CS170A — Mathematical Modeling & Methods for Computer Science — Fall 2016
HW#0 *Warmup*
Due: 11:55pm Tuesday October 11, 2016

D. Stott Parker, Xin Min, Xiangning Lyu, Arul Jeyaraj

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. **Images (30 points)**
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.

The Mandrill image file is included in this assignment, in both '.mat' and '.csv' formats. In Matlab you can load it with:

```
load mandrill                                                   %  An alternative approach, using csv files:
% loads in an image and its colormap:
%  X    480x500; entries are integer color codes                   X = csvread('mandrill_X.csv')
%  map  220x3;   rgb = map(i,:) gets the RGB triple for code i      map = csvread('mandrill_map.csv')

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`.

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

(a) `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.
Show the definition of your function `grayscale` and also its result for the `Mandrill` image.

(b) `saturate(A,t)`: **image saturation (and oversaturation)**
Given two RGB images A and B of the same size, and a value t in $[0, 1]$, the *linear interpolation* between A and B is

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

The *saturation* of an RGB image `A` can be varied by interpolating between `A` and the grayscale image `GrayA`. When $t > 1$, this is known as *extrapolation*; 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.
Show the definition of your function `saturate` and also its result for the `Mandrill` image when $t = 0.25$.

(c) `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.
Show the definition of your function `brighten` and also its result for the `Mandrill` image when $t = 0.25$.

2. **Color Models (30 points)**
In the attached files is a function `RGB_to_YCbCr(R,G,B)` that converts RGB values to another color model called YCbCr. It is defined by

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

where all RGB color values are integers in the range 0 to 255.

   (a) Prove that entries in the result of the `RGB_to_YCbCr(R,G,B)` are all in the range 0 to 255 (provided `R, G, B` are).
   (b) Develop a similar kind of function `RGB_to_CMY()` for converting RGB values to CMY values.
   (c) Show your result of `RGB_to_CMY()` for the `Mandrill` image by rendering it in RGB.
   (Please display the image in RGB — with Cyan as Red, Magenta as Green, Yellow as Blue.)

3. **Rotations (20 points)**
The **Euler angles** $(\phi,\ \theta,\ \psi)$ define a 3-dimensional rotation as a product of three 2-dimensional rotations:

$$R_{123}(\phi,\ \theta,\ \psi) \;=\; \underbrace{\begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{R_{12}(\psi)} \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}}_{R_{23}(\theta)} \underbrace{\begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{R_{12}(\phi)}$$

This rotates first in dimensions 1 and 2 xy plane (around the z axis), then in 2 and 3 (around the x axis), and finally in 1 and 2 again. Any rotation of 3-space can be put in this form.

The file `rotations_and_reflections.m` produces some 2x2 matrices, and shows how to define symbolic variables like 'theta'. Using symbolic values, find a 3x3 matrix for $R_{123}(\phi,\ \theta,\ \psi)$. (Hint: `blkdiag` might help.)

4. **Slices (20 points)**
GHCN is a large matrix of global historical temperature data, from 1880 to 2016 (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 $5° \times 5°$. 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 $2016 - 1880 + 1 = 137$ years, with 12 months per year. So the data set contains 137×12 grids. The `ghcn.csv` file storing the data is basically a $(36 \times 12 \times 137) \times 72$ matrix of temperature values. It is stored as a 2D matrix of size $(36 \times 12 \times 137) \times 72$ matrix, along with two initial columns giving the year and month.

The script `ghcn_script.m` reads in the data file and reshapes it for you into a 4D matrix of size $36 \times 12 \times 137 \times 72$.

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 the last century (years 1916-2015) with:

```
US_latitude  = 9:12
US_longitude = 15:20
my_years = 1916:2015
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) The `ghcn_script.m` script plots the average temperature anomaly for the United States, using the `US_latitude` and `US_longitude` values above, but for every year from 1916 to 2015 (the average covers all 12 months).

   Your problem: do the same thing for the whole planet: using the `missing_values` information in `ghcn_script.m`, plot the average (non-missing-value) temperature anomaly over the entire grid, for every year from 1916 to 2015.
   (b) Based on your plot, give your opinion on this question: *is 'global warming' real?*

To get more perspective on the data, you can watch the data as a movie: the script `ghcn_movie_demo.m` makes a movie of the anomaly for the month of July in every year from 1880 to 2016, showing how it has evolved.