MATLAB BASICS TUTORIAL

Matlab is an interactive program for numerical computation and data visualization; it is used extensively by control engineers for analysis and design. There are many different toolboxes available which extend the basic functions of Matlab into different application areas; in these tutorials, we will make extensive use of the Control Systems Toolbox. Matlab is suppor ted on Unix, Macintosh, and Windows environments; a student version of Matlab is available for personal computers. For more information on Matlab, contact the Mathworks.

The idea behind these tutorials is that you can view them in one window while running Matlab in another window. You should be able to re-do all of the plots and calculations in the tutorials by cutting and pasting text from the tutorials into Matlab or an m-file.

Vectors

Let's start off by creating something simple, like a vector. Enter each element of the vector (separated by a space) between brackets, and set it equal to a variable. For example, to create the vector a, enter into the Matlab command window (you can "copy" and "paste" from your browser into Matlab to make it easy):

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

Matlab should return:

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

Let's say you want to create a vector with elements between 0 and 20 evenly spaced in increments of 2 (this method is frequently used to create a time vector):

```
t = 0:2:20
t =
    0  2  4  6  8  10  12  14  16  18  20
```

Manipulating vectors is almost as easy as creating them. First, suppose you would like to add 2 to each of the elements in vector 'a'. The equation for that looks like:

```
b = a + 2

     b =

        3  4  5  6  7  8  11  10  9
```

Now suppose, you would like to add two vectors together. If the two vectors are the same length, it is easy. Simply add the two as shown below:

```
c = a + b

     c =

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

Subtraction of vectors of the same length works exactly the same way.

Functions

To make life easier, Matlab includes many standard functions. Each function is a block of code that accomplishes a specific task. Matlab contains all of the standard functions such as sin, cos, log, exp, sqrt, as well as many others. Commonly used constants such as pi, and i or j for the square root of -1, are also incorporated into Matlab.

```
sin(pi/4)

ans =

    0.7071
```

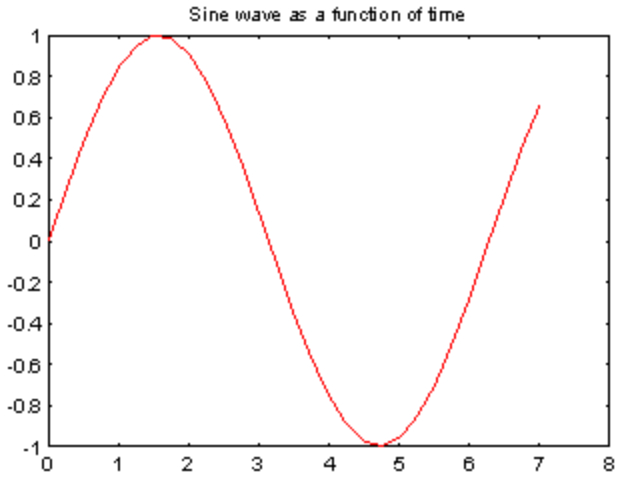
To determine the usage of any function, type `help [function name]` at the Matlab command window.

Matlab even allows you to write your own functions with the `function` command; follow the link to learn how to write your own functions and see a listing of the functions we created for this tutorial.

Plotting

It is also easy to create plots in Matlab. Suppose you wanted to plot a sine wave as a function of time. First make a time vector (the semicolon after each statement tells Matlab we don't want to see all the values) and then compute the sin value at each time.

```
t=0:0.25:7;
y = sin(t);
plot(t,y)
```



The plot contains approximately one period of a sine wave. Basic plotting is very easy in Matlab, and the `plot` command has extensive add-on capabilities. I would recommend you visit the plotting page to learn more about it.

Polynomials

In Matlab, a polynomial is represented by a vector. To create a polynomial in Matlab, simply enter each coefficient of the polynomial into the vector in descending order. For instance, let's say you have the following polynomial:

$$s^4 + 3s^3 - 15s^2 - 2s + 9$$

To enter this into Matlab, just enter it as a vector in the following manner

```
x = [1 3 -15 -2 9]
```

```
x =
     1  3  -15  -2  9
```

Matlab can interpret a vector of length n+1 as an nth order polynomial. Thus, if your polynomial is missing any coefficients, you must enter zeros in the appropriate place in the vector. For example,

$$s^4 + 1$$

would be represented in Matlab as:

```
y = [1 0 0 0 1]
```

You can find the value of a polynomial using the `polyval` function. For example, to find the value of the above polynomial at s=2,

```
z = polyval([1 0 0 0 1],2)
```

```
z =
    17
```

You can also extract the roots of a polynomial. This is useful when you have a high-order polynomial such as

$$s^4 + 3s^3 - 15s^2 - 2s + 9$$

Finding the roots would be as easy as entering the following command;

```
roots([1 3 -15 -2 9])
```

```
ans =
   -5.5745
    2.5836
   -0.7951
    0.7860
```

Let's say you want to multiply two polynomials together. The product of two polynomials is found by taking the convolution of their coefficients. Matlab's function `conv` that will do this for you.

```
x = [1 2];
```

```
y = [1 4 8];
z = conv(x,y)

    z =
        1  6  16  16
```

Dividing two polynomials is just as easy. The `deconv` function will return the remainder as well as the result. Let's divide z by y and see if we get x.

```
[xx, R] = deconv(z,y)

    xx =
        1  2
    R =
        0  0  0  0
```

As you can see, this is just the polynomial/vector x from before. If y had not gone into z evenly, the remainder vector would have been something other than zero.

If you want to add two polynomials together which have the same order, a simple z=x+y will work (the vectors x and y must have the same length). In the general case, the user-defined function, polyadd can be used. To use `polyadd`, copy the function into an m-file, and then use it just as you would any other function in the Matlab toolbox. Assuming you had the polyadd function stored as a m-file, and you wanted to add the two uneven polynomials, x and y, you could accomplish this by entering the command:

```
z = polyadd(x,y)

    x =
        1  2

    y =
        1  4  8

    z =
```

```
1  5  10
```

Matrices

Entering matrices into Matlab is the same as entering a vector, except each row of elements is separated by a semicolon (;) or a return:

```
B = [1 2 3 4;5 6 7 8;9 10 11 12]

B =
    1    2    3    4
    5    6    7    8
    9   10   11   12


B = [ 1  2  3  4
      5  6  7  8
      9 10 11 12]

B =
    1    2    3    4
    5    6    7    8
    9   10   11   12
```

Matrices in Matlab can be manipulated in many ways. For one, you can find the transpose of a matrix using the apostrophe key:

```
C = B'

C =
    1   5   9
    2   6   10
    3   7   11
    4   8   12
```

It should be noted that if C had been complex, the apostrophe would have actually given the complex conjugate transpose. To get the transpose, use .' (the two commands are the same if the matix is not complex).
Now you can multiply the two matrices B and C together. Remember that order matters when multiplying matrices.

```
D = B * C

D =
    30    70   110
    70   174   278
   110   278   446

D  = C * B

D =
   107   122   137   152
   122   140   158   176
   137   158   179   200
   152   176   200   224
```

Another option for matrix manipulation is that you can multiply the corresponding elements of two matrices using the .* operator (the matrices must be the same size to do this).

```
E = [1 2;3 4]
F = [2 3;4 5]
G = E .* F


E =
     1   2
     3   4

F =
     2   3
```

```
     4   5
```

```
G =
     2   6
    12  20
```

If you have a square matrix, like E, you can also multiply it by itself as many times as you like by raising it to a given power.

```
E^3
```

```
ans =
    37    54
    81   118
```

If wanted to cube each element in the matrix, just use the element-by-element cubing.

```
E.^3
```

```
ans =
     1     8
    27    64
```

You can also find the inverse of a matrix:

```
X = inv(E)
```

```
X =
   -2.0000    1.0000
    1.5000   -0.5000
```

or its eigenvalues:

```
eig(E)
```

```
ans =
   -0.3723
    5.3723
```

There is even a function to find the coefficients of the characteristic polynomial of a matrix. The "poly" function creates a vector that includes the coefficients of the characteristic polynomial.

```
p = poly(E)

p =

    1.0000   -5.0000   -2.0000
```

Remember that the eigenvalues of a matrix are the same as the roots of its characteristic polynomial:

```
roots(p)


ans =
    5.3723
   -0.3723
```

Printing

Printing in Matlab is pretty easy. Just follow the steps illustrated below:

Macintosh

To print a plot or a m-file from a Macintosh, just click on the plot or m-file, select Print under the File menu, and hit return.

Windows

To print a plot or a m-file from a computer running Windows, just selct Print from the File menu in the window of the plot or m-file, and hit return.

Unix

To print a plot on a Unix workstation enter the command:

print -P<printername>

If you want to save the plot and print it later, enter the command:

print plot.ps

Sometime later, you could print the plot using the command "lpr -P plot.ps" If you are using a HP workstation to print, you would instead use the command "lpr -d plot.ps"

To print a m-file, just print it the way you would any other file, using the command "lpr -P <name of m-file>.m" If you are using a HP workstation to print, you would instead use the command "lpr -d plot.ps<name of m-file>.m"

Using M-files in Matlab

There are slightly different things you need to know for each platform.

Macintosh

There is a built-in editor for m-files; choose "New M-file" from the File menu. You can also use any other editor you like (but be sure to save the files in text format and load them when you start Matlab).

Windows

Running Matlab from Windows is very similar to running it on a Macintosh.

However, you need to know that your m-file will be saved in the clipboard. Therefore, you must make sure that it is saved as filename.m

Unix

You will need to run an editor separately from Matlab. The best strategy is to make a directory for all your m-files, then cd to that directory before running both Matlab and the editor. To start Matlab from your Xterm window, simply type: matlab.

You can either type commands directly into matlab, or put all of the commands that you will need together in an m-file, and just run the file. If you put all of your m-files in the same directory that you run matlab from, then matlab will always find them.

Getting help in Matlab

Matlab has a fairly good on-line help; type

help `command name`

for more information on any given command. You do need to know the name of the command that you are looking for; a list of the all the ones used in these tutorials is given in the command listing; a link to this page can be found at the bottom of every tutorial and example page.

Here are a few notes to end this tutorial.

You can get the value of a particular variable at any time by typing its name.

```
B
```

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

You can also have more that one statement on a single line, so long as you separate them with either a semicolon or comma.
Also, you may have noticed that so long as you don't assign a variable a specific operation or result, Matlab with store it in a temporary variable called "ans".

User feedback

We would like to hear about difficulties you had with the tutorials, suggestions you have for improvement, errors that you found, or any other comments that you have. This feedback is anonymous; include your email address if you want a reply.

Some Useful MATLAB Commands

First, the most important command in MATLAB:		
`>> help command_name`		Displays complete information about any command. Take note of any cross references at the end of each help entry. `help` with no command specified lists all categories of available commands. **Note:** `helpwin` **brings up a separate gui help window -- useful if you're doing other things in the main window.**

Interacting with MATLAB:		
`>> command` `>> command;`		A semicolon after any command suppresses output from that command to the MATLAB window (but not, for example, to a figure) - especially useful if the output is a very long vector.
`>> <UP ARROW>`		Recalls the last command entered. You can repeat it to go farther back, or press <DOWN ARROW> to scroll forward.
`>> abc<UP ARROW>`		Recalls the last command entered that starts with "abc".
`>> more on` `>> more off`		Turns on and off `more`, which displays all output (including help information) one screen at a time.
`>> who`		Tells you what variables currently exist.
`>> clear` `>> clear variable`		Deletes all variables, or the specified variable.

`>> format long g`
 	
`>> format compact`
		Displays variables (but not systems) with more digits. Does not affect storage or computation.
		Gets rid of the extra blank lines in the display.

Commands for making and formatting plots:		
`>> p1=plot(t1,y1);` `>> delete(p1)`		Plots y1 vs. t1. Note that setting the `plot` equal to a name lets you delete it afterwards - useful if you're plotting several things in the

		same figure and make a mistake. y1 and t1 must be vectors of the same size.
`>> plot(y(:,1),y(:,2))`		Plots the second column of matrix y vs. the first column. See note at end of section.
`>> subplot(m,n,p)`		Breaks the figure up into mxn separate graphs, and selects the p^{th} one as current; if there are already mxn graphs, leaves graphs as they are and selects the p^{th} one.

`>> xlabel('label_text')`
`>> ylabel('label_text')`
`>> title('title_text')`
		Labels the axes or the entire subplot with specified text (note single quotes).

`>> g1 = gtext('plot_text');` `>> delete(g1)`		Lets you place the given text (in single quotes) by clicking on the graph. Note that setting the `gtext` equal to a name lets you delete it afterwards.
`>> grid`		Toggles on and off a grid superimposed over the current graph; you can also type `grid on` and `grid off`.
`>> sgrid`		Draws a grid on a plot of the s plane (root locus plot or pole and zero locations), which consists of lines of constant damping coefficient (zeta) and natural frequency (wn).
`>> hold`		Toggles on and off the plot hold, which adds new plots to any already in the graph (without it, new plots delete previous ones). You can also type `hold on` and `hold off`.

`>> axis([xmin xmax ymin ymax])` `>> axis auto`		Sets the limits of the x and y axes manually, or lets them be set automatically. There are also many other options available for `axis`.
`>> zoom`		Lets you select an area of the plot (using the mouse) to zoom in on. Also, clicking the left mouse button once will zoom in, and clicking the right button will zoom out, by a factor of two. Double clicking the right mouse button returns to the original scale; typing `zoom` again turns off the zoom function.
`>> [x, y] = ginput(n)` `>> [x, y] = ginput`		Lets you input the coordinates of points on the graph with the mouse; collects n points and stores them in the vectors x and y, or if n is absent, keeps collecting points until you press <ENTER>.
Matrix and vector manipulation commands:		
`>> k=linspace(k1,k2)` `>> k=linspace(k1,k2,n)`		Returns a vector of 100 (or, if specified, n) points equally spaced between x1 and x2.
`>> max(y)`		Returns the largest element in the vector y.
`>> inv(A)`		Returns the inverse of the square, nonsingular matrix A.
`>> det(A)`		Calculates the determinant of the matrix A.
`>> eig(A)` `>> [V,D] = eig(A)`		Returns the eigenvalues of A, or sets V to a matrix containing the eigenvectors of A and sets D to a diagonal matrix containing the

		corresponding eigenvalues.
`>> rank(A)`		Returns the rank of any matrix A.
Commands useful in system analysis:		
`>> roots(f)`		Returns the roots of a polynomial, where f is a vector containing the coefficients of the polynomial.
`>> conv(P1,P2)`		Multiplies two polynomials (P1 and P2 are vectors containing the coeffiecients of the polynomials) and returns the resulting coefficients. This is actually a convolution of the two vectors, which also works as coefficient multiplication.

`>> sys1=tf(num,den)`
`>> sys2=ss(A,B,C,D)`
 	
`>> sys1=tf(sys2)`
 	
`>> sys2=ss(sys1)`
		Creates a system, as a transfer function or state-space representation. Also converts between two different representations of a system. The `zpk` (zero/pole/gain) command works similarly.

`>> [R,P,K]=residue(num,den)`		Finds the partial fraction expansion of a function H(s), where num is a vector containing the coefficients of the numerator, and den of the denominator, of H(s). Returns the numerators (R) and poles (P) of the partial fractions and the remaining polynomial (K): $H(s) = R_1/(s-P_1) + R_2/(s-P_2) + ... + R_n/(s-P_n) + K(s)$.
`>> sys3=series(sys1,sys2)`		Finds the result of putting Systems 1 and 2 in series, and returns either the resulting transfer

		function numerator and denominator or the resulting state space matrices. Mixing system descriptions will work.

`>> sys3=feedback(sys1,sys2)`
`>>`
`sys2=feedback(sys1,tf(1,1))`
		Finds the result of adding System 2 as a feedback loop to System 1, assuming a negative feedback, and returns either the resulting transfer function numerator and denominator or the resulting state space matrices.

`>> impulse(sys)` `>> step(sys)`		Plots the impulse response or step response of the given system. Useful trick: if you have a Laplace transform F(s) of a time function f(t), plotting the impulse response of F(s) is the same as plotting f(t).
`>> [y,t,x]=impulse(sys)` `>> [y,t,x]=step(sys)`		Giving `impusle` and `step` output variables returns the output (y), time (t), and states(x) (if state space) vectors, which you can then plot or manipulate.
`>> initial(sys,x0)`		Plots the behavior of the given state-space system with initial condition `x0` and no input.
`>> lsim(sys,u,t,x0)`		Plots the response of the given system to the input signal `u(t)`. The initial condition `x0` can be added for state-space systems.

`>> bode(sys1,sys2,...)`
`>>`
`[mag.phase,w]=bode(sys,w)`
`>> nyquist(sys)`
`>> {re,im,w]=nyquist(sys,w)`
		Plots the Bode or Nyquist diagram for the given system(s). A plot is drawn if no return arguments. The frequency points may be specified in the vector `w` .

`>> margin(sys)`
`>>`
`[gm,pm,wcg,wcp]=margin(sys)`
		Finds the gain margin and phase margin, and the frequencies at which they occur, of the given system. If run with no output arguments, `margin` also displays the Bode plot, with the margins marked on it and their values displayed in the figure title.

`>> rlocus(sys,k)`		Plots the root locus for the given system, i.e. where den(s) + k*num(s) = 0 (or equivalent for the state space form). The vector of values for k is optional.
`>> rlocfind(sys)` `>> [k,poles]=rlocfind(sys)`		Lets you select a pole location from a root locus plot using the mouse, and returns the value of `k` needed to give such a pole, as well as all the resulting pole locations in the vector `poles` (if present). `rlocfind` picks the point on the locus closest to the crosshairs; note that you must already have the root locus graphed to be able to see points you might want to pick.
`>> sys2=canon(sys1,'form')`		Finds a canonical form of the given system; the argument "form" can be either "modal" or "companion" (in single quotes).

List of Lab Exercises (ROTOR-1)

LAB No.	Experiment No.	Name of Experiment	Assigned Batch
1.	Experiment-0	Zero Lab	
2.	Experiment-1	To Understanding Images	
3.	Experiment-2	Understanding Image basics	
4.	Experiment-3	Image Arithmetic and logical Operations	
5.	Experiment-4	Image Geometric operations	

Experiment-1

Aim: to understand and implement the following task in MATLAB

- **Image Resizing**
- **Image type Conversion**
- **Line profile**

Some important Keywords

- Imread: reads the image from the file specified by filename, inferring the format of the file from its contents. If filename is a multi-image file, then imread reads the first image in the file.
- B = imresize(A,scale) returns image B that is scale times the size of A.
- Improfile : Pixel-value cross-sections along line segments

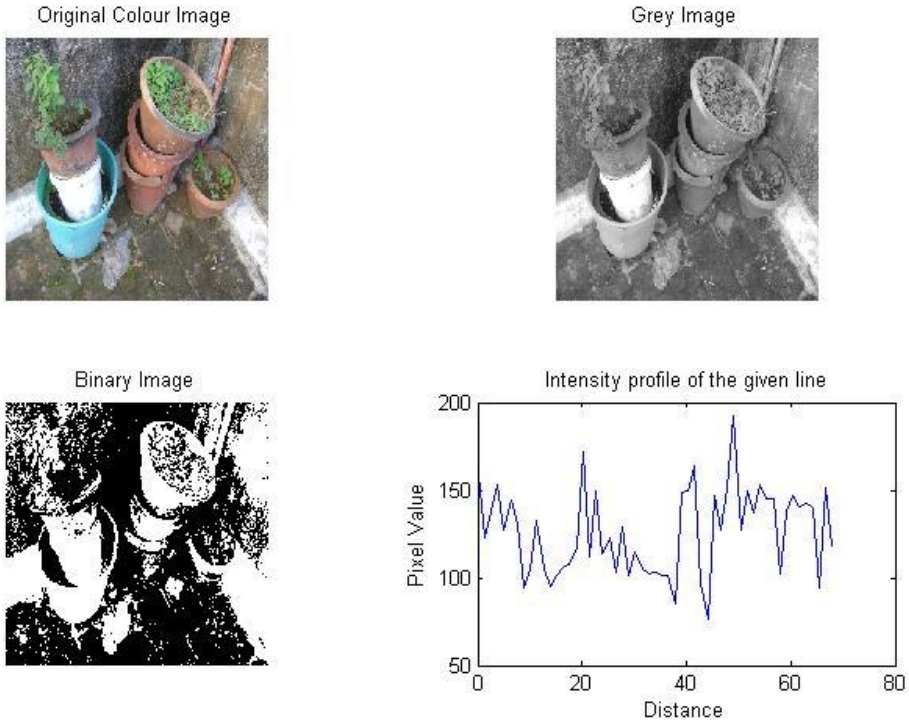
  This MATLAB function retrieves the intensity values of pixels along a line or a multiline path in the grayscale, binary, or RGB image in the current axes and displays a plot of the intensity values.

Matlab Code:

```
clc
clear all
close all

%read the colour image
myimage = imread('pot.jpg');

%resize the colour image to 256 x 256 image
mycolorimage=imresize(myimage,[256,256],'nearest');

%convert the colour image to a greylevel image
mygrayimage = rgb2gray(mycolorimage);
%convert the colour image to a greylevel image
mybinimage = im2bw(mycolorimage);

%Display the Original Image in a grid of 2 x 2 images
subplot(2, 2, 1);
imshow(mycolorimage); title('Original Colour Image');


%Display the Grey level Image in a grid of 2 x 2 images
subplot(2, 2, 2);
imshow(mygrayimage); title('Grey Image');


%Display the Original Image in a grid of 2 x 2 images
subplot(2, 2, 3);
imshow(mybinimage); title('Binary Image');


%Display the Line Profile of a line drawn with coordinates
```

```
%(10,45) and (50,100)

%Original Image in a grid of 2 x 2 images

subplot(2, 2, 4);
improfile(mygrayimage,[10,50],[45,100]);

%Display x axis, y-axis and Title of the Line profile graph

ylabel('Pixel Value');
xlabel('Distance');
title('Intensity profile of the given line');
```

Results:



Conclusion: Thus we have concluded that from this experiment we can perform image resizing, conversion to an image and calculate intensity graph of that particular image.

Experiment-2

Aim: to understand and implement the following task in MATLAB

- **Extraction of color bands**
- **performs pseudo-colouring**

Matlab Code:

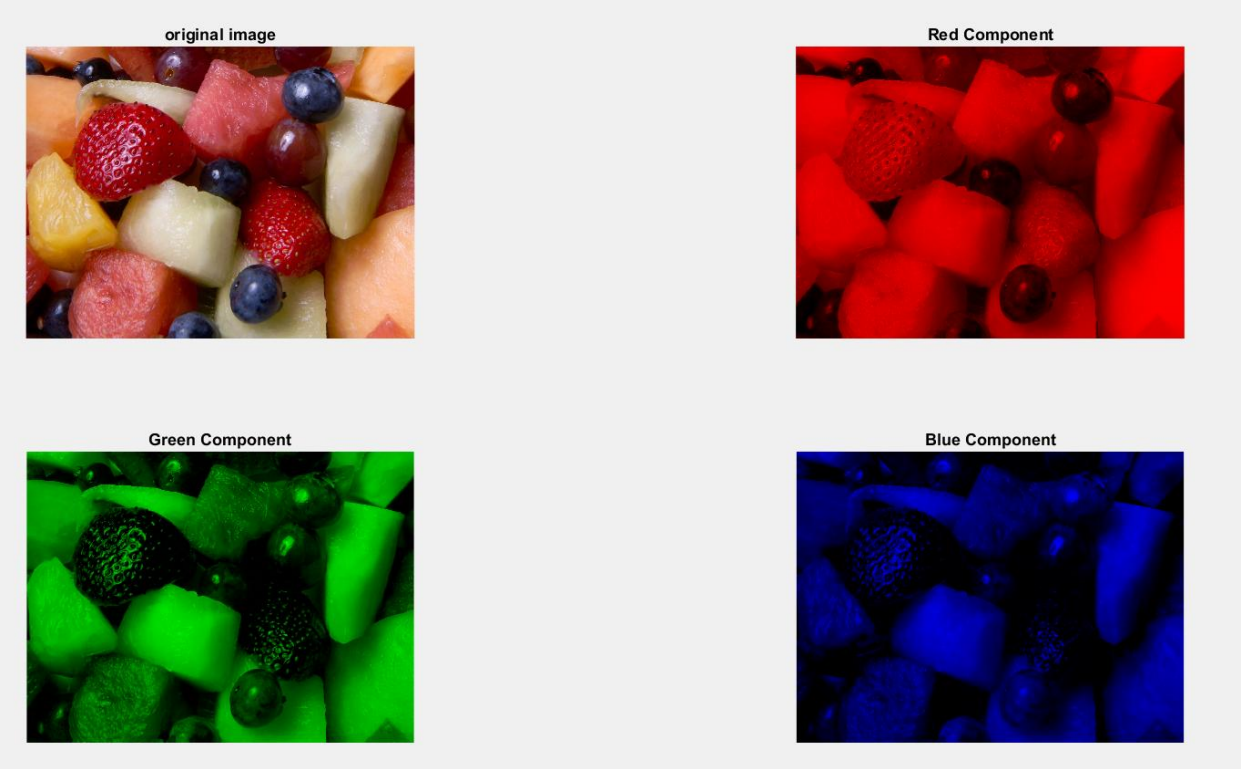
```
RGB=imread('mixedfruit.bmp');
R=RGB;
G=RGB;
B=RGB;
R(:,:,2)=0;
R(:,:,3)=0;
G(:,:,1)=0;
G(:,:,3)=0;
B(:,:,1)=0;
B(:,:,2)=0;
subplot(2,2,1),imshow(RGB),title('original image')
subplot(2,2,2),imshow(R),title('Red Component')
subplot(2,2,3),imshow(G),title('Green Component')
subplot(2,2,4),imshow(B),title('Blue Component')
```

Matlab Code

```
%This Code performs pseudo-colouring
clc;
clear all;
input_img=imread('rice.tif');
[m n]=size(input_img);
input_img=double(input_img);
for i=1:m
   for j=1:n
      if input_img(i,j)>=0 & input_img(i,j)<50
         output_img(i,j,1)=input_img(i,j,1)+50;
         output_img(i,j,2)=input_img(i,j)+100;
         output_img(i,j,3)=input_img(i,j)+10;
      end
      if input_img(i,j)>=50 & input_img(i,j)<100
         output_img(i,j,1)=input_img(i,j)+35;
         output_img(i,j,2)=input_img(i,j)+128;
         output_img(i,j,3)=input_img(i,j)+10;
      end
```

```
        if input_img(i,j)>=100 & input_img(i,j)<150
            output_img(i,j,1)=input_img(i,j)+152;
            output_img(i,j,2)=input_img(i,j)+130;
            output_img(i,j,3)=input_img(i,j)+15;
        end
        if input_img(i,j)>=150 & input_img(i,j)<200
            output_img(i,j,1)=input_img(i,j)+50;
            output_img(i,j,2)=input_img(i,j)+140;
            output_img(i,j,3)=input_img(i,j)+25;
        end
        if input_img(i,j)>=200 & input_img(i,j)<=256
            output_img(i,j,1)=input_img(i,j)+120;
            output_img(i,j,2)=input_img(i,j)+160;
            output_img(i,j,3)=input_img(i,j)+45;
        end
    end
end
subplot(2,2,1),imshow(uint8(input_img)),title('Input Image')
subplot(2,2,2),imshow(uint8(output_img)),title('Pseudo Coloured Image')
```

Results:



Conclusion: Thus we have concluded from this experiment that one can observe the quality reduction in the resultant image due to reduction of resolution and quantization and to generate phantom image.

Experiment-3

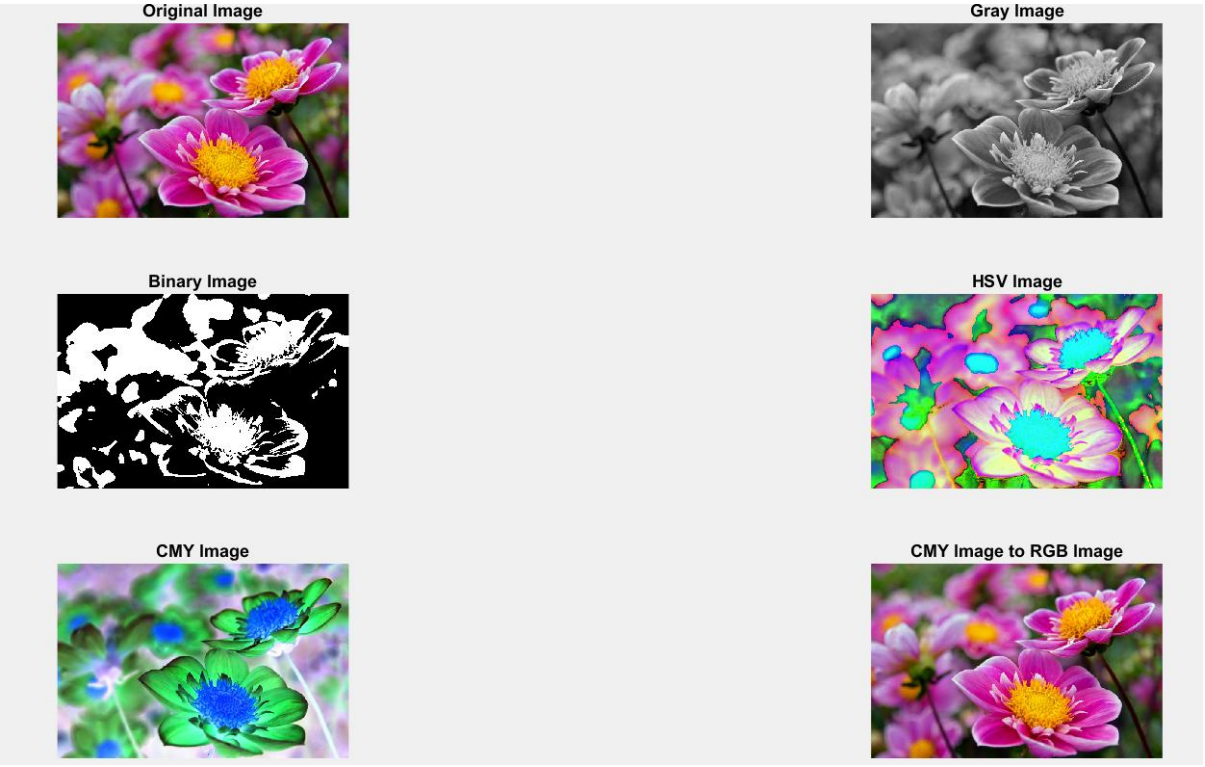
Aim: to understand and implement the following task in MATLAB

- **Different color formats**

Matlab Code:

```
I=imread('lavender.jpg');
subplot(321)
imshow(I)
title('Original Image')
J=rgb2gray(I);
subplot(322)
imshow(J)
title('Gray Image')
K=im2bw(I);
subplot(323)
imshow(K)
title('Binary Image')
L=rgb2hsv(I);
subplot(324)
imshow(L)
title('HSV Image')
Q=imcomplement(I);
subplot(325)
imshow(Q)
title('CMY Image')
W=imcomplement(Q);
subplot(326)
imshow(W)
title('CMY Image to RGB Image')
```

Results:



Conclusion: Thus we have concluded from this experiment to perform arithmetic operations as addition and complement of an image using MATLAB. A scalar value can be added to an image to increase the brightness of an image.

Experiment-4

Aim: to understand and implement the following task in MATLAB

- **Arithmetic operation on image**
- **Logical operations**
- **Understand the usefulness of logical operations**

Matlab code for addition

```
a = imread('image1.bmp');
b= imread('image2.bmp');
c = double(a)+double(b);
imshow(a), figure, imshow(b), figure, imshow(unit8(c))
```

Matlab code for Subtraction

```
a = imread('image3.bmp');
b= imread('image4.bmp');
c = double(a) - double(b);
imshow(a), figure, imshow(b), figure, imshow(unit8(c))
```

Matlab Code:

```
clc
close all

% Read Image A that is an image of an ellipse
% Convert Image A to a binary image
myimageA=imread('ellipse.jpg');
mybinaryimageA = im2bw(myimageA);


% Read Test image B
% Convert Image B to a binary image

myimageB=imread('test1.jpg');
myimageadjustB =imresize(myimageB,[256,256]);
mybinaryimageB = im2bw(myimageadjustB);


% Display the Original Image A

subplot(4,2,1)
imshow(mybinaryimageA),title('Binary Image - Image A ');

% Display the Original Image B

subplot(4,2,2)
imshow(mybinaryimageB),title('Binary Image - Image B');
```

```
% Take a complement of Image A and Display it

subplot(4,2,3)
resultor= ~mybinaryimageA ;
imshow(mat2gray(resultor)), title('Complement of Image A');

% Take a Ex-OR of Image A and Image B and Display it

subplot(4,2,4)
resultxor= xor(mybinaryimageA,mybinaryimageB);
imshow(mat2gray(resultxor)), title('Image A XOR Image B');

% Take OR of Image A and Image B and Display it
subplot(4,2,5)
resultor= mybinaryimageA | mybinaryimageB;
imshow(mat2gray(resultor)), title('Image A OR Image B ');

% Take AND of Image A and Image B and Display it
subplot(4,2,6)
resultand= mybinaryimageA & mybinaryimageB;
imshow(mat2gray(resultand)), title('Image A AND Image B ');

% Read a coour image and convert it to 8 bit grey level image

subplot(4,2,7)
mycolourimg = imread('zoo1.jpg');
mygrayimg = rgb2gray(mycolourimg);
subimage(mygrayimg),title('Original Grey Image');
axis off; axis equal;


% Extract seventh plane and display it
subplot(4,2,8)
planeimg = double(mygrayimg);
k = 128;
mysevengreyimg = mod(floor(planeimg/k),2);
imshow(mysevengreyimg),title('Seventh Plane');
```

Results:

Binary Image - Image A

Binary Image - Image B

Complement of Image A

Image A XOR Image B

Image A OR Image B

Image A AND Image B

Original Grey Image

Seventh Plane

Conclusion: Thus we have concluded from this experiment that it can be observed that and return the common area o returns all the areas of image A and Image B AND EXOR highlights the overlapped areas and also the 8 bit plane can be split to 8 planes namely 0 to 7.

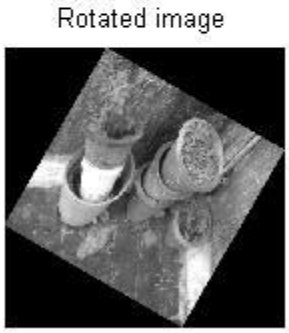
Experiment-5

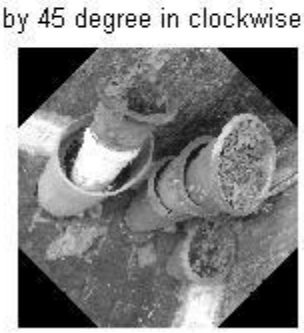
Aim: to implement the geometric operations on images using MATLAB.

Matlab Code:

```
clc
close all

% Read a colour Image and Resize it to 256 x 256
% Convert the image to grey image and display it

f = imread('pot.jpg');
g1=imresize(f,[256,256],'nearest');
g = rgb2gray(g1);
subplot(2, 2, 1);
imshow(g); title('Original Grey Image');


% Create a translation matrix for x direction - 50 units and y direction
% in 100 units. Use the command maketform to create the affine matrix
% for translation and imtransform to translate the image

t = [ 1 0 0; 0 1 0; 50 100 1];
tform = maketform('affine',t);
translateimg = imtransform(g,tform);
subplot(2, 2, 2);
imshow(translateimg),title('Translated Image');


% Create a rotation matrix for pi/6
% Use the command imtransform to rotate the image using the created
% rotation matrix

subplot(2, 2, 3);
t = pi/6;
r = [cos(t) sin(t) 0; -sin(t) cos(t) 0; 0 0 1];
tform = maketform('affine',r);
rotimg = imtransform(g,tform);
imshow(rotimg),title('Rotated image');

% Use the imrotate command to rotate the image in clockwise direction
% 45 degrees - Use negative values for angle if the rotation required is
% clockwise direction as the default rotation is in counter clockwise
% direction. Additional parameter 'crop' is given to ensure that the
% output image is same as the input image

subplot(2, 2, 4);
h = imrotate(g,-45,'crop');
imshow(h);title('Rotated by 45 degree in clockwise direction');
```

Results:





Conclusion: Thus we have concluded from this experiment that it can be observed that affine matrix is created first and then imtransform is used to translate the image. This logic can be extended for performing other affine transforms also and a negative angle is give to perform the rotation in clock wise direction and the additional parameter is used to ensure the output image is same as the input image.

Experiment-6

Aim: to understand and implement the following task in MATLAB

- **Histogram operations for Gray Image**
- **Histogram operation for color Image**

Matlab code for histogram for Gray Image

```
clc
clear all
close all
a=imread('babyincradle.png');
%perform histogram equalization
b=histeq(a);
subplot(2,2,1),imshow(a),title('original image'),
subplot(2,2,2),imshow(b),title('After histogram equalization'),
subplot(2,2,3),imhist(a),title('original histogram')
subplot(2,2,4),imhist(b),title('After histogram equalization')
```

Matlab Code for Color Image

```
clc
clear all
close all

a=imread('coconut.bmp');
%Conversion of RGB to YIQ format
b=rgb2ntsc(a);
%Histogram equalization of Y component alone
b(:,:,1)=histeq(b(:,:,1));
%Conversion of YIQ to RGB format
c=ntsc2rgb(b);
imshow(a),title('original image')
figure, imshow(c),title('Histogram equalized image')
```

Result

Experiment-7

AIM: Write code for computes the edges in the image and Boundary detection

```
% This program computes the edges in the image
clear all;
close all;
clc;
a=imread('deer1.jpg');
a=rgb2gray(a);
b=edge(a,'roberts');
c=edge(a,'sobel');
d=edge(a,'prewitt');
e=edge(a,'log');
f=edge(a,'canny');
imshow(a),title('Original Image')
figure,imshow(b),title('Roberts')
figure,imshow(c),title('Sobel')
figure,imshow(d),title('Prewitt')
figure,imshow(e),title('Log')
figure,imshow(f),title('Canny')
```

```
% This program boundary detection
close all;
clear all;
clc;
a=imread('morph2.bmp');
b=[0 1 0;1 1 1;0 1 0];
a1=imdilate(a,b);
a2=imerode(a,b);
a3=a-a2;
a4=a1-a;
a5=a1-a2;
imshow(a),title('Original image')
figure,imshow(a1),title('Dilated image')
figure,imshow(a2),title('Eroded image')
figure,imshow(a3),title('First property ')
figure,imshow(a4),title('Second property')
figure,imshow(a5),title('Third Property')
```

‘

Experiment-8

Aim: to implement and observe various types of noise in images using MATLAB.

Matlab Code:

```
a=imread('rose.jpg');
% Addition of noise to the input image
b=imnoise(a,'salt & pepper');
c=imnoise(a,'gaussian');
d=imnoise(a,'speckle');
% Defining 3x3 and 5x5 kernel
h1=1/9*ones(3,3);
h2=1/25*ones(5,5);
% Attempt to recover the image
b1=conv2(b,h1,'same');
b2=conv2(b,h2,'same');
c1=conv2(c,h1,'same');
c2=conv2(c,h2,'same');
d1=conv2(d,h1,'same');
d2=conv2(d,h2,'same');
%Displaying the result
figure,subplot(2,2,1),imshow(a),title('Original Image'),
subplot(2,2,2),imshow(b),title('Salt & Pepper noise'),
subplot(2,2,3),imshow(uint8(b1)),title('3 x 3 Averaging filter'),
subplot(2,2,4),imshow(uint8(b2)),title('5 x 5 Averaging filter')
%.....................
figure,subplot(2,2,1),imshow(a),title('Original Image'),
subplot(2,2,2),imshow(c),title('Gaussian noise'),
subplot(2,2,3),imshow(uint8(c1)),title('3 x 3 Averaging filter'),
subplot(2,2,4),imshow(uint8(c2)),title('5 x 5 Averaging filter'),
%.................
figure,subplot(2,2,1),imshow(a),title('Original Image'),
subplot(2,2,2),imshow(d),title('Speckle noise'),
subplot(2,2,3),imshow(uint8(d1)),title('3 x 3 Averaging filter'),
subplot(2,2,4),imshow(uint8(d2)),title('5 x 5 Averaging filter'),
```

Experiment 9

AIM: Perform restoration operation on image by inverse filtering

```
%This program performs inverse filtering
close all;
clear all;
clc;
x =imread(‘flower2.jpg');
x=double(rgb2gray(x));
[M N]=size(x);
h = ones(11,11)/121;
sigma = sqrt(4*10^(-7));
freqx = fft2(x);% Fourier transform of input image
freqh = fft2(h,M,N);%Fourier transform of degradation
y = real(ifft2(freqh.*freqx));
freqy = fft2(y);
powfreqx = freqx.^2/(M*N);
alpha = 0.5;%Indicates inverse filter
freqg = ((freqh.')').*abs(powfreqx) ...
    ./(abs(freqh.^2).*abs(powfreqx)+alpha*sigma^2);
Resfreqx = freqg.*freqy;
Resa = real(ifft2(Resfreqx));
imshow(uint8(x)),title('Original Image')
figure,imshow(uint8(y)),title('Degraded Image')
figure,imshow(uint8(Resa)),title('Restored Image')
```

Experiment-10

Aim: to understand and implement Image segmentation in MATLAB using the following threshold types

- **Simple thresholding**
- **Multiple thresholding**
- **Adaptive thresholding**
- **Optimal thresholding**

Matlab Code:

```
clc;
close all;
clear all;

a = imread('grayflower256.jpg');
a = rgb2gray(a);
subplot(3,3,1);
imshow(a); title('Original Image');

% Simple thresholding at 0.3
% This is equivalent to saying threshoding at pixel value
% 0.3 x 255 = 76.5 , approximatesly 77
% imlabel() is a Matlab command that can threshold any matrix


level = 0.3;
%% Display the threshold image
subplot(3,3,2);
segimage1 = im2bw(a,level);
imshow(segimage1); title('Simple Thresholding at 0.3');


% Simple thresholding at 0.6
% So threshold value is 0.6 x 255 = 153
% Single Thresholding can be done like this also


%% Display the threshold image
subplot(3,3,3);
imshow(a > 153); title('Simple Thresholding at 0.6');


% Multiple thresholding Algorithm
% Let us assume that the output should be zero if pixel value is
% if <= 0.1 x 255 = 25.5 = 26, pixel output 204 if pixel value is <= 0.9 x
% 255 = 230 and 0 if pixel value is above 230.
```

```
%% Create a temporary matrix g

tmp = a;

[m n]= find(a<26);
for j = 1: length(m)
   tmp(m(j),n(j))=0;
end

[m n]= find(a>26 & a <= 230);
for j = 1: length(m)
   tmp(m(j),n(j))=0.8;
end

[m n]= find(a>230);
for j = 1: length(m)
   tmp(m(j),n(j))=0;
end
subplot(3,3,4);
segimage2 = im2bw(tmp,0);
imshow(segimage2); title('Multiple threshoding(Between 27-230)');

%% Find the threshold Value using Otsu Algorithm

level = graythresh(a);

%% Display the threshold image

subplot(3,3,5);
segimage = im2bw(a,level);
imshow(segimage); title('Otsu - Optimal Segmented Image');

%% Display Blured Image

b = imread('bluredtxt.jpg');
subplot(3,3,6);
imshow(b); title('Badly illuminated Image');

level = graythresh(b);
subplot(3,3,7);
segimage = im2bw(b,level);
imshow(segimage); title('Otsu - Segmentation for bad illuminated Image');


b = imread('bluredtxt.jpg');
b = rgb2gray(b);

%Create an average Image
```

```
avgfilt = ones(13,13);
adaptfiltmask = avgfilt/sum(avgfilt);
im = imfilter(b,adaptfiltmask,'replicate');

%Create an median image
im1 = medfilt2(b,[20 20]);

%Adaptive threshold algorithm use
% threshold = mean + constant (Here 18)

thresh = im+18;
adaptthreshimg = b - thresh;
subplot(3,3,8);
imshow(adaptthreshimg > 0);


%Adaptive threshold algorithm used threshold = mean + constant (Here 2)
thresh1 = im1 + 2;
adaptthreshimg = b - thresh1;
subplot(3,3,9);
imshow(adaptthreshimg > 0);
```