Homework 4 Numerical Analysis Spring 2023

Instructions:

- Due 04/06 at 11:59pm on Gradescope.
- Write the names of anyone you work with on the top of your assignment. If you worked alone, write that you worked alone.
- Show your work.
- Include all code you use as copyable `monospaced` text in the PDF (i.e. not as a screenshot).
- Do not put the solutions to multiple problems on the same page.
- Tag your responses on gradescope. Each page should have a *single* problem tag. Improperly tagged responses will not receive credit.

Problem 1. Download the numpy data file from this link: `https://drive.google.com/file/d/1Ao0HQDnaGlYirr8pak6TRUd8IItqTy9U/view?usp=share_link`

Use the following code to import the file into numpy.

```
import numpy as np
import matplotlib.pyplot as plt

im = np.load('change this path/tandon.npy')
A = np.mean(im,axis=2)
```

Here we obtain **A** by averaging the red, green, and blue channels of the image. This result sin a black and white image.

(a) Plot the image using `plt.imshow`. You may want to use the colormap `'Greys_r'` so that it looks like a greyscale image.

(b) Compute the reduced SVD of **A**. You can use `full_matrices=False` to get the reduced SVD. This will be much faster than computing the full SVD.

For each $k = 1, 10, 100, 200$, make a plot of the best rank-k approximation $\mathbf{A}_k$ to **A** (i.e. via truncated SVD). Label each plot with the rank k as well as the relative error $\|\mathbf{A} - \mathbf{A}_k\|_\mathsf{F} / \|\mathbf{A}\|_\mathsf{F}$

(c) Remark on the quality of the plots.

How many floating point numbers are required to store **A**? How many are required to store the rank-k truncated SVD?

Problem 2. Computing the SVD is expensive, but randomization can help us!

(a) Randomized numerical linear algebra (RandNLA) is the study of the use of randomness in numerical linear algebra algorithms. One of the most famous randNLA algorithms is the randomized SVD. A simple version for approximating the SVD of a $m \times n$ matrix $\mathbf{A}$ can be described in several lines:
   - Choose a $n \times k$ matrix $\mathbf{R}$ with standard normal random entries
   - Compute $\mathbf{X} = \mathbf{AR}$
   - Compute $\mathbf{Q}, \_ = \mathrm{QR}(\mathbf{X})$
   - Compute SVD of $\mathbf{Q}^\mathsf{T}\mathbf{A}$: $\hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^\mathsf{T}$
   - Return approximate SVD of $\mathbf{A}$: $(\mathbf{Q}\hat{\mathbf{U}})\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^\mathsf{T}$

   Implement this algorithm and time it for $k = 100$ with the same matrix $\mathbf{A}$ as in problem 1. To generate the random matrix, you can use `np.random.randn(n ,k)`.

   Again make sure to use `full_matrices=False` when computing the SVD of $\mathbf{Q}^\mathsf{T}\mathbf{A}$. Compare this to long the whole randomized SVD took (all of the steps) against the time to compute the exact SVD in the previous problem.

(b) Make a plot of the rank $k = 100$ truncated SVD (from problem 1) and the $k = 100$ randomized SVD. Show the relative errors $\|\mathbf{A} - (\mathbf{Q}\hat{\mathbf{U}})\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^\mathsf{T}\|_\mathsf{F}/\|\mathbf{A}\|_\mathsf{F}$ for each.

(c) Prove that $\mathbf{Q}\hat{\mathbf{U}}$ has orthonormal columns.

Problem 3. (a) Suppose $\mathbf{v}$ is an eigenvector of $\mathbf{A}$ with eigenvalue λ. Show $c\mathbf{v}$ is an eigenvector of $\mathbf{A}$. What is the eigenvalue?

(b) Suppose $\mathbf{X}$ is a $n \times m$ matrix. Write $\|\mathbf{X}\|_\mathsf{F}$ in terms of the column-norms $\|[\mathbf{X}]_{:,i}\|_2$.

(c) Suppose $\mathbf{X}$ is a $n \times m$ matrix and $\mathbf{U}$ is a $n \times n$ orthogonal matrix ($\mathbf{U}^\mathsf{T}\mathbf{U} = \mathbf{I}$). Show that $\|\mathbf{UX}\|_\mathsf{F} = \|\mathbf{X}\|_\mathsf{F}$. Hint: use (b) and show that $\|\mathbf{Ux}\|_2 = \|\mathbf{x}\|_2$ for any vector $\mathbf{x}$.

Problem 4. This problem will illustrate that solving the normal equations is less stable than other approaches.

(a) For each $\kappa = 10^1, 10^2, 10^3 \ldots, 10^8$, construct a 500×100 matrix $\mathbf{A}$ whose condition number is κ. A simple way to do this is to generate $\mathbf{U}$ and $\mathbf{V}$ as random orthogonal matrices of size $m \times n$ and $n \times n$ and define $\mathbf{\Sigma}$ as a $n \times n$ diagonal matrix manually.

   The following code gets you started, you just need to modify the line for the singular values `s = ...`.

```
m,n = 500,100
U,_ = np.linalg.qr(np.random.randn(m,n))
V,_ = np.linalg.qr(np.random.randn(n,n))
s = #TODO

A = U@np.diag(s)@V.T
```

Let $\mathbf{b}$ be the all ones vector, and compute the "true" solution to the least squares problem $\min_{\mathbf{x}} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2$ by setting $\mathbf{x}_{\text{true}} = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^{\mathsf{T}}\mathbf{b}$. This can be done with the following code:

```
b = np.ones(m)
x_true = V@np.diag(1/s)@U.T@b
```

Now, compute the least squares solution via:

- numpy's least squares solver `np.linalg.lstsq`
- a QR based approach with numpy's `np.linalg.qr` and `np.linalg.solve` or `sp.linalg.solve_triangular`
- Solving the normal equations with `np.linalg.solve`

For each of these three methods and each value of κ, record the relative error $\|\mathbf{x}_{\text{method}} - \mathbf{x}_{\text{true}}\|_2 / \|\mathbf{x}_{\text{true}}\|_2$, where $\mathbf{x}_{\text{method}}$ is the solution obtained by the given method.

(b) Make a log-log plot with the following five (labeled) curves:

- κ vs $10^{-16}\kappa$
- κ vs $10^{-16}\kappa^2$
- κ vs relative error (for each of the three methods above)

Comment on what you observe about the plots. In particular, discuss how each method depends on κ and what the relative errors would be if we did everything in exact arithmetic