***MATLAB***

**MATLAB** (an abbreviation of "MATrix LABoratory"[[22]](https://en.wikipedia.org/wiki/MATLAB#cite_note-22)) is a [proprietary](https://en.wikipedia.org/wiki/Proprietary_software) [multi-paradigm](https://en.wikipedia.org/wiki/Multi-paradigm_programming_language) [programming language](https://en.wikipedia.org/wiki/Programming_language) and [numeric computing](https://en.wikipedia.org/wiki/Numerical_analysis) environment developed by [MathWorks](https://en.wikipedia.org/wiki/MathWorks). MATLAB allows [matrix](https://en.wikipedia.org/wiki/Matrix_(mathematics)) manipulations, plotting of [functions](https://en.wikipedia.org/wiki/Function_(mathematics)) and data, implementation of [algorithms](https://en.wikipedia.org/wiki/Algorithm), creation of [user interfaces](https://en.wikipedia.org/wiki/User_interface), and interfacing with programs written in other languages.

Although MATLAB is intended primarily for numeric computing, an optional toolbox uses the [MuPAD](https://en.wikipedia.org/wiki/MuPAD" \o "MuPAD) [symbolic engine](https://en.wikipedia.org/wiki/Computer_algebra_system) allowing access to [symbolic computing](https://en.wikipedia.org/wiki/Symbolic_computing) abilities. An additional package, [Simulink](https://en.wikipedia.org/wiki/Simulink), adds graphical multi-domain simulation and [model-based design](https://en.wikipedia.org/wiki/Model-based_design) for [dynamic](https://en.wikipedia.org/wiki/Dynamical_system) and [embedded systems](https://en.wikipedia.org/wiki/Embedded_system).

***BASICS OF MATLAB(MATLAB ONRAMP)***

1. ***COMMANDS***

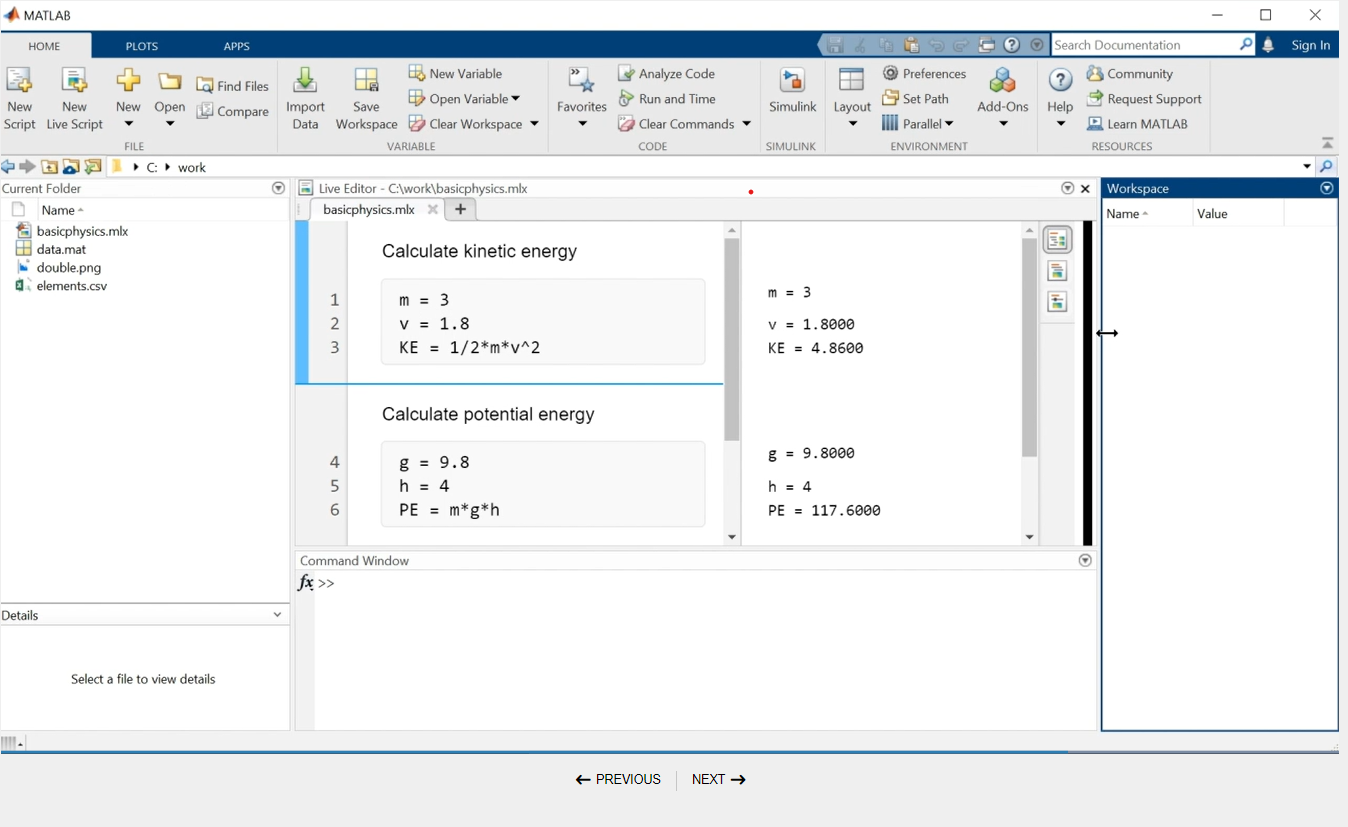
* We can execute commands by entering them in the Command Window after the MATLAB prompt (>>) and pressing the **Enter** key.
* Unless we specify an output variable, MATLAB stores calculations in a variable named Ans.
* Workspace window shows all the variables created. Semi colon stops the result of the command from displaying. However, it still gets reflected in the workspace.
* When you enter just a variable name at the command prompt, MATLAB displays the current value of that variable.
* ‘=’ is used to assign the right term to the left term.
* **Variable-** Variable names starts with a letter and can contain letter only a letter, a number or ‘\_’.
* We can save a file by using the command

>> save filename

* Clear function helps to clear all the variables from a workspace.
* Load function helps to load a specific file.
* To display the content of a variable we just need to enter the name of the variable in the command window.
* ‘clc’ command cleans up the command window.
* **Built-in function-**

**Examples-** sqrt, pi, abs, max, sin, cos, eig, etc.

***2.MATLAB Desktop and Editor***



* The MATLAB Editor-

on can enter commands in a script by clicking the grey code box.

* Live script-

 live script contains formatted text, code, and section breaks.

* Error

Errors will prevent your code from running. If an error occurs while MATLAB is running your program, or if MATLAB detects a significant issue in your code, the status indicator in the top right becomes an error icon . The incorrect code will also be underlined in red.

3- Vectors and Matrices

* All MATLAB variables are *arrays*. This means that each numeric variable can contain multiple numbers. You can use arrays to store related data in one variable.
* A single number, called a *scalar*, is actually a 1-by-1 array, meaning it contains 1 row and 1 column.
* You can create arrays with multiple elements using square brackets.
* x = [3 5]
* When we separate numbers by semicolons, MATLAB creates a *column vector* (*n*-by-1).
* We can combine spaces and semicolons to create a *matrix*, which is an array with multiple rows and columns. When entering a matrix, we enter them row by row
* Creating evenly spaced vectors- The’:’ operator creates evenly spaced vectors

Syntax- start:spacing:end default spacing in 1.

* Linspace- We use this when we know the number of elements instead of the spacing between them.

Syntax- linspace(*first*,*last*,*number\_of\_elements*)

* We use transpose operation (‘) to create column vectors.
* Rand- create matrix with random elements of the specified size.

Syntax- rand(x,y)

* ones and zeros- creates matrices of ones and zeroes.

4- Indexing into and Modifying arrays

* To extract an element from a matrix we use A(m,n).
* We can reference the last term as end.
* To extract a column vector we use A(:,n).
* To extract more than one row we use A(x:y,:).
* We can reference directly to a specific element of a row or a column vector.
* To extract a specific range of elements we use

x = v(3:end).

* We can use the assignment operator to change values of elements from matrices.

5- Array Calculations

* MATLAB is designed to work naturally with arrays. For example,
* we can add a scalar value to all the elements of an array,
* add two array of same size,
* multiply or divide all of the elements of an array by a scalar,
* use basic statistical functions on a vector to produce a single output,
* perform a function on every element of an array,
* The .\* operator performs elementwise multiplication and allows us to multiply the corresponding elements of two equally sized arrays.

6- Calling Functions

* The size function can be applied to an array to produce a single output variable containing the array size in a two element array, where the first element is the number of rows and the second element is the number of columns.

s = size(x)

* The size function can be applied to a matrix to produce either a single output variable or two output variables. Use square brackets ([ ]) to obtain more than one output.

[xrow,xcol] = size(x)

* We can find the maximum value of a vector and its corresponding index value using the max function. The first output from the max function is the maximum value of the input vector. When called with two outputs, the second output is the index value.

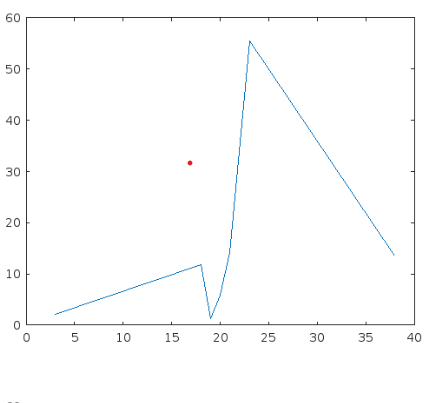
[xMax,idx] = max(x)

7- Obtaining Help

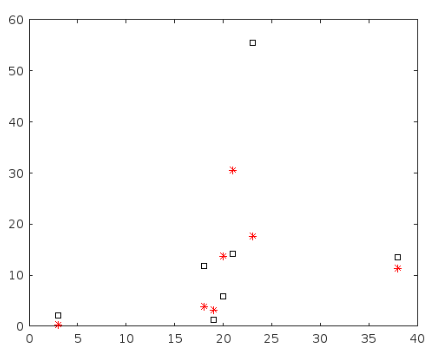
The help section in MATLAB can be used to obtain information on various features of MATLAB.

8- Plotting data

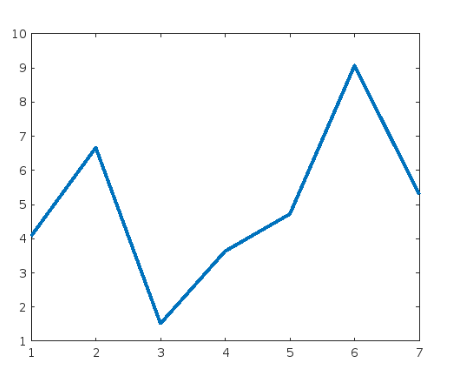
* Two vectors of the same length can be plotted against each other using the plot function.  
  plot(x,y)



* We can use another input to the plot function to specify the color, line style, and marker style using different symbols in double quotes.  
  plot(x,y,"r--o")



* The command above plots a red (r) dashed (--) line with circle (o) markers. We can learn more about the symbols available in the documentation for [Line Specification](https://www.mathworks.com/help/matlab/ref/linespec.html).
* Hold on command is used to plot the next set of variables on the previous axes.
* Hold off command is used to plot the next set of variables on new pair of axes.
* When we plot a single vector by itself, MATLAB uses the vector values as the y-axis data and sets the x-axis data to range from 1 to n (the number of elements in the vector).  
  plot(y)
* The plot function accepts optional additional inputs consisting of a property name and an associated value.  
  plot(y,"LineWidth",5)



* The command above plots a heavy line. We can learn more about available properties in the documentation for [Line Properties](https://www.mathworks.com/help/matlab/ref/matlab.graphics.chart.primitive.line-properties.html).
* ANNONATING PLOTS

-we can use “title” function to label the plot

-we can use ‘ylabel’ and’xlabel’ to label the axes

-we can use legend to add legends to the graph

Syntax- legend (“a”, “b”)

9- Importing data from a table

* To extract a variable from the table, we can use *dot notation*:  
  *data.VariableName*
* We can interact with a table by clicking on it in the output pane of a live script. For example, we can sort a table using one of its variables. Once we are happy with our table, we can make the changes permanent by updating the code in your script.

10- Logical Arrays

Logical Indexing

* Relational operators, such as >, <, ==, and ~= perform comparisons between two values. The outcome of a comparison for equality or inequality is either 1 (true) or 0 (false).
* We can compare an array to a single scalar value using relational operators. The result is a logical array of the same size as the original array
* We can use a logical array as an array index, in which case MATLAB extracts the array elements where the index is 1 (true).
* We can use logical indexing to reassign values in an array.

Syntax- x(x==999) = 1

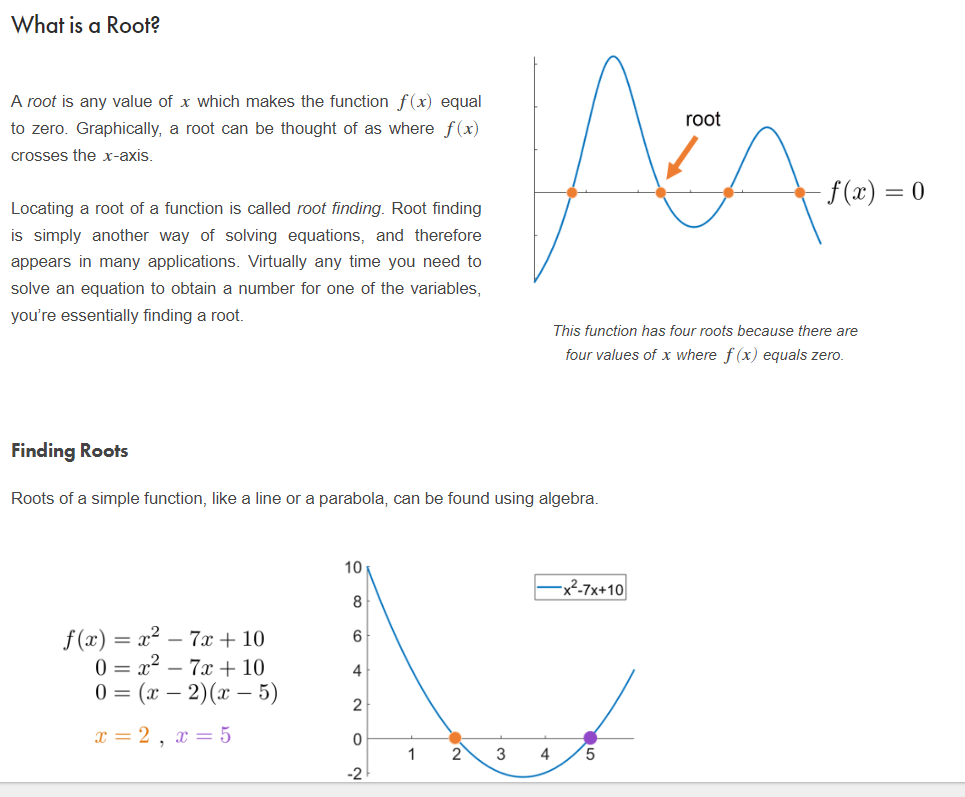
11-Programming

* If-else is similar to that in C, if the condition in if is true only then the next statement will be executed or else the else condition will be executed.
* For loop is used when we need to make iterations.
* While loops-[while](http://www.mathworks.com/help/matlab/ref/while.html)

Repeat when condition is true.

Solving Nonlinear Equations with MATLAB

* What is Root finding?

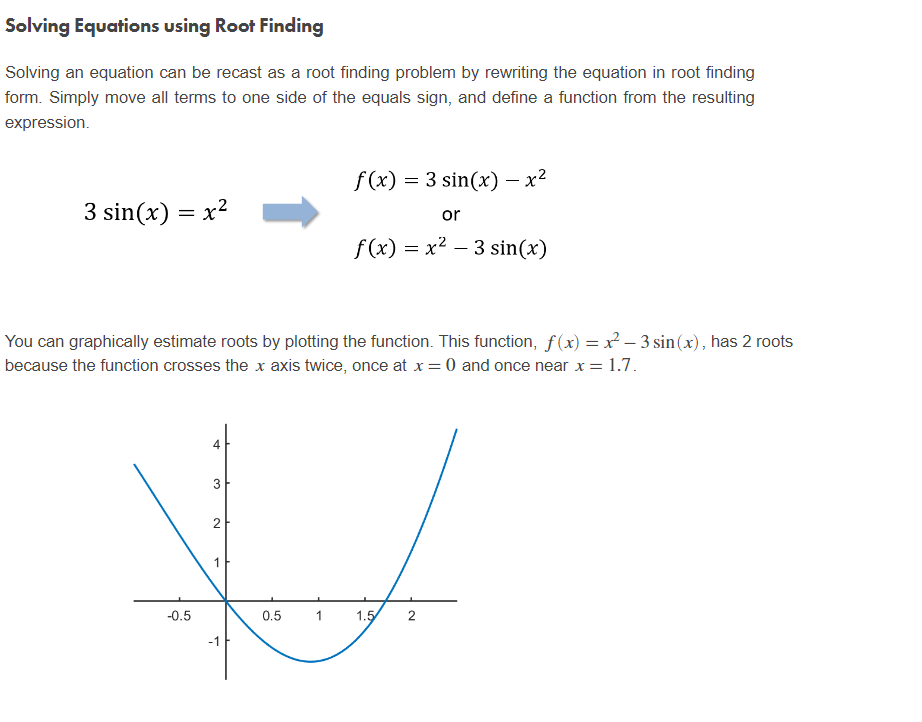


**Definitions**

* A *root* of a function *f*(*x*) is a value of *x* for which *f*(*x*)=0.

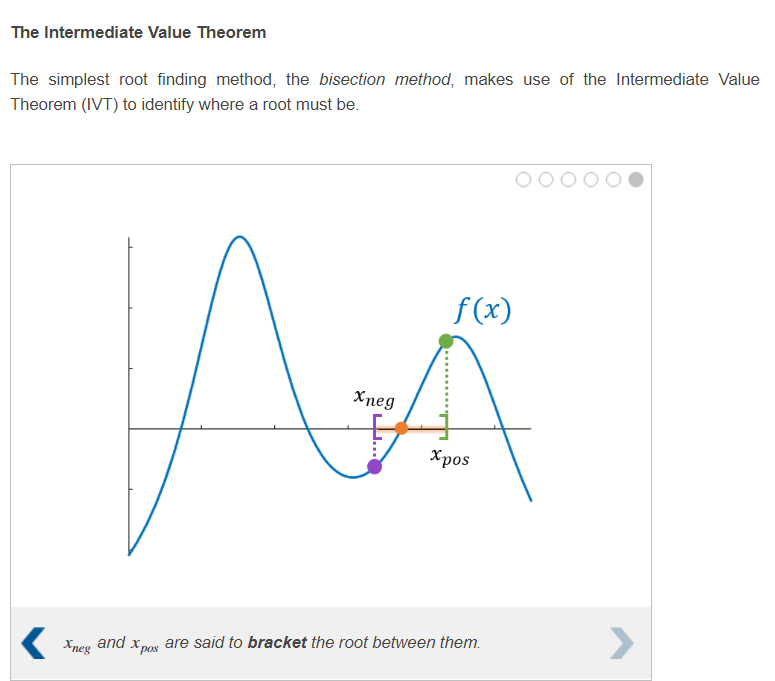
.

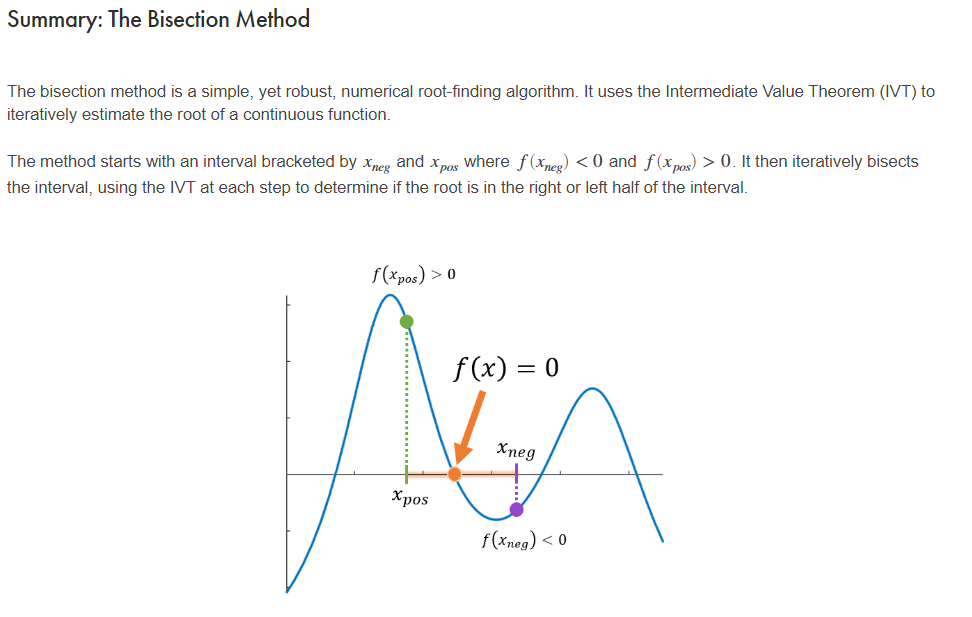
* *Root finding* is any method for locating the roots of a function.

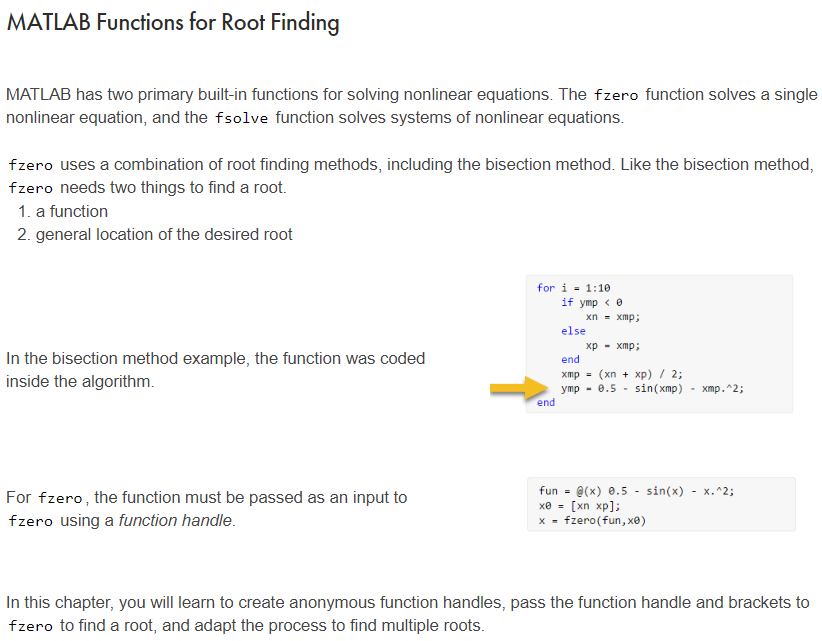


* Finding a Root

The Bisection Method-







### Finding the Roots of a Polynomial

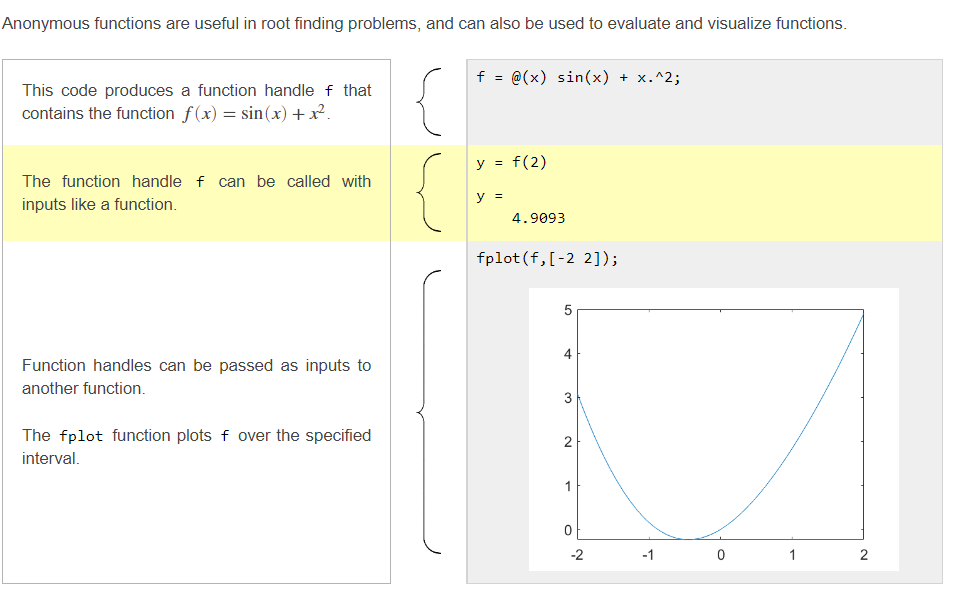
The fzero function is a broadly applicable MATLAB root finding function. If your root finding problem is a polynomial, you may choose to use the roots function instead. More information on roots can be found in the documentation.

* [**fzero**](https://www.mathworks.com/help/matlab/ref/fzero.html)
* ***Solve a nonlinear equation.***
* [**fsolve**](https://www.mathworks.com/help/optim/ug/fsolve.html)
* ***Solve a system of nonlinear equations.***
* [**roots**](http://www.mathworks.com/help/matlab/ref/roots.html)
* ***Find the roots of a polynomial.***

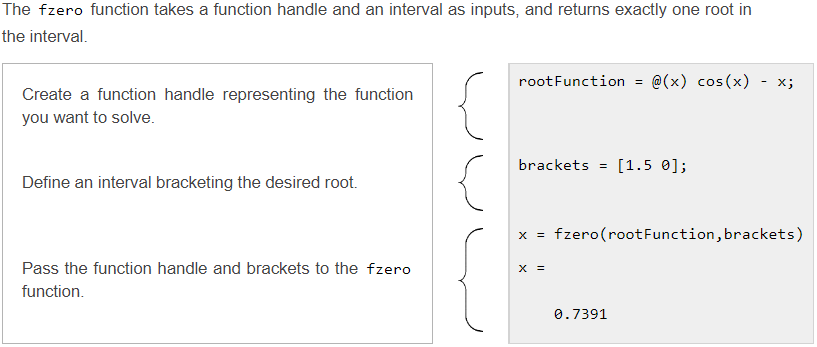
Anonymous Function-

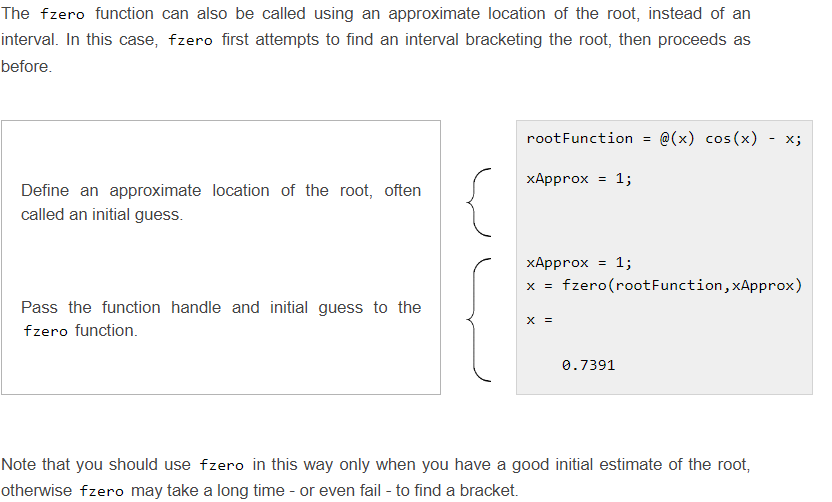
An anonymous function is a way to define a simple function and store it as a variable in the workspace.

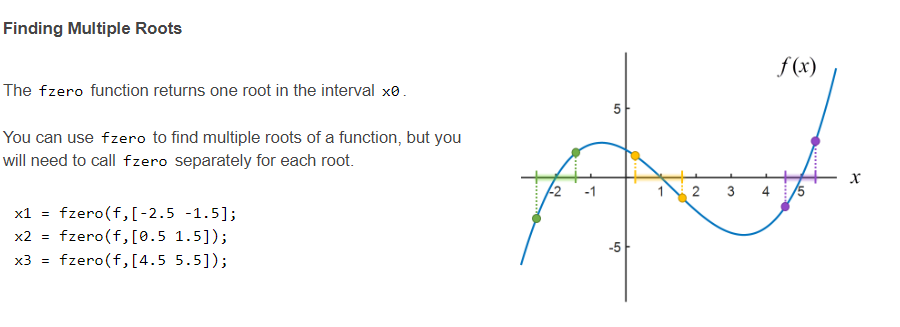
Syntax- myFun=@(x) x.^exp(-x)

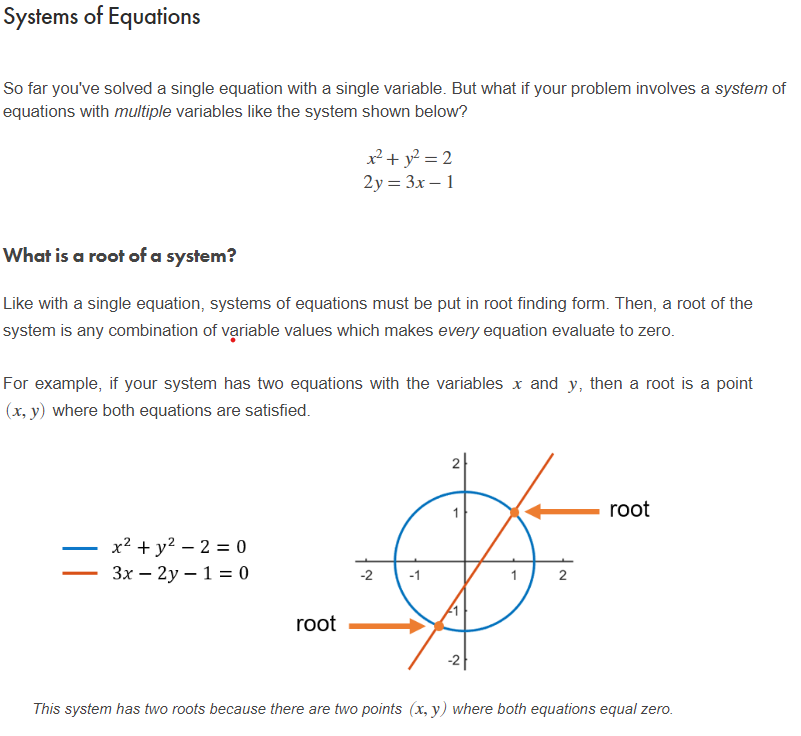


* fzero function-

******







# **Solving Systems in MATLAB**

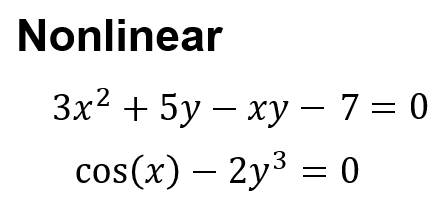
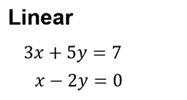
The general case of a system of nonlinear equations can be solved with the fsolve function.  
  
Like fzero, the fsolve function needs two things to find a root.

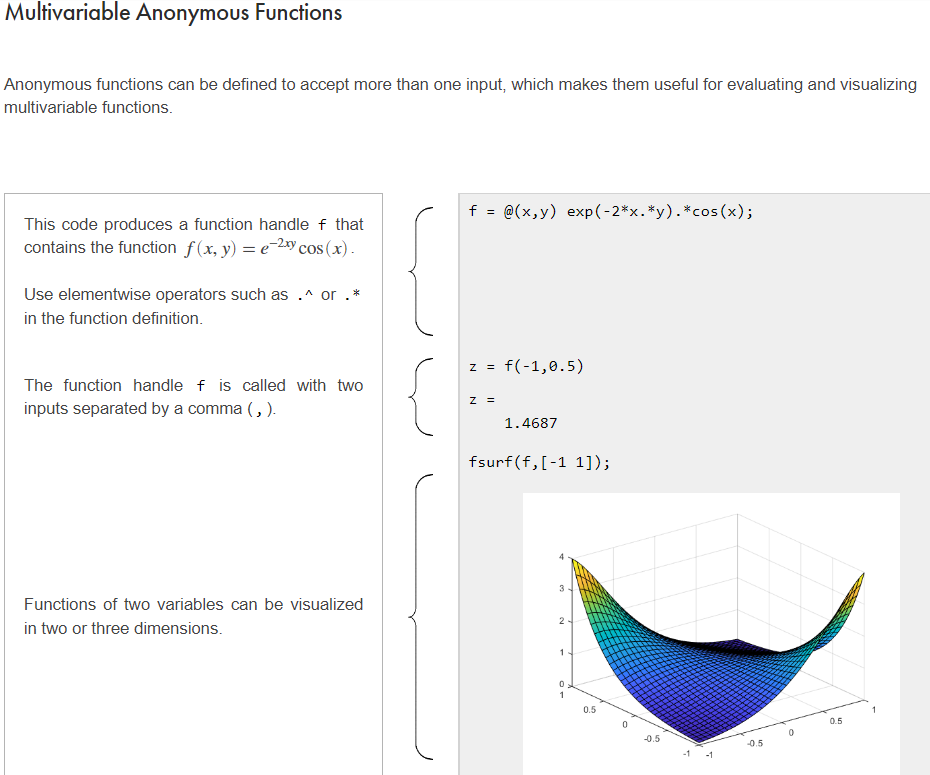
1. a function
2. general location of the desired root

In this chapter, you will learn to visualize multivariable functions, create anonymous function handles that represent a system of equations, and pass the function handle and initial guess to fsolve to solve the system.

### Special case: Linear Systems

A special case of systems of equations is a *linear* system. Linear systems can be put into matrix-vector form and solved with the backslash operator (\). You can learn how to solve linear systems in the course [Introduction to Linear Algebra with MATLAB](https://matlabacademy.mathworks.com/en/selfpaced/linalg).

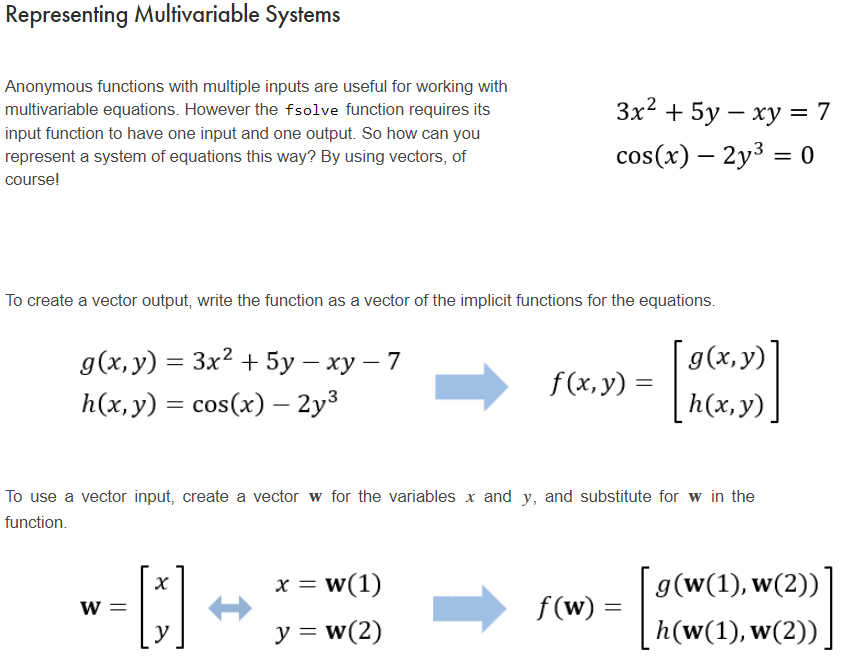


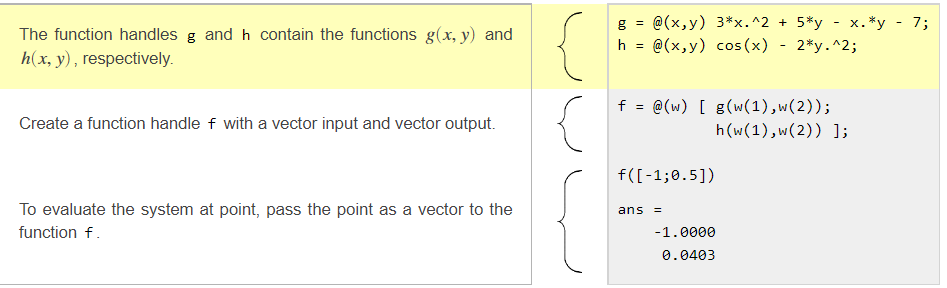


Implicit Functions-

The circle *x*2+*y*2=4is an example of an *implicit equation*. Implicit equations are used when you can't easily write *y* as a function of *x*. So instead of using the relationship *y*=*f*(*x*), the relationship is defined by an implicit function *f*(*x*,*y*)=0.

Use fimplicit to plot the graph of a implicit function similar to fsurf.





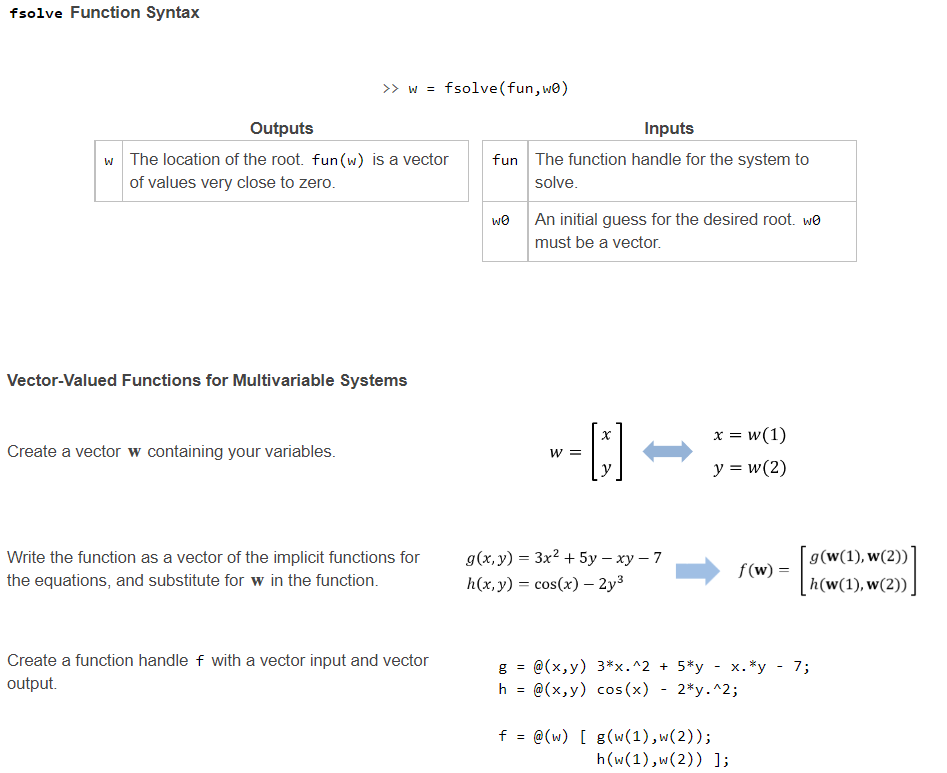
* fsolve function

fsolve takes two inputs — a function handle to the system of equations to solve and an initial guess at the solution — and it returns the location of the root.

. Systems and multiple variables should be treated as vectors.

* The variables *x* and *y* are represented as a two element vector w.
* The function definition returns a two element column vector, each element representing the corresponding equation in the system.
* The initial guess is also in the form of the vector w.
* The solution found by fsolve is also in the form of w.

Example-



**Solving Ordinary Differential Equations with MATLAB**

* The word *differential* is used to mean "related to derivatives", so an equation containing a derivative is called a *differential equation*. When there is only one independent variable (like all the examples in this course), the equation is called an ***ordinary differential equation*** (**ODE**).
* The *initial condition* (or *boundary condition*) is a value of the dependent variable defined for a value of the independent variable. You need this information in order to solve the ODE numerically.
* The solution to a differential equation is a function, not a single number. Methods of calculus can determine the solution exactly, but they are beyond the scope of this course. You can also solve ODEs approximately using numerical methods in MATLAB, which results in vectors of values representing the independent and dependent variables.

# **MATLAB ODE Solver Syntax**

>> [t,y] = ode45(@odefun,tspan,y0)

**Outputs**

|  |  |
| --- | --- |
| t | A vector containing the values of the independent variable at which the numerical solution was computed. |
| y | A vector of the computed value of the dependent variable at each time in t. |
| **Inputs** | |

|  |  |
| --- | --- |
| @odefun | A function handle to the derivative function. |
| tspan | A two element vector containing the initial and final values for the independent variable. |
| y0 | The initial value of the dependent variable. |
|  | |

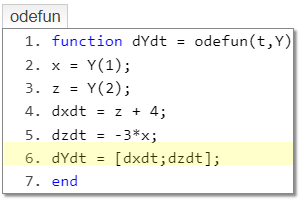
* A solution structure contains information about the ODE and its numerical solution, such as which solver was used and the evaluation points. You can use it to evaluate the solution at any point in the original interval.
* We can use the deval function to evaluate the solution at any point in the interval by passing the solution structure and the point to deval.

Syntax- deval(solution structure, value of independent variable)

Solving Systems of ODE Numerically

* To solve system of ODE using ode45 function take vector input.

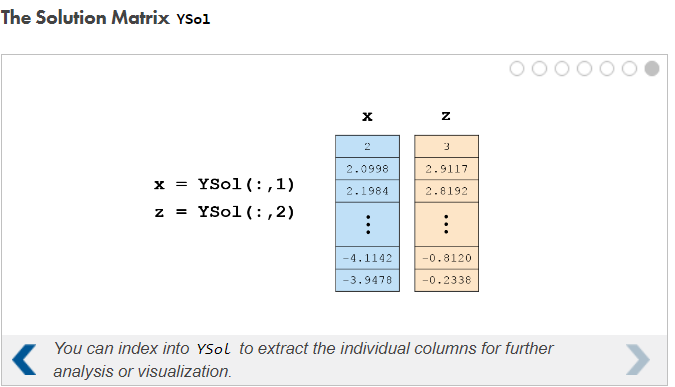
Ex-



**Writing Vector-Valued ODE functions**

Creating a vector-valued ODE function can be broken down into three parts:

* extracting the individual elements from the vector of dependent variables,
* implementing the derivative functions for the dependent variables,
* returning the output variable as a column vector of the derivatives.

* 

Higher Order Derivative

## A higher-order derivative simply refers to how many times a derivative has been applied to a function.

## Solving Second-Order ODEs using ode45

Because we can write any higher-order ODE as a system of first-order ODEs, the process for solving them is exactly the same as in Chapter 4, only with one extra first step.  
  
**1. Preparing the ODE**

* Rewrite the Nth-order ODE as a system of N first-order ODEs.

**2. Writing the vector-valued ODE function**

* Extract the dependent variable and its derivatives from the input vector in ode45.
* Implement the derivative functions for the extracted variables.
* Return the output variable as a column vector of the derivatives.

While using ode45 to solve higher order differential equations, initial value should be given of the different differentials and the solution also contains columns corresponding to differentials of different orders

