Lecture 16: Dimensionality Reduction Notes

Konstantinos Chatzilygeroudis
costashatz@upatras.gr

December 22, 2025

Contents

1 Why dimensionality reduction?

In many modern applications we observe data vectors in a *high-dimensional* ambient space. Concretely, we may collect

$$\mathbf{x}_i \in \mathbb{R}^D, \qquad i = 1, \dots, N,$$

where the dimension D can be very large: pixels in images, readings from multiple sensors, engineered features, or learned embeddings. Working directly in $\mathbb{R}^D$ is often expensive (computationally and statistically), and it may also obscure the underlying structure of the phenomenon we are studying.

A key motivating idea is that, although observations live in $\mathbb{R}^D$, the mechanism that generated them may be much simpler. In other words, the data may be well-explained by a *small number of degrees of freedom*. A common modeling viewpoint is that each observation can be approximated as

$$\mathbf{x}_i \approx g^{-1}(\mathbf{z}_i), \qquad \mathbf{z}_i \in \mathbb{R}^d, \;\; d \ll D,$$

where $\mathbf{z}_i$ is a low-dimensional *latent representation* and $g^{-1}(\cdot)$ is a (possibly nonlinear) generative mapping into the observation space. The inequality $d \ll D$ captures the core promise of dimensionality reduction: the data may have *intrinsic* dimension d even if it is measured in a much larger ambient space.

1.1 What are we trying to achieve?

The practical goal is to construct a representation

$$\mathbf{z}_i = g(\mathbf{x}_i) \in \mathbb{R}^d \quad (d \ll D)$$

that preserves the *important* structure of the data. Here the mapping $g(\cdot)$ is now understood as an *encoder* (a reduction map) that takes us from the high-dimensional observation space to a lower-dimensional coordinate system. The word "important" depends on the task, but several recurring objectives appear across domains.

First, dimensionality reduction enables **compression**: we can store or transmit $\mathbf{x}_i$ using fewer numbers by working with $\mathbf{z}_i$ instead of the full vector. Second, it supports **visualization**: choosing $d = 2$ or $d = 3$ lets us plot the data and inspect clusters, trends, or outliers. Third, it can act as **denoising**: if some directions in $\mathbb{R}^D$ mainly contain noise, projecting away from those directions can yield cleaner representations. Finally, it improves **learning efficiency**: reducing dimension typically reduces computation and, in many settings, lowers the sample complexity by shrinking the space in which we must estimate patterns.

1.2 The curse of dimensionality

As the ambient dimension D grows, familiar geometric intuition can become unreliable. With a fixed number of samples N, the data become *sparse*: the fraction of the volume of $\mathbb{R}^D$ that is "covered" by the samples becomes vanishingly small. Consequently, many tasks that depend on local neighborhoods (such as nearest neighbors, density estimation, or nonparametric regression) can become much harder unless we dramatically increase N.

Another symptom is that distances can become less informative. In high dimensions, many points may appear "equally far" under common metrics, weakening methods that rely on comparing distances. In addition, estimating quantities such as covariances or probability densities can require much larger datasets, because there are more degrees of freedom to learn. Informally, one often observes the following trade-off:

$$\text{more features} \quad \Rightarrow \quad \text{more parameters to estimate} \quad \Rightarrow \quad \text{more data needed.}$$

Dimensionality reduction is one of the main tools for mitigating this problem by identifying a lower-dimensional structure that still captures the essential variation.

1.3 Redundancy and correlations

High-dimensional feature vectors frequently contain correlated or redundant components. For example, consider

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_D \end{bmatrix}, \qquad \text{with relationships like} \qquad x_2 \approx 2x_1, \;\; x_5 \approx x_1 + x_3, \;\; \ldots$$

Even though $\mathbf{x} \in \mathbb{R}^D$, such approximate linear dependencies indicate that the data do not freely explore all D directions. Instead, the observations concentrate near a much lower-dimensional set. Geometrically, this set might be a line or a plane (a linear subspace), or it could be a curved object embedded in $\mathbb{R}^D$ (a manifold). Dimensionality reduction aims to describe this structure using a smaller number of coordinates.

1.4 A geometric picture: subspaces and low-rank structure

A particularly useful geometric model assumes the data lie approximately in a d-dimensional linear subspace. One way to express this is

$$\mathbf{x}_i \approx \boldsymbol{\mu} + \mathbf{W}\mathbf{z}_i, \qquad \mathbf{W} \in \mathbb{R}^{D \times d}, \;\; \mathbf{z}_i \in \mathbb{R}^d, \;\; d \ll D.$$

Here $\boldsymbol{\mu}$ is the mean of the data (a translation), and the columns of $\mathbf{W}$ span a d-dimensional subspace that captures most of the variation across samples. The vector $\mathbf{z}_i$ contains the coordinates of $\mathbf{x}_i$ in that subspace. Directions orthogonal to the span of $\mathbf{W}$ are assumed to carry little signal and often correspond to noise or minor variations.

This picture provides the simplest intuition behind methods like Principal Component Analysis (PCA): find a low-dimensional subspace that approximates the data well, and represent each point by its coordinates in that subspace.

1.5 Formalizing "keep what matters" via variance and reconstruction

To turn the idea of "preserving what matters" into mathematics, a common principle is to preserve *variance* (or energy). Suppose $\mathbf{P}$ is a projection onto a d-dimensional subspace.

Then a natural requirement is that the projection does not distort the data too much, meaning that

$$\|\mathbf{x}_i - \mathbf{P}\mathbf{x}_i\|^2 \text{ is small for most } i.$$

This quantity is the *reconstruction error*: it measures how much information is lost when we replace $\mathbf{x}_i$ by its projected version $\mathbf{P}\mathbf{x}_i$. Minimizing reconstruction error is closely related to maximizing how much of the data's spread (variance) is retained by the projection. Intuitively, if we want to keep as much signal as possible with only d directions, we should choose directions along which the data vary the most. Note that $\mathbf{P}\mathbf{x}_i \in \mathbb{R}^D$ is the projection of $\mathbf{x}_i$ onto a d-dimensional subspace, not the d-dimensional coordinate vector itself. The low-dimensional coordinates are given by $\mathbf{z}_i = \mathbf{W}^\top \mathbf{x}_i \in \mathbb{R}^d$.

Thus, for many dimensionality reduction methods (especially linear ones), two equivalent viewpoints recur: (i) choose directions of **maximum variance**, or (ii) choose a representation that minimizes **reconstruction error**. These principles will later lead directly to the formulation and solution of principal component analysis.

2 Linear algebra, geometry, and decompositions

Dimensionality reduction is, at its core, a geometric idea: we seek a low-dimensional set (often a subspace) that captures the important structure of the data. To make this viewpoint precise, we need a small toolkit from linear algebra. In particular, we will repeatedly interpret matrices as *geometric transformations* and use matrix decompositions to understand how those transformations act on vectors, lengths, and directions.

2.1 Symmetric matrices: orthogonal axes and eigenvectors

A helpful starting point is a symmetric 2×2 matrix

$$\mathbf{A} = \begin{bmatrix} a & b \\ b & c \end{bmatrix}, \qquad \mathbf{A} = \mathbf{A}^\top.$$

It defines a linear map $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ in $\mathbb{R}^2$. A key geometric fact is that, in the symmetric case, the action of $\mathbf{A}$ is especially clean: it maps the *unit circle* to an *ellipse* whose principal axes are *orthogonal*. Moreover, those principal axes are aligned with the eigenvectors of $\mathbf{A}$. Eigenpairs $(\lambda_k, \mathbf{v}_k)$ satisfy

$$\mathbf{A}\mathbf{v}_k = \lambda_k \mathbf{v}_k.$$

This equation has a simple geometric meaning: if a vector points exactly along an eigenvector direction $\mathbf{v}_k$, then applying $\mathbf{A}$ does not rotate it—it only scales it by the factor λ_k. Thus, the eigenvectors identify directions that are invariant under the transformation (up to scaling), and the eigenvalues quantify the amount of stretching or shrinking along those directions.

To see the circle-to-ellipse story explicitly, consider the set of all unit vectors,

$$\mathcal{C} = \{\mathbf{x} \in \mathbb{R}^2 : \|\mathbf{x}\| = 1\}.$$

Applying the transformation to every point on the circle yields

$$\mathcal{E} = \{\mathbf{A}\mathbf{x} : \mathbf{x} \in \mathcal{C}\}.$$

When $\mathbf{A}$ is symmetric, $\mathcal{E}$ is an ellipse. If we write the eigendecomposition

$$\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top, \qquad \mathbf{\Lambda} = \mathrm{diag}(\lambda_1, \lambda_2), \quad \mathbf{Q} = [\mathbf{v}_1\ \mathbf{v}_2], \quad \mathbf{Q}^\top\mathbf{Q} = \mathbf{I},$$

then $\mathbf{Q}$ performs a rotation (or reflection) into the eigenvector basis, $\mathbf{\Lambda}$ scales along the coordinate axes in that basis, and $\mathbf{Q}^\top$ rotates back. In that picture (Fig. 1), the ellipse axes point along $\mathbf{v}_1, \mathbf{v}_2$, and the corresponding semi-axis length scales are $|\lambda_1|$ and $|\lambda_2|$ when starting from unit vectors. (If an eigenvalue is negative, the associated direction is also flipped, see Fig. 2; the magnitude still controls stretching.)

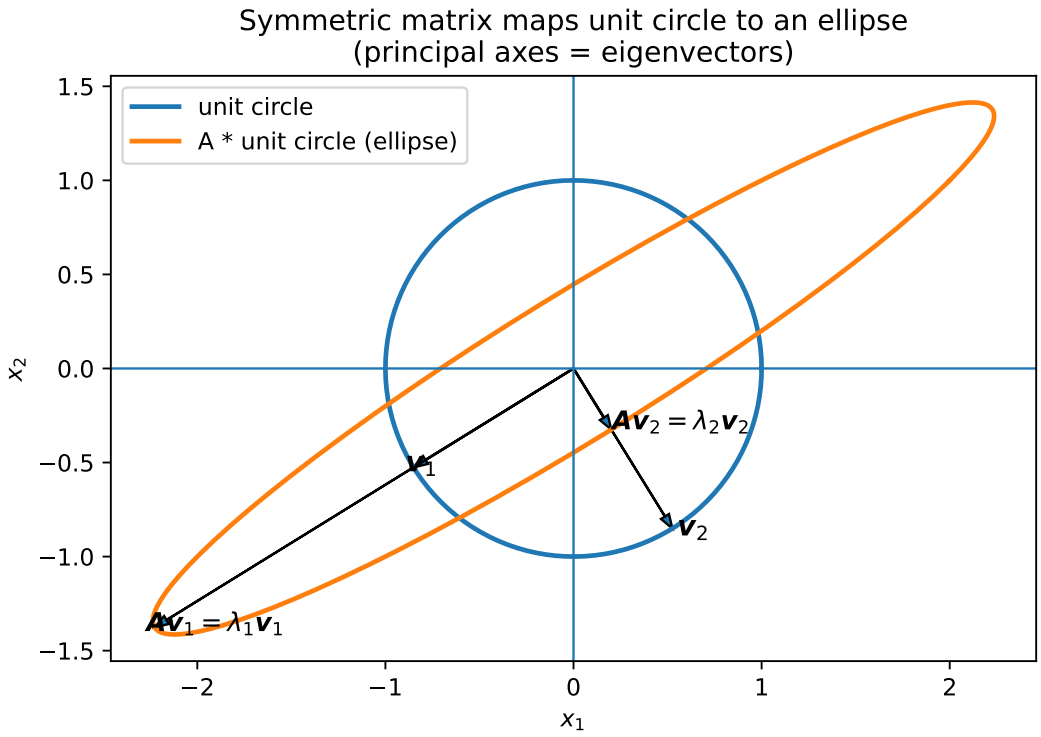

Figure 1: A symmetric linear transformation mapping the unit circle to an ellipse. Eigenvectors define orthogonal directions of pure scaling.

2.2 Rayleigh quotient: which direction "wins"?

A central question in dimensionality reduction is: *which direction captures the most signal?* For symmetric matrices, this is formalized by the Rayleigh quotient. For any nonzero $\mathbf{x}$ define

$$R(\mathbf{x}) = \frac{\mathbf{x}^\top\mathbf{A}\mathbf{x}}{\mathbf{x}^\top\mathbf{x}}.$$

If $\|\mathbf{x}\| = 1$, then $R(\mathbf{x}) = \mathbf{x}^\top\mathbf{A}\mathbf{x}$. In the symmetric case, the Rayleigh quotient has an extremal characterization:

$$\max_{\|\mathbf{x}\|=1} \mathbf{x}^\top\mathbf{A}\mathbf{x} = \lambda_{\max}, \quad \text{achieved at } \mathbf{x} = \mathbf{v}_{\max},$$

$$\min_{\|\mathbf{x}\|=1} \mathbf{x}^\top\mathbf{A}\mathbf{x} = \lambda_{\min}, \quad \text{achieved at } \mathbf{x} = \mathbf{v}_{\min}.$$

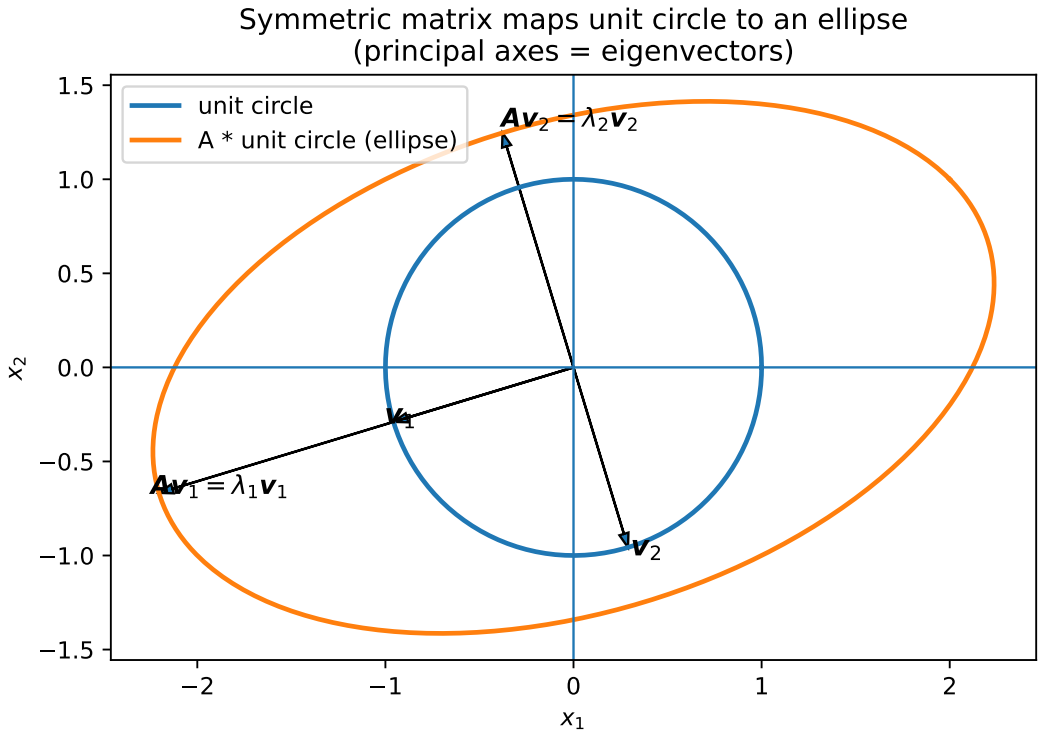

Figure 2: Effect of negative eigenvalues: scaling combined with a direction reversal.

Geometrically, this says that among all unit directions, the maximum value of the quadratic form $\mathbf{x}^\top \mathbf{A}\mathbf{x}$ occurs along the eigenvector associated with the largest eigenvalue. This is precisely the kind of statement we will later use in PCA, where $\mathbf{A}$ will be a covariance matrix: the direction of maximum variance will be the eigenvector of the largest eigenvalue.

2.3 What changes when the matrix is not symmetric?

The symmetric case is unusually well-behaved. If $\mathbf{A} \neq \mathbf{A}^\top$, then the transformation $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ can include effects such as shear in addition to stretching and rotation. Several important differences appear:

- Eigenvalues may be *complex*, meaning there may be no real invariant directions in $\mathbb{R}^2$.
- Even when real eigenvectors exist, they need not be orthogonal.
- The image of the unit circle under $\mathbf{A}$ is still an ellipse (for any real linear map), but the ellipse's principal axes are *not* generally given by eigenvectors.

So while eigen-decompositions are the right language for symmetric matrices (and, more generally, normal matrices), they do not always provide the cleanest geometric story for arbitrary linear transformations. For general matrices, the decomposition that always gives an orthogonal-axis geometric interpretation is the singular value decomposition.

2.4 Singular Value Decomposition (SVD): the universal geometric story

For any matrix

$$\mathbf{A} \in \mathbb{R}^{m \times n},$$

there exists a factorization

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\top},$$

where $\mathbf{U} \in \mathbb{R}^{m \times m}$ and $\mathbf{V} \in \mathbb{R}^{n \times n}$ are orthogonal matrices (their columns are orthonormal), and $\mathbf{\Sigma} \in \mathbb{R}^{m \times n}$ is diagonal in the sense that its only possibly nonzero entries lie on the main diagonal:

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq 0.$$

The numbers σ_k are the singular values of $\mathbf{A}$. The SVD describes the linear map $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ as a sequence of three simple transformations:

$$\mathbf{x} \xrightarrow{\mathbf{V}^{\top}} \text{rotate/reflect in the input space} \xrightarrow{\mathbf{\Sigma}} \text{axis-aligned scaling} \xrightarrow{\mathbf{U}} \text{rotate/reflect in the output space.}$$

This is exactly the clean "orthogonal axes + scaling" geometry that eigenvectors give in the symmetric case, but now it holds for *every* matrix.

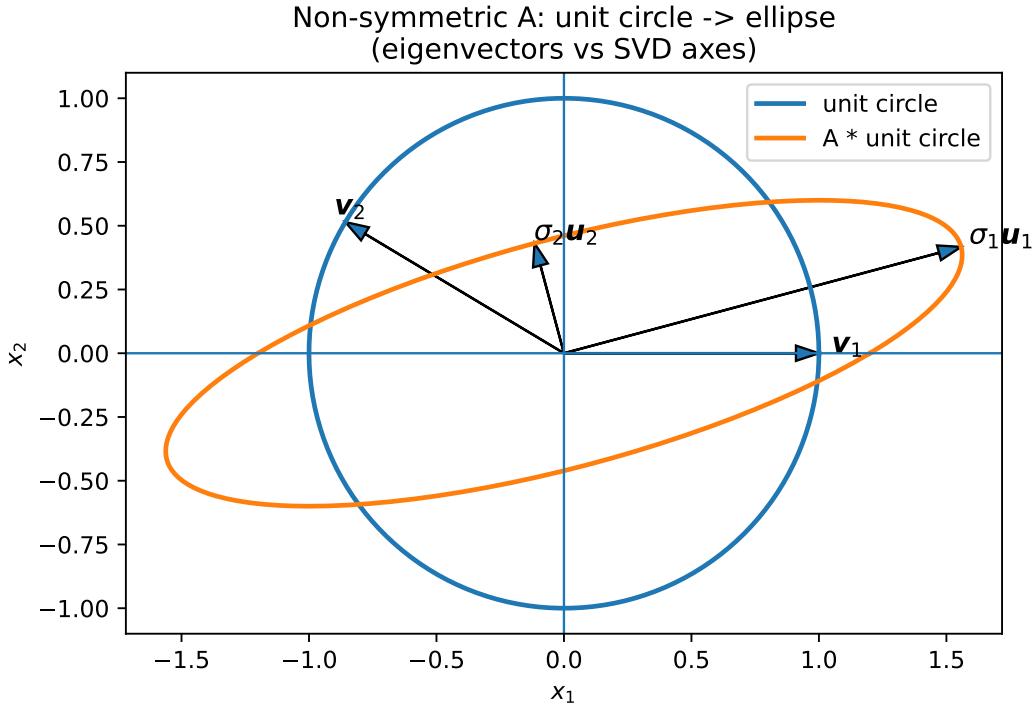

Figure 3: Geometric interpretation of the SVD: orthogonal input directions are scaled and mapped to orthogonal output directions.

A particularly intuitive picture arises in $\mathbb{R}^2$. Consider the unit circle $\{\mathbf{x} : \|\mathbf{x}\| = 1\}$ in the input space. Under the transformation $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$, its image is an ellipse. The SVD tells us precisely how to read that ellipse (Fig. 3):

- The columns of $\mathbf{V}$ are the **input directions** that map to the ellipse's axes.

- The columns of $\mathbf{U}$ are the **output directions** of those axes (where the ellipse axes point in the output space).
- The singular values σ_k are the **semi-axis lengths** of the ellipse.

Unlike eigenvectors, the SVD always exists, always yields orthogonal directions, and always provides a real geometric interpretation. This is why SVD sits at the foundation of many dimensionality reduction methods (including PCA): it provides a principled way to identify the dominant directions of action of a linear map and to approximate that map with lower rank by keeping only the largest singular values.

Interpreting the SVD one piece at a time

To understand the geometric meaning of the SVD, consider the transformation

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$$

acting on vectors $\mathbf{x} \in \mathbb{R}^n$. Rather than thinking of $\mathbf{A}$ as a single complicated operation, we interpret it as a sequence of three simple geometric steps.

Step 1: Input directions ($\mathbf{V}$) The matrix $\mathbf{V}$ has orthonormal columns

$$\mathbf{V} = [\mathbf{v}_1\ \mathbf{v}_2\ \cdots\ \mathbf{v}_n], \qquad \mathbf{V}^\top\mathbf{V} = \mathbf{I}.$$

Applying $\mathbf{V}^\top$ to a vector rotates (or reflects) the coordinate system without changing lengths or angles.

In particular, the vectors $\mathbf{v}_k$ define special *input directions*. If we choose $\mathbf{x} = \mathbf{v}_k$, then

$$\mathbf{V}^\top\mathbf{v}_k = \mathbf{e}_k,$$

the k-th standard basis vector. This means that $\mathbf{v}_k$ is mapped onto a coordinate axis before any scaling occurs.

Geometrically, these directions are precisely the directions in the input space that will become aligned with the principal axes of the output ellipse after the full transformation. This is why the columns of $\mathbf{V}$ are called the **input principal directions**.

Step 2: Axis-aligned scaling ($\mathbf{\Sigma}$) The diagonal matrix $\mathbf{\Sigma}$ has the form

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_1 & & \\ & \sigma_2 & \\ & & \ddots \end{bmatrix}, \qquad \sigma_1 \geq \sigma_2 \geq \cdots \geq 0.$$

It scales the k-th coordinate axis by the factor σ_k.

Thus, if a vector has already been aligned with a coordinate axis (as happens after applying $\mathbf{V}^\top$), $\mathbf{\Sigma}$ simply stretches or compresses it. Directions corresponding to large singular values are stretched strongly, while directions corresponding to small singular values are attenuated or possibly collapsed (if $\sigma_k = 0$).

This is the step where the *lengths of the ellipse axes* are determined.

Step 3: Output directions (U) Finally, the matrix $\mathbf{U}$, whose columns

$$\mathbf{U} = [\mathbf{u}_1 \ \mathbf{u}_2 \ \cdots]$$

are orthonormal, applies another rotation or reflection. This step does not change the lengths of vectors; it only changes their orientation in the output space.

If we start with $\mathbf{x} = \mathbf{v}_k$, then the full transformation gives

$$\mathbf{A}\mathbf{v}_k = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top \mathbf{v}_k = \mathbf{U}\mathbf{\Sigma}\mathbf{e}_k = \sigma_k \mathbf{u}_k.$$

This equation is the key geometric statement of the SVD. It shows that:

- the input direction $\mathbf{v}_k$ is mapped to the output direction $\mathbf{u}_k$,
- the length of the resulting vector is exactly σ_k.

Putting it all together: unit circle to ellipse Now consider the unit circle $\{\mathbf{x} : \|\mathbf{x}\| = 1\}$ in the input space. Every point on this circle can be written as a linear combination of the orthonormal basis vectors $\mathbf{v}_k$.

Under the transformation $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$:

- directions aligned with $\mathbf{v}_k$ are sent to directions aligned with $\mathbf{u}_k$,
- their lengths are scaled by σ_k.

As a result, the image of the unit circle is an ellipse whose principal axes:

- point along $\mathbf{u}_k$ in the output space,
- have semi-axis lengths σ_k.

Why this matters for dimensionality reduction The SVD orders directions by decreasing singular value. Keeping only the first d singular values and vectors corresponds to keeping the directions along which the transformation has the strongest effect. This provides a principled way to construct low-dimensional approximations and explains why SVD lies at the heart of PCA and other linear dimensionality reduction methods.

3 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is the canonical linear dimensionality reduction method. It formalizes the geometric ideas we have developed so far: projecting data onto a low-dimensional subspace that captures as much of the data's variability as possible.

3.1 PCA as a linear latent-variable model

Recall the general idea introduced earlier: although observations live in a high-dimensional space, they may be well-approximated by a low-dimensional representation,

$$\mathbf{x}_i \approx \boldsymbol{\mu} + g(\mathbf{z}_i), \qquad \mathbf{z}_i \in \mathbb{R}^d, \ \ d \ll D.$$

PCA corresponds to the special case where the generative mapping is *linear*. Specifically, PCA assumes that each data point can be approximated as

$$\boxed{\mathbf{x}_i \ \approx \ \boldsymbol{\mu} + \mathbf{W}_d \mathbf{z}_i,}$$

where:

- $\boldsymbol{\mu} \in \mathbb{R}^D$ is the data mean,
- $\mathbf{W}_d = [\mathbf{w}_1 \ \cdots \ \mathbf{w}_d] \in \mathbb{R}^{D \times d}$ has orthonormal columns (the principal directions),
- $\mathbf{z}_i \in \mathbb{R}^d$ are the low-dimensional latent coordinates (PCA scores).

The latent coordinates are obtained by orthogonal projection:

$$\mathbf{z}_i = \mathbf{W}_d^\top (\mathbf{x}_i - \boldsymbol{\mu}),$$

and reconstruction in the original space is given by

$$\hat{\mathbf{x}}_i = \boldsymbol{\mu} + \mathbf{W}_d \mathbf{z}_i.$$

3.2 Problem setup

We are given data points

$$\mathbf{x}_i \in \mathbb{R}^D, \qquad i = 1, \ldots, N.$$

The first step in PCA is to remove the mean:

$$\bar{\mathbf{x}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i, \qquad \tilde{\mathbf{x}}_i = \mathbf{x}_i - \bar{\mathbf{x}}.$$

Centering ensures that PCA focuses on *variation* rather than absolute position.

From the centered data we form the sample covariance matrix

$$\boldsymbol{\Sigma} = \frac{1}{N} \sum_{i=1}^{N} \tilde{\mathbf{x}}_i \tilde{\mathbf{x}}_i^\top \in \mathbb{R}^{D \times D}.$$

The covariance matrix encodes how the data vary jointly along different directions in space.

3.3 Variance-maximizing viewpoint

The first principal component is defined as the unit direction along which the projected data have maximum variance:

$$\mathbf{w}_1 = \arg\max_{\|\mathbf{w}\|=1} \mathrm{Var}(\mathbf{w}^\top \tilde{\mathbf{x}}) = \arg\max_{\|\mathbf{w}\|=1} \mathbf{w}^\top \boldsymbol{\Sigma} \mathbf{w}.$$

This is exactly a Rayleigh quotient maximization problem. From our earlier discussion, we know that its solution is the eigenvector of $\boldsymbol{\Sigma}$ corresponding to the largest eigenvalue.

Subsequent principal components are defined similarly, with the additional constraint that they be orthogonal to the previous ones:

$$\mathbf{w}_k = \arg\max_{\|\mathbf{w}\|=1,\ \mathbf{w} \perp \mathbf{w}_1, \ldots, \mathbf{w}_{k-1}} \mathbf{w}^\top \boldsymbol{\Sigma} \mathbf{w}.$$

The solution is given by the eigenvectors of $\boldsymbol{\Sigma}$, ordered by decreasing eigenvalue:

$$\boldsymbol{\Sigma} \mathbf{w}_k = \lambda_k \mathbf{w}_k, \qquad \lambda_1 \geq \lambda_2 \geq \cdots \geq 0.$$

3.4 Reconstruction (projection) viewpoint

An equivalent way to define PCA is through reconstruction error. Let

$$\mathbf{W}_d = [\mathbf{w}_1 \ \cdots \ \mathbf{w}_d] \in \mathbb{R}^{D \times d}$$

be a matrix with orthonormal columns. Projecting a centered data point onto this subspace gives

$$\hat{\mathbf{x}}_i = \mathbf{W}_d \mathbf{W}_d^\top \tilde{\mathbf{x}}_i.$$

PCA chooses $\mathbf{W}_d$ to minimize the total squared reconstruction error:

$$\mathbf{W}_d = \arg\min_{\mathbf{W}^\top \mathbf{W} = \mathbf{I}} \sum_{i=1}^{N} \left\| \tilde{\mathbf{x}}_i - \mathbf{W}\mathbf{W}^\top \tilde{\mathbf{x}}_i \right\|^2.$$

This optimization problem has the same solution: the columns of $\mathbf{W}_d$ are the top d eigenvectors of the covariance matrix. Thus, PCA can be seen either as *maximizing retained variance* or as *minimizing reconstruction error*.

The reduced (low-dimensional) coordinates are then

$$\mathbf{z}_i = \mathbf{W}_d^\top \tilde{\mathbf{x}}_i \in \mathbb{R}^d.$$

3.5 Explained variance

An important practical question in PCA is how many components to keep. This is quantified using the notion of *explained variance*.

Recall that the eigenvalues of the covariance matrix

$$\mathbf{\Sigma}\mathbf{w}_k = \lambda_k \mathbf{w}_k, \qquad \lambda_1 \geq \lambda_2 \geq \cdots \geq 0,$$

measure the variance of the data along each principal direction. The *total variance* in the data is

$$\mathrm{Var}_{\mathrm{total}} = \sum_{k=1}^{D} \lambda_k = \mathrm{tr}(\mathbf{\Sigma}).$$

If we keep only the first d principal components, the variance captured by the reduced representation is

$$\mathrm{Var}_{\mathrm{retained}}(d) = \sum_{k=1}^{d} \lambda_k.$$

The **explained variance ratio** is then defined as

$$\mathrm{EVR}(d) = \frac{\sum_{k=1}^{d} \lambda_k}{\sum_{k=1}^{D} \lambda_k}.$$

This quantity lies in $[0, 1]$ and measures the fraction of the data's variability preserved by the d-dimensional PCA subspace. In practice, one often chooses d such that $\mathrm{EVR}(d)$ exceeds a desired threshold (e.g. 90% or 95%).

Geometric interpretation Geometrically, explained variance measures how much of the data's "spread" is retained after orthogonal projection onto the PCA subspace. Directions with small eigenvalues correspond to thin directions of the data cloud; discarding them removes little variance but may remove noise.

Connection to reconstruction error Because PCA minimizes squared reconstruction error, explained variance is directly related to information loss. The average reconstruction error per data point when keeping d components is

$$\frac{1}{N}\sum_{i=1}^{N}\left\|\tilde{\mathbf{x}}_i - \mathbf{W}_d\mathbf{W}_d^\top\tilde{\mathbf{x}}_i\right\|^2 = \sum_{k=d+1}^{D}\lambda_k.$$

Thus, the variance *not* explained by PCA equals the sum of the discarded eigenvalues.

Key takeaway. Explained variance provides a principled criterion for selecting the dimensionality d and quantifies the trade-off between compression and information loss.

3.6 Worked example: PCA in $\mathbb{R}^2$

We illustrate PCA on a tiny 2D dataset (so every step can be done by hand). Consider the four points

$$\mathbf{x}_1 = \begin{bmatrix}2\\0\end{bmatrix}, \quad \mathbf{x}_2 = \begin{bmatrix}0\\2\end{bmatrix}, \quad \mathbf{x}_3 = \begin{bmatrix}3\\1\end{bmatrix}, \quad \mathbf{x}_4 = \begin{bmatrix}1\\3\end{bmatrix}.$$

We will compute the first principal component and the 1D projection/reconstruction.

Step 1: Compute the mean and center the data. The sample mean is

$$\bar{\mathbf{x}} = \frac{1}{4}\sum_{i=1}^{4}\mathbf{x}_i = \frac{1}{4}\begin{bmatrix}2+0+3+1\\0+2+1+3\end{bmatrix} = \begin{bmatrix}\frac{3}{2}\\\frac{3}{2}\end{bmatrix}.$$

The centered samples $\tilde{\mathbf{x}}_i = \mathbf{x}_i - \bar{\mathbf{x}}$ are

$$\tilde{\mathbf{x}}_1 = \begin{bmatrix}\frac{1}{2}\\-\frac{3}{2}\end{bmatrix}, \quad \tilde{\mathbf{x}}_2 = \begin{bmatrix}-\frac{3}{2}\\\frac{1}{2}\end{bmatrix}, \quad \tilde{\mathbf{x}}_3 = \begin{bmatrix}\frac{3}{2}\\-\frac{1}{2}\end{bmatrix}, \quad \tilde{\mathbf{x}}_4 = \begin{bmatrix}-\frac{1}{2}\\\frac{3}{2}\end{bmatrix}.$$

Step 2: Form the covariance matrix. Using

$$\mathbf{\Sigma} = \frac{1}{N}\sum_{i=1}^{N}\tilde{\mathbf{x}}_i\tilde{\mathbf{x}}_i^\top \quad \text{with } N = 4,$$

we compute the entries:

$$\Sigma_{11} = \frac{1}{4}\sum_{i=1}^{4}\tilde{x}_{i1}^2 = \frac{1}{4}\left(\frac{1}{4}+\frac{9}{4}+\frac{9}{4}+\frac{1}{4}\right) = \frac{5}{4},$$

$$\Sigma_{22} = \frac{1}{4}\sum_{i=1}^{4}\tilde{x}_{i2}^2 = \frac{1}{4}\left(\frac{9}{4}+\frac{1}{4}+\frac{1}{4}+\frac{9}{4}\right) = \frac{5}{4},$$

$$\Sigma_{12} = \Sigma_{21} = \frac{1}{4}\sum_{i=1}^{4}\tilde{x}_{i1}\tilde{x}_{i2} = \frac{1}{4}\left(-\frac{3}{4}-\frac{3}{4}-\frac{3}{4}-\frac{3}{4}\right) = -\frac{3}{4}.$$

Hence

$$\mathbf{\Sigma} = \begin{bmatrix}\frac{5}{4} & -\frac{3}{4}\\-\frac{3}{4} & \frac{5}{4}\end{bmatrix}.$$

Step 3: Eigenvalues/eigenvectors (principal directions). We solve $\boldsymbol{\Sigma}\mathbf{w} = \lambda\mathbf{w}$.

A convenient guess is to test the orthogonal directions

$$\mathbf{w}_1 = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\-1\end{bmatrix}, \qquad \mathbf{w}_2 = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix}.$$

Check $\mathbf{w}_1$:

$$\boldsymbol{\Sigma}\begin{bmatrix}1\\-1\end{bmatrix} = \begin{bmatrix}\frac{5}{4}\cdot 1 + \left(-\frac{3}{4}\right)(-1)\\ \left(-\frac{3}{4}\right)\cdot 1 + \frac{5}{4}(-1)\end{bmatrix} = \begin{bmatrix}2\\-2\end{bmatrix} = 2\begin{bmatrix}1\\-1\end{bmatrix}.$$

So $\mathbf{w}_1$ is an eigenvector with eigenvalue $\lambda_1 = 2$ (largest). Similarly,

$$\boldsymbol{\Sigma}\begin{bmatrix}1\\1\end{bmatrix} = \begin{bmatrix}\frac{5}{4} - \frac{3}{4}\\ -\frac{3}{4} + \frac{5}{4}\end{bmatrix} = \begin{bmatrix}\frac{1}{2}\\ \frac{1}{2}\end{bmatrix} = \frac{1}{2}\begin{bmatrix}1\\1\end{bmatrix},$$

so $\mathbf{w}_2$ has eigenvalue $\lambda_2 = \frac{1}{2}$.

Interpretation. The direction $\mathbf{w}_1$ is the *first principal component* (maximum variance), and the variance of the data along it equals $\lambda_1 = 2$. The orthogonal direction $\mathbf{w}_2$ captures the remaining (smaller) variance $\lambda_2 = \frac{1}{2}$.

Step 4: Project to 1D (scores). Keeping only the first component ($d = 1$), PCA coordinates are

$$z_i = \mathbf{w}_1^\top \tilde{\mathbf{x}}_i.$$

Compute:

$$z_1 = \frac{1}{\sqrt{2}}\left(\frac{1}{2} - (-\frac{3}{2})\right) = \frac{2}{\sqrt{2}} = \sqrt{2}, \quad z_2 = \frac{1}{\sqrt{2}}\left(-\frac{3}{2} - \frac{1}{2}\right) = -\frac{2}{\sqrt{2}} = -\sqrt{2},$$

$$z_3 = \frac{1}{\sqrt{2}}\left(\frac{3}{2} - (-\frac{1}{2})\right) = \sqrt{2}, \quad z_4 = \frac{1}{\sqrt{2}}\left(-\frac{1}{2} - \frac{3}{2}\right) = -\sqrt{2}.$$

So two points share $z = +\sqrt{2}$ and two points share $z = -\sqrt{2}$ (a typical "collapse" when going to 1D).

Step 5: Reconstruct from 1D. The 1D PCA reconstruction is

$$\hat{\mathbf{x}}_i = \bar{\mathbf{x}} + \mathbf{w}_1 z_i.$$

For $z = +\sqrt{2}$,

$$\hat{\mathbf{x}} = \begin{bmatrix}\frac{3}{2}\\ \frac{3}{2}\end{bmatrix} + \frac{1}{\sqrt{2}}\begin{bmatrix}1\\-1\end{bmatrix}\sqrt{2} = \begin{bmatrix}\frac{5}{2}\\ \frac{1}{2}\end{bmatrix},$$

and for $z = -\sqrt{2}$,

$$\hat{\mathbf{x}} = \begin{bmatrix}\frac{3}{2}\\ \frac{3}{2}\end{bmatrix} - \frac{1}{\sqrt{2}}\begin{bmatrix}1\\-1\end{bmatrix}\sqrt{2} = \begin{bmatrix}\frac{1}{2}\\ \frac{5}{2}\end{bmatrix}.$$

Thus the 1D model approximates all four points using only two reconstructed locations.

Explained variance. The total variance equals $\lambda_1 + \lambda_2 = 2 + \frac{1}{2} = \frac{5}{2}$, so the fraction explained by the first component is

$$\frac{\lambda_1}{\lambda_1 + \lambda_2} = \frac{2}{\frac{5}{2}} = 0.8.$$

So $d = 1$ retains 80% of the variance, even though it may still lose important geometric structure (depending on the task).

Summary. This worked example shows the full PCA pipeline:

$$\mathbf{x} \;\to\; \tilde{\mathbf{x}} = \mathbf{x} - \bar{\mathbf{x}} \;\to\; \boldsymbol{\Sigma} \;\to\; (\lambda_k, \mathbf{w}_k) \;\to\; z = \mathbf{w}_1^\top \tilde{\mathbf{x}} \;\to\; \hat{\mathbf{x}} = \bar{\mathbf{x}} + \mathbf{w}_1 z.$$

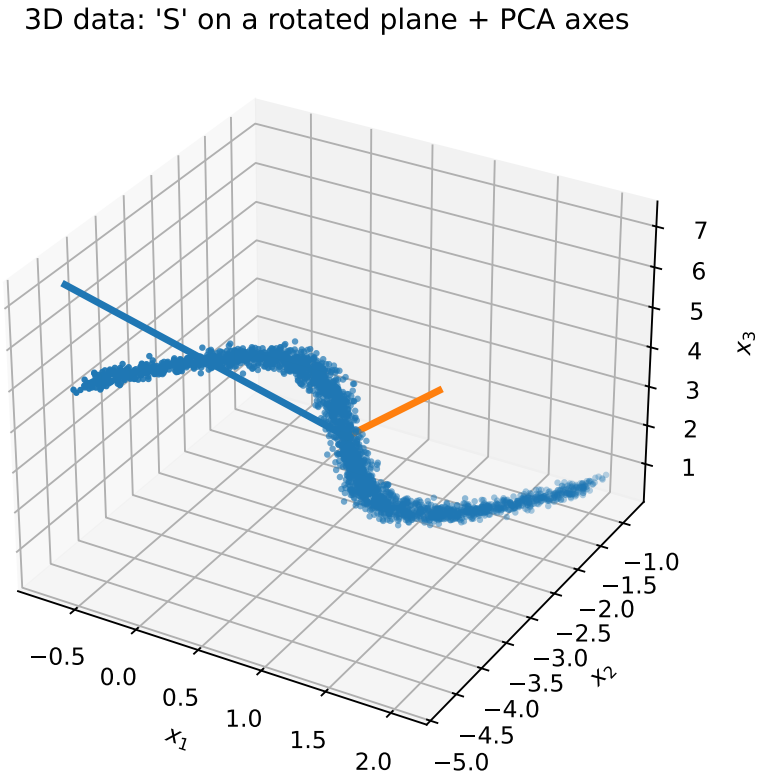

Figure 4: Three-dimensional data approximately lying on a plane, with additional nonlinear structure within the plane.

3.7 Example: 3D data lying near a plane

Consider data points in $\mathbb{R}^3$ that approximately lie on a plane (Fig. 4). Each point can be expressed as:

$$\mathbf{x}_i \approx \boldsymbol{\mu} + u_i\,\mathbf{p}_1 + v_i\,\mathbf{p}_2 + \varepsilon_i\,\mathbf{n},$$

where $\mathbf{p}_1$ and $\mathbf{p}_2$ span an unknown plane, $\mathbf{n}$ is its normal direction, and the noise ε_i is small.

Although the ambient dimension is 3, the intrinsic dimension of the data is close to 2. However, within the plane the data follow a nonlinear "S"-shaped curve. This immediately suggests that:

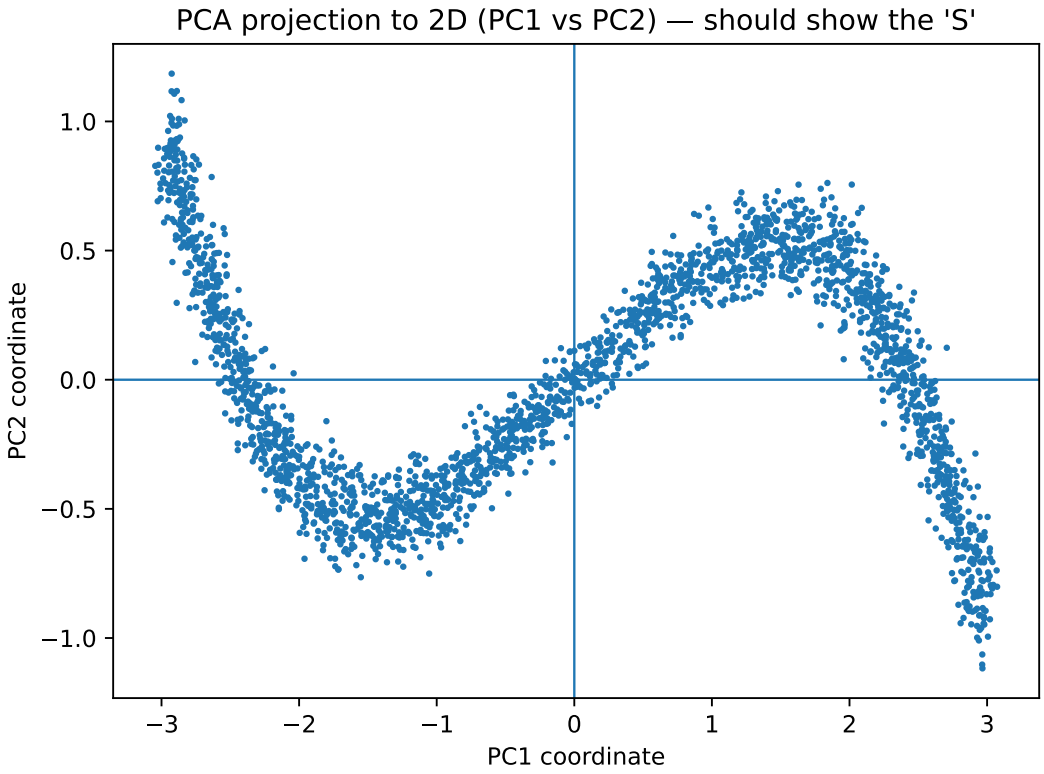

Figure 5: Projection onto the first two principal components. The intrinsic structure of the data is preserved.

- reducing from 3 dimensions to 2 should preserve most structure,
- reducing from 3 dimensions to 1 will generally destroy important information.

Why PCA discovers the plane

When we compute the covariance matrix of this dataset, its eigenvalues satisfy

$$\lambda_1 \text{ large}, \qquad \lambda_2 \text{ large}, \qquad \lambda_3 \text{ small}.$$

Geometrically, this means that:

- $\mathbf{w}_1$ and $\mathbf{w}_2$ span the plane along which the data vary most,
- $\mathbf{w}_3$ points approximately along the plane normal, where variance is minimal.

Projecting onto the first two principal components,

$$\mathbf{W}_2 = [\mathbf{w}_1\ \mathbf{w}_2] \in \mathbb{R}^{3\times 2}, \qquad \mathbf{z}_i = \mathbf{W}_2^\top \tilde{\mathbf{x}}_i,$$

produces a faithful 2D representation of the data.

Reconstruction and information loss

Given reduced coordinates $\mathbf{z}_i$, the PCA reconstruction in the original space is

$$\hat{\mathbf{x}}_i = \bar{\mathbf{x}} + \mathbf{W}_2 \mathbf{z}_i.$$

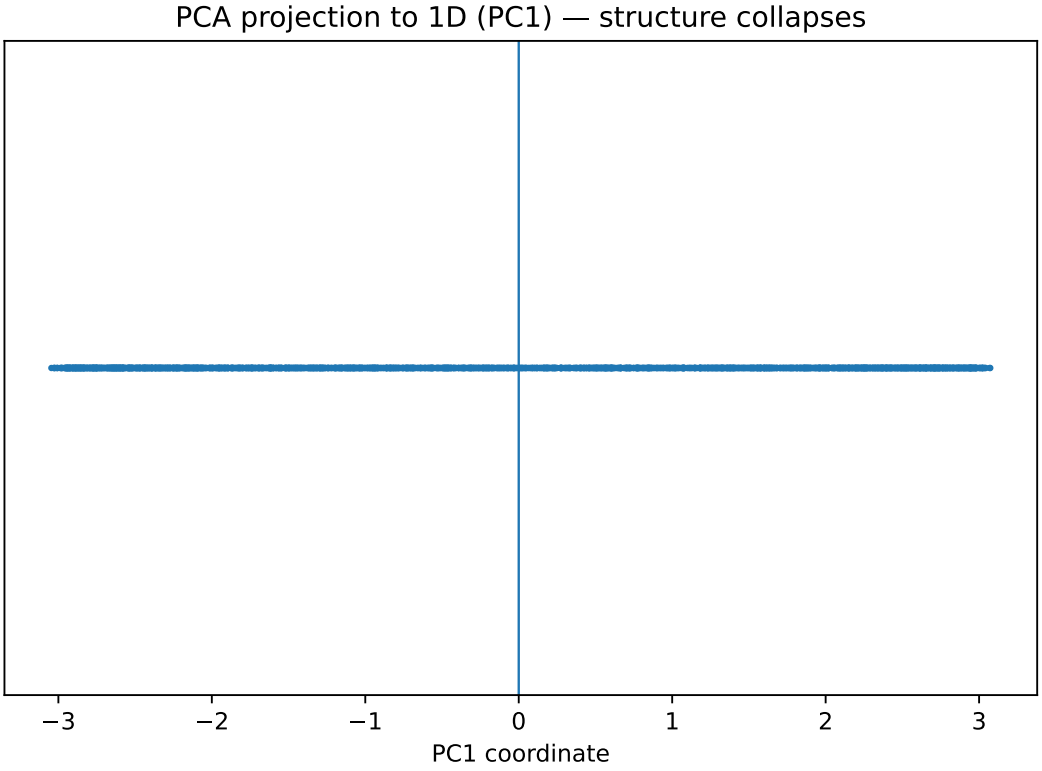

Figure 6: Projection onto a single principal component. Nonlinear structure collapses and becomes indistinguishable.

This is the orthogonal projection of $\mathbf{x}_i$ onto the PCA plane. The reconstruction error is

$$\|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2 = \|\tilde{\mathbf{x}}_i - \mathbf{W}_2\mathbf{W}_2^\top \tilde{\mathbf{x}}_i\|^2.$$

If we keep only one principal component, $\mathbf{W}_1 = \mathbf{w}_1$, the projection collapses the data onto a line. For nonlinear structures such as the "S" shape, distant parts of the curve may overlap after projection, resulting in severe loss of information.

3.8 PCA via SVD: same geometry, different viewpoint

An equivalent and often more practical way to compute PCA is via the singular value decomposition. Stack the centered data as rows of a matrix:

$$\tilde{\mathbf{X}} = \begin{bmatrix} \tilde{\mathbf{x}}_1^\top \\ \tilde{\mathbf{x}}_2^\top \\ \vdots \\ \tilde{\mathbf{x}}_N^\top \end{bmatrix} \in \mathbb{R}^{N \times D}.$$

Compute its SVD:

$$\tilde{\mathbf{X}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top.$$

Then the covariance matrix can be written as

$$\frac{1}{N}\tilde{\mathbf{X}}^\top\tilde{\mathbf{X}} = \mathbf{V}\left(\frac{\mathbf{\Sigma}^2}{N}\right)\mathbf{V}^\top.$$

This shows that:

- the principal components are the columns of $\mathbf{V}$,
- the PCA eigenvalues are $\lambda_k = \sigma_k^2/N$,
- the reduced coordinates (scores) satisfy

$$\tilde{\mathbf{X}}\mathbf{V}_d = \mathbf{U}_d\boldsymbol{\Sigma}_d \quad \Leftrightarrow \quad \mathbf{z}_i = \mathbf{V}_d^\top \tilde{\mathbf{x}}_i.$$

Key takeaway. PCA identifies orthogonal directions of maximal variance by projecting data onto the dominant singular directions of the centered data matrix. It is optimal among all linear methods for variance preservation and squared reconstruction error, but it cannot capture nonlinear structure.

4 SVD for Image Compression

4.1 Idea: an image is a matrix

A grayscale image with m rows and n columns can be represented as a matrix

$$\mathbf{X} \in \mathbb{R}^{m\times n},$$

where each entry X_{ij} is the intensity of pixel (i, j). If we normalize intensities to $[0, 1]$, then $X_{ij} = 0$ is black and $X_{ij} = 1$ is white.

The central idea of SVD compression is:

> Many natural images are *approximately low-rank*, meaning they can be well-approximated using only a small number of rank-1 "pattern" components.

4.2 SVD decomposition and geometric meaning

The singular value decomposition (SVD) of $\mathbf{X}$ is

$$\mathbf{X} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top,$$

where:

- $\mathbf{U} \in \mathbb{R}^{m\times r}$ has orthonormal columns (left singular vectors),
- $\mathbf{V} \in \mathbb{R}^{n\times r}$ has orthonormal columns (right singular vectors),
- $\boldsymbol{\Sigma} \in \mathbb{R}^{r\times r}$ is diagonal with entries

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0,$$

and $r = \operatorname{rank}(\mathbf{X})$ (or $r = \min(m, n)$ in the "economy" SVD representation).

A very useful way to read the SVD is via its rank-1 expansion:

$$\mathbf{X} = \sum_{k=1}^{r} \sigma_k\, \mathbf{u}_k\mathbf{v}_k^\top,$$

where $\mathbf{u}_k$ is the k-th column of $\mathbf{U}$ and $\mathbf{v}_k$ is the k-th column of $\mathbf{V}$. Each term $\sigma_k\, \mathbf{u}_k\mathbf{v}_k^\top$ is a rank-1 image-like pattern (an outer product), scaled by σ_k.

Intuitively:

- $\mathbf{u}_k$ captures a vertical/spatial pattern across rows,
- $\mathbf{v}_k$ captures a horizontal/spatial pattern across columns,
- σ_k says how important that pattern is.

4.3 Compression: keeping only the top k terms

To compress the image, we keep only the first k singular values/vectors:

$$\mathbf{X}_k = \sum_{j=1}^{k} \sigma_j \, \mathbf{u}_j \mathbf{v}_j^\top.$$

Equivalently, in matrix form:

$$\mathbf{X}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^\top,$$

where $\mathbf{U}_k \in \mathbb{R}^{m \times k}$ contains the first k columns of $\mathbf{U}$, $\mathbf{\Sigma}_k \in \mathbb{R}^{k \times k}$ is the top-left diagonal block, and $\mathbf{V}_k \in \mathbb{R}^{n \times k}$ contains the first k columns of $\mathbf{V}$.

Why this is optimal (Eckart–Young theorem). Among all rank-k matrices, $\mathbf{X}_k$ is the best approximation to $\mathbf{X}$ in Frobenius norm:

$$\mathbf{X}_k = \arg \min_{\text{rank}(\mathbf{Y}) \le k} \|\mathbf{X} - \mathbf{Y}\|_F.$$

This is the fundamental theoretical reason SVD is a principled compression method: it gives the *best* rank-k approximation.

4.4 Error and "energy" captured

A convenient quantity is the squared Frobenius norm:

$$\|\mathbf{X}\|_F^2 = \sum_{i,j} X_{ij}^2.$$

Using the SVD, one can show

$$\|\mathbf{X}\|_F^2 = \sum_{j=1}^{r} \sigma_j^2.$$

Therefore, the fraction of "energy" captured by keeping the top k singular values is

$$E(k) = \frac{\sum_{j=1}^{k} \sigma_j^2}{\sum_{j=1}^{r} \sigma_j^2}.$$

This matches the intuition: if singular values decay quickly, then a small k captures most of the image's content.

The reconstruction error has a clean expression:

$$\|\mathbf{X} - \mathbf{X}_k\|_F^2 = \sum_{j=k+1}^{r} \sigma_j^2.$$

So dropping small singular values discards little energy and typically removes fine detail/noise.

4.5 How many numbers do we store? (compression ratio)

The full image stores mn numbers (pixel intensities). A rank-k SVD representation stores:

$$\underbrace{mk}_{\mathbf{U}_k} + \underbrace{k}_{\sigma_1,\ldots,\sigma_k} + \underbrace{nk}_{\mathbf{V}_k} = k(m+n+1) \quad \text{numbers.}$$

A simple *compression ratio* estimate is:

$$\mathrm{CR}(k) = \frac{mn}{k(m+n+1)}.$$

If $\mathrm{CR}(k) > 1$, we are storing fewer numbers than the raw image. (In practice, real file formats also involve quantization and entropy coding; the above is a conceptual numerical-storage comparison.)

4.6 What the demo figure shows

Figure 7 shows:

- the original grayscale image matrix $\mathbf{X}$,
- several rank-k reconstructions $\mathbf{X}_k$ (for increasing k),
- the cumulative energy curve $E(k)$,
- the singular value spectrum (often on a log scale).

As k increases, the reconstruction becomes sharper, because more rank-1 components are included. Typically, the first few components capture coarse structure (large-scale shading/edges), while later components capture fine texture.

4.7 Code walkthrough (grayscale SVD compression demo)

The following Python script implements the steps above. Here is what each part does and how it maps to the theory.

1) Imports and output directory.

- `numpy` for linear algebra and arrays,
- `matplotlib` for image display and plots.

2) Loading a grayscale image as a matrix X. The function `load_grayscale(path)` reads an image (PNG/JPG). If the image is RGB (3 channels) or RGBA (4 channels), it converts to grayscale using a standard luminance combination:

$$Y = 0.2989\,R + 0.5870\,G + 0.1140\,B.$$

Then it normalizes values to $[0, 1]$ so that pixel intensities match our mathematical model.

If `IMG_PATH=None`, the script instead generates a `synthetic_image()` that has smooth variations and simple shapes. This is useful because it is guaranteed to run without external files.

At the end of this stage, the code has:

$$\mathbf{X} \in \mathbb{R}^{m\times n}, \quad \text{with} \quad m, n = \texttt{X.shape}.$$

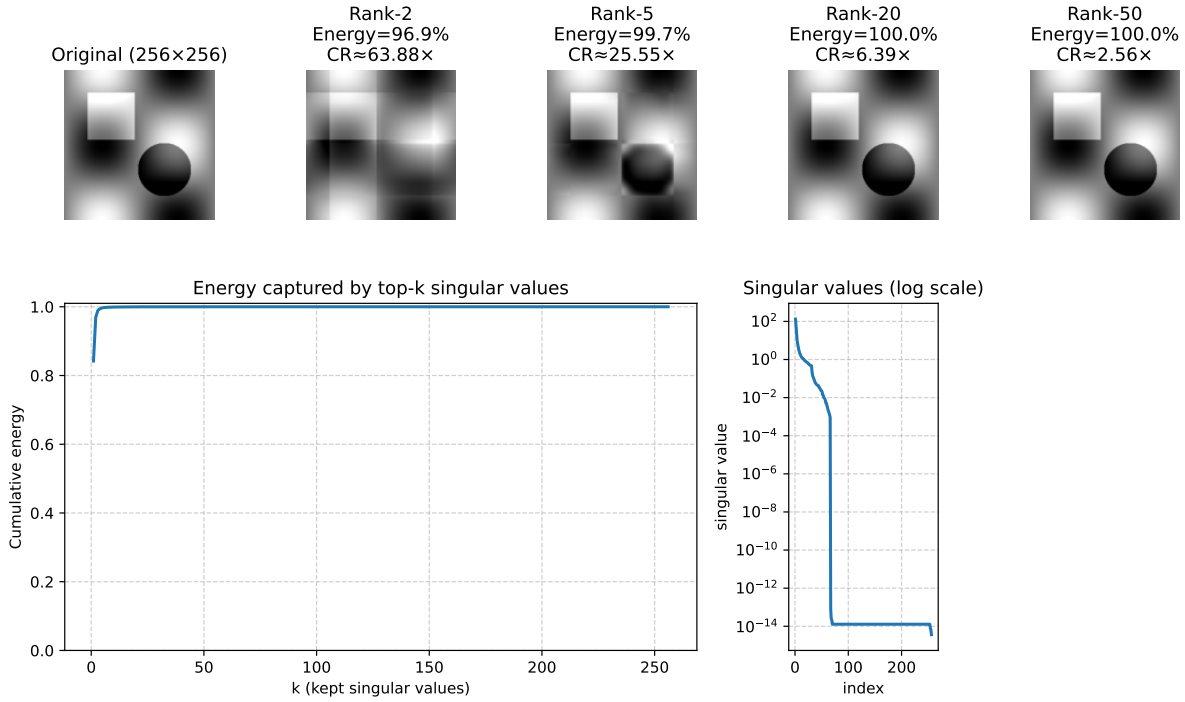

Figure 7: SVD image compression demo: original image, rank-k reconstructions, cumulative energy captured, and singular value spectrum.

3) Computing the SVD. The line

```
U, s, VT = np.linalg.svd(X, full_matrices=False)
```

computes the economy SVD:

$$\mathbf{X} = \mathbf{U}\,\mathrm{diag}(\mathbf{s})\,\mathbf{V}^\top.$$

In the code:

- `U` corresponds to $\mathbf{U}$,
- `s` is the vector $(\sigma_1, \dots)$ of singular values,
- `VT` corresponds to $\mathbf{V}^\top$.

The line `S = np.diag(s)` explicitly forms the diagonal matrix $\mathbf{\Sigma}$.

4) Building rank-k approximations. For chosen ranks `ks = [2, 5, 20, 50]`, the script forms:

$$\mathbf{U}_k = \mathbf{U}(:, 1{:}k), \quad \mathbf{\Sigma}_k = \mathbf{\Sigma}(1{:}k, 1{:}k), \quad \mathbf{V}_k^\top = \mathbf{V}^\top(1{:}k, :).$$

and reconstructs

$$\mathbf{X}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^\top.$$

This is exactly the best rank-k approximation guaranteed by Eckart–Young.

Finally, `np.clip(Xk, 0, 1)` keeps pixel values in the display range $[0, 1]$ (since truncation can create small overshoots).

5) Energy captured curve. The script computes:

$$\texttt{sing2 = s**2} \quad \Rightarrow \quad \sigma_j^2,$$

$$\texttt{cum\_energy = cumsum(sing2)/sing2.sum()} \quad \Rightarrow \quad E(k) = \frac{\sum_{j \le k} \sigma_j^2}{\sum_j \sigma_j^2}.$$

This is the same cumulative energy/explained-variance-like curve used in PCA.

6) Compression ratio estimate. The helper

$$\texttt{stored = k*(m + n + 1)}$$

implements the theoretical storage count $k(m+n+1)$ (store $\mathbf{U}_k$, $\mathbf{V}_k$, and $\sigma_1, \ldots, \sigma_k$). Then

$$\texttt{full = m*n}, \qquad \texttt{return full / stored}$$

computes

$$\mathrm{CR}(k) = \frac{mn}{k(m+n+1)}.$$

The title of each rank-k panel displays the energy $E(k)$ and the approximate compression ratio.

7) Visualization. The figure has:

- top row: original image and rank-k reconstructions,
- bottom left: energy curve $E(k)$ versus k,
- bottom right: singular values σ_k on a log scale (to show decay).

4.8 Practical notes and extensions

Choosing k. Common heuristics:

- pick the smallest k such that $E(k) \ge 0.90$ or 0.95,
- look for an "elbow" in the energy curve,
- pick k based on a desired compression ratio.

Color images. For RGB images, a simple approach is to apply SVD separately to each channel (R,G,B). More advanced approaches compress in a transformed color space (e.g., YCbCr) or use tensor methods.

Relation to PCA. If you stack centered image samples into a data matrix, PCA can be computed via SVD. In this compression demo we apply SVD to a *single* image matrix to obtain a low-rank approximation.

Limitations. SVD is optimal for squared error ($\|\cdot\|_F$), but human visual perception is not perfectly aligned with MSE. Also, modern codecs (JPEG/WebP/AVIF) use more specialized transforms and entropy coding, often achieving better compression for the same perceptual quality.

4.9 Python Implementation

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
SVD image compression demo (grayscale)
"""

import os
import numpy as np
import matplotlib.pyplot as plt

# ---- Load image (grayscale) ----
# Set IMG_PATH to a PNG/JPG if you have one; else a synthetic image is created.
IMG_PATH = None  # e.g., "lena_gray.png"

def load_grayscale(path):
    img = plt.imread(path)
    if img.ndim == 3:
        # Convert RGB/RGBA to grayscale via luminance
        img = img[..., :3]  # drop alpha if present
        coeffs = np.array([0.2989, 0.5870, 0.1140])
        img = (img * coeffs).sum(axis=2)
    # Normalize to [0,1]
    img = img.astype(np.float64)
    if img.max() > 1.0:
        img = img / 255.0
    return img

def synthetic_image(h=256, w=256):
    # Smooth gradient + shapes
    yy, xx = np.mgrid[0:h, 0:w]
    base = (0.5 + 0.5 * np.sin(2*np.pi*xx/w) * np.cos(2*np.pi*yy/h))
    # Add a bright square and a dark disk
    base[40:120, 40:120] += 0.4
    center = np.array([170, 170])
    rr = np.sqrt((yy - center[0])**2 + (xx - center[1])**2)
    base[rr < 45] -= 0.5
    return np.clip(base, 0, 1)

X = load_grayscale(IMG_PATH) if IMG_PATH else synthetic_image()

m, n = X.shape

# ---- SVD ----
U, s, VT = np.linalg.svd(X, full_matrices=False)  # X = U @ np.diag(s) @ VT
S = np.diag(s)

# ---- Reconstructions ----
ks = [2, 5, 20, 50]
recons = {}
for k in ks:
    Uk = U[:, :k]
    Sk = S[:k, :k]
    Vk = VT[:k, :]
    Xk = Uk @ Sk @ Vk
```

```
    recons[k] = np.clip(Xk, 0, 1)

# ---- Energy curve ----
sing2 = s**2
cum_energy = np.cumsum(sing2) / sing2.sum()

# ---- Compression ratios ----
def compression_ratio(k, m, n):
    # store U(:,1:k) + s(1:k) + V(:,1:k)
    stored = k*(m + n + 1)
    full = m*n
    return full / stored

ratios = {k: compression_ratio(k, m, n) for k in ks}

# ---- Plot layout ----
fig = plt.figure(figsize=(11, 7))

# Original
ax1 = plt.subplot2grid((2, len(ks) + 1), (0,0))
ax1.imshow(X, cmap='gray', vmin=0, vmax=1)
ax1.set_title(f"Original ({m}×{n})")
ax1.axis('off')

# Rank-k panels
for j, k in enumerate(ks, start=1):
    ax = plt.subplot2grid((2, len(ks) + 1), (0, j))
    ax.imshow(recons[k], cmap='gray', vmin=0, vmax=1)
    ax.set_title(f"Rank-{k}\nEnergy={cum_energy[k-1]*100:.1f}%\nCR≈{
        ratios[k]:.2f}×")
    ax.axis('off')

# Energy curve
axE = plt.subplot2grid((2, len(ks) + 1), (1,0), colspan=3)
axE.plot(np.arange(1, len(s)+1), cum_energy, lw=2)
axE.set_xlabel("k (kept singular values)")
axE.set_ylabel("Cumulative energy")
axE.set_ylim(0, 1.01)
axE.grid(True, linestyle='--', alpha=0.6)
axE.set_title("Energy captured by top-k singular values")

# Spectrum (singular values)
axS = plt.subplot2grid((2, len(ks) + 1), (1, len(ks) - 1))
axS.semilogy(np.arange(1, len(s)+1), s, lw=2)
axS.set_xlabel("index")
axS.set_ylabel("singular value")
axS.grid(True, linestyle='--', alpha=0.6)
axS.set_title("Singular values (log scale)")

plt.tight_layout()
plt.show()
```

Listing 1: SVD for Image Compression

4.10 Why treat an image as a matrix and apply SVD?

A grayscale image is naturally a function on a 2D grid:

$$I : \{1, \dots, m\} \times \{1, \dots, n\} \to \mathbb{R}, \qquad (i, j) \mapsto I(i, j),$$

where i indexes the row (height) and j indexes the column (width). Writing these samples in a table produces a matrix

$$\mathbf{X} \in \mathbb{R}^{m \times n}, \qquad X_{ij} = I(i, j).$$

This is not an arbitrary choice: it preserves the fact that each pixel intensity depends jointly on a *row coordinate* and a *column coordinate*. Many visual patterns in images (smooth shading, edges, repeated textures) create strong correlations across rows and columns, and this often makes $\mathbf{X}$ *approximately low-rank*.

What "low-rank" means for images. Recall the rank-1 outer product form:

$$\mathbf{X} \approx \sum_{k=1}^{K} \sigma_k \, \mathbf{u}_k \mathbf{v}_k^\top.$$

Each term $\mathbf{u}_k \mathbf{v}_k^\top$ is a separable 2D pattern: it is the product of a 1D vertical profile $\mathbf{u}_k$ (over rows) and a 1D horizontal profile $\mathbf{v}_k$ (over columns). Keeping only a few such terms says:

> The image can be expressed using a small number of "row patterns" and "column patterns" whose combinations explain most pixel intensities.

This is a strong inductive bias that is often reasonable for natural images, especially for smooth regions and simple geometric shapes.

Why SVD is the right tool. SVD is the correct tool for this viewpoint because it provides the *best* rank-K approximation in squared error:

$$\mathbf{X}_K = \arg \min_{\text{rank}(\mathbf{Y}) \le K} \|\mathbf{X} - \mathbf{Y}\|_F.$$

So if you want to compress an image by restricting it to rank K (i.e., to a model with K separable components), SVD gives the optimal answer.

4.10.1 Is this "PCA in 2 dimensions" (width/height)?

Yes and no — and the distinction is worth stating clearly.

Yes, in a geometric sense. SVD on the image matrix finds:

- orthonormal directions in the *row space* (height patterns) via $\mathbf{U}$,
- orthonormal directions in the *column space* (width patterns) via $\mathbf{V}$,
- and strengths (importance) via singular values σ_k.

This is analogous to PCA because PCA also finds orthonormal directions ordered by importance (variance/energy). In fact, the SVD is a more fundamental object, and PCA can be computed via SVD.

No, not in the "each pixel is a 2D point" sense. It is *not* meaningful to interpret each pixel value X_{ij} as a data point in $\mathbb{R}^2$ with coordinates (i, j) and then run PCA on those coordinates. PCA needs a vector-valued observation per sample. Here (i, j) are just spatial indices, while X_{ij} is a scalar intensity. Doing PCA on the coordinates (i, j) would only tell you about the rectangular grid (which is trivial), not about the image content.

What SVD on X is really doing. SVD is doing a *2D factorization*:

$$X_{ij} \approx \sum_{k=1}^{K} \sigma_k \, u_k(i) \, v_k(j),$$

which decomposes the intensity field into a sum of K separable components. This is sometimes called a *separable representation* and can be viewed as a low-dimensional model jointly over the two spatial dimensions.

4.10.2 How this relates to PCA more formally

If we form the matrices

$$\mathbf{X}\mathbf{X}^\top \in \mathbb{R}^{m \times m}, \qquad \mathbf{X}^\top\mathbf{X} \in \mathbb{R}^{n \times n},$$

then the SVD implies:

$$\mathbf{X}\mathbf{X}^\top = \mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^\top, \qquad \mathbf{X}^\top\mathbf{X} = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^\top.$$

So:

- columns of $\mathbf{U}$ are eigenvectors of $\mathbf{X}\mathbf{X}^\top$ (patterns across rows),
- columns of $\mathbf{V}$ are eigenvectors of $\mathbf{X}^\top\mathbf{X}$ (patterns across columns),
- eigenvalues are σ_k^2.

This looks exactly like PCA/eigendecomposition, but now it is applied to the row/column correlation structure of a *single* image matrix.

4.10.3 When this works well (and when it doesn't)

Works well when:

- the image has large smooth regions, gradual illumination changes,
- simple geometric structure dominates,
- noise is present in fine-scale details (small σ_k often capture high-frequency components).

Works less well when:

- the image contains complex textures everywhere,
- fine details are perceptually critical,
- you care about compression as an actual file format (JPEG/WebP/AVIF usually outperform plain SVD because they also quantize and entropy-code coefficients).

Key takeaway. Doing SVD on the image matrix is a principled way to build the best rank-K approximation, because it exploits correlations across the two spatial dimensions directly. It is "PCA-like" in that it finds orthogonal components ordered by importance, but it is not PCA on the spatial coordinates; it is a low-rank factorization of the 2D intensity field.