

## EE263 Homework 7 Solutions

Fall 2025

due Thursday 11/13, at 11:59 PM

**15.2290. Matrix gain compared with entry size.** A matrix can have all entries large and yet have small gain in some directions, that is, it can have a small  $\sigma_{\min}$ . For example,

$$A = \begin{bmatrix} 10^6 & 10^6 \\ 10^6 & 10^6 \end{bmatrix}$$

has “large” entries while  $\|A[1 \ -1]^T\| = 0$ . Can a matrix have all entries small and yet have a large gain in some direction, that is, a large  $\sigma_{\max}$ ? Suppose, for example, that  $|A_{ij}| \leq \epsilon$  for  $1 \leq i, j \leq n$ . What can you say about  $\sigma_{\max}(A)$ ?

**Solution.** The answer is no. If a matrix has all its entries small, then it cannot have a large gain in some direction. The precise statement is: if  $|A_{ij}| \leq \epsilon$  for all  $i, j$  then we have  $\sigma_{\max}(A) \leq \epsilon n$ . We can give a direct proof of this fact. Let  $x \in \mathbb{R}^n$ , and define  $y = Ax$ .  $y_i$ , the  $i$ th component of  $y$ , is the product of the  $i$ th row of  $A$  and  $x$ . By the Cauchy-Schwarz inequality,  $|y_i| \leq \|a_i\| \|x\|$ , where  $a_i$  is the  $i$ th row of  $A$ . Since every entry in  $a_i$  is less than  $\epsilon$ , we have  $\|a_i\| \leq \sqrt{n}\epsilon$ . Therefore, every entry of  $y$  has absolute value less than or equal to  $\sqrt{n}\epsilon \|x\|$ . It follows that  $\|y\| \leq \sqrt{n}\sqrt{n}\epsilon \|x\| = n\epsilon \|x\|$ . All of this shows that for any  $x$ ,  $\|Ax\| \leq n\epsilon \|x\|$ , and it follows immediately that  $\sigma_{\max}(A) \leq \epsilon n$ . In fact, it turns out that this bound cannot be improved. Let  $\mathbf{1}$  denote the vector in  $\mathbb{R}^n$  with all entries one. Let  $A = \epsilon \mathbf{1}\mathbf{1}^T$ , which is a matrix with all its entries equal to  $\epsilon$ .  $A^T A = \epsilon^2 (\mathbf{1}^T \mathbf{1}) \mathbf{1}\mathbf{1}^T$ , and so has eigenvalues  $\epsilon^2 n^2$  and  $n - 1$  zeros. Therefore,  $\sigma_{\max}(A) = n\epsilon$ .

**15.2300. Frobenius norm of a matrix.** The Frobenius norm of a matrix  $A \in \mathbb{R}^{n \times n}$  is defined as  $\|A\|_F = \sqrt{\text{trace } A^T A}$ . (Recall trace is the trace of a matrix, *i.e.*, the sum of the diagonal entries.)

a) Show that

$$\|A\|_F = \left( \sum_{i,j} |A_{ij}|^2 \right)^{1/2}.$$

Thus the Frobenius norm is simply the Euclidean norm of the matrix when it is considered as an element of  $\mathbb{R}^{n^2}$ . Note also that it is much easier to compute the Frobenius norm of a matrix than the (spectral) norm (*i.e.*, maximum singular value).

b) Show that if  $U$  and  $V$  are orthogonal, then  $\|UA\|_F = \|AV\|_F = \|A\|_F$ . Thus the Frobenius norm is not changed by a pre- or post- orthogonal transformation.

c) Show that  $\|A\|_F = \sqrt{\sigma_1^2 + \cdots + \sigma_r^2}$ , where  $\sigma_1, \dots, \sigma_r$  are the singular values of  $A$ . Then show that  $\sigma_{\max}(A) \leq \|A\|_F \leq \sqrt{r} \sigma_{\max}(A)$ . In particular,  $\|Ax\| \leq \|A\|_F \|x\|$  for all  $x$ .

**Solution.**

a) Simply by definition

$$\|A\|_F^2 = \text{trace } A^T A = \sum_i [A^T A]_{ii} = \sum_i \left( \sum_j A_{ij}^T A_{ji} \right) = \sum_{i,j} A_{ij}^2.$$

b) First note that  $\|UA\|_F = \|A\|_F$  because

$$\|UA\|_F^2 = \text{trace}(UA)^T(UA) = \text{trace } A^T U^T U A = \text{trace } A^T A = \|A\|_F^2.$$

and  $\|AV\|_F = \|A\|_F$  since

$$\|AV\|_F^2 = \text{trace}(AV)^T(AV) = \text{trace}(AV)(AV)^T = \text{trace } AVV^T A^T = \text{trace } AA^T = \text{trace } A^T A = \|A\|_F^2$$

where we have used the fact that  $\text{trace } XY = \text{trace } YX$ .

c) We start with the full SVD of  $A = U\Sigma V^T$ . By the previous problem,

$$\|A\|_F = \|U^T \Sigma V\|_F = \|\Sigma V\|_F = \|\Sigma\|_F = \sqrt{\sigma_1^2 + \dots + \sigma_r^2}.$$

Since  $\sigma_2^2, \dots, \sigma_r^2 \geq 0$ , we have  $\sigma_1 \leq \sqrt{\sigma_1^2 + \dots + \sigma_r^2} = \|A\|_F$ . Next, since  $\sigma_2, \dots, \sigma_r \leq \sigma_1$ , we have  $\|A\|_F = \sqrt{\sigma_1^2 + \dots + \sigma_r^2} \leq \sqrt{\sigma_1^2 + \dots + \sigma_1^2} = \sqrt{r} \sigma_1$ .

**15.2350. Two representations of an ellipsoid.** In the lectures, we saw two different ways of representing an ellipsoid, centered at 0, with non-zero volume. The first uses a quadratic form:

$$\mathcal{E}_1 = \left\{ x \mid x^T S x \leq 1 \right\},$$

with  $S^T = S > 0$ . The second is as the image of a unit ball under a linear mapping:

$$\mathcal{E}_2 = \{ y \mid y = Ax, \|x\| \leq 1 \},$$

with  $\det(A) \neq 0$ .

a) Given  $S$ , explain how to find an  $A$  so that  $\mathcal{E}_1 = \mathcal{E}_2$ .

b) Given  $A$ , explain how to find an  $S$  so that  $\mathcal{E}_1 = \mathcal{E}_2$ .

c) What about uniqueness? Given  $S$ , explain how to find *all*  $A$  that yield  $\mathcal{E}_1 = \mathcal{E}_2$ . Given  $A$ , explain how to find *all*  $S$  that yield  $\mathcal{E}_1 = \mathcal{E}_2$ .

**Solution.** *Finding A from S and vice-versa:* First we will show that

$$\mathcal{E}_2 = \left\{ y \mid y^\top (AA^\top)^{-1} y \leq 1 \right\} \quad (1)$$

$$\begin{aligned} \mathcal{E}_2 &= \{y \mid y = Ax, \|x\| \leq 1\} \\ &= \{y \mid A^{-1}y = x, \|x\| \leq 1\} \text{ since } A \text{ is invertible square matrix} \\ &= \{y \mid \|A^{-1}y\| \leq 1\} \\ &= \left\{ y \mid y^\top A^{-T} A^{-1} y \leq 1 \right\} = \left\{ y \mid y^\top (AA^\top)^{-1} y \leq 1 \right\} \end{aligned}$$

Since  $S$  is symmetric positive definite, the eigenvalues of  $S$  are all positive and we can choose  $n$  orthonormal eigenvectors. So  $S = Q\Lambda Q^\top$  where  $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n) > 0$  and  $Q$  is orthogonal. Let  $\Lambda^{\frac{1}{2}} = \text{diag}(\sqrt{\lambda_1}, \dots, \sqrt{\lambda_n})$ .

If we let  $A = Q(\Lambda^{\frac{1}{2}})^{-1} = Q\Lambda^{-\frac{1}{2}}$ ,

$$\begin{aligned} (AA^\top)^{-1} &= (Q\Lambda^{-\frac{1}{2}}\Lambda^{-\frac{1}{2}}Q^\top)^{-1} \\ &= (Q\Lambda^{-1}Q^\top)^{-1} = Q\Lambda Q^\top \\ &= S \end{aligned}$$

Therefore, by (1)  $A = Q\Lambda^{-\frac{1}{2}}$  yields  $\mathcal{E}_1 = \mathcal{E}_2$ . By (1),  $S = (AA^\top)^{-1}$  yields  $\mathcal{E}_1 = \mathcal{E}_2$ .

*Uniqueness:* We show that

$$\mathcal{E}_S = \mathcal{E}_T \Leftrightarrow S = T \quad (2)$$

where  $\mathcal{E}_S = \{x \mid x^\top Sx \leq 1\}$ ,  $\mathcal{E}_T = \{x \mid x^\top Tx \leq 1\}$ ,  $S^\top = S > 0$  and  $T^\top = T > 0$ .

It is clear that if  $S = T$ , then  $\mathcal{E}_S = \mathcal{E}_T$ . Now we show that  $\mathcal{E}_S = \mathcal{E}_T \Rightarrow x^\top Sx = x^\top Tx, \forall x \in \mathbb{R}^n$ . Without loss of generality let's assume  $\exists x_0 \in \mathbb{R}^n$  such that  $x_0^\top Sx_0 > x_0^\top Tx_0 = \alpha \neq 0$ . If we let  $x_1 = x_0/\sqrt{\alpha}$ , then  $x_1^\top Tx_1 = 1$ , but  $x_1^\top Sx_1 > 1$ , thus  $x_1 \in \mathcal{E}_T$  but  $x_1 \notin \mathcal{E}_S$ , and therefore  $\mathcal{E}_S \neq \mathcal{E}_T$ . Finally,  $\mathcal{E}_S = \mathcal{E}_T \Rightarrow x^\top Sx = x^\top Tx, \forall x \in \mathbb{R}^n \Rightarrow S = T$  by the uniqueness of the symmetric part in a quadratic form. Hence  $S$  is unique.

*Given S, find all A that yield  $\mathcal{E}_1 = \mathcal{E}_2$ .*

The answer is

$$A = Q\Lambda^{-\frac{1}{2}}V^\top$$

where  $V \in \mathbb{R}^{n \times n}$  is any orthogonal matrix and

$$S^\top = S = Q\Lambda Q^\top > 0$$

where  $Q$  is orthogonal and  $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n) > 0$ . Let the singular value decomposition of  $A$  be

$$A = U\Sigma V^\top$$

where  $U, V \in \mathbb{R}^{n \times n}$  are orthogonal and  $\Sigma = \text{diag}(\sigma_1, \dots, \sigma_n) > 0$  (since  $\det(A) \neq 0$ .) By (1),

$$\begin{aligned}\mathcal{E}_2 &= \left\{ y \mid y^\top (AA^\top)^{-1} y \leq 1 \right\} \\ &= \left\{ y \mid y^\top (U\Sigma^2 U^\top)^{-1} y \leq 1 \right\} \\ &= \left\{ y \mid y^\top U\Sigma^{-2} U^\top y \leq 1 \right\}\end{aligned}$$

Thus, if  $\mathcal{E}_1 = \mathcal{E}_2$ , then  $S = U\Sigma^{-2}U^\top$  by (2). Therefore  $U = Q$  and  $\Sigma = \Lambda^{-\frac{1}{2}}$ , and  $V$  can be any orthogonal matrix. You can also see why  $A$ 's are different only by right-side multiplication by an orthogonal matrix by the following argument. By (2) and (1),  $AA^\top = S^{-1}$ . Let

$$A = [\tilde{a}_1 \quad \tilde{a}_2 \quad \dots \quad \tilde{a}_n]^\top$$

Then we have,

$$\begin{aligned}\|\tilde{a}_i\|^2 &= (S^{-1})_{ii}, \\ \tilde{a}_i^\top \tilde{a}_j &= (S^{-1})_{ij},\end{aligned}$$

and

$$\cos \theta_{ij} = \frac{(S^{-1})_{ij}}{\sqrt{(S^{-1})_{ii}(S^{-1})_{jj}}}.$$

This means that the row vectors of any  $A$  satisfying  $AA^\top = S^{-1}$  have the same length and the same angle between any two of them. So the rows of  $A$  can vary only by the application of an identical rotation or reflection to all of them. These are the transformations preserving length and angle, and correspond to orthogonal matrices. Since we are considering row vectors, the orthogonal matrix should be multiplied on the right.

**15.2470. Uncovering a hidden linear explanation.** Consider a set of vectors  $y_1, \dots, y_N \in \mathbb{R}^n$ , which might represent a collection of measurements or other data. Suppose we have

$$y_i \approx Ax_i + b, \quad i = 1, \dots, N,$$

where  $A \in \mathbb{R}^{n \times m}$ ,  $x_i \in \mathbb{R}^m$ , and  $b \in \mathbb{R}^n$ , with  $m < n$ . (Our main interest is in the case when  $N$  is much larger than  $n$ , and  $m$  is smaller than  $n$ .) Then we say that  $y = Ax + b$  is a *linear explanation* of the data  $y$ . We refer to  $x$  as the vector of *factors* or *underlying causes* of the data  $y$ . For example, suppose  $N = 500$ ,  $n = 30$ , and  $m = 5$ . In this case we have 500 vectors; each vector  $y_i$  consists of 30 scalar measurements or data points. But these 30-dimensional data points can be ‘explained’ by a much smaller set of 5 ‘factors’ (the components of  $x_i$ ). This problem is about uncovering, or discovering, a linear explanation of a set of data, given only the data. In other words, we are given  $y_1, \dots, y_N$ , and the goal is to find  $m$ ,  $A$ ,  $b$ , and  $x_1, \dots, x_N$  so that  $y_i \approx Ax_i + b$ . To judge the accuracy of a proposed explanation, we’ll use the RMS explanation error, *i.e.*,

$$J = \left( \frac{1}{N} \sum_{i=1}^N \|y_i - Ax_i - b\|^2 \right)^{1/2}.$$

One rather simple linear explanation of the data is obtained with  $x_i = y_i$ ,  $A = I$ , and  $b = 0$ . In other words, the data explains itself! In this case, of course, we have  $y_i = Ax_i + b$ , so the RMS explanation error is zero. But this is not what we're after. To be a useful explanation, we want to have  $m$  substantially smaller than  $n$ , *i.e.*, substantially fewer factors than the dimension of the original data (and for this smaller dimension, we'll accept a nonzero, but hopefully small, value of  $J$ .) Generally, we want  $m$ , the number of factors in the explanation, to be as small as possible, subject to the constraint that  $J$  is not too large. Even if we fix the number of factors as  $m$ , a linear explanation of a set of data is not unique. Suppose  $A$ ,  $b$ , and  $x_1, \dots, x_N$  is a linear explanation of our data, with  $x_i \in \mathbb{R}^m$ . Then we can multiply the matrix  $A$  by two (say), and divide each vector  $x_i$  by two, and we have another linear explanation of the original data. More generally, let  $F \in \mathbb{R}^{m \times m}$  be invertible, and  $g \in \mathbb{R}^m$ . Then we have

$$y_i \approx Ax_i + b = (AF^{-1})(Fx_i + g) + (b - AF^{-1}g).$$

Thus,

$$\tilde{A} = AF^{-1}, \quad \tilde{b} = b - AF^{-1}g, \quad \tilde{x}_1 = Fx_1 + g, \quad \dots, \quad \tilde{x}_N = Fx_N + g$$

is another equally good linear explanation of the data. In other words, we can apply any affine (*i.e.*, linear plus constant) mapping to the underlying factors  $x_i$ , and generate another equally good explanation of the original data by appropriately adjusting  $A$  and  $b$ . To standardize or normalize the linear explanation, it is usually assumed that

$$\frac{1}{N} \sum_{i=1}^N x_i = 0, \quad \frac{1}{N} \sum_{i=1}^N x_i x_i^\top = I.$$

In other words, the underlying factors have an average value zero, and unit sample covariance. (You don't need to know what covariance is — it's just a definition here.) Finally, the problem.

- a) Explain clearly how you would find a hidden linear explanation for a set of data  $y_1, \dots, y_N$ . Be sure to say how you find  $m$ , the dimension of the underlying causes, the matrix  $A$ , the vector  $b$ , and the vectors  $x_1, \dots, x_N$ . Explain clearly why the vectors  $x_1, \dots, x_N$  have the required properties.
- b) Carry out your method on the data in the file `linexp_data.json` available on the course web site. The file gives the matrix  $Y = [y_1 \ \dots \ y_N]$ . Verify that  $y_i \approx Ax_i + b$  by calculating the norm of the error vector,  $\|y_i - Ax_i - b\|$ , for  $i = 1, \dots, N$ . Sort these norms in descending order and plot them. (This gives a good picture of the distribution of explanation errors.) By explicit (Julia) computation verify that the vectors  $x_1, \dots, x_N$  obtained, have the required properties.

### Solution.

- a) We have to find  $b$  and  $x_1, \dots, x_N$  that minimize

$$\tilde{J} = NJ^2 = \sum_{i=1}^N (y_i - Ax_i - b)^\top (y_i - Ax_i - b).$$

Taking the gradient with respect to  $b$  and setting it to zero gives,

$$\nabla_b(\tilde{J}) = \sum_{i=1}^N 2(y_i - Ax_i - b)(-1) = 0.$$

Since we want  $\sum_{i=1}^N x_i = 0$ , we get

$$b = \frac{1}{N} \sum_{i=1}^N y_i.$$

Let  $X$  be the matrix  $[x_1 \ \cdots \ x_N] \in \mathbb{R}^{m \times N}$ . Let  $z_i = y_i - b$ ,  $i = 1, \dots, N$ , and  $Z = [z_1 \ \cdots \ z_N] \in \mathbb{R}^{n \times N}$ . Note that  $Z$  is known from the data after  $b$  is calculated as shown above. The matrix  $(AX) \in \mathbb{R}^{n \times N}$  and has rank at most  $m$ . Then

$$\tilde{J} = \sum_{i=1}^N \|z_i - Ax_i\|^2 = \sum_{k=1}^n \sum_{j=1}^N (Z_{kj} - (AX)_{kj})^2 = \|Z - AX\|_F^2.$$

Thus minimizing  $\tilde{J}$  is minimizing the Frobenius norm of the matrix  $(Z - AX)$  where  $AX$  is at most rank  $m$ . Let the SVD of  $Z$  be  $Z = U\Sigma V^T$ , where  $U \in \mathbb{R}^{n \times r}$ ,  $\Sigma \in \mathbb{R}^{r \times r}$ ,  $V \in \mathbb{R}^{N \times r}$  and  $r$  is the rank of  $Z$ . The choice of  $m$  depends on the singular values  $\sigma_1, \dots, \sigma_r$  obtained for the particular data. A good choice of  $m$  would be when there is a significant jump in the singular values, *i.e.*,  $\sigma_m \gg \sigma_{m+1}$ ; or when the singular value becomes small enough ( $\sigma_{m+1}$  is negligible). Thus we pick a value for  $m$ . Then the  $m$  rank approximation to  $Z$  is

$$(AX) = \sum_{i=1}^m \sigma_i u_i v_i^T = U_m \Sigma_m V_m^T,$$

where  $U_m \in \mathbb{R}^{n \times m}$ ,  $\Sigma_m \in \mathbb{R}^{m \times m}$ ,  $V_m \in \mathbb{R}^{N \times m}$ . We pick  $A = \frac{1}{\sqrt{N}} U_m \Sigma_m$ , and  $x_i$  as  $\sqrt{N}$  times the  $i$ th column of  $V_m^T$ . Then

$$\frac{1}{N} \sum_{i=1}^N x_i x_i^T = \frac{1}{N} (\sqrt{N} V_m)^T (\sqrt{N} V_m) = I.$$

In order to show that  $\frac{1}{N} \sum_{i=1}^N x_i = 0$ , consider

$$Z\mathbf{1} = \sum_{i=1}^N z_i = \sum_{i=1}^N (y_i - b) = 0,$$

where  $\mathbf{1}$  is the vector of ones of size  $N$ . The vector  $\mathbf{1}$  is in the nullspace of  $Z$  which we can write as  $U\Sigma V^T \mathbf{1} = 0$ . The matrix  $U\Sigma$  is full rank, therefore  $V^T \mathbf{1} = 0$ . Hence  $V_m^T \mathbf{1} = 0$  as  $V_m$  are the first  $m$  columns of the matrix  $V$ . This means

$$\frac{1}{\sqrt{N}} V_m^T \mathbf{1} = \frac{1}{N} [x_1 \ \cdots \ x_N] \mathbf{1} = \frac{1}{N} \sum_{i=1}^N x_i = 0.$$

Thus we have found

$$b = \frac{1}{N} \sum_{i=1}^N y_i, \quad A = \frac{1}{\sqrt{N}} U_m \Sigma_m, \quad [x_1 \cdots x_N] = \sqrt{N} V_m^T,$$

with the required properties.

- b) The following Julia code implements solution method described in the part (a). We observe that there are 3 significant singular values, and therefore we take  $m = 3$ .

```
include("../..../julia_code/readclassjson.jl")
using Random, LinearAlgebra, Plots, LaTeXStrings

# uncovering the hidden linear explanations
data = readclassjson("../data/linexp_data.json")
Y = data["Y"]

n,N = size(Y)
# first we compute optimal b which is given by the average of y_i
b = sum(Y, dims=2)/N

# Define a matrix Z with the same shape as Y to be filled in the loop
Z = zeros(n,N)

# we subtract b from y_i, to get unbiased measurements
for i = 1:N
    Z[:, i] = Y[:, i] - b
end
# in Julia, S is a vector so we use Diagonal to make it a diagonal matrix
F = svd(Z)
U, S, V = F.U, Diagonal(F.S), F.V

# the singular values of Z are
# >> diag(S)
#
# 6.1795
# 1.8451
# 1.0226
# 0.0434
# 0.0131
# 0.0114
# 0.0084
# 0.0060
# 0.0047
# 0.0021
#
# and we pick the top three which are dominant
```

```

A = 1.0/sqrt(N)*U[:, 1:3]*S[1:3, 1:3]
# X = V' and svd in Matlab gives U, S, V for Z = USV'
X = sqrt(N)*V[:, 1:3]'

# check that X obeys constraints
@show round.(sum(X, dims=2) / N, digits=2)
@show round.(X*X'/N, digits=2)

# Now let's compute the error
error = Z - A*X
errNorm = zeros(N)
for i=1:N
    errNorm[i] = norm(error[:, i])
end

# primary = false means no legend label
plot(-sort(-errNorm), primary=false)
axis!([1 100 0 0.012])
xlabel!(L"sorted index  $\hat{i}$ ")
ylabel!(L" $|y_i - Ax_i - b|$ ");
savefig("../graphics/linexp_errnorm.pdf")

```

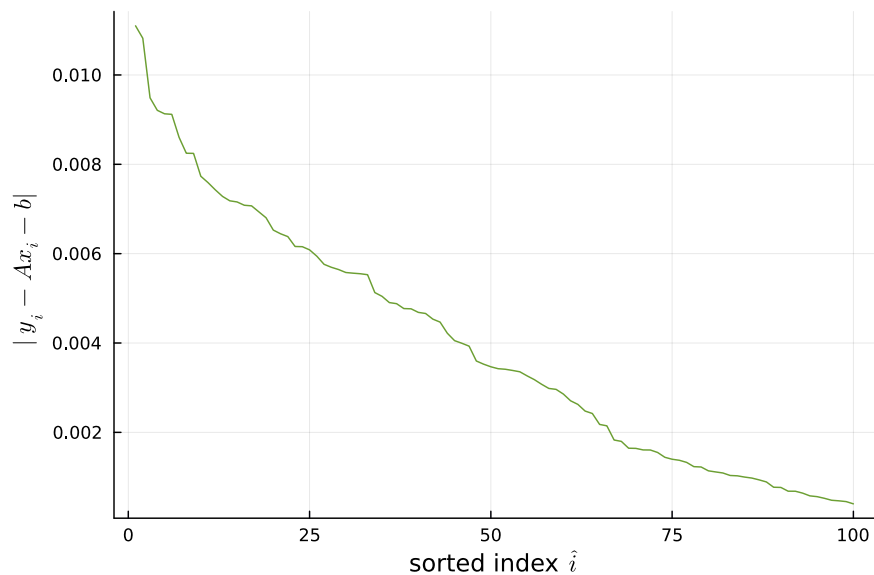
Here is explicit check shows that  $X$  satisfies given properties.

```

round.(sum(X, dims = 2) / N, digits = 2) = [0.0; -0.0; 0.0;;]
round.((X * X') / N, digits = 2) = [1.0 -0.0 0.0; -0.0 1.0 0.0; 0.0 0.0 1.0]

```

Plot of the sorted error norm is shown below:



**16.2570. Alternating projections for low rank matrix completion.** In the *low rank matrix completion problem*, you are given *some* of the entries of a matrix, along with an upper bound on its rank; you are to guess or estimate the remaining entries. This arises in several applications, one of which is described at the end of this problem. This question investigates a heuristic method for the low rank matrix completion problem.

You are told that  $A \in \mathbb{R}^{m \times n}$  has  $\text{rank} \leq r$ , and that  $A_{ij} = A_{ij}^{\text{known}}$  for  $(i, j) \in \mathcal{K}$ , where  $\mathcal{K} \subseteq \{1, \dots, m\} \times \{1, \dots, n\}$  is the set of indices of the known entries. (You are given  $A_{ij}^{\text{known}}$  for  $(i, j) \in \mathcal{K}$ .) We let  $p = |\mathcal{K}|$  denote the number of known entries. You are to estimate or guess the entries  $A_{ij}$ , for  $(i, j) \notin \mathcal{K}$ .

You will use an alternating projection method to find an estimate  $\hat{A}$  of  $A$ . After choosing an initial point  $\hat{A}^{(0)}$ , that has the known correct entries (*i.e.*,  $\hat{A}_{ij}^{(0)} = A_{ij}^{\text{known}}$  for  $(i, j) \in \mathcal{K}$ ), you will alternate between two projections. For  $k = 0, 1, \dots$  you carry out the following steps:

- *Project to the closest matrix satisfying the rank constraint.* Set  $\tilde{A}^{(k)}$  to be the matrix of rank  $\leq r$  that is closest to  $\hat{A}^{(k)}$  in Frobenius norm, *i.e.*, that minimizes

$$\|\tilde{A}^{(k)} - \hat{A}^{(k)}\|_F = \left( \sum_{i=1}^m \sum_{j=1}^n (\tilde{A}_{ij}^{(k)} - \hat{A}_{ij}^{(k)})^2 \right)^{1/2}$$

subject to  $\text{rank}(\tilde{A}^{(k)}) \leq r$ .

- *Project to the closest matrix with the known entries.* Set  $\hat{A}^{(k+1)}$  to be the matrix with the given known entries that is closest to  $\tilde{A}^{(k)}$  in Frobenius norm, *i.e.*, that minimizes

$$\|\hat{A}^{(k+1)} - \tilde{A}^{(k)}\|_F = \left( \sum_{i=1}^m \sum_{j=1}^n (\hat{A}_{ij}^{(k+1)} - \tilde{A}_{ij}^{(k)})^2 \right)^{1/2}$$

subject to  $\hat{A}_{ij}^{(k+1)} = A_{ij}^{\text{known}}$  for  $(i, j) \in \mathcal{K}$ .

This is a heuristic method: It can fail to converge at all, or it can converge to different limit points, depending on the starting point. But it often works well.

- Clearly explain how to perform each of these projections. We will subtract points for technically correct, but overly complicated methods. Do not use any Julia notation in your answer.
- Use 300 steps of the alternating projections algorithm to find a low rank matrix completion estimate for the problem defined in `low_rank_matrix_completion.json`. This file defines the rank upper bound  $r$ , the dimensions  $m$  and  $n$ , and the known matrix entries. The known matrix indices are given as a  $p \times 2$  matrix `K`, with each row giving  $(i, j)$  for one known entry. The  $p$ -vector `Aknown` gives the corresponding known values.

Initialize your method as follows. Let  $\mu$  denote the mean of all the known entries. Set  $\hat{A}_{ij}^{(0)} = A_{ij}^{\text{known}}$  for  $(i, j) \in \mathcal{K}$ , and  $\hat{A}_{ij}^{(0)} = \mu$  for  $(i, j) \notin \mathcal{K}$ .

To judge the performance of the algorithm, the data file also gives the actual matrix  $A$  as `Atrue`. (Of course in applications, you would not have access to the matrix  $A$ !) Plot  $\|\hat{A}^{(k)} - A\|_F$ , for  $k = 1, \dots, 300$ .

Make a very brief comment about how well the algorithm worked on this data set. You can allude to the fact that you are given only around one sixth of the entries of  $A$ .

**Remark.** *None of this is needed to solve the problem; it is only for your amusement and interest.* Algorithms like this can be used for problems like the *Netflix challenge*. Here  $A_{ij}$  represents the rating user  $i$  gives (or would give) to movie  $j$ . We have access to some of the ratings, and want to predict other ratings before they are given. (This would allow us to make recommendations, for example.) It is reasonable to assume (and is confirmed with real data) that ratings matrices like  $A$  have (approximately) low rank. This can be interpreted as meaning that a user's rating is (mostly) determined by a relatively small number of factors. The entries in the  $k$ th left singular vector tell us how much the user ratings are influenced (positively or negatively) by factor  $k$ ; the entries in the  $k$ th right singular vector tell us how much of factor  $k$  (positive or negative) is in each movie.

### Solution.

- a) To project  $\hat{A}^{(k)}$  to the closest matrix with the given rank, we use the SVD,  $\hat{A}^{(k)} = \sum_i \sigma_i u_i v_i^\top$ . The projection is then given by the SVD, truncated to  $r$  terms:

$$\tilde{A}^{(k)} = \sum_{i=1}^r \sigma_i u_i v_i^\top.$$

To project  $\tilde{A}^{(k)}$  to the closest matrix with the known entries is very simple. We simply set the known entries of  $\tilde{A}^{(k)}$  back to the given values:

$$\hat{A}_{ij}^{(k+1)} = \begin{cases} \tilde{A}_{ij}^{(k)}, & (i, j) \in \mathcal{K} \\ A_{ij}^{\text{known}} & (i, j) \notin \mathcal{K}. \end{cases}$$

This can be seen several ways. The easiest is to note that

$$\|\hat{A}^{(k+1)} - \tilde{A}^{(k)}\|_F^2 = \sum_{i,j} \left( \hat{A}_{ij}^{(k+1)} - \tilde{A}_{ij}^{(k)} \right)^2$$

is separable, *i.e.*, a sum of terms, each of which involves only one of the variables we are to choose (*i.e.*, the  $\hat{A}_{ij}^{(k+1)}$ ). So we can minimize each term in this expression separately. To minimize  $\left( \hat{A}_{ij}^{(k+1)} - \tilde{A}_{ij}^{(k)} \right)^2$ , with no constraints on  $\hat{A}_{ij}^{(k+1)}$  is easy—just take  $\hat{A}_{ij}^{(k+1)} = \tilde{A}_{ij}^{(k)}$ . To minimize  $\left( \hat{A}_{ij}^{(k+1)} - \tilde{A}_{ij}^{(k)} \right)^2$ , subject to  $\hat{A}_{ij}^{(k+1)} = A_{ij}^{\text{known}}$  is also easy, because there's no freedom of choice here—we must take  $\hat{A}_{ij}^{(k+1)} = A_{ij}^{\text{known}}$ .

- b) The following Julia code runs the alternating projection algorithm on the given data.

```
include("../..../julia_code/readclassjson.jl")
using Random, LinearAlgebra, Plots

function solution()
```

```

data = readclassjson("../data/low_rank_matrix_completion.json")
Aknown = data["Aknown"]
Atrue = data["Atrue"]
K = data["K"]
p = data["p"]
m = data["m"]
n = data["n"]
r = data["r"]

mu = sum(Aknown)/p; # average of known entries
Ahat = mu*ones(m,n);
for i = 1:p
    Ahat[K[i,1],K[i,2]] = Aknown[i]
end

# store historic norm values.
normhist = [norm(Ahat-Atrue, 2)]

for k = 1:300
    # SVD of Ahat
    F = svd(Ahat)
    U, S, Vt = F.U, F.S, F.Vt
    # In Julia, unlike in Matlab, S comes as a vector
    S = Diagonal(S)
    # Take the rank r approximation of A
    Ahat = U[:,1:r] * S[1:r,1:r] * Vt[1:r,:]
    # nearest matrix with given known entries
    for i = 1:p
        Ahat[K[i,1],K[i,2]] = Aknown[i]
    end
    push!(normhist, norm(Ahat-Atrue, 2))
end

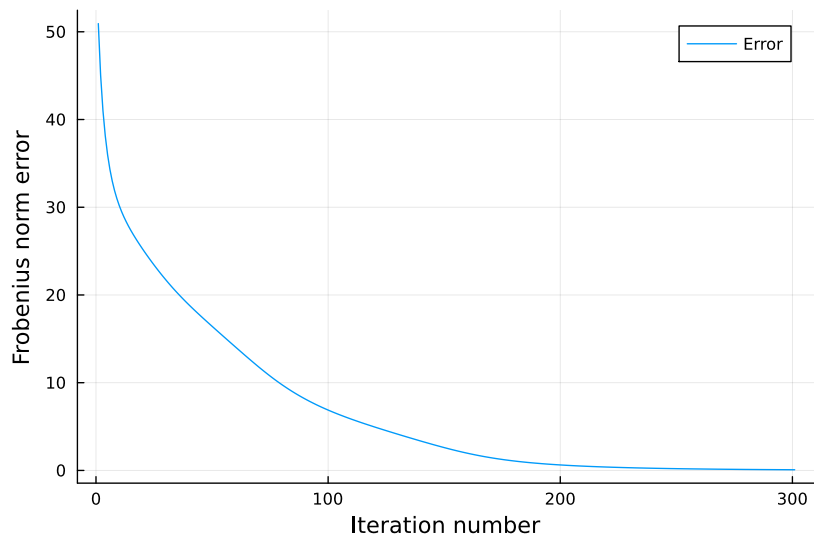
p = plot(normhist, label="Error")
xlabel!("Iteration number")
ylabel!("Frobenius norm error")
savefig(p, "../graphics/low_rank_matrix_completion.pdf")
end

solution()

```

This produces the plot shown below. Evidently the error reaches a small value (for

example, compared to  $\|A\|_F$ ).



Here is the brief comment: For this example, we nailed it! It seems the algorithm recovers  $A$  perfectly. This is amazing, considering that we started with only around one sixth of the entries known, and from these, we reconstructed the remaining five sixths using the alternating projections algorithm.

It goes without saying that it doesn't always work this well.