1 Singular Value Decomposition (SVD)

The singular value decomposition (SVD) is among the most important matrix factorizations of the computational era, providing a foundation for nearly all of the data methods in this book. The SVD provides a numerically stable matrix decomposition that can be used for a variety of purposes and is guaranteed to exist. We will use the SVD to obtain low-rank approximations to matrices and to perform pseudo-inverses of non-square matrices to find the solution of a system of equations $\mathbf{Ax} = \mathbf{b}$. Another important use of the SVD is as the underlying algorithm of principal component analysis (PCA), where high-dimensional data is decomposed into its most statistically descriptive factors. SVD/PCA has been applied to a wide variety of problems in science and engineering.

In a sense, the SVD generalizes the concept of the fast Fourier transform (FFT), which will be the subject of the next chapter. Many engineering texts begin with the FFT, as it is the basis of many classical analytical and numerical results. However, the FFT works in idealized settings, and the SVD is a more generic data-driven technique. Because this book is focused on data, we begin with the SVD, which may be thought of as providing a basis that is *tailored* to the specific data, as opposed to the FFT, which provides a *generic* basis.

In many domains, complex systems will generate data that is naturally arranged in large matrices, or more generally in arrays. For example, a time-series of data from an experiment or a simulation may be arranged in a matrix with each column containing all of the measurements at a given time. If the data at each instant in time is multi-dimensional, as in a high-resolution simulation of the weather in three spatial dimensions, it is possible to reshape or *flatten* this data into a high-dimensional column vector, forming the columns of a large matrix. Similarly, the pixel values in a grayscale image may be stored in a matrix, or these images may be reshaped into large column vectors in a matrix to represent the frames of a movie. Remarkably, the data generated by these systems are typically low rank, meaning that there are a few dominant patterns that explain the high-dimensional data. The SVD is a numerically robust and efficient method of extracting these patterns from data.

1.1 Overview

Here we introduce the SVD and develop an intuition for how to apply the SVD by demonstrating its use on a number of motivating examples. The SVD will provide a foundation for many other techniques developed in this book, including classification methods in Chapter 5, the dynamic mode decomposition (DMD) in Chapter 7, and the proper orthogonal decomposition (POD) in Chapter 11. Detailed mathematical properties are discussed in the following sections.

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

High dimensionality is a common challenge in processing data from complex systems. These systems may involve large measured data sets including audio, image, or video data. The data may also be generated from a physical system, such as neural recordings from a brain, or fluid velocity measurements from a simulation or experiment. In many naturally occurring systems, it is observed that data exhibit dominant patterns, which may be characterized by a low-dimensional attractor or manifold [252, 251].

As an example, consider images, which typically contain a large number of measurements (pixels), and are therefore elements of a high-dimensional vector space. However, most images are highly compressible, meaning that the relevant information may be represented in a much lower-dimensional subspace. The compressibility of images will be discussed in depth throughout this book. Complex fluid systems, such as the Earth's atmosphere or the turbulent wake behind a vehicle also provide compelling examples of the low-dimensional structure underlying a high-dimensional state-space. Although high-fidelity fluid simulations typically require at least millions or billions of degrees of freedom, there are often dominant coherent structures in the flow, such as periodic vortex shedding behind vehicles or hurricanes in the weather.

The SVD provides a systematic way to determine a low-dimensional approximation to high-dimensional data in terms of dominant patterns. This technique is *data-driven* in that patterns are discovered purely from data, without the addition of expert knowledge or intuition. The SVD is numerically stable and provides a hierarchical representation of the data in terms of a new coordinate system defined by dominant correlations within the data. Moreover, the SVD is guaranteed to exist for any matrix, unlike the eigendecomposition.

The SVD has many powerful applications beyond dimensionality reduction of high-dimensional data. It is used to compute the pseudo-inverse of non-square matrices, providing solutions to underdetermined or overdetermined matrix equations, $\mathbf{A}\mathbf{x} = \mathbf{b}$. We will also use the SVD to de-noise data sets. The SVD is likewise important to characterize the input and output geometry of a linear map between vector spaces. These applications will all be explored in this chapter, providing an intuition for matrices and high-dimensional data.

Definition of the SVD

Generally, we are interested in analyzing a large data set $\mathbf{X} \in \mathbb{C}^{n \times m}$:

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_m \\ | & | & & | \end{bmatrix}. \tag{1.1}$$

The columns $\mathbf{x}_k \in \mathbb{C}^n$ may be measurements from simulations or experiments. For example, columns may represent images that have been reshaped into column vectors with as many elements as pixels in the image. The column vectors may also represent the state of a physical system that is evolving in time, such as the fluid velocity at a set of discrete points, a set of neural measurements, or the state of a weather simulation with one square kilometer resolution.

The index k is a label indicating the k^{th} distinct set of measurements. For many of the examples in this book, $\mathbf{X}$ will consist of a *time-series* of data, and $\mathbf{x}_k = \mathbf{x}(k\Delta t)$. Often the *state-dimension* n is very large, on the order of millions or billions of degrees of freedom.

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

The columns are often called *snapshots*, and m is the number of snapshots in $\mathbf{X}$. For many systems $n \gg m$, resulting in a *tall-skinny* matrix, as opposed to a *short-fat* matrix when $n \ll m$.

The SVD is a unique matrix decomposition that exists for every complex-valued matrix $\mathbf{X} \in \mathbb{C}^{n \times m}$:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \tag{1.2}$$

where $\mathbf{U} \in \mathbb{C}^{n \times n}$ and $\mathbf{V} \in \mathbb{C}^{m \times m}$ are *unitary* matrices[1] with orthonormal columns, and $\mathbf{\Sigma} \in \mathbb{R}^{n \times m}$ is a matrix with real, nonnegative entries on the diagonal and zeros off the diagonal. Here * denotes the complex conjugate transpose[2]. As we will discover throughout this chapter, the condition that $\mathbf{U}$ and $\mathbf{V}$ are unitary is used extensively.

When $n \geq m$, the matrix $\mathbf{\Sigma}$ has at most m nonzero elements on the diagonal, and may be written as $\mathbf{\Sigma} = \begin{bmatrix} \hat{\mathbf{\Sigma}} \\ \mathbf{0} \end{bmatrix}$. Therefore, it is possible to *exactly* represent $\mathbf{X}$ using the *economy* SVD:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* = \begin{bmatrix} \hat{\mathbf{U}} & \hat{\mathbf{U}}^{\perp} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{\Sigma}} \\ \mathbf{0} \end{bmatrix} \mathbf{V}^* = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\mathbf{V}^*. \tag{1.3}$$

The full SVD and economy SVD are shown in Fig. 1.1. The columns of $\hat{\mathbf{U}}^{\perp}$ span a vector space that is complementary and orthogonal to that spanned by $\hat{\mathbf{U}}$. The columns of $\mathbf{U}$ are called *left singular vectors* of $\mathbf{X}$ and the columns of $\mathbf{V}$ are *right singular vectors*. The diagonal elements of $\hat{\mathbf{\Sigma}} \in \mathbb{C}^{m \times m}$ are called *singular values* and they are ordered from largest to smallest. The rank of $\mathbf{X}$ is equal to the number of nonzero singular values.

Computing the SVD

The SVD is a cornerstone of computational science and engineering, and the numerical implementation of the SVD is both important and mathematically enlightening. That said, most standard numerical implementations are mature and a simple interface exists in many modern computer languages, allowing us to abstract away the details underlying the SVD computation. For most purposes, we simply use the SVD as a part of a larger effort, and we take for granted the existence of efficient and stable numerical algorithms. In the sections that follow we demonstrate how to use the SVD in various computational languages, and we also discuss the most common computational strategies and limitations. There are numerous important results on the computation of the SVD [212, 106, 211, 292, 238]. A more thorough discussion of computational issues can be found in [214]. Randomized numerical algorithms are increasingly used to compute the SVD of very large matrices as discussed in Section 1.8.

Matlab. In Matlab, computing the SVD is straightforward:

```
>>X = randn(5,3);    % Create a 5x3 random data matrix
>>[U,S,V] = svd(X); % Singular Value Decomposition
```

[1] A square matrix $\mathbf{U}$ is unitary if $\mathbf{U}\mathbf{U}^* = \mathbf{U}^*\mathbf{U} = \mathbf{I}$.

[2] For real-valued matrices, this is the same as the regular transpose $\mathbf{X}^* = \mathbf{X}^T$.

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

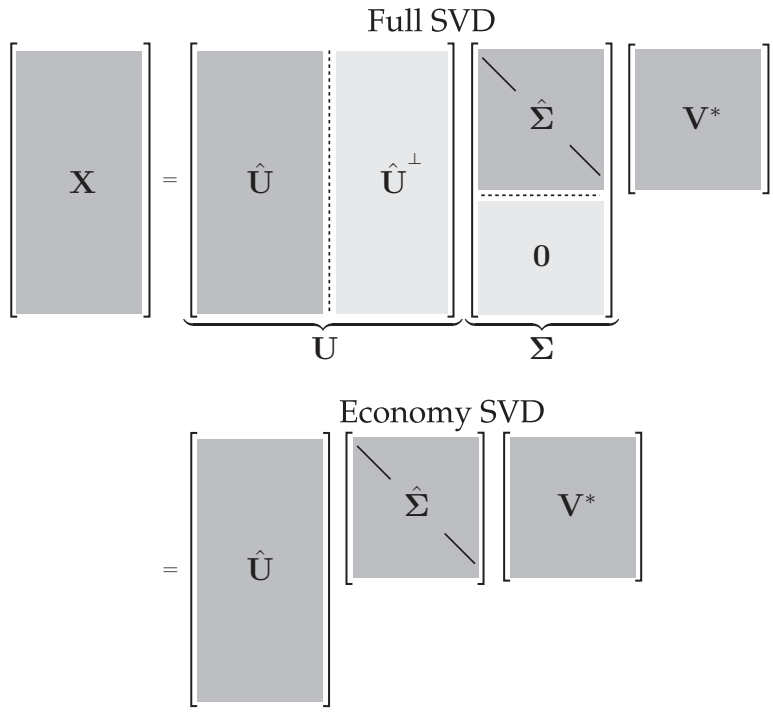

Figure 1.1 Schematic of matrices in the full and economy SVD.

For non-square matrices $\mathbf{X}$, the economy SVD is more efficient:

```
>>[Uhat,Shat,V] = svd(X,'econ');  % economy sized SVD
```

Python

```
>>> import numpy as np
>>> X = np.random.rand(5, 3)  % create random data matrix
>>> U, S, V = np.linalg.svd(X,full_matrices=True) % full SVD
>>> Uhat, Shat, Vhat = np.linalg.svd(X, full_matrices=False)
    % economy SVD
```

R

```
> X <- replicate(3, rnorm(5))
> s <- svd(X)
> U <- s$u
> S <- diag(s$d)
> V <- s$v
```

Mathematica

```
In:= X=RandomReal[{0,1},{5,3}]
In:= {U,S,V} = SingularValueDecomposition[X]
```

Other Languages

The SVD is also available in other languages, such as Fortran and C++. In fact, most SVD implementations are based on the LAPACK (Linear Algebra Package) [13] in Fortran. The

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

SVD routine is designated **DGESVD** in LAPACK, and this is wrapped in the C++ libraries **Armadillo** and **Eigen**.

Historical Perspective

The SVD has a long and rich history, ranging from early work developing the theoretical foundations to modern work on computational stability and efficiency. There is an excellent historical review by Stewart [502], which provides context and many important details. The review focuses on the early theoretical work of Beltrami and Jordan (1873), Sylvester (1889), Schmidt (1907), and Weyl (1912). It also discusses more recent work, including the seminal computational work of Golub and collaborators [212, 211]. In addition, there are many excellent chapters on the SVD in modern texts [524, 17, 316].

Uses in This Book and Assumptions of the Reader

The SVD is the basis for many related techniques in dimensionality reduction. These methods include principal component analysis (PCA) in statistics [418, 256, 257], the Karhunen–Loève transform (KLT) [280, 340], empirical orthogonal functions (EOFs) in climate [344], the proper orthogonal decomposition (POD) in fluid dynamics [251], and canonical correlation analysis (CCA) [131]. Although developed independently in a range of diverse fields, many of these methods only differ in how the data is collected and pre-processed. There is an excellent discussion about the relationship between the SVD, the KLT and PCA by Gerbrands [204].

The SVD is also widely used in system identification and control theory to obtain reduced order models that are balanced in the sense that states are hierarchically ordered in terms of their ability to be observed by measurements and controlled by actuation [388].

For this chapter, we assume that the reader is familiar with linear algebra with some experience in computation and numerics. For review, there are a number of excellent books on numerical linear algebra, with discussions on the SVD [524, 17, 316].

1.2 Matrix Approximation

Perhaps the most useful and defining property of the SVD is that it provides an *optimal* low-rank approximation to a matrix $\mathbf{X}$. In fact, the SVD provides a *hierarchy* of low-rank approximations, since a rank-r approximation is obtained by keeping the leading r singular values and vectors, and discarding the rest.

Schmidt (of Gram-Schmidt) generalized the SVD to function spaces and developed an approximation theorem, establishing truncated SVD as the optimal low-rank approximation of the underlying matrix $\mathbf{X}$ [476]. Schmidt's approximation theorem was rediscovered by Eckart and Young [170], and is sometimes referred to as the Eckart-Young theorem.

Theorem 1 (Eckart-Young [170]) *The optimal rank-r approximation to* $\mathbf{X}$*, in a least-squares sense, is given by the rank-r SVD truncation* $\tilde{\mathbf{X}}$*:*

$$\underset{\tilde{\mathbf{X}},\ s.t.\ \mathrm{rank}(\tilde{\mathbf{X}})=r}{\operatorname{argmin}} \quad \|\mathbf{X}-\tilde{\mathbf{X}}\|_F = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*. \tag{1.4}$$

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

Here, $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$ denote the first r leading columns of $\mathbf{U}$ and $\mathbf{V}$, and $\tilde{\mathbf{\Sigma}}$ contains the leading $r \times r$ sub-block of $\mathbf{\Sigma}$. $\|\cdot\|_F$ is the Frobenius norm.

Here, we establish the notation that a truncated SVD basis (and the resulting approximated matrix $\tilde{\mathbf{X}}$) will be denoted by $\tilde{\mathbf{X}} = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*$. Because $\mathbf{\Sigma}$ is diagonal, the rank-r SVD approximation is given by the sum of r distinct rank-1 matrices:

$$\tilde{\mathbf{X}} = \sum_{k=1}^{r} \sigma_k \mathbf{u}_k \mathbf{v}_k^* = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^* + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^* + \cdots + \sigma_r \mathbf{u}_r \mathbf{v}_r^*. \tag{1.5}$$

This is the so-called *dyadic* summation. For a given rank r, there is no better approximation for $\mathbf{X}$, in the ℓ_2 sense, than the truncated SVD approximation $\tilde{\mathbf{X}}$. Thus, high-dimensional data may be well described by a few dominant patterns given by the columns of $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$.

This is an important property of the SVD, and we will return to it many times. There are numerous examples of data sets that contain high-dimensional measurements, resulting in a large data matrix $\mathbf{X}$. However, there are often dominant low-dimensional patterns in the data, and the truncated SVD basis $\tilde{\mathbf{U}}$ provides a coordinate transformation from the high-dimensional measurement space into a low-dimensional pattern space. This has the benefit of *reducing* the size and dimension of large data sets, yielding a tractable basis for visualization and analysis. Finally, many systems considered in this text are *dynamic* (see Chapter 7), and the SVD basis provides a hierarchy of modes that characterize the observed attractor, on which we may project a low-dimensional dynamical system to obtain reduced order models (see Chapter 12).

Truncation

The truncated SVD is illustrated in Fig. 1.2, with $\tilde{\mathbf{U}}$, $\tilde{\mathbf{\Sigma}}$ and $\tilde{\mathbf{V}}$ denoting the truncated matrices. If $\mathbf{X}$ does not have full rank, then some of the singular values in $\hat{\mathbf{\Sigma}}$ may be zero, and the truncated SVD may still be exact. However, for truncation values r that are smaller than the number of nonzero singular values (i.e., the rank of $\mathbf{X}$), the truncated SVD only approximates $\mathbf{X}$:

$$\mathbf{X} \approx \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*. \tag{1.6}$$

There are numerous choices for the truncation rank r, and they are discussed in Sec. 1.7. If we choose the truncation value to keep all non-zero singular values, then $\mathbf{X} = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*$ is exact.

Example: Image Compression

We demonstrate the idea of matrix approximation with a simple example: image compression. A recurring theme throughout this book is that large data sets often contain underlying patterns that facilitate low-rank representations. Natural images present a simple and intuitive example of this inherent *compressibility*. A grayscale image may be thought of as a real-valued matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$, where n and m are the number of pixels in the vertical and horizontal directions, respectively[3]. Depending on the basis of representation (pixel-space, Fourier frequency domain, SVD transform coordinates), images may have very compact approximations.

[3] It is not uncommon for image size to be specified as horizontal by vertical, i.e. $\mathbf{X}^T \in \mathbb{R}^{m \times n}$, although we stick with vertical by horizontal to be consistent with generic matrix notation.

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

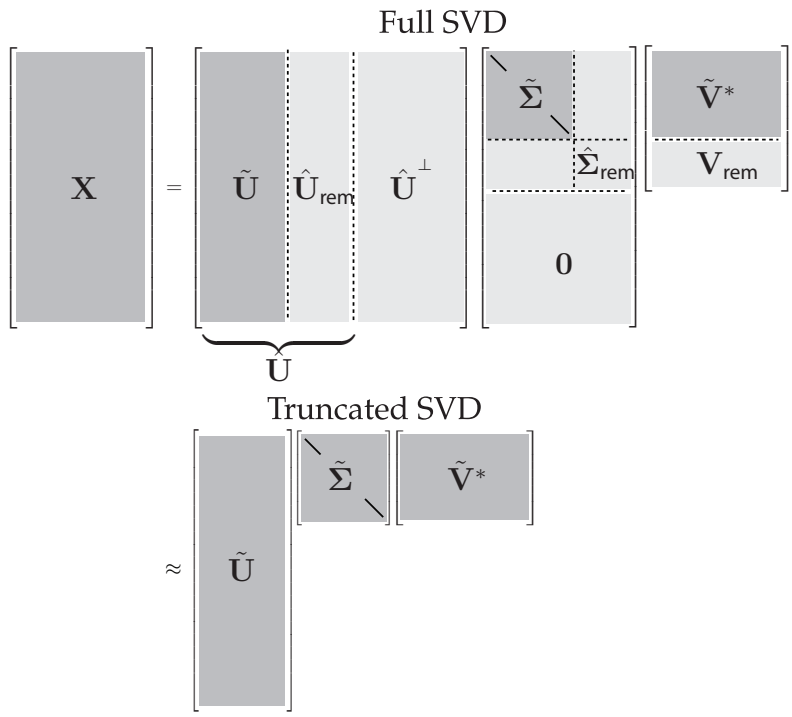

Figure 1.2 Schematic of truncated SVD. The subscript '$_{\text{rem}}$' denotes the remainder of $\hat{\mathbf{U}}$, $\hat{\mathbf{\Sigma}}$ or $\mathbf{V}$ after truncation.

Consider the image of Mordecai the snow dog in Fig. 1.3. This image has 2000×1500 pixels. It is possible to take the SVD of this image and plot the diagonal singular values, as in Fig. 1.4. Figure 1.3 shows the approximate matrix $\tilde{\mathbf{X}}$ for various truncation values r. By $r = 100$, the reconstructed image is quite accurate, and the singular values account for almost 80% of the image variance. The SVD truncation results in a compression of the original image, since only the first 100 columns of $\mathbf{U}$ and $\mathbf{V}$, along with the first 100 diagonal elements of $\mathbf{\Sigma}$, must be stored in $\tilde{\mathbf{U}}$, $\tilde{\mathbf{\Sigma}}$ and $\tilde{\mathbf{V}}$.

First, we load the image:

```
A=imread('../DATA/dog.jpg');
X=double(rgb2gray(A)); % Convert RBG->gray, 256 bit->double.
nx = size(X,1); ny = size(X,2);
imagesc(X), axis off, colormap gray
```

and take the SVD:

```
[U,S,V] = svd(X);
```

Next, we compute the approximate matrix using the truncated SVD for various ranks ($r = 5$, 20, and 100):

```
for r=[5 20 100];  % Truncation value
    Xapprox = U(:,1:r)*S(1:r,1:r)*V(:,1:r)'; % Approx. image
    figure, imagesc(Xapprox), axis off
    title(['r=',num2str(r,'%d'),']);
end
```

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

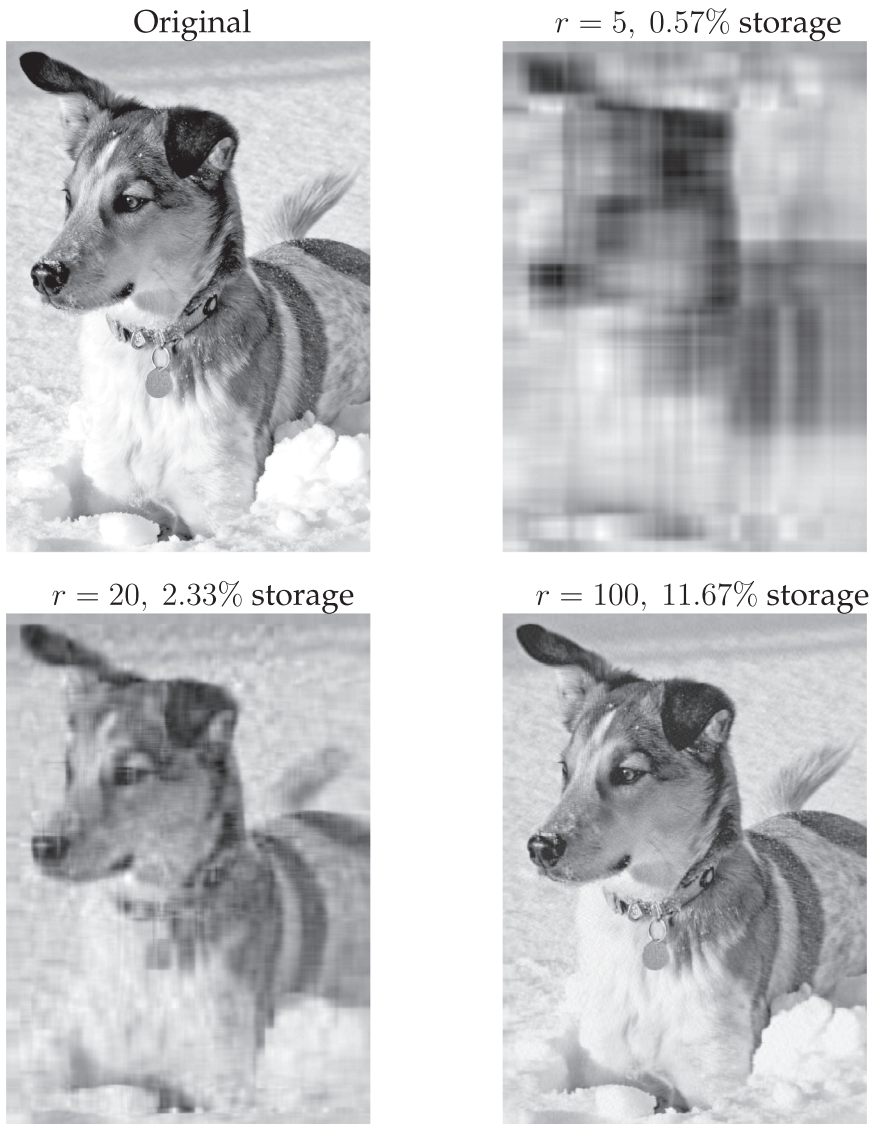

Figure 1.3 Image compression of Mordecai the snow dog, truncating the SVD at various ranks r. Original image resolution is 2000×1500.

Finally, we plot the singular values and cumulative energy in Fig. 1.4:

```
subplot(1,2,1), semilogy(diag(S),'k')
subplot(1,2,2), plot(cumsum(diag(S))/sum(diag(S)),'k')
```

1.3 Mathematical Properties and Manipulations

Here we describe important mathematical properties of the SVD including geometric interpretations of the unitary matrices $\mathbf{U}$ and $\mathbf{V}$ as well as a discussion of the SVD in terms of

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

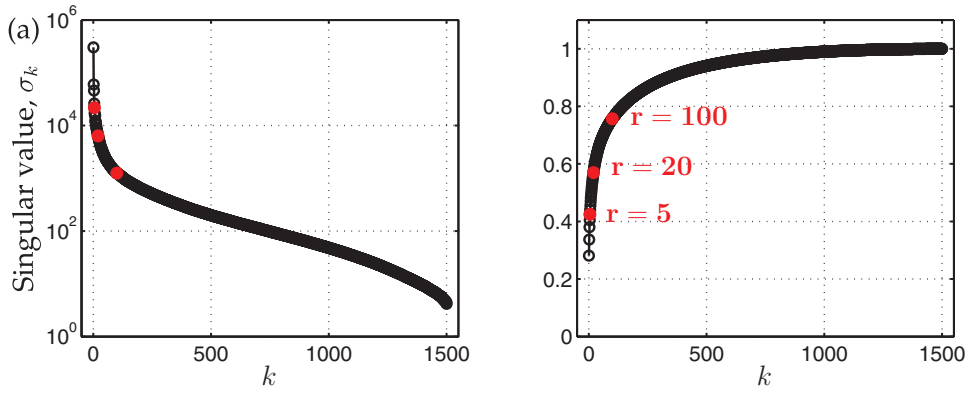

Figure 1.4 (a) Singular values σ_k. (b) Cumulative energy in the first k modes.

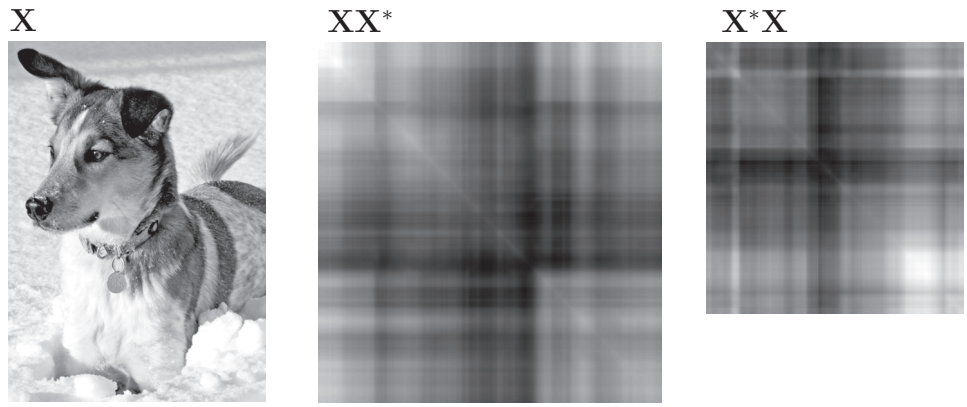

Figure 1.5 Correlation matrices $\mathbf{XX}^*$ and $\mathbf{X}^*\mathbf{X}$ for a matrix $\mathbf{X}$ obtained from an image of a dog. Note that both correlation matrices are symmetric.

dominant correlations in the data $\mathbf{X}$. The relationship between the SVD and correlations in the data will be explored more in Section 1.5 on principal components analysis.

Interpretation as Dominant Correlations

The SVD is closely related to an eigenvalue problem involving the correlation matrices $\mathbf{XX}^*$ and $\mathbf{X}^*\mathbf{X}$, shown in Fig. 1.5 for a specific image, and in Figs. 1.6 and 1.7 for generic matrices. If we plug (1.3) into the row-wise correlation matrix $\mathbf{XX}^*$ and the column-wise correlation matrix $\mathbf{X}^*\mathbf{X}$, we find:

$$\mathbf{XX}^* = \mathbf{U}\begin{bmatrix}\hat{\mathbf{\Sigma}}\\ \mathbf{0}\end{bmatrix}\mathbf{V}^*\mathbf{V}\begin{bmatrix}\hat{\mathbf{\Sigma}} & \mathbf{0}\end{bmatrix}\mathbf{U}^* = \mathbf{U}\begin{bmatrix}\hat{\mathbf{\Sigma}}^2 & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}\mathbf{U}^* \tag{1.7a}$$

$$\mathbf{X}^*\mathbf{X} = \mathbf{V}\begin{bmatrix}\hat{\mathbf{\Sigma}} & \mathbf{0}\end{bmatrix}\mathbf{U}^*\mathbf{U}\begin{bmatrix}\hat{\mathbf{\Sigma}}\\ \mathbf{0}\end{bmatrix}\mathbf{V}^* = \mathbf{V}\hat{\mathbf{\Sigma}}^2\mathbf{V}^*. \tag{1.7b}$$

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

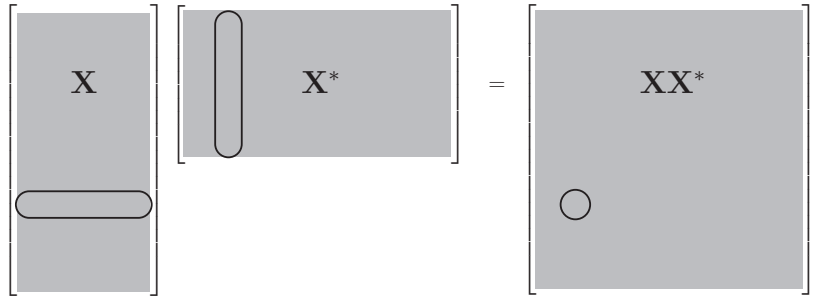

Figure 1.6 Correlation matrix $\mathbf{X}\mathbf{X}^*$ is formed by taking the inner product of rows of $\mathbf{X}$.

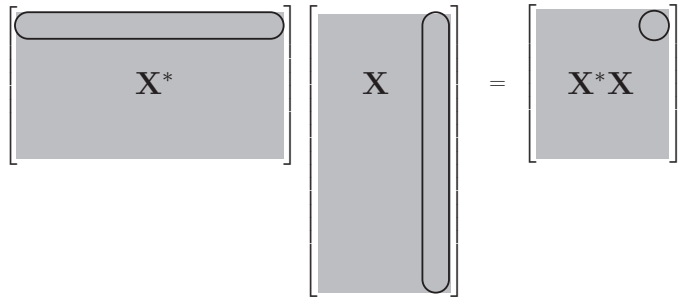

Figure 1.7 Correlation matrix $\mathbf{X}^*\mathbf{X}$ is formed by taking the inner product of columns of $\mathbf{X}$.

Recalling that $\mathbf{U}$ and $\mathbf{V}$ are unitary, $\mathbf{U}$, $\mathbf{\Sigma}$, and $\mathbf{V}$ are solutions to the following eigenvalue problems:

$$\mathbf{X}\mathbf{X}^*\mathbf{U} = \mathbf{U}\begin{bmatrix}\hat{\mathbf{\Sigma}}^2 & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}, \tag{1.8a}$$

$$\mathbf{X}^*\mathbf{X}\mathbf{V} = \mathbf{V}\hat{\mathbf{\Sigma}}^2. \tag{1.8b}$$

In other words, each nonzero singular value of $\mathbf{X}$ is a positive square root of an eigenvalue of $\mathbf{X}^*\mathbf{X}$ and of $\mathbf{X}\mathbf{X}^*$, which have the same nonzero eigenvalues. It follows that if $\mathbf{X}$ is self-adjoint (i.e. $\mathbf{X} = \mathbf{X}^*$), then the singular values of $\mathbf{X}$ are equal to the absolute value of the eigenvalues of $\mathbf{X}$.

This provides an intuitive interpretation of the SVD, where the columns of $\mathbf{U}$ are eigenvectors of the correlation matrix $\mathbf{X}\mathbf{X}^*$ and columns of $\mathbf{V}$ are eigenvectors of $\mathbf{X}^*\mathbf{X}$. We choose to arrange the singular values in descending order by magnitude, and thus the columns of $\mathbf{U}$ are hierarchically ordered by how much correlation they capture in the columns of $\mathbf{X}$; $\mathbf{V}$ similarly captures correlation in the rows of $\mathbf{X}$.

Method of Snapshots

It is often impractical to construct the matrix $\mathbf{X}\mathbf{X}^*$ because of the large size of the state-dimension n, let alone solve the eigenvalue problem; if $\mathbf{x}$ has a million elements, then $\mathbf{X}\mathbf{X}^*$

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

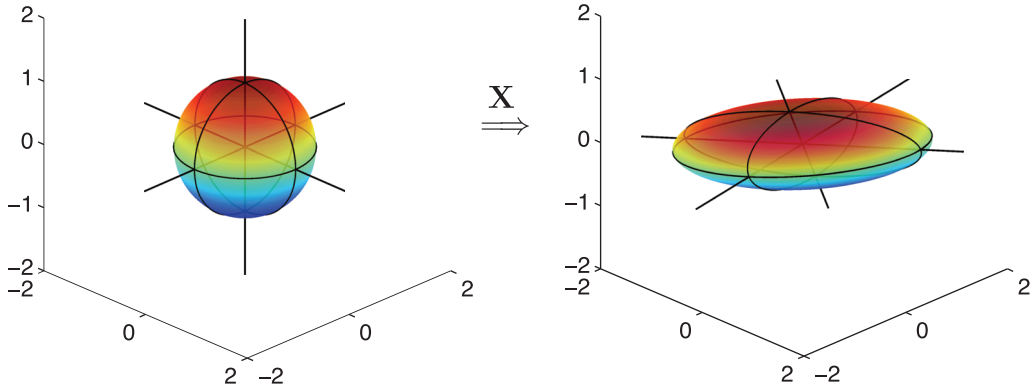

Figure 1.8 Geometric illustration of the SVD as a mapping from a sphere in $\mathbb{R}^n$ to an ellipsoid in $\mathbb{R}^m$.

has a trillion elements. In 1987, Sirovich observed that it is possible to bypass this large matrix and compute the first m columns of $\mathbf{U}$ using what is now known as the method of snapshots [490].

Instead of computing the eigen-decomposition of $\mathbf{X}\mathbf{X}^*$ to obtain the left singular vectors $\mathbf{U}$, we only compute the eigen-decomposition of $\mathbf{X}^*\mathbf{X}$, which is much smaller and more manageable. From (1.8b), we then obtain $\mathbf{V}$ and $\hat{\mathbf{\Sigma}}$. If there are zero singular values in $\hat{\mathbf{\Sigma}}$, then we only keep the r non-zero part, $\tilde{\mathbf{\Sigma}}$, and the corresponding columns $\tilde{\mathbf{V}}$ of $\mathbf{V}$. From these matrices, it is then possible to approximate $\tilde{\mathbf{U}}$, the first r columns of $\mathbf{U}$, as follows:

$$\tilde{\mathbf{U}} = \mathbf{X}\tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}^{-1}. \tag{1.9}$$

Geometric Interpretation

The columns of the matrix $\mathbf{U}$ provide an orthonormal basis for the column space of $\mathbf{X}$. Similarly, the columns of $\mathbf{V}$ provide an orthonormal basis for the row space of $\mathbf{X}$. If the columns of $\mathbf{X}$ are spatial measurements in time, then $\mathbf{U}$ encode spatial patterns, and $\mathbf{V}$ encode temporal patterns.

One property that makes the SVD particularly useful is the fact that both $\mathbf{U}$ and $\mathbf{V}$ are *unitary* matrices, so that $\mathbf{U}\mathbf{U}^* = \mathbf{U}^*\mathbf{U} = \mathbf{I}_{n\times n}$ and $\mathbf{V}\mathbf{V}^* = \mathbf{V}^*\mathbf{V} = \mathbf{I}_{m\times m}$. This means that solving a system of equations involving $\mathbf{U}$ or $\mathbf{V}$ is as simple as multiplication by the transpose, which scales as $\mathcal{O}(n^2)$, as opposed to traditional methods for the generic inverse, which scale as $\mathcal{O}(n^3)$. As noted in the previous section and in [57], the SVD is intimately connected to the spectral properties of the compact self-adjoint operators $\mathbf{X}\mathbf{X}^*$ and $\mathbf{X}^*\mathbf{X}$.

The SVD of $\mathbf{X}$ may be interpreted geometrically based on how a hypersphere, given by $S^{n-1} \triangleq \{\mathbf{x} \,|\, \|\mathbf{x}\|_2 = 1\} \subset \mathbb{R}^n$ maps into an ellipsoid, $\{\mathbf{y} \,|\, \mathbf{y} = \mathbf{X}\mathbf{x} \text{ for } \mathbf{x} \in S^{n-1}\} \subset \mathbb{R}^m$, through $\mathbf{X}$. This is shown graphically in Fig. 1.8 for a sphere in $\mathbb{R}^3$ and a mapping $\mathbf{X}$ with three non-zero singular values. Because the mapping through $\mathbf{X}$ (i.e., matrix multiplication) is linear, knowing how it maps the unit sphere determines how all other vectors will map.

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

For the specific case shown in Fig. 1.8, we construct the matrix $\mathbf{X}$ out of three rotation matrices, $\mathbf{R}_x$, $\mathbf{R}_y$, and $\mathbf{R}_z$, and a fourth matrix to stretch out and scale the principal axes:

$$\mathbf{X} = \underbrace{\begin{bmatrix} \cos(\theta_3) & -\sin(\theta_3) & 0 \\ \sin(\theta_3) & \cos(\theta_3) & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{R}_z} \underbrace{\begin{bmatrix} \cos(\theta_2) & 0 & \sin(\theta_2) \\ 0 & 1 & 0 \\ -\sin(\theta_2) & 0 & \cos(\theta_2) \end{bmatrix}}_{\mathbf{R}_y}$$
$$\times \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_1) & -\sin(\theta_1) \\ 0 & \sin(\theta_1) & \cos(\theta_1) \end{bmatrix}}_{\mathbf{R}_x} \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix}.$$

In this case, $\theta_1 = \pi/15$, $\theta_2 = -\pi/9$, and $\theta_3 = -\pi/20$, and $\sigma_1 = 3$, $\sigma_2 = 1$, and $\sigma_3 = 0.5$. These rotation matrices do not commute, and so the order of rotation matters. If one of the singular values is zero, then a dimension is removed and the ellipsoid collapses onto a lower-dimensional subspace. The product $\mathbf{R}_x\mathbf{R}_y\mathbf{R}_z$ is the unitary matrix $\mathbf{U}$ in the SVD of $\mathbf{X}$. The matrix $\mathbf{V}$ is the identity.

Code 1.1 Construct rotation matrices.

```
theta = [pi/15; -pi/9; -pi/20];
Sigma = diag([3; 1; 0.5]);              % scale x, y, and z

Rx = [1 0 0;                            % rotate about x-axis
    0 cos(theta(1)) -sin(theta(1));
    0 sin(theta(1)) cos(theta(1))];

Ry = [cos(theta(2)) 0 sin(theta(2));    % rotate about y-axis
    0 1 0;
    -sin(theta(2)) 0 cos(theta(2))];

Rz = [cos(theta(3)) -sin(theta(3)) 0; % rotate about z-axis
    sin(theta(3)) cos(theta(3)) 0;
    0 0 1];

X = Rz*Ry*Rx*Sigma;                     % rotate and scale
```

Code 1.2 Plot sphere.

```
[x,y,z] = sphere(25);
h1=surf(x,y,z);
```

Code 1.3 Map sphere through $\mathbf{X}$ and plot resulting ellipsoid.

```
xR = 0*x;  yR = 0*y;  zR = 0*z;
for i=1:size(x,1)
    for j=1:size(x,2)
        vecR = X*[x(i,j); y(i,j); z(i,j)];
        xR(i,j) = vecR(1);
        yR(i,j) = vecR(2);
        zR(i,j) = vecR(3);
    end
end
h2=surf(xR,yR,zR,z);  % using sphere z-coord for color
```

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

Invariance of the SVD to Unitary Transformations

A useful property of the SVD is that if we left or right-multiply our data matrix $\mathbf{X}$ by a unitary transformation, it preserves the terms in the SVD, except for the corresponding left or right unitary matrix $\mathbf{U}$ or $\mathbf{V}$, respectively. This has important implications, since the discrete Fourier transform (DFT; see Chapter 2) $\mathcal{F}$ is a unitary transform, meaning that the SVD of data $\hat{\mathbf{X}} = \mathcal{F}\mathbf{X}$ will be exactly the same as the SVD of $\mathbf{X}$, except that the modes $\hat{\mathbf{U}}$ will be be the DFT of modes $\mathbf{U}$: $\hat{\mathbf{U}} = \mathcal{F}\mathbf{U}$. In addition, the invariance of the SVD to unitary transformations enable the use of compressed measurements to reconstruct SVD modes that are sparse in some transform basis (see Chapter 3).

The invariance of SVD to unitary transformations is geometrically intuitive, as unitary transformations rotate vectors in space, but do not change their inner products or correlation structures. We denote a left unitary transformation by $\mathbf{C}$, so that $\mathbf{Y} = \mathbf{CX}$, and a right unitary transformation by $\mathbf{P}^*$, so that $\mathbf{Y} = \mathbf{XP}^*$. The SVD of $\mathbf{X}$ will be denoted $\mathbf{U_X\Sigma_X V_X^*}$ and the SVD of $\mathbf{Y}$ will be $\mathbf{U_Y\Sigma_Y V_Y^*}$.

Left Unitary Transformations

First, consider a left unitary transformation of $\mathbf{X}$: $\mathbf{Y} = \mathbf{CX}$. Computing the correlation matrix $\mathbf{Y}^*\mathbf{Y}$, we find

$$\mathbf{Y}^*\mathbf{Y} = \mathbf{X}^*\mathbf{C}^*\mathbf{CX} = \mathbf{X}^*\mathbf{X}. \tag{1.10}$$

The projected data has the same eigendecomposition, resulting in the same $\mathbf{V_X}$ and $\mathbf{\Sigma_X}$. Using the method of snapshots to reconstruct $\mathbf{U_Y}$, we find

$$\mathbf{U_Y} = \mathbf{YV_X\Sigma_X^{-1}} = \mathbf{CXV_X\Sigma_X^{-1}} = \mathbf{CU_X}. \tag{1.11}$$

Thus, $\mathbf{U_Y} = \mathbf{CU_X}$, $\mathbf{\Sigma_Y} = \mathbf{\Sigma_X}$, and $\mathbf{V_Y} = \mathbf{V_X}$. The SVD of $\mathbf{Y}$ is then:

$$\mathbf{Y} = \mathbf{CX} = \mathbf{CU_X\Sigma_X V_X^*}. \tag{1.12}$$

Right Unitary Transformations

For a right unitary transformation $\mathbf{Y} = \mathbf{XP}^*$, the correlation matrix $\mathbf{Y}^*\mathbf{Y}$ is:

$$\mathbf{Y}^*\mathbf{Y} = \mathbf{PX}^*\mathbf{XP}^* = \mathbf{PV_X\Sigma_X^2 V_X^*P}^*, \tag{1.13}$$

with the following eigendecomposition

$$\mathbf{Y}^*\mathbf{YPV_X} = \mathbf{PV_X\Sigma_X^2}. \tag{1.14}$$

Thus, $\mathbf{V_Y} = \mathbf{PV_X}$ and $\mathbf{\Sigma_Y} = \mathbf{\Sigma_X}$. We may use the method of snapshots to reconstruct $\mathbf{U_Y}$:

$$\mathbf{U_Y} = \mathbf{YPV_X\Sigma_X^{-1}} = \mathbf{XV_X\Sigma_X^{-1}} = \mathbf{U_X}. \tag{1.15}$$

Thus, $\mathbf{U_Y} = \mathbf{U_X}$, and we may write the SVD of $\mathbf{Y}$ as:

$$\mathbf{Y} = \mathbf{XP}^* = \mathbf{U_X\Sigma_X V_X^*P}^*. \tag{1.16}$$

1.4 Pseudo-Inverse, Least-Squares, and Regression

Many physical systems may be represented as a linear system of equations:

$$\mathbf{Ax} = \mathbf{b}, \tag{1.17}$$

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

where the constraint matrix $\mathbf{A}$ and vector $\mathbf{b}$ are known, and the vector $\mathbf{x}$ is unknown. If $\mathbf{A}$ is a square, invertible matrix (i.e., $\mathbf{A}$ has nonzero determinant), then there exists a unique solution $\mathbf{x}$ for every $\mathbf{b}$. However, when $\mathbf{A}$ is either singular or rectangular, there may be one, none, or infinitely many solutions, depending on the specific $\mathbf{b}$ and the column and row spaces of $\mathbf{A}$.

First, consider the *underdetermined system*, where $\mathbf{A} \in \mathbb{C}^{n\times m}$ and $n \ll m$ (i.e., $\mathbf{A}$ is a short-fat matrix), so that there are fewer equations than unknowns. This type of system is likely to have full column rank, since it has many more columns than are required for a linearly independent basis[4]. Generically, if a short-fat $\mathbf{A}$ has full column rank, then there are infinitely many solutions $\mathbf{x}$ for every $\mathbf{b}$. The system is called *underdetermined* because there are not enough values in $\mathbf{b}$ to uniquely determine the higher-dimensional $\mathbf{x}$.

Similarly, consider the *overdetermined system*, where $n \gg m$ (i.e., a tall-skinny matrix), so that there are more equations than unknowns. This matrix cannot have a full column rank, and so it is guaranteed that there are vectors $\mathbf{b}$ that have no solution $\mathbf{x}$. In fact, there will only be a solution $\mathbf{x}$ if $\mathbf{b}$ is in the column space of $\mathbf{A}$, i.e. $\mathbf{b} \in \text{col}(\mathbf{A})$.

Technically, there may be some choices of $\mathbf{b}$ that admit infinitely many solutions $\mathbf{x}$ for a tall-skinny matrix $\mathbf{A}$ and other choices of $\mathbf{b}$ that admit zero solutions even for a short-fat matrix. The solution space to the system in (1.17) is determined by the four fundamental subspaces of $\mathbf{A} = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*$, where the rank r is chosen to include all nonzero singular values:

- The column space, $\text{col}(\mathbf{A})$, is the span of the columns of $\mathbf{A}$, also known as the *range*. The column space of $\mathbf{A}$ is the same as the column space of $\tilde{\mathbf{U}}$;
- The orthogonal complement to $\text{col}(\mathbf{A})$ is $\ker(\mathbf{A}^*)$, given by the column space of $\hat{\mathbf{U}}^\perp$ from Fig. 1.1;
- The row space, $\text{row}(\mathbf{A})$, is the span of the rows of $\mathbf{A}$, which is spanned by the columns of $\tilde{\mathbf{V}}$. The row space of $\mathbf{A}$ is equal to $\text{row}(\mathbf{A}) = \text{col}(\mathbf{A}^*)$;
- The kernel space, $\ker(\mathbf{A})$, is the orthogonal complement to $\text{row}(\mathbf{A})$, and is also known as the *null space*. The null space is the subspace of vectors that map through $\mathbf{A}$ to zero, i.e., $\mathbf{Ax} = \mathbf{0}$, given by $\text{col}(\hat{\mathbf{V}}^\perp)$.

More precisely, if $\mathbf{b} \in \text{col}(\mathbf{A})$ and if $\dim(\ker(\mathbf{A})) \neq 0$, then there are infinitely many solutions $\mathbf{x}$. Note that the condition $\dim(\ker(\mathbf{A})) \neq 0$ is guaranteed for a short-fat matrix. Similarly, if $\mathbf{b} \notin \text{col}(\mathbf{A})$, then there are no solutions, and the system of equations in (1.17) are called *inconsistent*.

The fundamental subspaces above satisfy the following properties:

$$\text{col}(\mathbf{A}) \oplus \ker(\mathbf{A}^*) = \mathbb{R}^n \tag{1.18a}$$

$$\text{col}(\mathbf{A}^*) \oplus \ker(\mathbf{A}) = \mathbb{R}^n. \tag{1.18b}$$

Remark 1 *There is an extensive literature on random matrix theory, where the above stereotypes are almost certainly true, meaning that they are true with high probability. For example, a system* $\mathbf{Ax} = \mathbf{b}$ *is extremely unlikely to have a solution for a random matrix* $\mathbf{A} \in \mathbb{R}^{n\times m}$ *and random vector* $\mathbf{b} \in \mathbb{R}^n$ *with* $n \gg m$*, since there is little chance that* $\mathbf{b}$ *is in*

[4] It is easy to construct degenerate examples where a short-fat matrix does not have full column rank, such as $\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$.

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

the column space of $\mathbf{A}$. *These properties of random matrices will play a prominent role in compressed sensing (see Chapter 3).*

In the overdetermined case when no solution exists, we would often like to find the solution $\mathbf{x}$ that minimizes the sum-squared error $\|\mathbf{Ax} - \mathbf{b}\|_2^2$, the so-called *least-squares* solution. Note that the least-squares solution also minimizes $\|\mathbf{Ax} - \mathbf{b}\|_2$. In the underdetermined case when infinitely many solutions exist, we may like to find the solution $\mathbf{x}$ with minimum norm $\|\mathbf{x}\|_2$ so that $\mathbf{Ax} = \mathbf{b}$, the so-called *minimum-norm* solution.

The SVD is the technique of choice for these important optimization problems. First, if we substitute an exact truncated SVD $\mathbf{A} = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*$ in for $\mathbf{A}$, we can "invert" each of the matrices $\tilde{\mathbf{U}}$, $\tilde{\mathbf{\Sigma}}$, and $\tilde{\mathbf{V}}^*$ in turn, resulting in the Moore-Penrose *left pseudo-inverse* [425, 426, 453, 572] $\mathbf{A}^\dagger$ of $\mathbf{A}$:

$$\mathbf{A}^\dagger \triangleq \tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}^{-1}\tilde{\mathbf{U}}^* \quad \Longrightarrow \quad \mathbf{A}^\dagger\mathbf{A} = \mathbf{I}_{m\times m}. \tag{1.19}$$

This may be used to find both the minimum norm and least-squares solutions to (1.17):

$$\mathbf{A}^\dagger\mathbf{A}\tilde{\mathbf{x}} = \mathbf{A}^\dagger\mathbf{b} \quad \Longrightarrow \quad \tilde{\mathbf{x}} = \tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}^{-1}\tilde{\mathbf{U}}^*\mathbf{b}. \tag{1.20}$$

Plugging the solution $\tilde{\mathbf{x}}$ back in to (1.17) results in:

$$\mathbf{A}\tilde{\mathbf{x}} = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*\tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}^{-1}\tilde{\mathbf{U}}^*\mathbf{b} \tag{1.21a}$$

$$= \tilde{\mathbf{U}}\tilde{\mathbf{U}}^*\mathbf{b}. \tag{1.21b}$$

Note that $\tilde{\mathbf{U}}\tilde{\mathbf{U}}^*$ is not necessarily the identity matrix, but is rather a projection onto the column space of $\tilde{\mathbf{U}}$. Therefore, $\tilde{\mathbf{x}}$ will only be an exact solution to (1.17) when $\mathbf{b}$ is in the column space of $\tilde{\mathbf{U}}$, and therefore in the column space of $\mathbf{A}$.

Computing the pseudo-inverse $\mathbf{A}^\dagger$ is computationally efficient, after the expensive upfront cost of computing the SVD. Inverting the unitary matrices $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$ involves matrix multiplication by the transpose matrices, which are $\mathcal{O}(n^2)$ operations. Inverting $\tilde{\mathbf{\Sigma}}$ is even more efficient since it is a diagonal matrix, requiring $\mathcal{O}(n)$ operations. In contrast, inverting a dense square matrix would require an $\mathcal{O}(n^3)$ operation.

One-Dimensional Linear Regression

Regression is an important statistical tool to relate variables to one another based on data [360]. Consider the collection of data in Fig. 1.9. The red $\times$'s are obtained by adding Gaussian white noise to the black line, as shown in Code 1.4. We assume that the data is linearly related, as in (1.17), and we use the pseudo-inverse to find the least-squares solution for the slope x below (blue dashed line), shown in Code 1.5:

$$\begin{bmatrix} | \\ \mathbf{b} \\ | \end{bmatrix} = \begin{bmatrix} | \\ \mathbf{a} \\ | \end{bmatrix} x \;=\; \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^* x. \tag{1.22a}$$

$$\Longrightarrow \quad x = \tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}^{-1}\tilde{\mathbf{U}}^*\mathbf{b}. \tag{1.22b}$$

In (1.22b), $\tilde{\mathbf{\Sigma}} = \|\mathbf{a}\|_2$, $\tilde{\mathbf{V}} = 1$, and $\tilde{\mathbf{U}} = \mathbf{a}/\|\mathbf{a}\|_2$. Taking the left pseudo-inverse:

$$x = \frac{\mathbf{a}^*\mathbf{b}}{\|\mathbf{a}\|_2^2}. \tag{1.23}$$

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

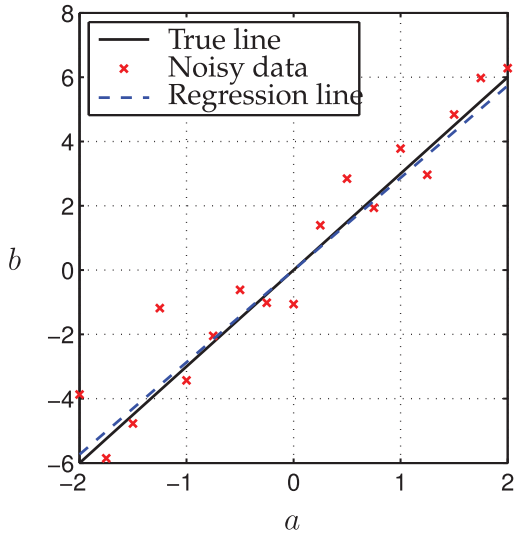

Figure 1.9 Illustration of linear regression using noisy data.

This makes physical sense, if we think of x as the value that best maps our vector **a** to the vector **b**. Then, the best single value x is obtained by taking the dot product of **b** with the normalized **a** direction. We then add a second normalization factor $\|\mathbf{a}\|_2$ because the **a** in (1.22a) is not normalized.

Note that strange things happen if you use row vectors instead of column vectors in (1.22). Also, if the noise magnitude becomes large relative to the slope x, the pseudo-inverse will undergo a phase-change in accuracy, related to the hard-thresholding results in subsequent sections.

Code 1.4 Generate noisy data for Fig. 1.9.

```
x = 3;                              % True slope
a = [-2:.25:2]';
b = a*x + 1*randn(size(a));         % Add noise
plot(a,x*a,'k')                     % True relationship
hold on, plot(a,b,'rx')             % Noisy measurements
```

Code 1.5 Compute least-squares approximation for Fig. 1.9.

```
[U,S,V] = svd(a,'econ');
xtilde = V*inv(S)*U'*b;             % Least-square fit
plot(a,xtilde*a,'b--')              % Plot fit
```

The procedure above is called *linear regression* in statistics. There is a **regress** command in Matlab, as well as a **pinv** command that may also be used.

Code 1.6 Alternative formulations of least-squares in Matlab.

```
xtilde1 = V*inv(S)*U'*b
xtilde2 = pinv(a)*b
xtilde3 = regress(b,a)
```

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

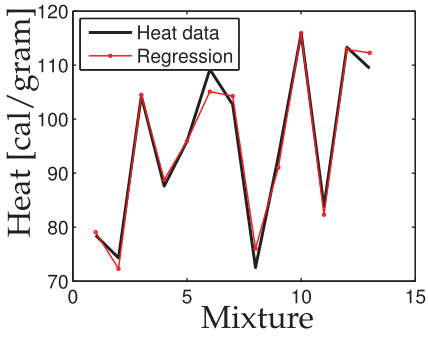

Figure 1.10 Heat data for cement mixtures containing four basic ingredients.

Multilinear regression

Example 1: Cement heat generation data

First, we begin with a simple built-in Matlab dataset that describes the heat generation for various cement mixtures comprised of four basic ingredients. In this problem, we are solving (1.17) where $\mathbf{A} \in \mathbb{R}^{13\times 4}$, since there are four ingredients and heat measurements for 13 unique mixtures. The goal is to determine the weighting $\mathbf{x}$ that relates the proportions of the four ingredients to the heat generation. It is possible to find the minimum error solution using the SVD, as shown in Code 1.7. Alternatives, using **regress** and **pinv**, are also explored.

Code 1.7 Multilinear regression for cement heat data.

```
load hald;  % Load Portlant Cement dataset
A = ingredients;
b = heat;

[U,S,V] = svd(A,'econ');
x = V*inv(S)*U'*b;          % Solve Ax=b using the SVD

plot(b,'k');  hold on       % Plot data
plot(A*x,'r-o',);           % Plot fit

x = regress(b,A);           % Alternative 1  (regress)
x = pinv(A)*b;              % Alternative 2  (pinv)
```

Example 2: Boston Housing Data

In this example, we explore a larger data set to determine which factors best predict prices in the Boston housing market [234]. This data is available from the UCI Machine Learning Repository [24].

There are 13 attributes that are correlated with house price, such as per capita crime rate and property-tax rate. These features are regressed onto the price data, and the best fit price prediction is plotted against the true house value in Fig. 1.11, and the regression coefficients are shown in Fig. 1.12. Although the house value is not perfectly predicted, the trend agrees quite well. It is often the case that the highest value outliers are not well-captured by simple linear fits, as in this example.

This data contains prices and attributes for 506 homes, so the attribute matrix is of size 506×13. It is important to pad this matrix with an additional column of ones, to take

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

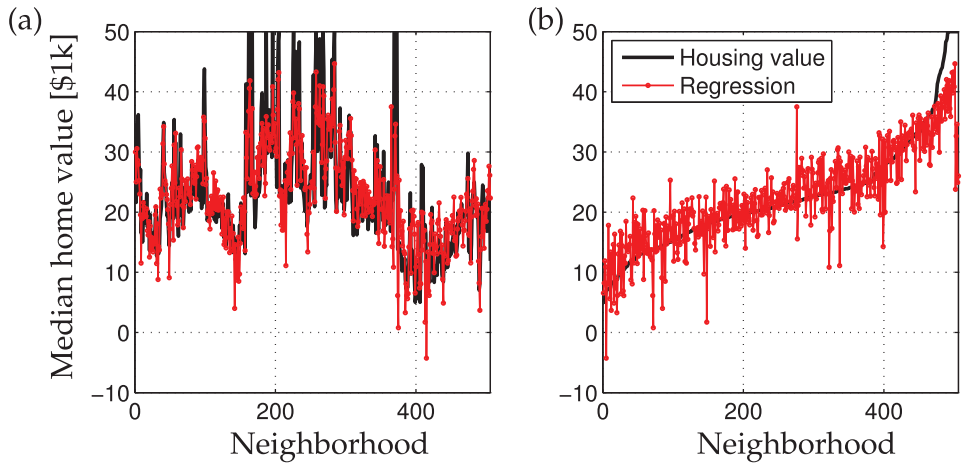

Figure 1.11 Multilinear regression of home prices using various factors. (a) Unsorted data, and (b) Data sorted by home value.

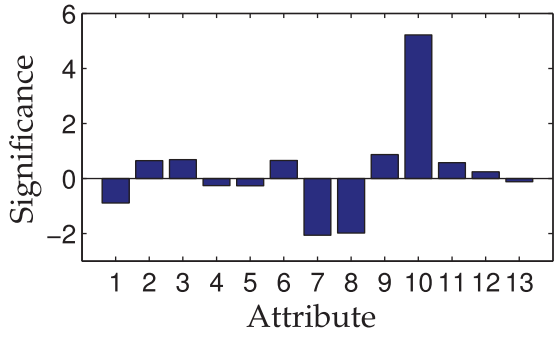

Figure 1.12 Significance of various attributes in the regression.

into account the possibility of a nonzero constant offset in the regression formula. This corresponds to the "y-intercept" in a simple one-dimensional linear regression.

Code 1.8 Multilinear regression for Boston housing data.

```
load housing.data

b = housing(:,14);        % housing values in $1000s
A = housing(:,1:13);      % other factors,
A = [A ones(size(A,1),1)];   % Pad with ones y-intercept

x = regress(b,A);
plot(b,'k-o');
hold on, plot(A*x,'r-o');

[b sortind] = sort(housing(:,14)); % sorted values
plot(b,'k-o')
hold on, plot(A(sortind,:)*x,'r-o')
```

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

Caution

In general, the matrix $\mathbf{U}$, whose columns are left-singular vectors of $\mathbf{X}$, is a unitary square matrix. Therefore, $\mathbf{U}^*\mathbf{U} = \mathbf{U}\mathbf{U}^* = \mathbf{I}_{n\times n}$. However, to compute the pseudo-inverse of $\mathbf{X}$, we must compute $\mathbf{X}^\dagger = \tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}^{-1}\tilde{\mathbf{U}}^*$ since only $\tilde{\mathbf{\Sigma}}$ is invertible (if all singular values are nonzero), although $\mathbf{\Sigma}$ is not invertible in general (in fact, it is generally not even square).

Until now, we have assumed that $\mathbf{X} = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*$ is an exact SVD, so that the rank r includes all nonzero singular values. This guarantees that the matrix $\tilde{\mathbf{\Sigma}}$ is invertible.

A complication arises when working with a truncated basis of left singular vectors $\tilde{\mathbf{U}}$. It is still true that $\tilde{\mathbf{U}}^*\tilde{\mathbf{U}} = \mathbf{I}_{r\times r}$, where r is the rank of $\mathbf{X}$. However, $\tilde{\mathbf{U}}\tilde{\mathbf{U}}^* \neq \mathbf{I}_{n\times n}$, which is easy to verify numerically on a simple example. Assuming that $\tilde{\mathbf{U}}\tilde{\mathbf{U}}^*$ is equal to the identity is one of the most common accidental misuses of the SVD[5].

```
>> tol = 1.e-16;
>> [U,S,V] = svd(X,'econ')
>> r = max(find(diag(S)>max(S(:))*tol));
>> invX = V(:,1:r)*S(1:r,1:r)*U(:,1:r)';  % only approximate
```

1.5 Principal Component Analysis (PCA)

Principal components analysis (PCA) is one of the central uses of the SVD, providing a data-driven, hierarchical coordinate system to represent high-dimensional correlated data. This coordinate system involves the correlation matrices described in Sec. 1.3. Importantly, PCA pre-processes the data by mean subtraction and setting the variance to unity before performing the SVD. The geometry of the resulting coordinate system is determined by principal components (PCs) that are uncorrelated (orthogonal) to each other, but have maximal correlation with the measurements. This theory was developed in 1901 by Pearson [418], and independently by Hotelling in the 1930s [256, 257]. Jolliffe [268] provides a good reference text.

Typically, a number of measurements are collected in a single experiment, and these measurements are arranged into a row vector. The measurements may be features of an observable, such as demographic features of a specific human individual. A number of experiments are conducted, and each measurement vector is arranged as a row in a large matrix $\mathbf{X}$. In the example of demography, the collection of experiments may be gathered via polling. Note that this convention for $\mathbf{X}$, consisting of rows of features, is different than the convention throughout the remainder of this chapter, where individual feature "snapshots" are arranged as columns. However, we choose to be consistent with PCA literature in this section. The matrix will still be size $n \times m$, although it may have more rows than columns, or vice versa.

Computation

We now compute the row-wise mean $\bar{\mathbf{x}}$ (i.e., the mean of all rows), and subtract it from $\mathbf{X}$. The mean $\bar{\mathbf{x}}$ is given by

$$\bar{\mathbf{x}}_j = \frac{1}{n}\sum_{i=1}^{n}\mathbf{X}_{ij}, \tag{1.24}$$

[5] The authors are not immune to this, having mistakenly used this fictional identity in an early version of [96].

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

and the mean matrix is

$$\bar{\mathbf{X}} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \bar{\mathbf{x}}. \tag{1.25}$$

Subtracting $\bar{\mathbf{X}}$ from $\mathbf{X}$ results in the mean-subtracted data $\mathbf{B}$:

$$\mathbf{B} = \mathbf{X} - \bar{\mathbf{B}}. \tag{1.26}$$

The covariance matrix of the rows of $\mathbf{B}$ is given by

$$\mathbf{C} = \frac{1}{n-1}\mathbf{B}^*\mathbf{B}. \tag{1.27}$$

The first principal component $\mathbf{u}_1$ is given as

$$\mathbf{u}_1 = \underset{\|\mathbf{u}_1\|=1}{\operatorname{argmax}} \ \mathbf{u}_1^*\mathbf{B}^*\mathbf{B}\mathbf{u}_1, \tag{1.28}$$

which is the eigenvector of $\mathbf{B}^*\mathbf{B}$ corresponding to the largest eigenvalue. Now it is clear that $\mathbf{u}_1$ is the left singular vector of $\mathbf{B}$ corresponding to the largest singular value.

It is possible to obtain the principal components by computing the eigen-decomposition of $\mathbf{C}$:

$$\mathbf{CV} = \mathbf{VD}, \tag{1.29}$$

which is guaranteed to exist, since $\mathbf{C}$ is Hermitian.

pca *Command*

In Matlab, there the additional commands **pca** and **princomp** (based on **pca**) for the principal components analysis:

```
>> [V,score,s2] = pca(X);
```

The matrix $\mathbf{V}$ is equivalent to the $\mathbf{V}$ matrix from the SVD of $\mathbf{X}$, up to sign changes of the columns. The vector **s2** contains eigenvalues of the covariance of $\mathbf{X}$, also known as principal component variances; these values are the squares of the singular values. The variable **score** simply contains the coordinates of each row of $\mathbf{B}$ (the mean-subtracted data) in the principal component directions. In general, we often prefer to use the **svd** command with the various pre-processing steps described earlier in the section.

Example: Noisy Gaussian Data

Consider the noisy cloud of data in Fig. 1.13 (a), generated using Code 1.9. The data is generated by selecting 10,000 vectors from a two-dimensional normal distribution with zero mean and unit variance. These vectors are then scaled in the x and y directions by the values in Table 1.1 and rotated by $\pi/3$. Finally, the entire cloud of data is translated so that it has a nonzero center $\mathbf{x}_C = \begin{bmatrix} 2 & 1 \end{bmatrix}^T$.

Using Code 1.10, the PCA is performed and used to plot confidence intervals using multiple standard deviations, shown in Fig. 1.13 (b). The singular values, shown in Table 1.1, match the data scaling. The matrix $\mathbf{U}$ from the SVD also closely matches the rotation matrix, up to a sign on the columns:

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

Table 1.1 Standard deviation of data and normalized singular values.

	σ_1	σ_2
Data	2	0.5
SVD	1.974	0.503

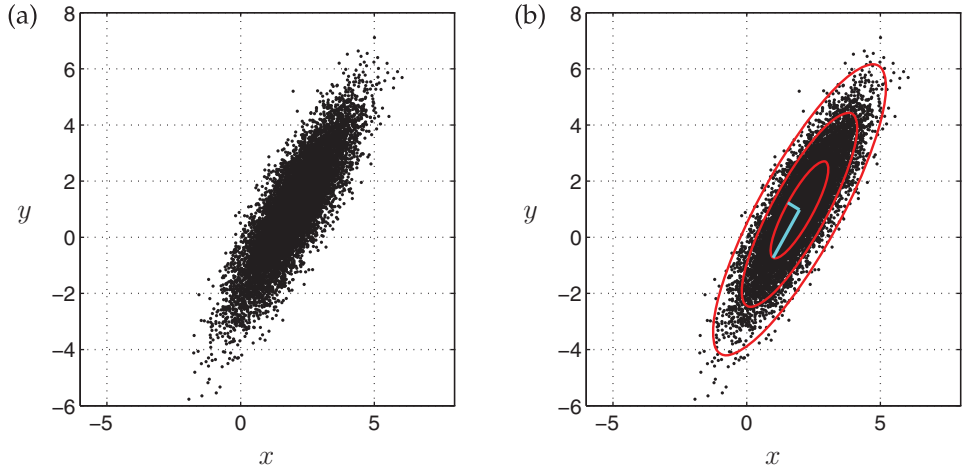

Figure 1.13 Principal components capture the variance of mean-subtracted Gaussian data (a). The first three standard deviation ellipsoids (red), and the two left singular vectors, scaled by singular values ($\sigma_1 u_1 + x_C$ and $\sigma_2 u_2 + x_C$, cyan), are shown in (b).

$$\mathbf{R}_{\pi/3} = \begin{bmatrix} 0.5 & -0.8660 \\ 0.8660 & 0.5 \end{bmatrix}, \qquad \mathbf{U} = \begin{bmatrix} -0.4998 & -0.8662 \\ -0.8662 & 0.4998 \end{bmatrix}.$$

Code 1.9 Generation of noisy cloud of data to illustrate PCA.

```
xC = [2; 1;];                              % Center of data (mean)
sig = [2; .5;];                            % Principal axes

theta = pi/3;                              % Rotate cloud by pi/3
R = [cos(theta) -sin(theta);               % Rotation matrix
     sin(theta) cos(theta)];

nPoints = 10000;                           % Create 10,000 points
X = R*diag(sig)*randn(2,nPoints) + diag(xC)*ones(2,nPoints);
scatter(X(1,:),X(2,:),'k.','LineWidth',2)
```

Code 1.10 Compute PCA and plot confidence intervals.

```
Xavg = mean(X,2);                          % Compute mean
B = X - Xavg*ones(1,nPoints);              % Mean-subtracted Data
[U,S,V] = svd(B/sqrt(nPoints),'econ'); % PCA via SVD
scatter(X(1,:),X(2,:),'k.','LineWidth',2) % Plot data

theta = (0:.01:1)*2*pi;
Xstd = U*S*[cos(theta); sin(theta)]; % 1-std conf. interval
```

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

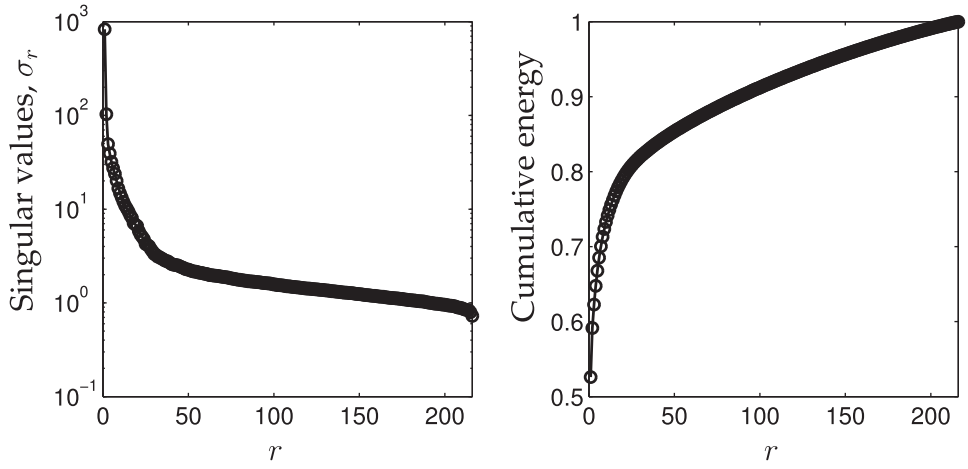

Figure 1.14 Singular values for the Ovarian cancer data.

```
plot(Xavg(1)+Xstd(1,:),Xavg(2) + Xstd(2,:),'r-')
plot(Xavg(1)+2*Xstd(1,:),Xavg(2) + 2*Xstd(2,:),'r-')
plot(Xavg(1)+3*Xstd(1,:),Xavg(2) + 3*Xstd(2,:),'r-')
```

Finally, it is also possible to compute using the **pca** command:

```
>> [V,score,s2] = pca(X);
>> norm(V*score' - B)

ans =
   2.2878e-13
```

Example: Ovarian Cancer Data

The ovarian cancer data set, which is built into Matlab, provides a more realistic example to illustrate the benefits of PCA. This example consists of gene data for 216 patients, 121 of whom have ovarian cancer, and 95 of whom do not. For each patient, there is a vector of data containing the expression of 4000 genes. There are multiple challenges with this type of data, namely the high dimension of the data features. However, we see from Fig. 1.14 that there is significant variance captured in the first few PCA modes. Said another way, the gene data is highly correlated, so that many patients have significant overlap in their gene expression. The ability to visualize patterns and correlations in high-dimensional data is an important reason to use PCA, and PCA has been widely used to find patterns in high-dimensional biological and genetic data [448].

More importantly, patients with ovarian cancer appear to cluster separately from patients without cancer when plotted in the space spanned by the first three PCA modes. This is shown in Fig. 1.15, which is generated by Code 1.11. This inherent clustering in PCA space of data by category is a foundational element of machine learning and pattern recognition. For example, we will see in Sec. 1.6 that images of different human faces will form clusters in PCA space. The use of these clusters will be explored in greater detail in Chapter 5.

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

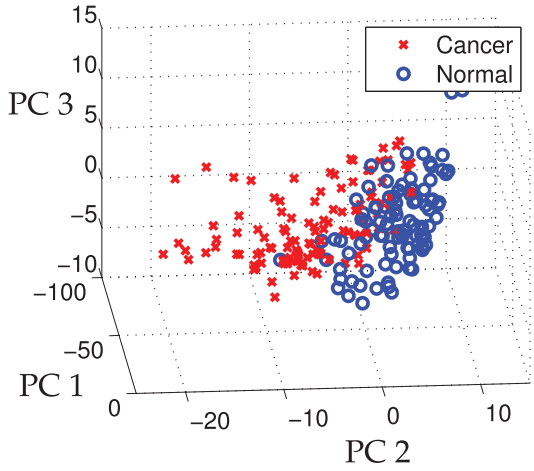

Figure 1.15 Clustering of samples that are normal and those that have cancer in the first three principal component coordinates.

Code 1.11 Compute PCA for ovarian cancer data.

```
load ovariancancer;  % Load ovarian cancer data
[U,S,V] = svd(obs,'econ');
for i=1:size(obs,1)
    x = V(:,1)'*obs(i,:)';
    y = V(:,2)'*obs(i,:)';
    z = V(:,3)'*obs(i,:)';
    if(grp{i}=='Cancer')
        plot3(x,y,z,'rx','LineWidth',2);
    else
        plot3(x,y,z,'bo','LineWidth',2);
    end
end
```

1.6 Eigenfaces Example

One of the most striking demonstrations of SVD/PCA is the so-called eigenfaces example. In this problem, PCA (i.e. SVD on mean-subtracted data) is applied to a large library of facial images to extract the most dominant correlations between images. The result of this decomposition is a set of *eigenfaces* that define a new coordinate system. Images may be represented in these coordinates by taking the dot product with each of the principal components. It will be shown in Chapter 5 that images of the same person tend to cluster in the eigenface space, making this a useful transformation for facial recognition and classification [510, 48]. The eigenface problem was first studied by Sirovich and Kirby in 1987 [491] and expanded on in [291]. Its application to automated facial recognition was presented by Turk and Pentland in 1991 [537].

Here, we demonstrate this algorithm using the Extended Yale Face Database B [203], consisting of cropped and aligned images [327] of 38 individuals (28 from the extended database, and 10 from the original database) under 9 poses and 64 lighting conditions[6].

[6] The database can be downloaded at http://vision.ucsd.edu/~iskwak/ExtYaleDatabase/ExtYaleB.html.

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

Figure 1.16 (left) A single image for each person in the Yale database, and (right) all images for a specific person. Left panel generated by Code (1.12).

Each image is 192 pixels tall and 168 pixels wide. Unlike the previous image example in Section 1.2, each of the facial images in our library have been reshaped into a large column vector with $192 \times 168 = 32,256$ elements. We use the first 36 people in the database (left panel of Fig. 1.16) as our training data for the eigenfaces example, and we hold back two people as a test set. An example of all 64 images of one specific person are shown in the right panel. These images are loaded and plotted using Code 1.12.

Code 1.12 Plot an image for each person in the Yale database (Fig. 1.16 (a))

```
load ../DATA/allFaces.mat

allPersons = zeros(n*6,m*6);       % Make an array to fit all
    faces
count = 1;
for i=1:6          % 6 x 6 grid of faces
    for j=1:6
        allPersons(1+(i-1)*n:i*n,1+(j-1)*m:j*m) ...
            =reshape(faces(:,1+sum(nfaces(1:count-1))),n,m);
        count = count + 1;
    end
end
imagesc(allPersons), colormap gray
```

As mentioned before, each image is reshaped into a large column vector, and the average face is computed and subtracted from each column vector. The mean-subtracted image vectors are then stacked horizontally as columns in the data matrix $\mathbf{X}$, as shown in Fig. 1.17. Thus, taking the SVD of the mean-subtracted matrix $\mathbf{X}$ results in the PCA. The columns of $\mathbf{U}$ are the eigenfaces, and they may be reshaped back into 192×168 images. This is illustrated in Code 1.13.

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

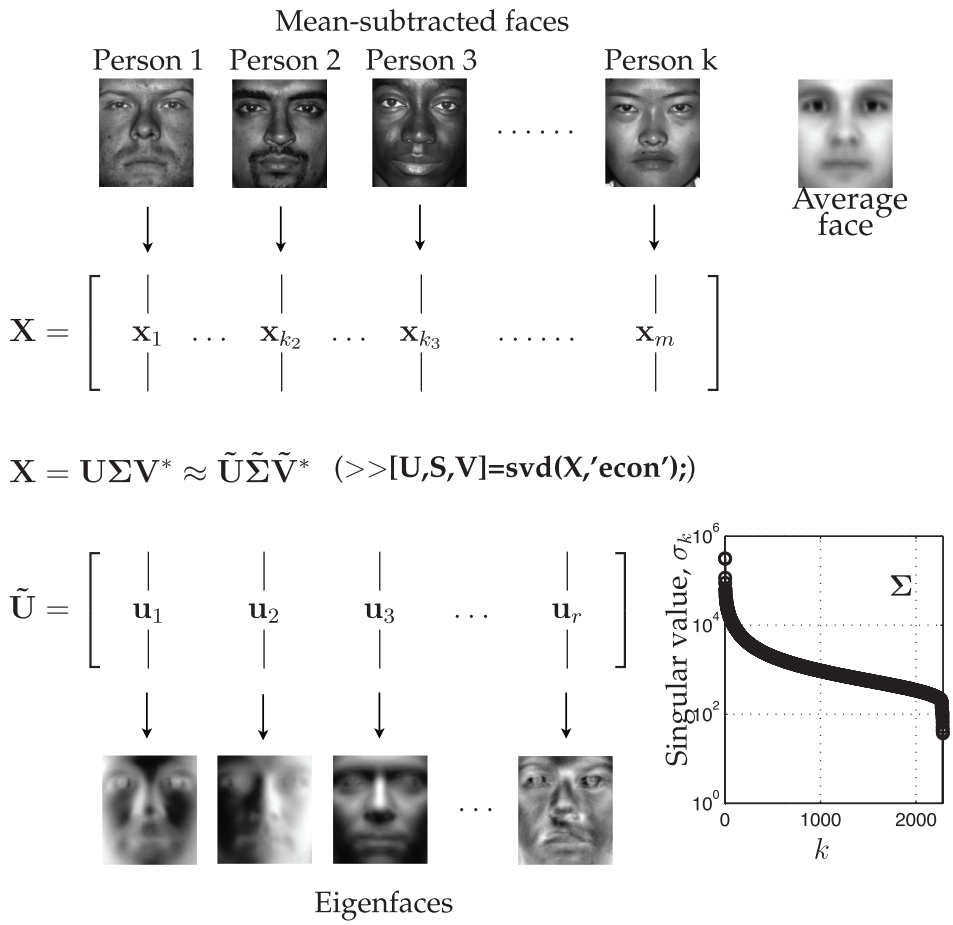

Figure 1.17 Schematic procedure to obtain eigenfaces from library of faces.

Code 1.13 Compute eigenfaces on mean-subtracted data.

```
% We use the first 36 people for training data
trainingFaces = faces(:,1:sum(nfaces(1:36)));
avgFace = mean(trainingFaces,2);  % size n*m by 1;

% Compute eigenfaces on mean-subtracted training data
X = trainingFaces-avgFace*ones(1,size(trainingFaces,2));
[U,S,V] = svd(X,'econ');

imagesc(reshape(avgFace,n,m)) % Plot avg face
imagesc(reshape(U(:,1),n,m))  % Plot first eigenface
```

Using the eigenface library, $\tilde{\mathbf{U}}$, obtained by this code, we now attempt to approximately represent an image that was not in the training data. At the beginning, we held back two individuals (the 37th and 38th people), and we now use one of their images as a test image, $\mathbf{x}_{\text{test}}$. We will see how well a rank-r SVD basis will approximate this image using the following projection:

$$\tilde{\mathbf{x}}_{\text{test}} = \tilde{\mathbf{U}}\tilde{\mathbf{U}}^*\mathbf{x}_{\text{test}}.$$

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

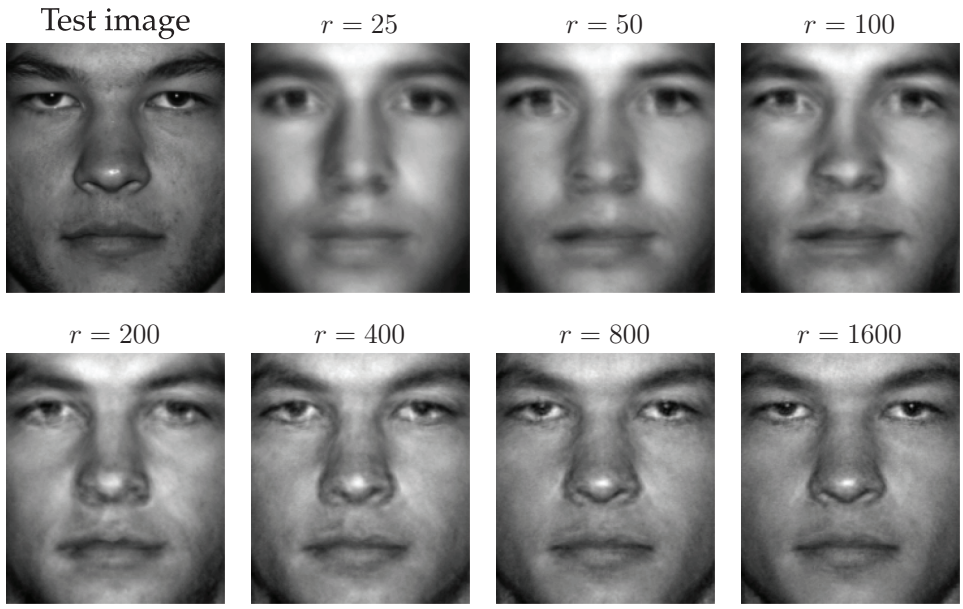

Figure 1.18 Approximate representation of test image using eigenfaces basis of various order r. Test image is not in training set.

The eigenface approximation for various values of r is shown in Fig. 1.18, as computed using Code 1.14. The approximation is relatively poor for $r \leq 200$, although for $r > 400$ it converges to a passable representation of the test image.

It is interesting to note that the eigenface space is not only useful for representing human faces, but may also be used to approximate a dog (Fig. 1.19) or a cappuccino (Fig. 1.20). This is possible because the 1600 eigenfaces span a large subspace of the 32256 dimensional image space corresponding to broad, smooth, nonlocalized spatial features, such as cheeks, forehead, mouths, etc.

Code 1.14 Approximate test-image that was omitted from training data.

```
testFaceMS = testFace - avgFace;
for r=[25 50 100 200 400 800 1600]
    reconFace = avgFace + (U(:,1:r)*(U(:,1:r)'*testFaceMS));
    imagesc(reshape(reconFace,n,m))
end
```

We further investigate the use of the eigenfaces as a coordinate system, defining an eigenface space. By projecting an image $\mathbf{x}$ onto the first r PCA modes, we obtain a set of coordinates in this space: $\tilde{\mathbf{x}} = \tilde{\mathbf{U}}^* \mathbf{x}$. Some principal components may capture the most common features shared among all human faces, while other principal components will be more useful for distinguishing between individuals. Additional principal components may capture differences in lighting angles. Figure 1.21 shows the coordinates of all 64 images of two individuals projected onto the 5th and 6th principal components, generated by Code 1.15. Images of the two individuals appear to be well-separated in these coordinates. This is the basis for image recognition and classification in Chapter 5.

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

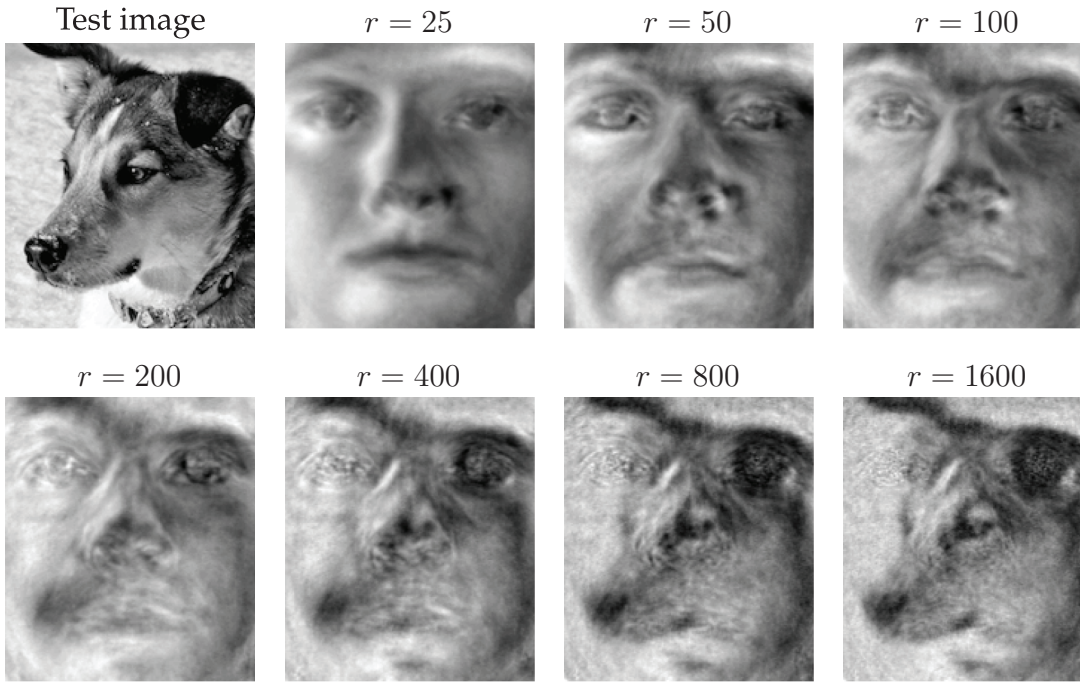

Figure 1.19 Approximate representation of an image of a dog using eigenfaces.

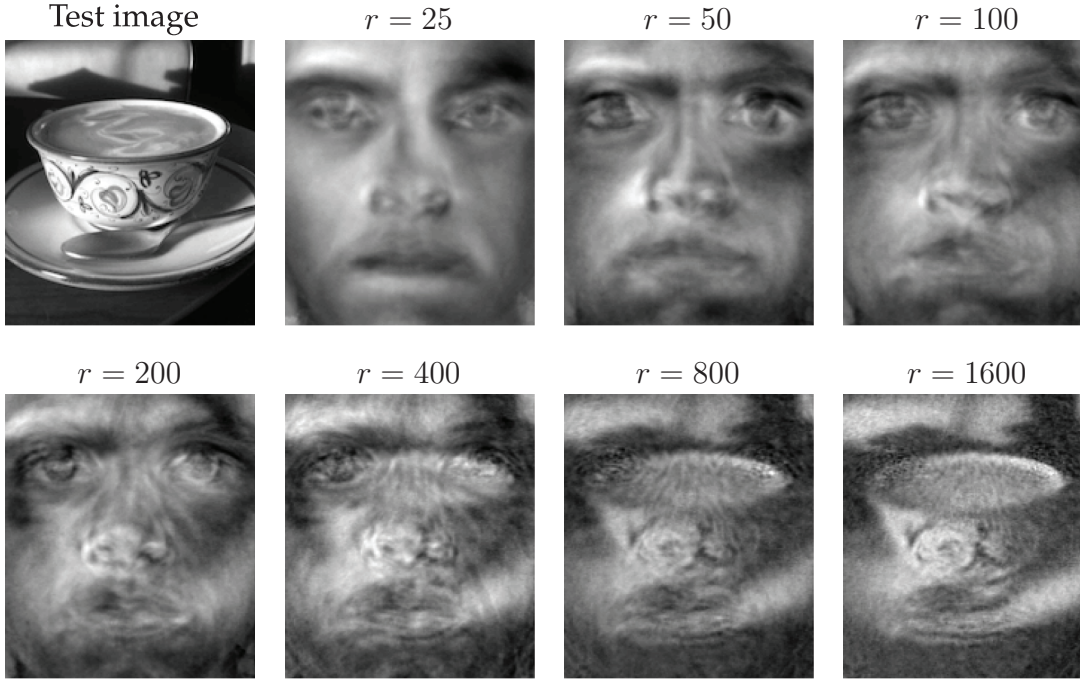

Figure 1.20 Approximate representation of a cappuccino using eigenfaces.

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

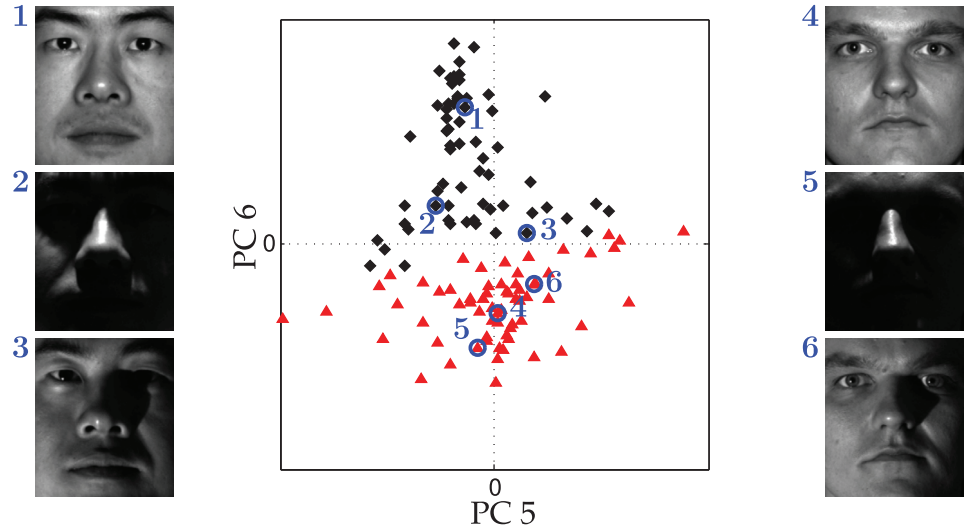

Figure 1.21 Projection of all images from two individuals onto the 5th and 6th PCA modes. Projected images of the first individual are indicated with black diamonds, and projected images of the second individual are indicated with red triangles. Three examples from each individual are circled in blue, and the corresponding image is shown.

Code 1.15 Project images for two specific people onto the 5th and 6th eigenfaces to illustrate the potential for automated classification.

```
P1num = 2;  % Person number 2
P2num = 7;  % Person number 7

P1 = faces(:,1+sum(nfaces(1:P1num-1)):sum(nfaces(1:P1num)));
P2 = faces(:,1+sum(nfaces(1:P2num-1)):sum(nfaces(1:P2num)));

P1 = P1 - avgFace*ones(1,size(P1,2));
P2 = P2 - avgFace*ones(1,size(P2,2));

PCAmodes = [5 6];    % Project onto PCA modes 5 and 6
PCACoordsP1 = U(:,PCAmodes)'*P1;
PCACoordsP2 = U(:,PCAmodes)'*P2;

plot(PCACoordsP1(1,:),PCACoordsP1(2,:),'kd')
plot(PCACoordsP2(1,:),PCACoordsP2(2,:),'r^')
```

1.7 Truncation and Alignment

Deciding how many singular values to keep, i.e. where to truncate, is one of the most important and contentious decisions when using the SVD. There are many factors, including specifications on the desired rank of the system, the magnitude of noise, and the distribution of the singular values. Often, one truncates the SVD at a rank r that captures a pre-determined amount of the variance or energy in the original data, such as 90% or 99% truncation. Although crude, this technique is commonly used. Other techniques involve identifying "elbows" or "knees" in the singular value distribution, which may denote the

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

transition from singular values that represent important patterns from those that represent noise. Truncation may be viewed as a hard threshold on singular values, where values larger than a threshold τ are kept, while remaining singular values are truncated. Recent work by Gavish and Donoho [200] provides an optimal truncation value, or hard threshold, under certain conditions, providing a principled approach to obtaining low-rank matrix approximations using the SVD.

In addition, the alignment of data significantly impacts the rank of the SVD approximation. The SVD essentially relies on a separation of variables between the columns and rows of a data matrix. In many situations, such as when analyzing traveling waves or misaligned data, this assumption breaks down, resulting in an artificial rank inflation.

Optimal Hard Threshold

A recent theoretical breakthrough determines the *optimal* hard threshold τ for singular value truncation under the assumption that a matrix has a low-rank structure contaminated with Gaussian white noise [200]. This work builds on a significant literature surrounding various techniques for hard and soft thresholding of singular values. In this section, we summarize the main results and demonstrate the thresholding on various examples. For more details, see [200].

First, we assume that the data matrix $\mathbf{X}$ is the sum of an underlying low-rank, or approximately low-rank, matrix $\mathbf{X}_{\text{true}}$ and a noise matrix $\mathbf{X}_{\text{noise}}$:

$$\mathbf{X} = \mathbf{X}_{\text{true}} + \gamma \mathbf{X}_{\text{noise}}. \tag{1.30}$$

The entries of $\mathbf{X}_{\text{noise}}$ are assumed to be independent, identically distributed (i.i.d.) Gaussian random variables with zero mean and unit variance. The magnitude of the noise is characterized by γ, which deviates from the notation in [200][7].

When the noise magnitude γ is known, there are closed-form solutions for the optimal hard threshold τ:

1. If $\mathbf{X} \in \mathbb{R}^{n \times n}$ is square, then

$$\tau = (4/\sqrt{3})\sqrt{n}\gamma. \tag{1.31}$$

2. If $\mathbf{X} \in \mathbb{R}^{n \times m}$ is rectangular and $m \ll n$, then the constant $4/\sqrt{3}$ is replaced by a function of the aspect ratio $\beta = m/n$:

$$\tau = \lambda(\beta)\sqrt{n}\gamma, \tag{1.32}$$

$$\lambda(\beta) = \left(2(\beta+1) + \frac{8\beta}{(\beta+1) + \left(\beta^2 + 14\beta + 1\right)^{1/2}}\right)^{1/2}. \tag{1.33}$$

Note that this expression reduces to (1.31) when $\beta = 1$. If $n \ll m$, then $\beta = n/m$.

When the noise magnitude γ is unknown, which is more typical in real-world applications, then it is possible to estimate the noise magnitude and scale the distribution of singular values by using σ_{med}, the *median* singular value. In this case, there is no closed-form solution for τ, and it must be approximated numerically.

[7] In [200], σ is used to denote standard deviation and y_k denotes the k^{th} singular value.

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

3. For unknown noise γ, and a rectangular matrix $\mathbf{X} \in \mathbb{R}^{n\times m}$, the optimal hard threshold is given by

$$\tau = \omega(\beta)\sigma_{\text{med}}. \tag{1.34}$$

Here, $\omega(\beta) = \lambda(\beta)/\mu_\beta$, where μ_β is the solution to the following problem:

$$\int_{(1-\beta)^2}^{\mu_\beta} \frac{\left[\left((1+\sqrt{\beta})^2 - t\right)\left(t - (1-\sqrt{\beta})^2\right)\right]^{1/2}}{2\pi t} dt = \frac{1}{2}.$$

Solutions to the expression above must be approximated numerically. Fortunately [200] has a Matlab code supplement[8] [151] to approximate μ_β.

The new method of optimal hard thresholding works remarkably well, as demonstrated on the examples below.

Example 1: Toy Problem

In the first example, shown in Fig. 1.22, we artificially construct a rank-2 matrix (Code 1.16) and we contaminate the signal with Gaussian white noise (Code 1.17). A de-noised and dimensionally reduced matrix is then obtained using the threshold from (1.31) (Code 1.18), as well as using a 90% energy truncation (Code 1.19). It is clear that the hard threshold is able to filter the noise more effectively. Plotting the singular values (Code 1.20) in Fig. 1.23, it is clear that there are two values that are above threshold.

Code 1.16 Compute the underlying low-rank signal. (Fig. 1.22 (a))

```
clear all, close all, clc

t = (-3:.01:3)';

Utrue = [cos(17*t).*exp(-t.^2) sin(11*t)];
Strue = [2 0; 0 .5];
Vtrue = [sin(5*t).*exp(-t.^2) cos(13*t)];

X = Utrue*Strue*Vtrue';
figure, imshow(X);
```

Code 1.17 Contaminate the signal with noise. (Fig. 1.22 (b))

```
sigma = 1;
Xnoisy = X+sigma*randn(size(X));
figure, imshow(Xnoisy);
```

Code 1.18 Truncate using optimal hard threshold. (Fig. 1.22 (c))

```
[U,S,V] = svd(Xnoisy);

N = size(Xnoisy,1);
cutoff = (4/sqrt(3))*sqrt(N)*sigma; % Hard threshold
r = max(find(diag(S)>cutoff)); % Keep modes w/ sig > cutoff
Xclean = U(:,1:r)*S(1:r,1:r)*V(:,1:r)';
figure, imshow(Xclean)
```

[8] http://purl.stanford.edu/vg705qn9070

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

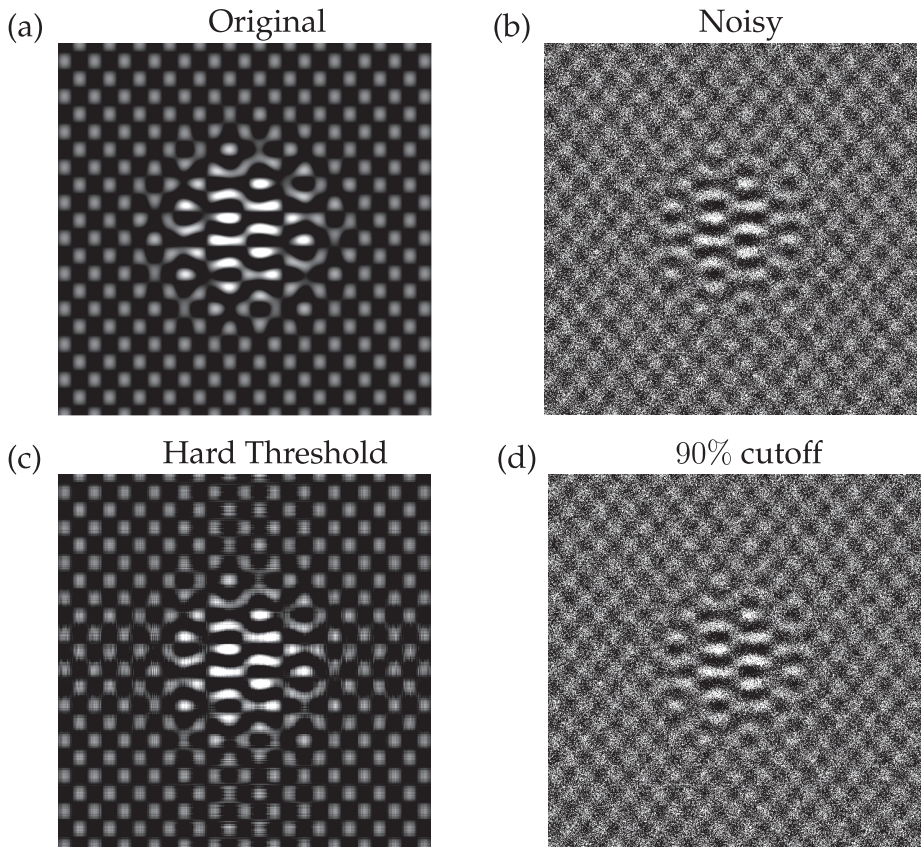

Figure 1.22 Underlying rank 2 matrix (a), matrix with noise (b), clean matrix after optimal hard threshold $(4/\sqrt{3})\sqrt{n}\sigma$ (c), and truncation based on 90% energy (d).

Code 1.19 Truncate using 90% energy criterion. (Fig. 1.22 (d))

```
cdS = cumsum(diag(S))./sum(diag(S));  % Cumulative energy
r90 = min(find(cdS>0.90));  % Find r to capture 90% energy

X90 = U(:,1:r90)*S(1:r90,1:r90)*V(:,1:r90)';
figure, imshow(X90)
```

Code 1.20 Plot singular values for hard threshold example. (Fig. 1.23)

```
semilogy(diag(S),'-ok','LineWidth',1.5), hold on, grid on
semilogy(diag(S(1:r,1:r)),'or','LineWidth',1.5)
```

Example 2: Eigenfaces

In the second example, we revisit the eigenfaces problem from Section 1.6. This provides a more typical example, since the data matrix $\mathbf{X}$ is rectangular, with aspect ratio $\beta = 3/4$, and the noise magnitude is unknown. It is also not clear that the data is contaminated with white noise. Nonetheless, the method determines a threshold τ, above which columns of $\mathbf{U}$ appear to have strong facial features, and below which columns of $\mathbf{U}$ consist mostly of noise, shown in Fig. 1.24.

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

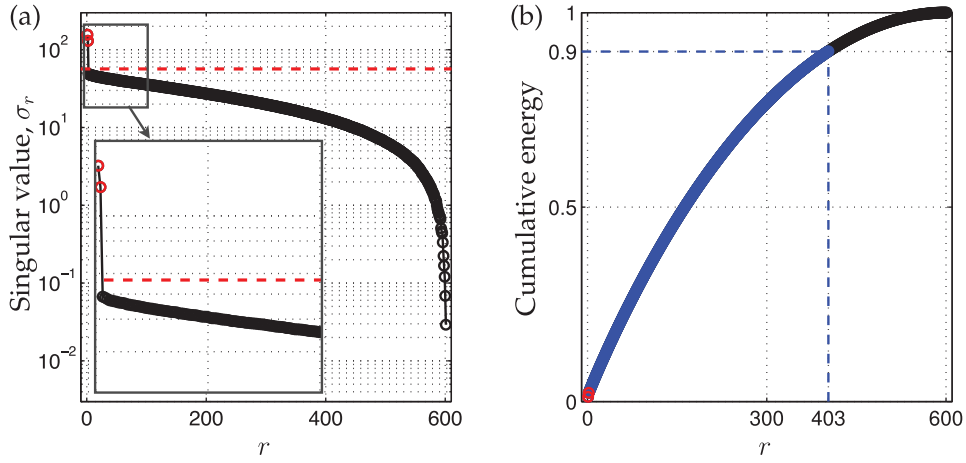

Figure 1.23 Singular values σ_r (a) and cumulative energy in first r modes (b). The optimal hard threshold $\tau = (4/\sqrt{3})\sqrt{n}\sigma$ is shown as a red dashed line (- -), and the 90% cutoff is shown as a blue dashed line (- -). For this case, $n = 600$ and $\sigma = 1$ so that the optimal cutoff is approximately $\tau = 56.6$.

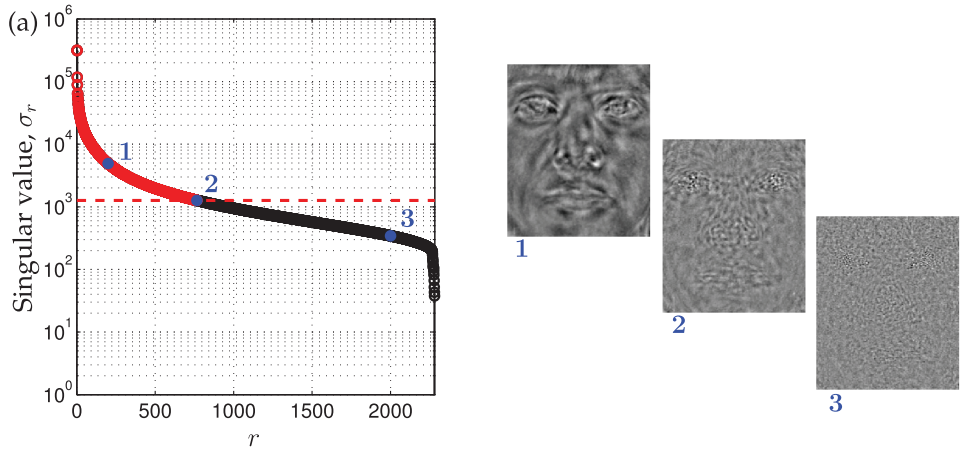

Figure 1.24 Hard thresholding for eigenfaces example.

Importance of Data Alignment

Here, we discuss common pitfalls of the SVD associated with misaligned data. The following example is designed to illustrate one of the central weaknesses of the SVD for dimensionality reduction and coherent feature extraction in data. Consider a matrix of zeros with a rectangular sub-block consisting of ones. As an image, this would look like a white rectangle placed on a black background (see Fig. 1.25 (a)). If the rectangle is perfectly aligned with the x- and y- axes of the figure, then the SVD is simple, having only one nonzero singular value σ_1 (see Fig. 1.25 (c)) and corresponding singular vectors $\mathbf{u}_1$ and $\mathbf{v}_1$ that define the width and height of the white rectangle.

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

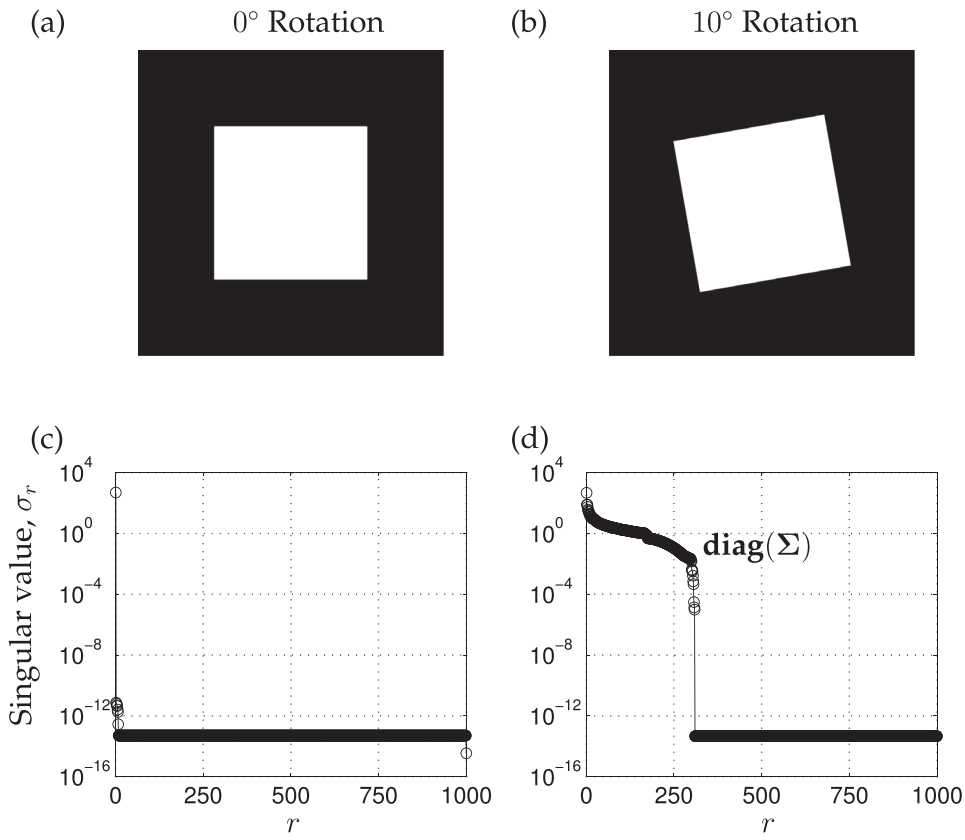

Figure 1.25 A data matrix consisting of ones with a square sub-block of zeros (a), and its SVD spectrum (c). If we rotate the image by 10°, as in (b), the SVD spectrum becomes significantly more complex (d).

When we begin to rotate the inner rectangle so that it is no longer aligned with the image axes, additional non-zero singular values begin to appear in the spectrum (see Figs. 1.25 (b,d) and 1.26).

Code 1.21 Compute the SVD for a well-aligned and rotated square (Fig. 1.25).

```
n = 1000;     % 1000 x 1000 square
X = zeros(n,n);
X(n/4:3*n/4,n/4:3:n/4) = 1;
imshow(X);

Y = imrotate(X,10,'bicubic');  % rotate 10 degrees
Y = Y - Y(1,1);
nY = size(Y,1);
startind = floor((nY-n)/2);
Xrot = Y(startind:startind+n-1, startind:startind+n-1);
imshow(Xrot);
[U,S,V] = svd(X);      % SVD well-aligned square
[U,S,V] = svd(Xrot);   % SVD rotated square
semilogy(diag(S),'-ko')
semilogy(diag(S),'-ko')
```

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

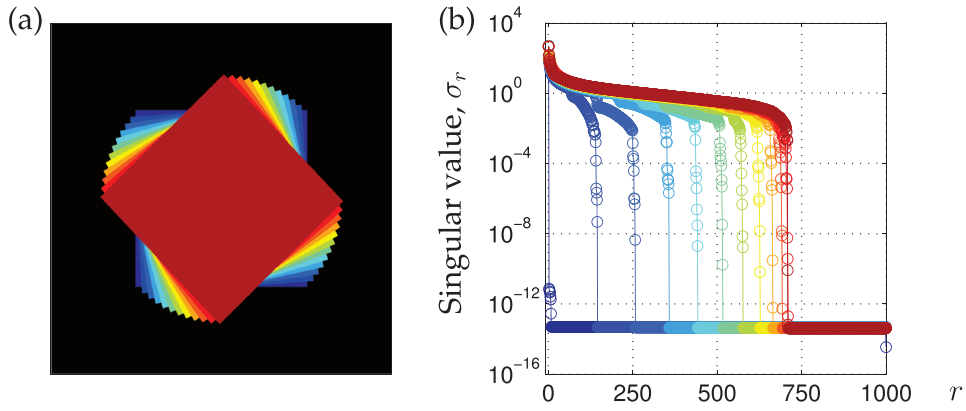

Figure 1.26 A data matrix consisting of zeros with a square sub-block of ones at various rotations (a), and the corresponding SVD spectrum, **diag(S)**, (b).

The reason that this example breaks down is that the SVD is fundamentally *geometric*, meaning that it depends on the coordinate system in which the data is represented. As we have seen earlier, the SVD is only generically invariant to unitary transformations, meaning that the transformation preserves the inner product. This fact may be viewed as both a strength and a weakness of the method. First, the dependence of SVD on the inner product is essential for the various useful geometric interpretations. Moreover, the SVD has meaningful units and dimensions. However, this makes the SVD sensitive to the alignment of the data. In fact, the SVD rank explodes when objects in the columns translate, rotate, or scale, which severely limits its use for data that has not been heavily pre-processed.

For instance, the eigenfaces example was built on a library of images that had been meticulously cropped, centered, and aligned according to a stencil. Without taking these important pre-processing steps, the features and clustering performance would be underwhelming.

The inability of the SVD to capture translations and rotations of the data is a major limitation. For example, the SVD is still the method of choice for the low-rank decomposition of data from partial differential equations (PDEs), as will be explored in Chapters 11 and 12. However, the SVD is fundamentally a data-driven separation of variables, which we know will not work for many types of PDE, for example those that exhibit traveling waves. Generalized decompositions that retain the favorable properties and are applicable to data with symmetries is a significant open challenge in the field.

Code 1.22 SVD for a square rotated through various angles (Fig. 1.26).

```
nAngles = 12;  % sweep through 12 angles, from 0:4:44
Xrot = X;
for j=2:nAngles
    Y = imrotate(X,(j-1)*4,'bicubic'); % rotate (j-1)*4
    startind = floor((size(Y,1)-n)/2);
    Xrot1 = Y(startind:startind+n-1, startind:startind+n-1);
    Xrot2 = Xrot1 - Xrot1(1,1);
    Xrot2 = Xrot2/max(Xrot2(:));
    Xrot(Xrot2>.5) = j;

    [U,S,V] = svd(Xrot1);
```

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

```
    subplot(1,2,1), imagesc(Xrot), colormap([0 0 0; cm])
    subplot(1,2,2), semilogy(diag(S),'-o','color',cm(j,:))
end
```

1.8 Randomized Singular Value Decomposition

The accurate and efficient decomposition of large data matrices is one of the cornerstones of modern computational mathematics and data science. In many cases, matrix decompositions are explicitly focused on extracting dominant low-rank structure in the matrix, as illustrated throughout the examples in this chapter. Recently, it has been shown that if a matrix $\mathbf{X}$ has low-rank structure, then there are extremely efficient matrix decomposition algorithms based on the theory of random sampling; this is closely related to the idea of sparsity and the high-dimensional geometry of sparse vectors, which will be explored in Chapter 3. These so-called *randomized* numerical methods have the potential to transform computational linear algebra, providing accurate matrix decompositions at a fraction of the cost of deterministic methods. Moreover, with increasingly vast measurements (e.g., from 4K and 8K video, internet of things, etc.), it is often the case that the *intrinsic* rank of the data does not increase appreciable, even though the dimension of the ambient measurement space grows. Thus, the computational savings of randomized methods will only become more important in the coming years and decades with the growing deluge of data.

Randomized Linear Algebra

Randomized linear algebra is a much more general concept than the treatment presented here for the SVD. In addition to the randomized SVD [464, 371], randomized algorithms have been developed for principal component analysis [454, 229], the pivoted LU decomposition [485], the pivoted QR decomposition [162], and the dynamic mode decomposition [175]. Most randomized matrix decompositions can be broken into a few common steps, as described here. There are also several excellent surveys on the topic [354, 228, 334, 177]. We assume that we are working with tall-skinny matrices, so that $n > m$, although the theory readily generalizes to short-fat matrices.

Step 0: Identify a target rank, $r < m$.

Step 1: Using random projections $\mathbf{P}$ to sample the column space, find a matrix $\mathbf{Q}$ whose columns approximate the column space of $\mathbf{X}$, i.e., so that $\mathbf{X} \approx \mathbf{QQ}^*\mathbf{X}$.

Step 2: Project $\mathbf{X}$ onto the $\mathbf{Q}$ subspace, $\mathbf{Y} = \mathbf{Q}^*\mathbf{X}$, and compute the matrix decomposition on $\mathbf{Y}$.

Step 3: Reconstruct high dimensional modes $\mathbf{U} = \mathbf{QU_Y}$ using $\mathbf{Q}$ and the modes computed from $\mathbf{Y}$.

Randomized SVD Algorithm

Over the past two decades, there have been several randomized algorithms proposed to compute a low-rank SVD, including the *Monte Carlo* SVD [190] and more robust approaches based on random projections [464, 335, 371]. These methods were improved by incorporating structured sampling matrices for faster matrix multiplications [559]. Here, we use the randomized SVD algorithm of Halko, Martinsson, and Tropp [228],

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

which combined and expanded on these previous algorithms, providing favorable error bounds. Additional analysis and numerical implementation details are found in Voronin and Martinsson [544]. A schematic of the rSVD algorithm is shown in Fig. 1.27.

Step 1: We construct a random projection matrix $\mathbf{P} \in \mathbb{R}^{m \times r}$ to sample the column space of $\mathbf{X} \in \mathbb{R}^{n \times m}$:

$$\mathbf{Z} = \mathbf{XP}. \tag{1.35}$$

The matrix $\mathbf{Z}$ may be much smaller than $\mathbf{X}$, especially for low-rank matrices with $r \ll m$. It is highly unlikely that a random projection matrix $\mathbf{P}$ will project out important components of $\mathbf{X}$, and so $\mathbf{Z}$ approximates the column space of $\mathbf{X}$ with high probability. Thus, it is possible to compute the low-rank QR decomposition of $\mathbf{Z}$ to obtain an orthonormal basis for $\mathbf{X}$:

$$\mathbf{Z} = \mathbf{QR}. \tag{1.36}$$

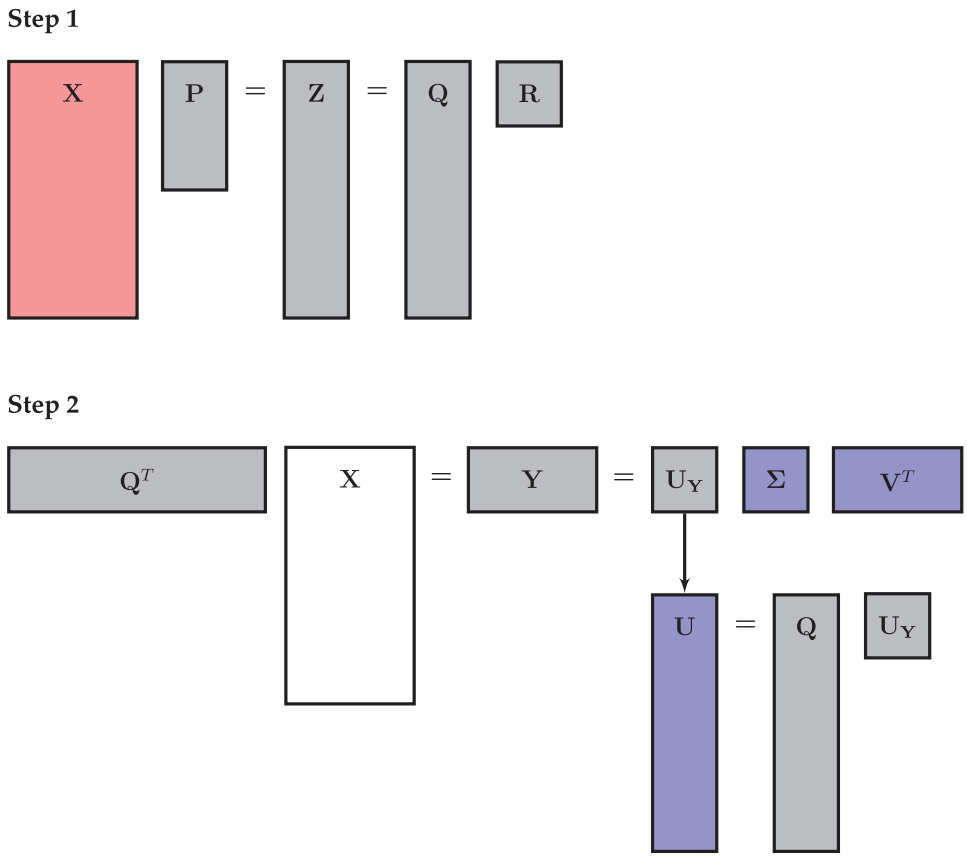

Figure 1.27 Schematic of randomized SVD algorithm. The high-dimensional data $\mathbf{X}$ is depicted in red, intermediate steps in gray, and the outputs in blue. This algorithm requires two passes over $\mathbf{X}$.

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

Step 2: With the low-rank basis $\mathbf{Q}$, we may project $\mathbf{X}$ into a smaller space:

$$\mathbf{Y} = \mathbf{Q}^*\mathbf{X}. \tag{1.37}$$

It also follows that $\mathbf{X} \approx \mathbf{QY}$, with better agreement when the singular values σ_k decay rapidly for $k > r$.

It is now possible to compute the singular value decomposition on $\mathbf{Y}$:

$$\mathbf{Y} = \mathbf{U}_\mathbf{Y}\mathbf{\Sigma}\mathbf{V}^*. \tag{1.38}$$

Because $\mathbf{Q}$ is a orthonormal and approximates the column space of $\mathbf{X}$, the matrices $\mathbf{\Sigma}$ and $\mathbf{V}$ are the same for $\mathbf{Y}$ and $\mathbf{X}$, as discussed in Section 1.3.

Step 3: Finally, it is possible to reconstruct the high-dimensional left singular vectors $\mathbf{U}$ using $\mathbf{U}_\mathbf{Y}$ and $\mathbf{Q}$:

$$\mathbf{U} = \mathbf{QU}_\mathbf{Y}. \tag{1.39}$$

Oversampling

Most matrices $\mathbf{X}$ do not have an exact low-rank structure, given by r modes. Instead, there are nonzero singular values σ_k for $k > r$, and the sketch $\mathbf{Z}$ will not exactly span the column space of $\mathbf{X}$. In general, increasing the number of columns in $\mathbf{P}$ from r to $r + p$, significantly improves results, even with p adding around 5 or 10 columns [370]. This is known as *oversampling*, and increasing p decreases the variance of the singular value spectrum of the sketched matrix.

Power Iterations

A second challenge in using randomized algorithms is when the singular value spectrum decays slowly, so that the remaining truncated singular values contain significant variance in the data $\mathbf{X}$. In this case, it is possible to preprocess $\mathbf{X}$ through q *power iterations* [454, 228, 224] to create a new matrix $\mathbf{X}^{(q)}$ with a more rapid singular value decay:

$$\mathbf{X}^{(q)} = \left(\mathbf{XX}^*\right)^q \mathbf{X}. \tag{1.40}$$

Power iterations dramatically improve the quality of the randomized decomposition, as the singular value spectrum of $\mathbf{X}^{(q)}$ decays more rapidly:

$$\mathbf{X}^{(q)} = \mathbf{U}\mathbf{\Sigma}^{2q-1}\mathbf{V}^*. \tag{1.41}$$

However, power iterations are expensive, requiring q additional passes through the data $\mathbf{X}$. In some extreme examples, the data in $\mathbf{X}$ may be stored in a distributed architecture, so that every additional pass adds considerable expense.

Guaranteed Error Bounds

One of the most important properties of the randomized SVD is the existence of tunable error bounds, that are explicit functions of the singular value spectrum, the desired rank r, the oversampling parameter p and the number of power iterations q. The best attainable error bound for a deterministic algorithm is:

$$\|\mathbf{X} - \mathbf{QY}\|_2 \geq \sigma_{r+1}(\mathbf{X}). \tag{1.42}$$

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

In other words, the approximation with the best possible rank-r subspace $\mathbf{Q}$ will have error greater than or equal to the next truncated singular value of $\mathbf{X}$. For randomized methods, it is possible to bound the *expectation* of the error:

$$\mathbb{E}\left(\|\mathbf{X}-\mathbf{QY}\|_2\right) \leq \left(1+\sqrt{\frac{r}{p-1}}+\frac{e\sqrt{r+p}}{p}\sqrt{m-r}\right)^{\frac{1}{2q+1}} \sigma_{k+1}(\mathbf{X}), \tag{1.43}$$

where e is Euler's number.

Choice of random matrix $\mathbf{P}$

There are several suitable choices of the random matrix $\mathbf{P}$. Gaussian random projections (e.g., the elements of $\mathbf{P}$ are i.i.d. Gaussian random variables) are frequently used because of favorable mathematical properties and the richness of information extracted in the sketch $\mathbf{Z}$. In particular, it is very unlikely that a Gaussian random matrix $\mathbf{P}$ will be chosen *badly* so as to project out important information in $\mathbf{X}$. However, Gaussian projections are expensive to generate, store, and compute. Uniform random matrices are also frequently used, and have similar limitations. There are several alternatives, such as Rademacher matrices, where the entries can be $+1$ or -1 with equal probability [532]. Structured random projection matrices may provide efficient sketches, reducing computational costs to $\mathcal{O}(nm\log(r))$ [559]. Yet another choice is a sparse projection matrix $\mathbf{P}$, which improves storage and computation, but at the cost of including less information in the sketch. In the extreme case, when even a single pass over the matrix $\mathbf{X}$ is prohibitively expensive, the matrix $\mathbf{P}$ may be chosen as random columns of the $m \times m$ identity matrix, so that it randomly selects columns of $\mathbf{X}$ for the sketch $\mathbf{Z}$. This is the fastest option, but should be used with caution, as information may be lost if the structure of $\mathbf{X}$ is highly localized in a subset of columns, which may be lost by column sampling.

Example of Randomized SVD

To demonstrate the randomized SVD algorithm, we will decompose a high-resolution image. This particular implementation is only for illustrative purposes, as it has not been optimized for speed, data transfer, or accuracy. In practical applications, care should be taken [228, 177].

Code 1.23 computes the randomized SVD of a matrix $\mathbf{X}$, and Code 1.24 uses this function to obtain a rank-400 approximation to a high-resolution image, shown in Fig. 1.28.

Code 1.23 Randomized SVD algorithm.

```
function [U,S,V] = rsvd(X,r,q,p);

% Step 1: Sample column space of X with P matrix
ny = size(X,2);
P = randn(ny,r+p);
Z = X*P;
for k=1:q
    Z = X*(X'*Z);
end
[Q,R] = qr(Z,0);

% Step 2: Compute SVD on projected Y=Q'*X;
Y = Q'*X;
```

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

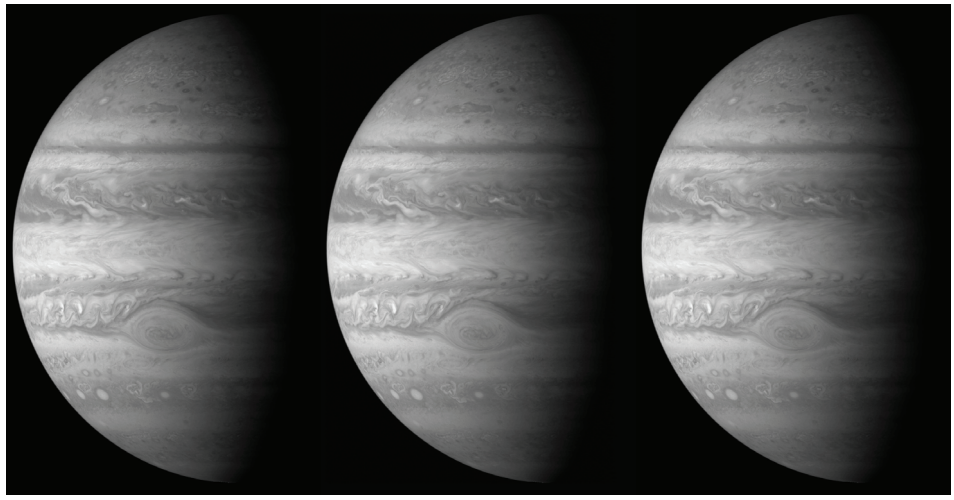

Figure 1.28 Original high-resolution (left) and rank-400 approximations from the SVD (middle) and rSVD (right).

```
[UY,S,V] = svd(Y,'econ');
U = Q*UY;
```

Code 1.24 Compute the randomized SVD of high-resolution image.

```
clear all, close all, clc
A=imread('jupiter.jpg');
X=double(rgb2gray(A));
[U,S,V] = svd(X,'econ');     % Deterministic SVD

r = 400; % Target rank
q = 1;   % Power iterations
p = 5;   % Oversampling parameter
[rU,rS,rV] = rsvd(X,r,q,p); % Randomized SVD

%% Reconstruction
XSVD = U(:,1:r)*S(1:r,1:r)*V(:,1:r)';      % SVD approx.
errSVD = norm(X-XSVD,2)/norm(X,2);
XrSVD = rU(:,1:r)*rS(1:r,1:r)*rV(:,1:r)'; % rSVD approx.
errrSVD = norm(X-XrSVD,2)/norm(X,2);
```

1.9 Tensor Decompositions and *N*-Way Data Arrays

Low-rank decompositions can be generalized beyond matrices. This is important as the SVD requires that disparate types of data be flattened into a single vector in order to evaluate correlated structures. For instance, different time snapshots (columns) of a matrix may include measurements as diverse as temperature, pressure, concentration of a substance, etc. Additionally, there may be categorical data. Vectorizing this data generally does not make sense. Ultimately, what is desired is to preserve the various data structures and types in their own, independent directions. Matrices can be generalized to N-way arrays, or

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

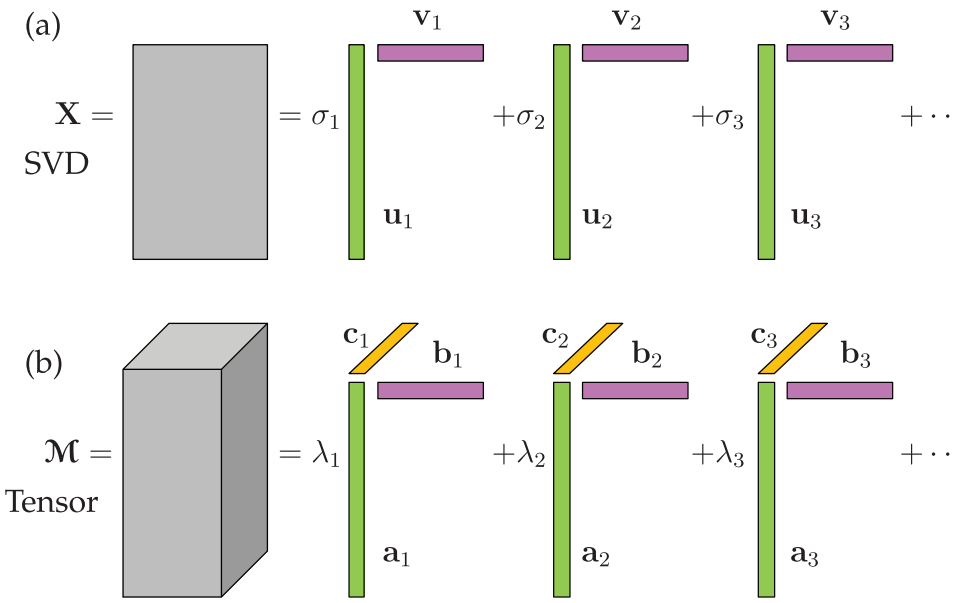

Figure 1.29 Comparison of the SVD and Tensor decomposition frameworks. Both methods produce an approximation to the original data matrix by sums of outer products. Specifically, the tensor decomposition generalizes the concept of the SVD to N-way arrays of data without having to flatten (vectorize) the data.

tensors, where the data is more appropriately arranged without forcing a data-flattening process.

The construction of data tensors requires that we revisit the notation associated with tensor addition, multiplication, and inner products [299]. We denote the rth column of a matrix $\mathbf{A}$ by $\mathbf{a}_r$. Given matrices $\mathbf{A} \in \mathbb{R}^{I\times K}$ and $\mathbf{B} \in \mathbb{R}^{J\times K}$, their Khatri-Rao product is denoted by $\mathbf{A} \odot \mathbf{B}$ and is defined to be the $IJ \times K$ matrix of column-wise Kronecker products, namely

$$\mathbf{A} \odot \mathbf{B} = \begin{pmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 & \cdots & \mathbf{a}_K \otimes \mathbf{b}_K \end{pmatrix}.$$

For an N-way tensor $\mathcal{A}$ of size $I_1 \times I_2 \times \cdots \times I_N$, we denote its $\mathbf{i} = (i_1, i_2, \ldots, i_N)$ entry by $a_{\mathbf{i}}$.

The inner product between two N-way tensors $\mathcal{A}$ and $\mathcal{B}$ of compatible dimensions is given by

$$\langle \mathcal{A}, \mathcal{B} \rangle = \sum_{\mathbf{i}} a_{\mathbf{i}} b_{\mathbf{i}}.$$

The Frobenius norm of a tensor $\mathcal{A}$, denoted by $\|\mathcal{A}\|_{\mathrm{F}}$, is the square root of the inner product of $\mathcal{A}$ with itself, namely $\|\mathcal{A}\|_{\mathrm{F}} = \sqrt{\langle \mathcal{A}, \mathcal{A} \rangle}$. Finally, the mode-$n$ matricization or unfolding of a tensor $\mathcal{A}$ is denoted by $\mathbf{mA}_{(n)}$.

Let $\mathcal{M}$ represent an N-way data tensor of size $I_1 \times I_2 \times \cdots \times I_N$. We are interested in an R-component CANDECOMP/PARAFAC (CP) [124, 235, 299] factor model

$$\mathcal{M} = \sum_{r=1}^{R} \lambda_r \, \mathbf{ma}_r^{(1)} \circ \cdots \circ \mathbf{ma}_r^{(N)}, \tag{1.44}$$

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

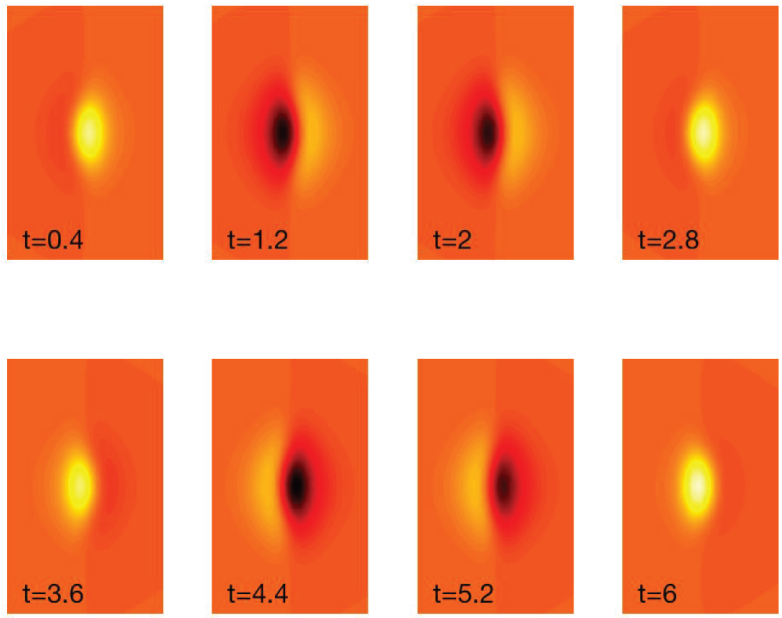

Figure 1.30 Example N-way array data set created from the function (1.45). The data matrix is $\mathbf{A} \in \mathbb{R}^{121\times 101\times 315}$. A CP tensor decomposition can be used to extract the two underlying structures that produced the data.

where $\circ$ represents outer product and $\mathbf{ma}_r^{(n)}$ represents the rth column of the *factor matrix* $\mathbf{mA}^{(n)}$ of size $I_n \times R$. The CP decomposition refers to CANDECOMP/PARAFAC which stand for *parallel factors analysis* (PARAFAC) and *canonical decomposition* (CANDECOMP) respectively. We refer to each summand as a *component*. Assuming each factor matrix has been column-normalized to have unit Euclidean length, we refer to the λ_r's as *weights*. We will use the shorthand notation where $\lambda = (\lambda_1, \ldots, \lambda_R)^{\mathsf{T}}$ [25]. A tensor that has a CP decomposition is sometimes referred to as a Kruskal tensor.

For the rest of this chapter, we consider a 3-way CP tensor decomposition (See Fig. 1.29) where two modes index state variation and the third mode indexes time variation:

$$\mathcal{M} = \sum_{r=1}^{R} \lambda_r \, \mathbf{A}_r \circ \mathbf{B}_r \circ \mathbf{C}_r.$$

Let $\mathbf{A} \in \mathbb{R}^{I_1\times R}$ and $\mathbf{B} \in \mathbb{R}^{I_2\times R}$ denote the factor matrices corresponding to the two state modes and $\mathbf{C} \in \mathbb{R}^{I_3\times R}$ denote the factor matrix corresponding to the time mode. This 3-way decomposition is compared to the SVD in Fig. 1.29.

To illustrate the tensor decomposition, we use the MATLAB N-way toolbox developed by Rasmus Bro and coworkers [84, 15] which is available on the Mathworks file exchange. This simple to use package provides a variety of tools to extract tensor decompositions and evaluate the factor models generated. In the specific example considered here, we generate data from a spatio-temporal function (See Fig. 1.30)

$$F(x, y, t) = \exp(-x^2 - 0.5y^2)\cos(2t) + \operatorname{sech}(x)\tanh(x)\exp(-0.2y^2)\sin(t). \quad (1.45)$$

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

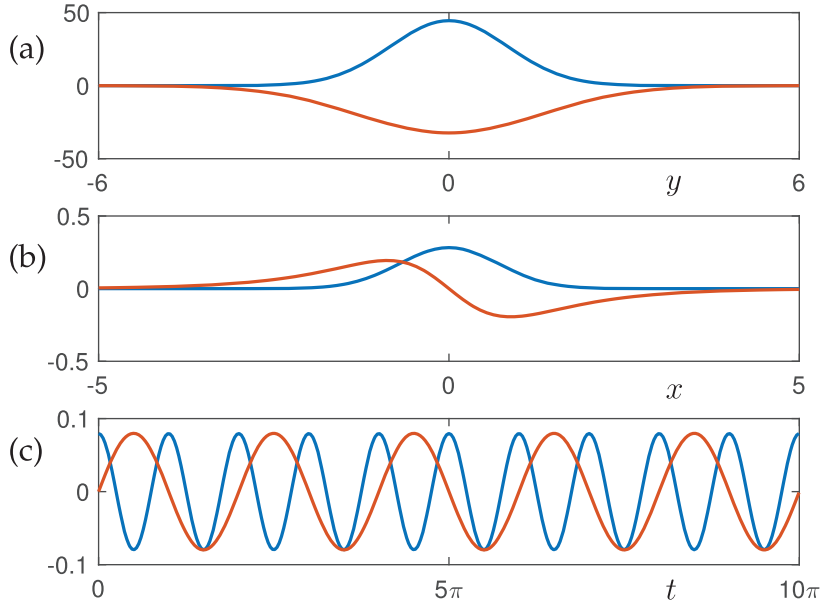

Figure 1.31 3-way tensor decomposition of the function (1.45) discretized so that the data matrix is $\mathbf{A} \in \mathbb{R}^{121\times101\times315}$. A CP tensor decomposition can be used to extract the two underlying structures that produced the data. The first factor is in blue, the second factor is in red. The three distinct directions of the data (parallel factors) are illustrated in (a) the y direction, (b) the x direction, and (c) the time t.

This model has two spatial modes with two distinct temporal frequencies, thus a two factor model should be sufficient to extract the underlying spatial and temporal modes. To construct this function in MATLAB, the following code is used.

Code 1.25 Creating tensor data.

```
x=-5:0.1:5; y=-6:0.1:6; t=0:0.1:10*pi;
[X,Y,T]=meshgrid(x,y,t);
A=exp(-(X.^2+0.5*Y.^2)).*(cos(2*T))+ ...
    (sech(X).*tanh(X).*exp(-0.2*Y.^2)).*sin(T);
```

Note that the **meshgrid** command is capable of generating N-way arrays. Indeed, MATLAB has no difficulties specifying higher-dimensional arrays and tensors. Specifically, one can easily generate N-way data matrices with arbitrary dimensions. The command $\mathbf{A} = \mathbf{randn}(\mathbf{10}, \mathbf{10}, \mathbf{10}, \mathbf{10}, \mathbf{10})$ generates a 5-way hypercube with random values in each of the five directions of the array.

Figure 1.30 shows eight snapshots of the function (1.45) discretized with the code above. The N-way array data generated from the MATLAB code produces $\mathbf{A} \in \mathbb{R}^{121\times101\times315}$, which is of total dimension 10^6. The CP tensor decomposition can be used to extract a two factor model for this 3-way array, thus producing two vectors in each direction of space x, space y, and time t.

The N-way toolbox provides a simple architecture for performing tensor decompositions. The PARAFAC command structure can easily take the input function (1.45) which is discretized in the code above and provide a two-factor model. The following code produces the output as **model**.

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

Code 1.26 Two factor tensor model.

```
model=parafac(A,2);
[A1,A2,A3]=fac2let(model);
subplot(3,1,1), plot(y,A1,'Linewidth',[2])
subplot(3,1,2), plot(x,A2,'Linewidth',[2])
subplot(3,1,3), plot(t,A3,'Linewidth',[2])
```

Note that in this code, the **fac2let** command turns the factors in the model into their component matrices. Further note that the **meshgrid** arrangement of the data is different from **parafac** since the x and y directions are switched.

Figure 1.31 shows the results of the N-way tensor decomposition for the prescribed two factor model. Specifically, the two vectors along each of the three directions of the array are illustrated. For this example, the exact answer is known since the data was constructed from the rank-2 model (1.45). The first set of two modes (along the original y direction) are Gaussian as prescribed. The second set of two modes (along the original x direction) include a Gaussian for the first function, and the anti-symmetric $\text{sech}(x)\tanh(x)$ for the second function. The third set of two modes correspond to the time dynamics of the two functions: $\cos(2t)$ and $\sin(t)$, respectively. Thus, the two factor model produced by the CP tensor decomposition returns the expected, low-rank functions that produced the high-dimensional data matrix $\mathbf{A}$.

Recent theoretical and computational advances in N-way decompositions are opening up the potential for tensor decompositions in many fields. For N large, such decompositions can be computationally intractable due to the size of the data. Indeed, even in the simple example illustrated in Figs. 1.30 and 1.31, there are 10^6 data points. Ultimately, the CP tensor decomposition does not scale well with additional data dimensions. However, randomized techniques are helping yield tractable computations even for large data sets [158, 175]. As with the SVD, randomized methods exploit the underlying low-rank structure of the data in order to produce an accurate approximation through the sum of rank-one outer products. Additionally, tensor decompositions can be combined with constraints on the form of the parallel factors in order to produce more easily interpretable results [348]. This gives a framework for producing interpretable and scalable computations of N-way data arrays.

Suggested Reading

Texts

(1) **Matrix computations**, by G. H. Golub and C. F. Van Loan, 2012 [214].

Papers and reviews

(1) **Calculating the singular values and pseudo-inverse of a matrix**, by G. H. Golub and W. Kahan, *Journal of the Society for Industrial & Applied Mathematics, Series B: Numerical Analysis*, 1965 [212].

(2) **A low-dimensional procedure for the characterization of human faces**, by L. Sirovich and M. Kirby, *Journal of the Optical Society of America A*, 1987 [491].

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press

(3) **Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions**, by N. Halko, P.-G. Martinsson, and J. A. Tropp, *SIAM Review*, 2011 [230].

(4) **A randomized algorithm for the decomposition of matrices**, by P.-G. Martinsson, V. Rokhlin, and M. Tygert, *Applied and Computational Harmonic Analysis*, 2011 [371].

(5) **The optimal hard threshold for singular values is** $4/\sqrt{3}$, by M. Gavish and D. L. Donoho, *IEEE Transactions on Information Theory*, 2014 [200].

https://doi.org/10.1017/9781108380690.002 Published online by Cambridge University Press