Nathan Tung
004-059-195
Discussion 1A

CS170A – Homework 1

1. **Mandrills**

```
function B = squareSubmatrix(A)
      [p,q]=size(A);
      n=min(p,q);
      centerp=p/2;
      centerq=q/2;
      B=A(centerp-n/2+1:centerp+n/2, centerq-n/2+1:centerq+n/2);
      %B=A(1:n, 1:n); would (incorrectly) provide the corner nxn matrix
```

In this function, we determine the dimensions of matrix A; our resulting square matrix has dimension n equal to the minimum of these two dimensions. Then we determine the center point of matrix A and move outwards by n from that point to obtain a centered square matrix . If we do not need a centered resulting matrix, it would suffice to set B=A(1:n,1:n).

Sample Usage:

```
Q =  1     2     3     4     5
     6     7     8     9    10
    11    12    13    14    15

squareSubmatrix(Q) =
     2     3     4
     7     8     9
    12    13    14
```

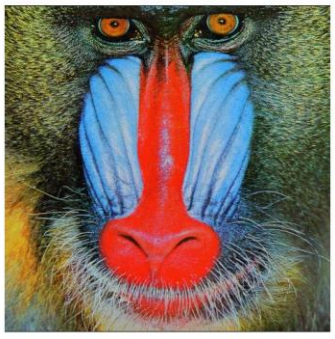
2. **Color Inversion**

```
function colorInversion(A)
      imshow( 1-double(A));
```

A quick glance at matrix A shows us that the numbers are all doubles. Using unsigned integer functions would make the resulting image all black or all white. Therefore, to invert the colors, we simply subtract the values of matrix A from 1 then proceed to show the matrix image.

Sample Usage:

`imshow(A)`



`colorInversion(A)`

Nathan Tung
004-059-195
Discussion 1A

3. **Color Inversion Movie**

```
function linearInterpolation(A,Z,N)
      t = linspace(0.0, 1.0, N+1);
      for i=1:N+1
            u=t(i);
            M(i)=im2frame( (1-u)*A + u*Z )
      end
movie(M);
```

This function takes in the image matrix A, the inversely colored image matrix Z, and some integer N in order to play N+1 frames in a movie of the transition from A to Z. Alternatively (and interestingly), we could have created an actual .avi file using the code below.

```
t = linspace(0.0, 1.0, N+1);
fig=figure;
aviobj = avifile('morphing_color_inversion.avi')
for i=1:N+1
    u=t(i);
    F=( (1-u)*A + u*Z );
    aviobj = addframe(aviobj,F);
end
close(fig)
aviobj = close(aviobj);
```

However, we will stick with using the former function. Calling `linearInterpolation(A,Z,10)` plays a movie with a transition between A to Z, with a gray image when t=0.5.

Sample Usage:

`linearInterpolation(A,Z,10)`

4. **imagesvd (Extra Credit)**

To specify a certain image we want to break down, change the `load('file')` to `load('mandrill')`.

Nathan Tung
004-059-195
Discussion 1A

5. **Blurring**

```
function A = Blur(n)
   c=2*ones(n,1);
   d=ones(n-1,1);
   A=diag(c)+diag(d,-1)+diag(d,1);
   A(1,1)=3;
   A(n,n)=3;
   A=0.25*A;
```

This function generates a vector of ones c (which we multiply by two) to form the center diagonal. The two surrounding diagonals are composed of a vector of ones d. We simply add the diagonals offset with each other to form tridiagonal matrix A, then assign the two top-left and bottom-right corners to have a value of 3. Lastly, we divide the entire matrix A by 4.

We are able to compute the blur matrices S^k to determine the integer k that makes all entries nonzero. Testing out small values of k shows us that for each k in S^k, k+1 out of n parts of the matrix are filled. Thus, to fill up a matrix with dimensions n, we need k=n-1, and finally to fill up the S^k matrix of n=480, we would need k=479.

<u>Sample Usage:</u>

```
Blur(5) =
    0.7500    0.2500         0         0         0
    0.2500    0.5000    0.2500         0         0
         0    0.2500    0.5000    0.2500         0
         0         0    0.2500    0.5000    0.2500
         0         0         0    0.2500    0.7500
```

6. **Progressive Blurredness**

```
A4=rgb2image(T^4*R*T^4,T^4*G*T^4,T^4*B*T^4)
A8=rgb2image(T^8*R*T^8,T^8*G*T^8,T^8*B*T^8)
```

Here we start with $T=S^8$. We obtain the matrices of the slightly-blurred image along with the much-more blurred image, which we call A4 and A8, respectively (shown above). To interpolate between these two blurred images, we can call the function we wrote above in Problem 3: `linearInterpolation(A4, A8, 10)`. The last parameter is any number N that will affect the length of the interpolation movie. To extrapolate, we simply extend the exponent on T to some integer as desired, rather than explicitly 8. We can put it into function form:

```
function B = progressiveBlurredness(A,N)
       [R,G,B] = image2rgb(A);
      t = linspace(0.0, 1.0, N+1);
      S=Blur(max(size(A)));
      T=S^8;
      A4=rgb2image(T^4*R*T^4,T^4*G*T^4,T^4*B*T^4);
      A8=rgb2image(T^8*R*T^8,T^8*G*T^8,T^8*B*T^8);
      for i=1:N+1
          u=t(i);
          M(i)=im2frame( (1-u)*A4 + u*A8 );
      end
      movie(M);
```

Nathan Tung
004-059-195
Discussion 1A

Sample Usage:

```
imshow(A4)
```

```
imshow(A8)
```

```
progressiveBlurredness(A,10)
```

7. **Extrapolation**

```
function linearExtrapolation(A,Z,T,N)

t = linspace(1.0, T, N+1);
for i=1:N+1
    u=t(i);
    X=(1-u)*A + u*Z;
    newX = (X-min(X(:)))./(max(X(:)-min(X(:)))); %scales X to [0,1]
    M(i)=im2frame( newX );
end

movie(M);
```

This function we wrote is similar to the linearInterpolation above. However, we've added a parameter T; the function extrapolates from 1.0 until some double T. It does not make sense to use large values of T, since extrapolation is essentially using past data to make an "educated guess" as to what the future data looks like. Straying too far from our real given data may return much more inaccurate results. We calculate a newX matrix in order to get rid of negatives and to normalize (rescale) the image back to a [0,1] range. As before, N+1 tells us the number of frames we want in our movie.

In order to sharpen an image, we can use either extrapolation or interpolation. Both of these functions produce a sharper image (relative to the blurred image input).

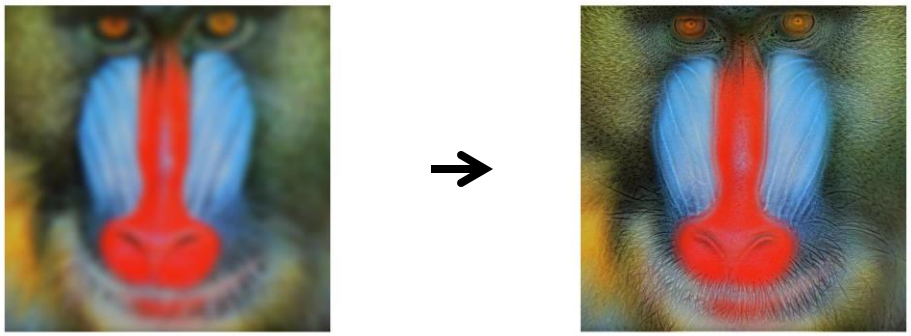
```
linearInterpolation(A8,A,50)
linearExtrapolation(A,A8,1.5,50)
```

Nathan Tung
004-059-195
Discussion 1A

Sample Usage:

```
linearInterpolation(A8,A,50)
```

```
linearExtrapolation(A,A8,1.5,50)
```



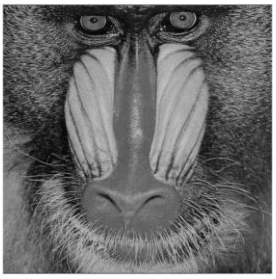
8. **Photoshopping**
    a. **Color-to-Grayscale Transformation**

```
function GrayA = color2grayscale(A)
      [R,G,B] = image2rgb(A);
      GrayA = rgb2image((R+G+B)/3,(R+G+B)/3,(R+G+B)/3);
```

When the components of RGB are equal, the color output is gray, and the image matrix becomes grayscale. In our function, we break down the image into RGB components, then send it back into the GrayA image matrix, but with all the RGB components averaged out and equaled.

Sample Usage:

```
GrayA = color2grayscale(A)
```



Nathan Tung
004-059-195
Discussion 1A

b. **Saturation and Oversaturation**

```
linearInterpolation(GrayA,A,50)
linearExtrapolation(A,GrayA,1.75,50)

BlackA=zeros(480,480,3)
```

Saturation is the intensity or amount of color in some image. More saturation means the image seems "fuller," while less saturation creates a washed-out image. Interpolation between A and GrayA provides a transition in saturation, whereas extrapolation (without normalizing) between the two image matrices causes oversaturation. In order to create a black image BlackA, we just fill a 3D matrix with zeros, as shown in the last command above.

Sample Usage:

```
linearInterpolation(GrayA,A,50) %saturation
```

```
linearExtrapolation(A,GrayA,3,50) %oversaturation
```

```
imshow( (1-t)*A + t*GrayA ) %t=+0.25, t=-0.25, respectively
```

Nathan Tung
004-059-195
Discussion 1A

c. **Brightening**

```
linearInterpolation(A,BlackA,50) %darkens
```

Since BlackA is a 480x480x3 matrix of all zeros, it does not play an important role in interpolation or extrapolation. A simpler way to adjust the brightness of some image matrix A would be to use the following function:

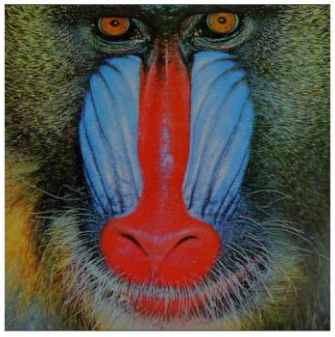
```
function B = brighten(A,T)
      B=(T*A);
      imshow(B);
```

Given an input matrix A and some double, we output a new brightness-adjusted matrix and display the image. When T is less than 1, the image becomes darker. When T is greater than 1, the image becomes brighter. The image becomes black (equivalent to BlackA) when T approaches 0 or below.

Sample Usage:

```
brighten(A,2)
```

```
imshow( (1-t)*A + t*BlackA ) %t=+0.25, t=-0.25, respectively
```



Nathan Tung
004-059-195
Discussion 1A

d. **Darkening**

```
function B = darken(A,T)
    [R,G,B] = image2rgb(A);
    [H,S,V] = rgb2hsv(R,G,B);
    [R,G,B] = hsv2rgb(H,S,T*V);
   B = rgb2image(R,G,B);
   imshow(B);
```

The darken function takes in the image matrix A and a parameter T for adjusting brightness using RGB and HSV systems. First, we convert image A into RGB, which is converted to HSV in turn. HSV, which adjusts hue, saturation, and value, can affect darkness by decreasing the V component. Thus we multiply V by our parameter T before converting it back to RGB and image matrix. Finally we display the generated image.

Sample Usage:

```
darken(A,0.8) %darkens image A by 20%
```