Numerical analysis: Project 1

Instructor: Anil Damle
Due: March 10, 2025

Policies

You are permitted to work in groups of up to three for this project and each group only needs to submit a single report. While it is okay to discuss the project with other groups (similar to the HW collaboration policy), every group must produce their own code and report. The report should address all of the items referred to as **goals** or **questions** in the project description. You may refer to inanimate sources (e.g., textbooks) to help you answer the questions and tackle the goals. However, your sources must be explicitly cited. For example, one of the goals requires writing pseudocode for an algorithm that exists, if you go to an external source to read more about it and help generate the pseudocoade you must cite where it came from.

Part of the purpose of this project is to provide you with an opportunity to practice writing more free form reports and carefully choosing what plots, results, etc. are needed to convince us your solution is correct. Therefore, some of the goals are leading, but do not give a concrete list of exactly what has to be included. The questions tend to be more concrete.

The report, including plots and requested output from your code, should be typeset and submitted via the Gradescope as a pdf file. This file must be self contained for grading. Additionally, please submit any code written for the assignment as zip file to the separate Gradescope assignment for code.

Preamble

For this project, we are going to work with images, and a simple form of matrix completion. We have picked a specific application and algorithm that you have all of the technical tools to complete. The state of the art in this field involves more complicated algorithms and, for images in particular, there are modifications that can be made to significantly improve the results. Furthermore, I have made some slight modifications to the underlying images to ensure things work out reasonablly here. If you are curious about this area I would be happy to chat more about it.

For the purposes of this project, all of the images we will work with are gray scale and represented by matrices with real valued entries between 0 and 1 (0 for black and 1 for white). So, an image that is $n_1 \times n_2$ pixels will be represented by an $n_1 \times n_2$ matrix. These techniques can be expanded to deal with color images, but here we will focus on the linear algebra underpinnings. When producing plots/images you should set the color space to gray so that it looks as you would expect for a gray-scale image.

Compressing images

Given a matrix $A \in \mathbb{R}^{n_1 \times n_2}$ we may want to store a compressed representation of it. One way to do this is to store a low-rank representation of A, and this can be accomplished using the SVD.

Say we compute the SVD

$$A = USV^T,$$

where $U \in \mathbb{R}^{n_1 \times p}$, $V \in \mathbb{R}^{n_2 \times p}$, $S \in \mathbb{R}^{p \times p}$, and $p = \min\{n_1, n_2\}$. Let U_k denote the first k columns of U, V_k denote the first k columns of V, and S_k denote the upper left $k \times k$ block of S. Now, we can write $A \approx U_k S_k V_k^T$ and, in fact, this is the optimal approximation of A in either the 2 or Frobenius norm. This means that of all rank-k matrices (those with k non-zero singular values), the one closest to A in 2 or Frobenius norm is $U_k S_k V_k^T$. The error of approximation is determined by the singular values of A, specifically

$$\left\|A - U_k S_k V_k^T\right\|_2 = \sigma_{k+1} \qquad \text{and} \qquad \left\|A - U_k S_k V_k^T\right\|_F = \left(\sum_{i=k+1}^{p} \sigma_i^2\right)^{1/2}.$$

So, if the singular values of A decay rapidly there may be a good low-rank approximation for A, and if k is small enough, storing U_k, V_k, and S_k may be cheaper than storing A.

Now, while we have not talked about them, there are good algorithms to compute the SVD. However, in certain settings it may not be possible to use these algorithms directly. So, let's consider an alternative. We can write any rank k approximation of A as WZ^T where $W \in \mathbb{R}^{n_1 \times k}$ and $Z \in \mathbb{R}^{n_2 \times k}$. Our low-rank approximation problem can then be cast as the optimization problem

$$\min_{W,Z} \left\|A - WZ^T\right\|_F^2.$$

Unfortunately, robustly solving this problem can be a bit tricky if we do not resort to the SVD. However, one possible approach is known as alternating least squares. It is based on the observation that if we fix W or Z then we can solve for the other factor via a (matrix) least-squares problem. This gives Algorithm 1, though for our purposes we will run it for a small, fixed number of iterations rather than checking for convergence (which may be preferable in practice).

Algorithm 1 Alternating least squares, without regularization

```
1: initialize $W \in \mathbb{R}^{n_1 \times k}$, $Z \in \mathbb{R}^{n_2 \times k}$
2: while not converged do
3:     $Z \leftarrow \arg\min_Z \left\|A - WZ^T\right\|_F^2$
4:     $W \leftarrow \arg\min_W \left\|A - WZ^T\right\|_F^2$
5: end while
6: return $W$, $Z$
```

Goal 1: Implement the Householder scheme for computing QR factorizations. Your QR factorization routine should take in a $n \times k$ matrix B with $k \leq n$ and returns a $n \times k$ matrix Q with orthonormal columns (or sufficient information to be able to apply it to a vector, e.g., the Householder vectors), and a $k \times k$ upper triangular matrix R. Please demonstrate that your algorithm behaves as expected (this means both that you are getting a valid QR factorization as output and that it achieves the desired computational scaling). **Demonstrate that your implementation achieves the expected scaling and explain how you tested your implementation.**

Goal 2: Using your QR factorization, implement Algorithm 1. This is written as a matrix least squares problem, but think about the Frobenius norm and how you might be able to split it up into problems we know how to solve. **Please explain how you do this in the project report.** Practically, it is possible to then block some of these operations together for efficiency; you may take advantage of this if you wish.

A note on file formats.

All matrices A provided for this project are saved as .csv files where each line (delineated by a newline character) of the file denotes a row of A and entries within a line are separated by commas. This should be easy enough to read in whatever language you are using.

Goal 3: Using your implementation, load the file Cornell.csv which has an image stored in the matrix C and try to compute the best rank 75 approximation of the image. Compare this with the result of the best rank 75 image from the SVD (you can use a built in routine to compute this), what do you observe qualitatively and quantitatively?

What about if we do not know all the entries of our image?

In the preceding section, alternating least squares seems like a roundabout way to get at something we know how to compute. But now we will explore a setting where we cannot compute the SVD as we might like. Specifically, we will consider being given an image A where we only know the true values of some subset of the pixels, the rest are unobserved. In this case we can still use a variant of alternating least squares to try and recover an underlying low-rank approximation that should roughly resemble our image.

The two key differences are that we will add regularization and only measure error on the observed pixels. Let Ω be a set of (i,j) pairs that represents the set of pixels we observe in the image. So, for example, if $\Omega = \{(1,2),(10,20)\}$ then we are only observing the $(1,2)$ and $(10,20)$ pixels (matrix entries) of the image. We now define a restricted variant of the Frobenius norm as follows

$$\|A - B\|_{F,\Omega}^2 = \sum_{(i,j)\in\Omega} \left(A_{i,j} - B_{i,j}\right)^2.$$

This norm simply measures the difference between A and B on a subset of their entries defined by Ω. The second adjustment we will make is to penalize W or Z from becoming large using a parameter β. This yields the following optimization problem

$$\min_{W,Z} \left\|A - WZ^T\right\|_{F,\Omega}^2 + \beta^2 \|W\|_F^2 + \beta^2 \|Z\|_F^2.$$

The hope is that WZ^T is then a good approximation of the underlying image.

Algorithm 2 is a version of alternating least squares with regularization and for incomplete information. Once again, you may simply run it for a fixed number of iterations (that you can tune based on the output) rather than checking for convergence as one would in practice.

Turns out, you have all of the knowledge and machinery to implement this algorithm (save for one small item we develop in the first point below).

Algorithm 2 Alternating least squares, with regularization and unknown entries

```
1: initialize $W \in \mathbb{R}^{n_1 \times k}$, $Z \in \mathbb{R}^{n_2 \times k}$
2: while not converged do
3:     $Z \leftarrow \arg\min_{Z} \left\| A - WZ^T \right\|_{F,\Omega}^2 + \beta^2 \left\| Z \right\|_F^2$
4:     $W \leftarrow \arg\min_{W} \left\| A - WZ^T \right\|_{F,\Omega}^2 + \beta^2 \left\| W \right\|_F^2$
5: end while
6: return $W$, $Z$
```

Question 1: Show that if we want to solve

$$\min_x \|Ax - b\|_2^2 + \beta^2 \|x\|_2^2$$

we can instead solve

$$\min_x \left\| \begin{bmatrix} A \\ \beta I \end{bmatrix} x - \begin{bmatrix} b \\ 0 \end{bmatrix} \right\|_2^2 .$$

Goal 4: Using your QR factorization, implement Algorithm 2. From the first set of goals, you should have worked out how to split this up into solving many least squares problems. Now, you have to think about what size those problems are, and which entries of the matrices they involve. Given that, you can then leverage your QR factorization and the preceding item to implement the algorithm. **Please explain how you do this in the project report.**

Goal 5: Using your implementation, load each of the image files in Image*.csv and associated masks Mask*.csv, where * is 1, 2, and 3. Each file contains an image and the associated mask file contains an image of the same size whose entries are 1 if the corresponding entry of C is observed and 0 if it is unobserved (so it defines the set Ω). The unknown entries of C have been set arbitrarily and you should not access or use them (as the will adversely affect your results). Using Algorithm 2, try and recover each of the underlying images. You now have some parameters to consider and we encourage exploration of their values. You should not have to go to k larger than 75 for any of the examples, and good β to try are between 10^{-2} and 1. Report the images you recover, and discuss what you observe.