Chapter 6

Low-rank approximation

Contents (class version)

6.0 Introduction

L§8.1

In many applications, **dimensionality reduction** is important. This chapter focuses on **low-rank approximation** of a matrix. There are theoretical models for why big matrices should be approximately low rank [1].
One of many applications is **image compression**, explored in HW.
Another application is source localization via multidimensional scaling, explored on p. 6.24.

Source material for this chapter includes [2, §8.1].

6.1 Low-rank approximation via Frobenius norm

We are given a matrix $\boldsymbol{A} \in \mathbb{F}^{M \times N}$ (often large), having rank $r \leq \min(M, N)$.
To perform **dimensionality reduction** we want to approximate $\boldsymbol{A}$ by another matrix $\hat{\boldsymbol{A}}_K$ having rank $K \leq r$.
Often we pick $K \ll r$.

To find the "best" $\hat{\boldsymbol{A}}_K$ we must define how closely $\hat{\boldsymbol{A}}_K$ approximates $\boldsymbol{A}$. The simplest metric is the **Frobenius norm** of the difference. This criterion leads to the follow **low-rank approximation problem**:

$$\hat{\boldsymbol{A}}_K \triangleq \qquad \mathcal{L}_K^{M \times N} \triangleq \left\{ \boldsymbol{B} \in \mathbb{F}^{M \times N} : \mathsf{rank}(\boldsymbol{B}) \leq K \right\}. \tag{6.1}$$

This is a **non-convex** optimization problem because the set $\mathcal{L}_K^{M \times N}$ of rank-K matrices is not convex.

Why rank is a non-convex function

To see why $\mathsf{rank}(\cdot)$ is a non-convex function, and $\mathcal{L}_K^{M \times N}$ is a nonconvex set, consider:

$$\boldsymbol{A} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \; \boldsymbol{B} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, \; \alpha \in (0,1) \implies \mathsf{rank}(\alpha \boldsymbol{A} + (1-\alpha)\boldsymbol{B}) = \mathsf{rank}\left(\begin{bmatrix} \alpha & 0 \\ 0 & 1-\alpha \end{bmatrix} \right) = 2 \not\leq$$
$$\alpha \, \mathsf{rank}(\boldsymbol{A}) + (1-\alpha) \, \mathsf{rank}(\boldsymbol{B}) = \alpha \cdot 1 + (1-\alpha) \cdot 1 = 1.$$

Very remarkably, despite this non-convexity, there is a simple solution for $\hat{\boldsymbol{A}}_K$ based on any **SVD** of $\boldsymbol{A}$ [3].

Theorem. The best rank (at most) K approximation uses the first (largest) K singular components of $\boldsymbol{A}$:

$$\hat{\boldsymbol{A}}_K = \qquad \text{where } \boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}' = \sum_{k=1}^{\min(M,N)} \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k'. \tag{6.2}$$

The approximation error depends on the singular values of the remaining (discarded) terms:

$$\left\| \hat{\boldsymbol{A}}_K - \boldsymbol{A} \right\|_{\mathrm{F}} = \qquad \leq \|\boldsymbol{B} - \boldsymbol{A}\|_{\mathrm{F}}, \; \forall \boldsymbol{B} \in \mathcal{L}_K^{M \times N}.$$

- Clearly the approximation error will be small if the remaining singular values are small compared to the first K singular values.
- Once again we see that the SVD is key tool. Here, the SVD is used both to construct the approximation and to quantify the approximation error.
- The original matrix $\boldsymbol{A}$ has MN values. The approximation $\hat{\boldsymbol{A}}_K$ is formed from $K(M+N+1)$ values. This saves memory if $K \ll \min(M,N)$.

What is the rank of $\sum_{k=1}^{K} \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k'$ when $1 \leq K \leq r = \mathsf{rank}(\boldsymbol{A})$? (Choose best answer.)

A: Always 1 B: Always r C: Usually r D: Always K E: Usually K **??**

Diagonal case: proof sketch

First consider a $M \times N$ (rectangular) diagonal matrix $\boldsymbol{\Sigma}$ having rank r, with descending diagonal values, where we want to approximate it by a matrix $\boldsymbol{C}$ of rank at most $K \leq r$, *i.e.*, we want to solve:

$$\hat{\boldsymbol{C}} \triangleq \underset{\boldsymbol{C} \in \mathcal{L}_K^{M \times N}}{\arg\min} \left\| \boldsymbol{C} - \boldsymbol{\Sigma} \right\|_{\mathrm{F}}^2 = \underset{\boldsymbol{C} \in \mathcal{L}_K^{M \times N}}{\arg\min} \left\| \begin{bmatrix} c_{11} & \dots & c_{1N} \\ & \vdots & \\ c_{M1} & \dots & c_{MN} \end{bmatrix} - \begin{bmatrix} \sigma_1 & & & & & & \\ & \ddots & & & & & \\ & & \sigma_K & & & & \\ & & & \sigma_{K+1} & & & \\ & & & & \ddots & & \\ & & & & & \sigma_r & \\ & & & & & & \mathbf{0} \end{bmatrix} \right\|_{\mathrm{F}}^2$$

$$= \underset{\boldsymbol{C} \in \mathcal{L}_K^{M \times N}}{\arg\min} \sum_{k=1}^{K} (c_{kk} - \sigma_k)^2 + \sum_{k=K+1}^{r} (c_{kk} - \sigma_k)^2 + \sum_{k=r+1}^{\min(M,N)} (c_{kk} - 0)^2 + \sum_{m \neq n} (c_{mn} - 0)^2.$$

To minimize this quadruple sum, subject to the constraint $\mathsf{rank}(\boldsymbol{C}) \leq K$, intuitively we prefer the last two terms to be zero. And to minimize the first two terms, because $\sigma_1 \geq \sigma_2 \geq \dots$, it seems natural to keep the first K (using $c_{kk} = \sigma_k$) and set the rest to zero, yielding the following solution:

$$\hat{c}_{mn} = \begin{cases} \sigma_m, & m = n \leq K \\ 0, & \text{otherwise} \end{cases} \implies \hat{\boldsymbol{C}} = \begin{bmatrix} \sigma_1 & & & & \\ & \ddots & & & \\ & & \sigma_K & & \\ & & & \mathbf{0}_{r-K \times r-K} & \\ & & & & \mathbf{0}_{M-r \times N-r} \end{bmatrix} = \sum_{k=1}^{K} \sigma_k \boldsymbol{e}_k \tilde{\boldsymbol{e}}_k', \qquad (6.3)$$

where $\boldsymbol{e}_k$ is the kth unit vector in $\mathbb{F}^M$ and $\tilde{\boldsymbol{e}}_k$ is the kth unit vector in $\mathbb{F}^N$.
The conclusion (6.3) is correct, but the above reasoning is not rigorous. For a rigorous proof, see [wiki] [4].

Proof for general case

Now we assume that (6.3) is correct (it is, though not proven here), and use it to prove the general case.

Denote an SVD of $\boldsymbol{A}$ by $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}'$. Rewrite any $\boldsymbol{B} \in \mathbb{F}^{M \times N}$ in terms of the $\boldsymbol{U}$ and $\boldsymbol{V}$ bases as follows:

$$\boldsymbol{B} = \underbrace{(\boldsymbol{U}\boldsymbol{U}')}_{\boldsymbol{I}} \boldsymbol{B} \underbrace{(\boldsymbol{V}\boldsymbol{V}')}_{\boldsymbol{I}} = \boldsymbol{U} \underbrace{(\boldsymbol{U}'\boldsymbol{B}\boldsymbol{V})}_{\hookrightarrow \triangleq \boldsymbol{C} \text{ (not diagonal in general)}} \boldsymbol{V}' = \boldsymbol{U}\boldsymbol{C}\boldsymbol{V}'. \tag{6.4}$$

Because $\boldsymbol{U}$ and $\boldsymbol{V}$ are unitary, $\mathsf{rank}(\boldsymbol{B}) = \mathsf{rank}(\boldsymbol{C})$, so (6.1) is equivalent to

$$\hat{\boldsymbol{A}}_K = \boldsymbol{U}\hat{\boldsymbol{C}}\boldsymbol{V}', \quad \hat{\boldsymbol{C}} \triangleq \underset{\boldsymbol{C} \in \mathcal{L}_K^{M \times N}}{\arg\min} \left\| \underbrace{\boldsymbol{U}\boldsymbol{C}\boldsymbol{V}'}_{\boldsymbol{B}} - \underbrace{\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}'}_{\boldsymbol{A}} \right\|_{\mathrm{F}} = \underset{\boldsymbol{C} \in \mathcal{L}_K^{M \times N}}{\arg\min} \left\| \boldsymbol{U}(\boldsymbol{C} - \boldsymbol{\Sigma})\boldsymbol{V}' \right\|_{\mathrm{F}}$$

$$= \underset{\boldsymbol{C} \in \mathcal{L}_K^{M \times N}}{\arg\min} \left\| \boldsymbol{C} - \boldsymbol{\Sigma} \right\|_{\mathrm{F}} = \sum_{k=1}^{K} \sigma_k \boldsymbol{e}_k \tilde{\boldsymbol{e}}_k',$$

using the result (6.3) for the diagonal case and the fact that the **Frobenius norm** is **unitarily invariant**.

Thus the final best approximation to $\boldsymbol{A}$ having rank (at most) K is:

$$\hat{\boldsymbol{A}}_K = \boldsymbol{U}\hat{\boldsymbol{C}}\boldsymbol{V}' = \sum_{k=1}^{K} \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k', \text{ where } \boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}' = \sum_{k=1}^{r} \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k',$$

because $\boldsymbol{U}\boldsymbol{e}_k = \boldsymbol{u}_k$. The approximation error depends on the remaining singular values:

$$\left\| \hat{\boldsymbol{A}}_K - \boldsymbol{A} \right\|_{\mathrm{F}} =$$

Alternative way to derive the approximation error expression: (Read)

$$\boldsymbol{A} - \hat{\boldsymbol{A}}_K = \sum_{k=1}^{r} \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k' - \sum_{k=1}^{K} \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k' = \sum_{k=K+1}^{r} \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k' \implies$$

$$\left\| \hat{\boldsymbol{A}}_K - \boldsymbol{A} \right\|_{\mathrm{F}} = \left\| \sum_{k=K+1}^{r} \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k' \right\|_{\mathrm{F}} = \left\| \boldsymbol{U}' \sum_{k=K+1}^{r} \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k' \boldsymbol{V} \right\|_{\mathrm{F}} = \left\| \sum_{k=K+1}^{r} \sigma_k \boldsymbol{e}_k \boldsymbol{e}_k' \right\|_{\mathrm{F}} = \sqrt{\sum_{k=K+1}^{r} \sigma_k^2},$$

which again is related to **Parseval's theorem**. □

Summary

In words: the best rank (at most) K approximation to a matrix is given by its **truncated SVD** (to K terms).

Implementation

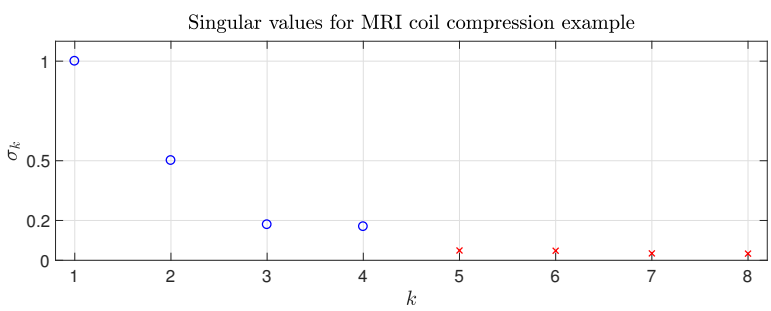
For implementation, we need to choose $K \le r = \mathsf{rank}(\boldsymbol{A}) \le \min(M, N)$.

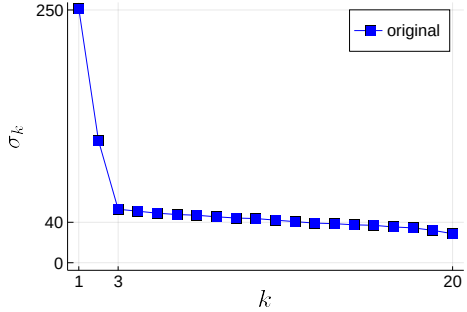
The approximation error is given by the 2-norm of the excluded singular values $(\sigma_{K+1}, \dots, \sigma_r)$, so it is useful to plot all the singular values versus k and look for a "knee in the curve."

Low-rank approximation is related to **factor analysis**, and a **scree plot** shows the singular values [5].

Example.

(Left figure from MRI example on p. 6.12; right figure from a textbook example with $r = 2$ and noise.)





You hope that your scree plot has a roughly linear section corresponding to the noise.

For moderate sized matrices, low-rank approximation needs just a few lines of JULIA code.
The simplest approach uses the sum of outer products solution (6.2) literally:

```
using LinearAlgebra: svd
(U, s, V) = svd(A)
B = zeros(size(A))
for k=1:K
  global B += U[:,k] * s[k] * V[:,k]' # sum of rank-1 outer products
end
```

The `global` declaration is unnecessary if the code is within a function.

To avoid a loop one can use matrix multiplication corresponding to the **compact SVD** form in (6.2):

```
using LinearAlgebra: svd, Diagonal
(U, s, V) = svd(A) # default full=false saves memory
B = U[:,1:K] * Diagonal(s[1:K]) * V[:,1:K]'
```

Why small `s` above? Reminds me that output is a vector, not a matrix.

In both versions, the final `B` is $\hat{\boldsymbol{A}}_K$.

Quality code would also include a bounds check that $1 \leq K \leq \min(M, N)$.

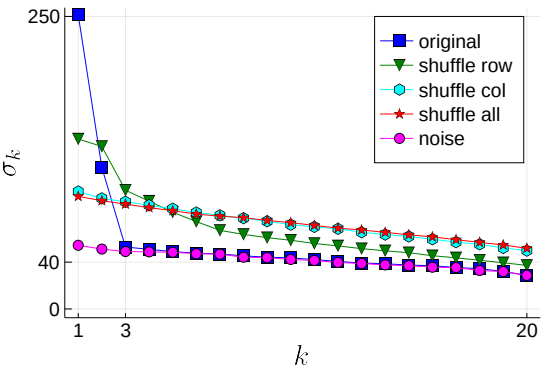
Yet another option is to compute only the first K SVD components using `svds`:

```
using LinearAlgebra: Diagonal
using Arpack: svds
tmp = svds(A, nsv=K, ritzvec=true)[1] # special SVD data structure
(U, s, Vt) = (tmp.U, tmp.S, tmp.Vt) # extract needed components
B = U * Diagonal(s) * Vt
```

Why `Vt` above? Avoids an unnecessary transpose operation.

Choosing rank via permutation

Looking for a jump in the **scree plot** seems subjective. A more quantitative approach uses a **permutation method** [6].
Consider a matrix that is the sum of a low-rank component and IID random noise. If we permute (randomly) each of the columns (or rows? or all?) of that matrix, the noise part will remain the same (statistically) but the low-rank structure will be destroyed. Now we plot the singular values of this shuffled matrix, which will basically correspond to noise. The singular values of the original data that are larger than those of the shuffled data might be considered to be the "significant" ones and can help guide rank choice. See [6] for theoretical analysis. One might need to permute multiple times and average to get a reliable baseline.



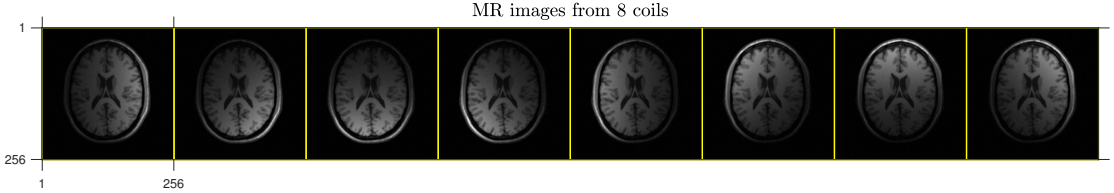
Example. This is from synthetic data that is rank 2 plus noise.
The two largest singular values stand out above the shuffled case.

Here is the key code for permuting each column:

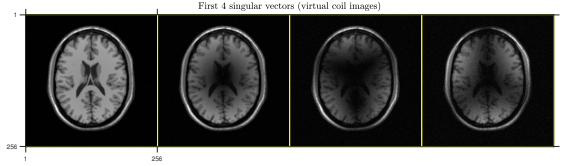
```
using Random: shuffle
Y = mapslices(shuffle, X, dims=1) # shuffle each column
Y = reshape(shuffle(X[:]), size(X)) # or, shuffle all
```

MRI Example

Modern MRI scanners use multiple receive coils to collect more image information at the same time. For example, the fMRI Center on UM's North Campus uses a 32-channel head coil. Processing all 32 sets of 3D images can require undesirably large computation times, and typically the data recorded by 32 coils is close to being linearly dependent. A technique called **coil compression** reduces data size [7, 8], essentially by making a **low-rank** approximation to the data and then just storing and processing the K singular vectors. Here are 8 MRI brain images recorded simultaneously by 8 coils around the head:



The scree plot on p. 6.8 suggests that most of the information is in the first 4 components. We reshape the data into a $(M = 256^2 \times N = 8)$ matrix, use an SVD to find the rank-4 approximation, and reshape the first $K = 4$ vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_4$ into 256×256 "virtual coil" images for display:



(In practice this operation is done on the raw data before making images.)

Non-uniqueness of SVD and low-rank approximation

As noted previously, "the" SVD of a matrix $\boldsymbol{A}$ is not unique. In particular, if two singular values are the same, then one can swap the corresponding columns of $\boldsymbol{U}$ and $\boldsymbol{V}$ (or perform a more general rotation of their subspaces) and have two equally valid SVD expressions. Mathematically, when we write $\boldsymbol{A} = \sum_{k=1}^{\min(M,N)} \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k'$, the order of the terms in the sum is unique if $\sigma_1 > \sigma_2 > \ldots > \sigma_{\min(M,N)}$, but if any of the singular values are identical (including two or more zero singular values) then the ordering is not unique, and one can swap corresponding columns of $\boldsymbol{U}$ and $\boldsymbol{V}$.

Example. Here are two different SVDs for a diagonal matrix with $7 = \sigma_1 > \sigma_2 = \sigma_3 = 5$:

$$\underbrace{\begin{bmatrix} 7 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 5 \end{bmatrix}}_{\boldsymbol{A}} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{U}} \underbrace{\begin{bmatrix} 7 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 5 \end{bmatrix}}_{\boldsymbol{\Sigma}} \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{V}'} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}}_{\tilde{\boldsymbol{U}}} \underbrace{\begin{bmatrix} 7 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 5 \end{bmatrix}}_{\boldsymbol{\Sigma}} \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}}_{\tilde{\boldsymbol{V}}'}.$$

The example on the next page shows that there are two distinct rank $K = 2$ approximations to this $\boldsymbol{A}$.

Even if all the singular values are distinct, "the" SVD is still not unique because we can multiply both $\boldsymbol{u}_k$ and $\boldsymbol{v}_k$ by $\mathrm{e}^{\imath\phi}$ and get a "different" $\boldsymbol{U}$ and $\boldsymbol{V}$. However, the low-rank approximation is still the same in this case because $\sigma_k (\mathrm{e}^{\imath\phi} \boldsymbol{u}_k)(\mathrm{e}^{\imath\phi} \boldsymbol{v}_k)' = \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k'$.

We have not focused on this issue too much yet because often uniqueness is relatively unimportant.

Now consider the question of whether the rank-K approximation of $\boldsymbol{A}$ is **unique**. This is simpler to answer.

- If $\sigma_K > \sigma_{K+1}$ then one can extend the proof on p. 6.6 to show that the rank-K approximation is **unique**.
- If $\sigma_K = \sigma_{K+1}$, then the rank-K approximation is *not unique* because we could swap the Kth and $(K+1)$ terms.

Example. Here are two distinct rank-1 approximations to a simple diagonal matrix:

$$\boldsymbol{A} \triangleq \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\boldsymbol{A} \approx \boldsymbol{B}_1 \triangleq \sigma_1 \boldsymbol{u}_1 \boldsymbol{v}_1' = 5 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix} = \begin{bmatrix} 5 & 0 \\ 0 & 0 \end{bmatrix}, \qquad \boldsymbol{A} \approx \boldsymbol{B}_2 \triangleq \sigma_1 \tilde{\boldsymbol{u}}_1 \tilde{\boldsymbol{v}}_1' = 5 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 5 \end{bmatrix}.$$

What is the approximation error $\|\boldsymbol{A} - \boldsymbol{B}_1\|_{\mathrm{F}}$ in the preceding example?

A: 0 B: $\sqrt{5}$ C: 5 D: 10 E: 5^2 **??**

In practice, any rank-K approximation (for suitable K) often is equally useful, so one infrequently worries about non-uniqueness.

Furthermore, in practical finite precision computing, it is dubious to test whether two floating-point values σ_K and σ_{K+1} are "exactly" equal. Avoid code like `if (a == b)` with floating-point numbers!

For analysis of the **numerical stability** of LR matrix approximation, including consideration of the "singular value gap" $\sigma_K - \sigma_{K+1}$, see [9].

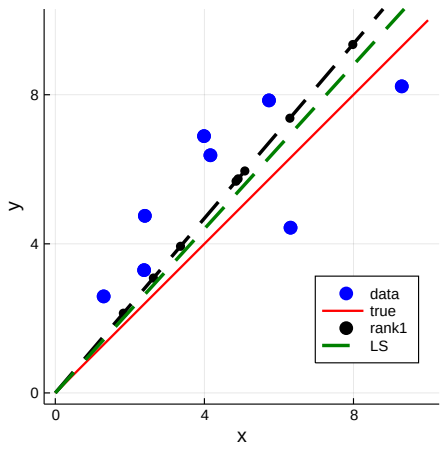
1D example

Here we consider a 2×9 matrix where each column is an (x_n, y_n) coordinate for $n = 1, \dots, N = 9$, *i.e.*, $\boldsymbol{A} = \begin{bmatrix} x_1 & \dots & x_9 \\ y_1 & \dots & y_9 \end{bmatrix}$. Based on the picture below, what is $\mathsf{rank}(\boldsymbol{A})$? **??**

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/06_low_rank1.html
https://web.eecs.umich.edu/~fessler/course/551/julia/demo/06_low_rank1.ipynb

Suppose we want to perform **dimensionality reduction** of the 2D data by reducing it to a 1D line.

- One option for processing this data is to perform a **rank-1 approximation**, leading to dashed black line in the figure. The black points are the nearest points in the subspace for each of the data points.
- An alternative is to perform a **linear least-squares** fit assuming $y_n \approx \alpha x_n$. This leads to the green line with slope $\hat{\alpha}$ in the figure.

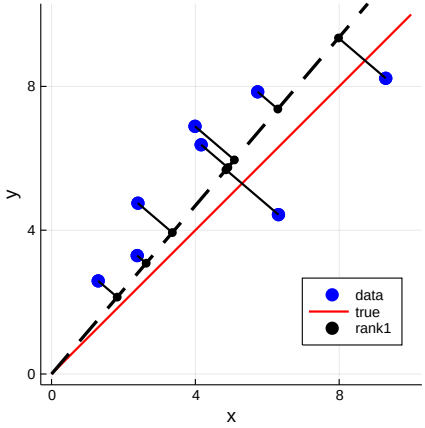


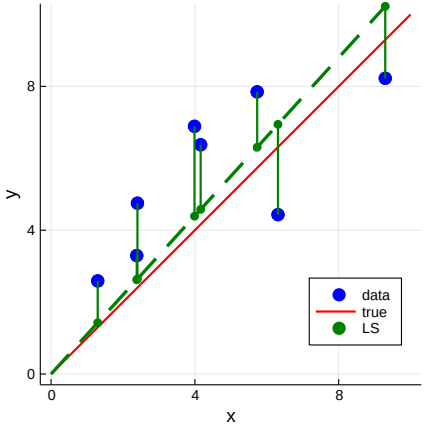
You might find it surprising that these two methods lead to different fits.

The **rank-1 approximation** is $\arg\min_{\boldsymbol{B} \in \mathcal{L}_1^{M \times N}} \left\| \begin{bmatrix} x_1 & \dots & x_N \\ y_1 & \dots & y_N \end{bmatrix} - \boldsymbol{B} \right\|_{\mathrm{F}}^2 = \arg\min_{\boldsymbol{B} \in \mathcal{L}_1^{M \times N}} \sum_{n=1}^{N} \left\| \begin{bmatrix} x_n \\ y_n \end{bmatrix} - B_{:,n} \right\|_2^2.$

The **linear least-squares** estimator is $\arg\min_{\alpha} \left\| \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} - \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} \alpha \right\|_2^2 = \arg\min_{\alpha} \sum_{n=1}^{N} |y_n - \alpha\, x_n|^2.$

These two methods measure approximation error differently, as illustrated in the following two plots.





Generalization to other norms

Theorem **(Eckart-Young-Mirsky)** (See [4] and [10] for a proof.)
For any **unitarily invariant** matrix norm $\|\!|\cdot|\!\|_{\text{UI}}$, the **low-rank approximation problem** has the same solution using the first (largest) K singular components of $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}' = \sum_{k=1}^{r} \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k'$:

$$\hat{\boldsymbol{A}}_K \triangleq \underset{\boldsymbol{B} \in \mathcal{L}_K^{M \times N}}{\arg\min} \underbrace{\|\!| \boldsymbol{B} - \boldsymbol{A} |\!\|_{\text{UI}}}_{\hookrightarrow \text{ unitarily invariant norm}} = \sum_{k=1}^{K} \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k'.$$

So even though the proof sketch on earlier pages was for the Frobenius norm, remarkably the same result holds (with a different proof) for other unitarily invariant norms such as the **spectral norm**.

To be robust to data outliers, sometimes one prefers other norms that are *not* unitarily invariant [11, 12], especially $\|\cdot\|_1$. Those other norms require different solution methods (typically iterative algorithms). SVD operations are still used as one part of many such algorithms [12].

What is $\left\| \hat{\boldsymbol{A}}_K - \boldsymbol{A} \right\|_2$ when $\boldsymbol{A}$ has rank r ?

A: $\sqrt{\sum_{k=K+1}^{r} \sigma_k^2}$ B: $\sum_{k=K+1}^{r} \sigma_k$ C: σ_{K+1} D: 0 E: None of these **??**

What is $\left\| \hat{\boldsymbol{A}}_K - \boldsymbol{A} \right\|_*$? (Choose from same answer list.) **??**

Proof for spectral norm (Read)

Claim: $\mathsf{rank}(\boldsymbol{B}) \le K \implies \left\| \boldsymbol{A} - \hat{\boldsymbol{A}}_K \right\|_2^2 \le \|\boldsymbol{A} - \boldsymbol{B}\|_2^2$ where $\hat{\boldsymbol{A}}_K \triangleq \sum_{k=1}^{K} \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k'$.

Proof. Let $\boldsymbol{W} = \begin{bmatrix} \boldsymbol{v}_1 & \dots & \boldsymbol{v}_{K+1} \end{bmatrix}$ so $\dim(\mathcal{R}(\boldsymbol{W})) = K+1$ because columns of $\boldsymbol{W}$ are orthonormal.

$\mathsf{rank}(\boldsymbol{B}) \le K \implies \dim(\mathcal{N}(\boldsymbol{B})) \ge N - K \implies \dim(\mathcal{N}(\boldsymbol{B})) + \dim(\mathcal{R}(\boldsymbol{W})) \ge (N-K) + (K+1) = N+1.$
But both $\mathcal{N}(\boldsymbol{B})$ and $\mathcal{R}(\boldsymbol{W})$ are subspaces in $\mathbb{F}^N$, so they must have a nontrivial intersection, by (3.4).
Thus there exists some $\boldsymbol{x} \neq \boldsymbol{0}$ such that $\boldsymbol{x} \in \mathcal{N}(\boldsymbol{B})$ and $\boldsymbol{x} \in \mathcal{R}(\boldsymbol{W})$. WLOG take $\|\boldsymbol{x}\|_2 = 1$.

$\boldsymbol{x} \in \mathcal{R}(\boldsymbol{W}) \implies \boldsymbol{x} = \boldsymbol{W}\boldsymbol{z}$ for $\boldsymbol{z} \in \mathbb{F}^{K+1} \implies 1 = \|\boldsymbol{x}\|_2 = \|\boldsymbol{W}\boldsymbol{z}\|_2 = \|\boldsymbol{z}\|_2 = \|\boldsymbol{W}'\boldsymbol{W}\boldsymbol{z}\|_2 = \|\boldsymbol{W}'\boldsymbol{x}\|_2$.

To complete the proof:

$$\begin{aligned}
\left\| \boldsymbol{A} - \hat{\boldsymbol{A}}_K \right\|_2^2 &= \left\| \sum_{k=K+1}^{r} \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k' \right\|_2^2 = \sigma_{K+1}^2 = \sigma_{K+1}^2 \|\boldsymbol{x}\|_2^2 = \sigma_{K+1}^2 \|\boldsymbol{W}'\boldsymbol{x}\|_2^2 \text{ because } \|\boldsymbol{x}\|_2 = 1 = \|\boldsymbol{W}'\boldsymbol{x}\|_2 \\
&= \sigma_{K+1}^2 \sum_{k=1}^{K+1} \|\boldsymbol{v}_k'\boldsymbol{x}\|_2^2 \le \sum_{k=1}^{K+1} \sigma_k^2 \|\boldsymbol{v}_k'\boldsymbol{x}\|_2^2 = \|\boldsymbol{A}\boldsymbol{x}\|_2^2 \text{ because } \boldsymbol{x} \in \mathcal{R}(\boldsymbol{W}) \\
&= \|(\boldsymbol{A} - \boldsymbol{B})\,\boldsymbol{x}\|_2^2 \text{ because } \boldsymbol{x} \in \mathcal{N}(\boldsymbol{B}) \\
&\le \|\boldsymbol{A} - \boldsymbol{B}\|_2^2 \|\boldsymbol{x}\|_2^2 = \|\boldsymbol{A} - \boldsymbol{B}\|_2^2 .
\end{aligned}$$

□

Bases for $\mathbb{F}^{M \times N}$

Recall that a set of **linearly independent** vectors $\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots$ is a **basis** for a vector space $\mathcal{V}$ iff

$$\mathsf{span}(\{\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots\}) = \mathcal{V},$$

i.e., we can write every vector in $\mathcal{V}$ as a linear combination of the basis vectors: $\boldsymbol{v} \in \mathcal{V} \implies \boldsymbol{v} = \sum_k \boldsymbol{b}_k \alpha_k$ for some $\alpha_k \in \mathbb{F}$. Now we consider two types of bases for the vector space of **matrices** $\mathbb{F}^{M \times N}$.

Canonical basis for $\mathbb{F}^{M \times N}$

First, the obvious (canonical) basis for $\mathbb{F}^{M \times N}$ is: $\mathcal{B} = \{\boldsymbol{e}_m \tilde{\boldsymbol{e}}_n' : m = 1, \ldots, M, \ n = 1, \ldots N\}$ where $\boldsymbol{e}_m$ denotes the mth unit vector of length M and $\tilde{\boldsymbol{e}}_n$ denotes the nth unit vector of length N, These are clearly linearly independent and $\mathsf{span}(\mathcal{B}) = \mathbb{F}^{M \times N}$ because $\boldsymbol{A} \in \mathbb{F}^{M \times N} \implies \boldsymbol{A} = \sum_{m=1}^{M} \sum_{n=1}^{N} a_{mn} \boldsymbol{e}_m \tilde{\boldsymbol{e}}_n'$.

To attempt "**dimensionality reduction**" using this basis, we could seek an approximation of the form $\boldsymbol{B} = \sum_{m=1}^{M} \sum_{n=1}^{N} b_{mn} \boldsymbol{e}_m \tilde{\boldsymbol{e}}_n'$, where we limit the number of nonzero coefficients b_{mn}.

Let $\|\mathsf{vec}(\boldsymbol{B})\|_0$ denote the number of nonzero elements of matrix $\boldsymbol{B}$. Then a natural optimization problem is:

$$\hat{\boldsymbol{B}} = \underset{\boldsymbol{B} \in \mathbb{F}^{M \times N}, \ \|\mathsf{vec}(\boldsymbol{B})\|_0 \le K}{\arg\min} \|\boldsymbol{A} - \boldsymbol{B}\|_{\mathrm{F}}.$$

This problem has a trivial solution: choose the K elements of $\boldsymbol{A}$ having the largest magnitudes $|a_{mn}|$ and set

all other elements of $\hat{\boldsymbol{B}}$ to zero. The approximation error is the square root of the sum of squared magnitudes of all those other elements.

Example. (Read)

$$\hat{\boldsymbol{B}} = \underset{\boldsymbol{B} \in \mathbb{F}^{M \times N},\ \|\boldsymbol{B}\|_0 \le 3}{\arg\min} \left\| \begin{bmatrix} 3 & 5 & 7 \\ 6 & 0 & 2 \end{bmatrix} - \boldsymbol{B} \right\|_{\mathrm{F}} = \begin{bmatrix} 0 & 5 & 7 \\ 6 & 0 & 0 \end{bmatrix}.$$

The approximation error is $\left\| \boldsymbol{A} - \hat{\boldsymbol{B}} \right\|_{\mathrm{F}} = \left\| \begin{bmatrix} 3 & 0 & 0 \\ 0 & 0 & 2 \end{bmatrix} \right\|_{\mathrm{F}} = \sqrt{2^2 + 3^2}$.

How large must K be here for this approximation to have zero error?
$K = 5$ because $\boldsymbol{A}$ has 5 nonzero elements.

How large must K be here for the rank-K approximation to have zero error?
$K = 2$ because $\boldsymbol{A}$ has rank 2.

The **low-rank approximation** seems more parsimonious than approximation using $\mathcal{B}$.
To state this rigorously, we must first show that our low-rank approximation also uses a **basis** for $\mathbb{F}^{M \times N}$.

Orthonormal bases for $\mathbb{F}^{M \times N}$

Let $\boldsymbol{U}$ denote any **unitary** $M \times M$ matrix and $\boldsymbol{V}$ denote any **unitary** $N \times N$ matrix.
Now endow $\mathbb{F}^{M \times N}$ with the (Frobenius) **inner product**: $\langle \boldsymbol{A}, \boldsymbol{B} \rangle =$
and corresponding (Frobenius) norm: $\|\boldsymbol{A}\|_{\mathrm{F}} = \sqrt{\langle \boldsymbol{A}, \boldsymbol{A} \rangle} = \sqrt{\mathsf{trace}\{\boldsymbol{A}'\boldsymbol{A}\}}$.

Claim: the following set of MN rank-1 $M \times N$ outer-product matrices is an **orthonormal basis** for $\mathbb{F}^{M \times N}$:

$$\{\boldsymbol{u}_m \boldsymbol{v}_n' : m = 1, \dots, M, \ n = 1, \dots N\}.$$

Proof of orthonormality (for the Frobenius inner product):

$$\begin{aligned}\langle \boldsymbol{u}_m \boldsymbol{v}_n', \boldsymbol{u}_k \boldsymbol{v}_l' \rangle &= \mathsf{trace}\{(\boldsymbol{u}_k \boldsymbol{v}_l')' (\boldsymbol{u}_m \boldsymbol{v}_n')\} = \mathsf{trace}\{\boldsymbol{v}_l \boldsymbol{u}_k' (\boldsymbol{u}_m \boldsymbol{v}_n')\} = \mathsf{trace}\{\boldsymbol{u}_k' \boldsymbol{u}_m \boldsymbol{v}_n' \boldsymbol{v}_l\} \\ &= (\boldsymbol{u}_k' \boldsymbol{u}_m)(\boldsymbol{v}_n' \boldsymbol{v}_l) = \begin{cases} 1, & k = m, \ l = n \\ 0, & \text{otherwise.} \end{cases}\end{aligned}$$

Being orthonormal, the set $\{\boldsymbol{u}_m \boldsymbol{v}_n'\}$ spans a MN-dimensional space and is a basis for $\mathbb{F}^{M \times N}$. Thus one can write any $\boldsymbol{X} \in \mathbb{F}^{M \times N}$ in terms of that basis, *i.e.*, $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{C}\boldsymbol{V}' = \sum_{m=1}^{M} \sum_{n=1}^{N} c_{mn} \boldsymbol{u}_m \boldsymbol{v}_n'$, where $\boldsymbol{C} \triangleq \boldsymbol{U}' \boldsymbol{X} \boldsymbol{V}$, consistent with (6.4).

When working with a matrix $\boldsymbol{A}$, using the basis for $\mathbb{F}^{M \times N}$ that is formed from its own left and right **singular vectors**, *i.e.*, $\boldsymbol{U}$ and $\boldsymbol{V}$, respectively, turns out to be provide the most parsimonious representation. Note that the original low-rank problem formulation (6.1) did not involve any SVD, but an **SVD** arose in the solution.

Does the canonical basis $\{\boldsymbol{e}_m \tilde{\boldsymbol{e}}_n' : m = 1, \dots, M, \ n = 1, \dots N\}$ form an orthonormal basis for $\mathbb{F}^{M \times N}$ when using the Frobenius inner product?

A: Yes B: No C: Insufficient information **??**

Low-rank approximation summary

The low-rank approximation problem (6.1) and its SVD-based solution (6.2) provides the best low-rank *representation* (or approximation) of a given matrix (in the Frobenius norm sense). Specifically, if $\boldsymbol{B}$ is any rank-K matrix having the same size as a matrix $\boldsymbol{A} = \sum_{k=1}^{r} \boldsymbol{u}_k \boldsymbol{v}_k'$, then:

$$\sqrt{\sum_{k=K+1}^{r} \sigma_k^2} = \left\| \boldsymbol{A} - \hat{\boldsymbol{A}}_K \right\|_{\mathrm{F}} = \left\| \boldsymbol{A} - \sum_{k=1}^{K} \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k' \right\|_{\mathrm{F}} \leq \| \boldsymbol{A} - \boldsymbol{B} \|_{\mathrm{F}}, \quad \forall \boldsymbol{B} \in \mathcal{L}_K^{M \times N}.$$

Generalizations (Read)

What if the data is corrupted by noise? What if

$$\boldsymbol{Y} = \boldsymbol{X} + \boldsymbol{\varepsilon}$$

where $\boldsymbol{X}$ is a low-rank matrix (or close to low rank). Due to the noise matrix $\boldsymbol{\varepsilon}$, typically $\boldsymbol{Y}$ is not low-rank. We really want to recover $\boldsymbol{X}$ here, not just represent or approximate $\boldsymbol{Y}$. How do we do that? An answer called OptShrink is given in [13] and discussed on p. 6.50.

Low-rank approximation is related closely to **principal component analysis** (**PCA**), discussed on p. 6.57. There are many interesting **PCA generalizations** including robust methods, nonlinear models, multi-linear (tensor) models, **nonnegative matrix factorizations** (**NMF**) [14], methods that use sparsity, ...

Rank and stability (Read)

Although **rank** is a simple mathematical concept, it is not stable numerically. Precisely, it is not a continuous function of the elements of a matrix.

Example. Consider the following two matrices:

$$\boldsymbol{A} = \begin{bmatrix} 1.7 & 1.7 \\ 1.7 & 1.7 \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} 1.7 & 1.7 \\ 1.7 + \varepsilon & 1.7 \end{bmatrix}, \quad 0 < |\varepsilon| \ll 1.$$

These two matrices are nearly indistinguishable numerically, yet $\mathsf{rank}(\boldsymbol{A}) = 1$ and $\mathsf{rank}(\boldsymbol{B}) = 2$.

An alternative matrix property used in some situations is the **stable rank** [15], defined (for $\boldsymbol{A} \neq \boldsymbol{0}$) as:

$$\frac{\|\boldsymbol{A}\|_{\mathrm{F}}^2}{\|\boldsymbol{A}\|_2^2}, \text{ where } 1 \leq \frac{\|\boldsymbol{A}\|_{\mathrm{F}}^2}{\|\boldsymbol{A}\|_2^2} = \frac{1}{\sigma_1^2} \sum_{k=1}^{\min(M,N)} \sigma_k^2 = 1 + \sum_{k=2}^{\min(M,N)} \frac{\sigma_k^2}{\sigma_1^2} \leq r,$$

because (see HW) $\|\boldsymbol{A}\|_2 \leq \|\boldsymbol{A}\|_{\mathrm{F}} \leq \sqrt{r}\|\boldsymbol{A}\|_2$. For the above example, $\|\boldsymbol{B}\|_{\mathrm{F}}^2 / \|\boldsymbol{B}\|_2^2 \approx 1 + \varepsilon/2$.

Example. For $\boldsymbol{A} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$, the stable rank is $\frac{\|\boldsymbol{A}\|_{\mathrm{F}}^2}{\|\boldsymbol{A}\|_2^2} = \frac{2^2 + 1^2}{2^2} = 1.25 \in [1, 2]$.

What is the **stable rank** of $\boldsymbol{A} = \boldsymbol{x}\boldsymbol{y}'$?

A: 0 B: 1 C: $\|\boldsymbol{x}\|_2 \|\boldsymbol{y}\|_2$ D: $\|\boldsymbol{x}\|_2^2 \|\boldsymbol{y}\|_2^2$ **??**

6.2 Sensor localization application (Multidimensional scaling)

We now turn to another application of matrix factorization: **sensor localization** via **multidimensional scaling**. Suppose $J \geq d$ sensors are located at unknown locations $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_J \in \mathbb{R}^d$, where typically $d = 2$ or $d = 3$. All we are given is the $J \times J$ **distance matrix** $\boldsymbol{D}$ having elements

$$d_{ij} = d_{ji} =$$

Clearly by definition the diagonal elements of $\boldsymbol{D}$ are zero: $d_{jj} = 0$.

Goal: given $\boldsymbol{D}$, determine the locations $\{\boldsymbol{c}_j\}$.

Limitation: there is a fundamental ambiguity because translating or rotating coordinates does not change $\boldsymbol{D}$. Specifically, if $\tilde{\boldsymbol{c}}_j = \boldsymbol{Q}\boldsymbol{c}_j + \boldsymbol{d}$ where $\boldsymbol{Q}$ is a $d \times d$ orthogonal matrix (such as a rotation matrix) and $\boldsymbol{d} \in \mathbb{R}^d$ is a displacement vector, then

$$\|\tilde{\boldsymbol{c}}_i - \tilde{\boldsymbol{c}}_j\|_2 = \|(\boldsymbol{Q}\boldsymbol{c}_i + \boldsymbol{d}) - (\boldsymbol{Q}\boldsymbol{c}_j + \boldsymbol{d})\|_2 = \|\boldsymbol{Q}(\boldsymbol{c}_i - \boldsymbol{c}_j)\|_2 = \|\boldsymbol{c}_i - \boldsymbol{c}_j\|_2$$

because the Euclidean norm is **unitarily invariant**. So $\boldsymbol{D}$ would be the same for $\{\tilde{\boldsymbol{c}}_j\}$ and $\{\boldsymbol{c}_j\}$.
Thus we must be content with determining locations $\{\boldsymbol{c}_j\}$ to within some translation and rotation factor.
Note the choice to use a matrix representation of the data!
And of course the locations are vectors.

Derivation (analysis)

To derive a solution (for noiseless data), define another matrix $\boldsymbol{S}$ by the square of the elements of $\boldsymbol{D}$:

$$s_{ij} \triangleq d_{ij}^2 = \|\boldsymbol{c}_i - \boldsymbol{c}_j\|_2^2 = \|\boldsymbol{c}_i\|_2^2 + \|\boldsymbol{c}_j\|_2^2 - 2 \langle \boldsymbol{c}_i, \boldsymbol{c}_j \rangle .$$

Naturally we write $\boldsymbol{S}$ in matrix form:

$$\begin{aligned}
\boldsymbol{S} &= \begin{bmatrix} s_{11} & \dots & s_{1J} \\ \vdots & \dots & \vdots \\ s_{J1} & \dots & s_{JJ} \end{bmatrix} = \begin{bmatrix} \|\boldsymbol{c}_1\|_2^2 & \dots & \|\boldsymbol{c}_1\|_2^2 \\ \vdots & \dots & \vdots \\ \|\boldsymbol{c}_J\|_2^2 & \dots & \|\boldsymbol{c}_J\|_2^2 \end{bmatrix} + \begin{bmatrix} \|\boldsymbol{c}_1\|_2^2 & \dots & \|\boldsymbol{c}_J\|_2^2 \\ \vdots & \dots & \vdots \\ \|\boldsymbol{c}_1\|_2^2 & \dots & \|\boldsymbol{c}_J\|_2^2 \end{bmatrix} - 2 \begin{bmatrix} \langle \boldsymbol{c}_1, \boldsymbol{c}_1 \rangle & \dots & \langle \boldsymbol{c}_1, \boldsymbol{c}_J \rangle \\ \vdots & \dots & \vdots \\ \langle \boldsymbol{c}_J, \boldsymbol{c}_1 \rangle & \dots & \langle \boldsymbol{c}_J, \boldsymbol{c}_J \rangle \end{bmatrix} \\
&= \begin{bmatrix} \|\boldsymbol{c}_1\|_2^2 \\ \vdots \\ \|\boldsymbol{c}_J\|_2^2 \end{bmatrix} \boldsymbol{1}_J' + \boldsymbol{1}_J \begin{bmatrix} \|\boldsymbol{c}_1\|_2^2 & \dots & \|\boldsymbol{c}_J\|_2^2 \end{bmatrix} - 2 \begin{bmatrix} \boldsymbol{c}_1' \\ \vdots \\ \boldsymbol{c}_J' \end{bmatrix} \begin{bmatrix} \boldsymbol{c}_1 & \dots & \boldsymbol{c}_J \end{bmatrix} \\
&= \boldsymbol{r} \boldsymbol{1}_J' + \boldsymbol{1}_J \boldsymbol{r}' - 2 \boldsymbol{C}' \boldsymbol{C}, \qquad \boldsymbol{r} \triangleq \begin{bmatrix} \|\boldsymbol{c}_1\|_2^2 \\ \vdots \\ \|\boldsymbol{c}_J\|_2^2 \end{bmatrix}, \qquad \boldsymbol{C} \triangleq \underbrace{\begin{bmatrix} \boldsymbol{c}_1 & \dots & \boldsymbol{c}_J \end{bmatrix}}_{d \times J \text{ unknown locations}}.
\end{aligned} \tag{6.5}$$

Unfortunately we do not know the values in the vector $\boldsymbol{r}$.

So we know the $J \times J$ matrix $\boldsymbol{S}$ and we want to solve for the $d \times J$ unknown location matrix $\boldsymbol{C}$ using

$$\boldsymbol{S} = \boldsymbol{r}\boldsymbol{1}_J' + \boldsymbol{1}_J\boldsymbol{r}' - 2\boldsymbol{C}'\boldsymbol{C}. \tag{6.6}$$

Because $\boldsymbol{S}$ on the left is symmetric and zero along its diagonal, it contains $(J^2 - J)/2$ distinct known values. The right-hand side depends on the dJ unknown values in $\boldsymbol{C}$.

Recall that source localization has a translation ambiguity, so we may as well use a coordinate system where the centroid is $\boldsymbol{0}$, *i.e.*, we can assume $\boldsymbol{C}\boldsymbol{1}_J = \sum_{j=1}^{J} \boldsymbol{c}_j = \boldsymbol{0}$ without losing any further generality.

Define the following "de-meaning" operator that projects onto the **orthogonal complement** of $\mathcal{R}(\boldsymbol{1}_J)$:

$$\boldsymbol{P}^\perp \triangleq$$

Multiplying $\boldsymbol{P}^\perp$ by any vector produces a new vector whose mean value is zero, so we say it "de-means" the input vector. (The English word "demean" has a very different definition.) In winter, sometimes an airplane must be "de-iced" by removing ice from the wings. Here **de-mean** has similar use: removing the mean.

Clearly $\boldsymbol{P}^\perp\boldsymbol{1}_J = \boldsymbol{0}$, so $\boldsymbol{1}_J'\boldsymbol{P}^\perp = \boldsymbol{0}'$ and $\boldsymbol{C}\boldsymbol{P}^\perp = \boldsymbol{C}(\boldsymbol{I} - \frac{1}{J}\boldsymbol{1}_J\boldsymbol{1}_J') = \boldsymbol{C}$ so from (6.6):

$$\boldsymbol{P}^\perp\boldsymbol{S}\boldsymbol{P}^\perp =$$

Rearranging leads to the following simple expression for the $J \times J$ **Gram matrix**:

$$\boldsymbol{G} \triangleq \boldsymbol{C}'\boldsymbol{C} =$$

In words, we remove the row and column means of $\boldsymbol{S}$ and divide by -2.

Now we have the **Gram matrix** $\boldsymbol{G} = \boldsymbol{C}'\boldsymbol{C}$, but we really want the coordinates matrix $\boldsymbol{C}$ itself.

Although $\boldsymbol{C}'\boldsymbol{C}$ is a $J \times J$ matrix, $d \leq J$ so typically $\mathsf{rank}(\boldsymbol{C}'\boldsymbol{C}) = d$ (in the absence of noise). If the locations happen to be linearly dependent, *i.e.*, if the sensors are along one line through the origin in 2D or in the same plane through the origin in 3D, then $\mathsf{rank}(\boldsymbol{C}'\boldsymbol{C}) < d$.

To elaborate, consider $d = 2$ where

$$\boldsymbol{C} \triangleq \begin{bmatrix} \boldsymbol{c}_1 & \dots & \boldsymbol{c}_J \end{bmatrix} = \begin{bmatrix} x_1 & \dots & x_J \\ y_1 & \dots & y_J \end{bmatrix},$$

where $\boldsymbol{c}_j = (x_j, y_j)$. Because $\boldsymbol{C}$ has two rows, $\mathsf{rank}(\boldsymbol{C}'\boldsymbol{C}) = \mathsf{rank}(\boldsymbol{C}) \leq 2$.

When does $\mathsf{rank}(\boldsymbol{C}) = 1$? Only if the two rows of $\boldsymbol{C}$ are linearly dependent, *i.e.*, if $\boldsymbol{y} = \alpha \boldsymbol{x}$ for some $\alpha \in \mathbb{R}$, *i.e.*, if $y_j = \alpha x_j$ for all j, which is the equation for points along a line through the origin with slope α.

The matrix $\boldsymbol{C}'\boldsymbol{C}$ consists of inner products, and it is invariant to rotations of the source locations:

$$\tilde{\boldsymbol{C}} = \boldsymbol{Q}\boldsymbol{C} \implies \tilde{\boldsymbol{C}}'\tilde{\boldsymbol{C}} =$$

Given the $J \times J$ Gram matrix $\boldsymbol{G} \triangleq \boldsymbol{C}'\boldsymbol{C} = -\frac{1}{2}\boldsymbol{P}^\perp \boldsymbol{S} \boldsymbol{P}^\perp$, we determine its **compact SVD**:

$$\boldsymbol{G} = \boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{V}' = \sum_{k=1}^{J} \sigma_k \boldsymbol{v}_k \boldsymbol{v}_k' = \qquad \boldsymbol{\Sigma}_d \triangleq \qquad \boldsymbol{V}_d \triangleq$$

- Because $\boldsymbol{G}$ is (symmetric) positive semidefinite (in the absence of noise), we know $\boldsymbol{U} = \boldsymbol{V}$.
- Because $\boldsymbol{G}$ has rank at most d (in the absence of noise), the above **low-rank** form is *exact*, not an approximation like we used earlier in this chapter.

We define our estimate of the source locations to be the J columns of this $d \times J$ matrix:

$$\hat{\boldsymbol{C}} = \boldsymbol{\Sigma}_d^{1/2} \boldsymbol{V}_d' = \underbrace{\begin{bmatrix} \sqrt{\sigma_1} & & \\ & \ddots & \\ & & \sqrt{\sigma_d} \end{bmatrix}}_{d \times d} \underbrace{\begin{bmatrix} - & \boldsymbol{v}_1' & - \\ & \vdots & \\ - & \boldsymbol{v}_d' & - \end{bmatrix}}_{d \times J},$$

because this estimate $\hat{\boldsymbol{C}}$ is consistent with $\boldsymbol{C}$ to within an unknown rotation and translation:

$$\hat{\boldsymbol{C}}'\hat{\boldsymbol{C}} = (\boldsymbol{\Sigma}_d^{1/2}\boldsymbol{V}_d')'(\boldsymbol{\Sigma}_d^{1/2}\boldsymbol{V}_d') = \boldsymbol{V}_d \boldsymbol{\Sigma}_d \boldsymbol{V}_d' =$$

Now it seems like we are done because we have an expression for $\boldsymbol{G}$ that is computable from $\boldsymbol{S}$, so we can use its eigendecomposition to find the source location estimates $\hat{\boldsymbol{C}}$. However, we assumed that $\boldsymbol{C}\mathbf{1}_J = \mathbf{0}$ so we must verify that $\hat{\boldsymbol{C}}\mathbf{1}_J = \mathbf{0}$ to ensure that our approach is self consistent.

Because $\boldsymbol{P}^\perp \mathbf{1}_J = \mathbf{0}$, it follows that $\boldsymbol{G}\mathbf{1}_J = \mathbf{0}$, so $\boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{V}'\mathbf{1}_J = \boldsymbol{V}_d\boldsymbol{\Sigma}_d\boldsymbol{V}_d'\mathbf{1}_J = \mathbf{0}$, which in turn implies that $\hat{\boldsymbol{C}}\mathbf{1}_J = \boldsymbol{\Sigma}_d^{1/2}\boldsymbol{V}_d'\mathbf{1}_J = \mathbf{0}$.

What is the size of the final $\mathbf{0}$ in the immediately preceding line?

A: $J \times J$ B: $J \times 1$ C: $J \times d$ D: $d \times J$ E: $d \times 1$ **??**

Multidimensional scaling

Summary of method for finding locations from distances (in the possible presence of noise):

- Given distances arranged in a matrix $\boldsymbol{D}$.
- Compute $\boldsymbol{S}$ from $\boldsymbol{D}$ by squaring elements.
- Compute $\hat{\boldsymbol{G}} \triangleq -\frac{1}{2}\boldsymbol{P}^\perp \boldsymbol{S} \boldsymbol{P}^\perp \approx \boldsymbol{C}'\boldsymbol{C}$ by de-meaning.
- Optional: force $\hat{\boldsymbol{G}}$ to be symmetric by replacing it with $(\hat{\boldsymbol{G}} + \hat{\boldsymbol{G}}')/2$
- Compute the **compact SVD** $\hat{\boldsymbol{G}} = \boldsymbol{U}_d\boldsymbol{\Sigma}_d\boldsymbol{V}_d'$. (If $\hat{\boldsymbol{G}}$ is symmetric then $\boldsymbol{U}_d = \boldsymbol{V}_d$.)
- Use rank-d terms for source location estimate: $\hat{\boldsymbol{C}} = \boldsymbol{\Sigma}_d^{1/2}\boldsymbol{V}_d'$.

This remarkably simple algorithm for finding $\hat{\boldsymbol{C}}$ is called (classical) **multidimensional scaling** [16, Ch. 12], and using the low-rank compact SVD is a key step. Because $\boldsymbol{G} = \boldsymbol{C}'\boldsymbol{C}$, $\boldsymbol{G}$ is positive semidefinite (in the absence of noise). If we force $\hat{\boldsymbol{G}}$ to be likewise symmetric, then its eigenvalues are all real and nonnegative so alternatively one could use an eigendecomposition of $\hat{\boldsymbol{G}}$ with suitably ordered eigenvalues.

Questions

Consider a given noiseless distance matrix $\boldsymbol{D}$ with $J > d$. The position estimates $\hat{\boldsymbol{C}}$ returned by the SVD-based MDS algorithm are unique (to within numerical precision), *i.e.*, are the same for any SVD of the Gram matrix.

A: True B: False **??**

In the possible presence of noise, the product $\hat{\boldsymbol{C}}'\hat{\boldsymbol{C}}$ is unique (to within numerical precision) for any SVD of the calculated Gram matrix $\hat{\boldsymbol{G}}$.

A: never B: usually C: always **??**

Practical implementation

The `MultivariateStats` package of JULIA has a `classical_mds` command.

But the method is so simple that it is just as easy to code our own.

Example. Consider $J = 3$ sources that are all equally distant: $\boldsymbol{S} = \boldsymbol{D}$ `.^2` $= \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$.

Just a few lines of JULIA suffices (see notebook linked on next page):

```
using Plots; using LinearAlgebra: svd, Diagonal; using Statistics: mean
D = [0 1 1; 1 0 1; 1 1 0] + 0e-9 * randn(3,3) # given distances
S = D.^2
tmp = S .- mean(S, dims=1)
G = -0.5 * (tmp .- mean(tmp, dims=2))
(_, s, V) = svd(G)
C = Diagonal(sqrt.(s[1:2])) * V[:,1:2]'
scatter(C[1,:], C[2,:], aspect_ratio=1, label="",
  xlabel="x", ylabel="y", title="s=$(s)",
  xtick=-.5:.5:.5, ytick=-.5:.5:.5, xlim=(-0.6,0.6), ylim=(-0.6,0.6))
```

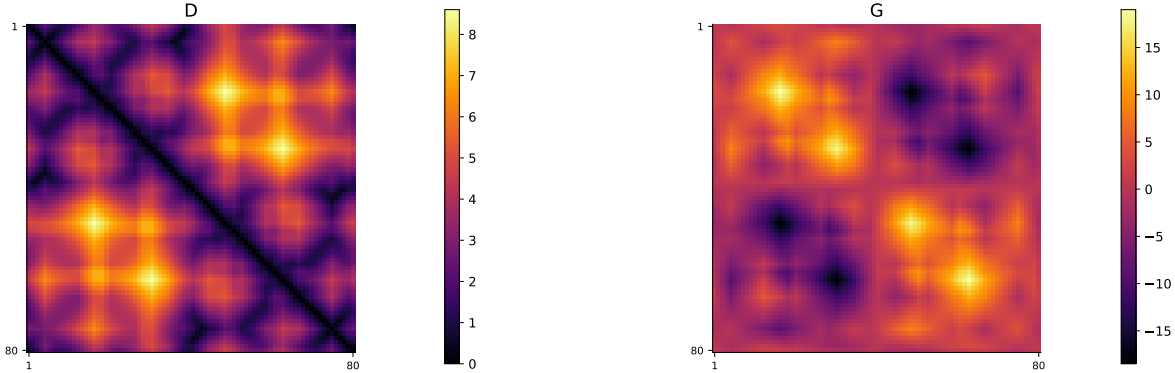
What "shape" should appear? **??**

Example. An unknown collection of $J = 80$ source points:

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/06_source_local1.html

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/06_source_local1.ipynb

Images of $\boldsymbol{D}$ and $\boldsymbol{G}$:



Run demo files to show $\hat{\boldsymbol{C}}$ and consider noisy case to examine robustness.

Extensions

- When the distance measurements are noisy then $\boldsymbol{D}$ may not be perfectly symmetric and $\boldsymbol{P}^{\perp}\boldsymbol{S}\boldsymbol{P}^{\perp}$ may have (hopefully small) negative eigenvalues.
  Use **truncated SVD**: truncate rank to dimension d of embedding.
- What if $\boldsymbol{D}$ is **geodesic distances** or distances along roads between locations?
  Distortion in location estimates.
- What if some sensors cannot reach some other sensors, *i.e.*, if $\boldsymbol{D}$ has some missing elements?
- Is there a way to formulate the problem that considers the possibility of noise (errors in the distance values) at the outset, instead of simply empirically investigating the effects of noise on our solution.
- What if there are outliers in the data, *e.g.*, some sensors that give incorrect values (either due to errors or because the sensor was hacked or otherwise compromised). Is MDS robust to such errors, or is a different formulation needed, perhaps based on a robust distance measure like $\|\cdot\|_1$?
- If we want to compare the location estimates $\hat{\boldsymbol{C}}$ to some ground truth, we can use the Procrustes method to align them, compensating for the inherent rotation/translation ambiguity.

6.3 Proximal operators

Our next topic is some alternative low-rank approximation methods, also formulated as optimization problems. To describe those alternatives, we first need to introduce an important operation.

Define. The **proximal operator** [17, 18] (also called the **proximal mapping** [19, Ch. 6]) associated with a (typically **convex**) function $f : \mathbb{F}^N \mapsto \mathbb{R}$, is defined, for any $\boldsymbol{v} \in \mathbb{F}^N$, as the following minimizer:

$$\operatorname{prox}_f(\boldsymbol{v}) \triangleq \arg\min_{\boldsymbol{x} \in \mathbb{F}^N} \frac{1}{2} \|\boldsymbol{v} - \boldsymbol{x}\|_2^2 + f(\boldsymbol{x}). \tag{6.7}$$

- The norm squared is **strictly convex**, so when f is convex the "arg min" is unique for any $\boldsymbol{v} \in \mathbb{F}^N$.
- Usually the function f is additively separable, in which case the "arg min" separates into N individual 1D minimization problems. See example below.
- The proximal operator transforms one function $f(\cdot)$ into another function $\operatorname{prox}_f(\cdot)$.

In general, the functions f and prox_f have the same (domain?, range?)

A: true,true B: true,false C: false,true D: false,false **??**

??

Example. The main case used in EECS 551 is when $\mathbb{F} = \mathbb{R}$ and $f(\boldsymbol{x}) = \beta \|\boldsymbol{x}\|_p^p = \beta \sum_{n=1}^N |x_n|^p$, for $\beta \geq 0$. The most important case is when $p = 1$, for which

$$\operatorname{prox}_f(\boldsymbol{v}) = \arg\min_{\boldsymbol{x} \in \mathbb{R}^N} \frac{1}{2} \|\boldsymbol{v} - \boldsymbol{x}\|_2^2 + \beta \|\boldsymbol{x}\|_1 = \arg\min_{\boldsymbol{x} \in \mathbb{R}^N} \sum_{n=1}^N \left(\frac{1}{2}(v_n - x_n)^2 + \beta |x_n| \right).$$

This sum is additively separable, meaning each x_n appears in just one term of the sum. So we can solve the minimization problem separately for each x_n term by solving

$$\hat{x}_n \triangleq \arg\min_{x_n \in \mathbb{R}} g(x_n), \quad g_n(x_n) \triangleq \frac{1}{2}(v_n - x_n)^2 + \beta |x_n|, \quad n = 1, \ldots, N.$$

Often for minimization problems we take the derivative and set it to zero. That approach does not quite work here because $|\cdot|$ is not differentiable at 0. So we need a "braces and cases" approach.

Suppose the minimizer satisfies $\hat{x}_n > 0$. In this regime the function g is differentiable with derivative

$$\dot{g}_n(x_n) = x_n - v_n + \beta 1.$$

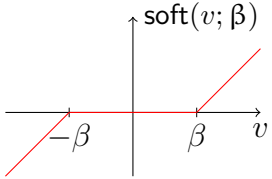
Equating to zero yields $\hat{x}_n = v_n - \beta$. However, we must keep in mind that we assumed here that $\hat{x}_n > 0$, so this solution is valid only when $v_n > \beta$.

Now suppose the minimizer satisfies $\hat{x}_n < 0$. In this regime the function g is differentiable with derivative

$$\dot{g}_n(x_n) = x_n - v_n + \beta(-1).$$

Equating to zero yields $\hat{x}_n = v_n + \beta$. We must keep in mind that we assumed here that $\hat{x}_n < 0$, so this solution is valid only when $v_n < -\beta$. Because we have considered all cases where the minimizer is nonzero, for any other input value v_n, the minimizer must be zero. Thus we have shown that

$$\arg\min_{x_n \in \mathbb{R}} \frac{1}{2}(v_n - x_n)^2 + \beta |x_n| = \mathsf{soft}(v_n; \beta), \quad \mathsf{soft}(v_n; \beta) \triangleq \begin{cases} v_n - \beta, & v_n > \beta \\ v_n + \beta, & v_n < -\beta \\ 0, & \text{otherwise.} \end{cases}$$



This operation is called **soft thresholding** and it induces **sparsity** because of the range of values set to zero.

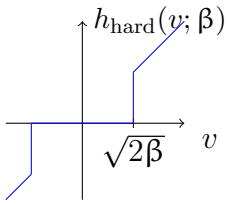
Putting it all together, we have shown that the proximal operator for the 1-norm scaled by β is element-wise soft thresholding:

$$\mathrm{prox}_{\beta \|\cdot\|_1}(\boldsymbol{v}) = \mathsf{soft}\,.(\boldsymbol{v}; \beta).$$

Example. Another main case of interest here is the (nonconvex) 0-norm $f(\boldsymbol{x}) = \beta \|\boldsymbol{x}\|_0$, for $\beta \geq 0$, for which the **proximal operator** is (element-wise) **hard thresholding**. Here is the derivation:

$$\operatorname{prox}_{\beta\|\cdot\|_0}(\boldsymbol{v}) = \arg\min_{\boldsymbol{x}\in\mathbb{F}^N} \frac{1}{2}\|\boldsymbol{v}-\boldsymbol{x}\|_2^2 + \beta\|\boldsymbol{x}\|_0 = \arg\min_{\boldsymbol{x}\in\mathbb{F}^N} \sum_{n=1}^{N} \left(\frac{1}{2}|v_n - x_n|^2 + \beta \mathbb{I}_{\{x_n \neq 0\}} \right) = h_{\text{hard}}.(\boldsymbol{v}; \beta)$$

$$h_{\text{hard}}(v;\beta) \triangleq \arg\min_{x\in\mathbb{R}} \frac{1}{2}|v-x|^2 + \beta\mathbb{I}_{\{x\neq 0\}} = \begin{cases} v, & \frac{1}{2}|v|^2 > \beta \\ 0, & \text{otherwise.} \end{cases} \tag{6.8}$$



Again we cannot derive the minimizer by simple differentiation, because the indicator function is not differentiable at zero. So we must apply "braces and cases."

If the minimizer is nonzero, then the second term is β and the minimizer of the first term is v, and the overall cost is $0 + \beta = \beta$. If the minimizer is 0, then the first term is $(v-0)^2/2$ and the second term is 0, so the cost is $v^2/2$. So 0 is the minimizer when $v^2/2 < \beta$, otherwise v is the minimizer.

The minimizer is not unique if $|v| = \sqrt{2\beta}$; in such cases, both v and 0 are global minimizers.
The threshold is $\sqrt{2\beta}$ due to the $1/2$ in front of the norm in the proximal operator definition.

Now we return to low-rank approximation where we will use these results.

6.4 Alternative low-rank approximation formulations

If $\boldsymbol{Y} \in \mathbb{F}^{M \times N}$ is a given data matrix with SVD $\boldsymbol{Y} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}'$ and $\boldsymbol{Y} = \boldsymbol{X} + \boldsymbol{\varepsilon}$ where we believe $\mathsf{rank}(\boldsymbol{X}) \le K$ and $\boldsymbol{\varepsilon}$ denotes a $M \times N$ noise matrix, then the constrained formulation discussed so far is natural [3]:

$$\hat{\boldsymbol{X}} = \underset{\boldsymbol{X} \in \mathcal{L}_K^{M \times N}}{\arg\min} \left\| \boldsymbol{Y} - \boldsymbol{X} \right\|_{\mathrm{F}}^2 = \boldsymbol{U}_K \boldsymbol{\Sigma}_K \boldsymbol{V}_K' = \sum_{k=1}^{K} \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k' = \sum_{k=1}^{r} \sigma_k \mathbb{I}_{\{k \le K\}} \boldsymbol{u}_k \boldsymbol{v}_k'.$$

This problem is nonconvex, but still has a nice solution (6.2), reiterated above in three different forms. The final form reminds us that we have "set to 0" all the singular values for $k > K$.

Unconstrained / regularized formulation

When the rank is unknown, an alternative approach is to consider an *unconstrained* cost function that discourages (penalizes) high rank (high complexity) rather than constraining it [20]:

$$\hat{\boldsymbol{X}} = \underset{\boldsymbol{X} \in \mathbb{F}^{M \times N}}{\arg\min} \qquad (6.9)$$

where (see proof sketch below) $h_{\mathrm{hard}}(\cdot; \beta)$ is the hard-thresholding function defined in (6.8).

General unitarily invariant formulations

There are several ways to form low-rank approximations that balance between data-fit and model complexity, all having the general form:

$$\hat{\boldsymbol{X}} = \underset{\boldsymbol{X} \in \mathbb{F}^{M \times N}}{\arg\min} \tag{6.10}$$

for some **unitarily invariant** matrix norm $\|\cdot\|$ and for some **regularizer** $R(\boldsymbol{X})$, where $R : \mathbb{F}^{M \times N} \mapsto \mathbb{R}$, such as $R(\boldsymbol{X}) = \mathsf{rank}(\boldsymbol{X})$. The regularization parameter (aka **hyperparameter**) $\beta > 0$ controls the trade-off between fit to the data $\boldsymbol{Y}$, as measured by $\|\boldsymbol{Y} - \boldsymbol{X}\|$, and model complexity, as quantified by $R(\boldsymbol{X})$.

We assume hereafter that the regularizer is also a unitarily invariant function, *i.e.*, $R(\boldsymbol{U}\boldsymbol{X}\boldsymbol{V}) = R(\boldsymbol{X})$ for *any* suitably sized unitary matrices $\boldsymbol{U}$ and $\boldsymbol{V}$. Because both the data-fit norm and the regularizer are unitarily invariant, using the symmetric gauge principles of [4, 10], one can show that any minimizer has the form

$$\hat{\boldsymbol{X}} = \sum_{k=1}^{r} \hat{w}_k \, \boldsymbol{u}_k \boldsymbol{v}_k' = \boldsymbol{U}_r \hat{\boldsymbol{\Sigma}}_r \boldsymbol{V}_r' \text{ where } \hat{\boldsymbol{\Sigma}}_r = \mathsf{diag}\{\hat{w}_k\}, \quad \hat{w}_k = h_k(\sigma_1, \ldots, \sigma_r; \beta), \quad \boldsymbol{Y} = \boldsymbol{U}_r \boldsymbol{\Sigma}_r \boldsymbol{V}_r', \tag{6.11}$$

for some **shrinkage** or **thresholding** function $h_k(\cdot; \beta)$ that depends on the data-fit norm and the regularizer.

What is the rank of the $\hat{\boldsymbol{X}}$ in (6.11) when $\mathsf{rank}(\boldsymbol{Y}) = r$? (Choose best answer.)

A: Usually 1 B: Always 1 C: Usually r D: Always r E: None of these

??

Accepting that (6.11) is the form of the solution, we can rewrite the general optimization problem (6.10) as

$$\hat{\boldsymbol{X}} = \boldsymbol{U}_r \hat{\boldsymbol{\Sigma}}_r \boldsymbol{V}_r', \quad \hat{\boldsymbol{\Sigma}}_r = \underset{\boldsymbol{S}=\mathsf{diag}\{s_1,\ldots,s_r\}}{\arg\min} \tag{6.12}$$

Singular value hard thresholding

Now return to the special case (6.9) with the Frobenius norm and the rank penalty. The equivalent minimization problem in terms of the singular values requires minimizing over $\{s_1, \ldots, s_r\}$:

(6.13)

We started with the cost function (6.9), simplified it to the equivalent problem with diagonal matrices (6.12), and now we have r separate 1D problems that we solved earlier when deriving the proximal operator for the 0-norm that resulted in hard thresholding; see (6.8).

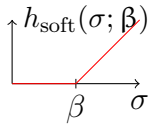
This analysis leads to $\hat{w}_k = h_{\text{hard}}(\sigma_k; \beta)$ in the method (6.9) that is called **singular value hard thresholding**.

Singular value soft thresholding

Any formulation involving $\mathsf{rank}(\cdot)$ is non-convex, and it is somewhat remarkable that a nice solution to (6.9) exists despite that non-convexity. Often it can be preferable to have a **convex** formulation. The **convex relaxation** [21] of rank is the **nuclear norm** that leads to **soft thresholding** of singular values:

$$\hat{\boldsymbol{X}} = \tag{6.14}$$

So $\hat{w}_k = h_{\text{soft}}(\sigma_k; \beta)$, $\quad h_{\text{soft}}(\sigma; \beta) \triangleq [\sigma - \beta]_+ = \begin{cases} \sigma - \beta, & \sigma > \beta, \\ 0, & \text{otherwise.} \end{cases}$



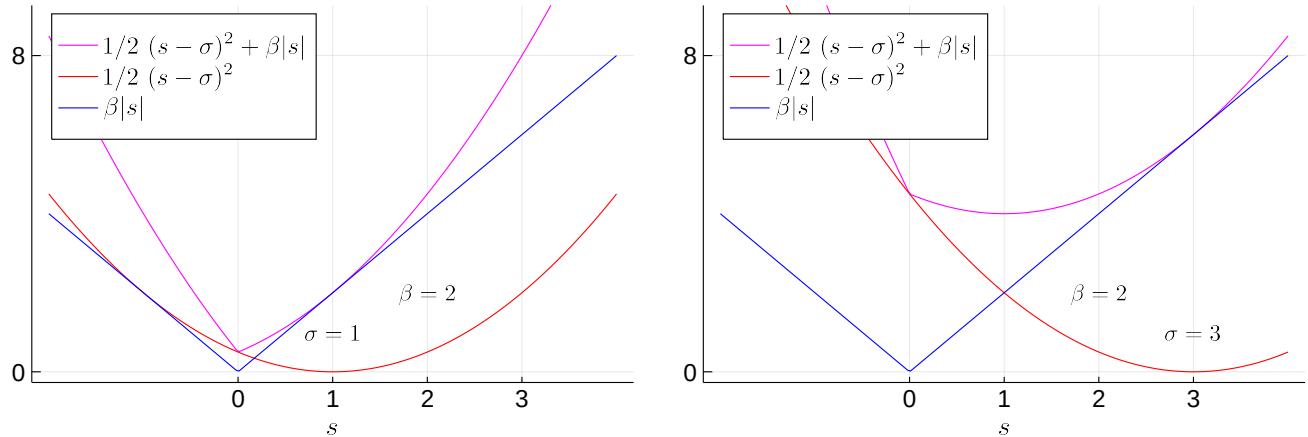
This approach is called **singular value soft thresholding** (**SVST**).

Proof sketch: $\frac{1}{2}\|\boldsymbol{\Sigma}_r - \boldsymbol{S}\|_{\text{F}}^2 + \beta\|\boldsymbol{S}\|_* = \sum_{k=1}^r \frac{1}{2}(\sigma_k - s_k)^2 + \beta s_k \implies \hat{w}_k = \arg\min_s \frac{1}{2}(\sigma_k - s)^2 + \beta\,|s|$.
Sketching we see that the minimizer will be some $s \geq 0$ because $\sigma_k \geq 0$.
Differentiating: $0 = (s - \sigma_k) + \beta\Big|_{s=\hat{w}_k} \implies \hat{w}_k = \sigma_k - \beta$ but we need $\hat{w}_k \geq 0$ so $\hat{w}_k = [\sigma_k - \beta]_+$.

The following figures illustrate the cases where $\sigma_k < \beta$ and $\sigma_k > \beta$, by plotting the following functions:

$$\frac{1}{2}(s-\sigma)^2 + \beta\,|s|.$$

- When $0 < \beta \leq \sigma$ the minimizer is at $s = \sigma - \beta$.
- When $0 \leq \sigma \leq \beta$, the minimizer is at $s = 0$.



One can generalize this to complex numbers and show that

$$\arg\min_{s \in \mathbb{C}} \frac{1}{2}\,|s-z|^2 + \beta\,|z| = [|z| - \beta]_+ \, \mathrm{e}^{\imath \angle z}.$$

If $\boldsymbol{Y}$ has nonzero singular values $3, 5, 6, 7, 9$ and $\beta = 6$, what is the rank of $\hat{\boldsymbol{X}}$ here?
A: 1 B: 2 C: 3 D: 4 E: 5 **??**

This $\hat{\boldsymbol{X}}$ equals the rank-K approximation to $\boldsymbol{Y}$, where K is answer to previous problem. (?)
A: True B: False **??**

In general, what is $\left\| \hat{\boldsymbol{X}} \right\|_2$ in terms of the notation used throughout here?
A: β B: σ_1 C: $\sigma_1 - \beta$ D: $\sqrt{\sigma_r - \beta}$ E: $[\sigma_1 - \beta]_+$ **??**

In general, if $\|\boldsymbol{Y}\|_2 \geq \beta$, then what is $\left\| \boldsymbol{Y} - \hat{\boldsymbol{X}} \right\|_2$ in terms of the notation used throughout here?
A: β B: σ_1 C: $\sigma_1 - \beta$ D: $\sqrt{\sigma_r - \beta}$ E: $[\sigma_1 - \beta]_+$ **??**

One motivation for these unconstrained formulations is for solving **matrix completion** or **matrix sensing** problems where we must replace $\|\boldsymbol{Y} - \boldsymbol{X}\|_{\mathrm{F}}^2$ with something like $\|\boldsymbol{y} - \boldsymbol{A}\,\mathsf{vec}(\boldsymbol{X})\|_2^2$ where $\boldsymbol{A}$ is a known matrix. These problems do not have closed-form solutions and require iterative methods, and convergence of iterative methods is better understood for convex problems.

Other extensions of low-rank approximation

- Sometimes we might want to retain some columns of the data exactly, leading to a different type of constrained low-rank approximation [22].
- Another variation replaces the Frobenius norm by the spectral norm: (HW)

$$\arg\min_{\boldsymbol{X}} \frac{1}{2} \|\boldsymbol{Y} - \boldsymbol{X}\|_2^2 + \beta \|\boldsymbol{X}\|_*.$$

- Another variation replaces the Frobenius norm by the nuclear norm:

$$\arg\min_{\boldsymbol{X}} \|\boldsymbol{Y} - \boldsymbol{X}\|_* + \beta \|\boldsymbol{X}\|_*.$$

  The solution to this variation is left as an exercise. It seems not to be useful.
- For further generalizations using weighted norms, see [23–25].

If we use two Frobenius norms, the solution has the following general form:

$$\hat{\boldsymbol{X}} = \arg\min_{\boldsymbol{X}} \frac{1}{2}\|\boldsymbol{Y} - \boldsymbol{X}\|_{\mathrm{F}}^2 + \beta \frac{1}{2}\|\boldsymbol{X}\|_{\mathrm{F}}^2 = \sum_{k=1}^{r} \hat{w}_k \boldsymbol{u}_k \boldsymbol{v}_k', \quad \hat{w}_k = h_{\mathrm{FF}}(\sigma_k, \beta).$$

What is $h_{\mathrm{FF}}(\sigma, \beta)$?

A: $[\sigma - \beta]_+$ B: $\sigma H(\sigma - \sqrt{\beta})$ C: $\max(\sigma, \beta)$ D: $\frac{\sigma}{1+\beta}$ E: $(\sigma - \beta)^2$

??

If $\boldsymbol{Y}$ is $M \times N$ with rank r, then what is the rank of the solution $\hat{\boldsymbol{X}}$ here?

A: 0 B: 1 C: r D: $\min(N, M)$ E: None of these.

??

6.5 Choosing the rank or regularization parameter

In all the above low-rank approximation formulations, we must either
- choose the rank K directly, or
- choose the regularization parameter β that influences the rank of $\hat{\boldsymbol{X}}$.

This selection process is non-trivial because when $\boldsymbol{Y} = \boldsymbol{X} + \boldsymbol{\varepsilon}$ and the **latent** matrix $\boldsymbol{X}$ has low-rank:
- $\left\| \hat{\boldsymbol{X}} - \boldsymbol{Y} \right\|_{\mathrm{F}}$: the difference between the noisy data $\boldsymbol{Y}$ and the low-rank approximation $\hat{\boldsymbol{X}}$ always
  - decreases monotonically as K increases (discretely)
  - increases monotonically as β increases (continuously).
- $\left\| \hat{\boldsymbol{X}} - \boldsymbol{X} \right\|_{\mathrm{F}}$: the difference between the low-rank matrix $\boldsymbol{X}$ and the low-rank approximation $\hat{\boldsymbol{X}}$:
  - usually decreases initially as K increases, then increases
  - usually decreases initially as β increases, then increases.

The figures on the next page illustrate these properties.

Often we normalize the difference between $\hat{\boldsymbol{X}}$ and $\boldsymbol{Y}$ or that between $\hat{\boldsymbol{X}}$ and $\boldsymbol{X}$ and define the (unitless) **normalized root mean-squared difference** (**NRMSD**) and **normalized root mean-squared error** (**NRMSE**):

$$\mathrm{NRMSD} = \frac{\left\| \hat{\boldsymbol{X}} - \boldsymbol{Y} \right\|_{\mathrm{F}}}{\left\| \boldsymbol{Y} \right\|_{\mathrm{F}}} \cdot 100\%, \qquad \mathrm{NRMSE} = \frac{\left\| \hat{\boldsymbol{X}} - \boldsymbol{X} \right\|_{\mathrm{F}}}{\left\| \boldsymbol{X} \right\|_{\mathrm{F}}} \cdot 100\%.$$

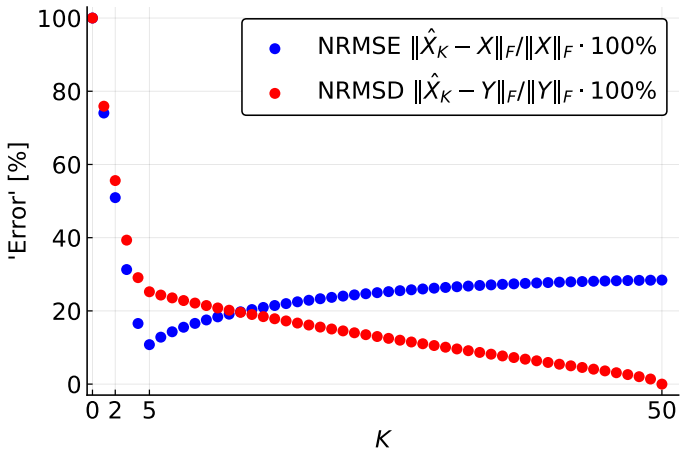
(There are many variations of these definitions in the literature.)

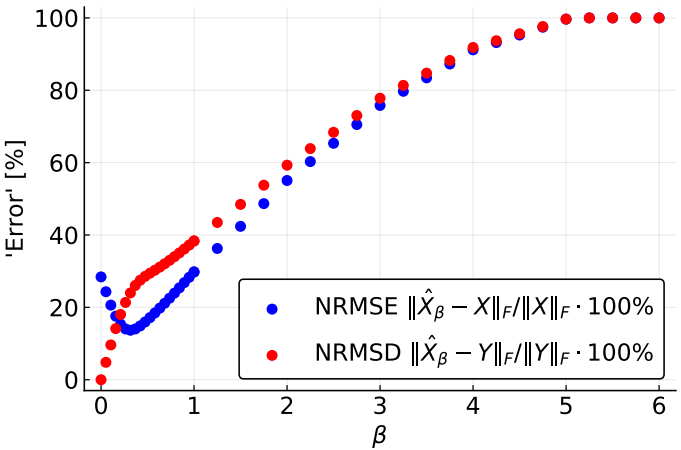
Example. In this case the 100×50 matrix $\boldsymbol{X}$ has rank 5 and $\boldsymbol{Y} = \boldsymbol{X} + \boldsymbol{\varepsilon}$ where $\boldsymbol{\varepsilon}$ is $M \times N$ matrix with IID Gaussian noise.

See JULIA demo:

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/06_lr_sure1.html

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/06_lr_sure1.ipynb





Ideally we would like to find K_* or β_* that minimizes $\left\|\hat{\boldsymbol{X}} - \boldsymbol{X}\right\|_{\mathrm{F}}$, the minimizers of the blue curves above.
Finding either of the choices exactly is impossible because $\boldsymbol{X}$ is unknown in practice!
All we have is $\boldsymbol{Y} = \boldsymbol{X} + \boldsymbol{\varepsilon}$, and usually we are *hoping* that $\boldsymbol{X}$ is low rank, without knowing for sure.

Researchers have developed surrogates for $\left\| \hat{\boldsymbol{X}} - \boldsymbol{X} \right\|_{\mathrm{F}}$ that help choose β.

One notable method is **Stein's unbiased risk estimate** (**SURE**). If $\hat{\boldsymbol{x}}(\boldsymbol{y}; \beta)$ is a **weakly differentiable** estimate of a parameter $\boldsymbol{x} \in \mathbb{R}^d$ from data with Gaussian noise: $\boldsymbol{y} \sim \mathsf{N}(\boldsymbol{x}, \sigma_0^2 \boldsymbol{I})$, then an unbiased estimate of the **risk**, *i.e.*, the mean-squared error (**MSE**), is:

$$\mathsf{MSE}\{\beta\} = \mathsf{E}\left[\|\hat{\boldsymbol{x}}(\boldsymbol{y}; \beta) - \boldsymbol{x}\|^2 \right] = \mathsf{SURE}\{\beta\} \triangleq \underbrace{\|\hat{\boldsymbol{x}}(\boldsymbol{y}; \beta) - \boldsymbol{y}\|_2^2 - d\sigma_0^2 + 2\sigma_0^2 \overbrace{\sum_i \frac{\partial}{\partial y_i} \hat{\boldsymbol{x}}_i(\boldsymbol{y}; \beta)}^{\textbf{Divergence of } \hat{\boldsymbol{x}}(\cdot; \beta)}}_{\text{Independent of } \boldsymbol{x}!} .$$

For a proof, see [wiki]. Note that σ_0^2 is a noise variance, not a singular value!

The SURE approach uses the approximation to select β as follows:

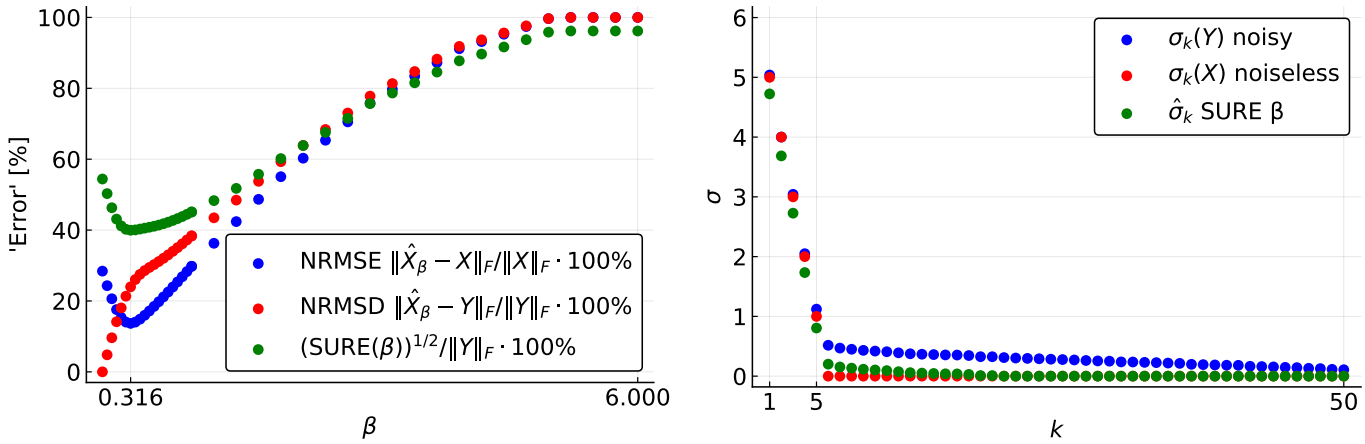
$$\beta_* \triangleq \arg\min_{\beta} \|\hat{\boldsymbol{x}}_\beta - \boldsymbol{x}\|^2 \approx \arg\min_{\beta} \mathsf{E}\left[\|\hat{\boldsymbol{x}}_\beta - \boldsymbol{x}\|^2 \right] = \arg\min_{\beta} \mathsf{SURE}\{\beta\} .$$

The hard thresholding function is *not* weakly differentiable, so we cannot apply the SURE method to rank constrained or rank regularized cases. The soft thresholding function *is* weakly differentiable. Remarkably the divergence expression derived in [20] for any method of the form $\hat{\boldsymbol{X}} = \sum_k \boldsymbol{u}_k h_k(\sigma_k; \beta) \boldsymbol{v}_k'$ turns out to

depend only on the singular values so it is practical to evaluate. When the singular values of $\boldsymbol{Y}$ are distinct:

$$\text{SURE}\{\beta\} = \left\| \hat{\boldsymbol{X}} - \boldsymbol{Y} \right\|_{\text{F}}^2 - MN\sigma_0^2 + 2\sigma_0^2 \left(|M-N| \sum_{i=1}^{\min(M,N)} \frac{h(\sigma_i;\beta)}{\sigma_i} + \sum_{i=1}^{\min(M,N)} \dot{h}_i(\sigma_i;\beta) + 2 \sum_{i \neq j}^{\min(M,N)} \frac{\sigma_i h_i(\sigma_i;\beta)}{\sigma_i^2 - \sigma_j^2} \right).$$

See the demo notebook. A practical challenge is that one must know σ_0^2 to apply this method.



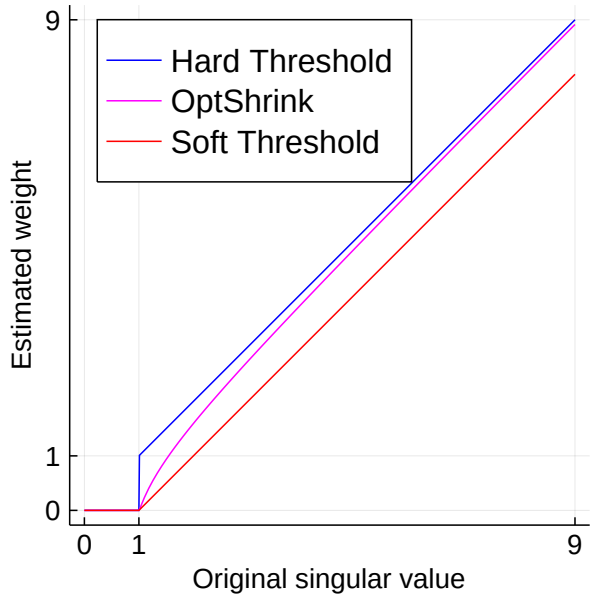
The minimizer β_* of the SURE function is remarkably close to the minimum MSE choice of β.

OptShrink

3002 IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. 60, NO. 5, MAY 2014

OptShrink: An Algorithm for Improved Low-Rank Signal Matrix Denoising by Optimal, Data-Driven Singular Value Shrinkage

Raj Rao Nadakuditi, *Member, IEEE*



Model: $\boldsymbol{Y} = \boldsymbol{X} + \boldsymbol{\varepsilon}$. Random matrix theory says, for quite general noise models (bi-unitarily invariant), the best estimate of $\boldsymbol{X}$ is $\hat{\boldsymbol{X}} = \sum_{k=1}^{r} \hat{w}_k \boldsymbol{u}_k \boldsymbol{v}_k'$, where $\hat{w}_k$ is the best "weight" for the kth SVD component [13].

Algorithm 1 OptShrink: A New Algorithm for Low-Rank Matrix Denoising by Optimal, Data-Driven Singular Value Shrinkage

1: Input: $\widetilde{X} = n \times m$ signal-plus-noise matrix
2: Input: $\widehat{r}$ = Estimate of the effective rank of the latent low-rank signal matrix
3: Compute $\widetilde{X} = \sum_{i=1}^{q} \widehat{\sigma}_i \widehat{u}_i \widehat{v}_i^H$
4: Compute $\widehat{\Sigma}_{\widehat{r}} = \operatorname{diag}(\widehat{\sigma}_{\widehat{r}+1}, \dots \widehat{\sigma}_q) \in \mathbb{R}^{(n-\widehat{r}) \times (m-\widehat{r})}$
5: **for** $i = 1, \dots \widehat{r}$ **do**
6: Compute $\widehat{D}(\widehat{\sigma}_i; \widehat{\Sigma}_{\widehat{r}})$ using (16a) and $\widehat{D}'(\widehat{\sigma}_i; \widehat{\Sigma}_{\widehat{r}})$ using (16b)
7: Compute $\widehat{w}_{i,\widehat{r}}^{\text{opt}} = -2 \dfrac{\widehat{D}(\widehat{\sigma}_i; \widehat{\Sigma}_{\widehat{r}})}{\widehat{D}'(\widehat{\sigma}_i; \widehat{\Sigma}_{\widehat{r}})}$
8: **end for**
9: **return** $\widehat{S}_{\text{opt}} = \sum_{i=1}^{\widehat{r}} \widehat{w}_{i,\widehat{r}}^{\text{opt}} \widehat{u}_i \widehat{v}_i^H$ = denoised estimate of the rank $\widehat{r}$ signal matrix
10: **return** (optional) Compute estimate of MSE using (17a)
11: **return** (optional) Compute estimate of relative MSE using (17b)

For a matrix $X \in \mathbb{K}^{n \times m}$. Define

$$\widehat{D}(z; X) := \frac{1}{n} \operatorname{Tr}\left(z\,(z^2 I - XX^H)^{-1}\right) \cdot \frac{1}{m} \operatorname{Tr}\left(z\,(z^2 I - X^H X)^{-1}\right) \tag{16a}$$

and

$$\widehat{D}'(z; X) := \frac{1}{n} \operatorname{Tr}\left[z\,(z^2 I - XX^H)^{-1}\right] \cdot \frac{1}{m} \operatorname{Tr}\left[-2z^2\,(z^2 I - X^H X)^{-2} + (z^2 I - X^H X)^{-1}\right] + \frac{1}{m} \operatorname{Tr}\left[z\,(z^2 I - X^H X)^{-1}\right] \cdot \frac{1}{n} \operatorname{Tr}\left[-2z^2\,(z^2 I - XX^H)^{-2} + (z^2 I - XX^H)^{-1}\right]. \tag{16b}$$

Note: "$\boldsymbol{X}$" in (16a) and (16b) is the $(n - \hat{r}) \times (m - \hat{r})$ (rectangular) diagonal matrix $\hat{\boldsymbol{\Sigma}}_{\hat{r}}$.
The online code http://web.eecs.umich.edu/~rajnrao/optshrink/ and the equations above are practical only if both n and m are sufficiently small. Here we adapt the approach to allow one of n or m (but not both) to be large.

Let $\boldsymbol{S}$ denote a $K \times L$ rectangular diagonal matrix with entries $s_1, \dots, s_{\min(K,L)}$. We want to evaluate:

$$D(z, \boldsymbol{S}) = \frac{1}{K} \mathsf{trace}\big\{ z(z^2 \boldsymbol{I} - \boldsymbol{S}\boldsymbol{S}')^{-1} \big\} \frac{1}{L} \mathsf{trace}\big\{ z(z^2 \boldsymbol{I} - \boldsymbol{S}'\boldsymbol{S})^{-1} \big\} .$$

Note that $D(z, \boldsymbol{S}) = D(z, \boldsymbol{S}')$ so WLOG assume $K \geq L$ (tall). Then $\boldsymbol{S} = \begin{bmatrix} \boldsymbol{S}_L \\ \boldsymbol{0} \end{bmatrix}$ and

$$z^2 \boldsymbol{I} - \boldsymbol{S}\boldsymbol{S}' = z^2 \boldsymbol{I} - \begin{bmatrix} \boldsymbol{S}_L^2 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} = \begin{bmatrix} z^2 \boldsymbol{I} - \boldsymbol{S}_L^2 & \boldsymbol{0} \\ \boldsymbol{0} & z^2 \boldsymbol{I} \end{bmatrix}$$

$$\implies \mathsf{trace}\big\{ z(z^2 \boldsymbol{I} - \boldsymbol{S}\boldsymbol{S}')^{-1} \big\} = \mathsf{trace}\left\{ z \begin{bmatrix} z^2 \boldsymbol{I} - \boldsymbol{S}_L^2 & \boldsymbol{0} \\ \boldsymbol{0} & z^2 \boldsymbol{I} \end{bmatrix}^{-1} \right\} = \sum_{k=1}^{L} \frac{z}{z^2 - s_k^2} + \frac{K-L}{z}.$$

Similarly, as long as $z \neq s_k$:

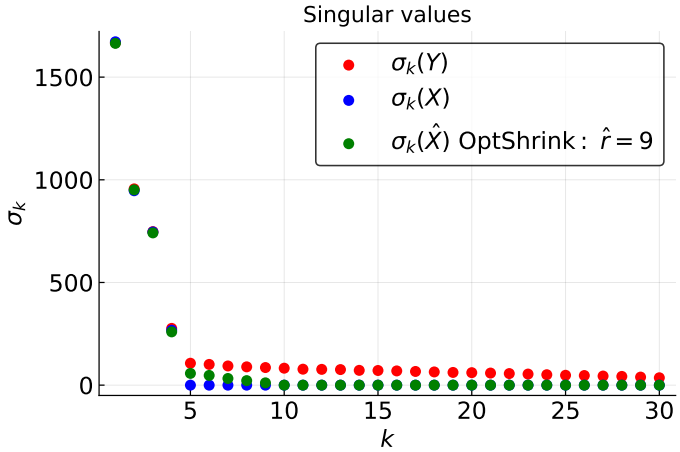
$$\mathsf{trace}\big\{ z(z^2 \boldsymbol{I} - \boldsymbol{S}'\boldsymbol{S})^{-1} \big\} = \sum_{k=1}^{L} \frac{z}{z^2 - s_k^2}, \quad \implies D(z, \boldsymbol{S}) = \frac{1}{KL} \left(\sum_{k=1}^{L} \frac{z}{z^2 - s_k^2} + \frac{K-L}{z} \right) \left(\sum_{k=1}^{L} \frac{z}{z^2 - s_k^2} \right).$$

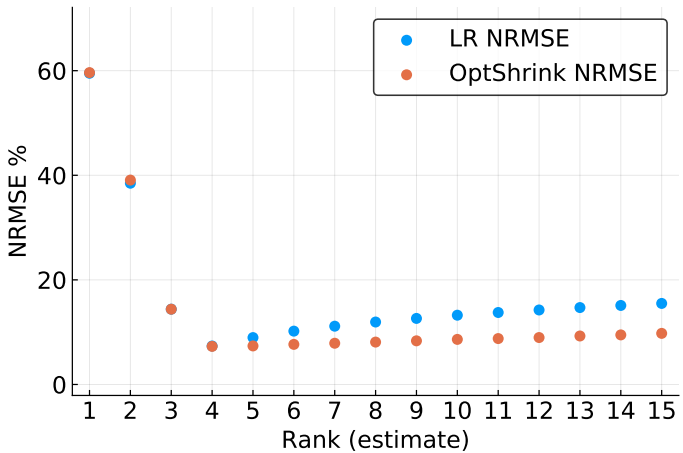
A similar simplification is feasible for $D'(z, \boldsymbol{S})$. (HW)

Combining, one can implement OptShrink to calculate the optimal SVD component weights when at most one of m or n is large.

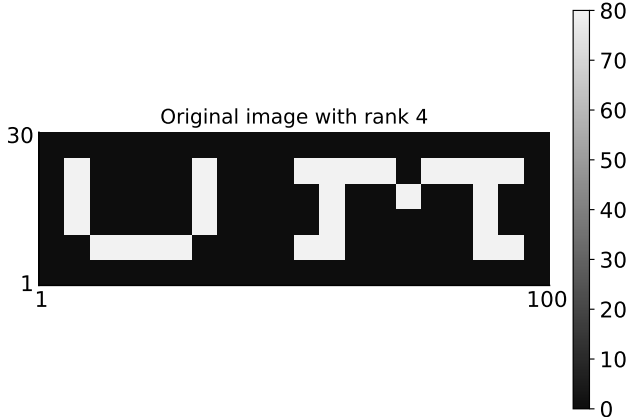
Example showing OptShrink's robustness to rank estimate

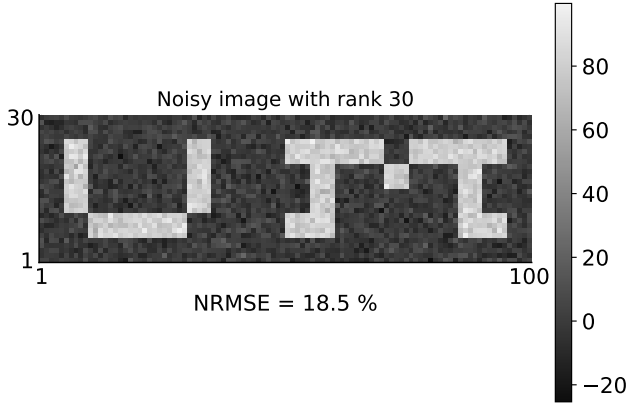
$\boldsymbol{X}$ is a 100×30 logo image (aka 2D array aka matrix) with true rank 4. The noisy data is $\boldsymbol{Y} = \boldsymbol{X} + \varepsilon$.
https://web.eecs.umich.edu/~fessler/course/551/julia/demo/06_optshrink1.html
https://web.eecs.umich.edu/~fessler/course/551/julia/demo/06_optshrink1.ipynb

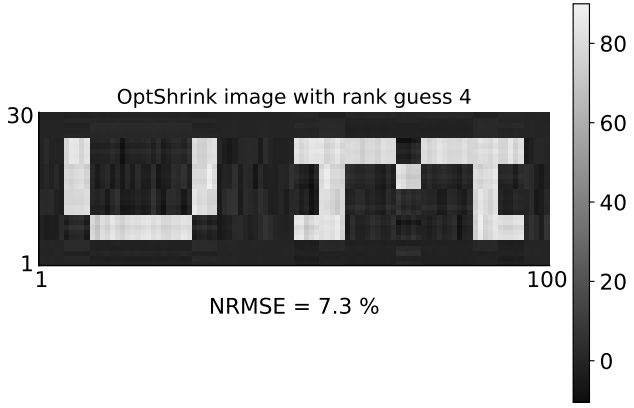


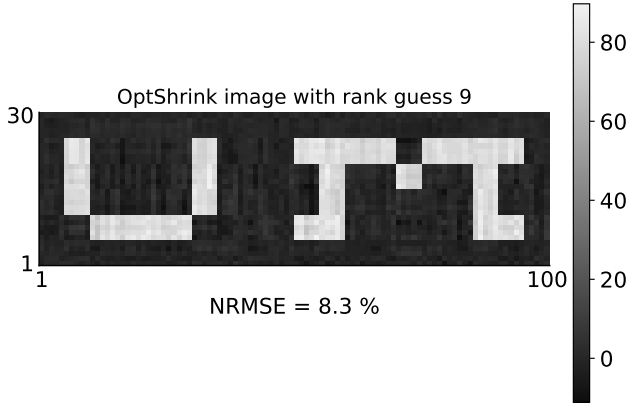


The error of $\hat{\boldsymbol{X}}$ for conventional low-rank approximation increases faster (due to over-fitting noise) than that of OptShrink. OptShrink finds the "threshold" for singular value shrinkage *adaptively* from the data, namely from the tail singular values, using random matrix theory.


Original image with rank 4
30
1
1
100
80
70
60
50
40
30
20
10
0


Noisy image with rank 30
30
1
1
100
NRMSE = 18.5 %
80
60
40
20
0
−20


OptShrink image with rank guess 4
30
1
1
100
NRMSE = 7.3 %
80
60
40
20
0


OptShrink image with rank guess 9
30
1
1
100
NRMSE = 8.3 %
80
60
40
20
0

6.6 Related methods: autoencoders and PCA

Relation to autoencoder with linear hidden layer

An **autoencoder** is a type of **artificial neural network** (**ANN**) that is designed to perform **dimensionality reduction**. An autoencoder consists of an **encoder** and a **decoder**:

$$\boldsymbol{x} \in \mathbb{F}^M \to \boxed{\text{encoder } \phi} \to \boldsymbol{z} \in \mathbb{F}^K \to \boxed{\text{decoder } \psi} \to \hat{\boldsymbol{x}} = \psi(\phi(\boldsymbol{x})) \in \mathbb{F}^M,$$

where typically $K \ll M$, *i.e.*, the dimension of the **code** or **latent variable** $\boldsymbol{z}$ is much less than that of a data point $\boldsymbol{x}$.

The simplest possible case is where the encoder and decoder are each linear, and hence each can be represented via a **matrix**:

$$\phi(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}, \qquad \boldsymbol{A} \in \mathbb{F}^{K \times M} \text{ (wide)}$$
$$\psi(\boldsymbol{z}) = \boldsymbol{B}\boldsymbol{z}, \qquad \boldsymbol{B} \in \mathbb{F}^{M \times K} \text{ (tall)}.$$

In this case, the output is $\hat{\boldsymbol{x}} = \boldsymbol{B}\boldsymbol{A}\boldsymbol{x}$ and the natural approach to **training** this ANN given training data $\boldsymbol{x}_1, \dots, \boldsymbol{x}_N \in \mathbb{F}^M$ is

$$(\hat{\boldsymbol{A}}, \hat{\boldsymbol{B}}) = \underset{\boldsymbol{A} \in \mathbb{F}^{K \times M},\, \boldsymbol{B} \in \mathbb{F}^{M \times K}}{\arg\min} \sum_{n=1}^{N} \|\boldsymbol{x}_n - \boldsymbol{B}\boldsymbol{A}\boldsymbol{x}_n\|_2^2 = \underset{\boldsymbol{A} \in \mathbb{F}^{K \times M},\, \boldsymbol{B} \in \mathbb{F}^{M \times K}}{\arg\min} \|\boldsymbol{X} - \boldsymbol{B}\boldsymbol{A}\boldsymbol{X}\|_{\mathrm{F}}^2, \quad \boldsymbol{X} \triangleq \begin{bmatrix} \boldsymbol{x}_1 & \dots & \boldsymbol{x}_N \end{bmatrix}.$$

Note that the product $\boldsymbol{BAX}$ has at most rank K due to the dimensions of $\boldsymbol{A}$ and $\boldsymbol{B}$. So we can use the low-rank approximation work to observe that one solution to this training problem is

$$\boldsymbol{A} = \boldsymbol{U}_K', \quad \boldsymbol{B} = \boldsymbol{U}_K,$$

where $\boldsymbol{U}_K$ is the first K left singular vectors of $\boldsymbol{U}$, and an **SVD** is $\boldsymbol{X} = \boldsymbol{U\Sigma V}'$. For that choice of $\boldsymbol{A}$ and $\boldsymbol{B}$, and assuming $K \leq \min(M, N)$, the product becomes

$$\begin{aligned}
\boldsymbol{BAX} &= \boldsymbol{U}_K \boldsymbol{U}_K' \underbrace{\boldsymbol{U\Sigma V}'}_{\boldsymbol{X}} = \boldsymbol{U}_K \boldsymbol{U}_K' \begin{bmatrix} \boldsymbol{U}_K & \boldsymbol{U}_{:,(K+1):M} \end{bmatrix} \boldsymbol{\Sigma V}' \\
&= \boldsymbol{U}_K \begin{bmatrix} \boldsymbol{I}_K & \boldsymbol{0}_{K \times M} \end{bmatrix} \begin{bmatrix} \boldsymbol{\Sigma}_K & \boldsymbol{0}_{K \times (N-K)} \\ \boldsymbol{0}_{(M-K) \times K} & \boldsymbol{\Sigma}_{(M-K) \times (N-K)} \end{bmatrix} \begin{bmatrix} \boldsymbol{V}_K & \boldsymbol{V}_{:,(K+1):N} \end{bmatrix}' = \boldsymbol{U}_K \boldsymbol{\Sigma}_K \boldsymbol{V}_K' = \boldsymbol{X}_K,
\end{aligned}$$

namely the optimal rank at most K approximation to $\boldsymbol{X}$.

So performing **low-rank approximation** of training data is comparable to learning an autoencoder with a single linear hidden layer of reduced dimension.

Nonlinear autoencoders are simply generalizations with nonlinear operations such as $\phi(x) = \sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$ where σ is some nonlinear function, $\boldsymbol{W}$ is a $M \times K$ matrix and $\boldsymbol{b} \in \mathbb{F}^M$ is **bias vector**.

Exercise. Revisit the training derivation above in the case where the encoder and decoder are **affine** instead of linear, *i.e.*, $\phi(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}_1$, $\psi(\boldsymbol{z}) = \boldsymbol{B}\boldsymbol{z} + \boldsymbol{b}_2$.

Relation to principal component analysis (PCA)

So far we have focused on low-rank approximation of a matrix. Often the low-dimensional subspaces $\mathsf{span}(\boldsymbol{U}_K)$ and $\mathsf{span}(\boldsymbol{V}_K)$ are more important to us than the overall approximation $\hat{\boldsymbol{A}}_K = \boldsymbol{U}_K \boldsymbol{\Sigma}_K \boldsymbol{V}_K'$.

The **principal component analysis** (**PCA**) approach focuses on learning a linear transform of the data (having **orthonormal** columns) that maximizes variance. PCA starts with a set of N training vectors $\boldsymbol{x}_1, \dots, \boldsymbol{x}_N \in \mathbb{F}^M$ and first subtracts from each the mean of all training vectors to form a first de-meaned data matrix:

$$\boldsymbol{X}_1 \triangleq \begin{bmatrix} \boldsymbol{x}_1 - \boldsymbol{\mu} & \dots & \boldsymbol{x}_N - \boldsymbol{\mu} \end{bmatrix} = \boldsymbol{X} - \boldsymbol{X} \frac{1}{N} \mathbf{1}_N \mathbf{1}_N', \quad \text{where } \boldsymbol{X} \triangleq \begin{bmatrix} \boldsymbol{x}_1 & \dots & \boldsymbol{x}_N \end{bmatrix}, \quad \boldsymbol{\mu} \triangleq \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{x}_n.$$

Note that $\boldsymbol{X}_1 \mathbf{1}_N = \mathbf{0}_M$. Now PCA seeks the (unit norm) linear combination of rows of $\boldsymbol{X}$ (elements of $\boldsymbol{x}$) that maximizes (empirical) variance as follows:

$$\underset{\boldsymbol{u} \,:\, \|\boldsymbol{u}\| = 1}{\arg\max} \frac{1}{N} \sum_{n=1}^{N} \left| \boldsymbol{u}' \boldsymbol{X}_1[:, n] \right|^2 = \underset{\boldsymbol{u} \,:\, \|\boldsymbol{u}\| = 1}{\arg\max} \left\| (\boldsymbol{u}' \boldsymbol{X}_1)' \right\|_2 = \underset{\boldsymbol{u} \,:\, \|\boldsymbol{u}\| = 1}{\arg\max} \left\| \boldsymbol{X}_1' \boldsymbol{u} \right\|_2 = \boldsymbol{u}_1,$$

where an **SVD** of $\boldsymbol{X}_1$ is $\boldsymbol{X}_1 = \boldsymbol{U} \boldsymbol{\Sigma} \boldsymbol{V}'$. Another way of writing this uses a **Rayleigh quotient**:

$$\underset{\boldsymbol{u} \in \mathbb{F}^M - \{\mathbf{0}\}}{\arg\max} \frac{\boldsymbol{u}' \boldsymbol{X}_1 \boldsymbol{X}_1' \boldsymbol{u}}{\|\boldsymbol{u}\|_2^2},$$

where we recognize $\boldsymbol{X}_1 \boldsymbol{X}_1' / N \in \mathbb{F}^{M \times M}$ as an **empirical covariance matrix**.

Having found that $\boldsymbol{u}_1$ is the (unit norm) linear combination that maximizes the (empirical) variance, we can now remove the component along that direction and then seek another linear combination that maximizes the variance of what is left. To remove the component along $\boldsymbol{u}_1$ we construct a new matrix as follows:

$$\boldsymbol{X}_2 \triangleq \boldsymbol{P}_{\mathcal{R}(\boldsymbol{u}_1)}^{\perp} \boldsymbol{X}_1 = (\boldsymbol{I} - \boldsymbol{P}_{\mathcal{R}(\boldsymbol{u}_1)}) \boldsymbol{X}_1 = (\boldsymbol{I} - \boldsymbol{u}_1 \boldsymbol{u}_1') \boldsymbol{X}_1.$$

Note that $\boldsymbol{X}_2 \mathbf{1} = \mathbf{0}$ so we still have a zero-mean array. Now we want to find

$$\arg\max_{\boldsymbol{u} \,:\, \|\boldsymbol{u}\| = 1} \left\| (\boldsymbol{u}' \boldsymbol{X}_2)' \right\|_2 = \arg\max_{\boldsymbol{u} \,:\, \|\boldsymbol{u}\| = 1} \left\| \boldsymbol{X}_1' (\boldsymbol{I} - \boldsymbol{u}_1 \boldsymbol{u}_1') \boldsymbol{u} \right\|_2 = \boldsymbol{u}_2.$$

To see why $\boldsymbol{u}_2$ is the solution here, use an SVD of $\boldsymbol{X}_1$ to write

$$\boldsymbol{X}_2 = (\boldsymbol{I} - \boldsymbol{u}_1 \boldsymbol{u}_1') \boldsymbol{X}_1 = (\boldsymbol{I} - \boldsymbol{u}_1 \boldsymbol{u}_1') \boldsymbol{U} \boldsymbol{\Sigma} \boldsymbol{V}' = \boldsymbol{U} \, \mathsf{diag}\{0, \sigma_2, \sigma_3, \ldots\} \, \boldsymbol{V}',$$

which is almost an SVD of $\boldsymbol{X}_2$ except for a permutation. Clearly the nonzero singular values of $\boldsymbol{X}_2$ are $(\sigma_2, \sigma_3, \ldots, \sigma_r)$ and the spectral norm of $\boldsymbol{X}_2$ is σ_2 and the corresponding left singular vector, which is its principal left singular vector, is $\boldsymbol{u}_2$.
One can continue this reasoning to show that $\boldsymbol{U}_K$, the first K columns of $\boldsymbol{U}$, provides a linear transform with orthonormal columns that maximizes the resulting variances of the **scores** $\boldsymbol{U}_K' \boldsymbol{X}_1$.

Using the same SVD of $\boldsymbol{X}_1$, we have $\boldsymbol{U}_K' \boldsymbol{X}_1 = \boldsymbol{U}_K' \boldsymbol{U} \boldsymbol{\Sigma} \boldsymbol{V}' = \boldsymbol{\Sigma}_K \boldsymbol{V}_K'$, where $\boldsymbol{V}_K$ denotes the first K columns of $\boldsymbol{V}$. Thus instead of storing all MN elements of $\boldsymbol{X}_1$ it suffices to store the MK and NK elements of $\boldsymbol{U}_K$ and $\boldsymbol{V}_K$, reflecting dimensionality reduction.

6.7 Subspace learning

This section discusses an application of low-rank matrix approximation to subspace learning. In this setting, the low-dimensional subspace $\mathcal{R}(\boldsymbol{U}_K)$ (or $\mathcal{R}(\boldsymbol{V}_K)$) is more important to us than the overall approximation $\hat{\boldsymbol{A}}_K = \boldsymbol{U}_K \boldsymbol{\Sigma}_K \boldsymbol{V}_K'$.

In supervised classification, we are given N labeled training data samples $\boldsymbol{x}_{j,1}, \ldots, \boldsymbol{x}_{j,N}$ for each of J classes of objects, where each feature vector $\boldsymbol{x}_{j,n} \in \mathbb{F}^M$. We would like to learn something about the nature of each class so that later when we get a new sample vector $\boldsymbol{x}_0$ we can assess which class it is most like.

One basic approach to classification is the **K-nearest-neighbors method**. When $K = 1$, this method simply chooses the class of the nearest neighbor among the training samples:

$$\hat{j} =$$

This basic approach requires JN comparisons, so complexity grows with training data size. Also, its performance may not improve as N increases when the marginal distributions overlap.

Instead, often we try to learn a model from the training data, and then use fit to the model as the basis for classification. Here we focus on learning a subspace model for each class. Then to classify a test point $\boldsymbol{x}_0$ we simply compute the distance of $\boldsymbol{x}_0$ to each of the J subspaces and see which is closest.

To simplify notation, we drop the class subscript j and focus on learning a subspace for a set of samples for one class type, $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ where each $\boldsymbol{x}_n \in \mathbb{F}^M$.

Saying that all the $\boldsymbol{x}_n$ vectors lie in a subspace of dimension K means that there is some $M \times K$ matrix $\boldsymbol{Q}$ having orthonormal columns such that $\boldsymbol{x}_n \in \mathcal{R}(\boldsymbol{Q})$. The columns of $\boldsymbol{Q}$ form an **orthonormal basis** for the subspace. In practice the data will only *approximately* lie in a subspace, so $\boldsymbol{x}_n \approx \boldsymbol{Q}\boldsymbol{z}_n$ for some coefficient vector $\boldsymbol{z}_n \in \mathbb{F}^K$. We want to *learn* the subspace basis matrix $\boldsymbol{Q}$ from the training data $\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 & \dots & \boldsymbol{x}_N \end{bmatrix}$, a $M \times N$ matrix. Writing $\boldsymbol{x}_n \approx \boldsymbol{Q}\boldsymbol{z}_n$ in matrix form:

$$\underbrace{\boldsymbol{X}}_{M \times N} \approx$$

To find $\boldsymbol{Q}$ and $\boldsymbol{Z}$ we could pursue the following optimization problem:

$$\hat{\boldsymbol{Q}} = \tag{6.15}$$

In this setting $K < \min(M, N)$ so the product $\boldsymbol{Q}\boldsymbol{Z}$ is matrix with (at most) rank K.

Thus this problem is essentially a low-rank approximation problem except that here we really care only about the subspace basis $\boldsymbol{Q}$ and not the coefficients $\boldsymbol{Z}$ (nor their product). The low-rank solution is

$$\hat{\boldsymbol{X}}_K = \sum_{k=1}^{K} \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k' =$$

So to learn the subspace $\boldsymbol{Q}$ we simply use the first K singular vectors of the training data $\boldsymbol{X}$.

After learning a subspace $\boldsymbol{Q}_j$ for each of the J classes, we perform classification of a test point $\boldsymbol{x}_0$ by finding the nearest subspace:

$$\hat{j} = \quad$$

$$f_j(\boldsymbol{x}_0) \triangleq$$

where $\boldsymbol{P}_{\boldsymbol{Q}}\boldsymbol{x}$ denotes the orthogonal projection of $\boldsymbol{x}$ onto the subspace $\mathcal{R}(\boldsymbol{Q})$.

A Discussion section task sheet will explore this method with hand-written digits from the MNIST data.

Learning subspaces via (6.15) is optimal in the Frobenius norm sense of low-rank approximation, but is *not* necessarily optimal for the purposes of classification. An extension called **supervised PCA** aims to find subspaces that are good for both approximation and classification [26–28].

See [29] for a special issue on subspace learning and a generalization called submanifold learning.

Which form above is the most compute efficient?
A: (a) B: (b) C: (c) **??**

For that efficient form, how many multiplies are needed per test sample?
A: $J2M(K+1)$ B: $JM(M+1)$ C: JM^2 D: $J2MK$ **??**

Subspace clustering (Read)

The **subspace learning** approach described above, especially (6.15), assumes that the training data is **labeled** by class type, so it is **supervised learning**. There are also **unsupervised** subspace learning methods that perform **subspace clustering** as part of the subspace learning process [30, 31] [10].

Given unlabeled training data $\boldsymbol{x}_1, \dots, \boldsymbol{x}_N \in \mathbb{F}^M$, in a J-subspace clustering approach we want to learn a set of J orthogonal bases $\boldsymbol{Q}_1, \dots, \boldsymbol{Q}_J \in \mathcal{V}_K(\mathbb{F}^M) \subset \mathbb{F}^{M \times K}$, such that each training point $\boldsymbol{x}_n$ is close to one of the J subspaces $\{\mathcal{R}(\boldsymbol{Q}_j)\}$. One possible formulation is

$$\underset{\boldsymbol{Q}_1,\dots,\boldsymbol{Q}_J \in \mathcal{V}_K(\mathbb{F}^M)}{\arg\min} \; \min_{j_1,\dots,j_N \in \{1,\dots,J\}} \sum_{n=1}^{N} \left\| \boldsymbol{x}_n - \boldsymbol{P}_{\boldsymbol{Q}_{j_n}} \boldsymbol{x}_n \right\|_2^2 .$$

An alternative way of writing this is to use weights where w_{nj} indicates whether the nth training point is assigned to the j class:

$$\underset{\boldsymbol{Q}_1,\dots,\boldsymbol{Q}_J \in \mathcal{V}_K(\mathbb{F}^M)}{\arg\min} \; \min_{\boldsymbol{W} \in \mathcal{W}} \sum_{n=1}^{N} \sum_{j=1}^{J} w_{nj} \left\| \boldsymbol{x}_n - \boldsymbol{P}_{\boldsymbol{Q}_j} \boldsymbol{x}_n \right\|_2^2, \quad \mathcal{W} \triangleq \left\{ \boldsymbol{W} \in \mathbb{R}^{N \times J} : w_{nj} \in \{0,1\}, \sum_{j=1}^{J} w_{nj} = 1 \right\}.$$

These are challenging discrete optimization problems called **integer programming** problems. One can alternate between updating the basis $\{\boldsymbol{Q}_j\}$ and the cluster assignments ($\{j_n\}$ or $\boldsymbol{W}$).
An alternative formulation called **sparse subspace clustering** (**SSC**) relaxes the discrete problem [30].

A related concept for unlabeled data is called **generalized principal component analysis** (**GPCA**) [32].

6.8 Summary

This chapter described several methods for computing a **low-rank approximation** to a matrix, all of which use a **SVD**. It also discussed methods for choosing the **rank** or **regularization parameter**. It applied the methods to sensor localization (**multidimensional scaling**) and to **subspace learning**, which is useful for **classification** problems.

There are two main perspectives in this chapter.

- We started by taking an arbitrary matrix $\boldsymbol{A}$ and finding a low-rank approximation $\hat{\boldsymbol{A}}_K \approx \boldsymbol{A}$.
  This perspective has applications in data compression and in dimensionality reduction.
  In these applications, we want K to be small, but we also want the approximation error $\left\| \hat{\boldsymbol{A}}_K - \boldsymbol{A} \right\|_{\mathrm{F}}$ to be small, and there is a trade-off between the two desires as we vary K.
- We then considered matrix **denoising** applications where we are given a noisy matrix $\boldsymbol{Y}$ that we model as $\boldsymbol{Y} = \boldsymbol{X} + \boldsymbol{\varepsilon}$ where we think $\boldsymbol{X}$ is a low-rank matrix. Then we process $\boldsymbol{Y}$ to make an estimate $\hat{\boldsymbol{X}}$ of the latent matrix $\boldsymbol{X}$.
  In this setting, we want the estimation error $\left\| \hat{\boldsymbol{X}} - \boldsymbol{X} \right\|_{\mathrm{F}}$ to be small, and we do not care much about the approximation error $\left\| \hat{\boldsymbol{X}} - \boldsymbol{Y} \right\|_{\mathrm{F}}$.

Bibliography

[1] M. Udell and A. Townsend. "Why are big data matrices approximately low rank?" In: *SIAM J. Math. of Data Sci.* 1.1 (2019), 144–60 (cit. on p. 6.2).

[2] A. J. Laub. *Matrix analysis for scientists and engineers*. Soc. Indust. Appl. Math., 2005 (cit. on p. 6.2).

[3] C. Eckart and G. Young. “The approximation of one matrix by another of lower rank”. In: *Psychometrika* 1.3 (1936), 211–8 (cit. on pp. 6.3, 6.38).

[4] L. Mirsky. “Symmetric gauge functions and unitarily invariant norms”. In: *The quarterly journal of mathematics* 11.1 (Jan. 1960), 50–9 (cit. on pp. 6.6, 6.17, 6.39).

[5] R. B. Cattell. “The scree test for the number of factors”. In: *Multivariate Behavioral Research* 1.2 (1966), 245–76 (cit. on p. 6.8).

[6] E. Dobriban. *Factor selection by permutation*. 2017 (cit. on p. 6.11).

[7] M. Buehrer, K. P. Pruessmann, P. Boesiger, and S. Kozerke. “Array compression for MRI with large coil arrays”. In: *Mag. Res. Med.* 57.6 (June 2007), 1131–9 (cit. on p. 6.12).

[8] T. Zhang, J. M. Pauly, S. S. Vasanawala, and M. Lustig. “Coil compression for accelerated imaging with Cartesian sampling”. In: *Mag. Res. Med.* 69.2 (Feb. 2013), 571–82 (cit. on p. 6.12).

[9] P. Drineas and I. C. F. Ipsen. “Low-rank matrix approximations do not need a singular value gap”. In: *SIAM J. Matrix. Anal. Appl.* 40.1 (Jan. 2019), 299–319 (cit. on p. 6.14).

[10] Y-L. Yu and D. Schuurmans. “Rank/norm regularization with closed-form solutions: application to subspace clustering”. In: *Proc. 27th Conf. Uncertainty in AI*. 2011, 778–85 (cit. on pp. 6.17, 6.39, 6.62).

[11] P. Verboon and W. J. Heiser. “Resistant lower rank approximation of matrices by iterative majorization”. In: *Comp. Stat. Data Anal.* 18.4 (Nov. 1994), 457–67 (cit. on p. 6.17).

[12] E. J. Candes, X. Li, Y. Ma, and J. Wright. “Robust principal component analysis?” In: *J. Assoc. Comput. Mach.* 58.3 (May 2011), 1–37 (cit. on p. 6.17).

[13] R. R. Nadakuditi. “OptShrink: an algorithm for improved low-rank signal matrix denoising by optimal, data-driven singular value shrinkage”. In: *IEEE Trans. Info. Theory* 60.5 (May 2014), 3002–18 (cit. on pp. 6.22, 6.50).

[14] Y. Shitov. “The nonnegative rank of a matrix: hard problems, easy solutions”. In: *SIAM Review* 59.4 (2017), 794–800 (cit. on p. 6.22).

[15] M. B. Cohen, J. Nelson, and D. P. Woodruff. *Optimal approximate matrix product in terms of stable rank*. 2016 (cit. on p. 6.23).

[16] I. Borg and P. J. F. Groenen. *Modern multidimensional scaling: Theory and applications*. Springer, 2005 (cit. on p. 6.29).

[17] J. J. Moreau. “Proximité et dualité dans un espace hilbertien”. In: *Bulletin de la Société Mathématique de France* 93 (1965), 273–99 (cit. on p. 6.34).

[18] N. Parikh and S. Boyd. “Proximal algorithms”. In: *Found. Trends in Optimization* 1.3 (2013), 123–231 (cit. on p. 6.34).

[19] A. Beck. *First-order methods in optimization*. Soc. Indust. Appl. Math., 2017 (cit. on p. 6.34).

[20] E. J. Candes, C. A. Sing-Long, and J. D. Trzasko. “Unbiased risk estimates for singular value thresholding and spectral estimators”. In: *IEEE Trans. Sig. Proc.* 61.19 (Oct. 2013), 4643–57 (cit. on pp. 6.38, 6.48).

[21] E. J. Candes and T. Tao. “The power of convex relaxation: near-optimal matrix completion”. In: *IEEE Trans. Info. Theory* 56.5 (May 2010), 2053–80 (cit. on p. 6.41).

[22] G. Golub, A. Hoffman, and G. Stewart. “A generalization of the Eckart-Young-Mirsky matrix approximation theorem”. In: *Linear Algebra and its Applications* 88 (Apr. 1987), 317–27 (cit. on p. 6.44).

[23] S. Gu, L. Zhang, W. Zuo, and X. Feng. “Weighted nuclear norm minimization with application to image denoising”. In: *Proc. IEEE Conf. on Comp. Vision and Pattern Recognition*. 2014, 2862–9 (cit. on p. 6.44).

[24] Y. Xie, S. Gu, Y. Liu, W. Zuo, W. Zhang, and L. Zhang. “Weighted Schatten p-norm minimization for image denoising and background subtraction”. In: *IEEE Trans. Im. Proc.* 25.10 (Oct. 2016), 4842–57 (cit. on p. 6.44).

[25] X. Liu, X-Y. Jing, G. Tang, F. Wu, and Q. Ge. “Image denoising using weighted nuclear norm minimization with multiple strategies”. In: *Signal Processing* 135 (June 2017), 239–52 (cit. on p. 6.44).

[26] E. Bair, T. Hastie, D. Paul, and R. Tibshirani. “Prediction by supervised principal components”. In: *J. Am. Stat. Assoc.* 101.473 (2006), 119–37 (cit. on p. 6.61).

[27] E. Barshan, A. Ghodsi, Z. Azimifar, and M. Z. Jahromi. “Supervised principal component analysis: Visualization, classification and regression on subspaces and submanifolds”. In: *Pattern Recognition* 44.7 (July 2011), 1357–71 (cit. on p. 6.61).

[28] A. Ritchie, C. Scott, L. Balzano, D. Kessler, and C. S. Sripada. “Supervised principal component analysis via manifold optimization”. In: *IEEE Data Science Workshop (DSW)*. 2019, 6–10 (cit. on p. 6.61).

[29] Y. Ma, P. Niyogi, G. Sapiro, and R. Vidal. “Dimensionality reduction via subspace and submanifold learning [From the guest editors]”. In: *IEEE Sig. Proc. Mag.* 28.2 (Mar. 2011), 14–126 (cit. on p. 6.61).

[30] R. Vidal. “Subspace clustering”. In: *IEEE Sig. Proc. Mag.* 28.2 (Mar. 2011), 52–68 (cit. on p. 6.62).

[31] J. Lipor, D. Hong, D. Zhang, and L. Balzano. *Subspace clustering using ensembles of K-subspaces*. 2017 (cit. on p. 6.62).

[32] R. Vidal, Y. Ma, and S. Sastry. “Generalized principal component analysis (GPCA)”. In: *IEEE Trans. Patt. Anal. Mach. Int.* 27.12 (Dec. 2005), 1945–59 (cit. on p. 6.62).