Problem Set 3

Due: November 7, 2023

1) [16 pts] In class, we said that Hermitian matrices are diagonalized by unitary matrices. We will use this idea to look at another matrix decomposition for non-square matrices. Let $\mathbf{A} \in \mathbb{C}^{M \times N}$ be an $M \times N$ matrix of rank $r \leq \min(M, N)$.

a) Show that $\mathbf{A}^*\mathbf{A}$ is a Hermitian matrix and is positive semi-definite.

b) We know $\mathbf{A}^*\mathbf{A}$ is diagonalized by a unitary matrix. How many positive eigenvalues does $\mathbf{A}^*\mathbf{A}$? [*Hint: Read the problem statement carefully.*]
Let l be this number, you can order its eigenvalues as $\sigma_1^2 \geq \sigma_2^2 > \cdots \geq \sigma_l^2 > 0$. Without loss of generality, we will also assume that the corresponding eigenvectors the first l respective columns of the unitary matrix $\mathbf{V}$ that diagonalizes $\mathbf{A}^*\mathbf{A}$. We will denote the k^{th} column of $\mathbf{V}$ as $\mathbf{v}_k$. What is $||\mathbf{v}_k||_2^2$ and why?

c) Now consider $\mathbf{u}_k = \frac{1}{\sigma_k}\mathbf{A}\mathbf{v}_k$ for $k \in 1, 2, \ldots, l$. Show that $||\mathbf{u}_k||_2^2 = 1$.

d) Show that $\mathbf{u}_k$ is an eigenvector of $\mathbf{A}\mathbf{A}^*$ with eigenvalue σ_k^2.

e) Let $\mathbf{V}_l$ be the $N \times l$ matrix whose k^{th} column is $\mathbf{v}_k$ above, and $\mathbf{U}_l$ be the $M \times l$ matrix whose k^{th} column is $\mathbf{u}_k$ above. Let $\mathbf{\Sigma}_l$ be the diagonal matrix with k^{th} diagonal entry σ_k. Show that

$$\mathbf{A}\mathbf{V}_l = \mathbf{U}_l\mathbf{\Sigma}_l.$$

f) Now show that

$$\mathbf{\Sigma}_l = \mathbf{U}_l^*\mathbf{A}\mathbf{V}_l.$$

With some additional work, this can be made into the compact or economical singular value decomposition of $\mathbf{A}$.

2) [9 pts] Programming Exercise 1: In this exercise, you will familiarize yourself with function handles. Write function handles to calculate the following:

a) The image derivative in x direction (the built-in function `numpy.diff`(Python)/`diff`(MATLAB) may be useful),

b) the image derivative in y direction,

c) the magnitude of the gradient vector in (x, y) directions,

d) discrete cosine transform (you may use the `scipy.fftpack.dct`(Python)/`dct2`(MATLAB) function for this) of each 8×8 distinct block in the image.

For a, b, and c make the derivative operator circulant. Thus, the output should be the same size as the image. Verify all 4 function handles on the cameraman image.

3) [10 pts] Programming Exercise 2: In this exercise, you will generate the image restoration results from the course, using both proximal gradient descent (PCG) and ADMM.

You can use a built-in Python function `TV_denoise`, which is imported in the main code you are given. For MATLAB, you are given a regularization function `TV_denoise.m`. These functions implement a total-variation-based denoising scheme, which essentially enforces sparsity in the gradient of the image. Both `TV_denoise.m` function takes two parameters, an image, `x` and the regularization weight, `aTV`; and outputs a denoised image, `z`, corresponding to

$$z = \arg\min_u \frac{1}{2}||x - u||_2^2 + aTV \cdot TV(u). \tag{1}$$

So think of $TV(x)$ is $\phi(x)$ regularizer that we talked about in class, and this function is the proximal operator for it.

In the scripts `Assignment3_main.py` and `Assignment3_main.m`, I have written out an implementation for a function handle that describes the blur $\mathbf{H}$ and its Hermitian transpose $\mathbf{H}^*$. The variable $\mathbf{y}$ is generated using this blur function and additive Gaussian noise:

$$\mathbf{y} = \mathbf{H}(\mathbf{x_{orig}}) + \sigma\mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{2}$$

You will need to de-blur this image, which is in `Assignment3_blurry_image.mat`

a) [4 pts] First, you will implement the proximal gradient descent algorithm. Initialize `x` to the blurry image. Set the gradient step-size to α^2/σ^2 which is defined in the main codes. For Python set the `aTV` = 1e-2 (named as weight in the built-in function) and for MATLAB set `aTV` = 1e-7 for the TV denoising part. Run it for $\approx$ 7500 iterations. Save your output at 500, 1000, 2500, and 5000 iterations. Notice, how slowly this algorithm converges.

b) [6 pts] Here, you will implement the ADMM algorithm. Initialize the two variables of ADMM images to all-zero images. The part, where one needs to implement

$$\mathbf{x} = (\mathbf{H}^*\mathbf{H} + \rho\mathbf{I})^{-\mathbf{1}}(\mathbf{H}^*\mathbf{y} + \rho(\mathbf{z} - \mathbf{u})) \tag{3}$$

should be implemented using conjugate gradient (CG). An example function is provided in both `cg_solve.py` and `cg_solve.m`, which you can use.

For this algorithm, use $\rho = 0.1$. Run it for $\approx$ 500 iterations, saving the output at 50, 100, and 250 iterations. Keep the `TV_denoise` regularization coefficient the same. For Python, TV-regularization in the main objective function should be 1e-2, and for MATLAB, it should be 1e-7. Note in the loop, this gets scaled as `aTV`$/\rho$. Run the CG algorithm used for solving for `x` for 20 iterations and starting from an all-zeros image. Note for the CG algorithm, you need to define a new function handle that corresponds to the matrix being inverted in Equation 3.