Contents

1 Singular Value Decomposition **3**

1.1 Overview (Big picture) . . . 3

1.1.1 Intuition and applications . . . 3

1.1.2 Definition . . . 4

1.1.3 Historical perspective . . . 5

1.1.4 Uses in this book and assumptions of the reader . . . 5

1.2 Matrix approximation . . . 6

1.2.1 Truncation . . . 7

1.2.2 Example: Image compression . . . 8

1.2.3 Aside on compressibility and the vastness of image space . . . 11

1.3 Mathematical properties and manipulations . . . 12

1.3.1 Interpretation as dominant correlations . . . 12

1.3.2 Geometric interpretation . . . 13

1.3.3 Invariance of the SVD to unitary transformations . . . 15

1.3.4 Method of snapshots . . . 16

1.4 Pseudo-inverse, least-squares, and regression . . . 17

1.4.1 One-dimensional linear regression . . . 18

1.4.2 Multilinear regression . . . 20

1.4.3 Caution . . . 23

1.5 Principal components analysis (PCA) . . . 24

1.5.1 Computation . . . 24

1.5.2 Example: Noisy Gaussian data . . . 25

1.5.3 Example: Ovarian cancer data . . . 27

1.6 Eigenfaces example . . . 30

1.7 Truncation and hard thresholding . . . 37

1.7.1 Optimal hard threshold . . . 37

1.8 Additional mathematical properties . . . 43

1.8.1 Important theorems . . . 43

1.8.2 Four fundamental subspaces . . . 43

1.9 Caveats and extensions . . . 45

1.9.1 Poorly aligned data . . . 45

1.9.2 Advanced extensions to the SVD . . . 48

1.10 Computing the SVD . . . 49

1.10.1 SVD in various languages . . . 49

Chapter 1

Singular Value Decomposition (SVD) and Principal Components Analysis (PCA)

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 discussed in this book. In particular, the SVD provides a numerically stable matrix decomposition that can be used for a variety of purposes. We will use the SVD to obtain low-rank approximations to matrices and to perform pseudo-inverses of non-square matrices to find the least-squares and minimum norm solutions of a matrix system of equations $\mathbf{A}\mathbf{x} = \mathbf{b}$. Another important use of the SVD is in the principal components analysis (PCA), whereby a large, high-dimensional data set is decomposed into its most statistically descriptive factors. SVD/PCA has been applied to a tremendous variety of problems in science and engineering, making it the most generally useful computational tool after the fast Fourier transform (FFT).

1.1 Overview (Big picture)

Here we introduce the singular value decomposition (SVD) and give an overview of some of the intuitive motivating examples. Detailed mathematical properties are discussed in the following sections.

1.1.1 Intuition and applications

In this chapter, we will develop an intuition for how to apply the SVD by demonstrating its use on a number of motivating examples. These methods will provide a foundation for many other techniques developed in this book, including classification methods in Chapter 2, the proper orthogonal decomposition (POD) in Chapter 3, and the dynamic mode decomposition (DMD) in Chapter 4.

High-dimensionality is a common challenge in processing data from large complex systems. These systems may involve large measured data sets including audio, image, or video data. The data may also be generated from a complex physical system, such as neural recordings from the mammalian cortex, or fluid velocity measurements from a simulation or experiment. In any case, it is observed that most data from naturally occurring systems exhibits dominant patterns of activity, which may be characterized by a low-dimensional attractor or manifold [26, 25].

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 turbulent wake behind a vehicle or the Earth's atmosphere also provide compelling examples 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 the dominant patterns underlying a high-dimensional system, providing a low-rank approximation to high-dimensional data. This technique is *data-driven* in that patterns are discovered purely from the data, without the addition of any expert knowledge or intuition. The SVD may be thought of as a numerically stable computation that provides a hierarchical representation of the data in terms of a new coordinate system defined by dominant correlations within the data.

The SVD has many powerful applications beyond dimensionality reduction of high-dimensional data. It will be useful in computing the pseudo-inverse of non-square matrices, providing solutions to underdetermined or overdetermined matrix equations that are either minimum norm solutions, or solutions that minimize the sum-squared error. We will also use the SVD in a principled approach to de-noising data sets. The SVD is also 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.

1.1.2 Definition

Generally, we are interested in analyzing a large data set $\mathbf{X}$:

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

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 each point in a discretized simulation or at each measurement location in a wind-tunnel experiment.

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 in the case of fluid systems. 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 $\mathbf{X} \in \mathbb{C}^{n\times m}$:

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

where $\mathbf{U} \in \mathbb{C}^{n\times n}$ and $\mathbf{V} \in \mathbb{C}^{m\times m}$ are *unitary* matrices[1] and $\mathbf{\Sigma} \in \mathbb{C}^{n\times m}$ is a matrix with non-negative 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 extremely powerful.

The matrix $\mathbf{\Sigma}$ has at most m non-zero elements on the diagonal, and may therefore 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 *reduced* 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 and reduced SVD are shown in Fig. 1.1. Here $\hat{\mathbf{U}}^\perp$ is complementary to $\hat{\mathbf{U}}$.

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 the are ordered from largest to smallest.

In Matlab, the computing the SVD is straightforward:

```
>>[U,S,V] = svd(X);  % Singular Value Decomposition
```

For non-square matrices $\mathbf{X}$, the reduced SVD may be computed more efficiently using:

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

1.1.3 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 [50], 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 seminar computational work of Golub and collaborators [20, 19]. In addition, there are many excellent chapters on the SVD in modern books [52, 2, 33].

1.1.4 Uses in this book and assumptions of the reader

The SVD is the basis for many related techniques in dimensionality reduction used to obtain reduced order models (ROMs). These methods indlude principal components analysis (PCA) in statistics [40, 27, 28], the Karhunen–Loève transform (KLT) [30, 36], empirical orthogonal functions (EOFs) in climate [37], the proper orthogonal decomposition (POD)

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

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

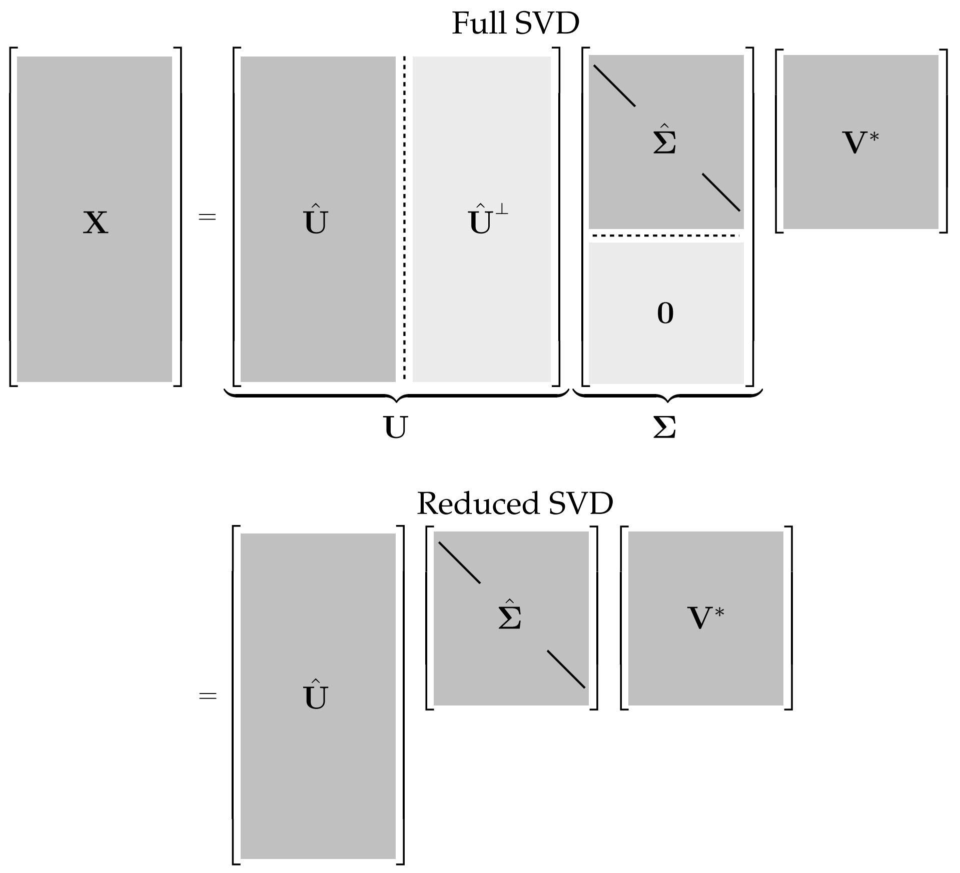


Figure 1.1: Schematic of matrices involved in the full SVD and the reduced SVD.

in fluid dynamics [25], and canonical correlation analysis (CCA) [10]. 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 [17].

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 [39].

For this chapter, we assume that the reader is familiar with linear algebra with some experience in computation and numerics. For review, there are number of excellent books on numerical linear algebra, with discussions on the SVD [52, 2, 33].

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 $\mathbf{X}$ [47]. Schmidt's approximation theorem was rediscovered by Eckart and Young [13], and is sometimes referred to as the Eckart-Young theorem.

Theorem 1 (Eckart-Young [13]) *The optimal rank-r approximation to $\mathbf{X}$, in an L_2 sense, is given by the rank-r SVD truncation $\tilde{\mathbf{X}}$:*

$$\underset{\tilde{\mathbf{X}}}{\operatorname{argmin}} \|\mathbf{X} - \tilde{\mathbf{X}}\|_2 = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*. \tag{1.4}$$

Here, $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$ denote the leading r columns of $\mathbf{U}$ and $\mathbf{V}$, and $\tilde{\mathbf{\Sigma}}$ contains the leading $r \times r$ sub-block of $\mathbf{\Sigma}$.

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 $\tilde{\mathbf{X}}$.

This is an extremely 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 $\tilde{\mathbf{X}}$ 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*, and the SVD basis provides a hierarchy of modes that characterize the observed attractor, on which we may project a low-dimensional dynamical system.

1.2.1 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 row rank, then some of the singular values in $\hat{\mathbf{\Sigma}}$ may be zero, and the truncated SVD may still be exact. However, in general, for truncation values r that are smaller than the row 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 be the rank of $\mathbf{X}$, so that $\hat{\mathbf{\Sigma}}_{\text{rem}}$ is zero and $\tilde{\mathbf{\Sigma}}$ contains non-zero singular values, then we may refer to $\tilde{\mathbf{U}}, \tilde{\mathbf{\Sigma}}, \tilde{\mathbf{V}}$ as $\hat{\mathbf{U}}_1, \hat{\mathbf{\Sigma}}_1, \hat{\mathbf{V}}_1$.

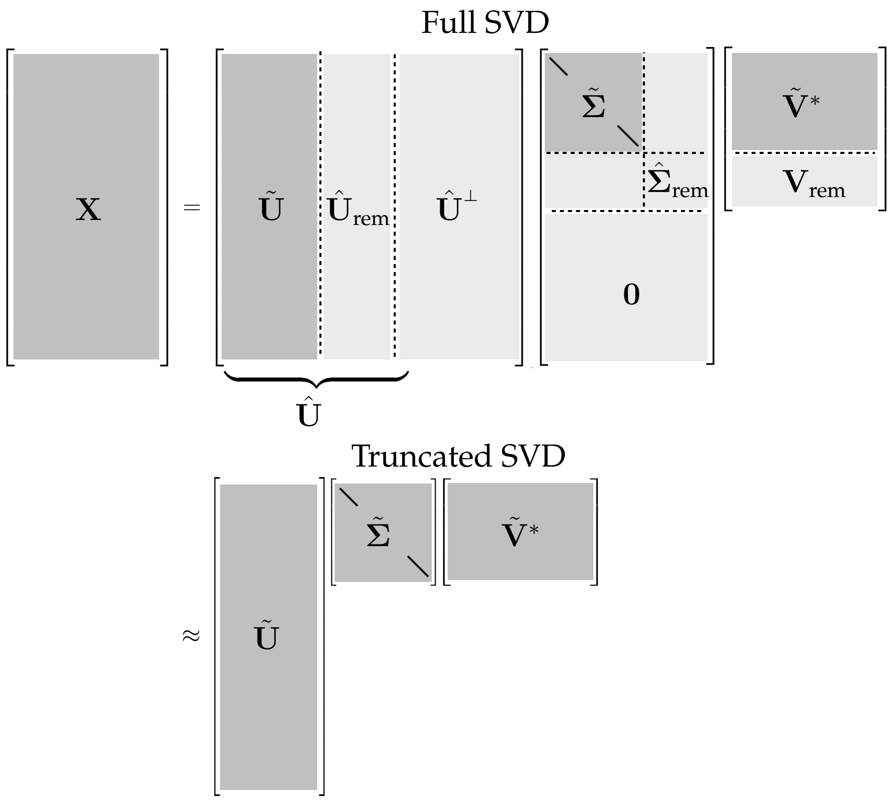

Full SVD
$\mathbf{X}$
$=$
$\tilde{\mathbf{U}}$
$\hat{\mathbf{U}}_{\text{rem}}$
$\hat{\mathbf{U}}^{\perp}$
$\hat{\mathbf{U}}$
$\tilde{\mathbf{\Sigma}}$
$\hat{\mathbf{\Sigma}}_{\text{rem}}$
$\mathbf{0}$
$\tilde{\mathbf{V}}^*$
$\mathbf{V}_{\text{rem}}$
Truncated SVD
$\approx$
$\tilde{\mathbf{U}}$
$\tilde{\mathbf{\Sigma}}$
$\tilde{\mathbf{V}}^*$

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.

1.2.2 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 direction, respectively[3]. Depending on the basis of representation (pixel-space, Fourier frequency domain, SVD transform coordinates), images may have very compact approximations.

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.

First, we load the image of the dog and compute the SVD, as in Code 1.1. Next, we compute the approximate matrix using the truncated SVD for various ranks ($r = 5, 20,$

[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.

and 100) in Code 1.2. Finally, we plot the singular values and cumulative energy in Fig. 1.4 using Code 1.3.

Code 1.1: Load image and take the SVD.

```
A=imread('../DATA/dog.jpg');
X=double(rgb2gray(A));   % convert RBG to gray, 256 bit to double.
nx = size(X,1); ny = size(X,2);
```

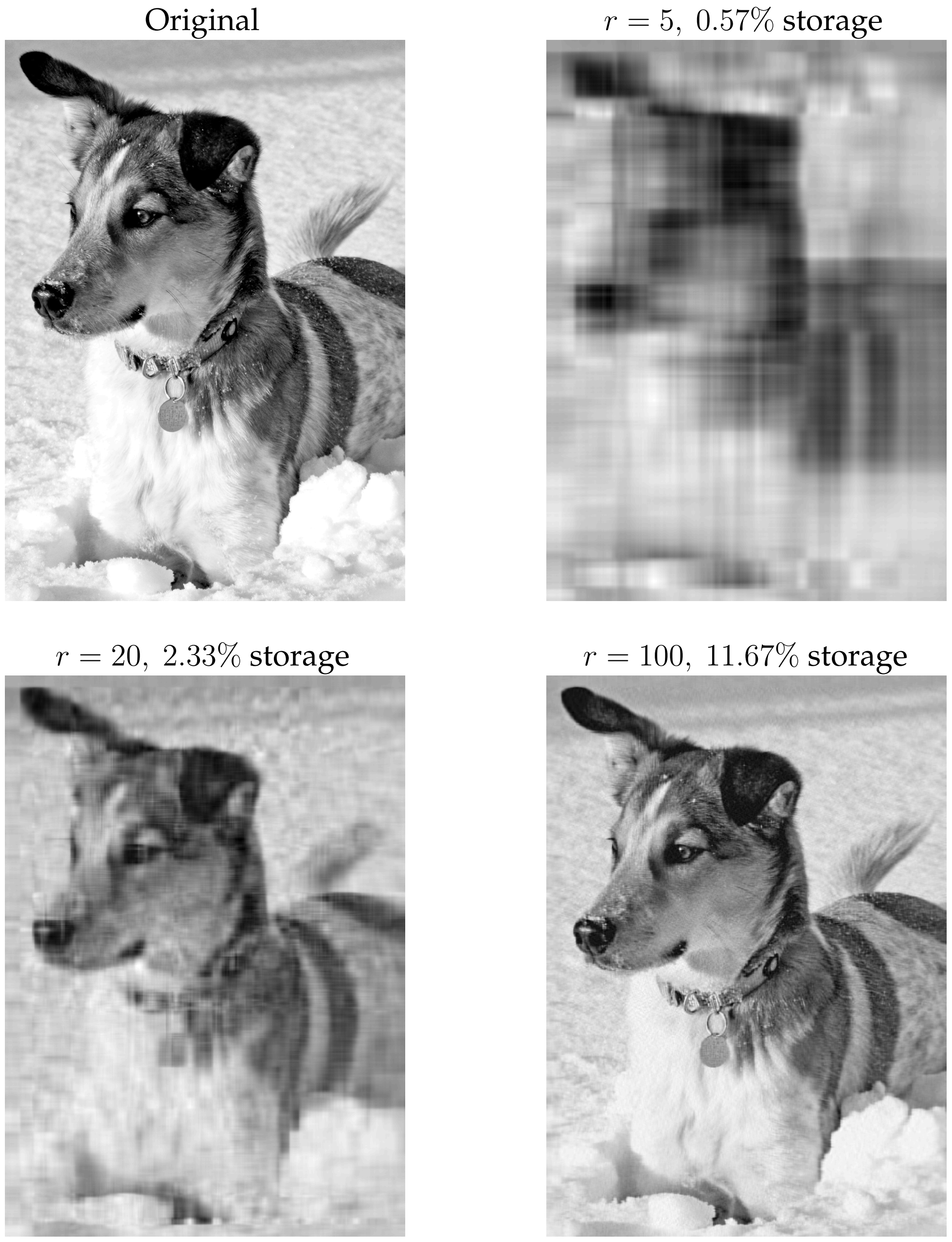


Figure 1.3: Image compression of Mordecai the snow dog, truncating the SVD at various ranks r. Original image resolution is 2000×1500. Generated by Codes 1.1 & 1.2.

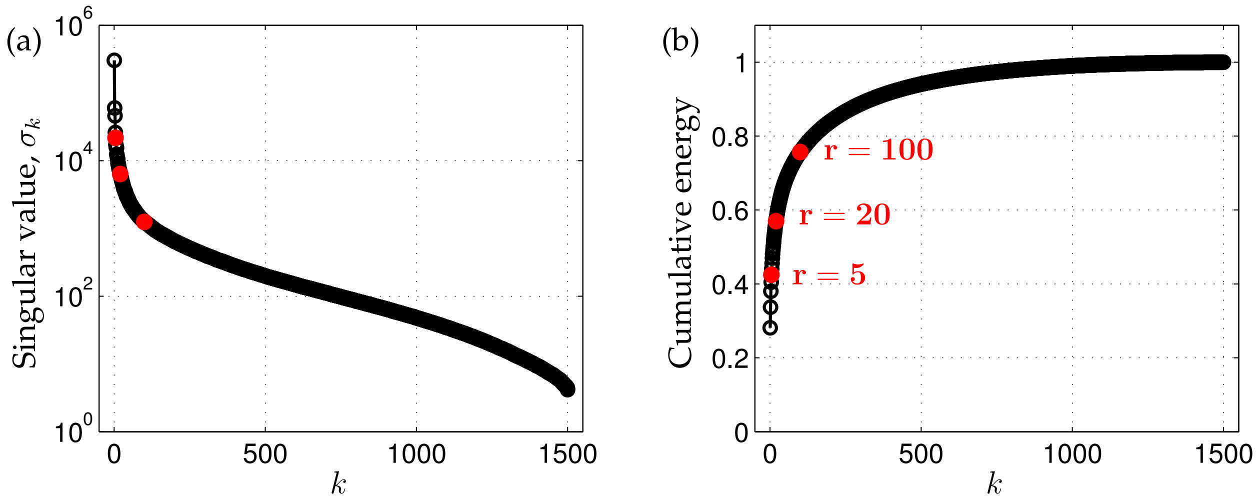


Figure 1.4: Singular values σ_r (a) and cumulative energy contained in the first r modes (b). Generated by Code 1.3.

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

figure, subplot(2,2,1)
imagesc(X), axis off, colormap gray
title('Original')
```

Code 1.2: Approximate image using truncated SVD for $r = 5, 20$ and 100. (Fig. 1.3)

```
plotind = 2;
for r=[5 20 100];   % truncation value
    Xapprox = U(:,1:r)*S(1:r,1:r)*V(:,1:r)';   % approximate image
    subplot(2,2,plotind), plotind = plotind + 1;
    imagesc(Xapprox), axis off
    title(['r=',num2str(r,'%d'),', ',num2str(100*r*(nx+ny)/(nx*ny),'
       %2.2f'),'% storage']);
end
set(gcf,'Position',[100 100 550 400])
```

Code 1.3: Plot the singular values and cumulative energy. (Fig. 1.4)

```
figure, subplot(1,2,1)
semilogy(diag(S),'k','LineWidth',1.2), grid on
xlabel('r')
ylabel('Singular value, \sigma_r')
xlim([-50 1550])
subplot(1,2,2)
plot(cumsum(diag(S))/sum(diag(S)),'k','LineWidth',1.2), grid on
xlabel('r')
ylabel('Cumulative Energy')
xlim([-50 1550])
ylim([0 1.1])
set(gcf,'Position',[100 100 550 240])
```

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 dominant correlations in the data $\mathbf{X}$. The relationship between the SVD and correlations in the data will be explored more in Sec. 1.5 on Principal Components Analysis.

1.3.1 Interpretation as dominant correlations

The SVD is closely related to an eigenvalue problem involving the correlation matrices $\mathbf{X}\mathbf{X}^*$ and $\mathbf{X}^*\mathbf{X}$, shown in Fig. 1.5. If we plug in Eq. 1.3 to the row-wise correlation matrix $\mathbf{X}\mathbf{X}^*$ and the column-wise correlation matrix $\mathbf{X}^*\mathbf{X}$, we find:

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

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

Therefore, the matrices $\mathbf{U}$, $\mathbf{\Sigma}$, and $\mathbf{V}$ are solutions to the following eigenvalue problems:

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

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

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 $\mathbf{X}^*\mathbf{X}$. Therefore, columns of $\mathbf{U}$ are ordered in terms of the vectors that most describe correlation among columns of $\mathbf{X}$, and likewise with $\mathbf{V}$ and rows of $\mathbf{X}$.

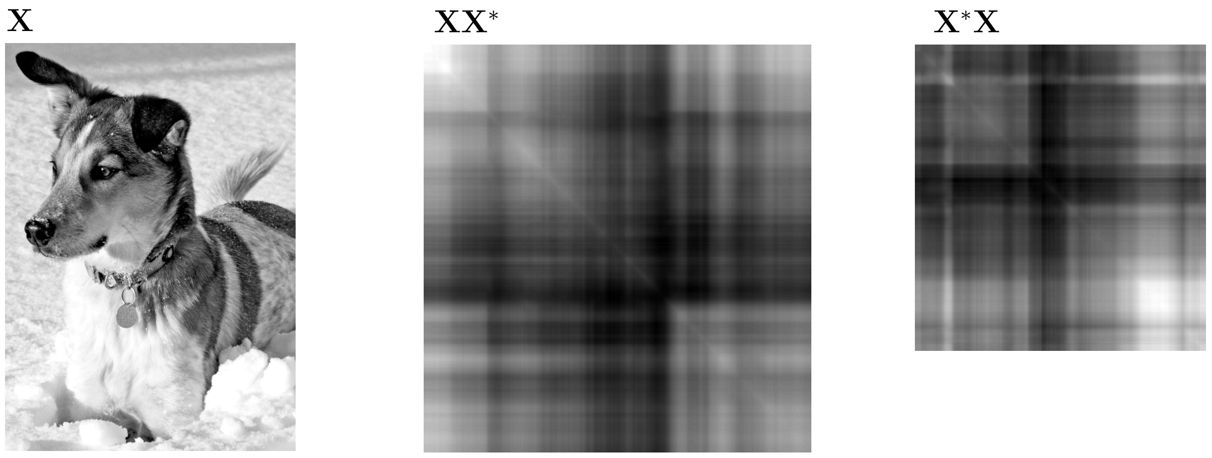


Figure 1.5: Correlation matrices $\mathbf{X}\mathbf{X}^*$ and $\mathbf{X}^*\mathbf{X}$ for a matrix $\mathbf{X}$ obtained from an image of Mordecai the snow dog. Note that both correlation matrices are symmetric.

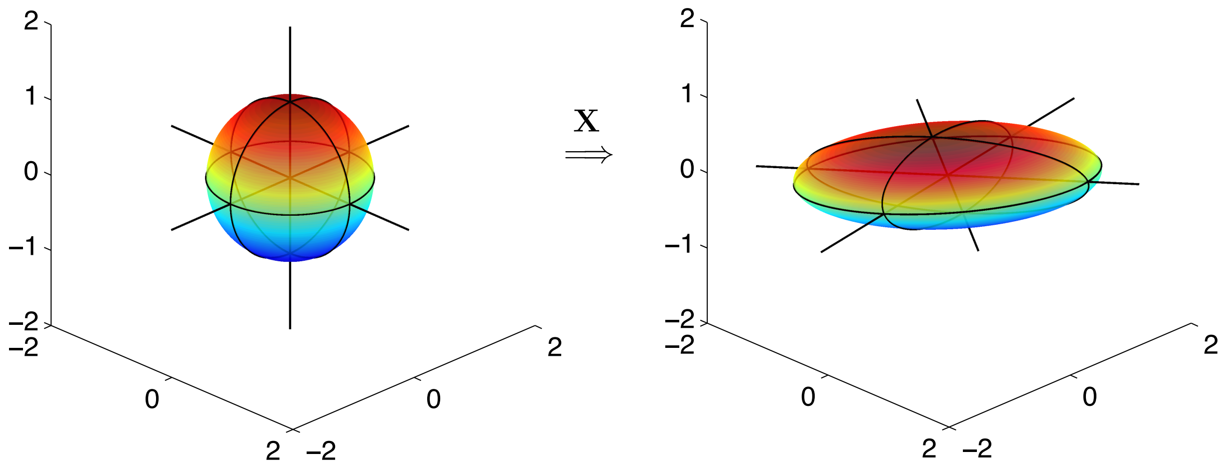


Figure 1.6: Geometric illustration of the SVD as a mapping from an $(n-1)$-sphere, $S^{n-1} \subset \mathbb{R}^n$, to an ellipsoid in $\mathbb{R}^m$.

1.3.2 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 $\mathbf{U}$ contains dominant spatial patterns, while $\mathbf{V}$ contains dominant temporal patterns.

One property that makes the SVD particularly useful is the fact that both $\mathbf{U}$ and $\mathbf{V}$ are *unitary* matrices, meaning that $\mathbf{U}\mathbf{U}^* = \mathbf{U}^*\mathbf{U} = \mathbb{I}_{n\times n}$ and $\mathbf{V}\mathbf{V}^* = \mathbf{V}^*\mathbf{V} = \mathbb{I}_{m\times m}$. This means that solving a system of equations involving $\mathbf{U}$ or $\mathbf{V}$ as simple as multiplication by the transpose, which scales as $\mathcal{O}(n^2)$, as opposed to traditional methods for generic methods, which scale as $\mathcal{O}(n^3)$. As noted in [5], the SVD is intimately connected to the spectral properties of compact self-adjoint operators, namely $\mathbf{X}\mathbf{X}^*$ and $\mathbf{X}^*\mathbf{X}$.

The SVD of $\mathbf{X}$ may be interpreted geometrically based on how a hyper sphere, 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. 1.6 for a sphere in $\mathbb{R}^3$ and a mapping $\mathbf{X}$ with three non-zero singular values. Because the mapping through $\mathbf{X}$ is linear, knowing how it maps the unit sphere determines how all other vectors will map.

For the specific case shown in Fig. 1.6, 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:

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

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 is important. 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 unitary, and is therefore the matrix $\mathbf{U}$ in the SVD of $\mathbf{X}$. The matrix $\mathbf{V}$ is the identity.

Code 1.4: Construct rotation matrices.

```
clear all, close all, clc
theta = [pi/15; -pi/9; -pi/20];
Sigma = diag([3; 1; 0.5]);              % scale x, then y, then 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 1.5: Plot sphere.

```
subplot(1,2,1), hold on
[x,y,z] = sphere(25);
h1=surf(x,y,z);
set(h1,'FaceAlpha',.7)
colormap jet, lighting phong,  axis equal
axis([-2 2 -2 2 -2 2]), view([45 26])
```

Code 1.6: Map sphere through **X** and plot resulting ellipsoid.

```
for i=1:size(x,1)
    for j=1:size(x,2)
        vec = [x(i,j); y(i,j); z(i,j)];
        vecR = X*vec;
        xR(i,j) = vecR(1);
        yR(i,j) = vecR(2);
        zR(i,j) = vecR(3);
    end
end

subplot(1,2,2), hold on
h2=surf(xR,yR,zR,z);  % using the z-position of sphere for color
set(h2,'FaceAlpha',.7)
colormap jet, lighting phong,  axis equal
axis([-2 2 -2 2 -2 2]), view([45 26])
set(gcf,'Position',[100 100 800 350])
```

1.3.3 Invariance of the SVD to unitary transformations

An extremely 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)[4] $\mathcal{F}$ 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 will be important in later chapters when we use compressive measurements to reconstruct SVD modes that are sparse in some transform basis. The advent of compressive sampling, and particularly the theory surrounding restricted isometry maps, makes this invariance particularly interesting.

The invariance of SVD to unitary transformations is simple to show using the method of snapshots. For consistency, we denote a left unitary transformation by $\mathbf{C}$, so that $\mathbf{Y} = \mathbf{C}\mathbf{X}$, and a right unitary transformation by $\mathbf{P}$, so that $\mathbf{Y} = \mathbf{X}\mathbf{P}^*$.

Left unitary transformations

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

$$\mathbf{Y}^*\mathbf{Y} = \mathbf{X}^*\mathbf{C}^*\mathbf{C}\mathbf{X} = \mathbf{X}^*\mathbf{X}. \tag{1.9}$$

The projected data has the same eigendecomposition, resulting in the same $\mathbf{V}_\mathbf{X}$ and $\mathbf{\Sigma}_\mathbf{X}$. Using the method of snapshots to reconstruct $\mathbf{U}_\mathbf{Y}$, we find

$$\mathbf{U}_\mathbf{Y} = \mathbf{Y}\mathbf{V}_\mathbf{X}\mathbf{\Sigma}_\mathbf{X}^{-1} = \mathbf{C}\mathbf{X}\mathbf{V}_\mathbf{X}\mathbf{\Sigma}_\mathbf{X}^{-1} = \mathbf{C}\mathbf{U}_\mathbf{X}. \tag{1.10}$$

Thus, $\mathbf{U}_\mathbf{Y} = \mathbf{C}\mathbf{U}_\mathbf{X}$, $\mathbf{\Sigma}_\mathbf{Y} = \mathbf{\Sigma}_\mathbf{X}$, and $\mathbf{V}_\mathbf{Y} = \mathbf{V}_\mathbf{X}$. Finally, we may write the SVD of $\mathbf{Y}$:

$$\mathbf{Y} = \mathbf{C}\mathbf{X} = \mathbf{C}\mathbf{U}_\mathbf{X}\mathbf{\Sigma}_\mathbf{X}\mathbf{V}_\mathbf{X}^*. \tag{1.11}$$

Right unitary transformations

Similarly, we consider a right unitary transformation $\mathbf{Y} = \mathbf{X}\mathbf{P}^*$. Computing the correlation matrix $\mathbf{Y}^*\mathbf{Y}$:

$$\mathbf{Y}^*\mathbf{Y} = \mathbf{P}\mathbf{X}^*\mathbf{X}\mathbf{P}^* = \mathbf{P}\mathbf{V}_\mathbf{X}\mathbf{\Sigma}_\mathbf{X}^*\mathbf{V}_\mathbf{X}^*\mathbf{P}^*, \tag{1.12}$$

with the following eigendecomposition

$$\mathbf{Y}^*\mathbf{Y}\mathbf{P}\mathbf{V}_\mathbf{X} = \mathbf{P}\mathbf{V}_\mathbf{X}\mathbf{\Sigma}_\mathbf{X}^2. \tag{1.13}$$

Thus, $\mathbf{V}_\mathbf{Y} = \mathbf{P}\mathbf{V}_\mathbf{X}$ and $\mathbf{\Sigma}_\mathbf{Y} = \mathbf{\Sigma}_\mathbf{X}$. We may use the method of snapshots to reconstruct $\mathbf{U}_\mathbf{Y}$:

$$\mathbf{U}_\mathbf{Y} = \mathbf{Y}\mathbf{P}\mathbf{V}_\mathbf{X}\mathbf{\Sigma}_\mathbf{X}^{-1} = \mathbf{X}\mathbf{V}_\mathbf{X}\mathbf{\Sigma}_\mathbf{X}^{-1} = \mathbf{U}_\mathbf{X}. \tag{1.14}$$

Thus, $\mathbf{U}_\mathbf{Y} = \mathbf{U}_\mathbf{X}$, and we may write the SVD of $\mathbf{Y}$ as:

$$\mathbf{Y} = \mathbf{X}\mathbf{P}^* = \mathbf{U}_\mathbf{X}\mathbf{\Sigma}_\mathbf{X}\mathbf{V}_\mathbf{X}^*\mathbf{P}^*. \tag{1.15}$$

[4]See Sec. **??**.

1.3.4 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 $\mathbf{X}\mathbf{X}^*$ has a trillion elements. 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 [48].

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 Eq. (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, $\hat{\mathbf{\Sigma}}_1$, and the corresponding columns $\hat{\mathbf{V}}_1$ of $\mathbf{V}$. From these matrices, it is then possible to approximate the first r columns of $\mathbf{U}$ as follows:

$$\hat{\mathbf{U}}_1 = \mathbf{X}\hat{\mathbf{V}}_1\hat{\mathbf{\Sigma}}_1^{-1}. \tag{1.16}$$

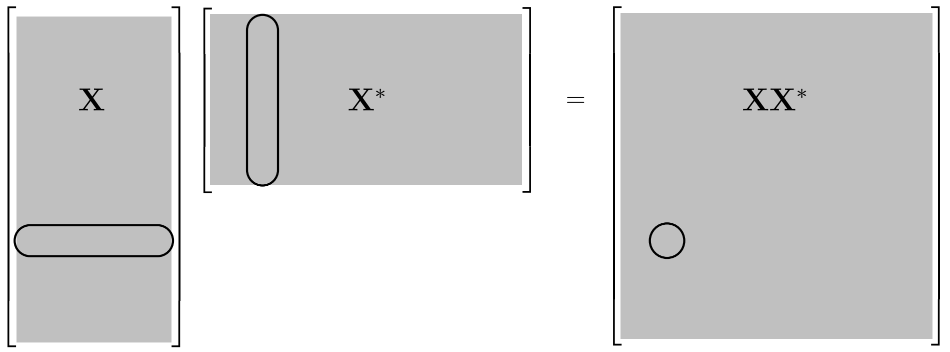


Figure 1.7: Correlation matrix $\mathbf{X}\mathbf{X}^*$ is formed by taking the inner-product of rows of $\mathbf{X}$.

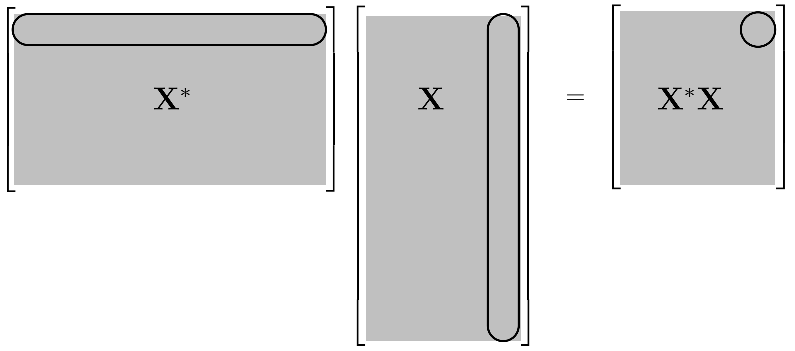


Figure 1.8: Correlation matrix $\mathbf{X}^*\mathbf{X}$ is formed by taking the inner-product of columns of $\mathbf{X}$.

1.4 Pseudo-inverse, least-squares, and regression

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

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{1.17}$$

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 less 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[5]. 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 there are guaranteed to be 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 Eq. (1.17) is determined by the four fundamental subspaces of $\mathbf{A}$, discussed more in Section 1.8.2. 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*.

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* $\mathbf{A}\mathbf{x} = \mathbf{b}$ *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 the column space of* $\mathbf{A}$*. These properties of random matrices will play a prominent role in later chapters on compressive sensing.*

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

The SVD is the technique of choice for these important problems. First, if we substitute $\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^*$ in for $\mathbf{A}$, we can "invert" each of the matrices $\mathbf{U}, \boldsymbol{\Sigma}$, and $\mathbf{V}^*$ in turn, resulting

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

in the Moore-Penrose *left pseudo-inverse* [41, 42, 45, 55] $\mathbf{A}^\dagger$ of $\mathbf{A}$:

$$\mathbf{A}^\dagger \triangleq \mathbf{V}\hat{\mathbf{\Sigma}}^{-1}\hat{\mathbf{U}}^* \quad \Longrightarrow \quad \mathbf{A}^\dagger\mathbf{A} = \mathbb{I}_{m\times m}. \tag{1.18}$$

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

$$\mathbf{A}^\dagger\mathbf{A}\tilde{\mathbf{x}} = \mathbf{A}^\dagger\mathbf{b} \quad \Longrightarrow \quad \tilde{\mathbf{x}} = \mathbf{V}\hat{\mathbf{\Sigma}}^{-1}\hat{\mathbf{U}}^*\mathbf{b}. \tag{1.19}$$

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

$$\begin{aligned} \mathbf{A}\tilde{\mathbf{x}} &= \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\mathbf{V}^*\mathbf{V}\hat{\mathbf{\Sigma}}^{-1}\hat{\mathbf{U}}^*\mathbf{b} && (1.20a)\\ &= \hat{\mathbf{U}}\hat{\mathbf{U}}^*\mathbf{b}. && (1.20b) \end{aligned}$$

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

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

1.4.1 One-dimensional linear regression

Regression is an important statistical tool to relate variables to one another based on data [38]. 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 1.7. We assume that the data is linearly related, as in Eq. (1.17), and we use the pseudo-inverse to find the least-square solution $\tilde{x}$ below (blue dashed line), shown in Code 1.8:

$$\begin{aligned} \begin{bmatrix} | \\ \mathbf{b} \\ | \end{bmatrix} &= \begin{bmatrix} | \\ \mathbf{a} \\ | \end{bmatrix} \tilde{x} = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\mathbf{V}^*\tilde{x}. && (1.21a)\\ \Longrightarrow \quad \tilde{x} &= \mathbf{V}\hat{\mathbf{\Sigma}}^{-1}\hat{\mathbf{U}}^*\mathbf{b}. && (1.21b) \end{aligned}$$

In Eq. (1.21b), $\hat{\mathbf{\Sigma}} = \|\mathbf{a}\|_2$, $\mathbf{V} = 1$, and $\hat{\mathbf{U}} = \mathbf{a}/\|\mathbf{a}\|_2$. Taking the left pseudo-inverse:

$$x = \frac{\mathbf{a}^*\mathbf{b}}{\|\mathbf{a}\|_2^2}. \tag{1.22}$$

This makes physical sense, if we think of x as the value that best maps our collection of $\mathbf{a}$ values to the corresponding values in $\mathbf{b}$. Then, the best single value x is obtained by taking the dot product of $\mathbf{b}$ with the normalized $\mathbf{a}$ direction. We then add a second normalization factor $\|\mathbf{a}\|_2$ because the $\mathbf{a}$ in Eq. (1.21a) is not normalized.

Note that strange things happen if you use row vectors instead of column vectors in Eq. (1.21) above. 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.

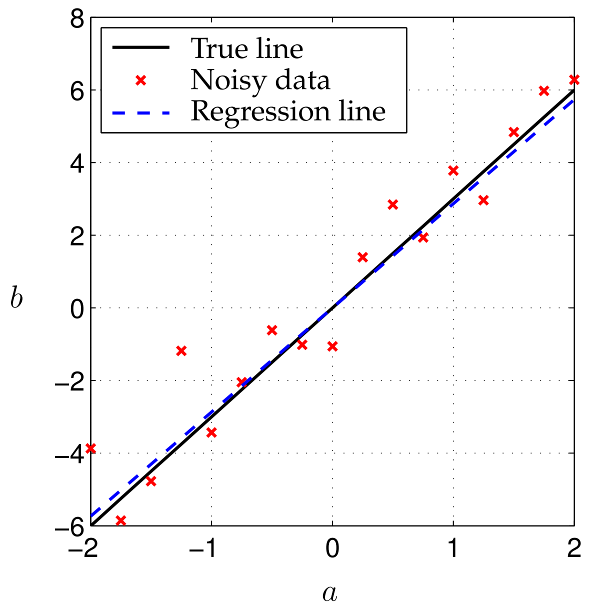


Figure 1.9: Illustration of linear regression using noisy data.

Code 1.7: Generate noisy data for Fig. 1.9.

```
clear all, close all, clc

x = 3;                                   % True slope
a = [-2:.25:2]';
b = a*x + 1*randn(size(a));              % Add noise
plot(a,x*a,'k','LineWidth',1.5)          % True relationship
hold on
plot(a,b,'rx','LineWidth',1.5)           % Noisy measurements
```

Code 1.8: 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--','LineWidth',1.5)    % Plot fit
l1=legend('True line','Noisy data','Regression line');
```

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 1.9: Alternative formulations of least-squares in Matlab.

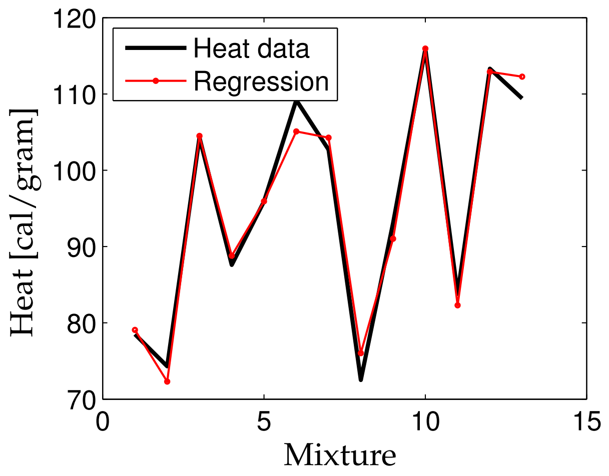
```
xtilde1 = V*inv(S)*U'*b
xtilde2 = pinv(a)*b
xtilde3 = regress(b,a)
```

1.4.2 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 Eq. (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 a formula, in the weighting $\mathbf{x}$, that would relate the proportions of the four basic ingredients to the heat generation.

It is possible to find the minimum error solution using the SVD, as shown in Code 1.10. Alternatives, using `regress` and `pinv`, are also explored.



	Ingredient	Regression
1	Tricalcium aluminate	2.1930
2	Tricalcium silicate	1.1533
3	Tetracalcium alumiferrite	0.7585
4	Beta-dicalcium silicate	0.4863

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

Code 1.10: Multilinear regression for cement heat data.

```
clear all, close all, clc

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','LineWidth',2);  hold on            % Plot data
plot(A*x,'r-o','LineWidth',1.,'MarkerSize',2); % Plot regression
l1 = legend('Heat data','Regression')

%% Alternative 1  (regress)
x = regress(b,A);

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

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 [23]. This data is available from the UCI Machine Learning Repository [3].

The various attributes that are correlated with house price are given in Table 1.1. 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 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.

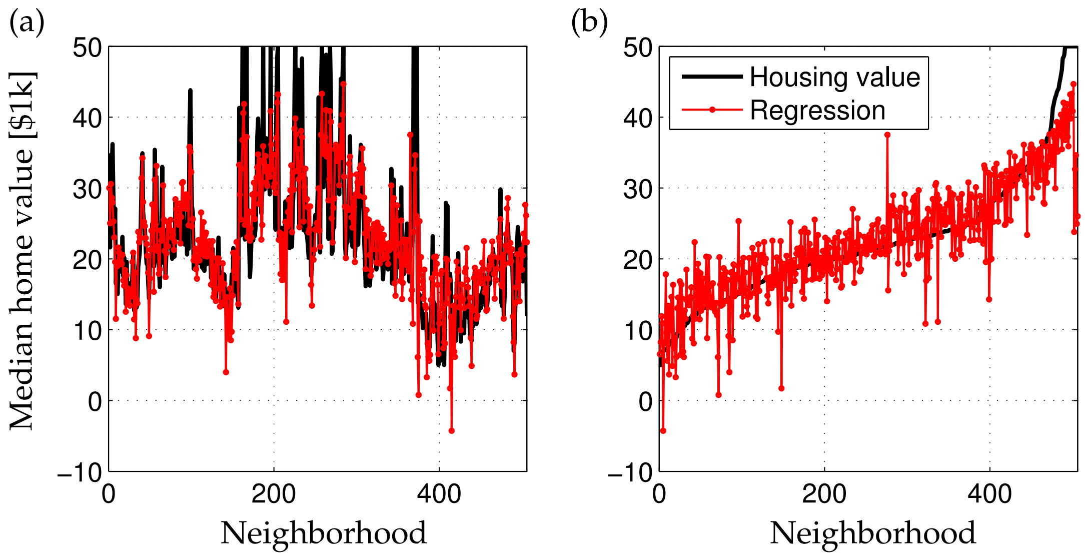


Figure 1.11: Multilinear regression of home prices using various factors described in Table 1.1. (a) Unsorted data, and (b) Data sorted by home value.

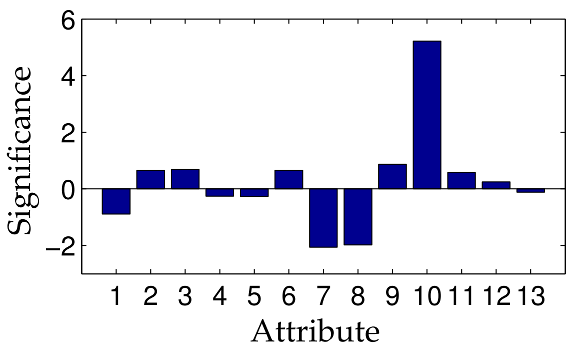


Figure 1.12: Significance of various attributes from Table 1.1 in the regression.

Table 1.1: Attributes used for Boston housing price data regression. Attributes are repeated verbatim from [3].

#	Attribute
1	CRIM: per capita crime rate by town
2	ZN: proportion of residential land zoned for lots over 25,000 sq.ft.
3	INDUS: proportion of non-retail business acres per town
4	CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
5	NOX: nitric oxides concentration (parts per 10 million)
6	RM: average number of rooms per dwelling
7	AGE: proportion of owner-occupied units built prior to 1940
8	DIS: weighted distances to five Boston employment centres
9	RAD: index of accessibility to radial highways
10	TAX: full-value property-tax rate per $10,000
11	PTRATIO: pupil-teacher ratio by town
12	B: $1000(Bk - 0.63)^2$ where Bk is the proportion of blacks by town
13	LSTAT: % lower status of the population

Code 1.11: Multilinear regression for Boston housing data.

```
clear all, close all, clc

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 for nonzero offset

x = regress(b,A);

subplot(1,2,1)
plot(b,'k-o');
hold on, grid on
plot(A*x,'r-o');
set(gca,'FontSize',13)
xlim([0 size(A,1)])

subplot(1,2,2)
[b sortind] = sort(housing(:,14)); % sorted values
plot(b,'k-o')
hold on, grid on
plot(A(sortind,:)*x,'r-o')
l1=legend('Housing value','Regression')
set(l1,'Location','NorthWest')
set(gca,'FontSize',13)
```

```
xlim([0 size(A,1)])

set(gcf,'Position',[100 100 600 250])


%%
A2 = A-ones(size(A,1),1)*mean(A,1);
for i=1:size(A,2)-1
    A2std = std(A2(:,i));
    A2(:,i) = A2(:,i)/A2std;
end
A2(:,end) = ones(size(A,1),1);

x = regress(b,A2)
figure
bar(x(1:13))

xlabel('Attribute'), ylabel('Correlation')
set(gca,'FontSize',13)
set(gcf,'Position',[100 100 600 250])
```

1.4.3 Caution

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

The first complication arises when $\hat{\mathbf{\Sigma}}$ contains singular vales that are zero, or very nearly machine-precision zero. In this case, we must truncate the SVD as in Eq. (1.6), where $\tilde{\mathbf{\Sigma}}_1$ is a diagonal matrix consisting of the nonzero singular values. The pseudo-inverse is then $\mathbf{X}^\dagger = \tilde{\mathbf{V}}^*\tilde{\mathbf{\Sigma}}^{-1}\tilde{\mathbf{U}}^*$.

The second complication arises when working with a truncated basis of left singular vectors $\hat{\mathbf{U}}$ (and $\hat{\mathbf{U}}_1$). It is still true that $\hat{\mathbf{U}}^*\hat{\mathbf{U}} = \mathbb{I}_{m\times m}$ (and $\hat{\mathbf{U}}_1^*\hat{\mathbf{U}}_1 = \mathbb{I}_{r\times r}$, where r is the rank of $\mathbf{X}$). However, $\hat{\mathbf{U}}\hat{\mathbf{U}}^* \neq \mathbb{I}_{n\times n}$, which is easy to verify numerically on a simple example. Assuming that $\hat{\mathbf{U}}\hat{\mathbf{U}}^*$ is equal to the identify is one of the most common accidental misuses of the SVD[6].

```
>> 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
```

[6]The authors are not immune to this, having mistakenly used this fictional identity in an early version of [8].

1.5 Principal components 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. PCA involves finding a new coordinate system in terms of the dominant correlations in a data set; this involves the correlation matrices described in Sec. 1.3.1, and a major difference is that in PCA we mean subtract the data before performing SVD. The geometry of the resulting coordinate system is determined by principal components (PCs) that are uncorrelated to each other, but have maximal correlation with the measurements. This theory was developed in 1901 by Pearson [40], and independently by Hotelling in the 1930s [27, 28]. Jolliffe [29] 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.

1.5.1 Computation

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

$$\bar{\mathbf{x}}_j = \frac{1}{n}\sum_{i=1}^{n} \mathbf{X}_{ij}, \tag{1.23}$$

and the mean matrix is

$$\bar{\mathbf{X}} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \bar{\mathbf{x}}. \tag{1.24}$$

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

$$\mathbf{B} = \mathbf{X} - \bar{\mathbf{B}}. \tag{1.25}$$

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

$$\mathbf{C} = \frac{1}{n-1}\mathbf{B}^*\mathbf{B}. \tag{1.26}$$

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

$$\mathbf{u}_1 = \underset{\|\mathbf{u}_1\|=1}{\operatorname{argmax}}\ \mathbf{u}_1^*\mathbf{B}^*\mathbf{B}\mathbf{u}_1, \tag{1.27}$$

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 $\mathbf{C}$:

$$\mathbf{V}^{-1}\mathbf{C}\mathbf{V}^{-1} = \mathbf{D}, \tag{1.28}$$

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 above.

1.5.2 Example: Noisy Gaussian data

Consider the noisy cloud of data in Fig. 1.13 (a), generated using Code 1.12. The data is generated by selecting 10000 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 1.2 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 1.13, 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 1.2, matches the data scaling quite closely. The matrix $\mathbf{U}$ from the SVD also closely matches the rotation matrix, up to a sign on the columns.

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

Table 1.2: Standard deviation of data and singular values from normalized SVD.

	σ_1	σ_2
Data	2	0.5
SVD	1.974	0.503

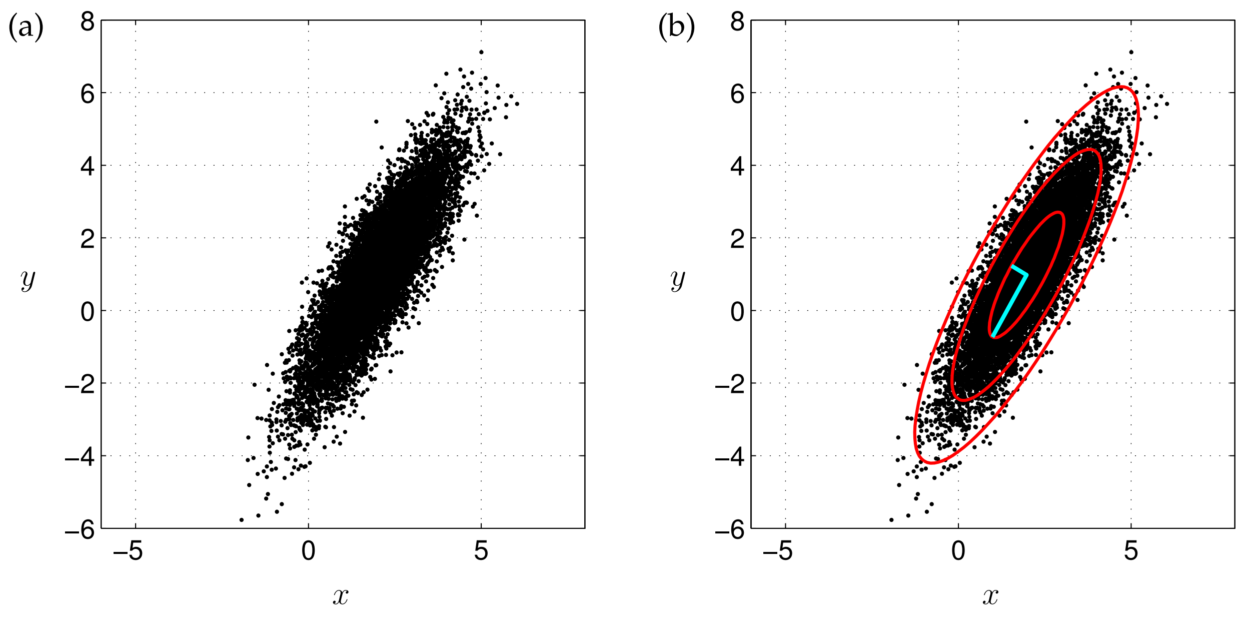


Figure 1.13: Principal components capture the variance of mean-subtracted Gaussian data (a). The first three standard deviation ellipsoids (red), and the two left singular vectors, scaled by singular values ($\sigma_1 u_1 + x_C$ and $\sigma_2 u_2 + x_C$, cyan), are shown in (b).

Code 1.12: Create noisy cloud of data to illustrate PCA.

```
clear all, close all, clc

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);

subplot(1,2,1)                          % Plot cloud of noisy data
scatter(X(1,:),X(2,:),'k.','LineWidth',2)
hold on, box on, grid on
axis([-6 8 -6 8])
```

Code 1.13: 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');  % Find principal components
    (SVD)
```

```
subplot(1,2,2)
scatter(X(1,:),X(2,:),'k.','LineWidth',2)  % Plot data to overlay
   PCA
hold on, box on, grid on
axis([-6 8 -6 8])

theta = (0:.01:1)*2*pi;
[Xstd] = U*S*[cos(theta); sin(theta)];  % 1-std confidence interval
plot(Xavg(1)+Xstd(1,:),Xavg(2) + Xstd(2,:),'r-','LineWidth',1.5)
plot(Xavg(1)+2*Xstd(1,:),Xavg(2) + 2*Xstd(2,:),'r-','LineWidth',1.5)
plot(Xavg(1)+3*Xstd(1,:),Xavg(2) + 3*Xstd(2,:),'r-','LineWidth',1.5)

% Plot principal components U(:,1)S(1,1) and U(:,2)S(2,2)
plot([Xavg(1) Xavg(1)+U(1,1)*S(1,1)],[Xavg(2) Xavg(2)+U(2,1)*S(1,1)
   ],'c-','LineWidth',2)
plot([Xavg(1) Xavg(1)+U(1,2)*S(2,2)],[Xavg(2) Xavg(2)+U(2,2)*S(2,2)
   ],'c-','LineWidth',2)

set(gcf,'Position',[100 100 600 300])
```

Finally, using the `pca` command:

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

ans =
   2.2878e-13
```

1.5.3 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 which have ovarian cancer, and 95 which do not. For each patient, there is a vector of data on 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 [44].

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 1.14. 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 much greater

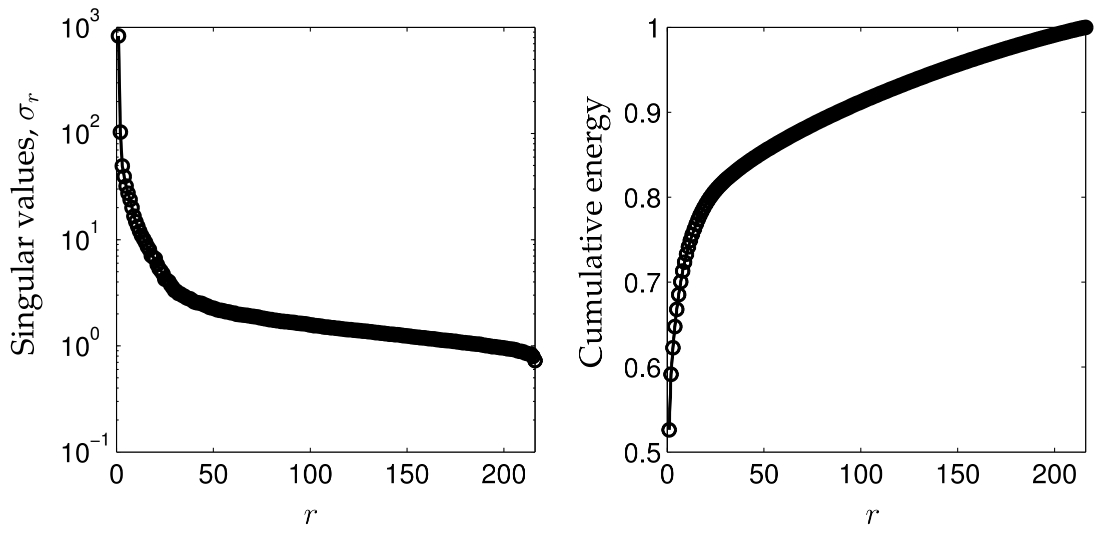


Figure 1.14: Singular values for the Ovarian cancer data.

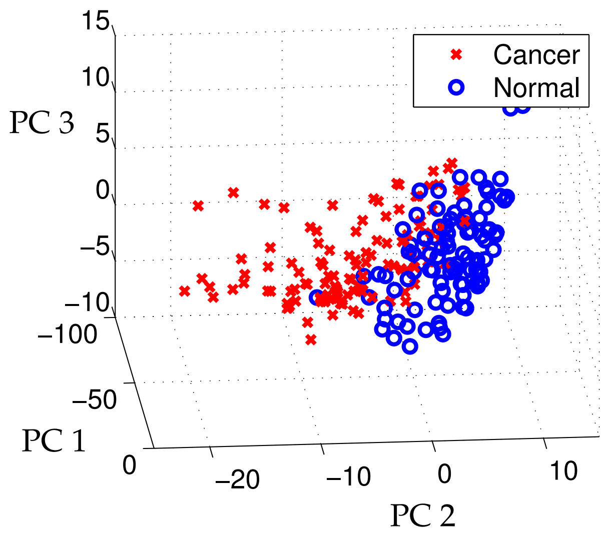


Figure 1.15: Clustering of samples that are normal and those that have cancer in the first three principal component coordinates.

detail in Chapter **??**.

Code 1.14: Compute PCA for ovarian cancer data.

```
clear all, close all, clc

load ovariancancer;   % Load ovarian cancer data

[U,S,V] = svd(obs,'econ');

figure
```

```
subplot(1,2,1)
semilogy(diag(S),'k-o','LineWidth',1.5)
set(gca,'FontSize',13), axis tight, grid on
subplot(1,2,2)
plot(cumsum(diag(S))./sum(diag(S)),'k-o','LineWidth',1.5)
set(gca,'FontSize',13), axis tight, grid on
set(gcf,'Position',[100 100 600 250])

figure, hold on
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
view(85,25), grid on, set(gca,'FontSize',13)
```

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 **??** that images of the same person tend to cluster in the eigenface space, making this a useful transformation for facial recognition and classification [51, 4]. The eigenface problem was first studied by Sirovich and Kirby in 1987 [49] and expanded on in [31]. Its application to automated facial recognition was presented by Turk and Pentland in 1991 [53].

Here, we demonstrate this algorithm using the Extended Yale Face Database B [16], consisting of cropped and aligned images [34] of 38 individuals (28 from the extended database, and 10 from the original database) under 9 poses and 64 lighting conditions[7]. Each image is 192 pixels tall and 168 pixels wide. Unlike the previous image example in Section 1.2.2, each of the facial images in our library have been reshaped into a large column vector with $192 * 168 = 32256$ 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 Codes 1.15 and 1.16.

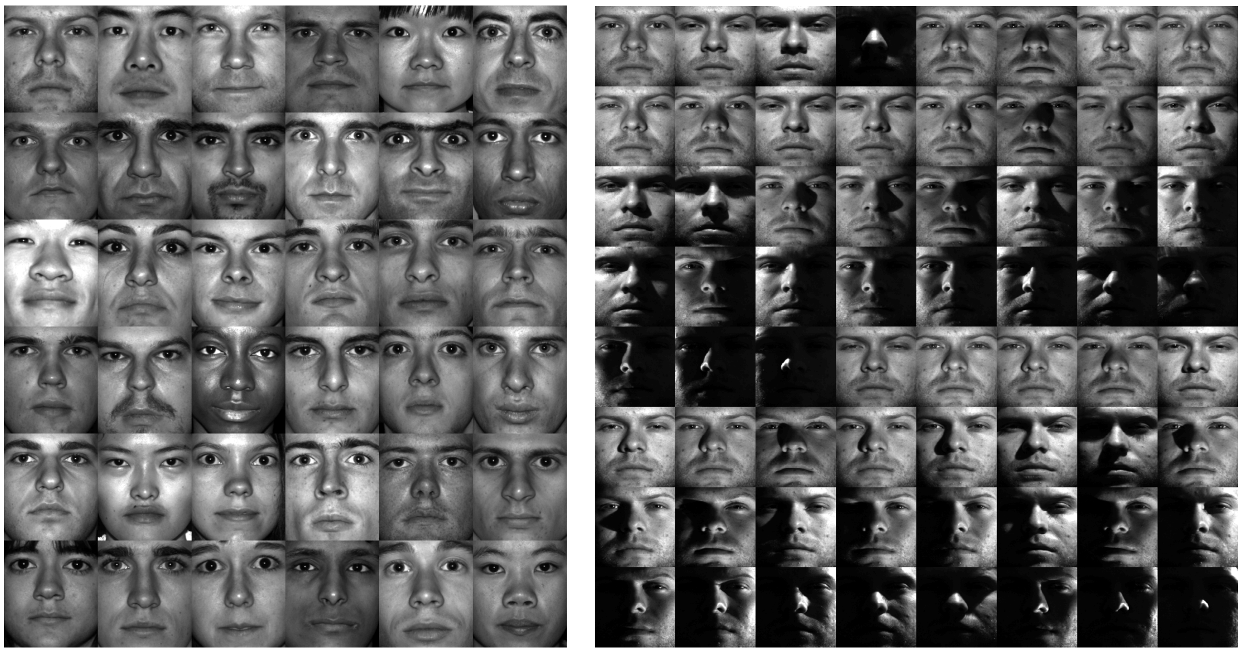


Figure 1.16: (left) A single image for each person in Yale database, and (right) all images for a specific person. Generated by Codes (1.15) & (1.16).

[7]The database can be downloaded at `http://vision.ucsd.edu/~leekc/ExtYaleDatabase/ExtYaleB.html`.

Code 1.15: Plot an image for each person in Yale database (Fig. 1.16 (a))

```
clear all, close all, clc

load ../DATA/allFaces.mat

allPersons = zeros(n*6,m*6);
count = 1;
for i=1:6
    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

figure(1), axes('position',[0  0  1  1]), axis off
imagesc(allPersons), colormap gray
```

Code 1.16: Plot each image for a specific person in Yale database (Fig. 1.16 (b))

```
for person = 1:length(nfaces)
    subset = faces(:,1+sum(nfaces(1:person-1)):sum(nfaces(1:person))
       );
    allFaces = zeros(n*8,m*8);

    count = 1;
    for i=1:8
        for j=1:8
            if(count<=nfaces(person))
                allFaces(1+(i-1)*n:i*n,1+(j-1)*m:j*m) ...
                    = reshape(subset(:,count),n,m);
                count = count + 1;
            end
        end
    end

    imagesc(allFaces), colormap gray
end
```

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 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 1.17.

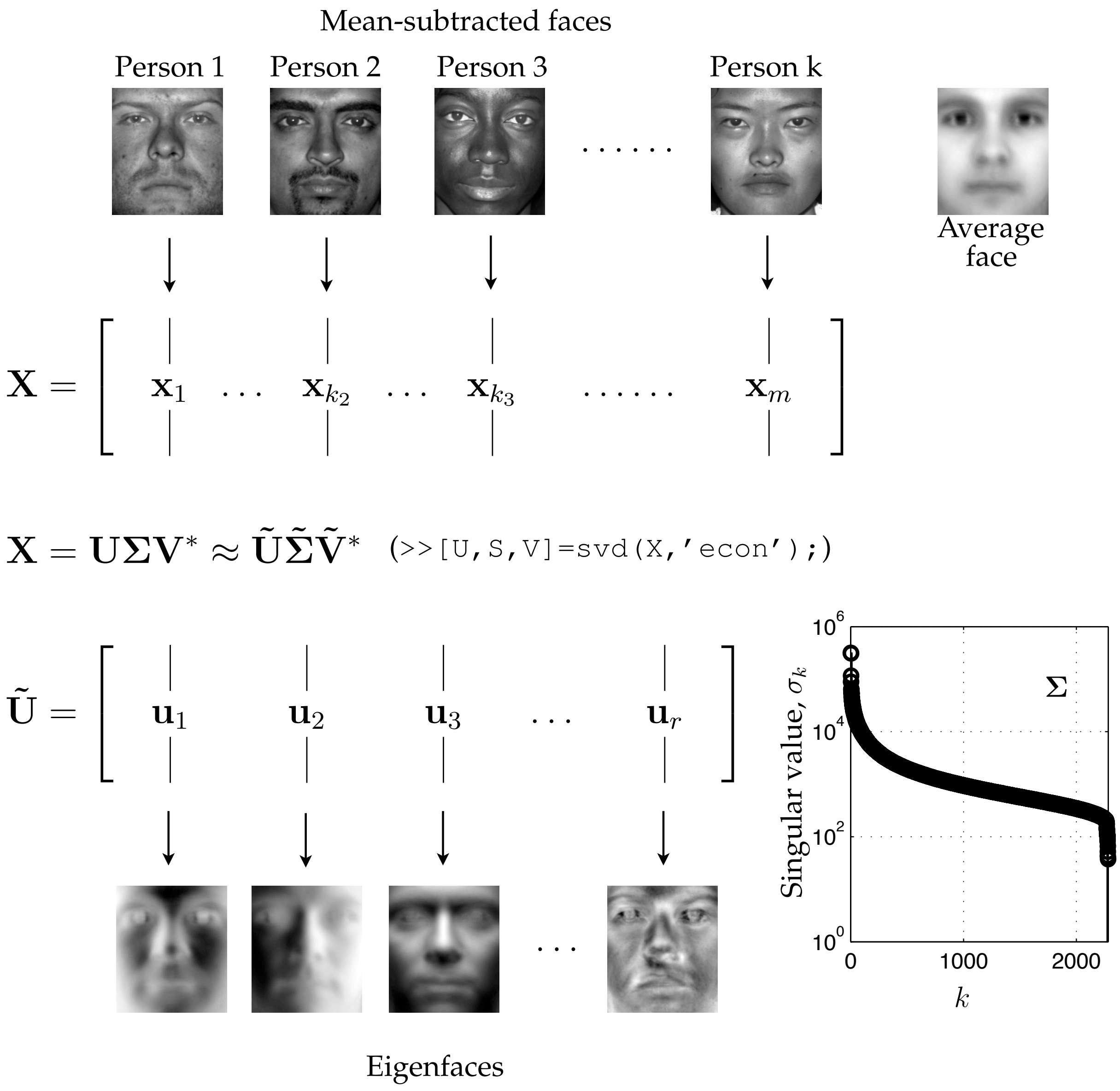

Mean-subtracted faces
Person 1
Person 2
Person 3
Person k
Average face
$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_{k_2} & \cdots & \mathbf{x}_{k_3} & \cdots & \mathbf{x}_m \end{bmatrix}$
$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \approx \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*$ (>>[U,S,V]=svd(X,'econ');)
$\tilde{\mathbf{U}} = \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \mathbf{u}_3 & \cdots & \mathbf{u}_r \end{bmatrix}$
Eigenfaces
Singular value, σ_k
Σ
10^6
10^4
10^2
10^0
0
1000
2000
k

Figure 1.17: Schematic of procedure to obtain eigenfaces from library of facial images.

Code 1.17: 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');

figure, axes('position',[0  0  1  1]), axis off
```

```
imagesc(reshape(avgFace,n,m)), colormap gray

for i=1:50  % plot the first 50 eigenfaces
    pause(0.1);  % wait for 0.1 seconds
    imagesc(reshape(U(:,i),n,m)); colormap gray;
end
```

Using the eigenface library, ($\mathbf{U}$), obtained above, we now attempt to approximately represent an image that was not in the training data. At the beginning, we held back two individuals (the 37^{th} and 38^{th} 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:

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

The eigenface approximation for various values of r is shown in Fig. 1.18, as computed using Code 1.18. The approximation is relatively poor for $r \leq 200$, although from $r = 400$ to $r = 1600$ 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, non-localized spatial features, such as cheeks, forehead, mouths, etc.

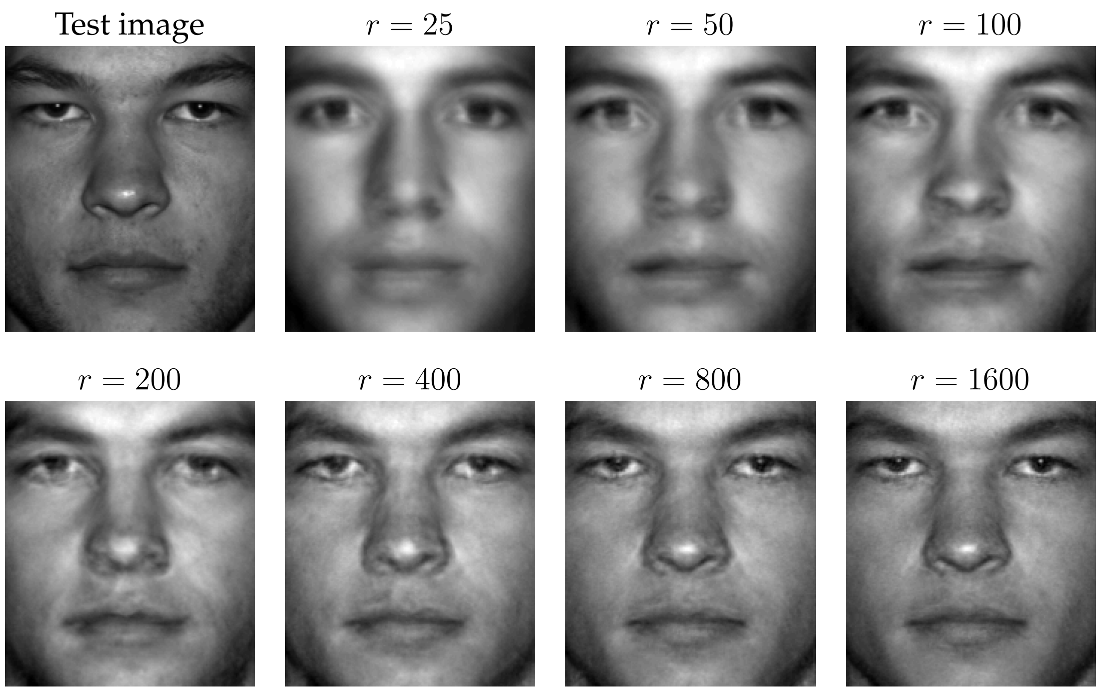


Figure 1.18: Approximate representation of test image using eigenfaces basis of various order r. Test image is not in training set.

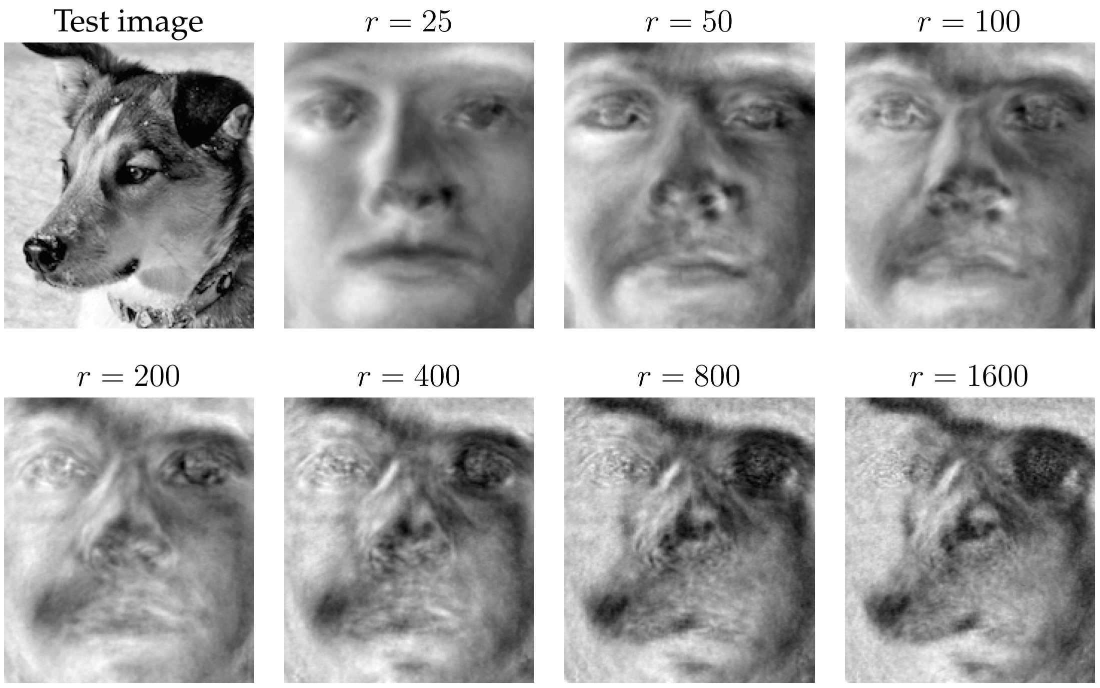


Figure 1.19: Approximate representation of an image of a dog using eigenfaces.

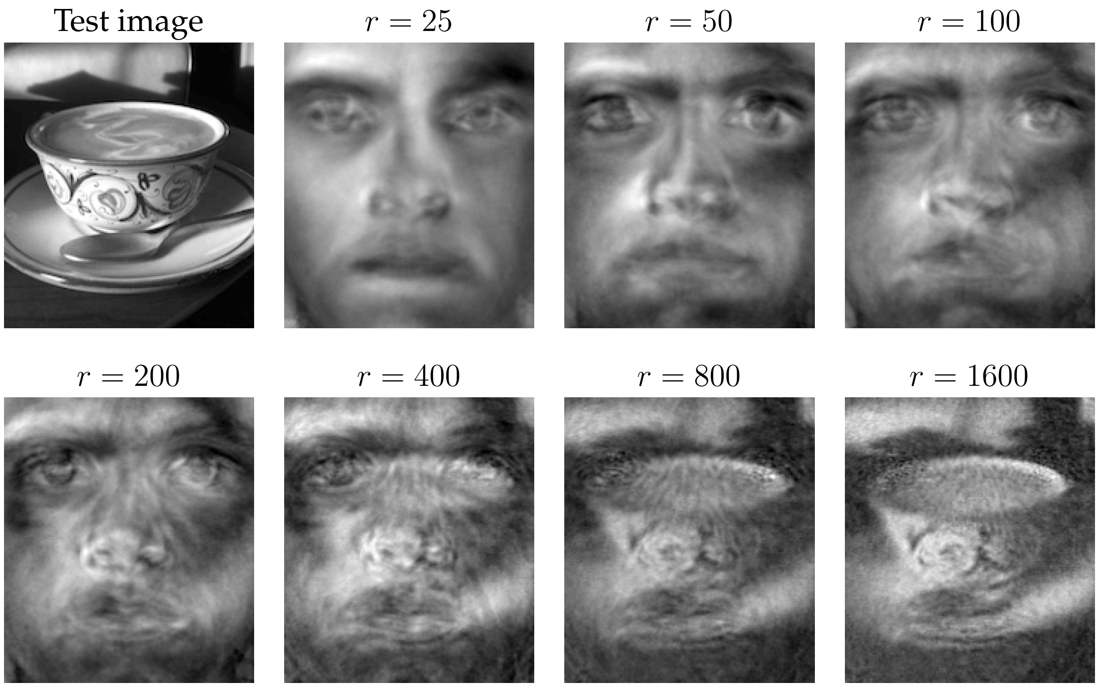


Figure 1.20: Approximate representation of a cappuccino using eigenfaces.

Code 1.18: Approximate test-image that was omitted from training data, using eigenfaces.

```
testFace = faces(:,1+sum(nfaces(1:36))); % first face of person 37
axes('position',[0  0  1  1]), axis off
imagesc(reshape(testFace,n,m)), colormap gray

testFaceMS = testFace - avgFace;
for r=25:25:2275
    reconFace = avgFace + (U(:,1:r)*(U(:,1:r)'*testFaceMS));
    imagesc(reshape(reconFace,n,m)), colormap gray
    title(['r=',num2str(r,'%d')]);
    pause(0.1)
end
```

We further investigate the use of the eigenfaces as a coordinate system, defining an eigenface space. By projecting an image x onto the first r PCA modes, we obtain a set of coordinates in this space: $\alpha = \tilde{\mathbf{U}}_r^* \mathbf{x}$. 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. Other 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 1.19. Images of the two individuals appear to be well-separated in these coordinates. This is the basis for image recognition and classification in Chapter **??**.

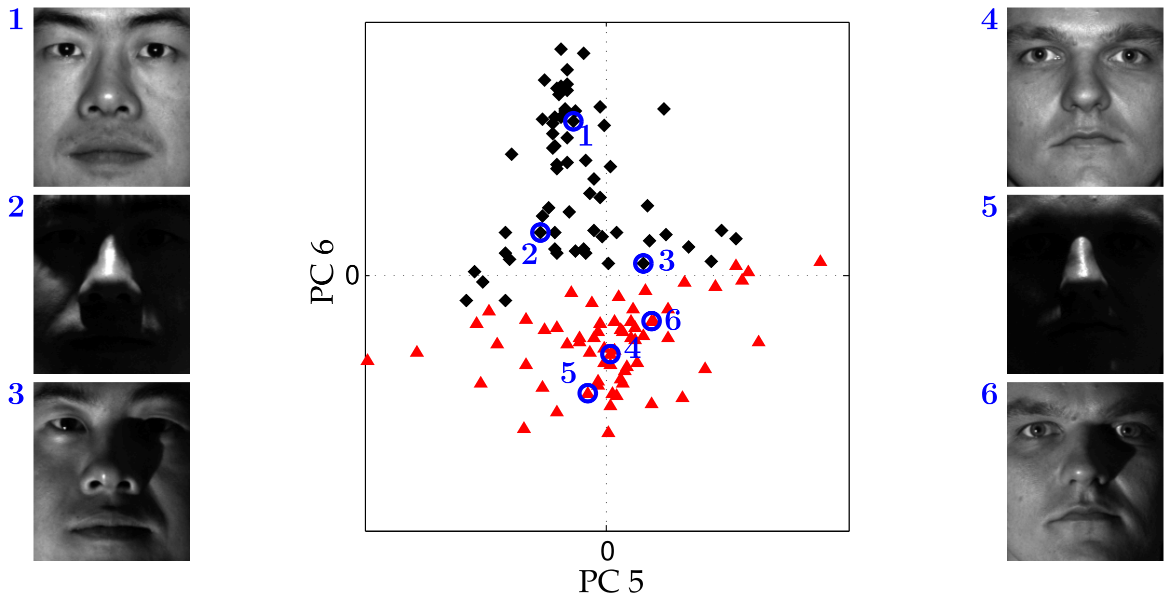


Figure 1.21: Projection of all images from two individuals onto the 5th and 6th PCA modes. Projected images of the first individual are indicated with black diamonds, and projected images of the second individual are indicated with red triangles. Three examples from each individual are circled in blue,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

Code 1.19: Project images for two specific people onto the 5th and 6th eigenfaces.

```
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));

figure
subplot(1,2,1), imagesc(reshape(P1(:,1),n,m)); colormap gray, axis
   off
subplot(1,2,2), imagesc(reshape(P2(:,1),n,m)); colormap gray, axis
   off

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

figure
plot(PCACoordsP1(1,:),PCACoordsP1(2,:),'kd','MarkerFaceColor','k')
axis([-4000 4000 -4000 4000]), hold on, grid on
plot(PCACoordsP2(1,:),PCACoordsP2(2,:),'r^','MarkerFaceColor','r')
set(gca,'XTick',[0],'YTick',[0]);
```

1.7 Truncation and hard thresholding

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 transition from singular values that represent important signal 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 provides an optimal truncation value, or hard threshold, under certain conditions [15], providing a principled approach to obtaining low-rank matrix approximations using the SVD.

1.7.1 Optimal hard threshold

A recent theoretical breakthrough determines the *optimal* hard threshold τ for singular value truncation under the assumption of Gaussian white noise on top of a low-rank signal [15]. 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 on examples. However, for more details, see [15].

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

$$\mathbf{X} = \mathbf{X}_{\text{true}} + \gamma \mathbf{X}_{\text{noise}}. \tag{1.29}$$

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 [15][8].

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
$$\tau = (4/\sqrt{3})\sqrt{n}\gamma. \tag{1.30}$$

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 $\beta = m/n$:
$$\begin{aligned} \tau &= \lambda(\beta)\sqrt{n}\gamma, & (1.31)\\ \lambda(\beta) &= \left(2(\beta+1) + \frac{8\beta}{(\beta+1) + (\beta^2 + 14\beta + 1)^{1/2}}\right)^{1/2}. & (1.32)\end{aligned}$$

Note that this expression reduces to Eq. (1.30) when $\beta = 1$. If $m \ll n$, then $\beta = n/m$.

[8]In [15], σ is used to denote standard deviation and y_k denotes the k^{th} singular value.

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, but it must be approximated numerically.

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

$$\tau = \omega(\beta)\sigma_{\text{med}}. \tag{1.33}$$

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 [15] has a Matlab code supplement [12] at http://purl.stanford.edu/vg705qn9070 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 1.20) and we contaminate the signal with Gaussian white noise (Code 1.21). A de-noised and dimensionally reduced matrix is then obtained using the threshold from Eq. (1.30) (Code 1.22), as well as using a 90% energy truncation (Code 1.23). It is clear that the hard threshold is able to filter the noise more effectively. Plotting the singular values (Code 1.24) in Fig. 1.23, it is clear that there are two values that are above threshold.

Code 1.20: Compute 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(Xnoisy);
```

Code 1.21: Contaminate signal with noise. (Fig. 1.22 (b))

```
%%
sigma = 1;
Xnoisy = X+sigma*randn(size(X));
```

Code 1.22: 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 with sing. values >
    cutoff
Xclean = U(:,1:r)*S(1:r,1:r)*V(:,1:r)';

figure, imshow(Xclean)
```

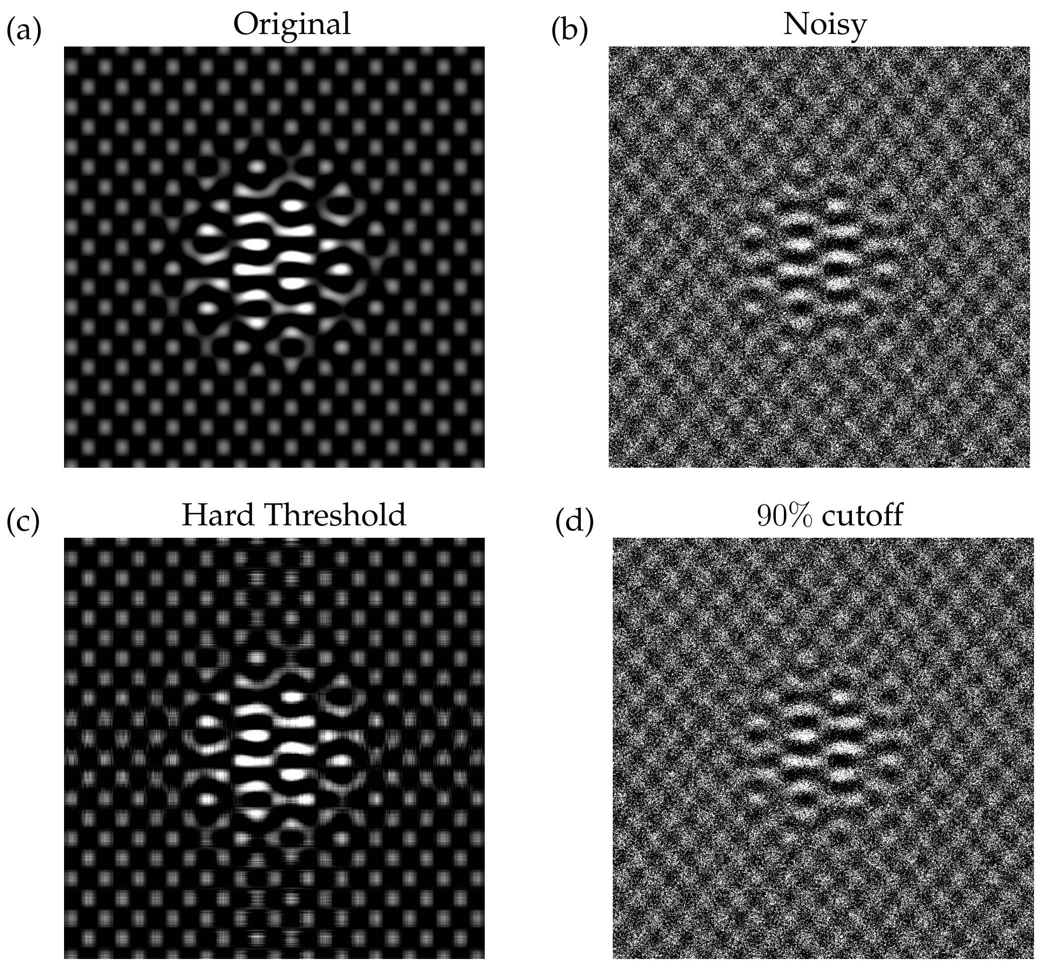


Figure 1.22: Underlying rank 2 matrix (a), matrix with noise (b), clean matrix after optimal hard threshold $(4/\sqrt{3})\sqrt{n}\sigma$ (c), and truncation based on 90% energy (d).

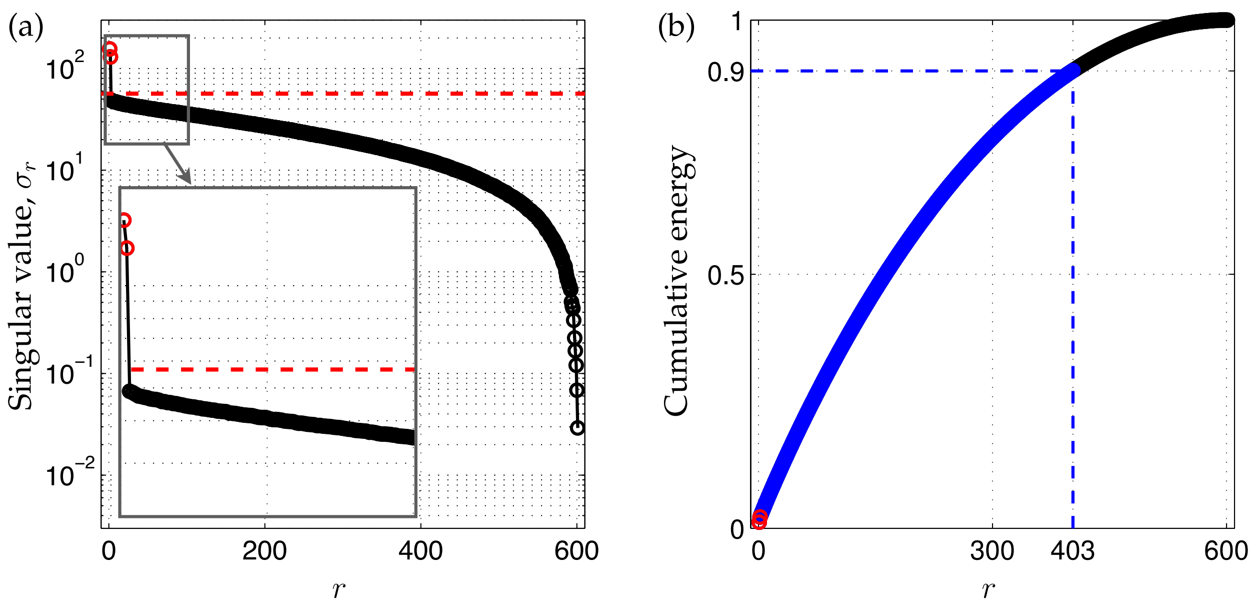


Figure 1.23: Singular values σ_r (a) and cumulative energy in first r modes (b). The optimal hard threshold $(4/\sqrt{3})\sqrt{n}\sigma$ is shown as a red dashed line (- -), and the 90% cutoff is shown as a blue dashed line (- -). For this case, $n = 600$ and $\sigma = 1$ so that the optimal cutoff is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

Code 1.23: 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 that captures 90% energy

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

figure, imshow(X90)
```

Code 1.24: Plot singular values for hard threshold example. (Fig. 1.23)

```
figure
semilogy(diag(S),'-ok','LineWidth',1.5)
hold on, grid on
semilogy(diag(S(1:r,1:r)),'or','LineWidth',1.5)
plot([-20 N+20],[cutoff cutoff],'r--','LineWidth',2)
axis([-10 610 .003 300])
rectangle('Position',[-5,20,100,200],'LineWidth',2,'LineStyle','--')

figure
semilogy(diag(S),'-ok','LineWidth',1.5)
hold on, grid on
semilogy(diag(S(1:r,1:r)),'or','LineWidth',1.5)
plot([-20 N+20],[cutoff cutoff],'r--','LineWidth',2)
axis([-5 100 20 200])
```

```
figure
plot(cdS,'-ok','LineWidth',1.5)
hold on, grid on
plot(cdS(1:r90),'ob','LineWidth',1.5)
plot(cdS(1:r),'or','LineWidth',1.5)
set(gca,'XTick',[0 300 r90 600],'YTick',[0 .5 0.9 1.0])
xlim([-10 610])
plot([r90 r90 -10],[0 0.9 0.9],'b--','LineWidth',1.5)
```

Example 2: Eigenfaces

In the second example, we revisit the eigenfaces problem from Section **??**. 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 system has 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.

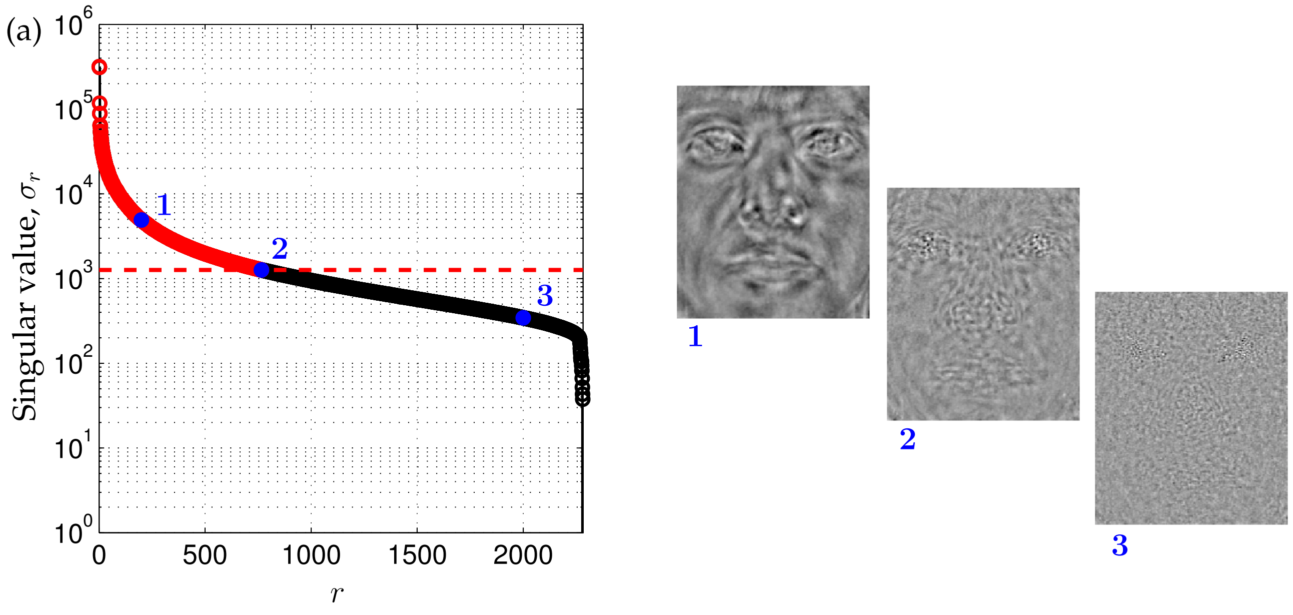


Figure 1.24: Hard thresholding for eigenfaces example.

Code 1.25: Compute hard threshold for eigenfaces example (Fig. 1.24). Assumes Code (1.17) has already been run.

```
sigs = diag(S);

beta = size(X,2)/size(X,1);  % aspect ratio of data matrix
thresh = optimal_SVHT_coef(beta,0) * median(sigs); % From Gavish &
    Donoho

figure
semilogy(sigs,'-ok','LineWidth',1.5)
axis([0 length(sigs) 1 10^6)], grid on, hold on

semilogy(sigs(sigs>thresh),'ro','LineWidth',1.5)
plot([-20 length(sigs)],[thresh thresh],'r--','LineWidth',2)

rvals = [200 766 2000];
semilogy(rvals,sigs(rvals),'bo','LineWidth',1.5,'MarkerFaceColor','b
    ');

for r = rvals
    figure
    imagesc(reshape(U(:,r),n,m)); colormap gray, axis off
end
```

1.8 Additional mathematical properties

1.8.1 Important theorems

Theorem 2 (Existence and Uniqueness) *The SVD exists and is unique.*

Theorem 3 *The rank of* $\mathbf{X}$ *is equal to the number of non-zero singular values.*

Theorem 4 *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 non-zero eigenvalues.*

Corollary 1 *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* $\mathbf{X}$*.*

Theorem 5 (Eckart-Young [13]) *The optimal rank-*r *approximation to* $\mathbf{X}$*, in an* L_2 *sense, is given by the rank-*r *SVD truncation* $\tilde{\mathbf{X}}$*:*

$$\underset{\tilde{\mathbf{X}}}{\operatorname{argmin}} \|\mathbf{X} - \tilde{\mathbf{X}}\|_2 = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*. \tag{1.34}$$

Here, $\tilde{\mathbf{U}}$ *and* $\tilde{\mathbf{V}}$ *denote the leading* r *columns of* $\mathbf{U}$ *and* $\mathbf{V}$*, and* $\tilde{\mathbf{\Sigma}}$ *contains the leading* $r \times r$ *sub-block of* $\mathbf{\Sigma}$*.*

The singular values are useful for quantifying the norms of a matrix, and of the error associated with matrix approximation.

Corollary 2 *The two-norm of* $\mathbf{X}$ *is given by* σ_1*:* $\|\mathbf{X}\|_2 = \sigma_1$*.*

Corollary 3 *The Frobenius norm of* $\mathbf{X}$ *is given by the two-norm of the vector of singular values:* $\|\mathbf{X}\|_F = \sqrt{\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_m^2} = \sqrt{\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_r^2}$*.*

The Frobenius norm is closely related to the mean-square error (MSE):

$$\text{MSE} \triangleq \sum_{i,j} \left(\mathbf{X}_{ij} - \tilde{\mathbf{X}}_{ij}\right)^2 = \|\mathbf{X} - \tilde{\mathbf{X}}\|_F^2. \tag{1.35}$$

Theorem 6 *The determinant of a square matrix* $\mathbf{X} \in \mathbb{R}^{n\times n}$ *is related to the singular values as:* $|det(\mathbf{X})| = \Pi_{k=1}^m \sigma_k$*.*

1.8.2 Four fundamental subspaces

There are four fundamental subspaces that define any matrix $\mathbf{X}$: the column space $\text{col}(\mathbf{X})$ and its orthogonal complement $\ker(\mathbf{X}^*)$, and the row space $\text{row}(\mathbf{X})$ and its orthogonal complement $\ker(\mathbf{X})$. The column space of $\mathbf{X}$, $\text{col}(\mathbf{X})$, is the span of the columns of $\mathbf{X}$; it is also known as the *range*. The kernel of $\mathbf{X}$, $\ker(\mathbf{X})$, is the subspace of vectors that map through $\mathbf{X}$ to zero, i.e. $\mathbf{X}\mathbf{v} = \mathbf{0}$; it is sometimes referred to as the *null space*. The row space of $\mathbf{X}$ is equal to $\text{row}(\mathbf{X}) = \text{col}(\mathbf{X}^*)$.

$$\text{col}(\mathbf{X}) \oplus \ker(\mathbf{X}^*) = \mathbb{R}^n \tag{1.36}$$
$$\text{col}(\mathbf{X}^*) \oplus \ker(\mathbf{X}) = \mathbb{R}^n \tag{1.37}$$

There are a number of properties that are related to the matrices in the SVD:

1. The column space of $\mathbf{X}$ is the same as the column space of $\hat{\mathbf{U}}_1$,
2. The orthogonal complement to col$(\mathbf{X})$ is given by the column space of $\hat{\mathbf{U}}_2$,
3. The null space of $\mathbf{X}$ is given by the column space of $\hat{\mathbf{V}}_2$,
4. The orthogonal complement to ker$(\mathbf{X})$ is given by the column space of $\hat{\mathbf{V}}_1$.

1.9 Caveats and extensions

Here, we discuss common pitfalls of the SVD associated with misaligned data. We also describe numerous powerful extensions to the basic SVD described above.

1.9.1 Poorly aligned 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. If the rectangle is perfectly aligned with the x- and y- axes of the figure, then the SVD is extremely simple, having only one nonzero singular value σ_1 and corresponding singular vectors $\mathbf{u}_1$ and $\mathbf{v}_1$ that define the width and height of the white rectangle.

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.

Code 1.26: Compute the SVD for a well-aligned and rotated square (Fig. 1.25).

```
clear all, close all, clc

n = 1000;
q = n/4;
X = zeros(n,n);
X(q:(n/2)+q,q:(n/2)+q) = 1;

subplot(2,2,1), imshow(X); colormap gray, axis off

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);
subplot(2,2,2), imshow(Xrot); colormap gray, axis off

[U,S,V] = svd(X);  % svd well-aligned square
[U,S,V] = svd(Xrot);  % svd rotated square

subplot(2,2,3), semilogy(diag(S),'-ko')
ylim([1.e-16 1.e4]), grid on
set(gca,'YTick',[1.e-16 1.e-12 1.e-8 1.e-4 1. 1.e4])
set(gca,'XTick',[0 250 500 750 1000])

subplot(2,2,4), semilogy(diag(S),'-ko')
ylim([1.e-16 1.e4]), grid on
set(gca,'YTick',[1.e-16 1.e-12 1.e-8 1.e-4 1. 1.e4])
set(gca,'XTick',[0 250 500 750 1000])
```

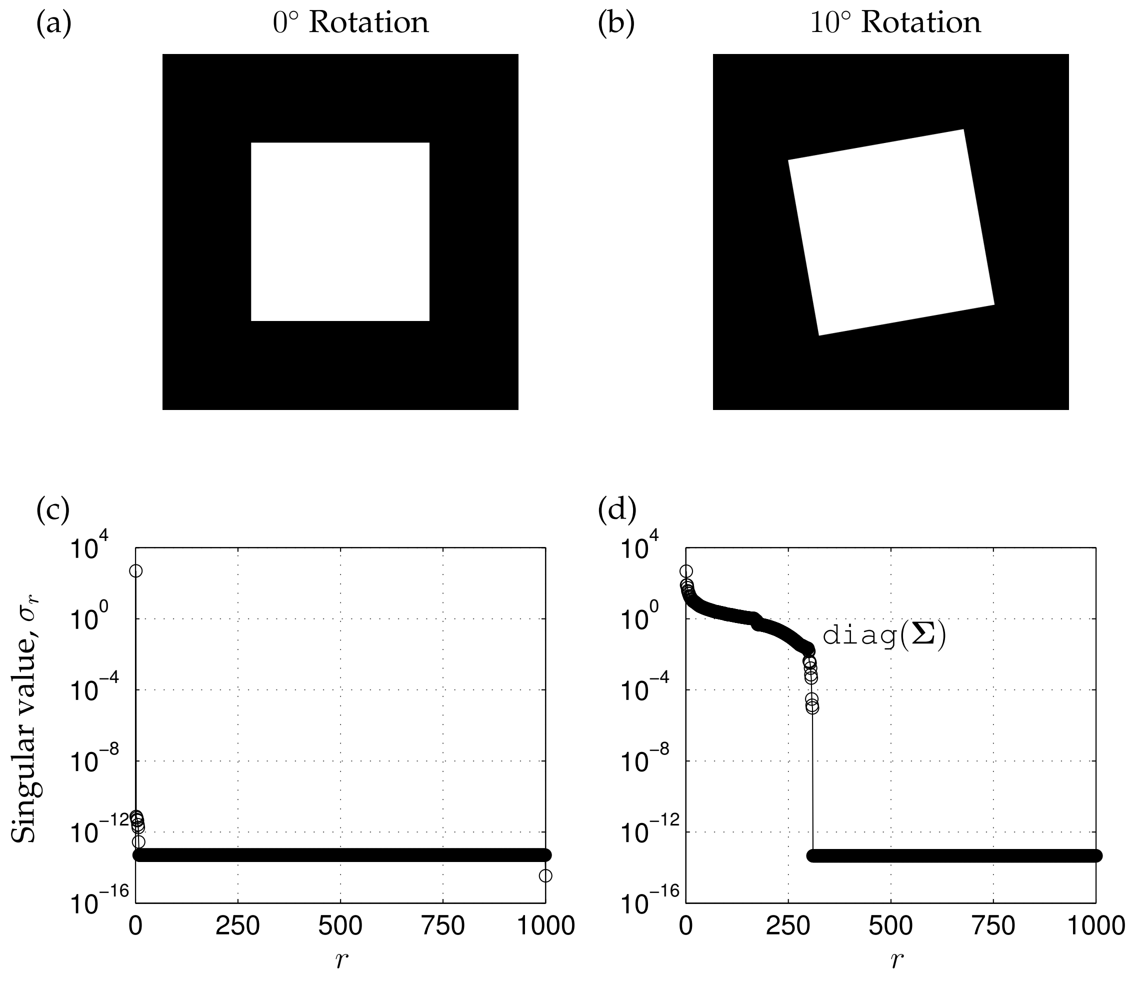


Figure 1.25: A data matrix consisting of ones with a square sub-block of zeros (a), and its SVD spectrum (c). If we rotate the image by 10°, as in (b), the SVD spectrum becomes significantly more complex (d).

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 is not invariant to translations, rotations, or scaling of data, 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.

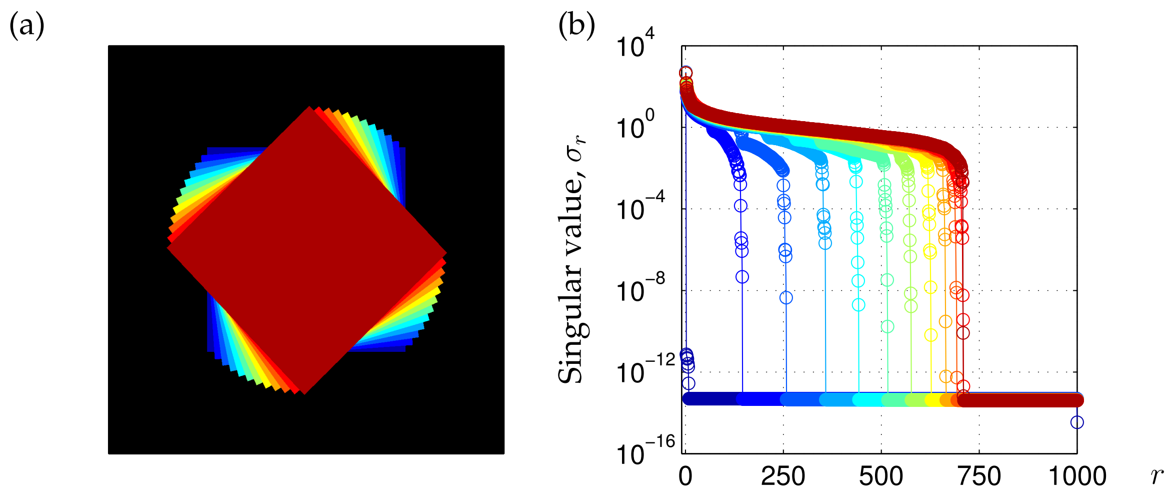


Figure 1.26: A data matrix consisting of ones with a square sub-block of zeros at various rotations (a), and the corresponding SVD spectrum, `diag(S)`, (c).

Code 1.27: SVD for a square rotated through various angles (Fig. 1.26).

```
clear all, close all, clc
n = 1000;
X = zeros(n,n);
X(n/4:3*n/4,n/4:3*n/4) = 1;
nAngles = 12;  % sweep through 12 different angles, from 0:4:44
cm=colormap(jet(nAngles));

[U,S,V] = svd(X);
subplot(1,2,1), imagesc(X), hold on;
subplot(1,2,2), semilogy(diag(S),'-o','color',cm(1,:)), hold on,
   grid on

Xrot = X;
for j=2:nAngles
    Y = imrotate(X,(j-1)*4,'bicubic');  % rotate  theta = (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);
    subplot(1,2,1), imagesc(Xrot), colormap([0 0 0; cm])
    subplot(1,2,2), semilogy(diag(S),'-o','color',cm(j,:))
end
axis([1.e-16 1.e3 -10 1000])
set(gca,'XTick',[0 250 500 750 1000])
set(gca,'YTick',[1.e-16 1.e-12 1.e-8 1.e-4 1. 1.e4]);
set(gcf,'Position',[100 100 550 230])
```

1.9.2 Advanced extensions to the SVD

There are numerous advanced variants of the SVD that have been developed in various contexts. The streaming or incremental SVD algorithm provides a fast algorithm to approximate the SVD for large data sets that are being continuously updated with streaming data [7, 6]. These streaming data sets include large recommender systems that provide suggestions and marketing to users [46]. These systems constantly assimilate new user purchasing data, and it would be prohibitively expensive to frequently recompute these large SVDs from scratch. Another important SVD algorithm based on streaming data is the sketched SVD [18].

SVD has also been extended to randomly projected or subsampled data [14, 43]. Low-rank SVD approximations may also be obtained using randomized SVD, which involves randomly selecting columns for inner products [35]. There are numerous other generalized SVD algorithms [54], including a multilinear SVD [11] and an SVD for tensors [22].

1.10 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 intents and 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 below, 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 [20, 9, 19, 32, 24]. A more thorough discussion of computational issues can be found in [21].

1.10.1 SVD in various languages

We have already seen that the SVD is quite simple to use in Matlab:

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

Below, we describe the SVD interface in some of the most common computational languages.

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 sized SVD
>>> U, S, V = 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]
```

C++

There are many options for the SVD in C++, and we describe just a few:

1. Armadillo

```
#include <armadillo>
using namespace arma;

mat X = randn<mat>(5,3);

mat U;
vec s;
mat S;
mat V;

svd(U,s,V,X);
S = diagmat(s);
```

2. Eigen (templated library)

```
#include <Eigen/Core>
#include <Eigen/SVD>

MatrixXf X = MatrixXf::Random(5,3);
JacobiSVD<MatrixXf> svd(X, ComputeThinU | ComputeThinV);

MatrixXf U = svd.matrixU();
MatrixXf V = svd.matrixV();
VectorXf s = svd.singularValues();
MatrixXf S = s.asDiagonal();
```

3. redsvd - This is a randomized SVD library for large matrices, based on the Eigen package.

Fortran and LAPACK

In Fortran, most SVD implementations are based on the LAPACK (Linear Algebra Package) [1]. In particular, the Fortran routine is designated `DGESVD` in LAPACK. Many other computational libraries wrap these basic LAPACK routines.

Bibliography

[1] Edward Anderson, Zhaojun Bai, Christian Bischof, Susan Blackford, James Demmel, Jack Dongarra, Jeremy Du Croz, Anne Greenbaum, S Hammerling, Alan McKenney, et al. *LAPACK Users' guide*, volume 9. Siam, 1999.

[2] Athanasios C Antoulas. *Approximation of large-scale dynamical systems*, volume 6. Siam, 2005.

[3] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[4] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman. Eigenfaces vs. Fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 19(7):711–720, 1997.

[5] G. Berkooz, P. Holmes, and J. L. Lumley. The proper orthogonal decomposition in the analysis of turbulent flows. *Annual Review of Fluid Mechanics*, 23:539–575, 1993.

[6] M. Brand. Fast low-rank modification of the thin singular value decomposition. *Linear Algebra and its Applications*, 415:20–30, 2006.

[7] Matthew Brand. Incremental singular value decomposition of uncertain data with missing values. In *Computer Vision—ECCV 2002*, pages 707–720. Springer, 2002.

[8] S. L. Brunton, J. L. Proctor, and J. N. Kutz. Compressive sampling and dynamic mode decomposition. *arXiv preprint arXiv:1312.5186*, 2014.

[9] Peter A Businger and Gene H Golub. Algorithm 358: singular value decomposition of a complex matrix [f1, 4, 5]. *Communications of the ACM*, 12(10):564–565, 1969.

[10] Steve Cherry. Singular value decomposition analysis and canonical correlation analysis. *Journal of Climate*, 9(9):2003–2009, 1996.

[11] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. A multilinear singular value decomposition. *SIAM journal on Matrix Analysis and Applications*, 21(4):1253–1278, 2000.

[12] D. L. Donoho and M. Gavish. Code supplement to "the optimal hard threshold for singular values is $4/\sqrt{3}$". http://purl.stanford.edu/vg705qn9070, 2014.

[13] C. Eckart and G. Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[14] J. E. Fowler. Compressive-projection principal component analysis. *IEEE Transactions on Image Processing*, 18(10):2230–2242, 2009.

[15] M. Gavish and D. L. Donoho. The optimal hard threshold for singular values is $4/\sqrt{3}$. *ArXiv e-prints*, 2014.

[16] A. S. Georghiades, P. N. Belhumeur, and D. J. Kriegman. From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 23(6):643–660, 2001.

[17] Jan J Gerbrands. On the relationships between svd, klt and pca. *Pattern recognition*, 14(1):375–381, 1981.

[18] A. C. Gilbert, J. Y. Park, and M. B. Wakin. Sketched SVD: Recovering spectral features from compressive measurements. *ArXiv e-prints*, 2012.

[19] G. H. Golub and C. Reinsch. Singular value decomposition and least squares solutions. *Numerical Mathematics*, 14:403–420, 1970.

[20] Gene Golub and William Kahan. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial &amp; Applied Mathematics, Series B: Numerical Analysis*, 2(2):205–224, 1965.

[21] Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

[22] Lars Grasedyck. Hierarchical singular value decomposition of tensors. *SIAM Journal on Matrix Analysis and Applications*, 31(4):2029–2054, 2010.

[23] David Harrison Jr and Daniel L Rubinfeld. Hedonic housing prices and the demand for clean air. *Journal of environmental economics and management*, 5(1):81–102, 1978.

[24] MT Heath, AJ Laub, CC Paige, and RC Ward. Computing the singular value decomposition of a product of two matrices. *SIAM Journal on Scientific and Statistical Computing*, 7(4):1147–1159, 1986.

[25] P. J. Holmes, J. L. Lumley, G. Berkooz, and C. W. Rowley. *Turbulence, coherent structures, dynamical systems and symmetry*. Cambridge Monographs in Mechanics. Cambridge University Press, Cambridge, England, 2nd edition, 2012.

[26] Philip Holmes and John Guckenheimer. *Nonlinear oscillations, dynamical systems, and bifurcations of vector fields*, volume 42 of *Applied Mathematical Sciences*. Springer-Verlag, Berlin, Heidelberg, 1983.

[27] H. Hotelling. Analysis of a complex of statistical variables into principal components. 24:417–441, September 1933.

[28] H. Hotelling. Analysis of a complex of statistical variables into principal components. 24:498–520, October 1933.

[29] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2005.

[30] K Karhunen. Über lineare methoden in der wahrscheinlichkeitsrechnung, vol. 37. *Annales Academiæ-Scientiarum Fennicæ, Ser. A. I*, 1947.

[31] M. Kirby and L. Sirovich. Application of the Karhunen-Loève procedure for the characterization of human faces. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 12(1):103–108, 1990.

[32] V. C. Klema and A. J. Laub. The singular value decomposition: Its computation and some applications. *IEEE Transactions on Automatic Control*, 25(2):164–176, 1980.

[33] J. N. Kutz. *Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data*. Oxford University Press, 2013.

[34] K.C. Lee, J. Ho, and D. Kriegman. Acquiring linear subspaces for face recognition under variable lighting. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 27(5):684–698, 2005.

[35] Edo Liberty, Franco Woolfe, Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert. Randomized algorithms for the low-rank approximation of matrices. *Proceedings of the National Academy of Sciences*, 104(51):20167–20172, 2007.

[36] M Loeve. Van nostrand, princeton, nj. *Probability Theory*, 1955.

[37] E. N. Lorenz. Empirical orthogonal functions and statistical weather prediction. Technical report, Massachusetts Institute of Technology, December 1956.

[38] John Mandel. Use of the singular value decomposition in regression analysis. *The American Statistician*, 36(1):15–24, 1982.

[39] B. C. Moore. Principal component analysis in linear systems: Controllability, observability, and model reduction. *IEEE Transactions on Automatic Control*, AC-26(1):17–32, 1981.

[40] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(7–12):559–572, 1901.

[41] Roger Penrose. A generalized inverse for matrices. In *Mathematical proceedings of the Cambridge philosophical society*, volume 51, pages 406–413. Cambridge Univ Press, 1955.

[42] Roger Penrose and John Arthur Todd. On best approximate solutions of linear matrix equations. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 52, pages 17–19. Cambridge Univ Press, 1956.

[43] H. Qi and S. M. Hughes. Invariance of principal components under low-dimensional random projection of the data. IEEE International Conference on Image Processing, October 2012.

[44] Soumya Raychaudhuri, Joshua M Stuart, and Russ B Altman. Principal components analysis to summarize microarray experiments: application to sporulation time series. In *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, page 455. NIH Public Access, 2000.

[45] Charles A Rohde. Generalized inverses of partitioned matrices. *Journal of the Society for Industrial &amp; Applied Mathematics*, 13(4):1033–1035, 1965.

[46] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Incremental singular value decom-

position algorithms for highly scalable recommender systems. In *Fifth International Conference on Computer and Information Science*, pages 27–28. Citeseer, 2002.

[47] E. Schmidt. Zur theorie der linearen und nichtlinearen integralgleichungen. i teil. entwicklung willkürlichen funktionen nach system vorgeschriebener. *Math. Ann.*, 3:433–476, 1907.

[48] L. Sirovich. Turbulence and the dynamics of coherent structures, parts I-III. *Q. Appl. Math.*, XLV(3):561–590, 1987.

[49] L. Sirovich and M. Kirby. A low-dimensional procedure for the characterization of human faces. *Journal of the Optical Society of America A*, 4(3):519–524, 1987.

[50] Gilbert W Stewart. On the early history of the singular value decomposition. *SIAM review*, 35(4):551–566, 1993.

[51] D. L. Swets and J. Weng. Using discriminant eigenfeatures for image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 18(8):831–836, 1996.

[52] Lloyd N Trefethen and David Bau III. *Numerical linear algebra*, volume 50. Siam, 1997.

[53] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[54] Charles F Van Loan. Generalizing the singular value decomposition. *SIAM Journal on Numerical Analysis*, 13(1):76–83, 1976.

[55] Sanjo Zlobec. An explicit form of the moore-penrose inverse of an arbitrary complex matrix. *SIAM Review*, 12(1):132–134, 1970.