

Factor and Component Analysis

Muchang Bahng

Spring 2025

Contents

1	Principal Component Analysis	3
1.1	L2 Residual Minimization Approach	3
1.2	Variance Maximization Approach	6
1.3	Decomposition Solvers	9
1.4	Iterative Solvers	11
1.4.1	Power Method	11
1.4.2	Lanczos Algorithms	13
1.4.3	LOBPCG	15
1.4.4	NIPALS	15
1.4.5	Oja's Neuron	15
1.5	The Importance of Standardizing	18
1.6	Asymptotic Analysis	18
2	Factor Analysis	19
2.1	Degrees of Freedom	20
2.2	Differences from Linear Regression and PCA	20
3	Robust PCA	21
4	Sparse PCA	22
5	Dynamic PCA	23
6	Functional PCA	24
7	Kernel PCA	25
8	Group PCA	26
9	Probabilistic PCA	27
10	Linear Independent Component Analysis	30
11	Slow Feature Analysis	32
12	Sparse Dictionary Learning	33
	Bibliography	35

Principal component analysis (PCA) and factor analysis (FA) originated independently by Pearson in 1901 and Spearman in 1904 [Pea01, Spe04]. Pearson gave the first formal treatment of it not to compute principal components, but to give a new measure of what a “best fit” line means. On the other hand, Spearman—frustrated by the lack of rigorous analyses on nontrivial in psychology—attempted to model the correlation between mental aptitude and sensory tasks. Though their discoveries were independent, the similarity of their models had inevitably caused their developments to coincide.

Note that PCA is similar to linear regression in that it fits some line (or hyperplane) of best fit to some data. However, linear regression—as a model that tries to use the covariates x to predict the response y —attempts to minimize the *residual* $(y - \hat{y})^2$. If we were to flip the model and try to predict x with y , then the best fit line would not be the same. As Pearson puts it, *the most probable stature of a man with a given length of leg l being s , the most probable length of a leg for a man of stature s will not be l* [Pea01]. This is further motivated by the fact that in many data collecting procedures, you do not collect a perfect measurement of x first and then a noisy measurement of y . Rather, you are usually collecting both x and y together at the same time, in which they may both be perceptible to error.

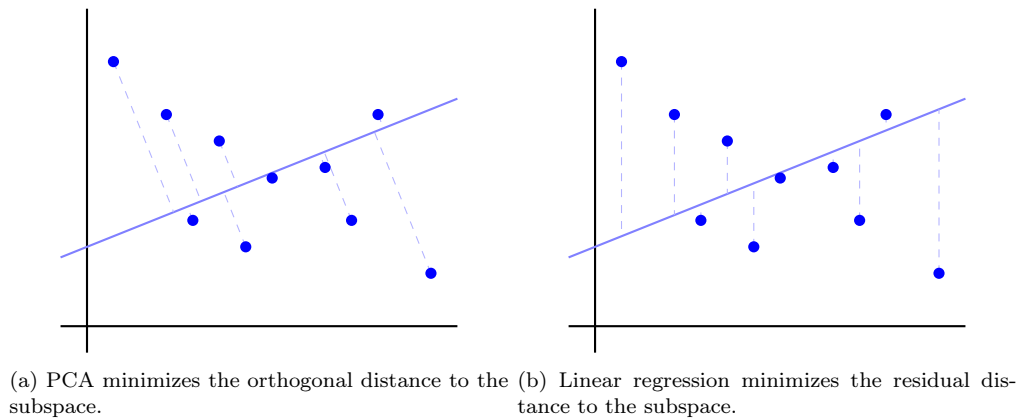


Figure 1: Note that this is in fact different from linear regression as it minimizes the expected *orthogonal distance* to the subspace, rather than the residual distance to the subspace as in linear regression.

1 Principal Component Analysis

Say that we have a random vector $x = (x_1, \dots, x_d)$. These d covariates will naturally be correlated, and we want to ask whether some more fundamental set of independent variables exist [Hot33] such that we can express

$$x = f(v_1, \dots, v_k) \quad (1)$$

Naturally, we think of f as a linear function.

We can think of PCA doing two things. First, it is a dimensionality-reduction algorithm where it takes samples $x \in \mathbb{R}^d$ and projects them into some smaller subspace \mathcal{L} of dimension k . Second, it identifies an orthonormal basis of \mathcal{L} that act as uncorrelated low-dimensional features. Because the projection map is linear and we are working in a lower-dimensional subspace, these new basis vectors are linear combinations of the original basis, which may reduce redundancy. Furthermore, by approximately modeling the original x as a linear combination of these features, we are able to get a more parsimonious representation.

In PCA literature, it is more common to work with row vectors $x \in \mathbb{R}^{1 \times d}$, so linear mappings are realized through right matrix multiplication xA . Furthermore, we will assume that the data are 0-mean.

1.1 L2 Residual Minimization Approach

To give some motivation, we try to find a best fit line in \mathbb{R}^d . A line ℓ can be parameterized by a unit vector v (note that it can be $-pmv!$), and so given some sample x , its projection onto ℓ is $\text{proj}_\ell(x) = \langle x, v \rangle v$. Therefore, the residual is

$$\|x - \langle x, v \rangle v\|^2 = \|x\|^2 - 2\langle x, \langle x, v \rangle v \rangle + \|\langle x, v \rangle v\|^2 \quad (2)$$

$$= \|x\|^2 - 2\langle x, v \rangle^2 + \langle x, v \rangle^2 \|v\|^2 \quad (3)$$

$$= \|x\|^2 - \langle x, v \rangle^2 \quad (4)$$

since $\|v\|^2 = 1$ [Sha19]. Now given a 0-mean random variable x (why we only consider 0-mean RVs will be clear later), our risk is

$$R(v) = \mathbb{E}_x[\|x - (x \cdot v)v\|^2] = \mathbb{E}_x[\|x\|^2] - \mathbb{E}_x[\langle x, v \rangle^2] \quad (5)$$

In practice, we want to minimize our empirical risk. Assume that we have sampled data $x^{(1)}, \dots, x^{(n)} \sim x$. Then,

$$\underset{v \in \mathbb{R}^d, \|v\|=1}{\text{argmin}} \hat{R}(v) = \underset{v \in \mathbb{R}^d, \|v\|=1}{\text{argmin}} \frac{1}{n} \left(\sum_{i=1}^n \|x^{(i)}\|^2 - \sum_{i=1}^n \langle x^{(i)}, v \rangle^2 \right) \quad (6)$$

$$= \underset{v \in \mathbb{R}^d, \|v\|=1}{\text{argmax}} \frac{1}{n} \sum_{i=1}^n \langle x^{(i)}, v \rangle^2 \quad (7)$$

We have our loss function! Now what if we wanted to look for best fitting subspaces in general? Let's first rigorously define such a space.

Definition 1.1 (Principal Subspace)

Let x be a 0-mean random variable in \mathbb{R}^d and let \mathcal{L}^k denote all k -dimensional linear subspaces of \mathbb{R}^d . The k th principal subspace is defined as

$$\ell_k = \underset{\ell \in \mathcal{L}^k}{\text{argmin}} \mathbb{E}_x[\|x - \text{proj}_\ell x\|_2] \quad (8)$$

This isn't a big step from what we had before. We just want to construct a subspace ℓ that minimizes the expected L^2 distance between x and ℓ . Now how do we do such a thing? The most natural extension would

be to identify an orthonormal basis v_1, \dots, v_k , and since

$$\text{proj}_\ell x = \sum_{i=1}^k \text{proj}_{v_i} x \quad (9)$$

our loss can be simplified to

$$R(\ell) = R(v_1, \dots, v_k) = \mathbb{E} \left[\|x - \text{proj}_\ell x\|^2 \right] \quad (10)$$

$$= \mathbb{E} \left[\|x\|^2 - 2 \langle x, \sum_{i=1}^k \text{proj}_{v_i} x \rangle + \left\| \sum_{i=1}^k \text{proj}_{v_i} x \right\|^2 \right] \quad (11)$$

$$= \mathbb{E} \left[\|x\|^2 - 2 \sum_{i=1}^k \langle x, \text{proj}_{v_i} x \rangle + \sum_{i=1}^k \|\text{proj}_{v_i} x\|^2 \right] \quad (12)$$

$$= \mathbb{E} \left[\|x\|^2 - 2 \sum_{i=1}^k \langle x, v_i \rangle^2 + \sum_{i=1}^k \langle x, v_i \rangle^2 \|v_i\|^2 \right] \quad (13)$$

$$= \mathbb{E} \left[\|x\|^2 - \sum_{i=1}^k \langle x, v_i \rangle^2 \right] \quad (14)$$

Now if x was not 0-mean, our intuition would tell us that the principal subspace should pass through its mean, or centroid. In fact [Pea01] showed this for a 1-dimensional subspace.

Lemma 1.1 (Principal Subspace Must Intersect Mean)

Assume that $\mathbb{E}[x] \neq 0$, and by abuse of notation let us denote $\ell + p \in \mathbb{R}^d$ as the affine subspace ℓ that goes through p orthogonally from the origin.

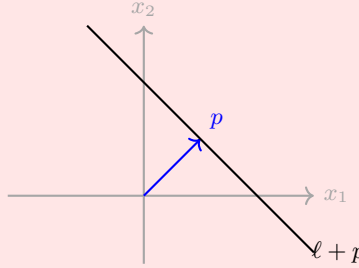


Figure 2: By orthogonal intersection, we mean that for any $w \in \ell$, $\langle w, p \rangle = 0$

Then, we claim that the value of p that minimizes the risk is the average projection distance to the subspace.

$$p = \mathbb{E}[x - \text{proj}_\ell(x)] \quad (15)$$

Proof.

The projection distance of x onto $\ell + p$ is the same as the projection distance from $x - p$ onto ℓ .

Therefore, noting that v_i 's are orthogonal, we can derive

$$R(\ell, p) = \mathbb{E}_x [\|(x - p) - \text{proj}_\ell(x)\|^2] \quad (16)$$

$$= \mathbb{E}_x \left[\left\| (x - p) - \sum_{i=1}^k \text{proj}_{v_i}(x) \right\|^2 \right] \quad (17)$$

$$= \mathbb{E}_x \left[\|x - p\|^2 + \left\| \sum_{i=1}^k \text{proj}_{v_i}(x) \right\|^2 - 2 \left\langle x - p, \sum_{i=1}^k \text{proj}_{v_i}(x) \right\rangle \right] \quad (18)$$

$$= \mathbb{E}_x \left[\|x - p\|^2 - \sum_{i=1}^k \langle x, v_i \rangle^2 + 2 \sum_{i=1}^k \langle x, v_i \rangle \langle p, v_i \rangle \right] \quad (19)$$

To find the minimum, we take the total derivative and set it equal to 0. We are taking the derivative w.r.t. p of an integral w.r.t. x , so we can push the derivative into the integral and solve.

$$0 = \frac{\partial R(\ell, p)}{\partial p} = \mathbb{E}_x \left[\frac{\partial}{\partial p} \left\{ \|x - p\|^2 - \sum_{i=1}^k \langle x, v_i \rangle^2 + 2 \sum_{i=1}^k \langle x, v_i \rangle \langle p, v_i \rangle \right\} \right] \quad (20)$$

$$= \mathbb{E}_x \left[2(p - x) + 2 \sum_{i=1}^k \langle x, v_i \rangle \frac{\partial}{\partial p} \langle p, v_i \rangle \right] \quad (21)$$

$$= \mathbb{E}_x \left[2(p - x) + 2 \sum_{i=1}^k \langle x, v_i \rangle v_i \right] \quad (22)$$

$$= \mathbb{E}_x [2(p - x) + 2 \text{proj}_\ell(x)] \quad (23)$$

and by solving for p , we get

$$p = \mathbb{E}_x [x - \text{proj}_\ell(x)] \quad (24)$$

Therefore, we can just normalize the data to 0 and simply use 14 without having to account for the affine translation p . In the empirical case, we can get rid of the fixed x and find

$$\underset{v_i \in \mathbb{R}^d}{\text{argmax}} \frac{1}{n} \sum_{j=1}^n \sum_{i=1}^k \langle x^{(j)}, v_i \rangle^2, \quad \text{subject to } \|v_i\|^2 = 1, \langle v_i, v_j \rangle = 0 \text{ for } i \neq j \quad (25)$$

By stacking the v_i 's left-to-right in matrix $V_k \in \mathbb{R}^{d \times k}$, we can get a cleaner form of the loss function.

$$\frac{1}{n} \sum_{j=1}^n \sum_{i=1}^k \langle x^{(j)}, v_i \rangle^2 = \frac{1}{n} \sum_{j=1}^n \sum_{i=1}^k (x^{(j)})^T v_i v_i^T x^{(j)} \quad (26)$$

$$= \frac{1}{n} \sum_{j=1}^n (x^{(j)})^T V_k V_k^T x^{(j)} \quad (27)$$

$$= \frac{1}{n} \text{Tr}(X V_k V_k^T X^T) = \frac{1}{n} \text{Tr}(V_k^T X^T X V_k) \quad (28)$$

This leads to an intuitive loss function for PCA.

Theorem 1.1 (Constrained Empirical Risk of k th Principal Subspace)

The empirical risk, or loss function, of PCA is

$$\underset{V \in \mathbb{R}^{d \times k}, V_k^T V_k = I_k}{\text{argmax}} \frac{1}{n} \|X - X V_k V_k^T\|^2 \quad (29)$$

Proof.

By the Frobenius norm expansion and since $V_k^T V_k = I_k$, we have

$$\|X - XV_k V_k^T\|_F^2 = \text{Tr}((X - XV_k V_k^T)^T (X - XV_k V_k^T)) \quad (30)$$

$$= \text{Tr}(X^T X - X^T X V_k V_k^T - V_k V_k^T X^T X + V_k V_k^T X^T X V_k V_k^T) \quad (31)$$

$$= \text{Tr}(X^T X) - 2 \text{Tr}(X^T X V_k V_k^T) + \text{Tr}(V_k V_k^T X^T X V_k V_k^T) \quad (32)$$

$$= \text{Tr}(X^T X) - 2 \text{Tr}(V_k^T X^T X V_k) + \text{Tr}(V_k^T V_k V_k^T X^T X V_k) \quad (33)$$

$$= \text{Tr}(X^T X) - 2 \text{Tr}(V_k^T X^T X V_k) + \text{Tr}(V_k^T X^T X V_k) \quad (34)$$

$$= \text{Tr}(X^T X) - \text{Tr}(V_k^T X^T X V_k) \quad (35)$$

and since the empirical risk does not depend on X , minimizing the Frobenius norm is equivalent to maximizing the second trace term, i.e. 28.

Definition 1.2 (Projection Operator)

Note that there are two distinct projection operators, which are realized through right matrix multiplication.

1. The linear map $V_k : \mathbb{R}^d \rightarrow \mathbb{R}^k$ is a projection operator of the samples x into the component space.
2. The linear map $V_k V_k^T : \mathbb{R}^d \rightarrow \mathbb{R}^d$ is called the **rank-k projection operator** onto the k th principal subspace.

Therefore, $XV_k \in \mathbb{R}^{n \times k}$ is the projection of the dataset into the component space. If we want to get the denoised samples in the sample space \mathbb{R}^d , we project it back out $XV_k V_k^T$.

1.2 Variance Maximization Approach

But we can turn this into a variance maximization problem. Note that $\text{Var}_x[\langle x, v \rangle] = \mathbb{E}_x[\langle x, v \rangle^2] - \mathbb{E}_x[\langle x, v \rangle]^2$, and so we can rewrite our true risk as

$$\underset{v \in \mathbb{R}^d, \|v\|=1}{\text{argmin}} R(v) = \underset{v \in \mathbb{R}^d, \|v\|=1}{\text{argmin}} \mathbb{E}_x[\|x\|^2] - \text{Var}_x[\langle x, v \rangle] - \mathbb{E}_x[\langle x, v \rangle]^2 \quad (36)$$

where the last term vanishes since x is 0-mean, and hence by linearity of expectation $\mathbb{E}_x[\langle x, v \rangle] = \langle \mathbb{E}[x], v \rangle = \langle 0, v \rangle = 0$. In parallel the empirical risk reduces to simply the sample variance.

$$\underset{v \in \mathbb{R}^d, \|v\|=1}{\text{argmax}} \hat{\text{Var}}[\langle x, v \rangle] = \underset{v \in \mathbb{R}^d, \|v\|=1}{\text{argmax}} \frac{1}{n} \left(\sum_{i=1}^n \langle x^{(i)}, v \rangle^2 \right) \quad (37)$$

Therefore, we can think of the L^2 minimization problem as equivalent to a variance maximization approach.

Lemma 1.2 (Variance Maximization Approach)

Minimizing the L^2 distance of a random variable x to a line ℓ in \mathbb{R}^d is equivalent to maximizing the scalar variance in the projected space.

$$\underset{v \in \mathbb{R}^d, \|v\|=1}{\text{argmin}} \mathbb{E}[\|x - \text{proj}_v(x)\|_2^2] = \underset{v \in \mathbb{R}^d, \|v\|=1}{\text{argmax}} \text{Var}_x[\langle x, v \rangle] \quad (38)$$

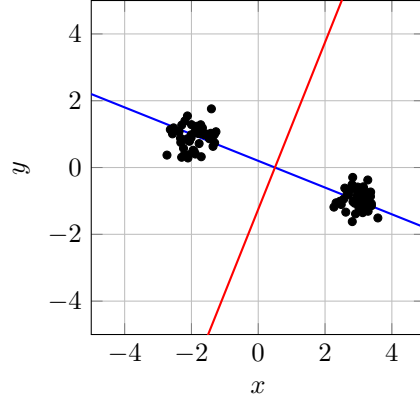


Figure 3: Projecting the dataset onto the blue line seems to retain more variance than projecting onto the red line.

Let's in fact try to directly maximize the variance. If we vertically stack our n data points into a matrix $X \in \mathbb{R}^{n \times d}$, then the projections of this data onto \mathbb{R} is simply $Xv \in \mathbb{R}^n$. Again, since this is 0-mean, the variance is

$$\widehat{\text{Var}}(Xv) = \frac{1}{n} (Xv)^T (Xv) \quad (39)$$

$$= \frac{1}{n} v^T X^T X v \quad (40)$$

$$= v^T \frac{X^T X}{n} v \quad (41)$$

$$= v^T \hat{\Sigma} v \quad (42)$$

where $\hat{\Sigma}$ is the empirical covariance matrix of X . We want to find

$$\max_v v^T \hat{\Sigma} v \text{ subject to } \|v\|^2 = 1 \quad (43)$$

This is a classic Lagrange multiplier problem. We construct the Lagrangian and compute its partial derivatives to set equal to 0.

$$\mathcal{L}(v, \lambda) = v^T \hat{\Sigma} v - \lambda (\|v\|^2 - 1) \quad (44)$$

$$\frac{\partial \mathcal{L}}{\partial v} = 2\hat{\Sigma}v - 2\lambda v = 0 \quad (45)$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = v^T v - 1 = 0 \quad (46)$$

which gives us

$$\hat{\Sigma}v = \lambda v, \quad v^T v = 1 \quad (47)$$

This tells us that v is a unit eigenvector, and the maximizing vector will be the one corresponding to the largest eigenvalue. Essentially, we have reduced this to an eigenvalue problem.

Theorem 1.2 (1st Principal Subspace as Eigenvector)

The first principal subspace of data matrix $X \in \mathbb{R}^{n \times d}$ is spanned by the eigenvector corresponding to the largest eigenvalue of the sample covariance matrix $\hat{\Sigma} = \frac{1}{n} X^T X$.

Now for higher dimensional subspaces, we take the same approach. Going through the same derivation gives

the expected risk in terms of the variance

$$R(v_1, \dots, v_k) = \mathbb{E}[\|x\|^2] - \sum_{i=1}^k \mathbb{E}[\langle x, v_i \rangle^2] \quad (48)$$

$$= \mathbb{E}[\|x\|^2] - \sum_{i=1}^k \text{Var}[\langle x, v_i \rangle] - \mathbb{E}[\langle x, v_i \rangle]^2 \quad (49)$$

By fixing the x 's and going through the same derivation as 39, we get our equivalent empirical risk.

$$\operatorname{argmax}_{v_i \in \mathbb{R}^d} \sum_{i=1}^k \hat{\text{Var}}[\langle x, v_i \rangle] = \operatorname{argmax}_{v_i \in \mathbb{R}^d} \sum_{i=1}^k v_i^T \hat{\Sigma} v_i \quad (50)$$

Theorem 1.3 (Constrained Empirical Risk of k th Principal Subspace)

The empirical risk tells us to find an orthonormal basis that maximizes the sum of the variance of projections.

$$\operatorname{argmax}_{v_i \in \mathbb{R}^d} \sum_{i=1}^k v_i^T \hat{\Sigma} v_i \text{ subject to } \|v_i\|^2 = 1, \langle v_i, v_j \rangle = 0 \text{ for } i \neq j \quad (51)$$

The variance-maximization loss is very insightful, and we may naively think of just taking the unit eigenvectors corresponding to the top k largest eigenvalues. Surprisingly, this greedy approach turns out to be correct.

Let's derive this further

$$\sum_{i=1}^k v_i^T \hat{\Sigma} v_i = \sum_{i=1}^k \text{Tr}(v_i v_i^T \hat{\Sigma}) \quad (52)$$

$$= \text{Tr}(V_k V_k^T \hat{\Sigma}) \quad (53)$$

$$= \text{Tr}(\hat{\Sigma} V_k V_k^T) \quad (54)$$

which again is equal to 28. By the spectral theorem, we can take the eigendecomposition of self-adjoint $\hat{\Sigma} = Q \Lambda Q^T$ with orthogonal matrix Q . Setting $W_k = Q^T V_k$, we have

$$\text{Tr}(\hat{\Sigma} V_k V_k^T) = \text{Tr}(Q \Lambda Q^T V_k V_k^T) = \text{Tr}(\Lambda Q^T V_k V_k^T Q) = \text{Tr}(\Lambda W_k W_k^T) = \sum_{i=1}^d \lambda_i (W_k W_k^T)_{ii} \quad (55)$$

where without loss of generality we have $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_d$. However, we have two constraints. First, since $W_k W_k^T$ is a projection matrix, the eigenvalues must be between 0 and 1. Second, by the cyclic trace property, we have $\text{Tr}(W_k W_k^T) = \text{Tr}(I_k) = k$.¹ So denoting $w_i = (W_k W_k^T)_{ii}$, we have the optimal allocation problem

$$\max \sum_{i=1}^d \lambda_i w_i \text{ subject to } \begin{cases} w_i \in [0, 1] \ \forall i = 1, \dots, d \\ \sum_i w_i = k \end{cases} \quad (56)$$

Since the eigenvalues are decreasing, it doesn't take too much to see that the optimal solution is to just put everything you have into the largest eigenvalues. So we fill the first $w_1 = \dots = w_k = 1$ and the rest $w_{k+1}, \dots, w_d = 0$. Therefore, this solution corresponds to $W_k = (e_1, e_2, \dots, e_k)$, and so

$$V_k = Q W_k = Q_k \quad (57)$$

which is the truncated matrix Q containing the first k eigenvectors of $\hat{\Sigma}$ corresponding to the largest eigenvalues. At this point, it does not suffice to talk about just a principal subspace anymore. We must identify its orthonormal basis, i.e. the eigenvectors.

¹Though it is *not* the case that $W_k W_k^T = I_k$!

Definition 1.3 (Principal Axis)

The eigenvectors v_1, \dots, v_k that span the k th principal subspace are called the top k **principal axes**, or **principal directions**.^a

^aThe terminology is misused and confusing sometimes. See <https://stats.stackexchange.com/questions/88118/what-exactly-is-called-principal-component-in-pca>.

This is similar to taking the first principal component v_1 on X , and then by computing the first principal component on the remaining residuals $X - \text{proj}_{v_1} X$, we get the second principal component, which is guaranteed to be orthogonal. But usually, we end up just computing all eigenvectors at once.

Now how do we know that this sample decomposition is a good approximation to the true decomposition? It comes from the fact that the sample covariance $\hat{\Sigma}$ is a good approximation of the true covariance Σ , which we will later prove using concentration of measure.

Theorem 1.4 (Risk)

The risk satisfies

$$R(k) = \mathbb{E}[\|x - P_k(x)\|^2] = \sum_{j=k+1}^D \lambda_j \quad (58)$$

It is essential that you plot the spectrum in decreasing order. This allows you to analyze how well PCA is working. People often use the “elbow” technique to determine where to choose K , and we value

$$\frac{\sum_{j=1}^k \lambda_j}{\sum_{j=1}^d \lambda_j} \quad (59)$$

accounts for the **variance explained**, which should be high with K low. If you have to go out to dimension $K = 50$ to explain 90% of the variance, then PCA is not working. It may not work because of many reasons, such as there being nonlinear structure within the data.

1.3 Decomposition Solvers**Theorem 1.5 (Construction of the k th Principle Subspace)**

Let $X \in \mathbb{R}^{n \times d}$ be a 0-mean data matrix. Given the SVD with the singular values listed in decreasing order^a

$$X = U\Sigma V^T, \quad U \in \mathbb{R}^{n \times n}, \Sigma \in \mathbb{R}^{n \times d}, V \in \mathbb{R}^{d \times d} \quad (60)$$

^aWe can make it decreasing by permuting the rows/columns of the unitary matrices U, V .

Since $\frac{1}{n} X^T X = \frac{1}{n} V \Sigma U^T U \Sigma V^T = \frac{1}{n} V \Sigma^2 V^T$, the columns of V are the principal axes. Recall from linear algebra that $\Lambda = \Sigma^2$.

Definition 1.4 (Principal Component Scores)

The columns of $U\Sigma$ are called the **principal component scores**.

Algorithm 1.1 (PCA with SVD)

Given a dataset $X \in \mathbb{R}^{n \times d}$, let us denote the rows as x_i , and say that we are looking for a subspace of dimension k .

1. Compute the mean

$$\mu = \frac{1}{n} \sum_{i=1}^n x_i \in \mathbb{R}^d \quad (61)$$

2. Standardize the data $\tilde{X} = X - \mu$, i.e. $\tilde{x}_i = x_i - \mu$.
3. Compute the SVD $\tilde{X} = U\Sigma V^T$.
4. Compute the submatrices $V_k \in \mathbb{R}^{k \times k}$ and $\Sigma_k \in \mathbb{R}^{D \times k}$.
5. Define the projection operator $P_k(x) = \mu + \sum_{j=1}^k \langle x - \mu, v_j \rangle v_j$, the change of basis operator T , and the embedding operator $T^{-1}(z) = \mu + V_k \Sigma_k z$.

A demonstration is done here.

Example 1.1 (Walkthrough)

Say that we have some dataset of 100 points in \mathbb{R}^3 . The data matrix is shown on the right, but in reality I just generated a toy dataset.

<pre> 1 def scatter(n=1000): 2 X_2d = np.random.multivariate_normal(3 np.zeros(2), np.eye(2), n) 4 A = np.array([[1, 1, 1], [-2, 2, 1]]) 5 X_3d = X_2d @ A + 6 np.random.multivariate_normal(7 np.zeros(3), np.eye(3), n) 8 return X_3d </pre>	<pre> 1 [[8.864e-01 3.975e-01 7.009e-01] 2 [-2.065e+00 3.258e+00 1.874e+00] 3 [3.970e-01 -5.400e-01 -3.054e-01] 4 [3.239e+00 -1.999e+00 -1.034e+00] 5 ... 6 [-1.295e-01 9.683e-01 2.861e-01] 7 [-7.097e-01 -4.060e-01 -1.058e+00] 8 [2.284e+00 -2.505e+00 -1.522e+00]] </pre>
--	--

We can take the SVD, which will give us

```

1 In [7]: U, S, Vt = np.linalg.svd(X)
2
3 In [8]: print(U)
4 [[ 4.804e-04  7.852e-02  4.071e-02 ...  5.320e-03  5.034e-02 -9.305e-02]
5  [ 1.420e-01  3.992e-02 -1.229e-02 ... -5.512e-02  4.460e-02  7.192e-02]
6  [-2.456e-02 -3.657e-03 -4.578e-04 ... -1.005e-01 -1.512e-01 -7.300e-02]
7  ...
8  [ 2.909e-02  2.754e-02 -1.075e-01 ...  9.871e-01 -1.325e-02 -3.559e-03]
9  [-8.714e-03 -8.159e-02 -1.436e-01 ... -1.297e-02  9.733e-01 -1.055e-02]
10 [-1.241e-01  2.077e-03 -6.028e-02 ... -3.435e-03 -8.286e-03  9.819e-01]]
11
12 In [9]: print(S)
13 [29.90178039 15.17454164  3.01267412]
14
15 In [10]: print(Vt.T)
16 [[-0.59855644  0.77282378 -0.21088764]
17  [ 0.70716875  0.38606989 -0.59233639]
18  [ 0.37635428  0.50367991  0.77760144]]

```

The `Vt.T` represents V , with its columns being our principal axes, and we wish to plot them along with our data points. We would like to scale them by their variance captured.

1. Take the singular values $(\sigma_1, \sigma_2, \sigma_3) = (29.9, 15.2, 3.0)$.
2. Square them to get the eigenvalues $(\lambda_1, \lambda_2, \lambda_3) = (894, 230, 9)$.
3. Normalize them to get the percent variance captured. $(0.789, 0.203, 0.008)$, and use this as a scale for each eigenvector.

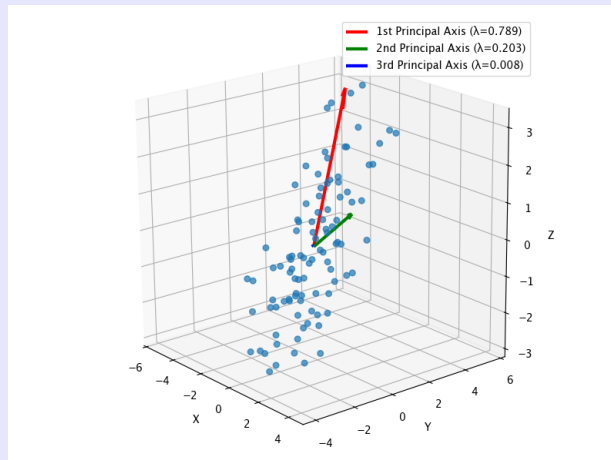


Figure 4: We can see that the data approximately lies on a 2-dimensional subspace of \mathbb{R}^3 .

Example 1.2 (Eigenfaces)

In 1991, Turk and Pentland presented an eigenface method of face recognition by taking the low-rank approximation of a dataset of face images [TP91].

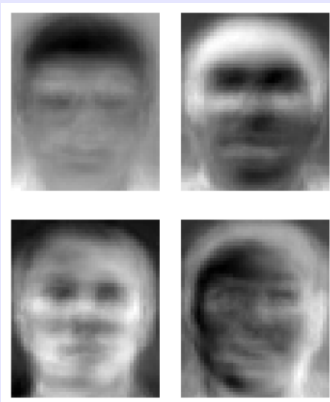


Figure 5: Some eigenfaces from AT&T Labs.

1.4 Iterative Solvers

A disadvantage of decomposition is that the time complexity of SVD on a $n \times m$ matrix is $O(nm \min\{n, m\})$. Therefore, we use iterative methods, which is really just applications of numerical linear algebra.

1.4.1 Power Method

The simplest iterative eigenvalue solver was due to von Mises in 1929 [MP29]. Intuitively, given a diagonalizable matrix that we want to solve for, we can “normalize” the spectrum by dividing all the eigenvectors by λ_1 , the largest eigenvalue. Then by composing these linear maps, all the other eigenvectors should die out while keeping the largest eigenvector in place.

Theorem 1.6 (Convergence of Power Iteration)

Let $A \in \mathbb{R}^{n \times n}$ be a diagonalizable matrix and $x_0 \in \mathbb{R}^n$ any random vector. Given the two assumptions:

1. A has a unique greatest eigenvalue λ_1 .
2. x_0 has a nonzero component in the direction of the eigenvalue v_1 associated with λ_1 .

Then, the sequence (x_t) by defined recursively as

$$x_{t+1} = \frac{Ax_t}{\|Ax_t\|} \quad (62)$$

converges to v_1 .

Proof.

We rewrite the recurrence relation as

$$x_{t+1} = \frac{Ax_t}{\|Ax_t\|} = \frac{A^{t+1}x_0}{\|A^{t+1}x_0\|} \quad (63)$$

Since A is diagonalizable, we can write $A = VSV^{-1}$ where V is the matrix of eigenvectors and S is the diagonal matrix of eigenvalues. Thus:

$$x_t = \frac{A^t x_0}{\|A^t x_0\|} = \frac{(VSV^{-1})^t x_0}{\|(VSV^{-1})^t x_0\|} = \frac{VS^t V^{-1} x_0}{\|VS^t V^{-1} x_0\|} \quad (64)$$

Since x_0 can be expressed as a linear combination of eigenvectors, we write $V^{-1}x_0 = c_1 e_1 + c_2 e_2 + \cdots + c_n e_n$ where e_i are the standard basis vectors and c_i are the coefficients. Therefore:

$$x_t = \frac{VS^t V^{-1}(c_1 v_1 + c_2 v_2 + \cdots + c_n v_n)}{\|VS^t V^{-1}(c_1 v_1 + c_2 v_2 + \cdots + c_n v_n)\|} \quad (65)$$

$$= \frac{VS^t(c_1 e_1 + c_2 e_2 + \cdots + c_n e_n)}{\|VS^t(c_1 e_1 + c_2 e_2 + \cdots + c_n e_n)\|} \quad (66)$$

Factoring out the dominant eigenvalue λ_1 :

$$x_t = \left(\frac{\lambda_1}{|\lambda_1|} \right)^t \frac{c_1}{|c_1|} \frac{v_1 + \frac{1}{c_1} V \left(\frac{1}{\lambda_1} S \right)^t (c_2 e_2 + \cdots + c_n e_n)}{\left\| v_1 + \frac{1}{c_1} V \left(\frac{1}{\lambda_1} S \right)^t (c_2 e_2 + \cdots + c_n e_n) \right\|} \quad (67)$$

Since $|\lambda_i/\lambda_1| < 1$ for $i > 1$, we have $\left(\frac{\lambda_i}{\lambda_1} \right)^t \rightarrow 0$ as $t \rightarrow \infty$. Therefore:

$$\frac{1}{c_1} V \left(\frac{1}{\lambda_1} S \right)^t (c_2 e_2 + \cdots + c_n e_n) \rightarrow 0 \text{ as } t \rightarrow \infty \quad (68)$$

Thus, as $t \rightarrow \infty$:

$$x_t \rightarrow \left(\frac{\lambda_1}{|\lambda_1|} \right)^t \frac{c_1}{|c_1|} \frac{v_1}{\|v_1\|} = \pm v_1 \quad (69)$$

Since we assumed that $c_1 \neq 0$, the sequence (x_t) converges to $\pm v_1$.

Therefore, a direct consequence of this theorem is.

Algorithm 1.2 (Power Iteration to Solve Largest Eigenvector)

Let $A \in \mathbb{R}^{n \times m}$. Then

Algorithm 1 Power Iteration Method

```

1: procedure POWERITERATION( $A$ )
Require:  $A \in \mathbb{R}^{n \times n}$  is a matrix,  $max\_iter > 0$  is maximum iterations
2:    $x \leftarrow$  random vector of unit norm
3:    $\lambda \leftarrow 0$ 
4:   for  $t \leftarrow 1$  to  $max\_iter$  do
5:      $y \leftarrow Ax$ 
6:      $\lambda \leftarrow x^T y$ 
7:      $x \leftarrow \frac{y}{\|y\|}$ 
8:   end for
9:   return  $\lambda, x$ 
10: end procedure

```

Corollary 1.1 (Rate of Convergence of Power Iteration)

The rate of convergence is

$$\frac{\lambda_2}{\lambda_1} \quad (70)$$

1.4.2 Lanczos Algorithms

The time complexity and the slow convergence of the power iteration led to faster algorithms, with one notable class being *Lanczos algorithms*, first developed in 1950 [Lan50].

Definition 1.5 (Krylov Subspace)

Given linear map $A : V \rightarrow V$ and vector $v \in V$, the m th **Krylov subspace** is

$$\mathcal{K}_m(A, v) = \text{span}\{v, Av, A^2v, \dots, A^{m-1}v\} \quad (71)$$

Algorithm 1.3 (Lanczos Algorithm)

The basic idea is to construct an orthonormal basis for the Krylov subspace $\mathcal{K}_m(A, v)$ using a three term recurrence relation. This process transforms the original matrix A into a tridiagonal matrix T that can be used to compute the eigenvectors efficiently.

Algorithm 2 Lanczos Algorithm

```

1: procedure LANCZOS( $A, m$ )
Require: Hermitian matrix  $A$  of size  $n \times n$ , number of iterations  $m$  (default:  $m = n$ )
Ensure:  $n \times m$  matrix  $V$  with orthonormal columns and tridiagonal real symmetric matrix  $T = V^*AV$  of size  $m \times m$ 
2:   Let  $v_1 \in \mathbb{C}^n$  be an arbitrary vector with Euclidean norm 1
3:    $w'_1 \leftarrow Av_1$ 
4:    $\alpha_1 \leftarrow w'^*_1 v_1$ 
5:    $w_1 \leftarrow w'_1 - \alpha_1 v_1$ 
6:   for  $j \leftarrow 2$  to  $m$  do
7:      $\beta_j \leftarrow \|w_{j-1}\|$  (Euclidean norm)
8:     if  $\beta_j \neq 0$  then
9:        $v_j \leftarrow w_{j-1}/\beta_j$ 
10:    else
11:      Pick  $v_j$  as arbitrary vector with Euclidean norm 1 orthogonal to all of  $v_1, \dots, v_{j-1}$ 
12:    end if
13:     $w'_j \leftarrow Av_j - \beta_j v_{j-1}$ 
14:     $\alpha_j \leftarrow w'^*_j v_j$ 
15:     $w_j \leftarrow w'_j - \alpha_j v_j$ 
16:  end for
17:  Let  $V$  be the matrix with columns  $v_1, \dots, v_m$  and

```

$$T = \begin{pmatrix} \alpha_1 & \beta_2 & & & & \\ \beta_2 & \alpha_2 & \beta_3 & & & \\ & \beta_3 & \alpha_3 & \ddots & & \\ & & \ddots & \ddots & \beta_{m-1} & \\ & & & \beta_{m-1} & \alpha_{m-1} & \beta_m \\ & & & & \beta_m & \alpha_m \end{pmatrix} \quad (72)$$

```

18:   return  $V, T$ 
19: end procedure

```

For the runtime,

1. Each iteration consists of an $O(nm)$ matrix multiplication with the rest of the operations $O(n)$.
2. If the matrix is sparse, then this is of order $O(nk)$, where k is the average number of nonzero elements in each row. If k is bounded, then we can compute each iteration in linear time $O(n)$.

Now that we have V, T , what do we do? The following theorem tells us how to compute the original eigenvectors.

Theorem 1.7 ()

If λ is an eigenvalue of T and x is its eigenvector (i.e., $Tx = \lambda x$), then $y = Vx$ is a corresponding eigenvector of A with the same eigenvalue.

Proof.

Let λ be an eigenvalue of T with eigenvector x , so that $Tx = \lambda x$. We need to show that $y = Vx$ is an eigenvector of A with eigenvalue λ .

Since the Lanczos algorithm produces the relation $T = V^*AV$, we have $A = VTV^*$. Therefore:

$$Ay = AVx \quad (73)$$

$$= VTV^*Vx \quad (74)$$

$$= VTIx \quad (75)$$

$$= VTx \quad (76)$$

$$= V(\lambda x) \quad (77)$$

$$= \lambda Vx \quad (78)$$

$$= \lambda y \quad (79)$$

One note of warning. This algorithm tends to be numerically unstable and sometimes there is a loss of orthogonality among the Lanczos vectors due to finite precision arithmetic. We can however just reorthogonalize them in the end.

1.4.3 LOBPCG

1.4.4 NIPALS

1.4.5 Oja's Neuron

With neural networks on the rise, Oja developed a simple neuron function that can compute the greatest eigenvector [Oja82]. Let's start with an extremely simple model of a neuron.

Definition 1.6 (Neuron)

Consider a linear model of a neuron that returns a sum of its inputs according to weighted presynaptic weights $y(x) = w^T x$.

Definition 1.7 (Hebb's Rule)

In neuroscience, **Hebb's rule** (aka Hebbian learning) colloquially states that neurons are activated together have greater strengths, like how muscles grow when you work them more.^a

$$w_{t+1} - w_t = \eta y(x_t)x_t \quad (80)$$

^aNeurons that fire together, wire together.

Since the weights are not constrained at all, we would find that the weights approach infinity if $\eta > 0$. Therefore, Oja restricted the weights so that the p -norm is 1, $0 \leq w_i \leq 1$, which models a form of competition for resources between the neurons [Oja82]. Oja's mathematically formalized this rule in the following.

Theorem 1.8 (Oja's Rule)

Hebbian learning of a neuron can be modeled with

$$w_{t+1} - w_t = \eta y_t(x_t - y_t w_t) \quad (81)$$

To parse this, let's focus on a component element $(x_t)_i$. If x_t represents the neurons firing and $y_t w_t$ represents which neurons' wiring scaled by how much the overall neuron has fired, we want the $y_t w_t$ to move closer to x_t . Therefore, we update w_t in the direction of this difference, and then scale it by ηy_t .

Proof.

Starting with Hebb's rule and applying normalization constraints, we have^a

$$\mathbf{w}_{t+1} = \frac{\mathbf{w}_t + \eta y_t \mathbf{x}_t}{\|\mathbf{w}_t + \eta y_t \mathbf{x}_t\|_p} \quad (82)$$

where $\|\mathbf{v}\|_p = \left(\sum_{j=1}^m |v_j|^p\right)^{1/p}$. For a small learning rate $|\eta| \ll 1$, we can expand this equation as a power series in η . Using the binomial approximation $(1 + \epsilon)^{-1/p} \approx 1 - \frac{\epsilon}{p}$ for small ϵ , we get:

$$\mathbf{w}_{t+1} = \frac{\mathbf{w}_t}{\|\mathbf{w}_t\|_p} + \eta \left(\frac{y_t \mathbf{x}_t}{\|\mathbf{w}_t\|_p} - \frac{\mathbf{w}_t \mathbf{w}_t^T \mathbf{x}_t y_t}{\|\mathbf{w}_t\|_p^{p+1}} \|\mathbf{w}_t\|_p^{p-1} \right) + O(\eta^2) \quad (83)$$

For small η , the higher-order terms $O(\eta^2)$ vanish. We specify a linear neuron where the output is:

$$y_t = \mathbf{x}_t^T \mathbf{w}_t^{(p-1)} \quad (84)$$

where $\mathbf{w}_t^{(p-1)}$ denotes the component-wise power $[\mathbf{w}_t^{(p-1)}]_j = (w_t)_j^{p-1}$. We also specify that our weights are normalized: $\|\mathbf{w}_t\|_p = 1$.

For $p = 2$, this simplifies to $y_t = \mathbf{x}_t^T \mathbf{w}_t$ and $\|\mathbf{w}_t\|_2 = 1$. Substituting these conditions into our expansion gives:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \eta y_t (\mathbf{x}_t - y_t \mathbf{w}_t) \quad (85)$$

^aOja had $p = 2$ in his original paper, but in fact, any type of normalization, even linear, will give the same result without loss of generality.

Finally, we claim that w_t asymptotically converges to the first principal axes.

Theorem 1.9 (Oja's Neuron Convergence to First Principal Component)

Consider a single neuron trained by Oja's rule with weight vector \mathbf{w}_t and input data $\{\mathbf{x}_t\}$ having correlation matrix $R = E[\mathbf{x}\mathbf{x}^T]$. Let \mathbf{q}_1 be the eigenvector corresponding to the largest eigenvalue λ_1 of R . Under the following conditions:

1. The learning rate $\eta(t)$ satisfies $\sum_{t=1}^{\infty} \eta(t) = \infty$ and $\sum_{t=1}^{\infty} \eta(t)^p < \infty$ for some $p > 1$
2. The activation function $y(\mathbf{x}(t))$ is continuously differentiable in both \mathbf{x} and \mathbf{w} with bounded derivatives
3. The correlation matrix R has a unique largest eigenvalue λ_1

Then the weight vector \mathbf{w}_t converges to $\pm \mathbf{q}_1$ as $t \rightarrow \infty$, and the variance of the neuron's output converges to the principal eigenvalue:

$$\lim_{t \rightarrow \infty} \sigma^2(t) = \lim_{t \rightarrow \infty} \langle y^2(t) \rangle = \lambda_1 \quad (86)$$

Proof.

TBD: Go over this. We analyze the convergence using Lyapunov function analysis. Consider the continuous-time version of Oja's rule:

$$\frac{d\mathbf{w}}{dt} = \eta(t) y(t) (\mathbf{x}(t) - y(t) \mathbf{w}(t)) \quad (87)$$

where $y(t) = \mathbf{w}(t)^T \mathbf{x}(t)$.

Taking the expected value and assuming ergodicity of the input process:

$$\frac{d\mathbf{w}}{dt} = \eta(t) E[y(\mathbf{x} - y\mathbf{w})] = \eta(t) E[\mathbf{w}^T \mathbf{x} (\mathbf{x} - \mathbf{w}^T \mathbf{x} \mathbf{w})] \quad (88)$$

This simplifies to:

$$\frac{d\mathbf{w}}{dt} = \eta(t)(R\mathbf{w} - \mathbf{w}^T R \mathbf{w} \mathbf{w}) \quad (89)$$

Since \mathbf{w} is normalized ($\|\mathbf{w}\| = 1$), we have $\mathbf{w}^T R \mathbf{w} = \mathbf{w}^T R \mathbf{w}$. Let $\mathbf{w} = \sum_{i=1}^n c_i \mathbf{q}_i$ where $\{\mathbf{q}_i\}$ are the orthonormal eigenvectors of R with eigenvalues $\{\lambda_i\}$ ordered as $\lambda_1 > \lambda_2 \geq \dots \geq \lambda_n$.

Substituting the eigendecomposition:

$$\frac{d\mathbf{w}}{dt} = \eta(t) \left(\sum_{i=1}^n c_i \lambda_i \mathbf{q}_i - \left(\sum_{i=1}^n c_i^2 \lambda_i \right) \sum_{j=1}^n c_j \mathbf{q}_j \right) \quad (90)$$

For the coefficient c_1 corresponding to the first principal component:

$$\frac{dc_1}{dt} = \eta(t) c_1 \left(\lambda_1 - \sum_{i=1}^n c_i^2 \lambda_i \right) \quad (91)$$

Since λ_1 is the largest eigenvalue and $\sum_{i=1}^n c_i^2 = 1$, we have $\lambda_1 - \sum_{i=1}^n c_i^2 \lambda_i > 0$ whenever $c_i \neq 0$ for $i > 1$.

For $i > 1$:

$$\frac{dc_i}{dt} = \eta(t) c_i \left(\lambda_i - \sum_{j=1}^n c_j^2 \lambda_j \right) \quad (92)$$

Since $\lambda_i < \lambda_1$, the term $(\lambda_i - \sum_{j=1}^n c_j^2 \lambda_j)$ becomes negative as c_1 grows, causing $c_i \rightarrow 0$ for $i > 1$. The conditions on $\eta(t)$ ensure convergence by the Robbins-Monro theorem: the divergent sum condition ensures the algorithm can reach any point in the space, while the convergent power sum condition ensures the noise diminishes sufficiently for convergence.

As $t \rightarrow \infty$, we have $c_1 \rightarrow \pm 1$ and $c_i \rightarrow 0$ for $i > 1$, implying $\mathbf{w}_t \rightarrow \pm \mathbf{q}_1$.

The variance of the output is:

$$\sigma^2(t) = E[y^2(t)] = E[(\mathbf{w}_t^T \mathbf{x}_t)^2] = \mathbf{w}_t^T R \mathbf{w}_t \quad (93)$$

As $\mathbf{w}_t \rightarrow \pm \mathbf{q}_1$, we get:

$$\lim_{t \rightarrow \infty} \sigma^2(t) = \mathbf{q}_1^T R \mathbf{q}_1 = \lambda_1 \quad (94)$$

Algorithm 1.4 (Oja's Rule)

Algorithm 3 Oja's Neuron Algorithm

1: **procedure** OJANEURON($\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T, \eta$)

Require: Input data vectors $\mathbf{x}_t \in \mathbb{R}^n$ for $t = 1, 2, \dots, T$, learning rate $\eta > 0$

Ensure: Weight vector \mathbf{w} that converges to the first principal component

2: Initialize $\mathbf{w}_0 \in \mathbb{R}^n$ randomly with small values

3: **for** $t \leftarrow 1$ to T **do**

4: $y_t \leftarrow \mathbf{w}_{t-1}^T \mathbf{x}_t$

5: $\mathbf{w}_t \leftarrow \mathbf{w}_{t-1} + \eta y_t (\mathbf{x}_t - y_t \mathbf{w}_{t-1})$

6: **end for**

7: **return** \mathbf{w}_T

8: **end procedure**

▷Neural output
▷Oja's learning rule

1.5 The Importance of Standardizing

1.6 Asymptotic Analysis

It turns out that the elements of $\hat{\Sigma}$ are close entry-wise to those of Σ . But if this is true, then does it mean that the eigenvalues of the sample covariance matrix are close to the true eigenvalues of the covariance matrix? It turns out that the answer is no, and we need a proper metric to satisfy this assumption. The metric, as we can guess from linear algebra, is the operator norm, and we will show some results from matrix perturbation theory.

Lemma 1.3 ()

It turns out that

$$\|\hat{\Sigma} - \Sigma\| = O_p\left(\frac{1}{\sqrt{n}}\right) \quad (95)$$

where $\|\cdot\|$ is the operator norm.

Theorem 1.10 (Weyl's Theorem)

If $\hat{\Sigma}$ and Σ are close in the operator norm, then their eigenvalues are close.

$$\|\hat{\Sigma} - \Sigma\| = O_p\left(\frac{1}{\sqrt{n}}\right) \implies |\hat{\lambda}_j - \lambda_j| = O_p\left(\frac{1}{\sqrt{n}}\right) \quad (96)$$

This only talks about their eigenvalues, but this does not necessarily imply that the eigenvectors are close. We need an extra condition.

Theorem 1.11 (David-Kahan Theorem)

If $\hat{\Sigma}$ and Σ are close in the operator norm, and if the eigenvectors of Σ are well-conditioned, then the eigenvectors of $\hat{\Sigma}$ are close to the eigenvectors of Σ . More specifically,

$$\|\hat{v}_j - v_j\| \leq \frac{2^{3/2} \|\hat{\Sigma} - \Sigma\|}{\lambda_j - \lambda_{j+1}} \quad (97)$$

2 Factor Analysis

As we have constantly seen, there are specific themes that run between models. In PCA, we have taken some data x in high-dimension d and reduced it to a lower-dimensional orthogonal representation in \mathbb{R}^k . In other words, for some sample $x \in \mathbb{R}^{1 \times d}$, the projection onto its component space $xV_k \in \mathbb{R}^{1 \times k}$ is a more parsimonious representation with respect to some other basis vectors. The v_1, \dots, v_k are new features that are linear combinations of the old vectors. Are they interpretable? In some cases yes and in most cases no, which is why we also call them *latent variables* that live in a *latent space*.

Another type of model that encodes covariates in a latent space are factor models, which was developed by Spearman in 1904 [Spe04]. The general idea was that we have some d -dimensional random vector x , and we would like to encode it in a k -dimensional random vector f , called the *factors*. Since we are trying to compress the data, generally $k < d$. The first thing that comes to mind is to try and compare how the variables x_i and f_j correlate to each other, and this is exactly what Spearman did.

Before we get into factor models, let's step back and talk more about latent variable models. Colloquially, we would like to find the distribution of some data, whether it'd be (x, y) supervised tasks or x for unsupervised. For the unsupervised case, say that we have some covariates x and we want to find its true distribution p^* . In density estimation so far, what we have done is define a family of distributions $\{p_\theta\}$ and optimize the loss by maximizing the MLE or something else.

$$\min_{\theta} L(p_\theta, p^*) = \max_{\theta} \prod_i p_\theta(x^{(i)}) \quad (98)$$

In order to do this we work with explicitly parameterized distribution families (e.g. Gaussian, Gamma, multinomial, etc.), but this is too simple to model complex things in real like (e.g. the distribution of faces).

Therefore, we consider *implicitly parameterized* probability distributions by “adding” a latent distribution z , creating the joint distribution (x, z) . This may look more complicated, but it captures a much richer family of distributions. For example, we might try modeling x as a function of z , and try to learn some function $x = f(z)$. If we have an accurate function f , we can do many things.

1. Given an x , we might find the closest point on the image of f , perhaps some manifold, as low-rank approximation of x . This dimensionality reduction is essentially what PCA does with projections.²
2. If we can sample from z , then we can forward it through f and can sample from x , making this is a generative model.

Like we do with everything else in math, we take a look at the simplest case when the class of functions are linear. This is known as *linear latent variable modeling*.

$$X = \mu + WZ + \epsilon \quad (99)$$

where the noise ϵ is typically Gaussian and diagonal (but not necessarily the same component-wise variances).

Remember that given data matrix $X \in \mathbb{R}^{n \times d}$, we can take the SVD of it $X = U\Sigma V^T$ where the columns of V represent the principal axes. Since V is orthogonal, $V^T V = V V^T = I_d$, and so $X = X V V^T$. This may seem pretty trivial, but in PCA, we did data compression by removing the last columns of V to get $X \approx X V_k V_k^T$. In a sense we can think of $X V_k \in \mathbb{R}^{n \times k}$ as the projection into the component space and then $(X V_k) V_k^T \in \mathbb{R}^{n \times d}$ as the embedding back into the original space.

In terms of random variables, we have x having a distribution over \mathbb{R}^d . Then, $V V^T x$ is our projected random variable that minimizes the expected L^2 distance between x and $V V^T x$.

²We will in fact extend PCA to probabilistic PCA soon to make it generative.

Definition 2.1 (Factor Analysis)

Let x be a random variable in \mathbb{R}^d . We would like to model it as

$$x - \mu = \Lambda\eta + \epsilon \quad (100)$$

where $\mu = \mathbb{E}[x]$ simply normalizes the distribution, $\eta \in \mathbb{R}^k$ are the **factors** that act as latent variables, $\Lambda \in \mathbb{R}^{d \times k}$ is the **loading matrix** that maps the factors to the samples, and ϵ is an error term.^a We have the following assumptions.

1. $\mathbb{E}[\eta] = 0$.
2. $\text{Var}[\eta] = I_k$ so that the factors are uncorrelated.
3. Λ and ϵ are independent.
4. Sometimes, ϵ is assumed to be an isotropic Gaussian.

^a η is analogous to $V^T x \in \mathbb{R}^k$. Λ is analogous to the embedding $V^T : \mathbb{R}^d \rightarrow \mathbb{R}^k$.

For simplicity, we will omit the mean term, and it is also common notation to use L and f as the loading matrix and factors.

If we were working with the data matrix $X \in \mathbb{R}^{n \times d}$, then we would be using right matrix multiplication, and so our model will look like

$$X = \eta\Lambda + \epsilon \quad (101)$$

where $\eta \in \mathbb{R}^{n \times k}$, $\Lambda \in \mathbb{R}^{k \times d}$, $\epsilon \in \mathbb{R}^{n \times d}$.

Theorem 2.1 (Likelihood)**Lemma 2.1 ()**

It should be clear to us that X should be Gaussian^a and that $\mathbb{E}[X] = \mu$, with

$$\text{Var}[X] = \mathbb{E}[(X - \mu)(X - \mu)^T] \quad (102)$$

$$= \mathbb{E}[(WZ + \epsilon)(Z^T W^T + \epsilon^T)] \quad (103)$$

$$= \mathbb{E}[Wzz^T W^T] + \mathbb{E}[\epsilon\epsilon^T] \quad (104)$$

$$= W\mathbb{E}[zz^T]W^T + \mathbb{E}[\epsilon\epsilon^T] \quad (105)$$

$$= WW^T + \text{diag}(\sigma_1^2, \dots, \sigma_d^2) \quad (106)$$

The W , μ , and σ_i 's can be estimated using MLE methods.

^aSince linear transformations of Gaussians are Gaussian

2.1 Degrees of Freedom**2.2 Differences from Linear Regression and PCA**

3 Robust PCA

4 Sparse PCA

5 Dynamic PCA

6 Functional PCA

7 Kernel PCA

Definition 7.1 (Kernel PCA)

Let N_i be the neighborhood around X_i . Then, we want to find a mapping $W : \mathbb{R}^n \rightarrow \mathbb{R}^k$ that minimizes

$$\min_W \sum_{i=1}^n \left\| X_i - \sum_{j \in N_i} W_{ij} X_j \right\|^2 \text{ where } \sum_j W_{ij} = 1 \quad (107)$$

We can constrain the weights in W so that anything that is not in the neighborhoods are 0.

8 Group PCA

9 Probabilistic PCA

Definition 9.1 (Generative Latent Variable Model)

A **latent variable model** is a model of a distribution $p^*(x)$ over a space \mathcal{X} using implicitly parameterized probability distributions p_θ constructed as such:

1. We define a simple random variable Z over \mathcal{Z} with its distribution $p(z)$, called the **prior**.^a
2. We define a family of functions $\{f_\theta\}$ defined over z and parameterized by θ .
3. We define a way to convert any $f_\theta(z)$ into a distribution $p(x | z)$, called the **likelihood** or **generative component**. There are generally two ways to do this:
 - (a) Let the random variable $X | Z = z$ be an explicitly parameterized distribution, and have $f_\theta(z)$ be the parameters of $X | Z = z$. Therefore, we take the output of $f_\theta(z)$ and plug in these values as the parameters of $X | Z = z$.^b
 - (b) Have f_θ be a transformation of random variables, i.e. $X = f(Z)$. This may result in a conditional pdf that is not explicitly parameterizable.

This defines the family of joint distributions p_θ over $(\mathcal{Z}, \mathcal{X})$. It is easy to sample $(x, z) \sim p_\theta$: sample $z \sim p$, then compute $f_\theta(z)$, use this to define $p_\theta(x | z)$, and finally sample from the likelihood. Therefore, the joint is also of a simple nature.

While we assume simple, explicitly parameterized forms for the prior and the likelihood, we do not assume anything about

1. the **marginal** $p_\theta(x)$. Usually this is an extremely complicated distribution, which is equivalent to

$$p_X(x) = \int_{z \in \mathbb{R}^k} p(x | z) p_Z(z) dz = \mathbb{E}_Z[p(X | Z)] \quad (108)$$

from marginalizing but is computationally impossible to integrate.

2. the **posterior** $p_\theta(z | x)$ that describes the hidden features given some data point. This is also known as the **inference component**. By Bayes rule, we have

$$p_\theta(z | x) = \frac{p_\theta(x | z) p(z)}{p_\theta(x)} \iff p_\theta(z | x) \propto p_\theta(x | z) p(z) \quad (109)$$

which we might be able to sample from using MCMC.

^aAlmost always a uniform or normal distribution suffices. If not, we can constrain it to be factorable (i.e. is the product of its marginal distributions: $p(z) = \prod_i p(z_i)$) so that it is easy to sample from. Occasionally, the stronger assumption of the z_i 's being iid is made.

^bFor example, let $f_\theta(z) = (f_1(z), f_2(z))$. Then we define the corresponding distribution $X | Z = z \sim \mathcal{N}(f_1(z), e^{f_2(z)})$.

We want to take PCA and extend it to be a *generative model*, which allows you to sample data. In regular PCA, we saw that for some $z \in \mathbb{R}^k$ in the latent space, $\hat{x} = \mu + V_k \Sigma_k z$. Therefore, if we just change z from a point to a probability distribution (e.g. Gaussian), we can take a random variable $z \sim \mathcal{N}(0, I)$ from \mathbb{R}^k , and then transform it to get a random variable $x = \mu + U_k \Sigma_k z$, which will give a density.

$$x \sim \mathcal{N}(\mu, (V_k \Sigma_k)(V_k \Sigma_k)^T) = \mathcal{N}(\mu, V_k \Sigma_k U_k^T U_k \Sigma_k V_k^T) = \mathcal{N}(\mu, X_k^T X_k) \quad (110)$$

Note that in here, x is a random variable that we are trying to fit to the data X_k . However, $X_k \in \mathbb{R}^{n \times d}$ with $d \ll n$, and so $X_k^T X_k \in \mathbb{R}^{d \times d}$ is not full rank, and so the distribution is restricted to strictly the k -dimensional subspace $L_k \subset \mathbb{R}^D$. We want to add a bit of noise beyond the subspace, so we add an extra small Gaussian ϵ around it. In general factor analysis above, we set ϵ to have an arbitrary diagonal Gaussian, but for PPCA we just use an isotropic one $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$, giving us

$$x = \mu + U_k \Sigma_k z + \epsilon \implies X \sim \mathcal{N}(\mu, X_k^T X_k + \sigma^2 I) \quad (111)$$

Now rather than treating X_k^T as a data matrix that we use to calculate the principal subspace, we treat it as a parameter matrix $W \in \mathbb{R}^{d \times n}$ that we want to fit [TB99]. Note that PPCA is really a specific instance of factor analysis, and we assume that the latent variable z follows a standard Gaussian $\mathcal{N}(0, 1)$.

Definition 9.2 (Probabilistic PCA)

The **probabilistic PCA** model is a latent factor model with $Z \sim \mathcal{N}(0, I)$ and

$$X = f_\theta(Z) = \mu + (WW^T + \sigma^2 I)^{1/2} Z \quad (112)$$

and $\theta = \{\mu, W, \sigma\}$, which gives

$$X \sim \mathcal{N}(\mu, WW^T + \sigma^2 I) \quad (113)$$

Optimizing this model is actually quite easy.

Theorem 9.1 (MLE of PPCA Model)

Given $x^{(i)} \sim X$ iid, the MLEs for W, μ, σ are

$$\mu_{MLE} = \frac{1}{N} \sum_{i=1}^N x^{(i)} \implies \hat{\mu}_{MLE} = \frac{1}{N} \sum_{i=1}^N x^{(i)} \quad (114)$$

$$\hat{\sigma}_{MLE}^2 = \frac{1}{d-k} \sum_{j=k+1}^d \lambda_j \quad (115)$$

$$W_{MLE} = U_q(\Lambda_d - \hat{\sigma}_{MLE}^2 I_d)^{1/2} R \quad (116)$$

Proof.

Given $x^{(i)} \sim X$ iid, the MLEs for W, μ, σ have a closed form, and model parameter estimation can be performed iteratively and efficiently. We have

$$\mu_{MLE} = \frac{1}{N} \sum_{i=1}^N x^{(i)} \implies \hat{\mu}_{MLE} = \frac{1}{N} \sum_{i=1}^N x^{(i)} \quad (117)$$

and setting the biased MLE estimator of the variance,

$$\widehat{\text{Var}}_{MLE}(\mu_{MLE}) = S = \frac{1}{N} \sum_{i=1}^N (x^{(i)} - \mu_{MLE})(x^{(i)} - \mu_{MLE})^T \quad (118)$$

we can derive the MLE of W .^a We can find the MLE estimate of σ first by taking a look at $C = \text{Var}[X] = WW^T + \sigma^2 I$. It is the sum of positive semidefinite matrices that are also symmetric, so by the spectral theorem it is diagonalizable and has full rank d . But WW^T is rank k , so $d-k$ of the eigenvalues of WW^T is 0, indicating that the same $d-k$ smallest eigenvalues of C is σ^2 . Therefore, we can take the smallest $d-k$ eigenvalues of our MLE estimator of C , which is S , and average them to get our MLE for σ .

$$\hat{\sigma}_{MLE}^2 = \frac{1}{d-k} \sum_{j=k+1}^d \lambda_j \quad (119)$$

We can approximate $WW^T = C - \sigma^2 I \approx S - \hat{\sigma}_{MLE}^2 I$, and by further taking the eigendecomposition $C = U\Sigma U^T \implies WW^T = U(\Sigma - \sigma^2 I)U^T$ and cutting off the last $d-k$ smallest eigenvalues and their corresponding eigenvectors, we can get

$$W_{MLE} = U_q(\Lambda_d - \hat{\sigma}_{MLE}^2 I_d)^{1/2} R \quad (120)$$

where the R just accounts for any unitary matrix.

^aNote that W_{MLE} is not unique. Say that W^* is an MLE, then, for any unitary $U \in \mathbb{R}^{k \times k}$, we have $W^* W^{*T} = (W^* U)(W^* U)^T$.

Now as $\sigma \rightarrow 0$, the density model defined by PPCA becomes very sharp around these d dimensions spanned by the columns of W . At 0, our MLE of W is simplified and we have

$$X = W_{MLE}z + \mu_{MLE} + \epsilon = U_q \Lambda_q^{1/2} z + \mu_{MLE} \quad (121)$$

which essentially reduces to regular PCA. That is, the conditional expected value of z given X becomes an orthogonal projection of $X - \mu$ onto the subspace spanned by the columns of W . Intuitively, we can see that we are estimating the Gaussian, which corresponds to the mean squared distance from each $x^{(i)}$ to ℓ_k .

10 Linear Independent Component Analysis

ICA is a method to separate a multivariate signal into additive, statistically independent components. It does come with a lot of assumptions, and is a specific instance of a linear factor model where $\mu = 0$ and $\epsilon = 0$.

Definition 10.1 (Linear ICA)

In **linear ICA**, we have the simple model.

$$x = Wz \quad (122)$$

In here, $X \in \mathbb{R}^d$ is a mixture vector and $W \in \mathbb{R}^{d \times k}$ is a **mixing matrix**. Both W and z are unknown, and we need to recover them given x . We have 2 strong assumptions.

1. Each component of z is independent (not just uncorrelated). This is an easy enough assumption to intuit.
2. Independent components of z must *not* be Gaussian.^a

^aThis is needed for us to be able to “unmix” the signals. To see why, just suppose z was Gaussian, and so the vector Rz is also Gaussian for any invertible R . Therefore, we could find an infinite number of solutions of form $x = WR^{-1}Rz$ and have no way to separate them.

Algorithm 10.1 (Fitting)

Now let’s see how linear ICA actually estimates W and z . Once W is estimated, the latent components of a given test mixture vector, x^* is computed by $z^* = W^{-1}x^*$. So now all there’s left to do is to estimate W , which we want to estimate so that $W^{-1}x$ is far from Gaussian. The reason for this is that given a bunch of independent non-Gaussian h_i ’s, if we mix them with a matrix that is not $\pm I$, then by CLT, a linear combination of random variables will tend to be Gaussian, and so for an arbitrary W we would expect x to be Gaussian. Therefore, what we want to do is guess some matrix A , and compute

$$Ax = AWz \quad (123)$$

and if we get things right, $A \approx W^{-1}$, and the result of Ax would look pretty non-Gaussian. If it is not the case, then AW will still be some mixing matrix, and so Ax would look Gaussian. So now the question reduces to how do we choose this A ? There are multiple ways to measure non-Gaussianity:

1. The absolute or squared kurtosis, which is 0 for Gaussians. This is a differentiable function w.r.t. W , so we can try maximizing it. This is done for the sample kurtosis, of course.
2. Another measure is by maximizing the neg-entropy.

There are further ambiguities with ICA regarding uniqueness of a best representation. For one, we can only estimate the latent components up to a scaling factor since we will still get

$$x = (\alpha W)\left(\frac{1}{\alpha}z\right) \text{ for some } \alpha > 0 \quad (124)$$

We can fix this by forcing $\mathbb{E}[z_i^2] = 1$. However, there is still an ambiguity for the sign of hidden components, but this is insignificant in most applications. Second, we can estimate the components up to permutation. We have

$$x = WP^{-1}Pz \quad (125)$$

for some permutation matrix P .

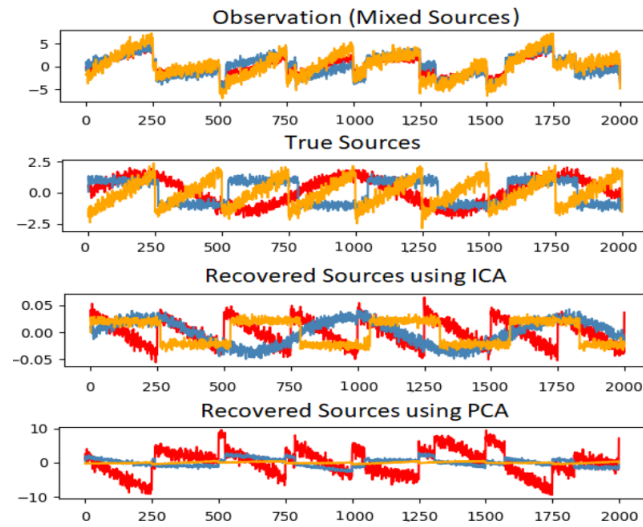


Figure 6: We can perform this on three mixed signals with additive noise, and ICA does very well, though again some recovered signals are scaled or permuted weirdly.

11 Slow Feature Analysis

Slow feature analysis is another special case of a linear factor model that uses information from time signals to learn invariant features. It is motivated by a general principle called the **slowness principle**. The idea is that the important characteristics of scenes change very slowly compared to the individual measurements that make up a description of a scene. For example, in computer vision, individual pixels can change very rapidly. If a zebra moves from left to right across the image, an individual pixel will rapidly change from black to white. By comparison, the feature indicating whether a zebra is in the image will not change at all, and the feature describing the zebra's position will change slowly. Therefore, we want to regularize our model to learn features that change slowly over time.

We can apply the slowness principle to any differentiable model trained with gradient descent. That is, we can add the following term to the loss function:

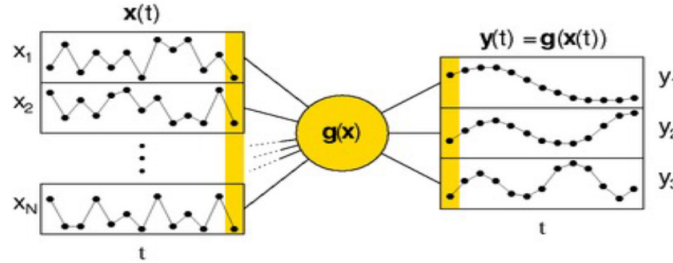
$$\lambda \sum_i d(f(x^{(t+1)}), f(x^{(t)})) \quad (126)$$

where λ is a hyperparameter determining the strength of the slowness regularization term, t is the time index, f is the feature extractor to be regularized, and d is the distance between $f(x^{(t)})$ and $f(x^{(t+1)})$. A common choice for d is the mean squared difference.

Essentially, given a set of time-varying input signals $x^{(t)}$, SFA learns a nonlinear function f that transforms x into slowly-varying output signals y . Obviously, we can't just take some trivial function like $f = 0$, so we have the following constraints

$$\mathbb{E}_t[f(x^{(t)})_i] = 0 \quad (127)$$

$$\mathbb{E}_t[f(x^{(t)})_i^2] = 1 \quad (128)$$



We can restrict the nonlinear f to some subspace of functions, and this becomes a standard optimization problem where we solve

$$\min_{\theta} \mathbb{E}_t[(f(x^{(t+1)})_i - f(x^{(t)})_i)^2] \quad (129)$$

12 Sparse Dictionary Learning

Latent variables can help us represent data in lower dimensions, but another advantage is that we can get *sparse* representations as well. What we want to do in sparse coding is that for each input $x^{(i)}$, we want to find a latent representation $z^{(i)}$ such that it is sparse (i.e. has many 0s) and also we can reconstruct the original input $x^{(i)}$ well. We have basically two things to optimize: the latent representations z and the decoding mechanism, which we can do with a *dictionary matrix* D . Note that we are optimizing for *both* the latent encodings and the decoding mechanism, and so this isn't a generative model.

Definition 12.1 (Sparse Dictionary Encoding Model)

The **sparse dictionary encoding model** is a representation model defined

$$X = g_D(Z) = DZ \quad (130)$$

where $D \in \mathbb{R}^{d \times k}$ is a **dictionary matrix** that decodes the latent $Z \in \mathbb{R}^k$ to $X \in \mathbb{R}^d$. Note that both the $z^{(i)}$'s and D are optimized, so we want to perform the *joint* optimization^a

$$\min_D \frac{1}{N} \sum_{i=1}^N \min_{z^{(i)}} \underbrace{\frac{1}{2} \|x^{(i)} - Dz^{(i)}\|_2^2}_{\text{reconstruction error}} + \underbrace{\lambda \|z^{(i)}\|_1}_{\text{sparsity penalty}} \quad (131)$$

^aTo break this term down, let's just assume that we have a fixed dictionary D . Then, we just need to minimize with respect to each $h^{(t)}$. Now we can add the dictionary parameter back again.

Note that the reconstruction, or decoding, of $x = Dz$ is linear and explicit, but if we want to encode $x \mapsto z$, we need to substitute the x into the term above and minimize it w.r.t. D and z to solve it. Therefore, this encoder is an implicit and *nonlinear* function of x .

$$\hat{\mathbf{x}}^{(t)} = \mathbf{D} \mathbf{h}(\mathbf{x}^{(t)}) = \sum_{\substack{k \text{ s.t.} \\ h(\mathbf{x}^{(t)})_k \neq 0}} \mathbf{D}_{:,k} h(\mathbf{x}^{(t)})_k$$

Figure 7: We can reconstruct an image of a seven as a linear combination of a set of images. Note that each of the images of strokes are columns of W and the coefficients make up the sparse vector h .

Let's think about how we can optimize the objective function w.r.t. h , keeping D constant. We can do stochastic gradient descent, which gives us the steps

$$\nabla_{h^{(t)}} \mathcal{L}(x^{(t)}) = D^T(Dh^{(t)} - x^{(t)}) + \lambda \text{sign}(h^{(t)}) \quad (132)$$

but this wouldn't achieve sparsity since it overshoots the 0 all the time. Therefore, we can clip it, or we can use proximal gradient descent/ISTA to take a step, and shrink the parameters according to the L1 norm.

$$h^{(t)} = h^{(t)} - \alpha D^T(Dh^{(t)} - x^{(t)}) \quad (133)$$

$$h^{(t)} = \text{shrink}(h^{(t)}, \alpha\lambda) \quad (134)$$

where $\text{shrink}(a, b) = [\dots, \text{sign}(a_i) \max(|a_i| - b_i, 0), \dots]$. This is guaranteed to converge if $1/\alpha$ is bigger than the largest eigenvalue of $D^T D$.

Bibliography

- [Hot33] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24:498–520, 1933.
- [Lan50] Cornelius Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Natl. Bur. Stand. B*, 45:255–282, 1950.
- [MP29] R. V. Mises and H. Pollaczek-Geiringer. Praktische Verfahren der Gleichungsauflösung . *Zeitschrift Angewandte Mathematik und Mechanik*, 9(1):58–77, January 1929.
- [Oja82] Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15(3):267–273, 1982.
- [Pea01] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11):559–572, 1901.
- [Sha19] Cosma Rohilla Shalizi. *Advanced Data Analysis from an Elementary Point of View*. Cambridge University Press, 2019.
- [Spe04] C. Spearman. "general intelligence," objectively determined and measured. *The American Journal of Psychology*, 15(2):201–292, 1904.
- [TB99] Michael E. Tipping and Christopher M. Bishop. Probabilistic Principal Component Analysis. *Journal of the Royal Statistical Society Series B*, 61(3):611–622, 1999.
- [TP91] M.A. Turk and A.P. Pentland. Face recognition using eigenfaces. *Journal of Cognitive Neuroscience*, pages 586–591, 1991.