by typing commands into the command window. These 6 lines of MATLAB code, entered line-by-line into the command window, creates the same plot generated through manipulating the variable `Ivcurve` in the workspace.

```
>> voltage=[0,1,2,3,4,5,6,7,8,9,10];  
>> current=[0,1,4,9,16,25,36,49,64,81,100];  
>> plot(voltage,current)  
>> title('fake I-V curve')  
>> xlabel('voltage (V)')  
>> ylabel('current (A)')  
>>
```

The first two statements are used to enter the voltage and current data into two Matlab vector

The next 4 statements use pre-defined Matlab functions *plot*, *title*, *xlabel*, and *ylabel* open a window and create basic plot (*plot*), add a title to the plot (*title*), and add labels to the x- and y-axes (*xlabel*, *ylabel*).





## How MATLAB Stores and Interprets Data – Common Data Structures

### Number

- a single stored value is considered a data structure, like a table with 1 row and 1 column

### Vector

- a data structure that holds a list of values is stored as a 1xn data structure with 1 row and n columns

### Matrix

- a general data structure with an arbitrary number of dimensions. *Numbers* and *vectors* are considered matrices

```
Command Window

This is a Classroom License for instructional use only.
Research and commercial use is prohibited.

>> a=123
a =
    123
>> size(a)
ans =
     1     1
>> b=[1,2,3,4,5,6,7,8,9,10]
b =
     1     2     3     4     5     6     7     8     9    10
>> size(b)
ans =
     1    10
>> c=[11,12,13;21,22,23;31,32,33]
c =
    11    12    13
    21    22    23
    31    32    33
>> size(c)
ans =
     3     3
>> d=magic(15);
>> e=1:100;
fx >> f=[1:5;6:10;11:15]
```

## Entering Different Data Types from the command line -

The image shows two windows from the MATLAB environment: the Command Window and the Workspace.

**Command Window:**

```
This is a Classroom License for instructional use only.  
Research and commercial use is prohibited.  
>> a=123  
a =  
    123  
>> size(a)  
ans =  
     1     1  
>> b=[1,2,3,4,5,6,7,8,9,10]  
b =  
     1     2     3     4     5     6     7     8     9    10  
>> size(b)  
ans =  
     1    10  
>> c=[11,12,13;21,22,23;31,32,33]  
c =  
    11    12    13  
    21    22    23  
    31    32    33  
>> size(c)  
ans =  
     3     3  
>> d=magic(15);  
>> e=1:100;  
fx >> f=[1:5;6:10;11:15]
```

**Workspace:**

Name	Value	Min	Max
a	123	123	123
ans	[3,3]	3	3
b	[1,2,3,4,5,6,7,8,9,10]	1	10
c	[11,12,13;21,22,23;31,32,33]	11	33
d	<15x15 double>	1	225
e	<1x100 double>	1	100

Annotations:

- An arrow points from the text box "size(variable\_name) is a function that returns the size of each dimension of the matrix 'variable\_name'" to the `size(a)` command in the Command Window.
- An arrow points from the text box "Entering matrices with 2 or more dimensions requires additional syntax – a semi-colon is used in this example to separate data in different rows" to the `c=[11,12,13;21,22,23;31,32,33]` command in the Command Window.

All of the variables show up in the Workspace

## Accessing Different Data Types from the command line -

```
>> a1=[1,2,3;4,5,6;7,8,9]
```

```
a1 =
```

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

```
>> a2=[1,2,3,4,5,6,7,8,9];
```

```
>> a2=reshape(a2,3,3)
```

```
a2 =
```

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

```
>> a2=transpose(a2)
```

```
a2 =
```

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

2 ways to create the same 3x3 matrix

```
>> a1(2,3)
```

```
ans =
```

```
    6
```

← Accesses a single element of a the 3x3 matrix 'a1'

```
>> a1(:,3)
```

```
ans =
```

```
    3
```

```
    6
```

```
    9
```

← Accesses a whole column of a the 3x3 matrix 'a1'. Guess how you access a single row...

```
>> a1(3,3)
```

```
ans =
```

```
    9
```

← Matlab lets you know when you have breached the array boundaries

```
>> a1(3,4)
```

```
??? Index exceeds matrix dimensions.
```

```
>> a1(9)
```

← Isn't a1 a 3x3 matrix?

```
ans =
```

```
    9
```

```
>>
```

When you used the command...

>plot(voltage, current)

... you are using the pre-defined MATLAB function plot. Below is the help page describing how to use the plot function. As you can see there are many properties of the plot that you can set as a parameter inside the plot function. The title, x-axis label, and the y-axis label can all be specified inside the plot function. As you use MATLAB more and more you will realize that there are at least 5 different ways of doing any one task.

[output1,output2,output3,...]=function\_name(input1,input2,input3) ← Standard syntax for calling functions

The screenshot shows the MATLAB Help browser interface. The search bar contains 'plot'. The left pane displays search results for 'plot' across various toolboxes, including MATLAB, Simulink, Fixed-Point Toolbox, System Identification Toolbox, Financial Toolbox, Wavelet Toolbox, and Curve Fitting Toolbox. The right pane shows the main documentation for the 'plot' function, including its syntax, description, and examples.

**plot**  
2-D line plot

**Syntax**

```
plot(Y)
plot(X1,Y1,...,Xn,Yn)
plot(X1,Y1,LineStyle,...,Xn,Yn,LineStyle)
plot(X1,Y1,LineStyle,'PropertyName',PropertyValue)
plot(axes_handle,X1,Y1,LineStyle,'PropertyName',PropertyValue)
h = plot(X1,Y1,LineStyle,'PropertyName',PropertyValue)
```

**Description**

**plot(Y)** plots the columns of Y versus the index of each value when Y is a real number. For complex Y, **plot(Y)** is equivalent to **plot(real(Y),imag(Y))**.

**plot(X1,Y1,...,Xn,Yn)** plots each vector Yn versus vector Xn on the same axes. If one of Yn or Xn is a matrix and the other is a vector, plots the vector versus the matrix row or column with a matching dimension to the vector. If Xn is a scalar and Yn is a vector, plots discrete Yn points vertically at Xn. If Xn or Yn are complex, imaginary components are ignored. If Xn or Yn are matrices, they must be 2-D and the same size, and the columns of Yn are plotted against the columns of Xn. **plot** automatically chooses colors and line styles in the order specified by **ColorOrder** and **LineStyleOrder** properties of current axes.

**plot(X1,Y1,LineStyle,...,Xn,Yn,LineStyle)** plots lines defined by the Xn,Yn,LineStyle triplets, where **LineStyle** specifies the line type, marker symbol, and color. You can mix Xn,Yn,LineStyle triplets with Xn,Yn pairs:  
**plot(X1,Y1,X2,Y2,LineStyle,X3,Y3)**.

**plot(X1,Y1,LineStyle,'PropertyName',PropertyValue)** manipulates **plot** characteristics by setting **lineseries** **properties** (of **lineseries** graphics objects created by **plot**). Enter properties as one or more name and value pairs.

**plot(axes\_handle,X1,Y1,LineStyle,'PropertyName',PropertyValue)** plots using axes with the handle **axes\_handle** instead of the current axes (**gca**).

**h = plot(X1,Y1,LineStyle,'PropertyName',PropertyValue)** returns a column vector of handles to **lineseries** objects, one handle per line.

**Examples**

**Plot a sine curve:**

```
x = -pi:1:pi;
```