Advanced Homework

To earn the grade without passing final oral examination you need to solve and explain at least 80% of the problems. Any attempts of cheating will be noted and will negatively affect your final grade including loosing the right for exam-free grade.

1. **Fitting a planet's orbit**.
   A planet follows an elliptical orbit, which can be represented in a Cartesian (x, y) coordinate system by the equation

   $$b_0 + b_1 x + b_2 y + b_3 xy + b_4 y^2 = x^2$$

   (a) Use a linear least-squares fit for the parameters b_0, b_1, b_2, b_3, b_4 , given the following observations of the planet's position: The rest of the

x	1.02	0.95	0.87	...
y	0.39	0.32	0.27	...

   data is in the file `problem1/q1a_data.txt`. Plot the data as points and the resulting continuous curve. What is the norm of residual?

   (b) The observation data is nearly rank-deficient, which implies that the matrix $A^T A$ is nearly singular and hence the parameter fit will be sensitive to perturbations in the data (i.e. the least-squares fit is poorly conditioned in this case). To show this, compute the best fit to the perturbed data $\hat{x} = x + \Delta x$ and $\hat{y} = y + \Delta y$, where $\Delta x, \Delta y$ are given in the file `problem1/q1b_data.txt`. Overlay the two sets of observations and corresponding orbits on the same plot.

   Use `numpy.loadtxt` to get data from text files.

2. **Find your circle**.
   In the directory for the problem you will find bunch of images (numbered 2- 5) with CD-discs on them, just like on the figure.

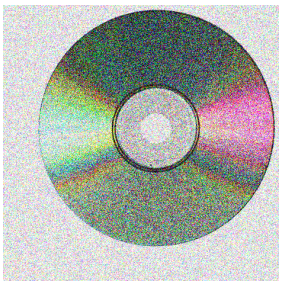
   You need to read the grayscale image and perform edge detection:

   (a) Calculate the gradient I_x and I_y

   (b) Caculate the magnitude of the gradient $I_x^2 + I_y^2$

   (c) Perform thresholding so that roughly 10-20% of points left

   (d) Binarized version (above threshold assigned to 1, below - to 0) is the image we are going to process



We can assume that there are two generative models contributing to this image: one model is a circle parametrized by coordinates of the center x_c, y_c and it's radius r. You can assume that distances of the points from the center are distributed as $e^{\frac{-d^2}{\sigma}}$ where d is the distance amd σ is the variance. Another model that contributes to the image is random noise uniformly distributed with probability of a point being a noise is 10%.

Use EM-algorithm to fit the circle. You can use initial values for $\sigma = 128$ and for the circle radius $r = 190$, and center of the image as the initial center of the circle. Plot separately the image with the predicted circle on top of it and point assignment map (black-and-white image with value of the pixel proportional to probability of lying on the circle). Are there any failed fits? If yes, what is the reason for failure?

You might find the results (algebraic algorithm) from the paper *Gander W., Golub G., Strebe R.* ***Least-Squares Fitting of Circles and Ellipses*** useful. Copy of the paper is in the problem data directory.

3. **Auto-regressive time series prediction.**
Suppose that x is an N-vector representing time series data. The (one step ahead) prediction problem is to guess x_{t+1} , based on $x_1, \ldots, x_t$. We will base our prediction $\hat{x}_{t+1}$ of x_{t+1} on the previous M values, $x_t, x_t - 1, \ldots, x_{t-M+1}$. (The number M is called the *memory length* of our predictor). When the prediction is a linear function,

$$\hat{x}_{t+1} = \beta_1 x_t + \beta_2 x_{t-1} + \cdots + \beta_M x_{t-M+1}$$

it is called an auto-regressive predictor. (It is possible to add an offset to $\hat{x}_{t+1}$, but we will leave it out for simplicity.) Of course we can only use our auto-regressive predictor for $M \leq t \leq N-1$. Some very simple and natural predictors have this form. One example is the predictor $\hat{x}_{t+1} = x_t$, which guesses that the next value is the same as the current one. Another one is $\hat{x}_{t+1} = x_t + (x_t - x_{t-1})$, which guesses what x_{t+1} is by extrapolating a line that passes through x_t and x_{t-1}.
We judge a predictor (i.e., the choice of coefficients β_i) by the mean-square prediction error

$$J = \frac{1}{N-M} \sum_{t=M}^{N-1} (\hat{x}_{t+1} - x_{t+1})^2$$

A sophisticated choice of the coefficients β_i is the one that minimizes J. We will call this the least-squares auto-regressive predictor.

(a) Find the matrix A and the vector b for which $J = \frac{||A\beta - b||^2}{N-M}$. This allows you to find the coefficients that minimize J, i.e., the auto-regressive predictor that minimizes the mean-square prediction error on the given time series. Be sure to check the dimensions of A and b.

(b) For $M = 2, \ldots, 12$, find the coefficients that minimize the mean-square prediction error on the time series **x_train** given in time_series_data.npz. The same file has a second time series **x_test** that you can use to test or validate your predictor on. Give the values of the mean-square error on the train and test series for each value of M. What is a good choice of M? Also find J for the two simple predictors described above.

Hint. Be sure to use the **scipy.linalg.toeplitz** function, it'll make your life easier.

4. **Image reconstruction using low light**.
In the homework files, you will find a directory called `problem4/objects` that contains several photos of a still-life scene with different objects. One is a regular photo and the other three are low-light photos that were illuminated in different colors.
Each pixel in the image can be represented as a three-component vector $p = (R, G, B)$ for the red, green, and blue components. Let p_k^A be the kth pixel of the regular photo, and let p_k^B, p_k^C, and p_k^D be the kth pixel of the three low-light photos. Here, k is indexed from 0 up to $MN - 1$.

(a) Consider reconstructing the regular photo from the three low-light photos. The regular photo pixel could be obtained from the low-light photo pixels using

$$p_k^A = F^B p_k^B + F^C p_k^C + F^D p_k^D + p_{\text{const}} \tag{1}$$

where F^B, F^C, F^D are 3×3 matrices and p_{const} is a vector. Write a program to find the least-squares fit for the 30 components of all the parameters (three matrices and a vector). Specifically, your program should minimize the objective function:

$$S = \frac{1}{MN} \sum_{k=0}^{MN-1} || - p_k^A + F^B p_k^B + F^C p_k^C + F^D p_k^D + p_{\text{const}}||^2 \tag{2}$$

Calculate S for the fitted values F^B, F^C, F^D, and p_{const}. Reconstruct a regular image using the pixel values given by 1. Compare it to the original regular image.

(b) Apply obtained parameters to reconstruct image from `problem4/bears`. Compare the result quantitatively with the original regular image (find mean squared error over all pixels).

5. **Going down the hill**
Define a noisy Rosenbrock function:

$$f(x, y) = R(x, y) + \epsilon(x, y) = (a - x)^2 + b(y - x^2)^2 + \epsilon(x, y)$$

Set the following constants: $a = 1, b = 100$.
Calculate noise at every point as normally distributed with zero mean and standard deviation of 0.001 of the real value at this point

$$\epsilon(x, y) \sim \mathcal{N}(0, 0.001R(x, y)) \ .$$

Implement Nelder–Mead method[1] and any gradient-based method of your choice. Optimize the function defined above with starting point at $(2, -1)$ and set convergence tolerance at 10^{-10}. Compare two methods: final result, number of steps, number of function calls. Implement two versions of the gradient method: one with analytically obtained gradient, another with gradient calculated numerically, compare with Nelder-Mead method. Visualize the process (not in real-time, just plot level lines and points of the optimization process).

6. **The Good, the Bad, and the Ugly**
Dealing with noisy annotations is a common problem in computer vision, especially when using crowd- sourcing tools, like Amazon's Mechanical Turk. For this problem, you've collected photo aesthetic ratings for 150 images. Each image is labeled 5 times by a total of 25 annotators (each annotator provided 30 labels). Each label consists of a continuous score from 0 (unattractive) to 10 (attractive). The problem is that some users do not understand instructions or are trying to get paid without attending to the image. These "bad" annotators assign a label uniformly at random from 0 to 10. Other "good" annotators assign a label to the i-th image with mean μ_i and standard deviation σ (σ is the same for all images). Your goal is to solve for the most likely image scores and to figure out which annotators are trying to cheat you.
Notation:

   - $x_{ij} \in [0; 10]$: the score for i-th image from the j-th annotator
   - $m_j \in 0, 1$: whether each j-th annotator is "good" ($m_j = 1$) or "bad" ($m_j = 0$)
   - $P(x_{ij}|m_j = 0) = \frac{1}{10}$: iniform distribution of scores from bad annotators
   - $P(x_{ij}|m_j = 1; \mu_i, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{1}{2}\frac{(x_{ij}-\mu_i)^2}{\sigma^2})$: normal distribution for good annotators
   - $P(m_j; \beta) = \beta$: prior probability for being a good annotator

   You are given a data file `problem3/annotation_data.mat` containing

   - `annotator_ids(i)` provides the ID of the annotator for the i-th annotation entry
   - `image_ids(i)` provides the ID of the image for the i-th annotation entry
   - `annotation_scores(i)` provides i-th annotation

[1] https://en.wikipedia.org/wiki/Nelder-Mead_method

Derive and implement EM algorithm to solve for all variables. You will need to initialize the probability that each annotator is good (set initial value to 0.5 to each annotator).

Report the indices of "bad" annotators (for which m_j is most likely 0), estimated value of σ, plot the estimated means μ_i of the image scores.

Notes:

- Useful function: `scipy.stats.norm.pdf`
- Be aware of numerical issues when calculating the joint probabilities, which could be close to zero. Instead you might want to use logarithmic scale if needed.

7. **Why so serious?**
   You are given two set of frames (sampled at 30fps) with still human heads. Your task is to estimate the pulse of the person. You may use the algorithm described in the paper by Poh, Ming-Zher, Daniel J. McDuff, and Rosalind W. Picard (provided in the problem directory) with necessary adaptation (due to difference in data).

8. **Going down the hill-2**
   Implement Conjugate Gradient Descent method and Coordinate Descent method and use both to find the minimum of Matyas function[2]. Start at one of the corners of the recommended input domain. Compare performance for all four corners.

9. **What happens in Monte-Carlo stays in Monte-Carlo**
   In class practice we used simulation to compute the integral using Monte-Carlo methid with uniform sampling. Now, suppose, we need to evaluate the following complex integral:

$$I = \int_0^\infty \frac{x^4 \sin(\sqrt{\ln(x+1)})e^{-x}}{2+(x-4)^2} dx$$

   A clever way to apply Monte-Carlo technique would be to split the integrand as $h(x)f_X(x)$, where $f_X(x)$ would represent a pdf of some random variable. In that case we could sample X_i from $f_X(x)$ and calculate $I \approx \frac{1}{N}\sum_{i=1}^{N} h(X_i)$

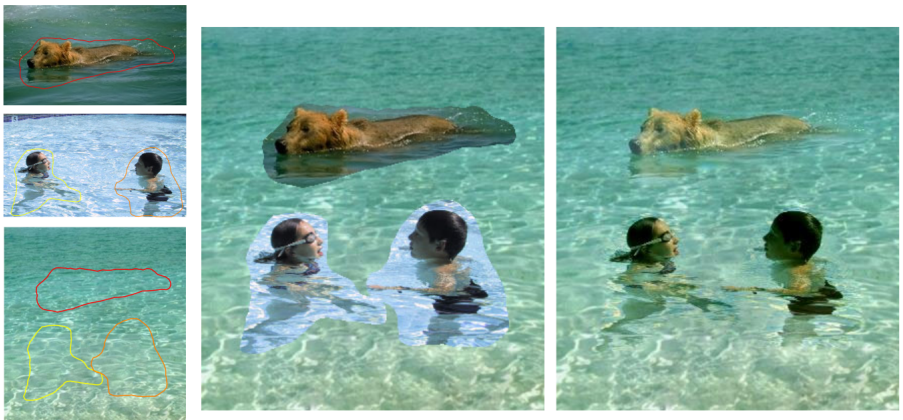
   - Use $\frac{1}{2+(x-4)^2}$ to create your pdf f_X. Implement Metropolis algorithm to sample from f_X. Run the simulation 50 times for 150,000 points. Report the value of $\hat{I}$ and of that $Var[\hat{I}]$
   - Use xe^{-x} to create your pdf f_X. Implement Metropolis algorithm to sample from f_X. Run the simulation 50 times for 150,000 points. Report the value of $\hat{I}$ and of that $Var[\hat{I}]$.

10. **Too much noise in here**
    Image `noisy.jpg` is obtained from the original image by adding random

[2]https://www.sfu.ca/ ssurjano/matya.html

Gaussian noise with variance 100 (on the scale 0-255). Use Markov Random Fields to restore the image. Report MSE and SSIM with the original image.
Useful function: `skimage.measure.compare_ssim`.

11. **Blend with the crowd**
In the paper "Poisson Image Editing" by Perez et al. (copy is in the data directory) authors suggest a method for seamless images blending. The idea is to cut out a piece of one image, paste it into another image and make it look smooth. For every sample we have a source image, a



destination image, and a mask. Even though the method is coming from solving a Poisson equation (second order PDE with boundary condition), it all boils down to solving a linear system of equations $Ax = b$ where A is a matrix representing coefficients of discretized PDE, x is the blended image we seek, and b is the vector containing two types of values:

(a) For rows of A which correspond to pixels that are not under the mask, b will simply contain the already known value from 'target' and the row of A will be a row of an identity matrix. Basically, this is our system of equations saying "do nothing for the pixels we already know".

(b) For rows of A which correspond to pixels under the mask, we will specify that the gradient (actually the discrete Laplacian) in the output should equal the gradient in 'source', according to the final equation in the webpage:

$$4x_{i,j} - x_{i-1,j} - x_{i+1,j} - x_{i,j-1} - x_{i,j+1} = 4s_{i,j} - s_{i-1,j} - s_{i+1,j} - s_{i,j-1} - s_{i,j+1}$$

The right hand side are measurements from the source image. The left hand side relates different (mostly) unknown pixels in the output image. At a high level, for these rows in our system of equations we are saying "For this pixel, I don't know its value, but I know that its value relative to its neighbors should be the same as it was in the source image".

In the data directory you have a set of images with corresponding masks, text file specifying offsets (how much to translate the 'source' pixels when copying them to 'target' in the form [y, x] where positive values mean shifts down and to the right, respectively), and a reference starters code in Matlab (that code is making corresponding image matrices, shifting the mask and the source image). Your task is as easy as it can be: implement Poisson Image editing method and apply it to sample images.

Code snippet for loading and showing image in Python:

```
import skimage.io as io
import matplotlib.pyplot as plt

# this will load grayscale image, result is array MxN
im_gray = io.imread('filename.jpg', as_gray = True)

# this will load RGB image, result is array MxNx3
im_gray = io.imread('filename.jpg')

# this will plot RGB image from array A
# (must be MxMx3 and in the range [0,1])
io.imshow(A)
plt.show()

# this will plot matrix A (MxN in the range [0;1])
# as grayscale image
io.imshow(A, cmap='gray')
plt.show()
```