CS170A — Mathematical Modeling & Methods for Computer Science — Fall 2014
HW#1 *Linear Algebra*
Due: 11:59pm Sunday October 19, 2014

D. Stott Parker, Luis Angel Larios Cardenas
stott@cs.ucla.edu, langel@cs.ucla.edu

Using CCLE, please upload *a single PDF document* including your answers, proofs, Matlab code, and the resulting output (enough to demonstrate that the program is working).

1. **Visualizing Linear Transformations of the Plane (10 points)**
When a linear transformation $T : \mathbb{R}^n \to \mathbb{R}^n$ is given, it can be identified with a matrix, and this is a way to specify the function. Let $T : \mathbb{R}^n \to \mathbb{R}^n$ be a linear transformation and let $\mathbf{e}_1, \mathbf{e}_2, \ldots \mathbf{e}_n$ denote the columns of the $n \times n$ identity matrix. Figure out what each $T(\mathbf{e}_i)$ should be and write each $T(\mathbf{e}_i)$ as a column vector. If you then define the matrix $A = [T(\mathbf{e}_1)\ T(\mathbf{e}_2)\ \ldots\ T(\mathbf{e}_n)]$, then it will be true that $T(\mathbf{x}) = \mathbf{A}\mathbf{x}$ for all $\mathbf{x}$, and $\mathbf{A}\mathbf{x}$ gives a formula for the function. In other words, given a linear transformation $T : \mathbb{R}^n \to \mathbb{R}^n$, if you know its values only at the n independent vectors $\mathbf{e}_1, \mathbf{e}_2, \ldots \mathbf{e}_n$, then you know its value at every vector $\mathbf{x}$.

   (a) What is the 3×3 matrix transformation A that maps the standard basis vectors as follows?
   A maps $\mathbf{e}_1$ to $\begin{pmatrix} 3 \\ -2 \\ 1 \end{pmatrix}$, $\mathbf{e}_2$ to $\begin{pmatrix} 6 \\ 0 \\ 7 \end{pmatrix}$, and $\mathbf{e}_3$ to $\begin{pmatrix} 5 \\ 4 \\ -1 \end{pmatrix}$.

   (b) Give a 2×2 matrix that maps $\mathbf{e}_1$ to $4\mathbf{e}_2$ and $\mathbf{e}_2$ to $-\mathbf{e}_1$.

   (c) Let $M = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$. Explain why the function $T(\mathbf{x}) = M\mathbf{x}$ maps the x-axis onto the line $y = x$, and why it maps the line $y = 2$ onto the line $y = x + 2$. (Hint: a general point on the x-axis is of the form $\begin{pmatrix} t \\ 0 \end{pmatrix}$; calculate $M \begin{pmatrix} t \\ 0 \end{pmatrix}$ and interpret. Similarly, calculate $M \begin{pmatrix} t \\ 2 \end{pmatrix}$ and interpret.)

   (d) The matrix $A = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}$ represents a transformation called *vertical shear*. It leaves the y-axis fixed and increases the slope of all other lines parallel to the x-axis. Write a Matlab function that takes as input the coordinates of the corners of a square, and transforms these points into a skewed version of the original square. Your function must plot the input square and the skewed square on the same figure so that you can see how A acts on the xy-plane.
   Show how the unit square (whose coordinates are $(0,0), (1,0), (1,1)$, and $(0,1)$) looks when you transform each of its corners.

   (e) Find the transformation matrix S that *stretches* a vector $\mathbf{z}$ by 3 times along the x-axis, and 2 times along the y-axis.
   As in the previous exercise, write a Matlab function that takes as input the four points that define a square, transforms these points, and plots the original square against the transformed square on the same figure. Show your results by applying S to the unit square.

   (f) The matrix $\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$ represents a *rotation*, and $\begin{pmatrix} \cos\theta & \sin\theta \\ \sin\theta & -\cos\theta \end{pmatrix}$ represents a *reflection*. First, plot the reflection of the unit square defined above with angle $\theta = \pi/5$ and $\theta = \pi/3$. As in the previous exercise, write a Matlab function that takes as input the four points that define a square, transforms these points, and plots the original square against the transformed square on the same figure.
   Prove by induction that every product of an even number of reflections is a rotation. Also prove by induction that every product of an odd number of reflections is a reflection.

2. **Matlab (10 points)**
The following functions are already defined in Matlab. Reimplement them as functions by yourself, using only matrix algebra. In other words, you can define functions, but *you cannot use any builtin functions or Matlab features other than vector and matrix operators.* You can get a list of available operators with the command `help ops` in Matlab.

   This may require some cleverness. For example, to define your versions of the builtin function 'max', you could define

```
maximum = @(X,Y)  X .* (X>Y)  +  Y .* (Y>=X)
```

   Implement each of the following functions, assuming that n is a positive integer, X is a $n \times p$ (possibly rectangular) matrix with $n \geq 1$, $p \geq 1$. That is, your function is permitted to start with a command like `[n,p] = size(X)` to obtain the values for n and p.

(a) The *identity matrix* eye(n)

(b) The *absolute value matrix* abs(X)

(c) The *covariance matrix* cov(X)

(d) The *lower-triangular part of a matrix* tril(X)

(e) The *LU-factorization* [L,U] = lu(X). For this function only you are permitted to use *for*-loops; you cannot use builtin functions. (Hint: look at Moler's Matlab code mentioned in the syllabus.)

Note: do *not* implement all the features of the Matlab implementation; only implement the basic function.

3. **Images**

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
```

The Mandrill image is in the subdirectory `toolbox/matlab/demos/` – it is easy to load it into Matlab:

```
load mandrill
% loads in an image and its colormap:
%   X    480x500;  entries are integer color codes
%   map  220x3;    rgb = map(i,:)   gets the RGB triple for code i
imshow(X, map)
Mandrill = ind2rgb( X, map ); %  convert to an RGB image
size(Mandrill)
imshow(Mandrill)
```

The command using `ind2rgb` should create and display a $480 \times 500 \times 3$ array `Mandrill`.

Important: 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. It may help to 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));
```

With `image2rgb` any RGB image is split into 3 matrices, one for each color. These can be transformed mathematically and the converted back with `rgb2image`.

(a) **Image Interpolation and Extrapolation (10 points)**

Develop a Matlab function `interpolate(A, B, t)` that, given two RGB images A and B of the same size, and a value t in the interval $[0, 1]$, computes the *linear interpolation* between A and B as

$$(1 - t)\, A \;+\; t\, B.$$

When $t > 1$, this is known as *extrapolation*.

Then, find a photo of someone you do not like, and crop it and the Mandrill image to the same size. Try to line up the eyes so the two faces are aligned. Name the resulting two images A and B.

Display the result of interpolating to $t = 0.25$. Also display the result of extrapolating to $t = 1.25$.

Then make a movie of some number $N + 1$ frames, showing the *morphing* of image A to image B for each t in $\{\ 0/N,\ 1/N,\ 2/N,\ ...,\ N/N\ \} = \{\ 0,\ 1/N,\ 2/N,\ ...,\ 1\ \}$. Use the Matlab `im2frame` and `movie` functions to do this.

(In the document you upload for this assignment, only include your code for this movie — you are not required to show movie output.)

(b) **Image Editing (20 points)**

Develop Matlab functions to perform the following functions on RGB images:

- `grayscale(A)`: **color-to-grayscale transformation**
  The RGB values of gray colors are `[0 0 0]`, `[1 1 1]`, `[2 2 2]`, ..., `[255 255 255]`. In other words, any gray color has a RGB triple whose entries are all equal.
  We can convert a color image `A` to a *grayscale image* `GrayA` by converting all its RGB colors to gray values. For example, if `A` has components `R`, `G`, and `B`, then `GrayA` could use the same 2D matrix `uint8((R+G+B)/3)` for its red, green, and blue components.
- `saturate(A,t)`: **image saturation (and oversaturation)**
  The *saturation* of an RGB image `A` can be varied by interpolating between `A` and the grayscale image `GrayA`. Extrapolating yields *oversaturation*.
  We can also produce the *black image* `Black` as just a zero 3D matrix, since RGB value `[0 0 0]` is black.
- `brighten(A,t)`: **image 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 `Black`. When t is in $[0, 1)$, the brightness is decreased. When $t > 1$, the brightness is increased.
- `darken(A,t)`: **image darkening**
  The usual 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 the indicated Matlab functions to convert images to grayscale, saturate image colors, brighten images, and darken images. Each of these functions takes 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` **(5 points each)**:

- `grayscale( A )`
- `saturate( A, t )` at the two points `t`$= +0.25$ and `t`$= -0.25$.
- `brighten( A, t )` at the two points `t`$= +0.25$ and `t`$= -0.25$.
- `darken( Mandrill, t )` at `t`$= +0.25$ (using the HSV color system).

(c) **Image Segmentation (20 points)**

Image segmentation is a fundamental technique in image processing and computer vision. Even robust software packages like Adobe Photoshop use image thresholding and segmentation to isolate objects in pictures and literally 'cut' those entites from their background. We can implement thresholding and segmenting objects by purely using Matlab matrix operations.

Consider the image `floppy.jpg` attached to this assignment. Your task is to isolate the red diskette's case from the rest of the objects. In other words, you will create a basic cutting tool by looking at the distribution of pixel colors in the entire image. To achieve this, consider the following steps.

i. Load the `floppy.jpg` image into Matlab and transform it into a grayscale picture by averaging the RGB pixel values in the 3D matrix. Display the grayscale image to show how the red diskette's case looks after removing color.

ii. Explain how can you isolate the pixel intensities that belong to the grayscaled (previously red) diskette's case. (*Hint*: type `help hist` and identify a target range of intensity values).

iii. After you spot your target pixel intensities in the grayscale image, create a "mask". A mask is just a matrix that contains 0s in pixel locations that you will discard and 1s in the places that correspond to the pixels you want to preserve. Plot the mask you just created and check that its pixel values indeed match the same locations of the pixels on the red floppy. This process of creating a mask is known as *thresholding*.

iv. Finally, apply your mask to each of the three color channels of the original image. Imagine that you put the "mask" onto the floppy's RGB matrices, and "erase" the content of pixel locations that lie right "under" the mask's 0s while preserving those pixels that lie "under" the 1s. The resulting image (which now preserves only the red case) should have a solid color of your preference wherever you decided to "erase" with the mask. This process is known as *segmentation*.

For this exercise, please plot the grayscale image, provide a brief explanation of what you did in step 2, plot the mask you obtained, and plot the resulting image with the red case of the third floppy isolated, and with your favorite color as background.

(d) **Color Shifting (10 points)**

Problem: write a function `colorShift(A,T)` where A is a RGB image (like the Mandrill) but the RGB colors of A are transformed by the 3×3 matrix T and offset $\mathbf{v}$. In other words, if the 3D vector $\mathbf{c} = [rgb]$ is the color of a pixel of A, then $T\mathbf{c} + \mathbf{v}$ is the *color shifting* of $\mathbf{c}$. To color shift an entire image, we shift each of its pixels.

One popular color shift uses the 'YC_bC_r' color scheme, defined by

$$T = \begin{pmatrix} 0.29900 & 0.58700 & 0.11400 \\ -0.16874 & -0.33126 & 0.50000 \\ 0.50000 & -0.41869 & -0.08131 \end{pmatrix}, \qquad \begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix} = T \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix}$$

(assuming all RGB color values are integers in the range 0 to 255). As this matrix suggests, the three YC_bC_r dimensions are just weighted combinations of the RGB cone output signals. The luminance value Y is a weighted average of R, G, B. Also C_b measures prevalence of blue (the B coefficient is largest), and C_r measures prevalence of red. When $C_b = C_r = 0$, the result is just the grayscale (luminance) combination of R, G, and B. The eye perceives Y accurately, but is not as accurate with C_b and C_r, so JPEG can compress by averaging C_b and C_r values over small (2×2) neighborhoods in the image.

i. First, implement a function `color_inversion(A)` that takes an $n \times p \times 3$ RGB image A, and returns another RGB image Y of the same size such that, if c is the color of a pixel of A, then $255 - c$ is the color of the corresponding pixel in Y. Display the result of color inversion for `Mandrill`.
In terms of the definition of color shifting above, what is its matrix T and offset vector $\mathbf{v}$?

ii. Next, implement the YCbCr transform as a function `YCbCr(A)` that takes as input an $n \times p \times 3$ RGB image `A`, and returns a $n \times p \times 3$ array of YCbCr values. Assume that all values in the inputs are uint8 values (between 0 and 255), and have your function yield uint8 values also. Show the result of `YCbCr(Mandrill)` using your function.

iii. Prove that all entries in the result $T \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix}$ are always in the range 0 to 255, provided R, G, B are.

4. **Overheating (20 points)**

GHCN is a large matrix of global historical temperature data, from 1880 to 2014 (it is freely available; we downloaded it from `ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/v3/grid/`). A not-very-easy-to-read technical paper describing the dataset is at `http://onlinelibrary.wiley.com/doi/10.1029/2011JD016187/pdf`).

The data stores values for the whole globe using a grid, which has a resolution of $5^\circ \times 5^\circ$. In other words, the grid has 36 rows (for latitude) and 72 columns (for longitude). Although the earth is spherical and a 36×72 rectangular grid introduces distortion, for this problem we will follow this approach.

The GHCN dataset covers $2014 - 1880 + 1 = 135$ years, with 12 months per year. So the data set contains 135×12 grids. The `ghcn.csv` file storing the data is basically a $(36 \times 12 \times 135) \times 72$ matrix of temperature values. It is stored as a 2D matrix of size $(36 * 12 * 135) \times 72$ matrix, along with two initial columns giving the year and month.

The script `ghcn.m` reads in the data file and reshapes it for you into a 4D matrix of size $36 \times 12 \times 135 \times 72$.

Actually, the GHCN data gives 'anomaly' values instead of temperature values. That is, it gives values of how far the temperature was from normal in that grid square in that month. Positive values are above normal; all values are in Celsius. (Why does it give anomaly values only? The paper explains why computing actual temperatures is much harder.)

The point of this dataset in the assignment is that it permits us to use 'slices' to do a *lot* of work. For example, it turns out the coordinates of most of the continental U.S. are the rectangle with rows `(9:12)` and columns `(14:20)`. As a more complex example we can select the anomaly values for the U.S. during years 1900-2013 with:

```
US_latitude  = 9:12
US_longitude = 15:20
my_years = 1900:2013
my_slice = temperature_anomaly( US_latitude, :, my_years - 1880, US_longitude )
```

To obtain the average value for this area for each year, since it turns out there are no missing values:

```
total_number_of_grid_squares = length(US_latitude) * length(US_longitude) * 12;
N = total_number_of_grid_squares;
average_US_anomaly_by_year = reshape( sum(sum(sum( my_slice, 4),2),1) / N, [length(my_years) 1] );
```

(a) Plot the average temperature anomaly for the United States, using the `US_latitude` and `US_longitude` values above, for the month of July in every year from 1900 to 2013 (the average should cover all 12 months).

(b) Do the same thing for the whole planet: using the `missing_values` information in `ghcn.m`, plot the average (non-missing-value) temperature anomaly over the entire grid, for every year from 1914 to 2013.

(c) Using a for-loop, make a movie of the anomaly for the month of July in every year from 1880 to 2014, showing how it has evolved. (Use `colormap(hot)`, and to display higher temperatures with greater colormap resolution, change all values below -5 to -5. In the document you turn in for this assignment, show the result for July 2010.)