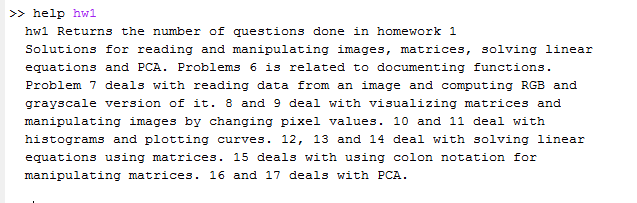
Homework – 1

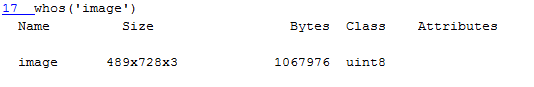
6. Documenting functions



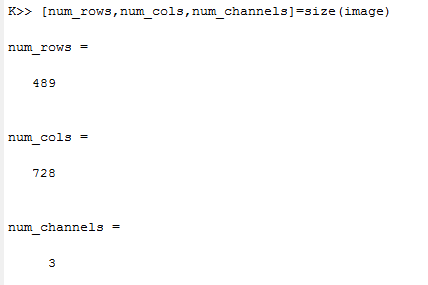
Above is the screenshot of the help command for the ‘hw1’ function.

7. Basic image data structures

(i) illustration of *whos()* function



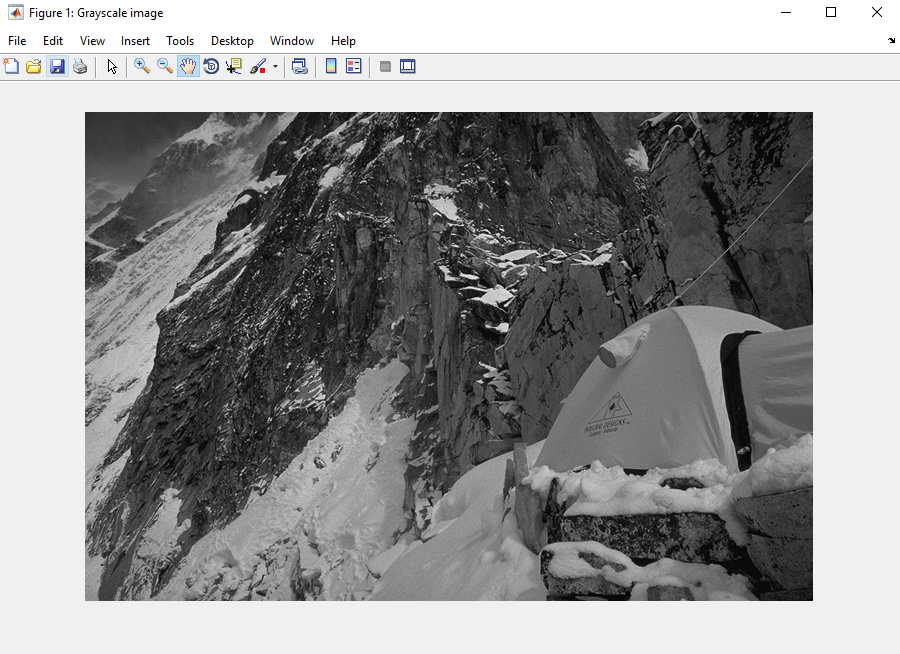
(ii) Calculating size of matrix using *size()*



The range is calculated using the min and max functions. For red it is [0,251]. For blue it is [0,253]. For green it is [0,248]. The overall range is [0,253].

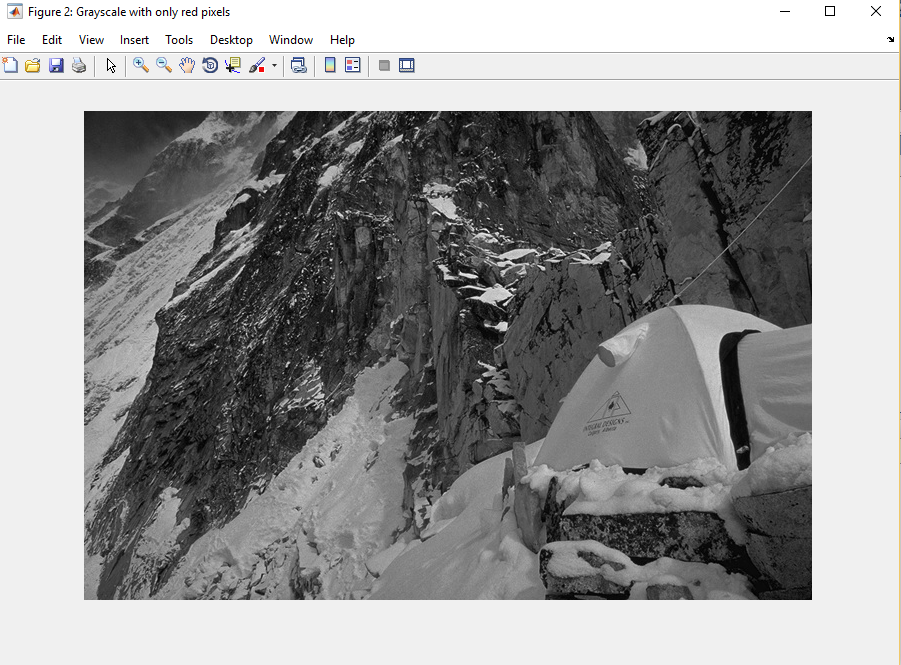
(iii) Converting image to grayscale

Using rgb2gray function, we convert the image to a 2D grayscale image of size 489x728.

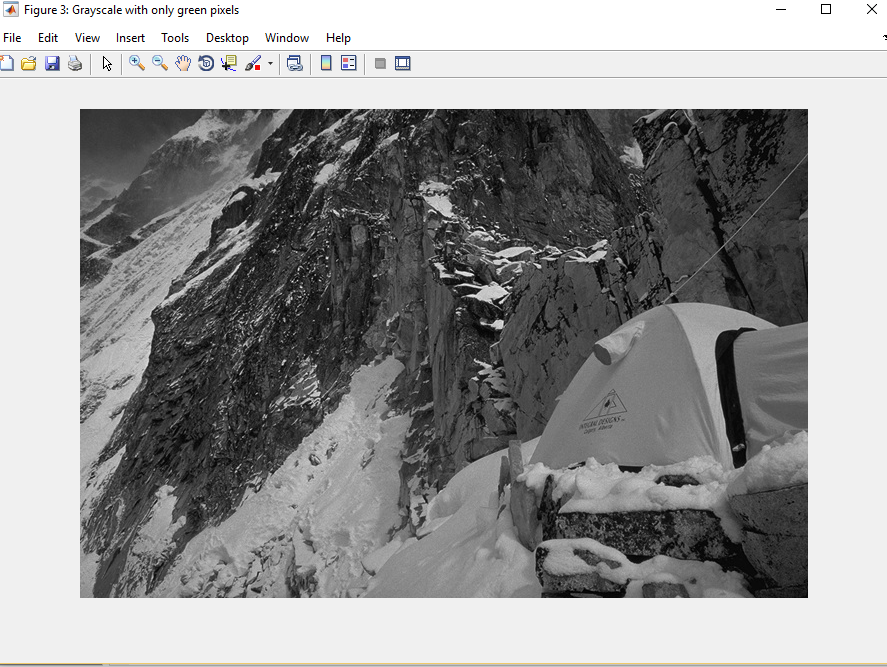


A MATLAB Figure showing the grayscale image obtained using rgb2gray function

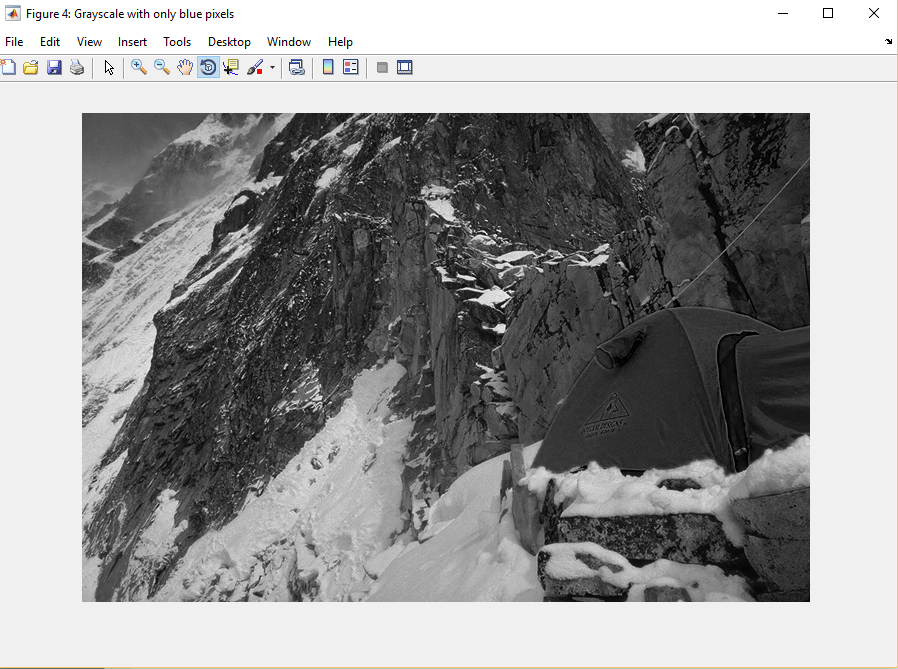
Next, we create 3 separate grayscale images by extracting red, green and blue pixels from the color image into separate arrays.



Grayscale image with only red pixels

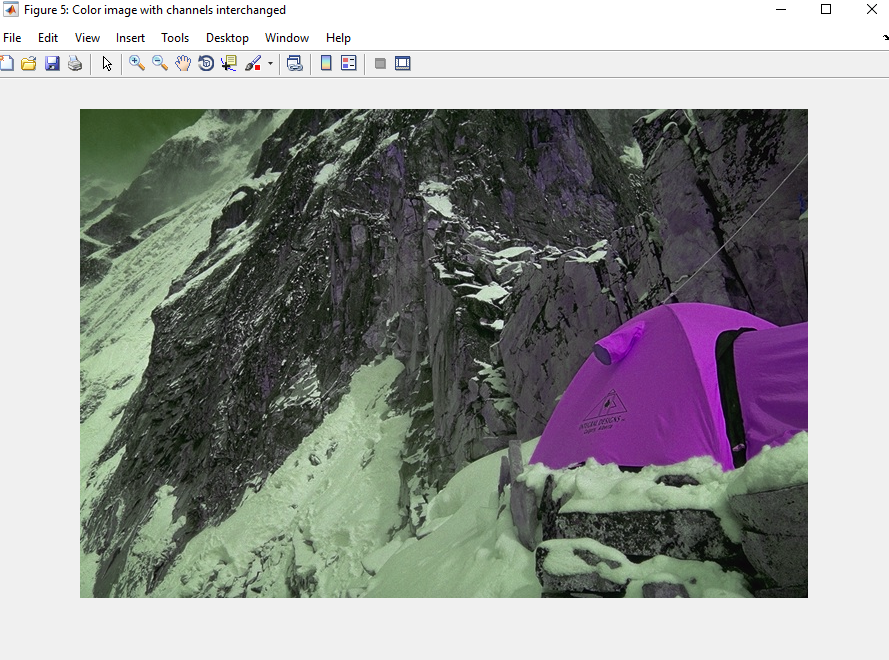


Grayscale picture with only green pixels



Grayscale image with only blue pixels

The images are expected. The image with only blue pixels has a darker shade around the tent compared to the color image. The color image had that part lighter because of the presence of red and green which in combination resulted in yellow. However, in the above image because of the absence of red and green it is darker.



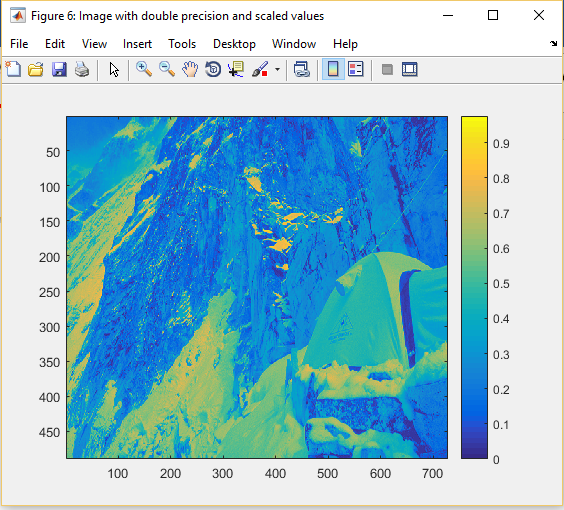
Color image with Red interchanged with Green, Green with Blue and Blue with Red pixels

The above image shows some differences with the color image. Since the channel values are interchanged in this image, the range for red becomes [0,248], for green [0,253] and blue[0,251]. We notice that the range of green has increased from the original image and hence the greenish tone of the image, especially in areas around the top left corner. The tent which was yellow is now purple. Yellow appears with the overlap of red and green. Purple appears with the overlap of red and blue. The intensity and the corresponding values for red and blue are higher compared to the previous image which is responsible for the purple color.

8. ) Visualizing matrices

(i) Display grayscale image with scaled values

We use the grayscale image obtained in the 7th problem and divide the matrix by 255 after type conversion to double to get values in the range [0,1]. The resulting grayscale image is displayed below:

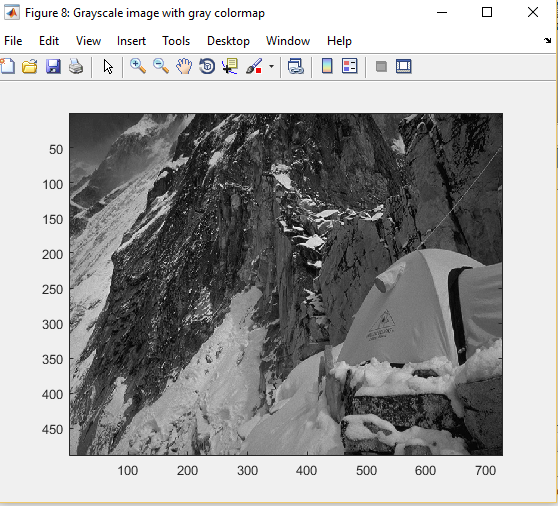


Scaled image displayed using imagesc function and using colorbar

Imagesc is specifically used for visualization as it displays the image with reference to the whole range of color pixels. Hence, the image appears strange.

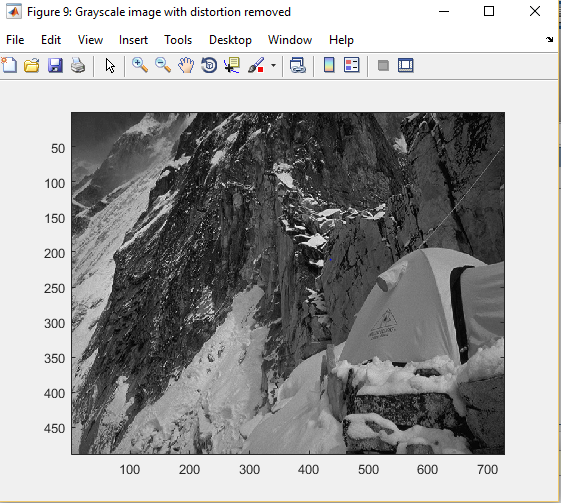
(ii) Grayscale image with gray colormap

Below is the same grayscale image with a gray colormap. A colormap is a 3-column matrix of real numbers. Matlab by default uses a colormap called parula.



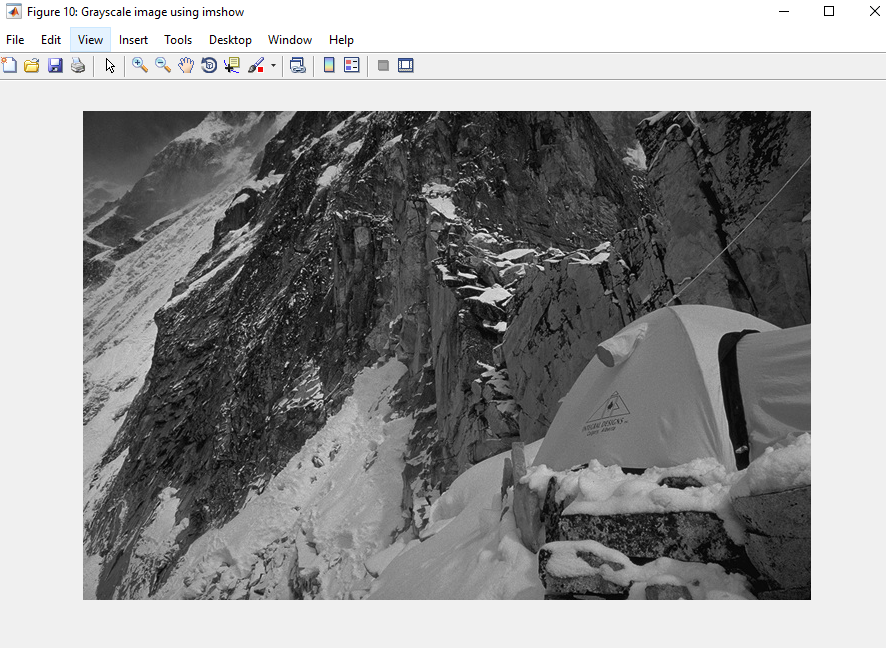
(iv) Grayscale image with distortion and without distortion

We remove the distortion observed in the grayscale by using the axis function. Setting axis to auto we observe that distortion is removed.



(v) Displaying image using imshow

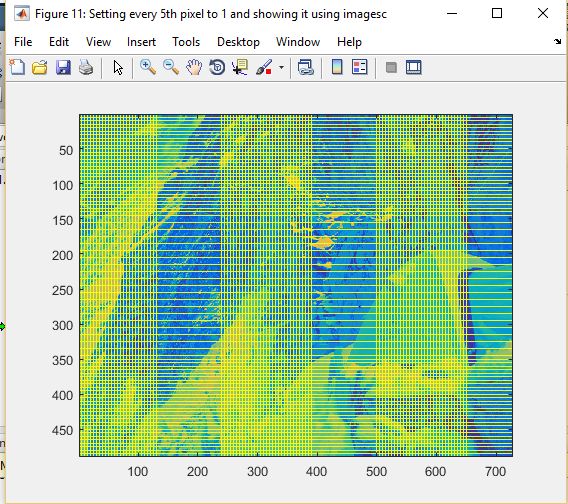
We use the imshow function to display images and not for studying it further in terms of metrics. Imagesc is used for that.



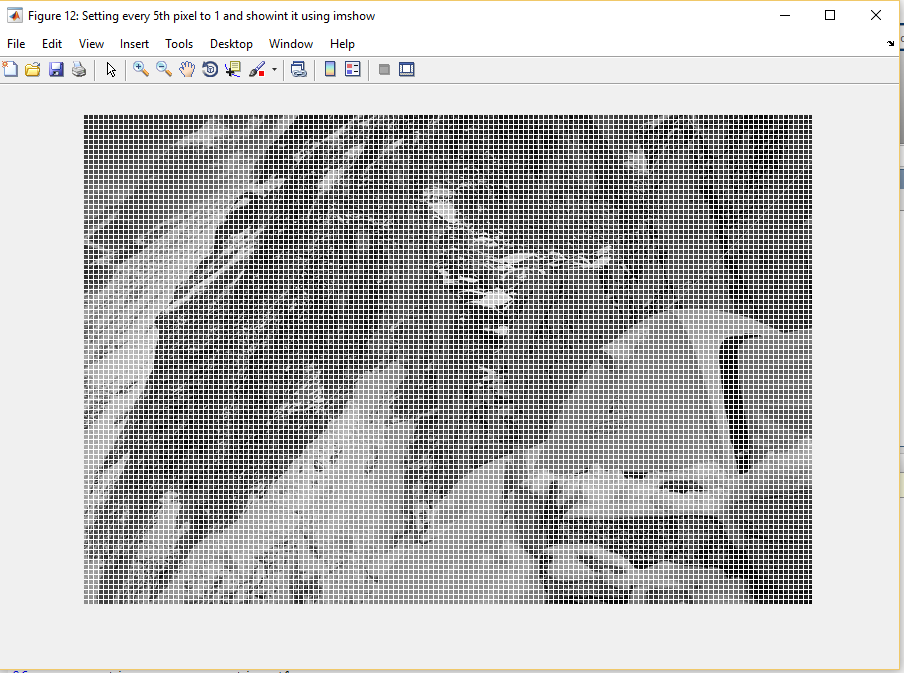
9. ) Manipulating matrices

(i) Setting every 5th pixel to 1

Below is the image when displayed using imagesc after setting every 5th pixel to 1.

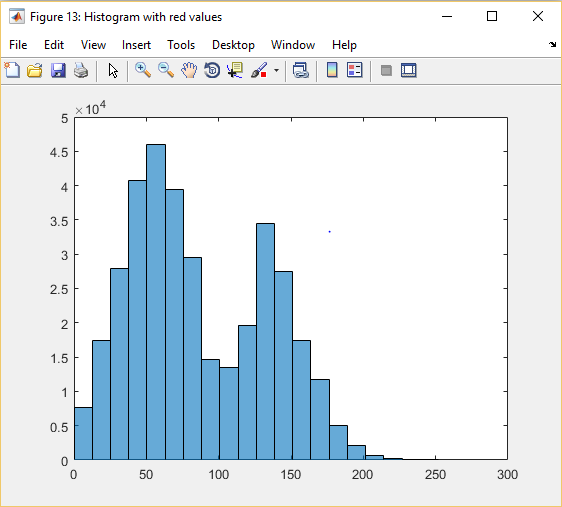


The same image when displayed using imshow is as below. In this case, imagesc seems more appropriate to use as the image has horizontal and vertical lines for scaling. It makes sense to measure areas where color intensities are more and hence I feel imagesc is more suited.



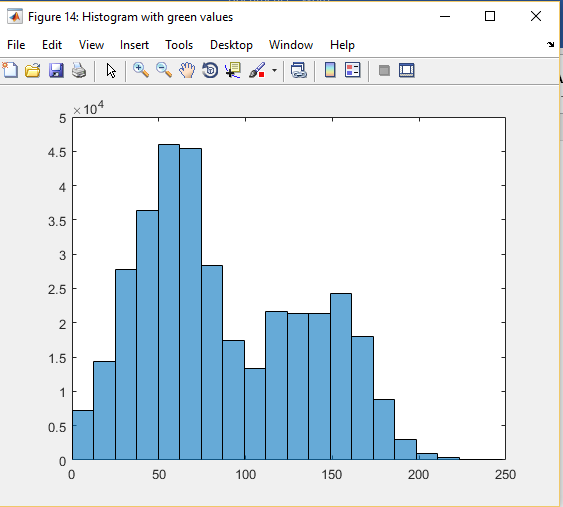
10. ) Histograms

(i) Displaying histograms for Red, Blue and Green over 20 bins



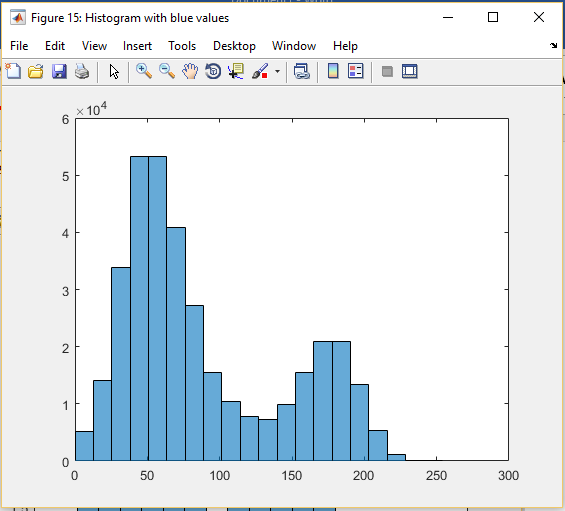
Histogram of the red pixels

From the above histogram, we can infer that there are a large number of red pixels with values around 50-100 and some also around 130-150.



Histogram of the green pixels

The distribution of green pixels seems similar to red but there’s more concentration of it in the 50-100 range comparatively.



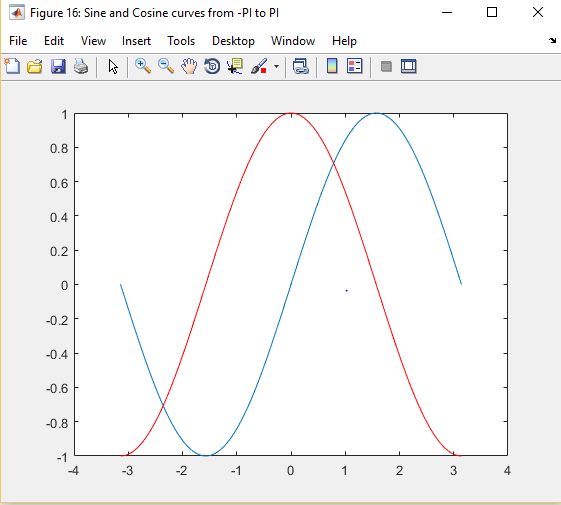
Histogram of the blue pixels

The distribution of blue cells is much higher in the 0-100 range than the remaining values.

These histograms can be used to explain distribution of each set of pixels(R/G/B) independently but cannot explain the behavior when they are combined. (the resulting color combinations)

11. Plotting

Below is a sine and cosine curve form -PI to PI. Both are highlighted in different colors. Sine is the blue curve while Cosine is the red one.



Sin(x) and Cos(x) from -PI to PI

12. Linear Algebra(I)

The solution is x = 1.9375, y = 0.2500, z = 2.1875. To prove that this solution is correct, we can substitute the values in the equations and check whether they hold true.

3\*x+4\*y+z = 3\*1.9375+4\*0.25+2.1875 = 9 (=RHS)

2\*x-y+2\*z = 2\*1.9375-0.25+2\*2.1875 = 8 (RHS)

x + y – z = 1.9375 + 0.25 – 2.1875 = 0 (RHS)

Hence, proving that the solution is correct.

1. Using linsolve and inverse.

Linsolve gives more accurate results compared to inv(). Hence the difference is non-zero. Below is the difference.

1.0e-15 \* -0.2220

0.0278

0

13.) Linear Algebra (II)

The solution for the system of equations is x = 1.8230, y = 0.3451, z = 2.3277. We use the pinv() method to calculate Moore Primrose Inverse and use it in the equation to get the solution.

The error vector is

0.1771

-0.2529

-0.0437

0.1041

-0.1596

0.2555

14. ) Linear Algebra (III)

We take the below random 4x4 matrix A .

0.0474 0.5449 0.3037 0.7218

0.3424 0.6862 0.0462 0.8778

0.7360 0.8936 0.1955 0.5824

0.7947 0.0548 0.7202 0.0707

The eigen vector for the above matrix v is

-0.4295 + 0.0000i -0.7414 + 0.0000i -0.0648 + 0.2045i -0.0648 - 0.2045i

-0.4881 + 0.0000i -0.0247 + 0.0000i 0.5223 + 0.1545i 0.5223 - 0.1545i

-0.6108 + 0.0000i 0.6048 + 0.0000i 0.2454 - 0.5165i 0.2454 + 0.5165i

-0.4520 + 0.0000i 0.2897 + 0.0000i -0.5747 + 0.0000i -0.5747 + 0.0000i

A\*v is as below

-0.7980 + 0.0000i 0.3442 + 0.0000i -0.0588 - 0.0630i -0.0588 + 0.0630i

-0.9069 + 0.0000i 0.0115 + 0.0000i -0.1569 + 0.1522i -0.1569 - 0.1522i

-1.1349 + 0.0000i -0.2808 + 0.0000i 0.1323 + 0.1876i 0.1323 - 0.1876i

-0.8398 + 0.0000i -0.1345 + 0.0000i 0.1133 - 0.2010i 0.1133 + 0.2010i

A\*v./v is as below

1.8581 + 0.0000i -0.4642 + 0.0000i -0.1971 + 0.3497i -0.1971 - 0.3497i

1.8581 + 0.0000i -0.4642 + 0.0000i -0.1971 + 0.3497i -0.1971 - 0.3497i

1.8581 + 0.0000i -0.4642 + 0.0000i -0.1971 + 0.3497i -0.1971 - 0.3497i

1.8581 + 0.0000i -0.4642 + 0.0000i -0.1971 + 0.3497i -0.1971 - 0.3497i

which is the value of k.

15. ) Matrix manipulations without explicit loops

Setting every 5th pixel to 0 using colon notation. This can be done by code snippet below.

rangeX = 5:5:num\_rows;

rangeY = 5:5:num\_cols;

tempImage(rangeX,rangeY)=0;%sets every 5th pixel vertically and horizontally %to 0.

figure('Name','Image created by setting every 5th pixel to 0 using colon notation')

imshow(tempImage)

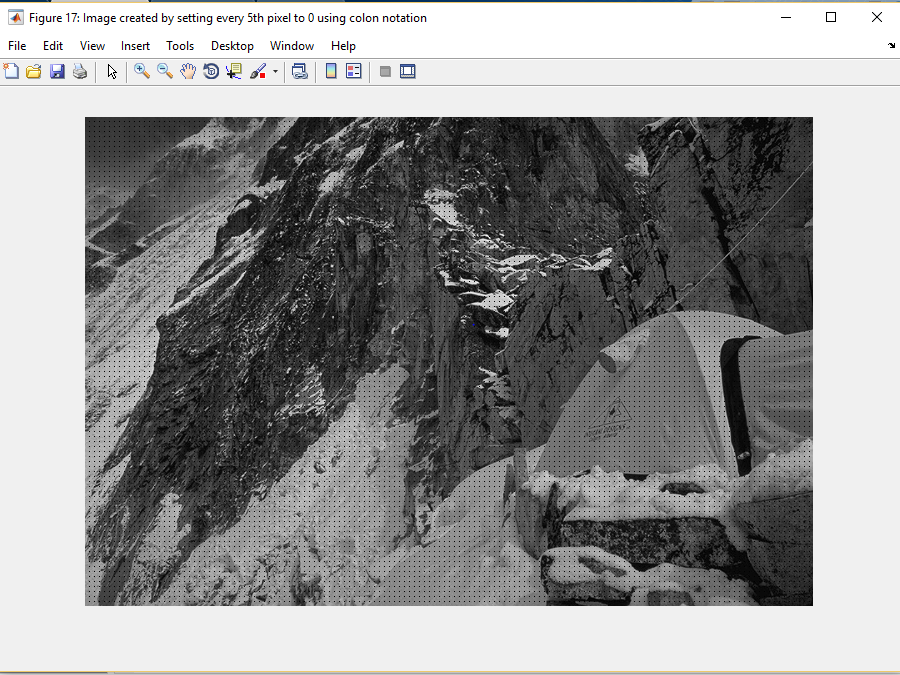


Image showing the square lattice pattern after setting every 5th pixel to 0

And the next step is to set every pixel greater than 0.5 to 0. The resultant figure is

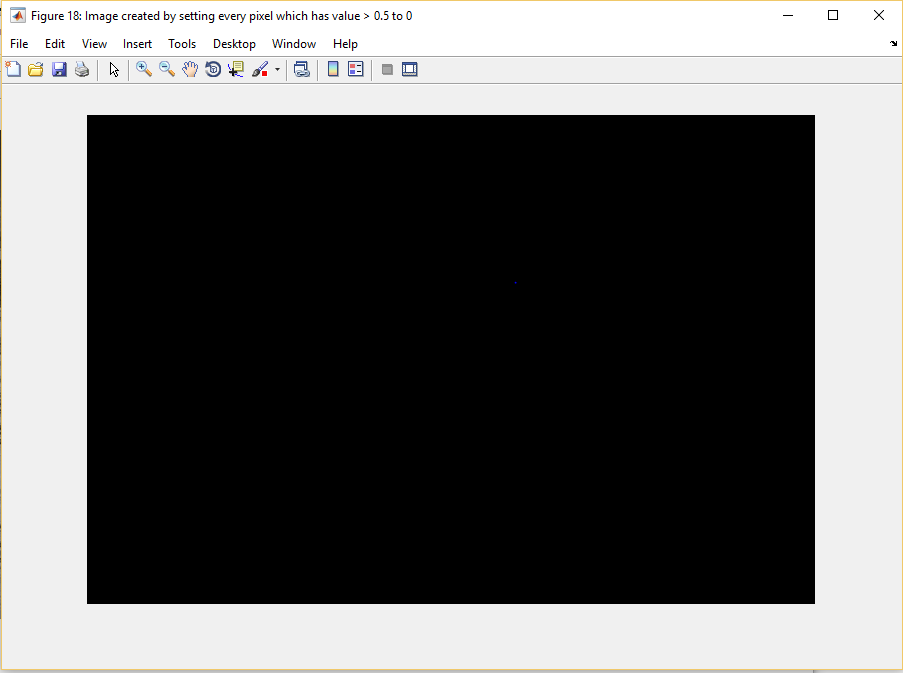
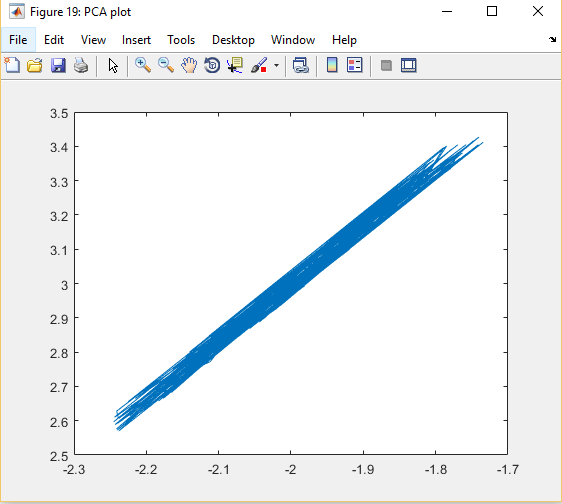


Image on setting all pixels greater than 0.5 to 0

16. ) PCA(I)

On reading the file pca.txt into two separate vectors X and Y for two columns, we plot the graph and it is as below:



Plot of the data

Covariance matrix is as below

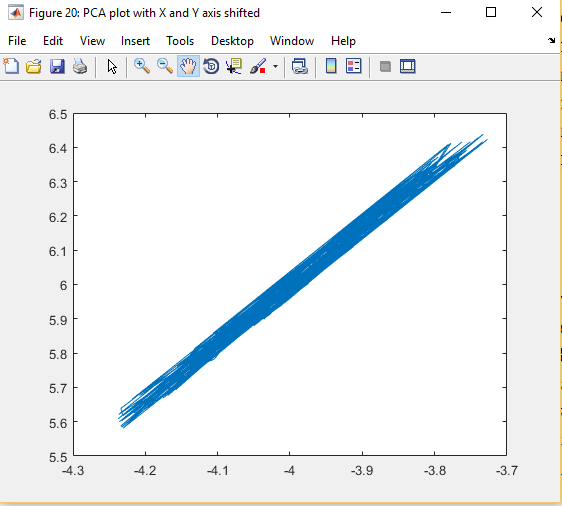
covariance =

0.0211 0.0356

0.0356 0.0608

We mean-center the data and then plot the graph. Mean-centering can be done by computing the mean and adding it to the points so that the mean becomes the origin.

The plot then becomes as below



Plot after data is mean centered.

The natural coordinate system for this would be one with a single dimension. Basically, the Y-axis can be rotated to coincide with the X-axis. The covariance would then be zero as the data is single dimensional.

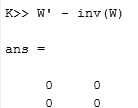
17. ) PCA(II)

We zero center the data and then compute the eigen vectors for the covariance matrix of this zero centered data. The eigen vector W is

-0.7071 0.7071

0.7071 0.7071

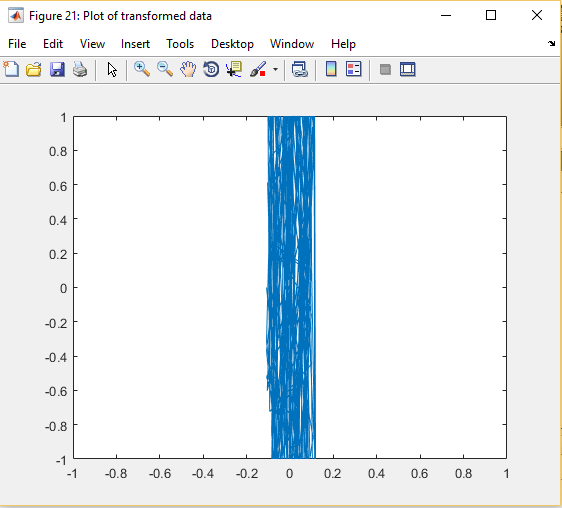
We prove that this is orthogonal by showing W’ – inv(W) = 0



We order the eigen vector matrix in the descending order of values so that we can multiply the largest one with the data.

Multiplying this with the zero-centered data matrix we get a 200\*2 matrix which consists of values called principal components.

The plot of the resultant X and Y values is as below. We chose both the X and Y axes to be in [-1,1] range. We see the data seems to be tending towards single dimension.



The covariance matrix for the transformed data is

1.9954 0.0000

0.0000 0.0046

We see that the covariance values are zero and the variance values for X and Y are 1.9954 and 0.0046 respectively. This makes sense because the dimensionality of the data has been reduced and hence there seems to be no dependency between X and Y values. In the original covariance matrix, the covariance values were non-zero which meant that there was a dependency between the two vectors, X and Y.