

Homework 4 Numerical Analysis Spring 2023

Instructions:

- Problem 1.** Download the numpy data file from this link: https://drive.google.com/file/d/1Ao0HQDnaGLYirr8pak6TRUd8IIItqTy9U/view?usp=share_link

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

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

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

(c) Remark on the quality of the plots.

1

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 $\mathbf{X} = \mathbf{A}\mathbf{R}$
- Compute $\mathbf{Q}, _ = \text{QR}(\mathbf{X})$
- Compute SVD of $\mathbf{Q}^T \mathbf{A}$: $\hat{\mathbf{U}} \hat{\mathbf{\Sigma}} \hat{\mathbf{V}}^T$
- Return approximate SVD of \mathbf{A} : $(\mathbf{Q}\hat{\mathbf{U}}) \hat{\mathbf{\Sigma}} \hat{\mathbf{V}}^T$

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 $\mathbf{Q}^T \mathbf{A}$. 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 $\|\mathbf{A} - (\mathbf{Q}\hat{\mathbf{U}}) \hat{\mathbf{\Sigma}} \hat{\mathbf{V}}^T\|_F / \|\mathbf{A}\|_F$ 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 $\|\mathbf{X}\|_F$ 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 ($\mathbf{U}^T \mathbf{U} = \mathbf{I}$). Show that $\|\mathbf{U}\mathbf{X}\|_F = \|\mathbf{X}\|_F$. Hint: use (b) and show that $\|\mathbf{U}\mathbf{x}\|_2 = \|\mathbf{x}\|_2$ 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 $\kappa = 10^1, 10^2, 10^3 \dots, 10^8$, 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 $\mathbf{x}_{\text{true}} = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T\mathbf{b}$. 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