Homework 1

Meta information	
Name	Savan Kiran
Program	Masters in Computer Science
Questions skipped	N/A
Questions substituted	N/A
Extra credit questions	N/A

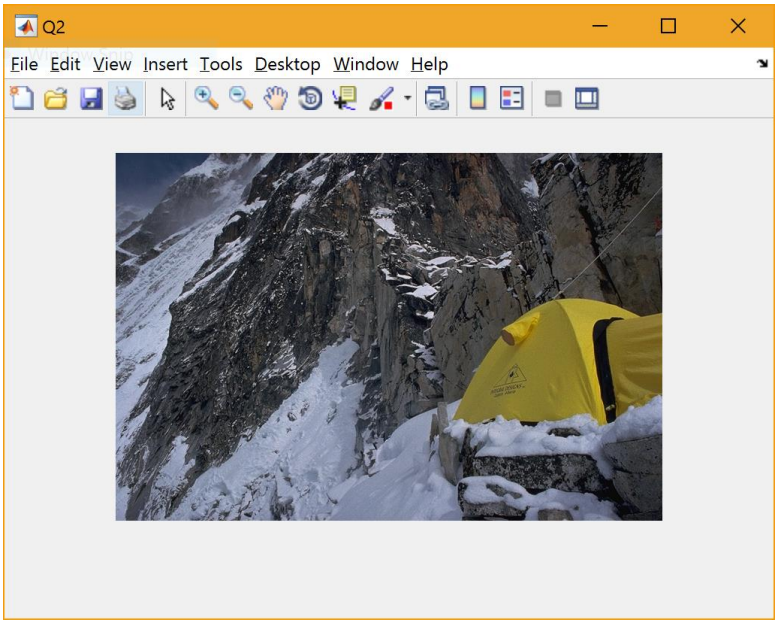
1. Become familiar with Matlab (No deliverable expected)

Went through the documentation for Matlab on http://www.mathworks.com/access/helpdesk/help/helpdesk.shtml.

2. Reading and Displaying Images (No deliverable expected)

Read the 'tent.jpg' image provided using `imread` and displayed it in a `figure` using `imshow`. Below is the code snippet and a screenshot of the output.

```
%Q2
%   Read image data and display using 'imread' and 'imshow'.
imdata = imread(infile);
figure('Name','Q2','NumberTitle','off');
imshow(imdata);
```



3. Writing Images (No specific deliverable expected)

Wrote the image data that was read in the previous step into an output file, 'out.jpg'. Below is the code snippet.

```
%Q3
%   Write out image using 'imwrite'
imwrite(imdata, 'out.jpg', 'JPEG');
```

4. Scripts, Files and Paths (No specific deliverable expected)

All the code related to hw1 can be found in the submitted `hw1.m` script file.

5. Functions (No specific deliverable expected)

The submitted script has function hw1 with the following format as was mentioned.

```
function [num_questions] = hw1(infile)
```

6. Documenting functions

The `hw1` function is well documented and below is the help snippet.

```
>> help hw1

hw1 Homework 1

    hw1(infile) takes an image file name from current directory as input.

    It does the following operations:

        Q2: Read image data and display image

        Q3: Write out image to 'out.jpg'

        Q7:

            1. Display the information about the variable containing image
data.

            2. Display the size of the image

            3. Display range of values for Red, Green, Blue and Overall

            4. Convert the image to grayscale using 'rgb2gray', display the

            image and print the new image data information

        Q8:

            1. Create and display grayscale image based on

                1. red channel

                2. green channel

                3. blue channel

            2. Circular shift color values to the left and display resultant
image

        Q9:

            1. Convert grayscale into double precision type.
```

2. Display it using 'imagesc'

a. show colorbar for it

b. change default colormap(i.e., JET) to gray

c. rectify axis to have square pixels rather than distorted pixels

3. Use 'imshow' to display image

Q10:

Set every 5th pixel, horizontally and vertically to 1 i.e., white and

display result using 'imagesc' and 'imshow'

Q11:

Show histogram distribution for each of the color channels (R,G,B)

Q12:

Plot sin and cos curves over domain [-pi:pi] using 'linspace' and 'plot'

Q13:

Solve linear equations using -

1. 'inv'

2. 'linsolve'

Q16:

1. Set every 5th pixel horizontally and vertically to 0 without using

explicit loops i.e., in one simple command and show resulting image.

2. Set every pixel with value >0.5 to black and show resulting image.

Q17:

1. Import data from file and plot

2. Find covariance matrix

3. "Mean-center" the plot to

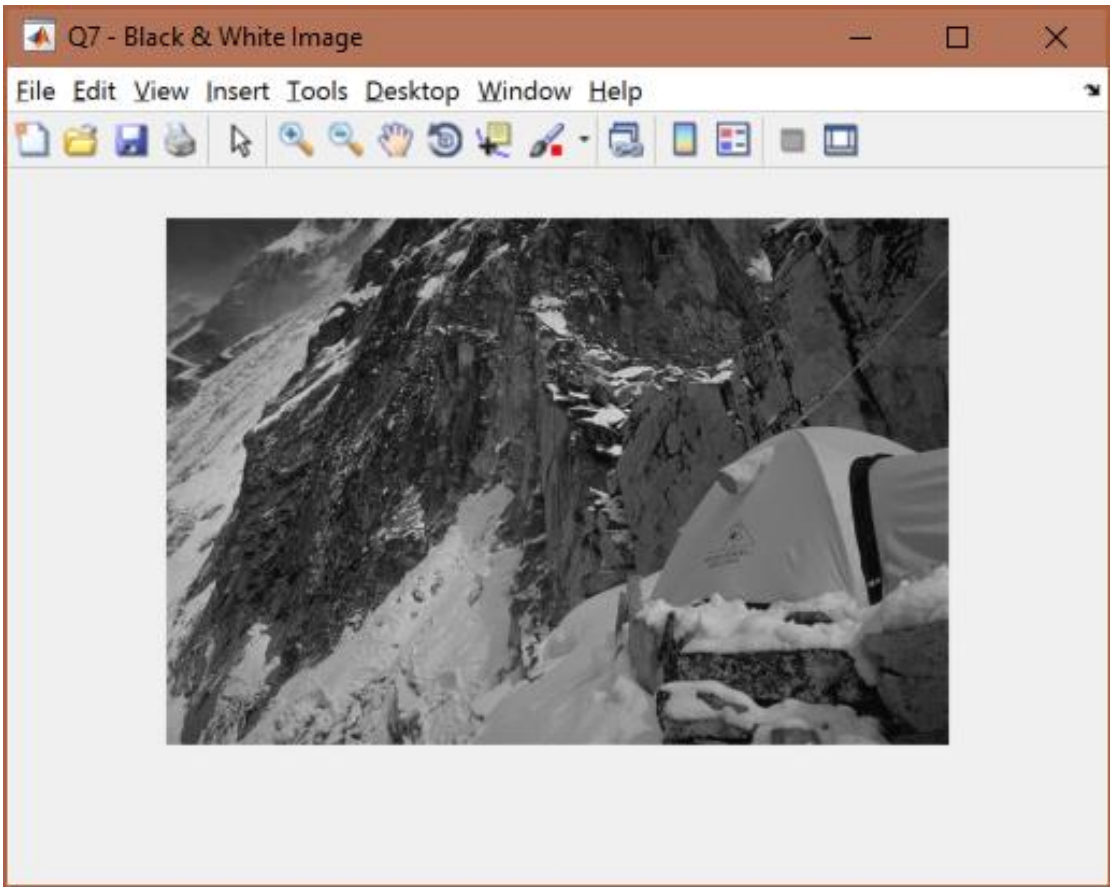
Q18:

1. Rotate the data and display

2. Compare variance before and after transformation

7. Basic image data structures

- Variable information: `whos` function shows information (type, size, bytes) on all the variables. I used `whos <variable_name>` to show information of only the image variable.
- Size: We can use `size` to get the size of each dimension of the multidimensional array. It returns all the sizes which have to be saved into variables `num_rows`, `num_cols` and `num_channels` as depicted below.
- Range: (Find code snippet below for both)
    - To get range, i.e., [min, max] for red, blue and green channels individually, I filtered the image data to contain only values for either red/green/blue channel. Later flattened it to a 1D array and feed to `min` & `max` functions respectively to find the range for each channel.
    - To get the overall range, flatten all the channels (red, green & blue) into a 1D array and find min and max using respective functions.
- Gray scale: I used the `rgb2gray` to convert the image to gray scale image. The resulting data structure is a 2D array with the values representing shades of gray in range [1,255]. The resulting image is displayed in a new figure.



```
%Q7
%   1. Display the information about the variable containing image
data.
%   2. Display the size of the image
%   3. Display range of values for Red, Green, Blue and Overall
%   4. Convert the image to grayscale using 'rgb2gray', display the
%   image and print the new image data information
whos imdata;

[num_rows, num_cols, num_channels]=size(imdata);
disp(['Size: ' num2str(num_rows) 'x' num2str(num_cols) 'x'
num2str(num_channels)]);

red=imdata(:,:,1);
red_linear=red(:);
disp(['Red range [' num2str(min(red_linear)) ','
num2str(max(red_linear)) ']']);
green=imdata(:,:,2);
green_linear=green(:);
disp(['Green range [' num2str(min(green_linear)) ','
num2str(max(green_linear)) ']']);
blue=imdata(:,:,3);
blue_linear=blue(:);
disp(['Blue range [' num2str(min(blue_linear)) ','
num2str(max(blue_linear)) ']']);
overall=imdata(:);
disp(['Overall range [' num2str(min(overall)) ','
num2str(max(overall)) ']']);

imdata_gray=rgb2gray(imdata);
whos imdata_gray;
figure('Name','Q7 - Black & White Image','NumberTitle','off');
imshow(imdata_gray);
```

8. Image channels

In the last step, we extracted the 3 channels separately into 3 2D arrays, red, green and blue as mentioned in the code snippet. I used these 3 2D arrays to display their respective grayscale images. Below is a screenshot of all these images side by side with their natural grayscale and color counterparts.

Are they what you expect?

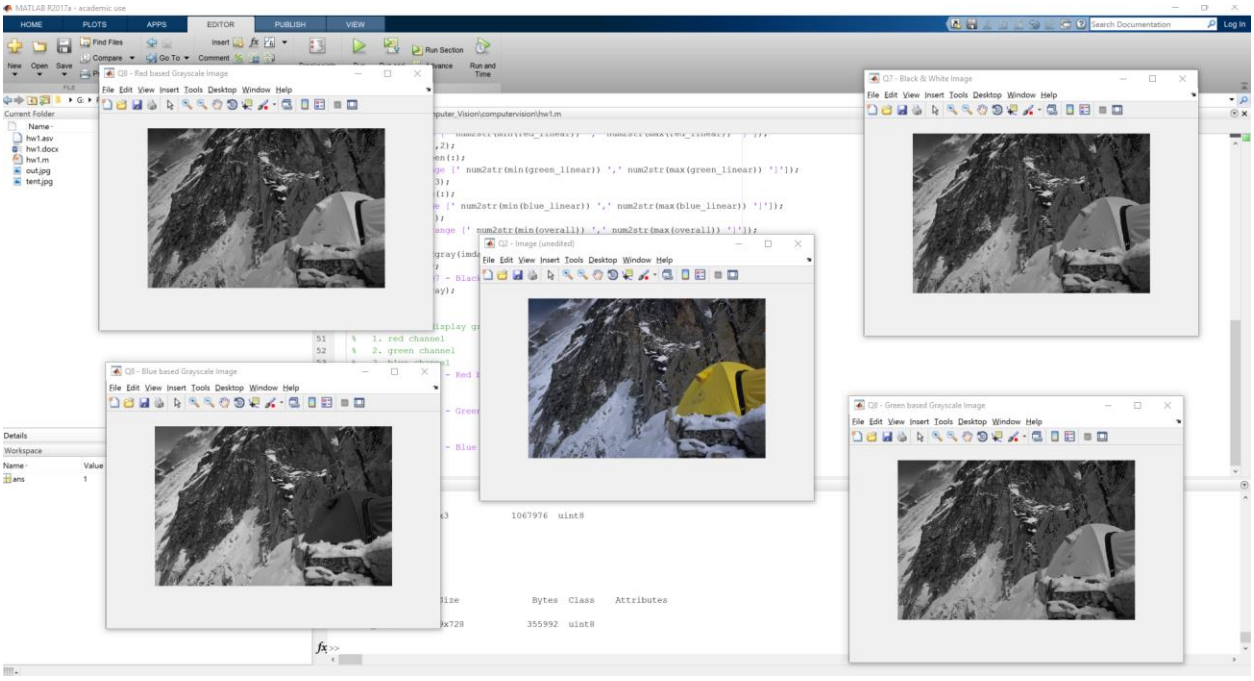
To some extent yes. I anticipated them to be very different from the natural grayscale, however, the difference is very subtle. Below are some of the observations.

Tent color w.r.t natural grayscale

- Tent color is brighter in red based grayscale
- Tent color is almost indistinguishable in green based grayscale
- Tent color is much darker in blue based grayscale

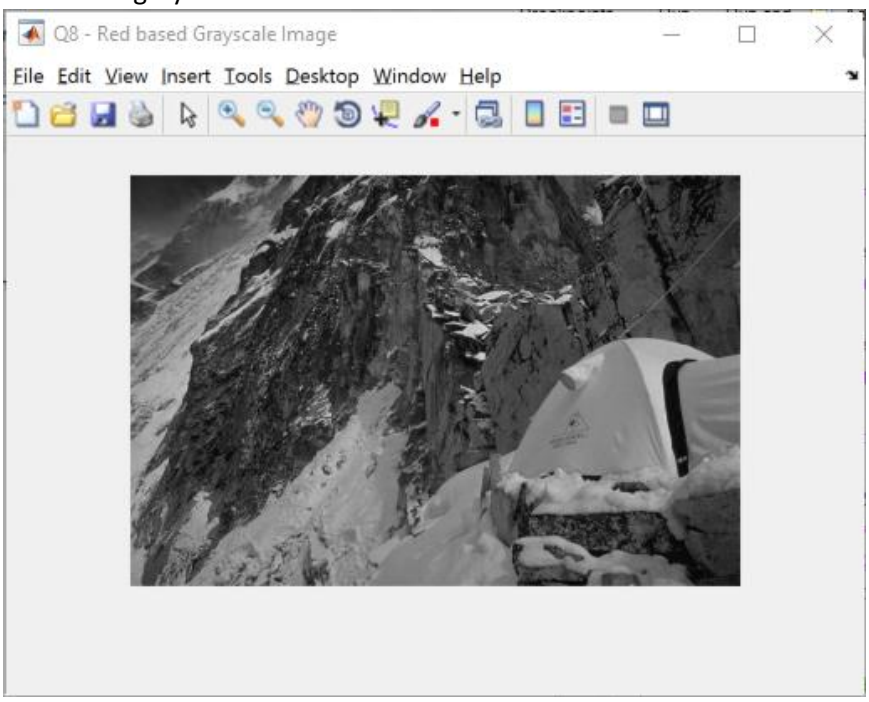
The tent color is very dark in the blue based and brighter in red based grayscale. I think this is because yellow is made up of blue + green. And there is a constant large amount of blue and the amount of green determines the shade of yellow. In this case, we have a moderate amount of green which drives the yellow. In absence of green in blue based grayscale, the tent appears very dark. On the other hand, red has very little role to play in yellow because of which, tent appears brighter in red based grayscale as absence of red takes it closer to white in grayscale.

Also, the snow appears a little brighter in blue based grayscale which can be explained similar as above.

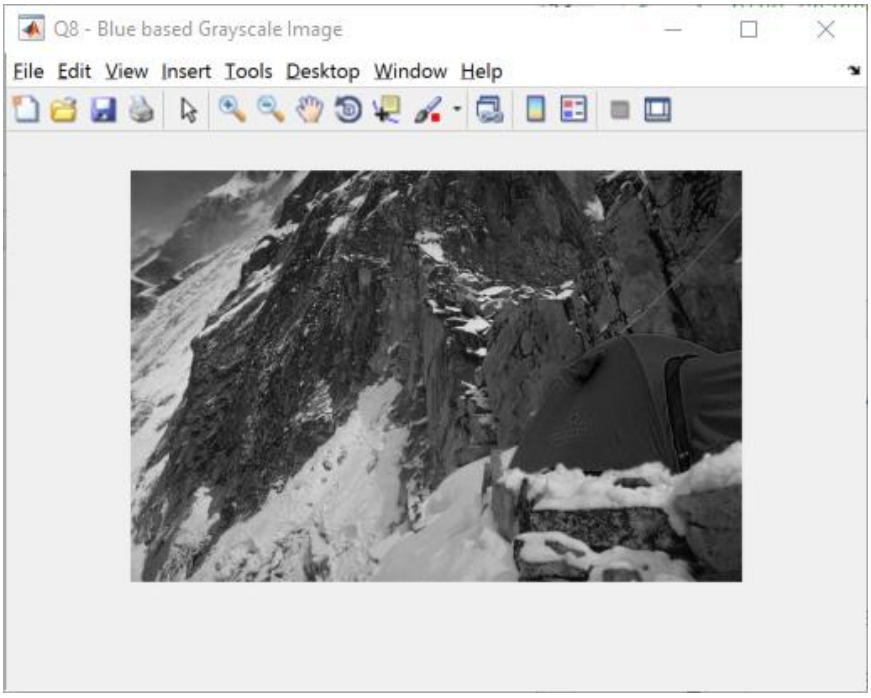


```
%Q8
%   Create and display grayscale image based on
%   1. red channel
%   2. green channel
%   3. blue channel
figure('Name','Q8 - Red based Grayscale Image','NumberTitle','off');
imshow(red);

figure('Name','Q8 - Green based Grayscale Image','NumberTitle','off');
imshow(green);

figure('Name','Q8 - Blue based Grayscale Image','NumberTitle','off');
imshow(blue);
```

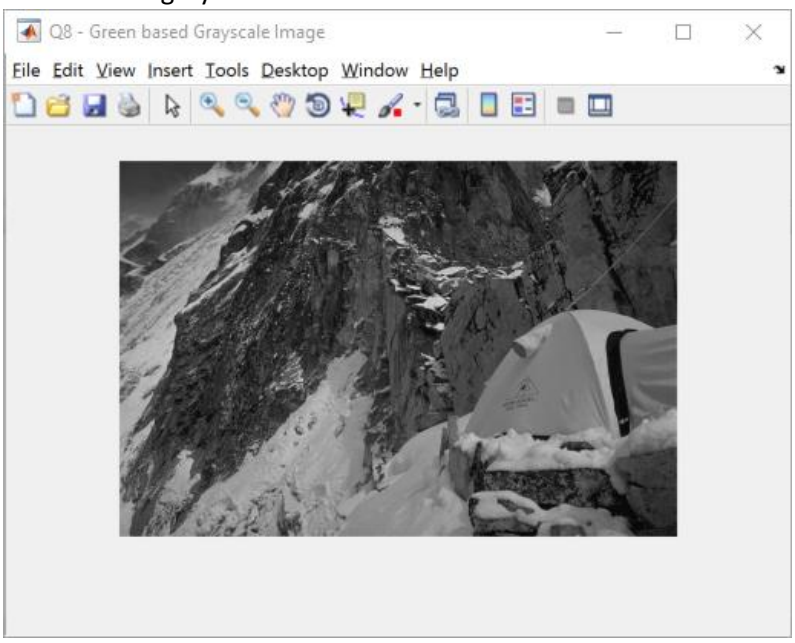
Red based grayscale:



Blue based grayscale:



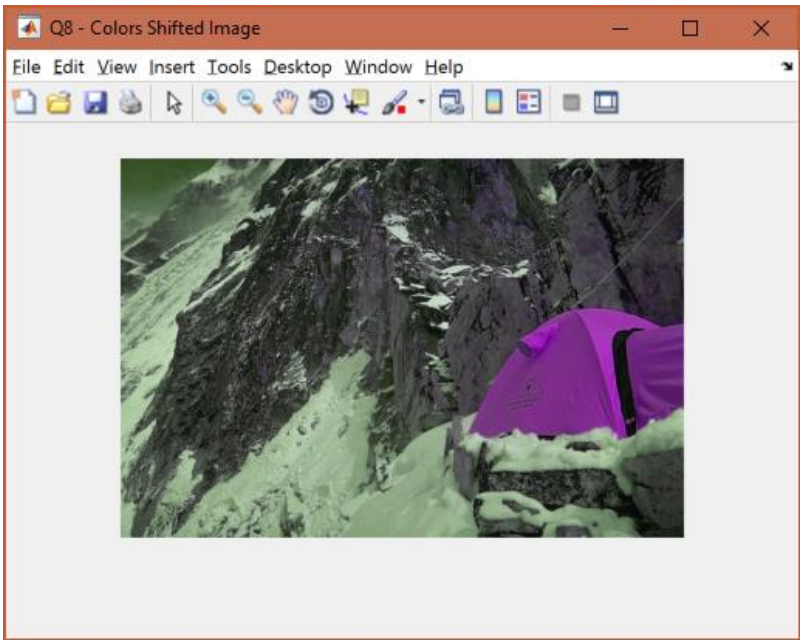
Green based grayscale:



Color (left) shifted image: In the 3D image data structure, I (circular) shifted the channels to the left such that following transformations: {green->red, blue->green, red->blue} takes place. I came across `circshift` function that helps me do it one call when I was reading through the matlab documentation. Below is the code snippet for the same and a screenshot of the output.

Yellow is formed from blue + green. In the new color scheme, blue is actually red and green is actually blue, therefore, the tent is no more yellow, but pink/violet.

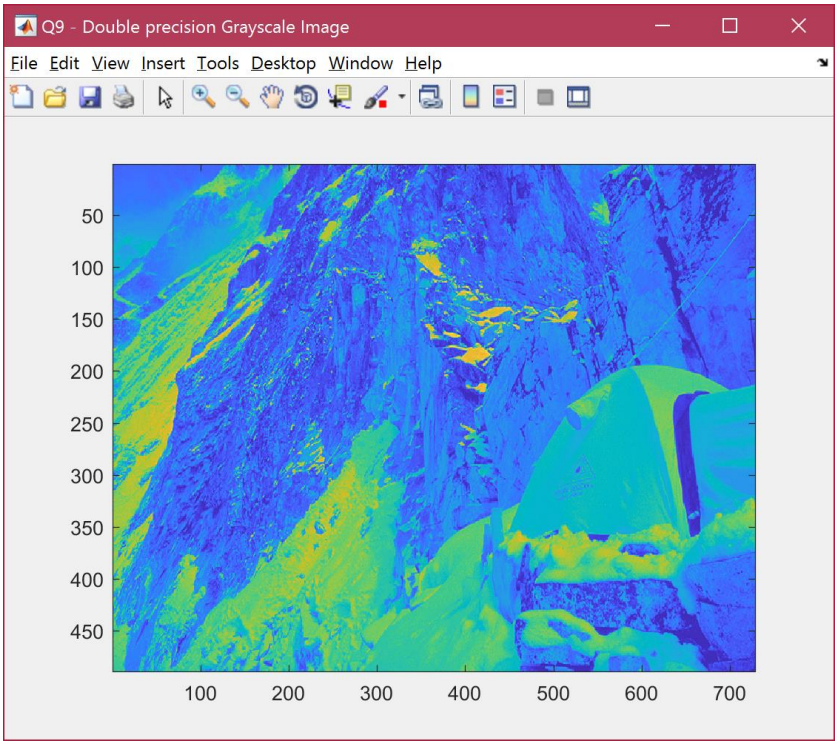
```
new_color_img=circshift(imdata,[0 0 -1]);
figure('Name','Q8 - Colors Shifted Image','NumberTitle','off');
imshow(new_color_img);
```



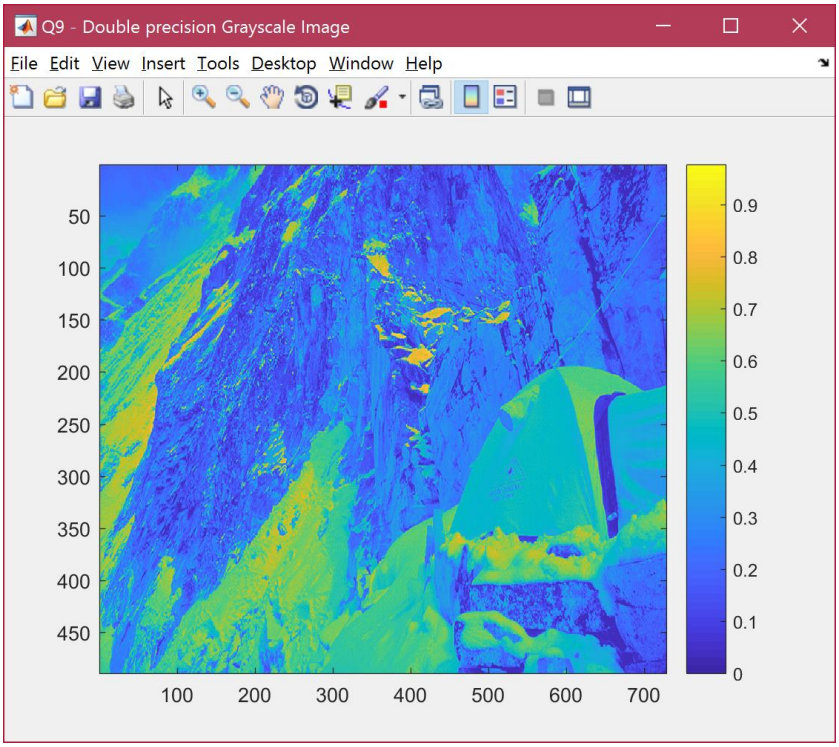
9. Visualizing Matrices

I changed the precision of values in `imdata` to double and did an element-wise division by 255(upper threshold) to get values in range [0,1]. Displayed the resultant data using `imagesc`. Below is a screenshot.

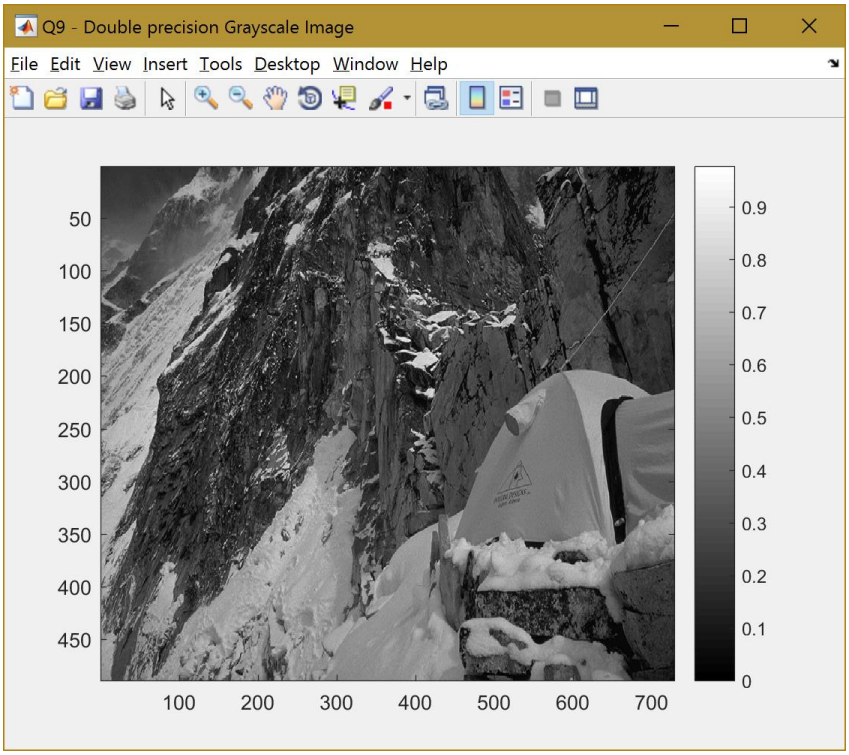
```
%Q9
%   1. Convert grayscale into double precision type.
%   2. Display it using 'imagesc'
%       a. show colorbar for it
%       b. change default colormap(i.e., JET) to gray
%       c. rectify axis to have square pixels rather than distorted
pixels
%   3. Use 'imshow' to display image
colormap('JET');
imdata_double=double(imdata);
imdata_double=imdata_double(:,:,:)./255;
imdata_double=rgb2gray(imdata_double);
figure('Name','Q9 - Double precision Grayscale
Image','NumberTitle','off');
imagesc(imdata_double);
colorbar;
colormap(gray);
axis image;
```



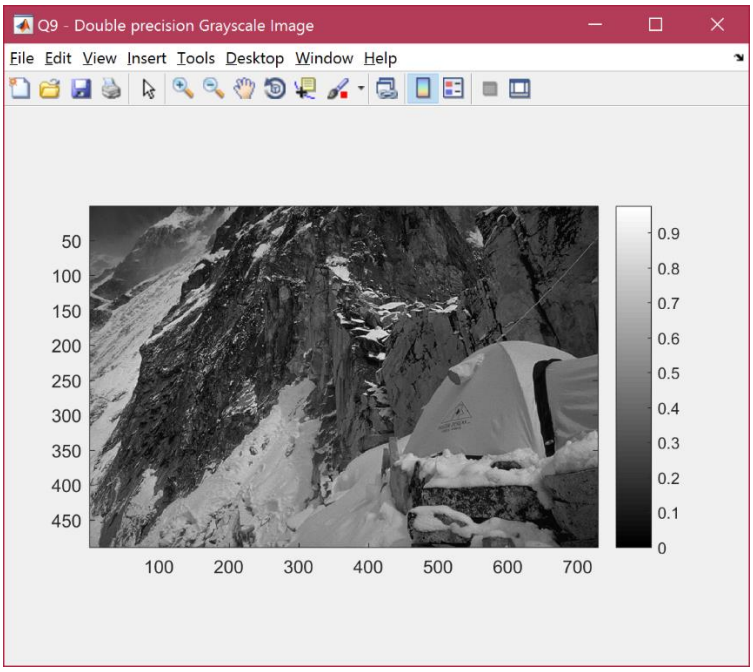
To enable colorbar, I hit `colorbar`. Below is the screenshot. It shows the how the values are mapped to color as described in the hw.



The default colormap was `JET`. Changed it to gray using `colormap` command as shown in the code snippet below.



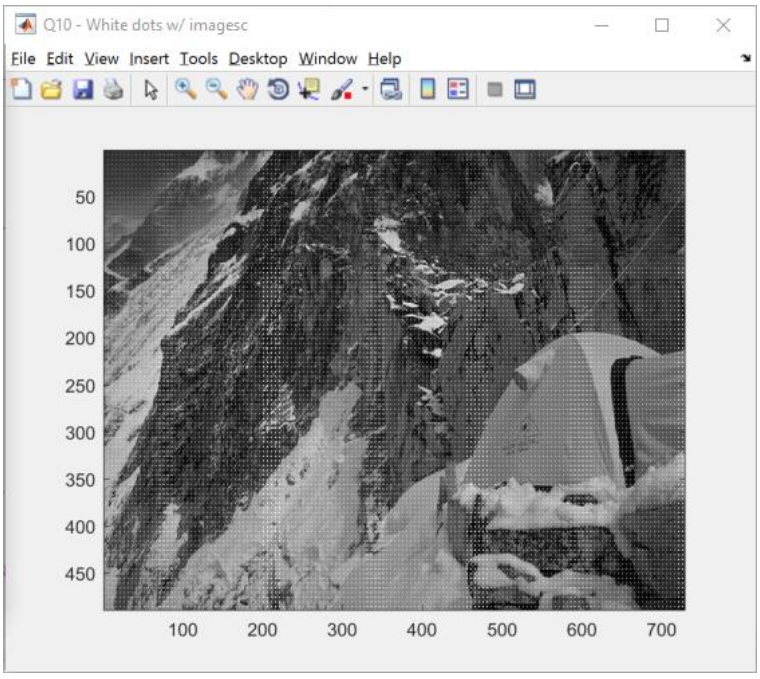
All the images above seem distorted because the original aspect ratio of the image is rectangle where as it is displayed in a square area. As a result, the pixels are distorted and not square anymore. We can use `axis` command with `image` specification to fix this so that plot box fits tightly around the data.



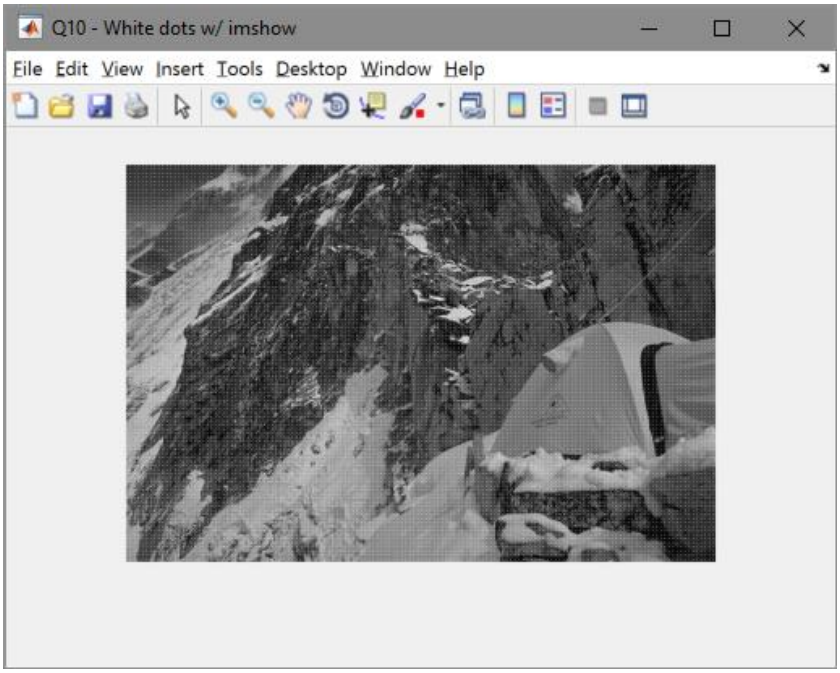
10. Manipulating Matrices

As indicated in the hw, I used nested for loops to iterate over 5pixel interval horizontally and vertically and set it to 1 i.e., white. Used this modified image data to render using `imagesc` and `imshow`.

Below is the screenshot of using `imagesc`.



Below is the screenshot of using `imshow`.



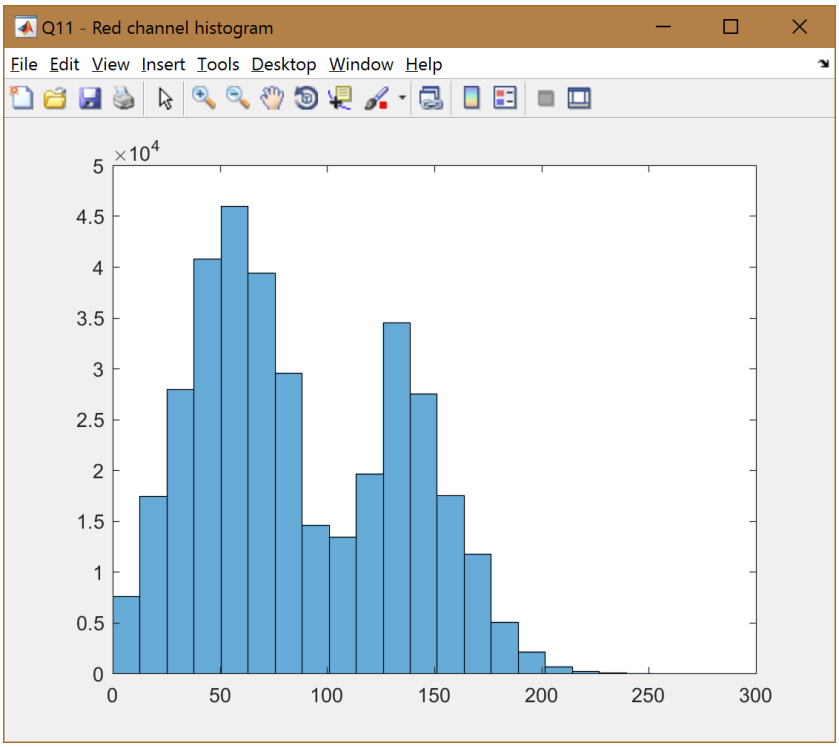
Because the axis is not set according to actual aspect ratio of the image, the white dots are not uniform in the image displayed using `imagesc`. Whereas, it is uniform as expected, in the image displayed using `imshow`. When I fixed the axis in `imagesc`, the two outputs were identical. If it is just rendering the image, `imshow` is the simplest and best option as we don't need to tweak any configurations. If we need to visualize data, `imagesc` is a better option.

```
%Q10
%   Set every 5th pixel, horizontally and vertically to 1 i.e., white
and
%   display result using 'imagesc' and 'imshow'
for R=1:5:num_rows
    for C=1:5:num_cols
        imdata_double(R,C)=1;
    end
end
axis auto;
figure('Name','Q10 - White dots w/ imagesc','NumberTitle','off');
imagesc(imdata_double);
colormap(gray);
figure('Name','Q10 - White dots w/ imshow','NumberTitle','off');
imshow(imdata_double);
```

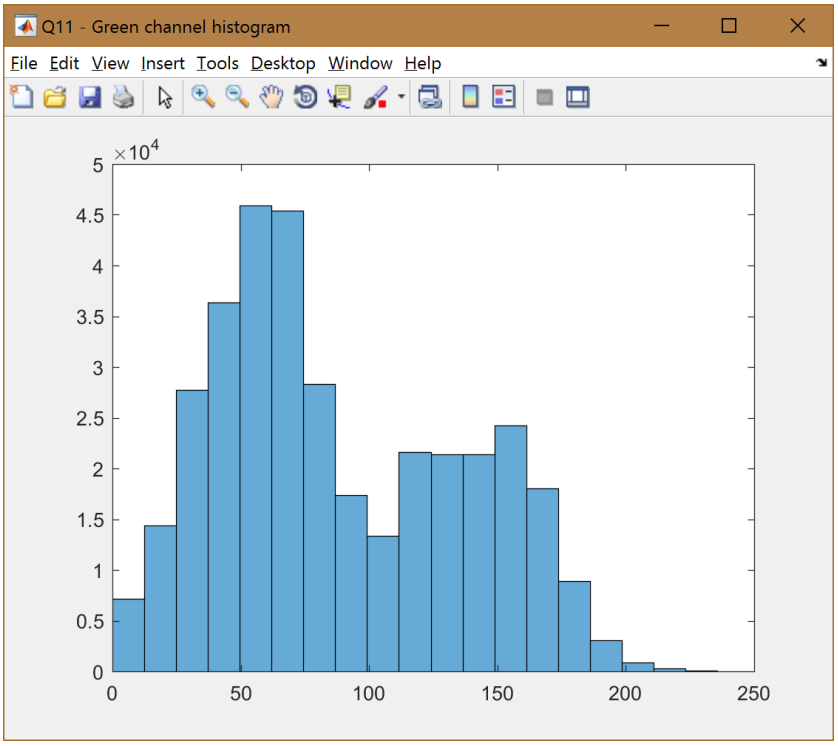
11. Histograms

I used the 1D array I created in Q7 to calculate an overview of the distribution of each channel (i.e., Red, Blue and Green) in terms of histograms. The function `histogram` calculates the distribution and plots it onto a figure. Below are the images of the 3 channel's distributions.

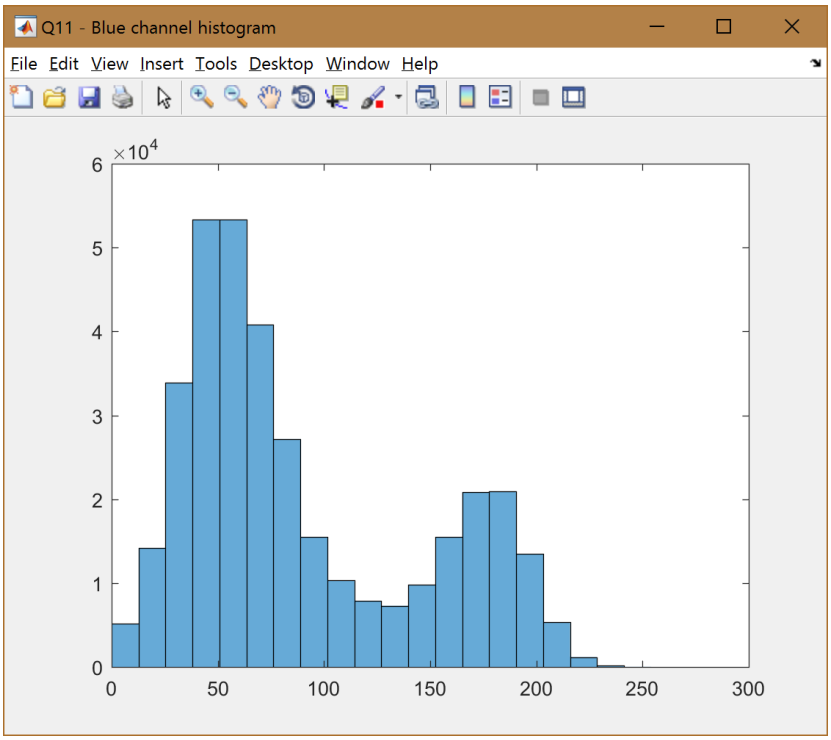
Red channel distribution:



Green channel distribution:



Blue channel distribution:



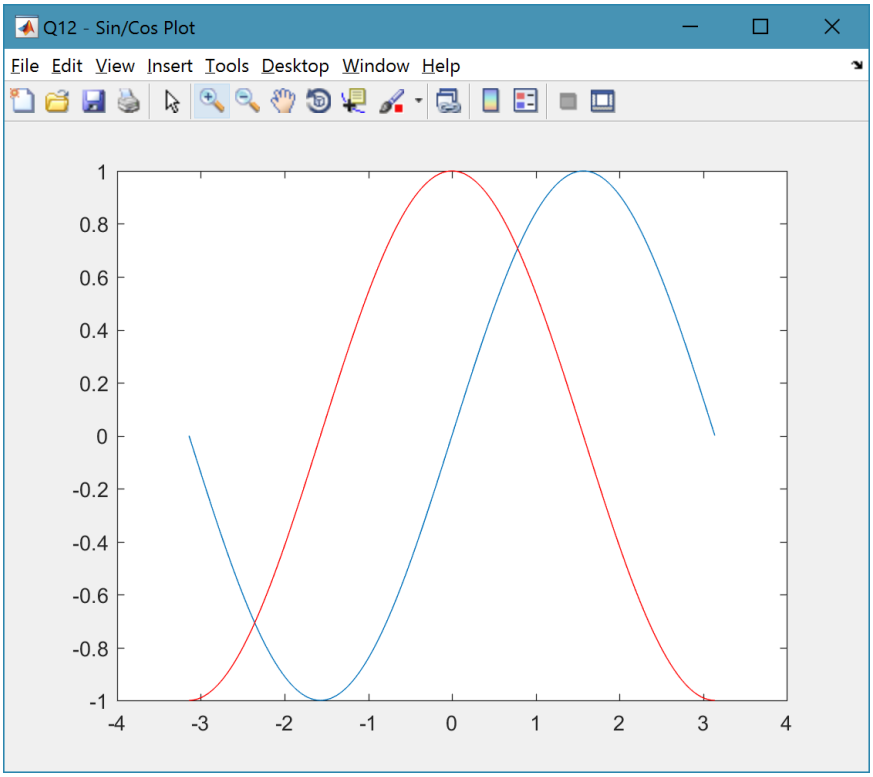
Below is the code snippet:

```
%Q11
%   Show histogram distribution for each of the color channels (R,G,B)
figure('Name','Q11 - Red channel histogram','NumberTitle','off');
histogram(red, 20);
figure('Name','Q11 - Green channel histogram','NumberTitle','off');
histogram(green, 20);
figure('Name','Q11 - Blue channel histogram','NumberTitle','off');
histogram(blue, 20);
```

The histogram only gives the distribution of values in the specified range. It loses a lot of information in the process like spatial information of these values and exact value distribution as there are only 20 buckets. Also loses the context of the color, ex: Blue has a large count in and around 50. A lot of pixels have a dark blue component in them, however, what is the red and green component for these pixels? This information is lost with the histogram.

12. Plotting

As advised, I used the `linspace` command to create 1D domain vector from `-pi:pi`. Then used `plot` command with `sin` to plot sine curve. The default color was blue. Then I used `hold on` to ensure that next curve gets plotted on same graph. In the next `plot` command, I used `cos` function and passed `'r'` to indicate the color of the curve drawn.




```
%Q12
%   Plot sin and cos curves over domain [-pi:pi] using 'linspace' and
'plot'
figure('Name','Q12 - Sin/Cos Plot','NumberTitle','off');
x=linspace(-pi,pi,300);
plot(x,sin(x));
hold on;
plot(x,cos(x),'r');
```

13. Playing with Linear Algebra

Used `inv` and `linsolve` functions to solve the below linear equation:

$$3 * x + 4 * y + z = 9$$

$$2 * x - y + 2 * z = 8$$

$$x + y - z = 0$$

Proof that we can solve linear equations using inverse function,

$$AX = B$$

$$where$$

$$A = \begin{matrix} 3 & 4 & 1 \\ 2 & -1 & 2 \\ 1 & 1 & -1 \end{matrix}$$

$$B = \begin{matrix} 9 \\ 8 \\ 0 \end{matrix}$$

$$X = \begin{matrix} x \\ y \\ z \end{matrix}$$

$$Multiplying\ both\ sides\ with\ A^{-1}$$

$$A^{-1}AX = A^{-1}B$$
$$IX = A^{-1}B$$

$$where\ I\ is\ the\ identity\ matrix\ and\ IX = X$$

$$X = A^{-1}B$$

From the above equation, we can solve for `X`. Doing so in matlab gives the solution as below,

```
Linear eqn solved using inverse =

    1.8235

    0.7059

    2.5294
```

Below is the code snippet for the same,

```
A=[2 4 1; 2 -1 2; 1 1 -1];
B=[9; 8; 0];
X1=inv(A)*B;
```

As was mentioned in the hw, we could also use `linsolve` function with "`\`" operator shorthand to solve the linear equations. Below is the code snippet for it.

```
X2=A \ B;
```

Output:

```
Linear eqn solved using linsolve =

    1.8235

    0.7059

    2.5294
```

When the result is printed out, both approaches seem to generate same solution. However, when we subtract the 2 results, we get to know that they differ by a very small fraction which would otherwise be easily ignored. Below is the difference between the results,

```
Difference in above results =

   1.0e-15 *

    0.2220

         0

         0
```

The difference in the results is probably because of the rounding up errors of floating point numbers and the fact that both results are generating by different methods. With `linsolve`, it uses LU factorization to quickly solve without doing the cumbersome approach and my guess is we lose some precision in the process unlike the inverse approach.

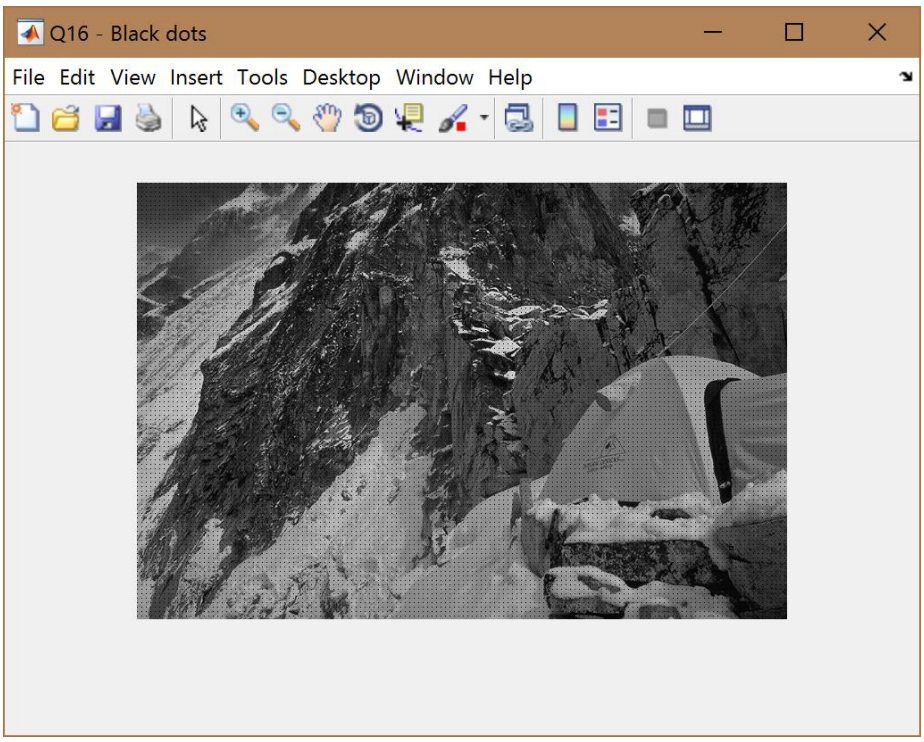
14. Playing with Linear Algebra II (No specific deliverable)

15. Playing with Linear Algebra III (No specific deliverable)

16. Matrix manipulations without explicit loops

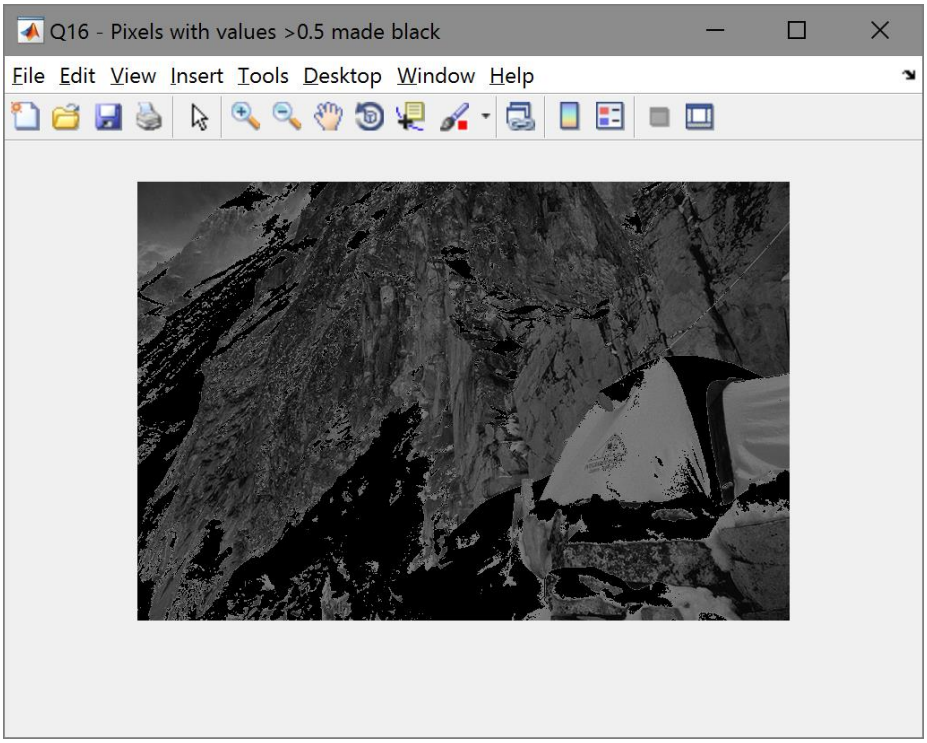
As was mentioned in the hw, we could set every 5th pixel horizontally and vertically to 0 with just one statement by using `colon (:)` operator. Find code snippet and screenshot below.

```
imdata_double_black = imdata_double;
imdata_double_black(1:5:num_rows,1:5:num_cols)=0;
```



For setting all pixels with value >0.5 to 0, I used `find` function to find all the indices where value was >0.5 and set all these indices to 0. It was indeed quite easy once you figure out how to search through the array/matrix.

```
imdata_double_r = imdata_double;
imdata_double_r(find(imdata_double_r>0.5))=0;
```

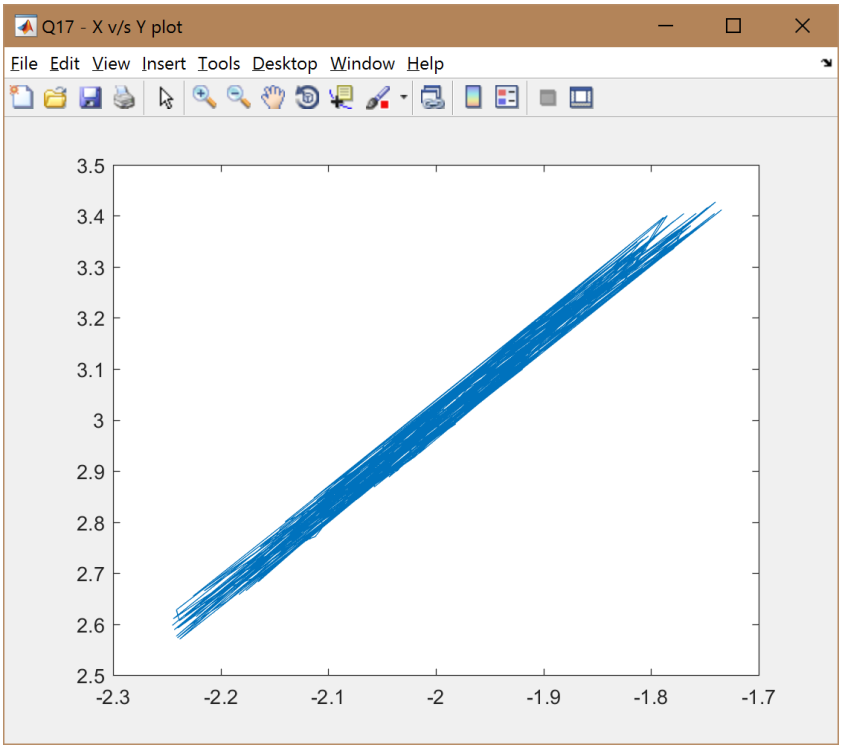


17. PCA I

I used `importdata` function to read the values in and displayed `X` v/s `Y` using `plot`. The covariance matrix for the data was found using `cov` function and below are the values:

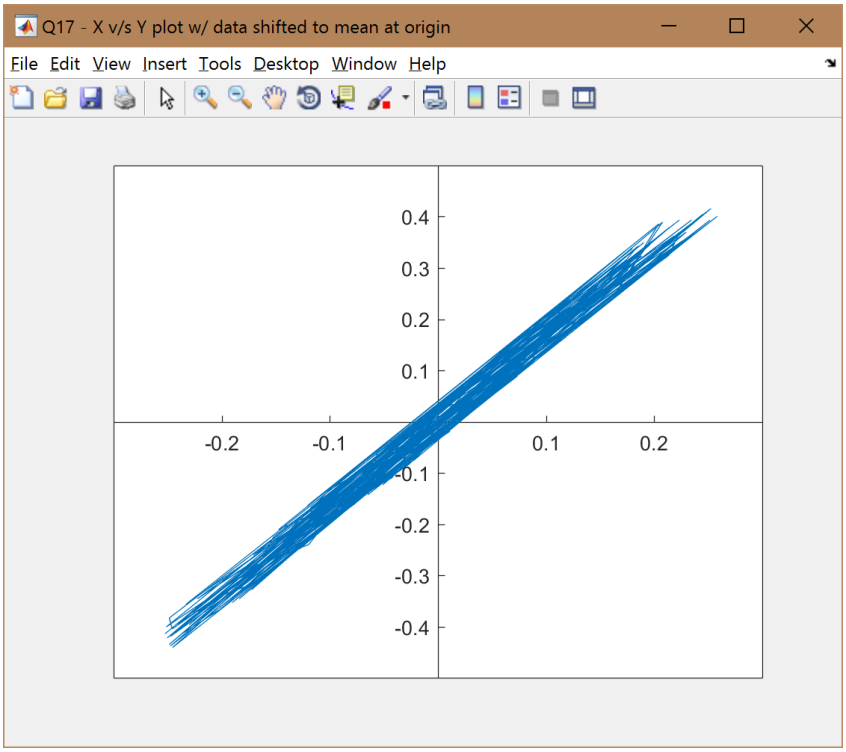
```
Covariance matrix:

    0.0211    0.0356

    0.0356    0.0608
```

```
A=importdata('pca.txt');
X=A(:,1);
Y=A(:,2);
figure('Name','Q17 - X v/s Y plot','NumberTitle','off');
plot(X,Y);
c=cov(A);
disp(['Covariance matrix:']);
disp(c);
```



I shifted the data of `X` and `Y` by an offset of their respective mean values to “mean-center” the data.

In the new coordinate system after rotation, I expect the variance to be along only one variable. The other variable should probably remain constant.



18. PCA II

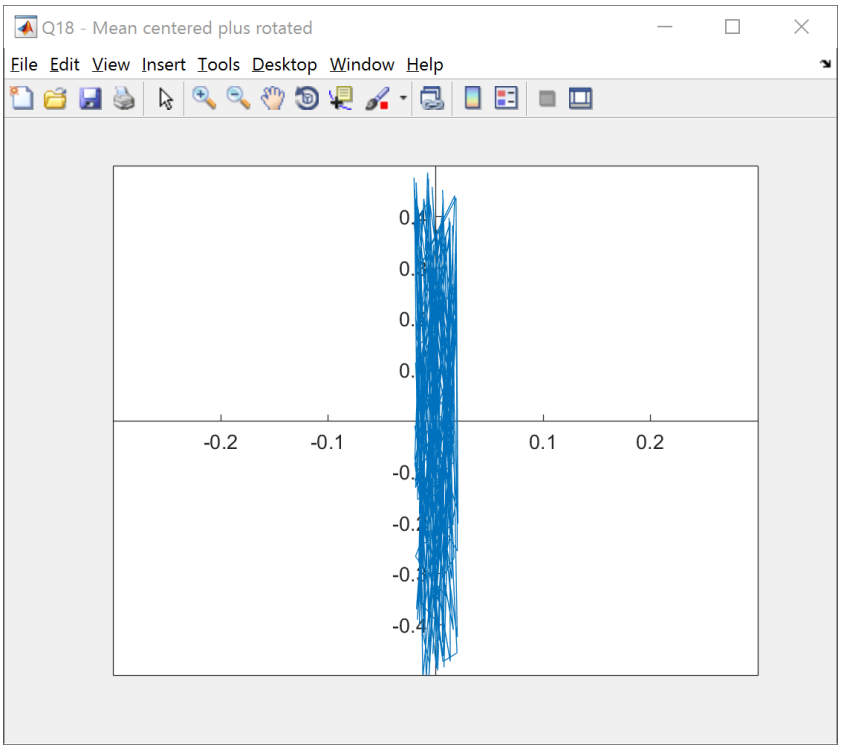
To begin with, I mean-centered the data by subtracting their mean values. I found the covariance matrix of the data and then found eigen vectors of the covariance by using `eig` function. I verified that the `eigen vector (eV)` matrix is orthogonal by the equality of transpose of eV and inverse of eV. Below is the code snippet and output.

```
X=X-xMean;
Y=Y-yMean;
A=[X Y];
c1=cov(A);
[eV,eD]=eig(c1);
disp(['Proving orthogonality:']);
disp(['eigenVector(eV) Matrix:']);
disp(eV);
disp(['transpose(eV):']);
disp(eV');
disp(['inv(eV):']);
disp(inv(eV));
disp(['diff']);
disp(eV'-inv(eV));
```

```
Proving orthogonality:

eigenVector(eV) Matrix:

   -0.8624    0.5063

    0.5063    0.8624

transpose(eV):

   -0.8624    0.5063

    0.5063    0.8624

inv(eV):

   -0.8624    0.5063

    0.5063    0.8624

diff

   1.0e-15 *

   -0.1110    0.1110

    0.1110    0.2220
```

In the next step, I used the `eigen vector (eV)` from before as rotation matrix and transformed the data. Below is the screenshot of the plot of transformed data. The axes are scaled as before to be able to compare the two plots.

```
A_rot=A*eV;
figure('Name','Q18 - Mean centered plus rotated','NumberTitle','off');
plot(A_rot(:,1),A_rot(:,2));
xlim([-0.3 0.3]);
ylim([-0.5 0.5]);
ax=gca;
ax.XAxisLocation='origin';
ax.YAxisLocation='origin';
```



As we can see, the data is now rotated and the variance is mostly along one axis. Re-computing the covariance, we get,

```
Covariance before transformation:

    0.0211    0.0356

    0.0356    0.0608

Covariance after transformation:

    0.0001    0.0000

    0.0000    0.0818

Sum of variance before transformation:

    0.0819

Sum of variance after transformation:

    0.0819
```

We can confirm from the new variance values that the it has drastically decreased along X where as it has increased along Y. However, the sum of both the variance is still same. Also, applying the transformation made sure that the co-variance between X and Y is eliminated which I didn't anticipate earlier.