CS170A — Mathematical Modeling & Methods for Computer Science
HW#1 *Photoshop in Matlab*
Due: 11:59pm Monday April 22, 2013

D. Stott Parker, Scott G-H. Tu
stott@cs.ucla.edu, ghtu@cs.ucla.edu

Using CourseWeb, please turn in your Matlab code and resulting output (enough to show that the program is working) as a document, such as a MSWord file or PDF document.

Images are often represented as matrices, and Matlab is a popular tool for analyzing them. In this problem you will develop a few image editing tools with matrix operations. You may want to look up these commands:

```
help imshow
help image
help imagesc
```

1. **Mandrills (10 points)**
   Write a Matlab function `squareSubmatrix(A)` that takes a $p \times q$ matrix `A` as input, and returns a centered $n \times n$ submatrix such that $n = \min(p, q)$.

   The Mandrill image we discussed in class is in the subdirectory `toolbox/matlab/demos/` – load it into Matlab and crop it to a square image using your `squareSubmatrix` function:

   ```
   load('mandrill');
   % this creates X (a Mandrill image) and map (its colormap)
   imshow(X, map)
   Mandrill = ind2rgb( squareSubmatrix(X), map );
   %  this converts the mandrill to a RGB image
   size(Mandrill)
   imshow(Mandrill)
   ```

   This should create and display a $480 \times 480 \times 3$ array `Mandrill`. In the following questions, let the 3D array A be the `Mandrill` image.

2. **Color inversion (10 points)**
   If $c = [rgb]$ is the color of a pixel of A, then $255 - c$ is the *color inversion* of c. Problem: write a function `colorInversion(A)` that yields an image whose colors are inversions of colors in A.

   Color intensities are often `uint8` (8-bit unsigned integer) values, so using the `double()` and `uint8()` conversion functions may be needed to perform arithmetic operations on images. For example, `imshow( uint8(min(255, 1.5 * double( A ))))` multiplies each color intensity in `A` by 1.5. It may help to define define functions to convert between these 3D arrays (like `A`) and 2D matrices (`R`, `G`, and `B`) such as these:

   ```
   function [R,G,B] = image2rgb(A)
            R = double(A(:,:,1))
            G = double(A(:,:,2))
            B = double(A(:,:,3))

   function A = rgb2image(R,G,B)
            A(:,:,1) = uint8(R);
            A(:,:,2) = uint8(G);
            A(:,:,3) = uint8(B);
            %  equivalent:  A = cat(3, uint8(R),uint8(G),uint8(B))
   ```

3. **Color inversion movie (10 points)**
With the matrix A, create a movie (using the Matlab `im2frame` and `movie` functions) of some number $N+1$ frames, showing the *morphing* of image A to its color inversion Z given by

$$(1-t)\,A \;+\; t\,Z \quad \text{for } t \text{ in } [\,0/N,\ 1/N,\ 2/N,\ ...,\ N/N\,] = \texttt{linspace}(0.0, 1.0, \texttt{N}+1).$$

This is *linear interpolation* between A and Z when t is in the interval $[0,1]$.

4. **imagesvd (10 points Extra Credit)**
Get the script `imagesvd.m` from `http://www.mathworks.com/moler/ncmfilelist.html`. (You may also want to get a GUI script called `interpgui.m` since it might be helpful.) Both show how to develop an interactive Matlab script.

Modify these scripts to give an interactive program that allows you to interpolate between two images, using the slider to pick an interpolation point.

5. **Blurring (10 points)**
An image A can be *blurred* by averaging it out a little. If S is the $n \times n$ 'blurring' matrix

$$S \;=\; \frac{1}{4}\begin{pmatrix} 3 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 1 & 3 \end{pmatrix}$$

Notice that if we compute the product $B = S\,A$, each entry b_{ij} in the result is a weighted average of adjacent elements in A, something like $b_{ij} = (a_{i(j-1)} + 2a_{ij} + a_{i(j+1)})/4$.

First, write a Matlab function `Blur(n)` that yields the $n \times n$ blurring matrix S for any $n > 0$. Hint: S is a tridiagonal matrix; check out the `gallery` function.

Problem: let $n = 480$ as with the Mandrill, and let S be the $n \times n$ blur matrix computed by your `Blur(n)` function. What is the least integer k such that no entries of S^k are zero? (Hint: You can answer this simply by looking at the results for a few small values of k.)

6. **Progressive Blurredness (10 points)**
Now let $T = S^8$. Then $T\,A\,T'$ is a blurred version of A — and since $T = T'$, this is the same as $T\,A\,T$. (Conceptually, that is; again $T\,A\,T$ cannot be computed directly with 3D matrix multiplication on A, but instead on its individual RGB component matrices.). $T^2\,A$ is an even more blurred version of A, etc.

Problem: show the result of interpolation and extrapolating between the blurred image $T^4\,A\,T^4$ and the even more blurred version $T^8\,A\,T^8$.

7. **Extrapolation (10 points)**
Extrapolation is just interpolation when t is outside the interval $[0,1]$ — if we also use rescaling to ensure that the result intensity values are all in $[0,1]$. (Matlab requires) all intensity values to be between 0 and 1.)

We can *sharpen* an image A by interpolating or extrapolating between it and a blurred version of A. Show this.

8. **Photoshopping (40 points)**
Write Matlab functions to perform the following transformations on RGB images:

   (a) **color-to-grayscale transformation**
   If `r=g=b`, then the color `[r g b]` will be a shade of gray. This lets us convert a color image `A` to a *grayscale image* `GrayA` by converting all its RGB colors to gray values. For example, `GrayA` could use the same 2D matrix `uint8(sum([r g b])/3)` for its red, green, and blue components.

(b) **saturating (and oversaturating)**
The *saturation* of `A` can be varied by interpolating between `A` and the B+W image `GrayA`. By extrapolating you can produce *oversaturation*.

We can also produce the *black image* `BlackA` as just a 3D zero matrix, since RGB value `[0 0 0]` is black.

(c) **brightening**
The *brightness* of an image `A` can be varied by multiplying `A` by a constant t, or equivalently by interpolating between `A` and the black image `BlackA`. When t is in $[0, 1)$, the brightness is decreased. When $t > 1$, the brightness is increased.

(d) **darkening**
As explained in the notes, the RGB color space defines a cube with vertices `[0 0 0]` (black), `[1 0 0]` (red), `[0 1 0]` (green), `[0 0 1]` (blue), `[0 1 1]` (cyan), `[1 1 0]` (yellow), `[1 0 1]` (magenta), `[1 1 1]` (white). (Thus, literally, red+blue = magenta if we view these as vectors.)

In the HSV color system, these hues define one dimension, and there are two others: saturation and value. We can convert between the HSV and RGB color systems with the Matlab functions `rgb2hsv` and `hsv2rgb`.

The 'value' dimension V in the HSV system is one of darkness. By decreasing the V component in an HSV-color, you can develop a function that makes colors in an image darker.

Write Matlab functions to convert images to grayscale, saturate image colors, brighten images, and darken images. Have each of these functions take a parameter t that normally ranges between 0 and 1, and performs the transformation required on the input image.

Using your functions, display these images when `A = Mandrill` **(10 points each)**:

- `GrayA`
- `(1-t)*A+t*GrayA` at the two points `t`$= +0.25$ and `t`$= -0.25$.
- `(1-t)*A+t*BlackA` at the two points `t`$= +0.25$ and `t`$= -0.25$.
- a 20% darker version of the Mandrill image (using HSV).