**Matlab Tutorial**

**Entering Commands**

* Simple mathematical operations can be performed just by entering them on the console window. Eg: 3\*5 This prints the output as 15.
* Variables can be used to store values. Data type of the variable is not mentioned. Eg: m=3\*5 This prints the output as m=15.
* Values in variables can be modified in the following way : m=m+1. This stores 15+1 =16 in m.
* Writing a semicolon at the end prevents the statement from executing. Eg: m=3\*5 ; This doesn’t print anything.
* Suppose m=16, n=m/2 is executed. Then if m is changed to 20, n doesn’t change to 10 unless n=m/2 is run again.

**Variable names**

* Variable names consist of only letters, numbers and underscores and starts only with letters. They are case sensitive.

**Workspace / Commands for command window.**

* To save a workspace, type save ‘filename’.mat.
* To load variables from a file in the workspace, type load ‘filename’.mat.
* Use clear to empty the workspace.
* Type the name of the variable to display its contents.
* Use clc to clean the command window.
* To load only a variable from an existing file, use load filename variable\_name.
* To save a variable data in another file, use save filename variable\_name.

**Built-in Functions :** Some constants like pi, trigonometric functions, sqrt are built-in. To print a long decimal number, use : format long (enter) variable\_name. Else to print till 4 decimal places, format short (enter) variable\_name.

**Understanding Live Editor**

* Section Break creates a new section of code.
* Run section runs a particular section.
* Run executes the entire code.
* Text and code buttons for respective environments.

**Arrays:** To initialise an array, use square brackets.

**Creating a matrix :** matrix = [2 3 4; 1, 2, 4; 3,7,9]

This creates the matrix :

2 3 4

1 2 4

3 7 9

**Creating a row vector having an Arithmetic Progression:**

d= linspace( first, last, number of elements)

Default number of divisions is 100.

Or

d= (first: common difference: last)

**Creating a column vector:**

d=d’. Returns the transpose.

**Rand( size) :** Creates a matrix of random inputs of given size. Size is given as (number of rows, number of columns). If the number of columns is not specified, a square matrix is taken.

**Size(matrix):** Returns the size of a given matrix.

**Zeros(size), Ones(size) :** Returns matrices of given size consisting of only 0, 1 respectively.

**Indexing :**

* Starts from 1 (not 0).
* Vector (index) .
* Matrix(row,column)
* Matrix(:,column): All rows of a column.
* Matrix(row,:) : All columns of a row.(end,6) or (6,end)
* Matrix(index) : Returns the value . Iterates through the rows of a column. i.e . go downwards.
* Matrix(:,2:4) Returns 2nd ,3rd, 4th columns of a matrix.
* Variable = matrix([1 3 5]) Stores 1st,3rd and 5th values of vector in the vector called variable.
* matrix(i,j)=9. To Change the value.

Array Operations

1. x=[1 2 3]

y=x+1

y =[2 3 4]

1. y=x\*2
2. y=x/2
3. y=sqrt(x)
4. y=x1 + x2. Add 2 arrays of the same size.
5. Maximum =max(array)
6. v=round(v) : Round off elements to the nearest integer
7. M=m1\*m2. ==matrix multiplication.
8. m =m1 .\* m2 == element wise multiplication (dot product)
9. a=[1 2;3 4;5 6; 7 8]

b=[1;2;3;4]

x = a.\*b

X = 1 2 ; 6 8 ; 15 18 ; 28 32

If a function returns more than one output, store the outputs by defining the same number of variables. Eg:

1. [xrow,xcol] = size(x)
2. [xMax,idx] = max(x)
3. [xMin,idx] = min(x)
4. Use a tilde (~) to ignore specific outputs. Eg: [~,idx] = min(x)

**Plotting a Curve**

plot(x,y, “colour shape line/no-line”, “Linewidth”,5)

Colour: r,b,k,g

Shape: s(square), o (circle), \*(stars)

Hold on : same axes for both graphs. Add next line on same plot

Hold off: different axes

title(“Title name”)

ylabel(“name”), xlabel

legend(“a”,”b”,”c”)

Importing Files

Access a column from a file : file\_name.column\_name

**Working with Tables**

Add another column using data from other columns.

data.Mass = data.Density .\* data. Volume1

Sorting a column:

data = sortrows(data,"Colm\_name")

Logical Indexing:

* extract all elements in v1 that are greater than six. v = v1(v1 > 6)
* contains the elements in the sample corresponding to where v1 is less than 4. a=sample(v1<4)
* any value less than 4 is replaced with the value 0. v1(v1<4)=0
* Symbol for and : & , symbol for or : |

**IF-ELSE**

if condition1

code

elseif condition2

code

end

For printing, use disp() function

**FOR LOOP**

for c = 1:10

disp(c)

End

**Finding Roots**

Method :

X = linspace(-a,a)

Y = f(X)

plot(X,Y)

To write f(X), transfer everything into one side of the equation.

Intermediate value theorem guarantees a root between xp and xn if y(xp)>0 and y(xn)<0.

**Bisection Method :**

The method starts with an interval bracketed by *xneg* and *xpos* where *f*(*xneg*)<0 and *f*(*xpos*)>0. It then iteratively bisects the interval, using the IVT at each step to determine if the root is in the right or left half of the interval.

**MATLAB has two primary built-in functions for solving nonlinear equations. The fzero function solves a single nonlinear equation, and the fsolve function solves systems of nonlinear equations.**

**Creating a function handle:**

f = @(x) sin(x) + x.^2;

fplot(fun,[-x0,x1]) : This will plot the graph.

# **Using the fzero Function**

The fzero function takes a function handle and an interval as inputs, and returns exactly one root in the interval.

fzero(function, interval of variable)

**fsurf(f,[-1 1])** to visualise 3d surfaces.

The function handles g and h contain the functions *g*(*x*,*y*) and *h*(*x*,*y*), respectively.

g = @(x,y) 3\*x.^2 + 5\*y - x.\*y - 7;

h = @(x,y) cos(x) - 2\*y.^2;

Create a function handle f with a vector input and vector output : f = @(w) [ g(w(1),w(2));

h(w(1),w(2)) ];

To evaluate the system at point, pass the point as a vector to the function f : f([-1;0.5])

**fimplicit(f,interval) : to plot implicit functions.**

**Example**

g=@(x,y) 2\*x.\*y -2\*y.^2 -1

h=@(x,y) y.^3 - cos(x)

fimplicit(g,[-3,3])

hold on

fimplicit(h,[-3,3])

hold off

f=@(w) [g(w(1),w(2)),h(w(1),w(2))]

w=[-2;-1]

wRoot=fsolve(f,w)

**fsolve**

w = fsolve(fun,w0)

| w | The location of the root. fun(w) is a vector of values very close to zero. |
| --- | --- |

| fun | The function handle for the system to solve. |
| --- | --- |
| w0 | An initial guess for the desired root. w0 must be a vector. |

**ODE SOLVING**

ODE45 Function

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

| 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. |

| @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. |

Defining a function:

function dCdt = reactor(t,C)

dCdt = 2\*t - 1/2\*t^2;

end

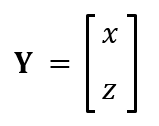
Here dCdt is the variable, reactor is the function handle.

Function is always defined at the last of the file.

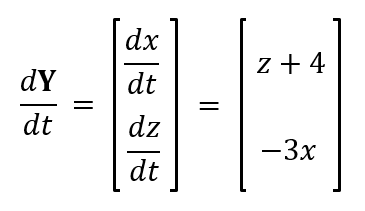
* Sol =ode45(@odefun,tspan,y0) . Here Sol will be a structure containing both t and y. A solution structure containing information about the solution, such as the solver and evaluation points. Use deval to evaluate the solution sol at any point in tspan.

**To solve a system of ODE:**

The dependent variables *x* and *z* as a single dependent variable column vector, **Y**.



The derivative of **Y** is now also a column vector, and it contains the derivative functions of *x* and *z*.



function dYdt = solve(t,Y)

x = Y(1);

z = Y(2);

dxdt = z + 4;

dzdt = -3\*x;

dYdt = [dxdt;dzdt];

end

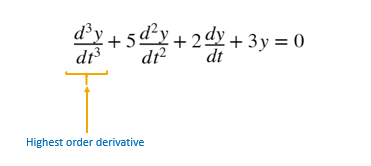
Also, create a column vector Y0 for the vector of initial conditions:

Y0 = [2;3];

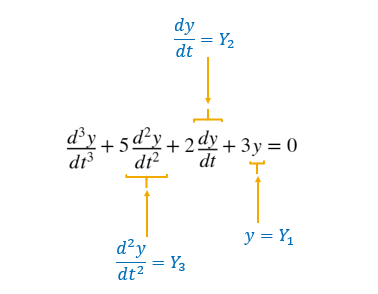
Solving higher order derivatives

Solving a third-order ODE requires us to rewrite it as a system of three first-order ODEs. Fortunately, the process is the same as with rewriting second-order ODEs.

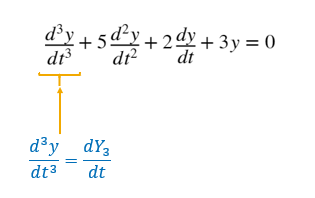
The highest order derivative indicates the ODE is third-order.



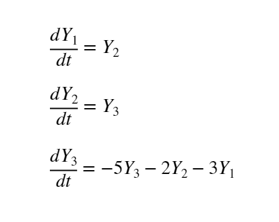
Rewrite the lower order terms that include the dependent variable as *Y*1 , *Y*2, and *Y*3.



Rewrite the third derivative as the derivative of *Y*3.



We now have a system of three first-order ODEs that can be input to ode45 as vector-valued equations.



Example of a function:

function dYdt = harmonicMotion(t,Y)

%Define constants k, m, and b

k = 50;

m = 0.05;

b = 0.5;

% TODO - Extract the position x from the first element of Y.

y = Y(1);

% TODO - Extract the velocity v from the second element of Y.

v = Y(2);

% TODO - Complete expression for dxdt

dxdt = v;

% TODO - Complete expression for dvdt

dvdt = -(k/m)\*y - (b/m)\*v;

% TODO - Create dYdt, column vector containing dxdt, and dvdt

dYdt = [dxdt;dvdt];

end

Question : Define the ODE function for 

Answer:

function dYdt = tacomaNarrows(t,Y)

Theta=Y(1);

Omega=Y(2);

Y1=Theta;

dY1dt = Omega;

dY2dt = -0.01\*dY1dt - 2.4\*cos(Y1).\*sin(Y1) + 0.05\*sin(1.3\*t);

dYdt=[dY1dt;dY2dt];

end