CS 696 Applied Computer Vision

Homework Assignment 1

Question 1.2:

a) >> x = randperm(5)

This function returns a series of random permutations of integers between 0 and the specified number in the brackets, which in this case is 5.
For example [2, 4, 1, 3, 5]

b) >> **a = [1:10];**
>> **b = a([1:3:end]);**

The first command a, creates a series of integers starting from 1 and ending at 10, with the default step being 1 i.e. no element is skipped. This is then stored in the variable a.
The second command references the elements of the array a with the first parameter being the starting index, the second parameter being the step value, and the last parameter being the ending index.
This returns the values:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 4, 7, 10]

c) >> **f = [1501:2000];**
>> **g = find(f > 1850);**
>> **h = f(g);**

The first command f, initializes a series of numbers starting from 1501 and ending at 2000 and stores it in the variable f.
The second function returns the indices of all the elements which match the condition placed inside the brackets. Here, it would return the indices of all the elements which are greater than 1850 and store it in the g variable.
The last function would return the actual values of all the elements whose indices are stored in g. Since they are the indices of the elements from f, they can be accessed by having the index location within the bracket.
Example: f = [1501, 1502, 1503… 2000]
g = [351, 352, 353... 500]
h = [1851, 1852, 1853… 2000]

d) >> **x = 22.*ones(1,10);**
>> **y = sum(x);**

The initial statement first creates 10 ones from the ones function and then each element, the 10 ones, is multiplied by the 22. All the values are stored in the x.
The second statement adds all the elements of the variable x and then stores the sum in the variable y.
Example: x = [22, 22, 22, 22, 22, 22, 22, 22, 22, 22]
Y = 220

e) >> a = [1:1000];
>> b = a([end:-1:1]);

The first statement creates a series of elements from 1 to 1000 and stores it in the variable a.

The next statement gets the values of all the elements from a starting from the end (1000), and ending at 1. The middle parameter in the function is the step value that is supposed to be -1 in order for the numbers to decrease from 1000 to 1.

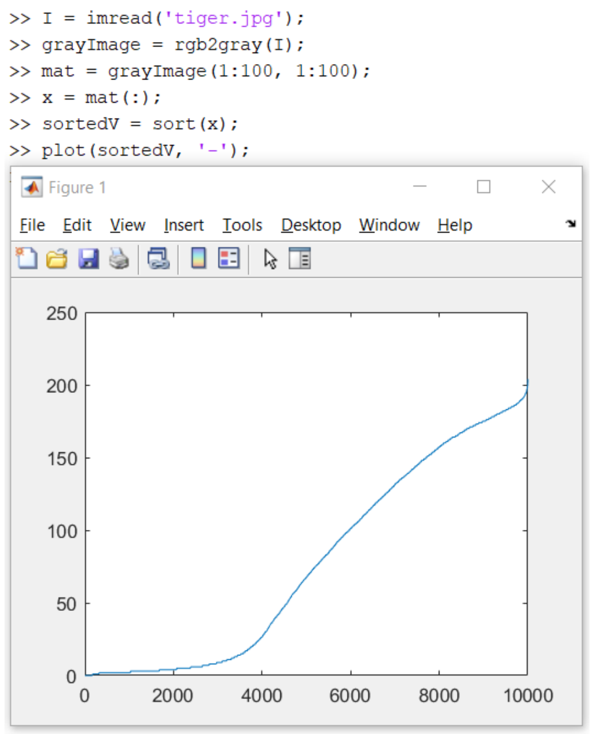
Question 3:

There is a default code written by me to read an image, convert it to a grayscale and store it in a variable and initialize a matrix of 100 pixels by 100 pixels for the image.
The code is as follows:

```
I = imread('tiger.jpg');
grayImage = rgb2gray(I);
mat = grayImage(1:100, 1:100);
```

a)
```
X = mat(:);
sorted = sort(x);
plot(sorted, '-');
```

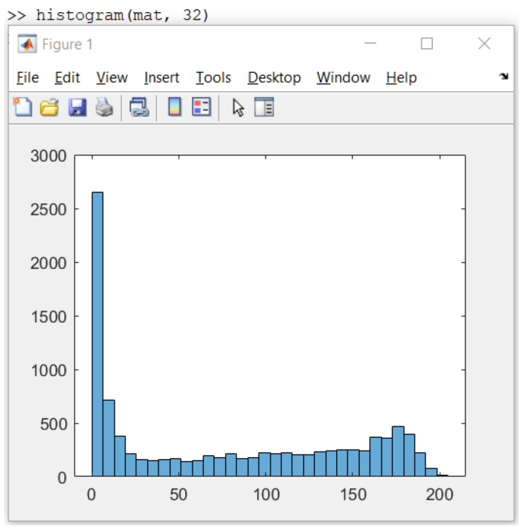
plots the vector x into the graph



```
>> I = imread('tiger.jpg');
>> grayImage = rgb2gray(I);
>> mat = grayImage(1:100, 1:100);
>> x = mat(:);
>> sortedV = sort(x);
>> plot(sortedV, '-');
```

b) `Histogram(mat, 32);`

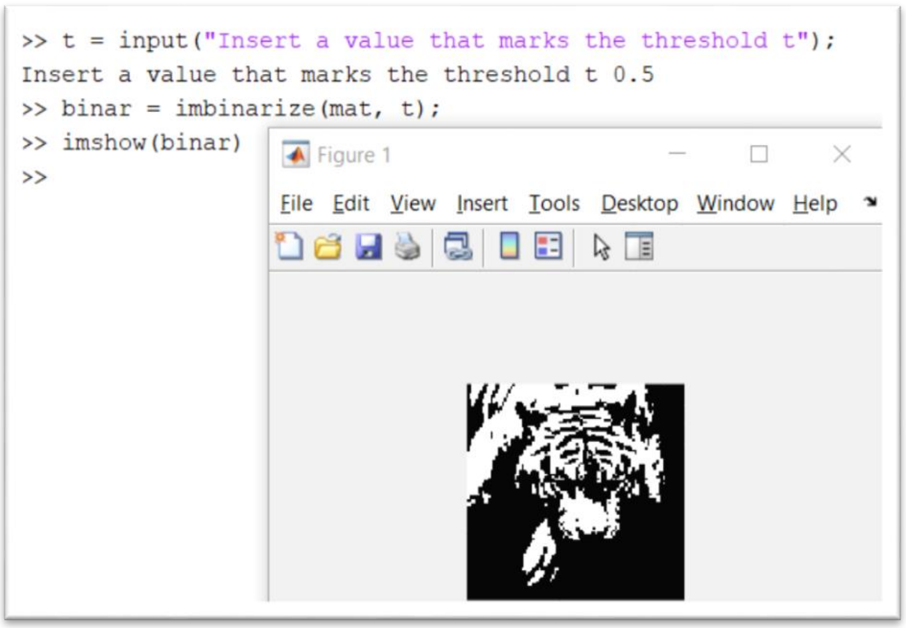
Creates a histogram for the matrix mat with 32 bins



```
>> histogram(mat, 32)
```

c)
```
t = input("Insert a value between 0 and 1 that marks the threshold t");
binar = imbinarize(mat, t);
imshow(binar)
```

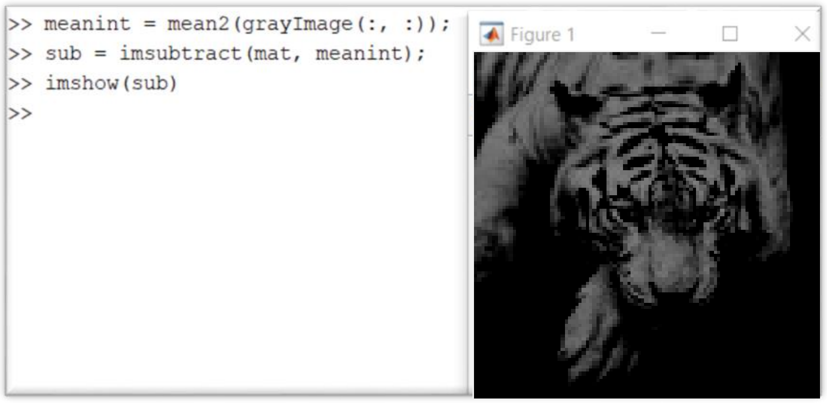
takes an input between 0 to 1 and uses it to create a binary image with any pixel below the threshold t will be 0 that it white and black elsewhere.



```
>> t = input("Insert a value that marks the threshold t");
Insert a value that marks the threshold t 0.5
>> binar = imbinarize(mat, t);
>> imshow(binar)
>>
```

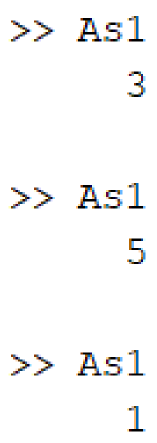
d)
```
meanint = mean2(grayImage(:, :));
sub = imsubtract(mat, meanint);
imshow(sub)
```

The matrix is provided below:

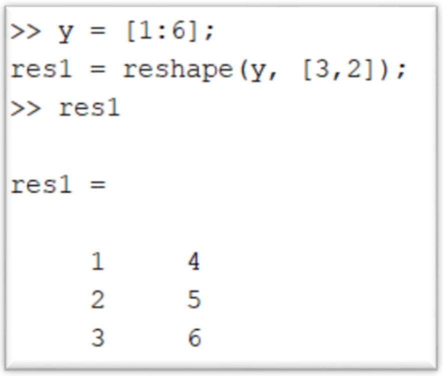


```
>> meanint = mean2(grayImage(:, :));
>> sub = imsubtract(mat, meanint);
>> imshow(sub)
>>
```

e)
```
rollDice();
% Function for Q5 - e)
function dice = rollDice()
min = 1;
max = 6;
r = round(min + (max-min).*rand(1));
%r = rand();
if mod(r, 1) == 0
    dice = 1;
elseif mod(r, 2) == 0
    dice = 2;
elseif mod(r, 3) == 0
    dice = 3;
elseif mod(r,4) == 0
    dice = 4;
elseif mod(r,5) == 0
    dice  = 5;
else
    dice = 6;
end
disp(dice);
end
```


```
>> As1
     3

>> As1
     5

>> As1
     1
```

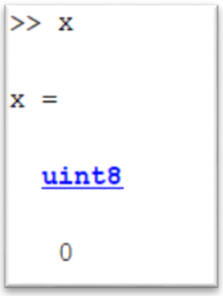
f)
```
Y = [1:6];
res1 = reshape(y, [3,2]);
```

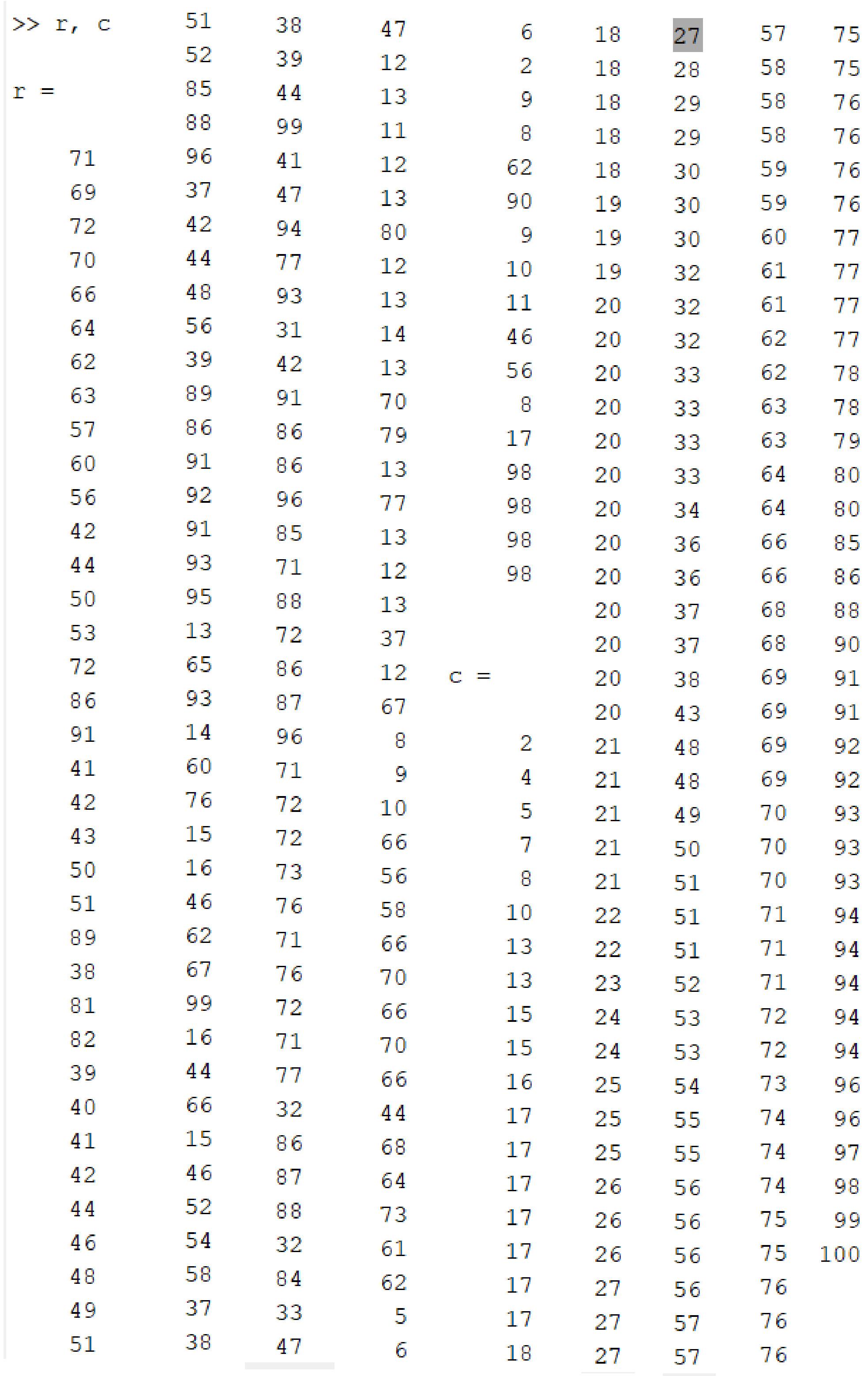

```
>> y = [1:6];
res1 = reshape(y, [3,2]);
>> res1

res1 =

     1     4
     2     5
     3     6
```

g)
```
Single = mat(:);
X = min(single);
[r, c] = find(mat == x);
```

Since my image has several minimum value, there are multiple values of rows and columns stored in the vectors.

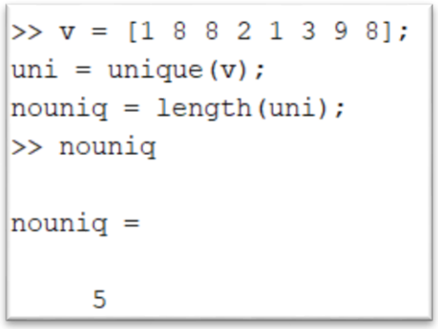
The values of rows and columns are given below:


```
>> x

x =

  uint8

   0
```



```
>> r, c     51    38    47     6    18    27    57    75
            52    39    12     2    18    28    58    75
r =         85    44    13     9    18    29    58    76
            88    99    11     8    18    29    58    76
    71      96    41    12    62    18    30    59    76
    69      37    47    13    90    19    30    59    76
    72      42    94    80     9    19    30    60    77
    70      44    77    12    10    19    32    61    77
    66      48    93    13    11    20    32    61    77
    64      56    31    14    46    20    32    62    77
    62      39    42    13    56    20    33    62    78
    63      89    91    70     8    20    33    63    78
    57      86    86    79    17    20    33    63    79
    60      91    86    13    98    20    33    64    80
    56      92    96    77    98    20    34    64    80
    42      91    85    13    98    20    36    66    85
    44      93    71    12    98    20    36    66    86
    50      95    88    13          20    37    68    88
    53      13    72    37          20    37    68    90
    72      65    86    12   c =    20    38    69    91
    86      93    87    67          20    43    69    91
    91      14    96     8     2    21    48    69    92
    41      60    71     9     4    21    48    69    92
    42      76    72    10     5    21    49    70    93
    43      15    72    66     7    21    50    70    93
    50      16    73    56     8    21    51    70    93
    51      46    76    58    10    22    51    71    94
    89      62    71    66    13    22    51    71    94
    38      67    76    70    13    23    52    71    94
    81      99    72    66    15    24    53    72    94
    82      16    71    70    15    24    53    72    94
    39      44    77    66    16    25    54    73    96
    40      66    32    44    17    25    55    74    96
    41      15    86    68    17    25    55    74    97
    42      46    87    64    17    26    56    74    98
    44      52    88    73    17    26    56    75    99
    46      54    32    61    17    26    56    75   100
    48      58    84    62    17    27    56    76
    49      37    33     5    17    27    57    76
    51      38    47     6    18    27    57    76
```

h)
```
v =[1 8 8 2 1 3 9 8];
uni = unique(v);
nounique = length(uni);
```



```
>> v = [1 8 8 2 1 3 9 8];
uni = unique(v);
nouniq = length(uni);
>> nouniq

nouniq =

     5
```

Part 2

Question 2.1

```
%-------------------------------------------------------------
% To compute the average image in grayscale

sumOfImage1 = zeros(215, 300, 3, 'double');
listOfFiles = dir('E:\Data\SDSU\Sem 1\ACV\Assignments\HA 1\ha1\set1\*.jpg');
for i=1:length(listOfFiles)
    imgName = ['E:\Data\SDSU\Sem 1\ACV\Assignments\HA 1\ha1\set1\'
listOfFiles(i).name];
    newim = imread(imgName);
    sumOfImage1 = sumOfImage1 + im2double(newim);
end
sumOfImage1 = sumOfImage1./length(listOfFiles);
%imshow(sumOfImage1);

sumOfImage2 = zeros(164, 398, 3, 'double');
listOfFiles2 = dir('E:\Data\SDSU\Sem 1\ACV\Assignments\HA 1\ha1\set2\*.jpg');
for i=1:length(listOfFiles2)
    imgName = ['E:\Data\SDSU\Sem 1\ACV\Assignments\HA 1\ha1\set2\'
listOfFiles2(i).name];
    newim = imread(imgName);
    sumOfImage2 = sumOfImage2 + im2double(newim);
end
sumOfImage2 = sumOfImage2./length(listOfFiles2);
sumOfImage1 = rgb2gray(sumOfImage1);
sumOfImage2 = rgb2gray(sumOfImage2);

%-------------------------------------------------------------
```

Here, a base matrix consisting of zeroes with the column and row size of the other images is created.
After that, all the images are looped to access each of them and each one is added to the sumOfImage variable. Each value in the matrix is then divided by the total number of images and the average of the images is created in sumOfImage variable.
It was then converted to the grayscale using rgb2gray() function.

Question 2.2

```
%-------------------------------------------------------------
% To compute the average image in color, by averaging per
RGB channel.

rgbSumOfImage1 = sumOfImage1;
rgbSumOfImage2 = sumOfImage2;

%-------------------------------------------------------------
```

Here, the average of colored images was already created in the earlier step, so we just needed to place it in a second variable without converting it to the grayscale.

Question 2.3

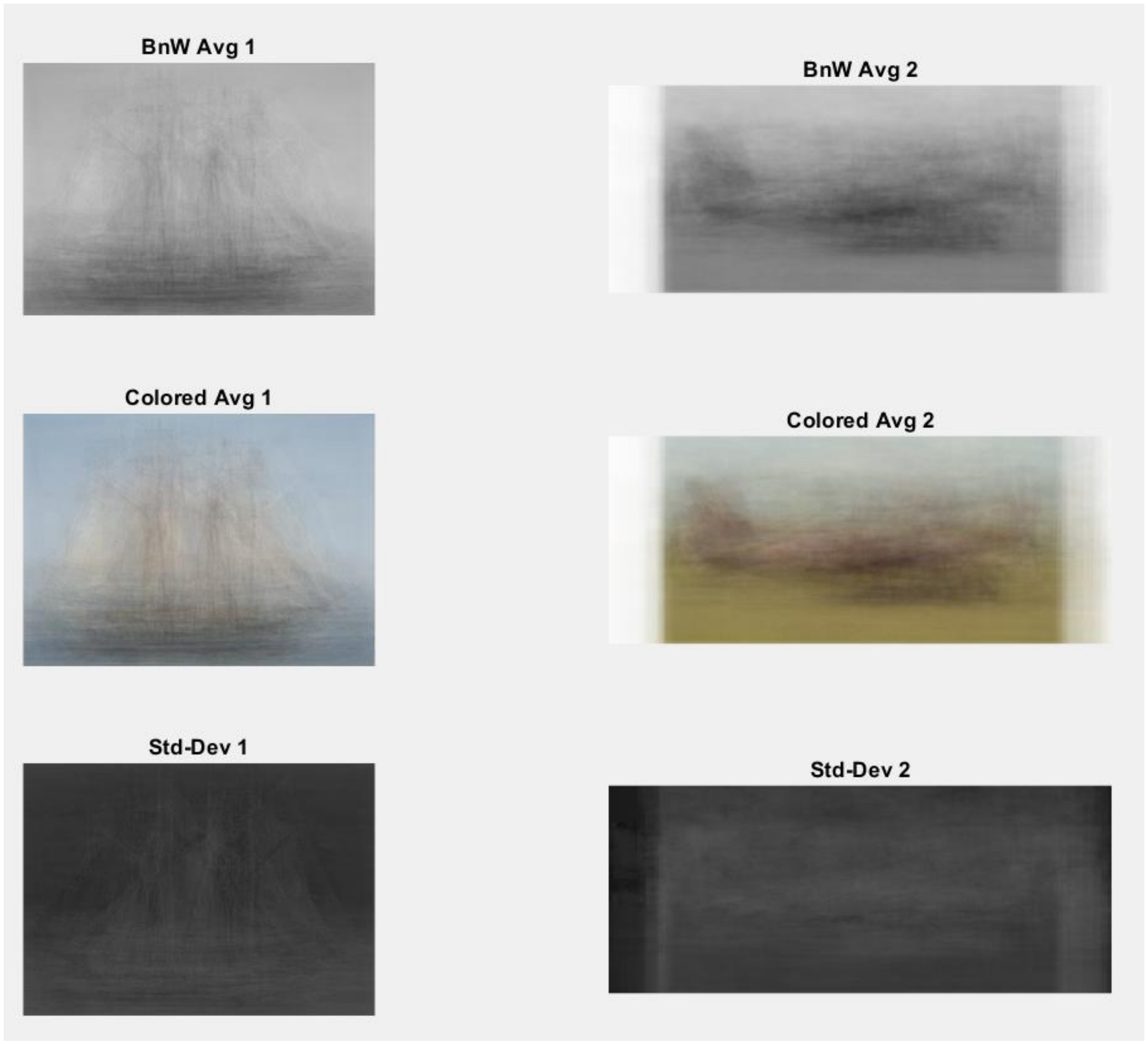
```
%-----------------------------------------------------------
% To compute a matrix holding the grayscale images’
standard deviation at each pixel

listOfFiles = dir('E:\Data\SDSU\Sem 1\ACV\Assignments\HA 1\ha1\set1\*.jpg');

sumImage1 = zeros(215, 300, 3, numel(listOfFiles), 'double');

sumImage1_gray = zeros(215, 300, numel(listOfFiles), 'double');

for i=1:length(listOfFiles)
    imgName = ['E:\Data\SDSU\Sem 1\ACV\Assignments\HA 1\ha1\set1\'
listOfFiles(i).name];
    newim = imread(imgName);
    sumImage1(:,:,:,i) = im2double(newim);
    sumImage1_gray(:,:,i) = rgb2gray(sumImage1(:,:,:,i));
end

sumImage_gray_avg = mean(sumImage1_gray, 3);
sumImage_rgb = mean(sumImage1, 4);
sumImage_stddev = std(sumImage1_gray, [], 3);

listOfFiles2 = dir('E:\Data\SDSU\Sem 1\ACV\Assignments\HA 1\ha1\set2\*.jpg');

sumImage2 = zeros(164, 398, 3, numel(listOfFiles2), 'double');

sumImage2_gray = zeros(164, 398, numel(listOfFiles2), 'double');

for i=1:length(listOfFiles2)
    imgName = ['E:\Data\SDSU\Sem 1\ACV\Assignments\HA 1\ha1\set2\'
listOfFiles2(i).name];
    newim = imread(imgName);
    sumImage2(:,:,:,i) = im2double(newim);
    sumImage2_gray(:,:,i) = rgb2gray(sumImage2(:,:,:,i));
end

sumImage_gray_avg_2 = mean(sumImage2_gray, 3);
sumImage_rgb_2 = mean(sumImage2, 4);
sumImage_stddev_2 = std(sumImage2_gray, [], 3);

%-----------------------------------------------------------
```

Here, 2 separate dummy matrices are created, one for the colored average image, other for the grayscaled average image with the dimensions of the other images from the set.
We similarly traverse all the images and add the double values of the images to the dummy matrices (This is done since we need to have mathematical calculations on it to get the standard deviation). This is done for both the colored and grayscale images/matrices.
We then calculate mean of the images using the mean() function. We also get the standard deviation of the images using the std() function. We get the result in the form of matrix which can even further be translated to be used to plot an image.

Question 2.4

```
% -----------------------------------------------------------------
% Displaying each of them

subplot(3,2,1), imshow(rgb2gray(sumOfImage1)), title('BnW Avg 1');
subplot(3,2,2), imshow(rgb2gray(sumOfImage2)), title('BnW Avg 2');

subplot(3,2,3), imshow(rgbSumOfImage1), title('Colored Avg 1');
subplot(3,2,4), imshow(rgbSumOfImage2); title('Colored Avg 2');

subplot(3,2,5), imshow(sumImage_stddev), title('Std-Dev 1');
subplot(3,2,6), imshow(sumImage_stddev_2), title('Std-Dev 2');
```

Here, the images are simply displayed to the user using the subplot() and the imshow() functions.



The Average Images are just the overlapping of all the images and so they look like that.
For the standard deviation, it shows by how much are the images different from the average image, this can be used for examples such as Image Sharpening etc.