CS170A — Mathematical Modeling & Methods for Computer Science — Winter 2016
HW#3 and HW#4 *Images and Audio*
Due: 11:55 pm Sunday February 28, 2016

D. Stott Parker, Yun Lu, Yunzhong He, Luis Angel Larios-Cardenas

This is a *double* homework assignment – your score will count for *both HW#3 and HW#4*.

Using CCLE, please upload a PDF document with your work (including programs and their output).

1. **Eigenfaces (25 points)** Chapter 11 of the course notes is on Eigenfaces. For this assignment you can use the files in the directory `faces`, which includes some Matlab scripts and a database of 177 face images, each a `.bmp` bitmap file of size 64×64 pixels. The face images have been pre-processed so that the background and hair are removed and the faces have similar lighting conditions.

   In the `eigenfaces` program (available in this assignment), some of the faces are classified as 'Evil' while the others are classified as 'Good'.

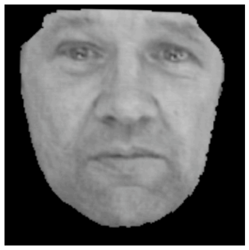
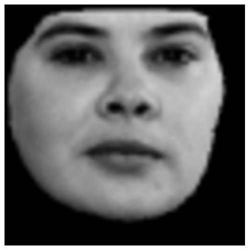

Figure 1: Sample 64×64 face images – of a Good face and an Evil face

   As explained in the notes, if we reshape each face image as a $1 \times 64^2 = 1 \times 4096$ row vector, and collect them into a matrix, the principal components of the matrix define the main dimensions of variance in the faces. These principal components are called *eigenfaces*. The `eigenfaces` program shows how to do this.

   Please perform the following steps:

   (a) Modify the `eigenfaces` program so as to create a function that takes as input a string array of filenames of face images, an integer k, and an integer sample size s — and yields the average face and the first k singular values and eigenfaces as output values for a sample of size s.

   (b) Use your function and the information in `face_descriptions` file (and omitting the images it suggests), to compute the first $k = 30$ eigenfaces for: (1) Good faces, (2) Evil faces.
   Print the top k singular values for each.

   (c) Render (using `imshow`) the average Good face $\overline{\mathbf{f}}_0$ and the average Evil face $\overline{\mathbf{f}}_1$.

   (d) Also render $|\ \overline{\mathbf{f}}_0 - \overline{\mathbf{f}}_1\ |$, the absolute value of the difference between the average Good and Evil face.

   (e) Use your eigenfaces to determine whether there are any pairs of images containing very similar faces. (In other words, look for people whose image occurs twice.)
   If you find any such pairs, display them; if not, state that you found none.

   (f) Develop a Good/Evil face classifier using the eigenfaces you've computed: write a function that, when given as input the filename of a face image, yields the output value 0 if the face is classified as Good, and 1 if the face is classified as Evil. You can implement this any way you like.
   (For example, in order to decide whether a face $\mathbf{f}$ is Evil, you might measure the 'distances' from $\mathbf{f}$ to the average Evil face $\overline{\mathbf{f}}_0$ and the average Good face $\overline{\mathbf{f}}_1$, and classify accordingly.)
   Print the classifications your classifier produces for all Evil faces. (Determine for each of the Evil images whether your classifier agrees with its Good/Evil description in the `face_descriptions` file.)

2. **Finding Harmony (25 points)**
With a function like `audioread`, read in `tune.mp3` and obtain the audio tracks for the file. In Matlab:

```
audiofile = 'tune.mp3';
[tune, Fs] = audioread(audiofile);
y = tune(:,1);  % first track of the stereo clip
```

After this, executing `sound(tune, Fs)` will play the sound track at the sampling frequency `Fs`. There are several questions below about `y`, which is the first of the 2 stereo audio tracks.

(a) What is `Fs`? How does this compare with CD-quality sound?

(b) What is the length of `y`? Find the integer factors of n by using a function like `factor`.

(c) Using a function like `profile`, determine how much cpu time it takes to compute `fft(y);`

(d) For this assignment we define the *power* of a complex value z to be its complex absolute value: $\text{power}(z) = |z| = \sqrt{z\,\bar{z}}$.
The *frequency spectrum* is a sequence of frequency values that has the same length n as the power values. The lowest frequency value is 0, and the highest frequency value is `Fs`. The frequency spectrum for `y` should be a linearly spaced between these two limits.
Plot the power of the Fourier transform, using the frequency spectrum values for the x-coordinate and power values for the y-coordinate. Your plot must show correctly how the frequency axis is scaled.

(e) Do the same plot, but only for the 'first half' of the power (since the second half is a kind of mirror image when `y` is real). Here by *'first half'* of a sequence $s = [s_0, \ldots, s_{n-1}]$ we mean $[s_1, \ldots, s_m]$ where $m = \lfloor n/2 \rfloor$. For example, the 'first half' of $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$ is $[1, 2, 3, 4, 5]$.
The highest frequency of this half is called the *Nyquist frequency.* What is its value?

(f) Write a function `[spike_freq,spike_power] = top_spike(frequency_values,power_values)` that finds the highest value in a music power spectrum `power_values` from the frequency range specified by `frequency_values`. (Ignore spikes from frequencies below 100Hz.) Using this function, by picking four appropriate intervals of frequencies (excluding zero), find the frequencies of the 'top 4 spikes' – i.e., four frequencies at which the power spectrum has the highest power values.
(Hint: if you are confused about how to convert subscripts to frequency values, the Matlab `sunspots` demo may help; try `playshow sunspots`.)

(g) Harmony is generally produced by adding both a frequency (like 264 Hz, the note C) with the same frequency times the factor 5/4 (like 330 Hz, the note E).
The **piano scale** assumes that the k-th note in the scale has frequency c^k times the frequency of C, where $c = 2^{1/12} = 1.059463\ldots$ and all frequencies of C are powers of two as in the table below.

note	frequency (Hz)	frequency/C	interval
Middle C	256.0	$c^0 = 1.000$	
C#	271.2	$c^1 = 1.059$	minor 2nd
D	287.3	$c^2 = 1.122$	major 2nd
D#	304.4	$c^3 = 1.189$	minor 3rd
E	322.5	$c^4 = 1.260$	major 3rd
F	341.7	$c^5 = 1.334$	4th
F#	362.0	$c^6 = 1.414$	diminished 5th
G	383.6	$c^7 = 1.498$	5th
G#	406.4	$c^8 = 1.587$	minor 6th
A	430.5	$c^9 = 1.682$	major 6th
A#	456.1	$c^{10} = 1.782$	minor 7th
B	483.3	$c^{11} = 1.888$	major 7th
High C	512.0	$c^{12} = 2.000$	octave/8th

In this scale C is 256 Hz, and E is 322 Hz; notice that 5/4 (256 Hz) = (320 Hz), very close to E.
For `tune.mp3`: determine its key (the note nearest its top spike), and whether it has harmony.

(h) Do this also for `untune.mp3`: does it have a top spike that stands out? does it have harmony?

3. **De-noising Images with the FFT (25 points)**

So far, we have worked with images on the *spatial domain*, which allows to manipulate pixel intensity values directly and compress the images by using SVD. However, we can also view any picture like a composition of frequencies that live in a two-dimensional realm called *frequency domain.* Like in music analysis, the Fourier Transform is a very useful tool for processing and/or filtering images by extracting their bi-dimensional power spectrum.

In this problem we will use the Fourier Transform to de-noise an image. If we are able to spot the frequencies that make up the undesirable noise, it's possible to remove these frequencies from the image power spectrum and use the *Inverse FFT* to put the picture back together.

(a) In the homework `*.zip` file you'll find the 500-by-500-pixel grayscale image `taeyeon.bmp`, which we deliberately corrupted with a patterned, anoying noise. Load it into your working environment (e.g. using `imread()` in `Matlab`. (If you to use `taeyeon.jpg`, take the mean of the RGB values to transform it into a grayscale, 500×500 matrix of `double` values.)

(b) Apply the bi-dimensional FFT, `fft2()`, to the grayscale image. This transform will identify image frequencies in both x and y directions and will yield a complex-valued power spectrum, Z, that is symmetric with respect to the coordinate axes. In order to better analyze the frequencies in the power spectrum, compute the *logarithm* of the *absolute value* of Z:

$$P = \log |Z|$$

Plot P and make sure its values are properly scaled to the range of [0,1].

(c) From your previous plot, you can verify the power spectrum is made of quadrants. The four bright spots seen on the corners correspond to the lowest frequencies in the image. Next, you'll move the quadrants around, so that the (corner) low frequencies will lie in the center of the frequency square. Use the function `fftshift()` to swap the quadrants in **both** P, and Z:

$$Z_c = \text{fftshift}(Z)$$
$$P_c = \text{fftshift}(P)$$

`fftshift()` operates as follows: if `a = [1 2; 3 4]`, then `fftshift(a) = [4 3; 2 1]`. Show your plot for P_c —the centered log-absolute-valued power spectrum of the grayscale image.

(d) In the plot for P_c, you may notice *two bright bands* running from left to right, parallel to the x axis. They partly correspond to undesired noise in the original image. The first of these bands (the one above the "equator") has its center pixel at **row number 187** in the P_c and Z_c matrices. The *mirror* band is at the same latitude but from the bottom up.
Now, for each of these two bands, **zero out** (i.e. put zeros in) their entries in P_c and Z_c. You may set the band height to 20 pixels.
Likewise, go further and neutralize (e.g. zero out) the bright stain whose center is at **row 22 and column 234** of matrices P_c and Z_c. Since the spectrum is symmetric, do not forget to neutralize the "twin" bright stain that lives in the opposite side of the square. Again, you may have a coverage of 10 pixels around the bright stain center pixel.
After these operations, show your plot for the modified P_c.

(e) Finally, reorder the frequency square quadrants by using `ifftshift()` on Z_c:

$$Z_{new} = \text{ifftshift}(Z_c)$$

Here, if `a = [4 3; 2 1]`, then `ifftshift(a) = [1 2; 3 4]`. Since Z_{new} is our *de-noised*, complex-valued power spectrum, we can use the *Inverse Bi-dimensional FFT* to put the improved version of the original image back together:

$$I = \text{ifft2}(Z_{new})$$

Plot I next to the grayscale version of `taeyeon.jpg`. Do not forget to scale I's pixel intensity values to the range of [0,1]. Was there any improvement or noise removal at all?

4. **A Simple 'Fake Photo' Detector (15 points)**
A simple way to check if a $m \times n$ RGB image has been faked (with Photoshop, say) is to use the `reshape` function to convert it into a $(m\,n) \times 3$ matrix, compute the 3 principal components of its 3×3 covariance matrix, project the reshaped image on each of the PCs, and then reshape each projection result back to a $m \times n$ grayscale image. Then display all three of the resulting principal component images.

   The idea is that if any of these grayscale images has bright or dark spots, it is possible that the color distribution in that area (and perhaps also that part of the image) has been altered.

   For example, Figure 2 shows the 2nd PC image for a famous Moon landing photo that was later admitted to be edited — notice that some areas of the 2nd PC image are relatively bright or dark.
(Some people believed the whole Moon landing was faked because they thought this image was edited.)

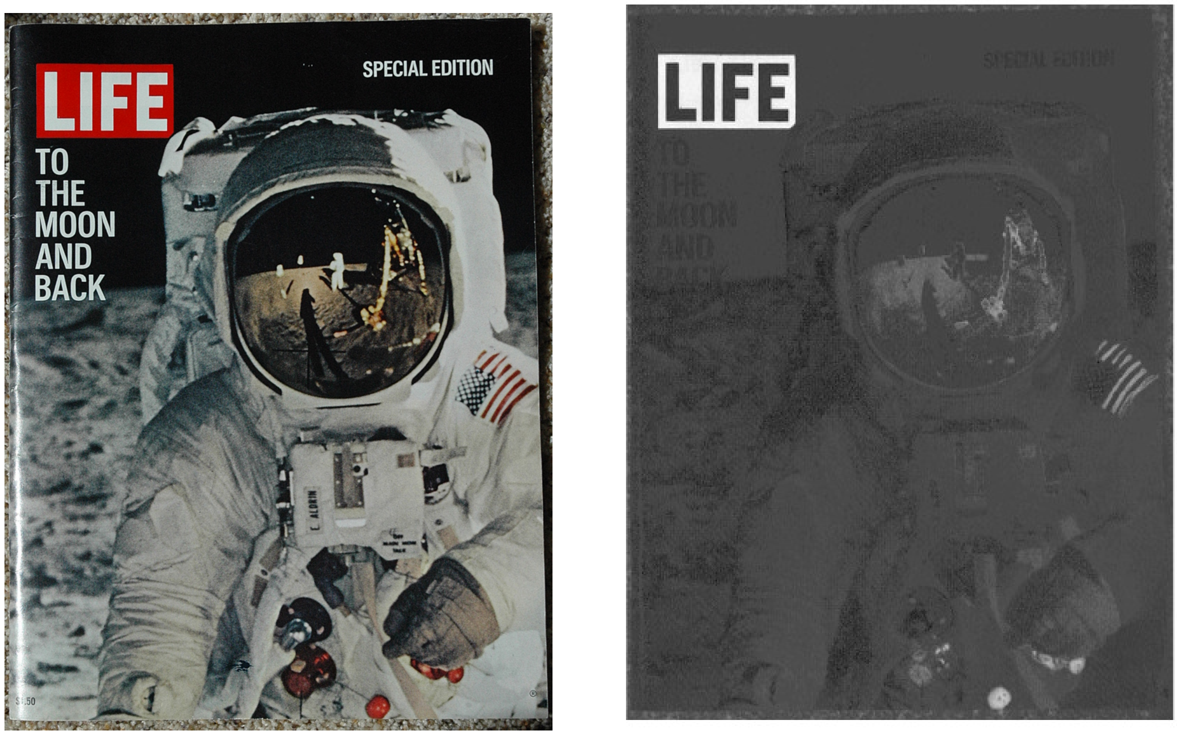

Figure 2: LIFE magazine's famous Moon landing photo, along with a grayscale image showing its 2nd PC. Because values near zero are rendered as black, the smallest values are dark, while large values are bright. Unusually dark and unusually bright areas ('LIFE', flag stripes, red buttons, visor reflection) are suspicious. It looks in this case as though the 2nd PC emphasizes the color red and neglects white.

   (a) Write a program to analyze any image file. Show the three PC images for the famous image `eclipse.jpg` (a Solar Eclipse viewed from the International Space Station). Is it fake?
   (b) Show the results on the famous image `shark_attack.jpg` (known as 'Helicopter Shark'). Is it fake?
   (c) Some people have claimed the 2nd PC image is the most important, since the 1st emphasizes overall averages whereas edits are often local. www.hackerfactor.com/papers/bh-usa-07-krawetz-wp.pdf argues however that we can detect fake images by looking at the projections onto the *1st* and *3rd* PC. It claims furthermore that projecting onto the 3rd PC tells us the 'save' history of chunks of pixels (because of the nature of JPEG, which divides the image into 8×8-pixel blocks and compresses them independently of each other). Identifying blocks with different save histories hints at image insertion. At factually.gizmodo.com/76-viral-images-from-2015-that-were-totally-fake-1747317711 there is an image of flooding in Texas froom 2009, and an image of a raccoon carrying a little cat. Show the result of your fake photo detector on both of these, and evaluate the claim of the hackerfactor document about the 1st and 3rd PC. *Is either the 1st or 3rd PC a better indicator than the 2nd PC?*

5. **Optional Possible Course Project:** this is a very simple fake photo detector. Can you improve it?

6. **Randomness (10 points)**

(a) Consider the triangular **pdf**

$$f(x) \quad = \quad \begin{cases} (x-2) & 2 \le x \le 3 \\ (4-x) & 3 < x \le 4 \\ 0 & \text{otherwise.} \end{cases}$$

Implement a random generator for this distribution *as the sum of two uniform random variables.* Your generator should yield n random values when given n as input.

Use your generator to estimate the <u>variance</u> of the distribution, by computing the average of 10,000 generated values.

(b) Give a formula for the **cdf** $F(x)$ that corresponds to the triangular pdf $f(x)$.
Hint: you can use symbolic algebra tools to solve this. For example, if we define the pdf

$$h(x) \quad = \quad \begin{cases} x & 0 \le x \text{ and } x < 1 \\ 2-x & 1 \le x \text{ and } x < 2 \\ 0 & \textit{otherwise} \end{cases}$$

then by executing the commands `syms x; int(f,x)` in Matlab we can compute its cdf:

$$H(x) \quad = \quad \begin{cases} 0 & x < 0 \\ \frac{1}{2}\,x^2 & 0 \le x \text{ and } x < 1 \\ 2\,x - \frac{1}{2}\,x^2 - 1 & 1 \le x \text{ and } x < 2 \\ 1 & 2 \le x \end{cases}$$

(c) Section 17.13 in the Course Notes, in the chapter about the Central Limit Theorem, describes an efficient way to generate random values from *any distribution*, provided we also have the **icdf** (inverse cumulative density function) for that distribution. The icdf is just the functional inverse of the cdf.

(The same idea is used for **histogram matching** — giving a translation of values in any histogram to another histogram. Histogram matching is important for matching color distributions in images.)

Implement the random generator for the triangular pdf f using this approach.

Again: use your generator to estimate the mean of the distribution, by computing the average of 10,000 generated values.

(d) A random variable ξ is called **lognormal** if $\log(\xi)$ is normal. Develop a random generator for lognormal random variables (logs of the values should have mean μ and variance σ^2),

(e) Generate 10,000 random normal 50×50 matrices X_i, for $1 \le i \le 10,000$ (with mean 0 and variance 1), and compute the singular values of each matrix.

Make an **'average scree plot'** for these matrices, showing $\overline{\sigma_j}$, the average j-th singular value (for $1 \le j \le 50$) over the 10,000 matrices. (Thus the x-axis gives j values, and the y-axis gives $\overline{\sigma_j}$ values.)

Describe how $\overline{\sigma_j}$ decreases as j increases — does the plot look closest to: exponential, polynomial, or linear?

(f) Using the singular values you just conputed from the matrices, plot the histogram of the 10,000 σ_1 values (the largest singular value of each matrix).

Does the shape of the histogram looks closest to: a uniform, normal, or lognormal distribution?

Note: to check if the distribution of x looks lognormal, check if the distribution of $\log(x)$ looks normal.