UNIVERSITY OF WISCONSIN-MADISON

CS/ECE/ME 532 MATRIX METHODS IN MACHINE LEARNING

FINAL PROJECT

# Matrix Completion Through Nuclear Norm Regularization

*Author*
Michael LIU
Yimiao CAO
Pumeng LYU

*Supervisor*
Prof. Matthew MALLOY

December 11, 2019

# 1 Abstract

The matrix completion is the problem of finding the matrix that best fits the observed entries. Naturally, a wide range of datasets are organized in matrix form and in many situations that matrix is incomplete. Taken the Netflix problem as an example, where every entry is a particular user rating for a movie. This matrix is incomplete since Netflix cannot collect the ratings for every user on every movie. This problem arises in a variety of application, such as statistics(e.g. entropy methods for missing data), linear regression(least square systems), data compression, discrete optimization(relaxation methods), etc.

This project will explore two efficient algorithms for large matrix factorization and completion. We will briefly recap the assumptions might make on that underlying matrix to successfully reconstruct the entire matrix such as uniform sampling of observed entries and incoherent matrix. However, this rank-minimization problem is NP hard due to the non-convexity structure of the rank and therefore can't be solved directly. Employing the nuclear norm regularization to achieve shrinkage and low rank solutions is accessible but still complicated to solve. In order to simplify it even more, we will introduce the soft impute approach and randomized SVD algorithm to reduce computation. Precisely, instead of recomputing the data in each iteration, we will refine the predictive rule each time increasingly.

In this project, students will get hands-on experience with matrix completion from a series of practical case-studies. At the end of the workshop, students will be able to:

1. Fully understand nuclear norm after touching with the concept of nuclear norm.

2. Explain the assumptions for which rank minimization problem is possible.

3. Apply rank minimization and nuclear norm minimization to matrix completion problems. For example, they will exploit given code to recover incomplete matrices via using two convex relaxation methods and compare the advantages and disadvantages of these two algorithms.

# 2 Background

## 2.1 Problem Introduction

Matrix completion is the task of filling in the missing entries of a partially observed matrix. For more than twenty years, matrix completion problem remains as an open questions. It is a widely used technique for image inpainting and personalized recommender system, etc.In recent ten years we have had made tremendous progress in both speed and accuracy of algorithms for matrix completion. It might confuse someone with the necessity of studying matrix completion problem. So to begin with our activity, we would like to first give three problems simple but important in our daily lives to let us understand the motivation to study matrix-completion problem and what it helps us in the future.

### 2.1.1 Problem 1: Netflix Recommendation System

In class, we start practising our matrix computation using the Netflix Recommend System as an example. However, in real life some users might not fill all the ratings for every movies. In fact, there is a challenge for Netflix because it provides highly incomplete ratings from 0.5 million users for 17,770 movies.

So here comes the problem, how we can predict unseen user ratings for movies? In general, we cannot infer missing entries, unless this incomplete matrix has other hidden structure, i.e. the matrix has low rank. If we are lucky and are able to exploit low-rank structure, we can thus predict most of the data. The histogram in Figure2 gives an example that low-rank matrix with small number of eigenvalues which decreases quickly set a good example for us to try matrix-completion algorithm.

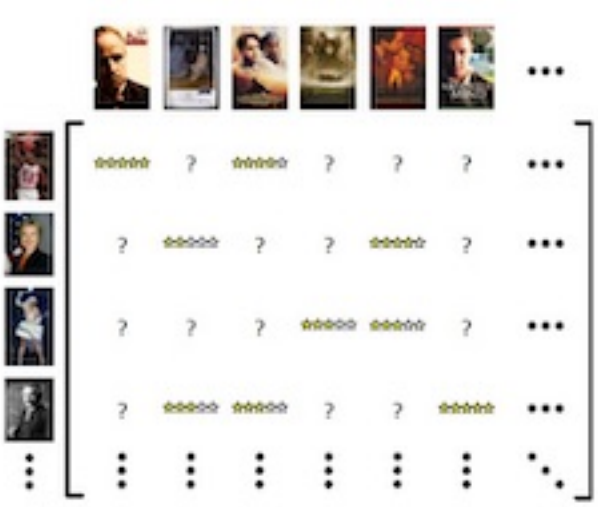

Figure 1: Problem 1: the Netflix Challenge for largely missing ratings.

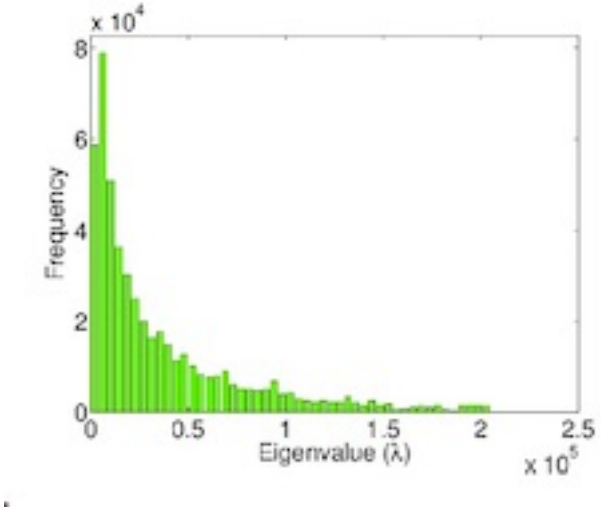


Figure 2: Histograms of eigenvalues for Netflix incomplete matrix.

Figure 3: reconstruct 3D locations from multiple 2D images.

### 2.1.2 Problem 2: Structure from Motion

This problem happens a lot in computer vision research. We take pictures, but pictures or images are two-dimensional. So if we are given multiple images and a few correspondences between image features, how we can reconstruct the 3D images? The process needs to use multiple images to estimate the locations of 3D points.

We thus reconstruct three-dimensional scene geometry (structure) and camera poses (motion) from multiple images. Figure3 gives two examples on reconstruct three-dimensional locations from two-dimensional images. One famous approach is called Tomasi and Kanade's factorization, first proposed by Carlo Tomasi and Takeo Kanade in the early 1990s. [3] The general idea is the following:

- Suppose we have n 3D points in m different 2D frames.
- We use $\mathbf{x}_{i,j}$ to represent locations of the $j^{th}$ point in the $i^{th}$ frame.

$$\mathbf{x}_{i,j} = \mathbf{M}_i \mathbf{p}_j$$

Here $M_i$ is the 2-by-3 projection matrix (in order to reconstruct from 2D to 3D) and $p_j$ stands for the locations in three-dimension.

- Matrix of 2D locations

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & \cdots & X_{1,n} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \cdots & x_{m,n} \end{bmatrix} = \begin{bmatrix} M_1 \\ \vdots \\ M_m \end{bmatrix} \begin{bmatrix} p_1 & \cdots & p_n \end{bmatrix} \tag{1}$$

Now we can apply matrix-completion algorithm to fill in missing entries of $\mathbf{X}$ given a small number of entries.

### 2.1.3 Problem 3: Missing Phase Problem

In physics, chemistry, biology research, we need sensors and detectors. For example, in physics, when do optics, we need detectors to record intensities of diffracted rays. However, due to research environment or other factors, the recorded ray signal might be blurred or not complete. So we need matrix-recovery algorithm to recover these distorted signals. Further explanation requires discussion on Fourier transform, which we will not work on in this introduction part.

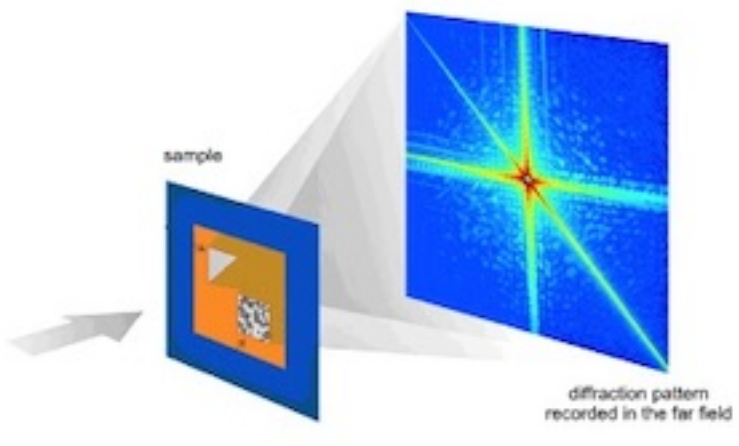


Figure 4: Problem 3: Missing Phase Problem. Photo credit: Stanford SLAC

## 2.2 Matrix Rank and Nuclear Norm

Before we move on to further discussion on matrix completion problems and two additional algorithms, let's first do a quick warm up on rank and nuclear norm, because our two introduced matrix-completion algorithms are based on finding the minimum rank or minimum nuclear norm of our partially known matrix. As we have had a lot practice in class, we will introduce them very briefly.

You can think of a r-by-c matrix **X** has as a set of r row vectors, each having c elements; or you can think of it as a set of c column vectors, each having r elements. The rank of a matrix is defined as (a) the maximum number of linearly independent column vectors in the matrix or (b) the maximum number of linearly independent row vectors in the matrix. Both definitions are equivalent.
On Nuclear norm, its definition is:

$$||\mathbf{X}|| = \sum_i \sigma_i(\mathbf{X})$$

Where i stands for the $i^{th}$ largest singular value. Here are some further remarks on Nuclear norm [1]:

- Nuclear norm is a counterpart of $l_1$ norm for rank function.
- Equivalence among different norms:

$$||\mathbf{X}|| \leq ||\mathbf{X}||_F \leq ||\mathbf{X}||_* \leq \sqrt{r}||\mathbf{X}||_F \leq r||\mathbf{X}||_*$$

Now we move on to explain and prove the additivity of nuclear norm.

Fact: Let **A** and **B** be matrices of the same dimensions. If $\mathbf{A}\mathbf{B}^T = 0$ and $\mathbf{A}^T\mathbf{B} = 0$, then $||\mathbf{A} + \mathbf{B}||_*$ = $||\mathbf{A}||_*$ + $||\mathbf{B}||_*$.

*Proof.* Suppose **A** = $\mathbf{U}_A\Sigma_A\mathbf{V}_A^T$ and **B** = $\mathbf{U}_B\Sigma_B\mathbf{V}_B^T$, based on $\mathbf{A}\mathbf{B}^T = 0$ and $\mathbf{A}^T\mathbf{B} = 0$, we have:

$$\mathbf{V}_A^T\mathbf{V}_B = 0$$
$$\mathbf{U}_A^T\mathbf{U}_B = 0$$

Then we can write

$$\mathbf{A} = \begin{bmatrix} U_A & U_B & U_C \end{bmatrix} \begin{bmatrix} \Sigma_A & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} V_A & V_B & V_C \end{bmatrix}^T \tag{2}$$

$$\mathbf{B} = \begin{bmatrix} U_A & U_B & U_C \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & \Sigma_B & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} V_A & V_B & V_C \end{bmatrix}^T \tag{3}$$

and from (1), (2), hence

$$||\mathbf{A} + \mathbf{B}||_* = || \begin{bmatrix} U_A & U_B \end{bmatrix} \begin{bmatrix} \Sigma_A & 0 \\ 0 & \Sigma_B \end{bmatrix} \begin{bmatrix} V_A & V_B \end{bmatrix}^T || = ||\mathbf{A}||_* + ||\mathbf{B}||_* \tag{4}$$

■

We leave two closing remarks to summarize of discussion on nuclear norm.

- If row or column spaces of **A** and **B** are orthogonal, then $||\mathbf{A} + \mathbf{B}||_*$ = $||\mathbf{A}||_*$ + $||\mathbf{B}||_*$.
- Similar to l1 norm, when **x** and **y** have disjoint support, then $||\mathbf{x} + \mathbf{y}||_1$ = $||\mathbf{x}||_1$ + $||\mathbf{y}||_1$.

## 2.3 Matrix Completion Overview

Missing data is a common occurrence in modern application, including collaborative filtering, dimensionality reduction, image processing and multi-class classification. One of the motivations for the matrix completion problem comes from recommending the items to users based on their previous ratings of some products which are put into a matrix **M**. The entries $\mathbf{M}_{ij}$ of the matrix correspond to the $j^{th}$ user's rating of product i. Assuming there exist an ideal matrix that can predict the ratings of all the products by all the users by given some ratings from some users. Basically, our goal is to recover the missing elements in M from the limited data. However, this problem is virtually impossible to solve if given no presumption about the nature of matrix entries since the missing $\mathbf{M}_{ij}$ could be arbitrary. Consider that, we will recover the matrix relies on the following assumptions:

1. Based on the high-dimensional and massive essence of today's data driven community, it is arguable that the target matrix is approximately low rank since there will be lots of correlation between the rows and columns.

The best way to understand is would be through the classical Netflix problem. Saying that there were m movies and n users, so the m*n matrix was actually rank r, then its SVD can be written as $\mathbf{M} = \sum_i \mathbf{u}_i v_i^T \sigma_i$. Each $\mathbf{u}_i$ could represent a movie genre and then $\mathbf{v}_i$ corresponds how much each user likes the genre. Indeed, $\mathbf{u}_i \mathbf{v}_i^T \sigma_i$ is an m*n matrix that contains the contribution of that given genre to all the elements. We need to compute the inner product and want to do it faster, so want these vectors to have as few dimensions as possible, which is where low rank comes into play.

2. Both the column space and the row space are "incoherent".

We say it is incoherent, it means that the projection of any vector onto the space has a small $\mathbf{l}_2$ norm. Here, we need to know that this is quantified in a measure of "coherence" for a subspace, which measured how "spread out" the basis vectors are. Let **U** be a r-dimensional subspace, $\mathbf{P}_u$ be the orthogonal projection onto **U**, we can define the coherence of the r-dimensional subspace **U** of $\mathbf{R}^n$ as

$$\mu(U) = \frac{n}{r} max_{1<i<n} ||\mathbf{P}_u \mathbf{e}_i||_2^2$$

where r = rank(**M**). This can be regarded as a measure of "how will can the subspace represent $\mathbf{e}_i$." That is, how ordered are they.

3. The set is sampled uniformly at random.

To make the result tractable, it is often assumed that the observed entries and fixed cardinality is sampled uniformly at random from the collection of all subsets of entries of cardinality.

Under these assumptions, one may easily come up with the solution by seeking a matrix with the lowest rank that also satisfies the constraints given by the observed entries:

Minimize rank(X);
Subject to $\mathbf{X}_{ij}$ = $\mathbf{M}_{ij}$ for all i,j in the set

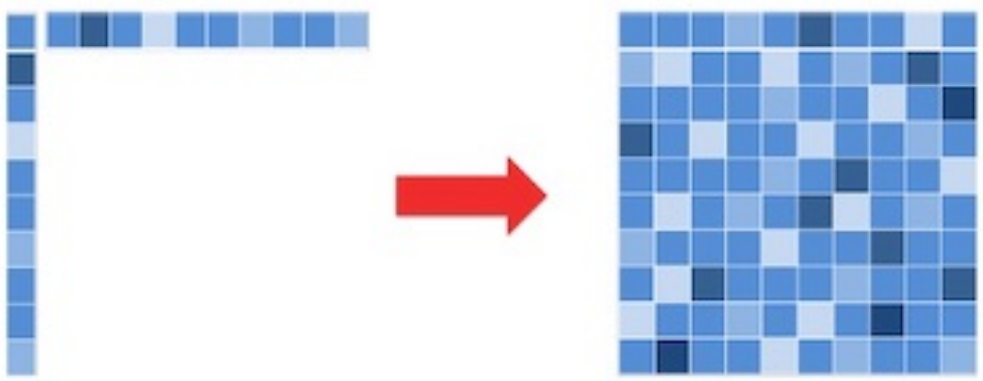

Figure 5: Matrix Recovery Idea, assuming Matrix has rank 1

Unfortunately, this idea is of little practical since the problem is NP-hand and cannot be solved in polynomial time. We will therefore consider the following alternative that minimizes instead the nuclear norm which is a convex problem. Namely,

$$\text{Minimize } ||\mathbf{X}||_*,$$

$$\text{Subject to: } \mathbf{X}_{ij} = \mathbf{M}_{ij} \text{ for all i,j in the set}$$

*Note: the nuclear norm of X defined as the sum of the singular values of X, i.e.* $||\boldsymbol{X}|| = \sum_i \sigma_i(\boldsymbol{X})$

Classical algorithm for solving the problem can be expensive for large datasets and sometimes outperform the rank minimized solution. In other words, the setup may not handle well the dependence among the missing entries. To break through the limitation of the current algorithm, we will introduce a softImpute algorithm. Although an SVD needs to be computed each time in the iteration, we can use the previous solution as a warm start. As one gets closer to the solution, the warm starts tend to be better and so the final iterations tend to be faster. r that matches the entries in a high percentage.

## 2.4 Two Algorithms

Our next objective is going to introduce two matrix completion algorithm: Soft Impute and Randomized SVD. We will further illustrate usefulness of these two algorithms and their advantages and disadvantages. Two methods all require Singular Value Decomposition, which we have discussed before in class. As we stated lots of times in previous section, to do matrix completion, we assume that the incomplete matrix has low-rank. Figure5 gives the idea of recovering rank-1 incomplete matrix.

### Soft Impute

Soft impute [2] is an useful method proposed initially by (author) as an online matrix completion scheme. Given an observed matrix $\mathbf{X}$, we define $\mathbf{X}_{ij}$ with $(i, j) \in$ sampling set $\omega$. With sampling set $\omega$, we define an operator $P_\omega$, which is the orthogonal projection onto subspace of matrices supported on $\omega$. So the idea of Soft Inpute is to find matrix $\mathbf{Z}$ with the lowest rank such that $P_\omega(\mathbf{Z}) \approx P_\omega(\mathbf{X})$. To achieve this, we also need apply ideas we learned in class such as iterations and convergence to find the minimum rank($\mathbf{Z}$).

To start our algorithm, we need a list of $\lambda$'s in descending order. And we also need a threshold value $\epsilon$ for convergence check. With a partially observed matrix X, we first initialize $\mathbf{Z}^{(1)} = 0$ and j = 1.

As k going from 1,2,3 and so on, for every k, we initially set $\gamma = \epsilon + 1$. Every time we compute the SVD of $P_\omega(\mathbf{X}) - P_\omega(\mathbf{Z})$, which is the $\omega$ projection on our matrices $\mathbf{X}$ and $\mathbf{Z}$. We assign the result of SVD to $\mathbf{Z}^{(j+1)}$. We iterate

as j increases until $\mathbf{Z}^{(j)}$ shows convergence smaller than our threshold $\epsilon$. We save our final $\mathbf{Z}^{(j)}$ as $\mathbf{Z}_{\lambda_i}$ for i=1,2,3 until k. So at last we receive a list of $\mathbf{Z}'s$: $\mathbf{Z}_{\lambda_1}$, $\mathbf{Z}_{\lambda_2}$, $\mathbf{Z}_{\lambda_3}$,..., $\mathbf{Z}_{\lambda_k}$. The seudo-code for soft input method is the following (Note that the expression $\mathbf{S}_{\lambda_i}(\ P_\omega(\mathbf{X}) + P_\omega(\mathbf{Z}^{(j)}))$ is just the SVD of $P_\omega(\mathbf{X}) - P_\omega(\mathbf{Z})$ which we have already had a lot of computation practice in class):

**Algorithm 1:** Algorithm: Soft Impute

**Require:** Partially known matrix $\mathbf{X}$, regularisation parameters $\lambda_1 > \lambda_2$ ... $> \lambda_k$ , error threshold $\epsilon$;
initialization: $\mathbf{Z}^{(1)} = 0$, j = 1;
**for** $i = 1 \to k$ **do**
  $\gamma = \epsilon + 1$ ;
  **while** $\gamma < \epsilon$ **do**
    $\mathbf{Z}^{(j+1)} \leftarrow \mathbf{S}_{\lambda_i}(\ P_\omega(\mathbf{X}) + P_\omega(\mathbf{Z}^{(j)}))$;
    $\gamma = \frac{||\mathbf{Z}^{(j+1)} - \mathbf{Z}^{(j)}||^2}{||\mathbf{Z}^{(j)}||^2}$;
    j $\leftarrow$ j + 1
  **end**
  $\mathbf{Z}_{\lambda_i} \leftarrow \mathbf{Z}^{(j+1)}$
**end**
**return**:$\mathbf{Z}_{\lambda_1}$, $\mathbf{Z}_{\lambda_2}$, $\mathbf{Z}_{\lambda_3}$,..., $\mathbf{Z}_{\lambda_k}$.

Soft Impute Algorithm has two disadvantages. One disadvantage is that at each stage one must compute the SVD of a large matrix. The step to compute SVD of $\omega$ projection on matrix $\mathbf{X}$ and $\mathbf{Z}$ is very expensive. Another disadvantage is that every time we compute the SVD, we ignore the previous computation of this SVD. To improve our algorithm, we are going to introduce our second algorithm for today, Randomised SVD algorithm.

**Randomised SVD**

In recent years, randomized matrix computation has gained significant increase in popularity. Compared with classic algorithms, the randomized algorithm involves the same or fewer floating-point operations (flops), and is more efficient for truly large data sets. An idea of randomization is using random projection to identify the subspace capturing the dominant actions of a matrix. Then, a near-optimal low-rank decomposition of the matrix can be computed. Besides, as sparse matrix is processed in matrix completion, special technique should be devised to make the randomized SVD approach really competitive.

So the key idea of randomised algorithms is to project the rows onto a subspace which captures most of the "action" of the matrix. In addition to use SVD, this randomized technique has been extended to compute PCA of data sets. In our activity, we are going to use the idea of random projection to identify the subspace capturing the dominant actions of matrix $\mathbf{A}$. To do this, we can multiply a random Gaussian Matrix.

For the pseudo-code of our algorithm below, the purpose of the first three steps is to find a matrix $\mathbf{V} \in \mathbb{R}^{m\times(k+p)}$ such that the projection of $\mathbf{A}$ onto $\mathbf{V}$ is a good approximation of $\mathbf{Y}$. In other words, we hope to find $\mathbf{V}$ with orthogonal columns such that $||\mathbf{A} - \mathbf{V}\mathbf{V}^T\mathbf{A}||_2$ is small, where $||A||_2 = \sigma_1$ is the spectral norm. This is similar to the least square problem we did to find minimum norm.

**Algorithm 2:** Algorithm: Randomised SVD

**Require:** Matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, target rank k, oversampling projection vectors p, exponent q;
Generate a random Gaussian matrix $\Omega \in \mathbb{R}^{m\times(k+p)}$;
Create Y = $(\mathbf{A}\mathbf{A}^T)^q\mathbf{A}\Omega$ by alternative multiplication with A and A Transpose;
Compute **Y = VR** using the QR-decomposition;
Form **B = $\mathbf{V}^T$A** and compute SVD of **B = P'** $\Sigma\mathbf{Q}^T$;
Set **P = VP'**
**return**: Approximate SVD $\mathbf{A} \approx \mathbf{P}\ \Sigma\mathbf{Q}^T$

Before we move on to exercise on randomised SVD algorithm, we have to keep one thing in mind. Firstly, a fast randomized SVD technique is proposed for sparse matrix. Also, for this Randomized SVD, it has two advantages

compare to Soft Input. the first is that the above algorithm can be effective even when there is not a jump in the spectrum in the incomplete matrices. Secondly, one can trivially compute $\mathbf{A}\Omega$ in parallel, so the computational cost for this one-step, non-iterative computation is less expensive. We are now going to do exercises on the two algorithms we discuss step by step. But first we will start with trivial warm-up exercises.

### 2.5 Ethical Issues

As the speed and accuracy of matrix completion algorithm increases, we should also be concerned of the ethical problems behind it. Overuse of advanced matrix-completion algorithm can cause security issues. For example, without full information of a person, we can easily find his or her missing personal information. What's more, matrix-completion can be used to reconstruct online payment information which often is partially disclose by customers. So we need carefully take advantage of matrix completion algorithms.

## 3 Warm-Up Questions

*Estimated Time: 15 minutes.*

1. Which of the following is the nuclear norm?
   A. p = 1: $||\mathbf{X}||_* = \sum_i \sigma_i(\mathbf{X})$
   B. p = 2: $||\mathbf{X}|| = \sum_i \sigma_i(\mathbf{X})$
   C. p = $\infty$: $||\mathbf{X}||_2 = \max \sigma_i(\mathbf{X})$
2. Which of the following are true about the soft impute algorithm?
   A. Soft Impute algorithm needs to be computed each time from the start
   B. Soft Impute algorithm is fast in the beginning
   C. Soft Impute algorithm is a non-convex problem
   D. Soft impute algorithm is good at dealing with relatively larger matrices
3. What are the advantages of randomized SVD algorithm over the traditional one?
   A. This algorithm is effective even when there is not a jump in the spectrum in the incomplete matrices
   B. This method gains a very significant speedup when one seeks a low-rank approximation with a small rank relative to the ambient matrix dimension
   C. Randomized algorithms provide an efficient computational framework for reducing the computational demands of traditional matrix factorization.
   D. A and C
   E. All of the things above
4. We've discussed a reason that matrix completion techniques fail for top-n recommendation. What is that reason?
   A. These techniques depend on the assumption that the data we observe is randomly distributed from all data, and this assumption is almost never true in real recommendation environments
   B. Accurate matrix completion techniques generally require between 10% and 20% observed values, but most recommender applications have much less available data. The result is that these techniques are unbiased but have a very large noise component that hurts our recommendations.
   C. We have accurate methods for matrix completion, but they are computationally intractable. Instead we will use approximation techniques that we know are biased towards filling in values too close to the mean.

# 4 Main Activity

*Estimated Time: 20 minutes for (a), 15 minutes for (b), 15 minutes for (c).*

## 4.1 Question

Like Activity 15, incomplete.mat contains a 16-by-16, rank-2 matrix Xtrue along with three other incomplete matrices $Y_1$, $Y_2$, and $Y_3$ that has some Xtrue's entries. Xtrue has 256 entries, $Y_1$ has 120 entries, $Y_2$ has 180 entries, and $Y_3$ has 240 entries. The missing entries in $Y_1$, $Y_2$, and $Y_3$ are denoted as NaN (np.nan). The script `matrix_completion_questions.ipynb` uses three methods to complete the matrices, each with their required arguments introduced below (and in the script's comments).

(a) Apply the iterative singular value thresholding function on the three incomplete matrices like in Activity 15. Then follow the hints in the comments and apply the soft impute algorithm with your own arguments. Compare the outcome of the two algorithms.

(b) Now follow the hints in the comments and apply the randomized SVD algorithm with your own arguments. It might be a good idea to try a large number of arguments in order to get a better result. In order to do this, consider using itertools.product. Compare the outcome of the three algorithms on $Y_1$.

(c) To find which specific p, q minimizes randomized SVD, complete part b) but use a two-dimensional array to replace the generator that itertools.product generates. (This page might be a good reference: https://stackoverflow.com/questions/22883348/itertools-product-return-list-instead-of-tuple) and use numpy.argmin to answer this question.

# 5 Conclusion

In the matrix completion problems, many researchers have given attention on it. In this project, we have presented the pseudo-code and Python code for two new algorithms for matrix completion, suitable for solving large number sets in efficient ways. The Soft Impute algorithm works better than Iterative Singular Value Thresholding across all matrices, even if it wastes time in early iterations for computing a low-rank SVD, but by the time using, it still has the solution with far fewer iterations. It can be thought as filling in the matrix at alternative steps. The Randomized SVD, however, only works well and outperforms Iterative Singular Value Thresholding when the given matrix is considerably sparse.

# References


[1] Yuxin Chen. *Low-Rank Matrix Recovery*. Princeton University, 2017.

[2] Charapal Dhanjal and Stephan Clemenc ON. *Online Matrix Completion Through Nuclear Norm Regularization*. stat.ML, 2014.

[3] Carlo Tomasi and Takeo Kanade. *Shape and motion from image streams under orthography: a factorization method.* International Journal of Computer Vision 9.2 (1992): 137-154., 1992.

# Appendix

## Solutions to warm-up activities

1. Which of the following is the nuclear norm?
    A. p = 1: $||\mathbf{X}||_* = \sum_i \sigma_i(\mathbf{X})$
    B. p = 2: $||\mathbf{X}|| = \sum_i \sigma_i(\mathbf{X})$
    C. p = $\infty$: $||\mathbf{X}||_2 = \max \sigma_i(\mathbf{X})$
2. Which of the following are true about the soft impute algorithm?
    A. Soft Impute algorithm needs to be computed each time from the start
    B. Soft Impute algorithm is fast in the beginning
    C. Soft Impute algorithm is a non-convex problem
    D. Soft impute algorithm is good at dealing with relatively larger matrices
3. What are the advantages of randomized SVD algorithm over the traditional one?
    A. This algorithm is effective even when there is not a jump in the spectrum in the incomplete matrices
    B. This method gains a very significant speedup when one seeks a low-rank approximation with a small rank relative to the ambient matrix dimension
    C. Randomized algorithms provide an efficient computational framework for reducing the computational demands of traditional matrix factorization.
    D. A and C
    E. All of the things above
4. We've discussed a reason that matrix completion techniques fail for top-n recommendation. What is that reason?
    A. These techniques depend on the assumption that the data we observe is randomly distributed from all data, and this assumption is almost never true in real recommendation environments
    B. Accurate matrix completion techniques generally require between 10% and 20% observed values, but most recommender applications have much less available data. The result is that these techniques are unbiased but have a very large noise component that hurts our recommendations.
    C. We have accurate methods for matrix completion, but they are computationally intractable. Instead we will use approximation techniques that we know are biased towards filling in values too close to the mean.

## Solutions to Main Activity

(a) Soft impute performs much better than iterative singular value thresholding when the rank is correct (in this case, 2) when the matrix is relatively sparse (when there are 180 out of 256 entries). Using the lambdas in the answer notebook, $Y_1$'s error dropped from 8999 to 124, while $Y_2$'s error dropped from 14 to 1.5. Soft impute is comparable to iterative singular value thresholding when the matrix is almost complete (for $Y_3$, which contains 240 out of 256 entries, iterative singular value thresholding performs slightly better than soft impute, which has an error of 0.0009) Here are the errors plotted against number of iterations for each matrix:

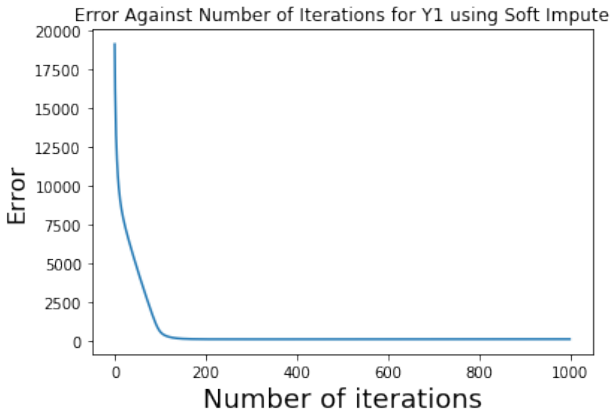


Figure 6: Error against iterations for Y1

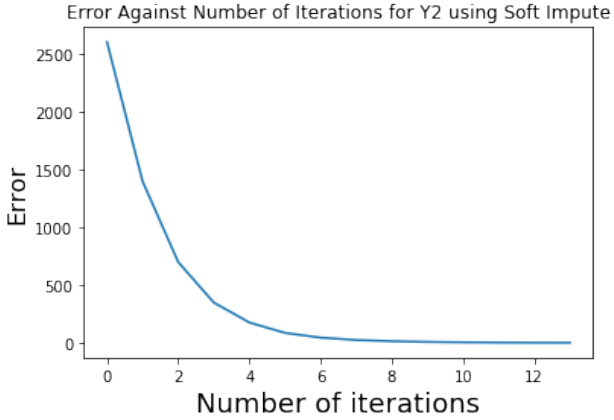


Figure 7: Error against iterations for Y2

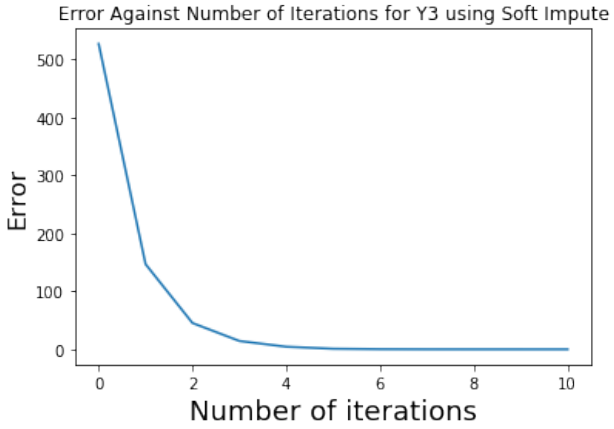


Figure 8: Error against iterations for Y3

(b) Using something like

```
itertools.product([0,0,1,1,2,2,5,5,10,10,50,50],repeat=2)
```

we can get an error below 200 for $Y_1$, which is pretty large drop from the iterative singular value thresholding algorithm. But it is hard to get better result for $Y_2$ and $Y_3$.

(c) The right answer is to minimize the oversampling projection vector and maximize exponent here. For the list of values used in part b, this means p is 0 and q is 50. One can also write the following lines in the notebook to figure this out:

```
from itertools import product
pqproduct=product([0,0,1,1,2,2,5,5,10,10,50,50],repeat=2)
allpqs=[list(x) for x in pqproduct]
allresults=[np.linalg.norm(Xtrue-RandSVD(Y1,2,p,q),'fro') for p,q in allpqs]
min(allresults)
minidx=np.argmin(allresults)
print(allpqs[minidx])
```