Dimensionality Reduction: Lecture Notes and Labs

Mathematics Foundations for Machine Learning

Adérito Araújo

November, 2025

Contents

Chapter 1

Basics on Linear Algebra

1.1 Multichannel image processing

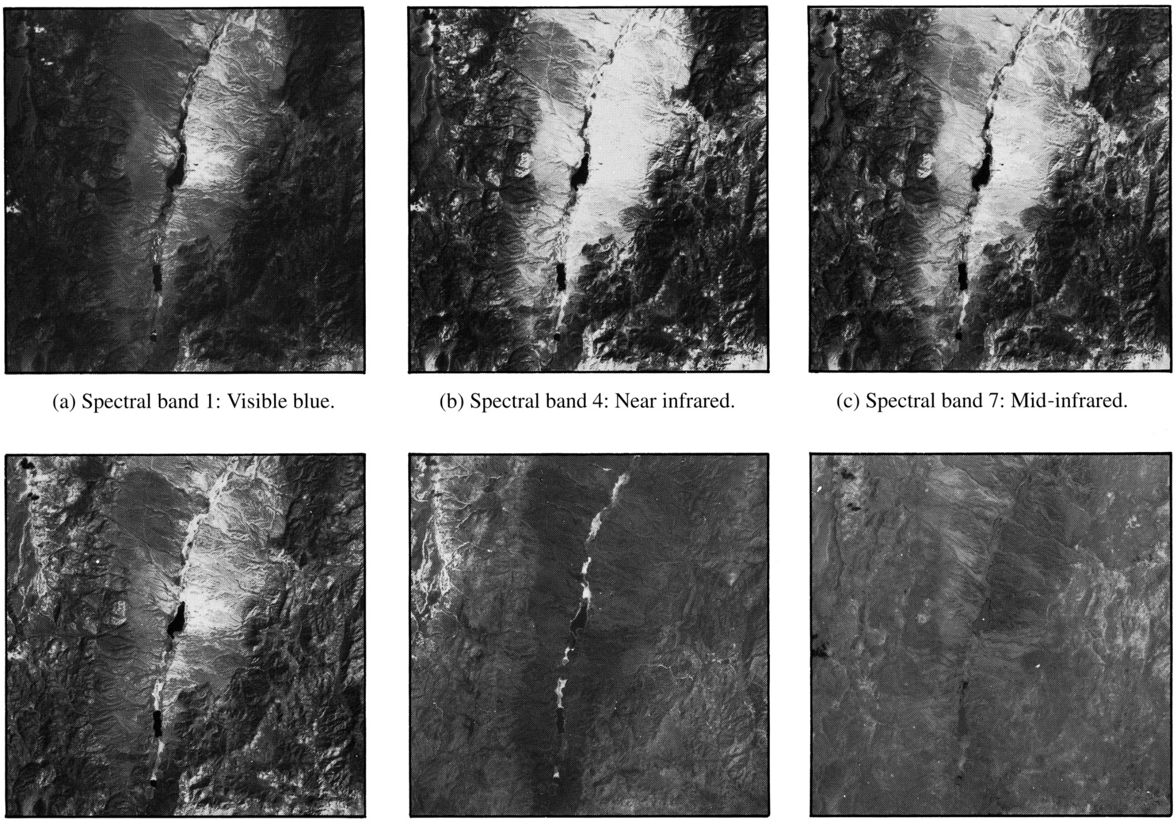
Around the world in little more than 80 minutes, the two Landsat satellites streak silently across the sky in near polar orbits, recording images of terrain and coastline, in swaths 185 kilometres wide. Every 16 days, each satellite passes over almost every square kilometre of the earth's surface, so any location can be monitored every 8 days.

Sensors aboard the satellite acquire seven simultaneous images of any region on earth to be studied. The sensors record energy from separate wavelength bands— three in the visible light spectrum and four in infrared and thermal bands. Each image is digitised and stored as a rectangular array of numbers, each number indicating the signal intensity at a corresponding small point (or pixel) on the image. Each of the seven images is one channel of a multichannel or multispectral image.

The seven Landsat images of one fixed region typically contain much redundant information, since some features will appear in several images. Yet other features, because of their colour or temperature, may reflect light that is recorded by only one or two sensors. One goal of multichannel image processing is to view the data in a way that extracts information better than studying each image separately.

Principal component analysis is an effective way to suppress redundant information and provide in only one or two composite images most of the information from the initial data. Roughly speaking, the goal is to find a special linear combination of the images, that is, a list of weights that at each pixel combine all seven corresponding image values into one new value. The weights are chosen in a way that makes the range of light intensities – the scene variance – in the composite image (called the first principal component) greater than that in any of the original images.

Principal component analysis is illustrated in the photos below, taken over Railroad Valley, Nevada. Images from three Landsat spectral bands are shown in Figure 1.1 (a)–(c). The total information in the three bands is rearranged in the three principal component images in Figure 1.1 (d)–(f). The first component (d) displays (or "explains") 93.5% of the scene variance present in the initial data. In this way, the three-channel initial data have been reduced to one-channel data, with a loss in some sense of only 6.5% of the scene variance. Earth Satellite Corporation of Rockville, Maryland, which kindly supplied the photos shown here, is experimenting with images from 224 separate spectral bands. Principal component analysis, essential for such massive data sets, typically reduces the data to about 15 usable principal components.



(a) Spectral band 1: Visible blue. (b) Spectral band 4: Near infrared. (c) Spectral band 7: Mid-infrared.

(d) Principal component 1: 93.5%. (e) Principal component 2: 5.3%. (f) Principal component 3: 1.2%.

Figure 1.1: Principal component analysis of satellite images over Railroad Valley, Nevada.

Symmetric matrices arise more often in applications, in one way or another, than any other major class of matrices. The theory is rich and beautiful, depending in an essential way on both diagonalisation and orthogonality. The diagonalisation of a symmetric matrix, described in Section 1.1, is the foundation for the discussion in Sections 1.2 and 1.3 concerning quadratic forms. Section 1.3, in turn, is needed for the final two sections on the singular value decomposition and on the image processing described in the introductory example. Throughout the chapter, all vectors and matrices have real entries.

1.2 Symmetric matrices

A symmetric matrix is a matrix A such that $A^T = A$ .Such a matrix is necessarily square. Its main diagonal entries are arbitrary, but its other entries occur in pairs – on opposite sides of the main diagonal.

There are some relevant properties of symmetric matrices.

Theorem 1.1. *If A is symmetric, then any two eigenvectors from different eigenspaces are orthogonal.*

Proof. Exercise. □

An $n \times n$ matrix A is said to be **orthogonally diagonalisable** if there are an orthogonal matrix P (with $P^{-1} = P^T$) and a diagonal matrix D such that

$$A = PDP^T = PDP^{-1}. \tag{1.1}$$

Exercise 1.1. *Show that if A is orthogonally diagonalisable, then A is symmetric.*

Theorem 1.2 below shows that, conversely, every symmetric matrix is orthogonally diagonalisable. The proof is much harder and is omitted; the main idea for a proof is given in the proof of Theorem 1.3.

Theorem 1.2. *An $n \times n$ matrix A is orthogonally diagonalisable if and only if A is a symmetric matrix.*

The set of eigenvalues of a matrix A is called the **spectrum** of A, and the following description of the eigenvalues is called a spectral theorem.

Theorem 1.3 (Spectral theorem for symmetric matrices)**.** *Let A be an $n \times n$ real symmetric matrix. Then:*

(a) A has n real eigenvalues, counted with multiplicities.

(b) For each eigenvalue λ, the dimension of its eigenspace (the geometric multiplicity) equals the multiplicity of λ as a root of the characteristic polynomial (the algebraic multiplicity).

(c) Eigenvectors corresponding to distinct eigenvalues are mutually orthogonal.

(d) A is orthogonally diagonalisable; that is, there exists an orthogonal matrix P such that $P^T AP$ is diagonal.

Proof. Part (a) is left as an exercise. Part (b) follows from part (d) as follows: if $A = P^T DP$ with P orthogonal and D diagonal, then $A - \lambda I = P^T(D - \lambda I)P$. Since P is invertible,

$$\dim \mathrm{Nul}(A - \lambda I) = \dim \mathrm{Nul}(D - \lambda I),$$

and for a diagonal matrix D, $\mathrm{Nul}(D - \lambda I)$ has dimension equal to the number of times λ appears on the diagonal. Hence, the geometric multiplicity of λ equals its algebraic multiplicity. Part (c) is a standard property of symmetric matrices (see Theorem 1.1): eigenvectors corresponding to distinct eigenvalues are orthogonal. For part (d), by Schur's Theorem, there exists an orthogonal matrix Q such that $Q^T AQ = T$ is upper triangular. Since A is symmetric,

$$T^T = (Q^T AQ)^T = Q^T A^T Q = Q^T AQ = T,$$

so T is both symmetric and upper triangular, which implies that T is diagonal. Therefore, A is orthogonally diagonalisable. $\square$

Suppose $A = PDP^{-1}$, where the columns of P are orthonormal eigenvectors $u_1, \dots, u_n$ of A and the corresponding eigenvalues $\lambda_1, \dots, \lambda_n$ are in the diagonal matrix D. Then, since $P^{-1} = P^T$,

$$A = PDP^T = \begin{bmatrix} u_1 & \cdots & u_n \end{bmatrix} \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix} \begin{bmatrix} u_1^T \\ \vdots \\ u_n^T \end{bmatrix} = \lambda_1 u_1 u_1^T + \cdots + \lambda_n u_n u_n^T. \tag{1.2}$$

This representation of A is called the **spectral decomposition** (or the **eigenvalue decomposition**) of A because it breaks up A into pieces determined by the spectrum (eigenvalues) of A. Each term in 1.2 is an $n \times n$ matrix of rank 1. For example, every column of $\lambda_1 u_1 u_1^T$ is a multiple of u_1 (check with an example). Furthermore, each matrix $u_j u_j^T$ is a **projection matrix** in the sense that for each $x \in \mathbb{R}^n$, the vector $(u_j u_j^T)x$ is the orthogonal projection of x onto the subspace spanned by u.

Remark 1.1 (Orthogonal projections)**.** *Given a nonzero vector $u \in \mathbb{R}^n$, consider the problem of decomposing a vector $y \in \mathbb{R}^n$ into the sum of two vectors, one a multiple of u and the other orthogonal to u. We wish to write*

$$y = \hat{y} + z \tag{1.3}$$

where $\hat{=}\alpha u$ for some scalar α and z is some vector orthogonal to u. Given any scalar α, let $z = y - \alpha u$, so that (1.3) *is satisfied. Then $y - \hat{y}$ is orthogonal to u if and only if*

$$(y - \hat{y})^T u = y^T u - \alpha u^T u = 0.$$

That is, (1.3) *is satisfied with z orthogonal to u if and only if*

$$\alpha = \frac{y^T u}{u^T u} \quad \textit{and} \quad \hat{y} = \frac{y^T u}{u^T u} u.$$

The vector $\hat{y}$ is called the ***orthogonal projection*** *of y onto u, and the vector z is called the* ***component of*** *y* ***orthogonal to*** *u. The* ***Orthogonal Decomposition Theorem*** *states the following: let W be a subspace of $\mathbb{R}^n$. Then each y in $\mathbb{R}^n$ can be written uniquely in the form* (1.3) *where $\hat{y}$ is in W and z is in $W^\perp$. In fact, if $\{u_1, \dots, u_r\}$ is any orthogonal basis of W, then*

$$\hat{y} = \frac{y^T u_1}{u_1^T u_1} u_1 + \cdots + \frac{y^T u_r}{u_r^T u_r} u_r$$

and $z = y - \hat{y}$. The vector $\hat{y}$ is called the ***orthogonal projection*** *of y onto W and the scalars $\alpha_j = \frac{y^T u_j}{u_j^T u_j}$, $j = 1, \dots, r$, are the* ***components of*** *y* ***in the basis*** *$\{u_1, \dots, u_r\}$.* □

Exercise 1.2. *Let u be a unit vector in $\mathbb{R}^n$, and let $B = uu^T$.*

(a) Given any x in $\mathbb{R}^n$, compute Bx and show that Bx is the orthogonal projection of x onto u.

(b) Show that B is a symmetric matrix and $B^2 = B$.

(c) Show that u is an eigenvector of B. What is the corresponding eigenvalue?

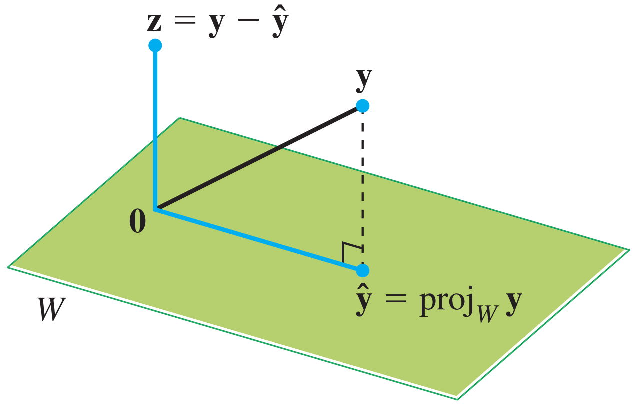


Figure 1.2: Orthogonal projection.

Exercise 1.3. *Let B be an $n \times n$ symmetric matrix such that $B^2 = B$. Any such matrix is called a* ***projection matrix*** *(or an* ***orthogonal projection matrix****). Given any y in $\mathbb{R}^n$, let $\hat{y} = By$ and $z = y - \hat{y}$.*

(a) Show that z is orthogonal to $\hat{y}$.

(b) Let W be the column space of B. Show that y is the sum of a vector in W and a vector in $W^{\perp}$. Why does this prove that By is the orthogonal projection of y onto the column space of B?

In MATLAB, computing the spectral decomposition is straightforward.

Listing 1.1: Spectral decomposition in MATLAB

```
% Create a random symmetric matrix 4x4
X =rand(4); A = X'*X;

% Compute the eigenvalues and eigenvectors; V contains the
   eigenvectors, D contains the eigenvalues on the diagonal
[P, D] = eig(A);

% Check the results (A= PDP')
A == P*D*P'
```

1.3 Quadratic forms

A **quadratic form** on $\mathbb{R}^n$ is a function Q defined on $\mathbb{R}^n$ whose value at a vector $x \in \mathbb{R}^n$ can be computed by an expression of the form $Q(x) = x^T Ax$, where A is an $n \times n$ symmetric matrix. The matrix A is called the **matrix of the quadratic form**. The simplest example of a nonzero quadratic form is $Q(x) = x^T Ax = \|x\|$.

Exercise 1.4. *Let* $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$. *Compute* $x^T Ax$ *for the following matrices:*

(a) $A = \begin{bmatrix} 4 & 0 \\ 0 & 3 \end{bmatrix}$; *(b)* $A = \begin{bmatrix} 3 & -2 \\ -2 & 7 \end{bmatrix}$.

Exercise 1.5. *For* $x \in \mathbb{R}^3$, *let* $Q(x) = 5x_1^2 + 3x_2^2 + 2x_3^3 - x_1x_2 + 8x_2x_3$. *Write this quadratic form as* $x^T Ax$.

In some cases, quadratic forms are easier to use when they have no cross-product terms – that is, when the matrix of the quadratic form is a diagonal matrix. Fortunately, the cross-product term can be eliminated by making a suitable change of variable.

If x represents a variable vector in $\mathbb{R}^n$, then a **change of variable** is an equation of the form

$$x = Py \quad \Leftrightarrow \quad y = P^{-1}x,$$

where P is an invertible matrix and y is a new variable vector in $\mathbb{R}^n$. Here y is the coordinate vector of x relative to the basis of $\mathbb{R}^n$ determined by the columns of P.

If the change of variable is made in a quadratic form $x^T Ax$, then

$$x^T Ax = (Py)^T A(Py) = y^T (P^T AP)y$$

and the new matrix of the quadratic form is $P^T AP$. Since A is symmetric, there is an orthogonal matrix P such that $P^T AP$ is a diagonal matrix D, and the quadratic form in becomes $y^T Dy$.

Exercise 1.6. *Make a change of variable that transforms the quadratic form in Exercise 1.4 into a quadratic form with no cross-product term.*

Solution. The matrix of the quadratic form in Exercise 1.4 is

$$A = \begin{bmatrix} 1 & -4 \\ -4 & -5 \end{bmatrix}.$$

The first step is to orthogonally diagonalise A. Its eigenvalues turn out to be

$$\lambda_1 = 3 \quad \text{and} \quad \lambda_2 = -7.$$

Associated unit eigenvectors are

$$v_1 = \frac{1}{\sqrt{5}} \begin{bmatrix} 2 \\ 11 \end{bmatrix}, \quad v_2 = \frac{1}{\sqrt{5}} \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$

These vectors are automatically orthogonal (because they correspond to distinct eigenvalues) and therefore provide an orthonormal basis for $\mathbb{R}^2$. Let

$$P = \begin{bmatrix} \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \\ \frac{-1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \end{bmatrix}, \quad D = \begin{bmatrix} 3 & 0 \\ 0 & -7 \end{bmatrix}.$$

Then $A = PDP^{-1}$ and $D = P^{-1}AP = P^TAP$, as noted earlier.

A suitable change of variable is

$$x = Py, \quad \text{where } x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \; y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}.$$

Then

$$x_1^2 - 8x_1x_2 - 5x_2^2 = x^TAx = (Py)^TA(Py) = y^TP^TAPy = y^TDy = 3y_1^2 - 7y_2^2.$$

The quadratic form in the new variables y_1 and y_2 has no cross-product term. $\square$

To illustrate the meaning of the equality of quadratic forms in the previous exercise, we can compute $Q(x)$ for $x = \begin{bmatrix} 2 \\ -2 \end{bmatrix}$ using the new quadratic form.

First, since $x = Py$, we have $y = P^{-1}x = P^Tx$, so

$$y = P^Tx = \begin{bmatrix} \frac{2}{\sqrt{5}} & \frac{-1}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \end{bmatrix} \begin{bmatrix} 2 \\ -2 \end{bmatrix} = \begin{bmatrix} \frac{6}{\sqrt{5}} \\ \frac{-2}{\sqrt{5}} \end{bmatrix}.$$

Then the quadratic form in the y-variables is

$$Q(x) = 3y_1^2 - 7y_2^2 = 3\left(\frac{6}{\sqrt{5}}\right)^2 - 7\left(-\frac{2}{\sqrt{5}}\right)^2 = 3 \cdot \frac{36}{5} - 7 \cdot \frac{4}{5} = 16.$$

Thus, $Q(x) = 16$ when $x = \begin{bmatrix} 2 \\ -2 \end{bmatrix}$. See Figure 1.3.

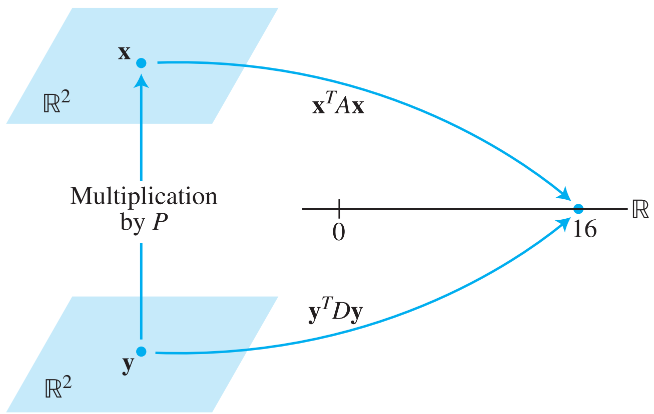


Figure 1.3: Change of variable in x^TAx.

Theorem 1.4 (Principal Axes Theorem)**.** *Let A be an $n \times n$ symmetric matrix. Then there is an orthogonal change of variable, $x = Py$, that transforms the quadratic form x^TAx into a quadratic form y^TDy with no cross-product term.*

The columns of P in the theorem are called the **principal axes** of the quadratic form x^TAx. The vector y is the coordinate vector of x relative to the orthonormal basis of $\mathbb{R}^n$ given by these principal axes.

When A is an $n \times n$ matrix, the quadratic form $Q(x) = x^T Ax$ is a real-valued function with domain $\mathbb{R}^n$. For each point $x = (x_1, x_2)$ in the domain of a quadratic form Q, the graph displays the point

$$(x_1, x_2, z) \quad \text{where} \quad z = Q(x).$$

Notice that, except at $x = 0$, the values of $Q(x)$ are all positive in Figure 4(a) and all negative in Figure 4(d). The horizontal cross-sections of the graphs are ellipses in Figures 4(a) and 4(d), and hyperbolas in Figure 4(c).

Definition 1.1. *A quadratic form Q is:*

(a) ***positive definite*** *if $Q(x) > 0$ for all $x \neq 0$,*

(b) ***negative definite*** *if $Q(x) < 0$ for all $x \neq 0$,*

(c) ***indefinite*** *if $Q(x)$ assumes both positive and negative values.*

Also, Q is said to be **positive semidefinite** if $Q(x) \geq 0$ for all x, and **negative semidefinite** if $Q(x) \leq 0$ for all x.

Next theorem characterises quadratic forms in terms of eigenvalues.

Theorem 1.5 (Quadratic forms and eigenvalues)**.** *Let A be an $n \times n$ symmetric matrix. Then the quadratic form $Q(x) = x^T Ax$ is*

(a) positive definite *if and only if the eigenvalues of A are all positive,*

(b) negative definite *if and only if the eigenvalues of A are all negative, or*

(c) indefinite *if and only if A has both positive and negative eigenvalues.*

Proof. By the Principal Axes Theorem, there exists an orthogonal change of variables $x = Py$ such that

$$Q(x) = x^T Ax = y^T Dy = \lambda_1 y_1^2 + \lambda_2 y_2^2 + \cdots + \lambda_n y_n^2, \tag{1.4}$$

where $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of A. Since P is invertible, there is a one-to-one correspondence between all nonzero vectors x and all nonzero vectors y. Thus, the values of $Q(x)$ for $x \neq 0$ coincide with the values of the expression on the right-hand side of (1.4), which is clearly determined by the signs of the eigenvalues $\lambda_1, \ldots, \lambda_n$, in the three cases described in the theorem. □

Example 1.1. *Is the quadratic form $Q(x) = 3x_1^2 + 2x_2^2 + x_3^2 + 4x_1x_2 + 4x_2x_3$ positive definite? Because of all the plus signs, this form may appear to be positive definite. However, the symmetric matrix of the form is*

$$A = \begin{bmatrix} 3 & 2 & 0 \\ 2 & 2 & 2 \\ 0 & 2 & 1 \end{bmatrix},$$

and the eigenvalues of A turn out to be 5, 2, and -1. Therefore the quadratic form is indefinite, *not positive definite.* □

The classification of a quadratic form is often carried over to the associated matrix. Thus, a *positive definite matrix* A is a symmetric matrix for which the quadratic form $x^T Ax$ is positive definite. Other terms, such as *positive semidefinite matrix*, are defined analogously.

Remark 1.2 (Numerical note). *A fast way to determine whether a symmetric matrix A is positive definite is to attempt to factor A in the form $A = R^T R$, where R is upper triangular with positive diagonal entries. (A slightly modified algorithm for an LU factorisation is one approach.) Such a* ***Cholesky factorisation*** *is possible if and only if A is positive definite.* □

Listing 1.2: Classify a symmetric matrix efficiently

```
function classification = classifySymmetric(A)
%CLASSIFYSYMMETRIC Classify a symmetric matrix efficiently.
%   Returns a string: 'PD', 'PSD', 'ND', 'NSD', or 'Indefinite'

% Ensure matrix is symmetric
if ~isequal(A, A')
    error('Matrix must be symmetric.');
end

% Fast check for positive definite using Cholesky
[~, p] = chol(A);
if p == 0
    classification = 'PD';  % Positive Definite
    return
end

% Compute eigenvalues for all other classifications
eigvals = eig(A);

if all(eigvals >= 0)
    classification = 'PSD'; % Positive Semidefinite
elseif all(eigvals <= 0)
    if any(eigvals < 0)
        classification = 'ND';  % Negative Definite
    else
        classification = 'NSD'; % Negative Semidefinite
    end
else
    classification = 'Indefinite';
end
end
```

Exercise 1.7. *Let A be an $n \times n$ invertible symmetric matrix. Show that if the quadratic form $Q(x) = x^T Ax$ is positive definite, then the quadratic form $Q_{inv}(x) = x^T A^{-1}x$ is also positive definite. Hint: Consider the eigen-decomposition of A.*

1.4 Constrained optimisation

Often, we are interested in finding the *maximum* or *minimum* of a quadratic form

$$Q(x) = x^T Ax$$

subject to a constraint on the vector $x \in \mathbb{R}^n$. A particularly important case is when x is restricted to be a unit vector, which can be expressed in several equivalent ways:

$$\|x\| = 1, \quad \|x\|_2 = 1, \quad x^T x = 1,$$

or, in expanded form,

$$x_1^2 + x_2^2 + \cdots + x_n^2 = 1. \tag{1.5}$$

The expanded form (1.5) is often used in applications because it explicitly shows the contribution of each coordinate.

Remark 1.3 (Connection to the Rayleigh quotient)**.** *The* ***Rayleigh quotient*** *of a real $n \times n$ matrix A and a nonzero vector $x \in \mathbb{R}^n$ is defined as*

$$R(x) = \frac{x^T Ax}{x^T x}.$$

Since $R(\alpha x) = R(x)$ for any scalar $\alpha \neq 0$, it suffices to consider vectors x on the unit sphere, $\|x\| = 1$, where the Rayleigh quotient coincides with the quadratic form

$$R(x) = x^T Ax.$$

Consequently, finding the maximum or minimum of the quadratic form over unit vectors is equivalent to finding the maximum or minimum of the Rayleigh quotient. □

Denote the left and right endpoints of this interval by m and M, respectively. That is, let

$$m = \min\{x^T Ax \mid \|x\| = 1\}, \quad M = \max\{x^T Ax \mid \|x\| = 1\}. \tag{1.6}$$

Exercise 1.8. *Show that that for any symmetric matrix A, the set of all possible values of*

$$x^T Ax, \quad \textit{for } \|x\| = 1,$$

is a closed interval on the real axis.

Solution. In fact, for a symmetric matrix A, it is well known that the Rayleigh quotient satisfies

$$\lambda_{\min} \leq R(x) \leq \lambda_{\max},$$

where $\lambda_{\min}$ and $\lambda_{\max}$ are the smallest and largest eigenvalues of A, and the extremal values are attained when x is an eigenvector corresponding to these eigenvalues. In this way, the Rayleigh quotient provides a direct connection between the quadratic form and the eigenvalues of A. Restricting to the unit sphere, $\|x\| = 1$, we have $R(x) = x^T Ax$. Since the unit sphere is compact and $R(x)$ is continuous, by the intermediate value theorem, all values between the minimum and maximum are attained. Hence,

$$\{x^T Ax : \|x\| = 1\} = [\lambda_{\min}, \lambda_{\max}],$$

a closed interval on the real axis. □

Exercise 1.9. *Show that every value between m and M is attained by the Rayleigh quotient for some unit vector.*

Solution. Let u_n and u_1 be orthonormal eigenvectors of A with eigenvalues m and M, respectively. For any $t \in [m, M]$, define

$$\alpha = \frac{t-m}{M-m} \in [0,1], \qquad x = \sqrt{1-\alpha}\, u_n + \sqrt{\alpha}\, u_1.$$

Since $u_n^T u_1 = 0$ and $\|u_n\| = \|u_1\| = 1$, we have $\|x\|^2 = 1$. Moreover, using $Au_1 = Mu_1$ and $Au_n = mu_n$,

$$x^T Ax = (1-\alpha)m + \alpha M = t.$$

Hence, for each $t \in [m, M]$, there exists a unit vector x such that $x^T Ax = t$. □

Listing 1.3: MATLAB code for constructing x for a given t in between m and M

```
%% Define a symmetric matrix A
A = [4 2 0; 2 3 0; 0 0 5]; % example 3x3 symmetric matrix

% Eigen-decomposition
[V,D] = eig(A);
[eigenvals, idx] = sort(diag(D), 'descend'); % descending
V = V(:, idx);

M = eigenvals(1); % max eigenvalue
m = eigenvals(end); % min eigenvalue
u1 = V(:,1); % eigenvector for max
un = V(:,end); % eigenvector for min

fprintf('Maximum eigenvalue M = %.2f\n', M);
fprintf('Minimum eigenvalue m = %.2f\n', m);

%% Choose a value t between m and M
t = (m + M)/2; % example: midpoint
alpha = (t - m)/(M - m); % compute alpha such that t = (1-alpha)*m
    + alpha*M

%% Construct x as convex combination of u1 and un
x = sqrt(alpha)*u1 + sqrt(1-alpha)*un;

%% Normalize to ensure x is unit vector (numerical safety)
x = x / norm(x);

%% Verify
t_calc = x' * A * x;
fprintf('Chosen t = %.2f\n', t);
fprintf('Calculated x^T A x = %.4f\n', t_calc);
fprintf('Norm of x = %.4f\n', norm(x));
```

The next theorem strengthens this result by showing that m and M are themselves eigenvalues of A, just as in the previous example.

Theorem 1.6 (Extremal values of quadratic forms). *Let A be a symmetric $n \times n$ matrix. Then*

$$m = \min_{\|x\|=1} x^T Ax \quad \text{and} \quad M = \max_{\|x\|=1} x^T Ax$$

are eigenvalues of A. Moreover, the minima and maxima are attained at corresponding eigenvectors.

The following MATLAB script, together with the subsequent theorem, illustrates and formalises the relationship between the extremal values of a quadratic form on the unit sphere and the eigenvalues of the matrix that defines it.

Listing 1.4: MATLAB code for random symmetric 2×2 matrix with eigenvectors plotted

```
% Generate a random symmetric 2x2 matrix
A = randn(2,2); A = (A + A')/2;

% Compute eigenvalues and eigenvectors
[V,D] = eig(A); lambda = diag(D);
[lambda_max, idx_max] = max(lambda);
[lambda_min, idx_min] = min(lambda);
u_max = V(:,idx_max);   % eigenvector for max
u_min = V(:,idx_min);   % eigenvector for min

% Normalize eigenvectors to have norm 1
u_max = u_max / norm(u_max); u_min = u_min / norm(u_min);

% Sample unit vectors on the circle
theta = linspace(0,2*pi,200);
x = [cos(theta); sin(theta)];  % 2xN unit vectors

% Compute quadratic form for each sample
Q = zeros(1,length(theta));
for k = 1:length(theta)
    Q(k) = x(:,k)'*A*x(:,k);
end

%% Plot unit circle colored by quadratic form
figure('Color','w');

% Scatter points (do NOT appear in legend)
scatter(x(1,:),x(2,:), 50, Q, 'filled', 'HandleVisibility','off');
     hold on
colormap(jet); colorbar
axis equal; grid on
xlabel('x_1'); ylabel('x_2')
title('Unit circle colored by x^T A x')

% Plot eigenvectors as arrows (legend only for these)
```

```
h1 = quiver(0,0,u_max(1),u_max(2),0,'r','LineWidth',2,'MaxHeadSize
    ',0.5);
h2 = quiver(0,0,u_min(1),u_min(2),0,'b','LineWidth',2,'MaxHeadSize
    ',0.5);

% Add legend only for arrows
legend([h1,h2], {'Max eigenvector','Min eigenvector'})

%% Plot quadratic form values vs angle
figure('Color','w');
plot(theta, Q, 'LineWidth', 2); hold on
yline(lambda_max,'r--','LineWidth',2,'Label','\lambda_{max}');
yline(lambda_min,'b--','LineWidth',2,'Label','\lambda_{min}');
xlabel('\theta'); ylabel('x^T A x')
title('Quadratic form values on the unit circle')
grid on
```

Theorem 1.7. *Let A be a symmetric $n \times n$ matrix, and define*

$$m = \min_{\|x\|=1} x^T Ax, \quad M = \max_{\|x\|=1} x^T Ax.$$

Then M is the greatest eigenvalue λ_1 of A and m is the least eigenvalue λ_n. The value $x^T Ax$ attains M when x is a unit eigenvector u_1 corresponding to λ_1, and it attains m when x is a unit eigenvector u_n corresponding to λ_n.

Proof. Since A is symmetric, it admits an orthogonal diagonalization $A = PDP^T$, where P is an orthogonal matrix whose columns $u_1, \dots, u_n$ are orthonormal eigenvectors of A, and $D = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$ with eigenvalues ordered so that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$.

For any vector $x \in \mathbb{R}^n$, write $x = Py$. Because P is orthogonal,

$$\|x\| = \|Py\| = \|y\|.$$

Moreover,

$$x^T Ax = (Py)^T A(Py) = y^T P^T (PDP^T)Py = y^T Dy.$$

Thus

$$x^T Ax = y^T Dy = \sum_{i=1}^{n} \lambda_i y_i^2, \qquad \text{whenever } \|x\| = \|y\| = 1.$$

Let y be any unit vector. Since $y_1^2 + \cdots + y_n^2 = 1$ and $\lambda_1 \geq \lambda_i$ for all i,

$$y^T Dy = \sum_{i=1}^{n} \lambda_i y_i^2 \leq \sum_{i=1}^{n} \lambda_1 y_i^2 = \lambda_1.$$

Thus

$$\max_{\|x\|=1} x^T Ax \leq \lambda_1.$$

But equality is achieved by choosing $y = e_1 = (1, 0, \dots, 0)^T$, for which

$$y^T Dy = \lambda_1.$$

The corresponding x is

$$x = Pe_1 = u_1,$$

the unit eigenvector associated with λ_1. Hence

$$M = \max_{\|x\|=1} x^T Ax = \lambda_1.$$

A completely analogous argument, using $\lambda_n \leq \lambda_i$ for all i, shows that

$$m = \min_{\|x\|=1} x^T Ax = \lambda_n,$$

with the minimum attained at $x = u_n$. This completes the proof. $\square$

Exercise 1.10. *Let*

$$A = \begin{bmatrix} 3 & 2 & 1 \\ 2 & 3 & 1 \\ 1 & 1 & 4 \end{bmatrix}.$$

Find the maximum value of $x^T Ax$ subject to $\|x\| = 1$ and a corresponding unit vector.

In the next theorem and in later applications, the values of $x^T Ax$ are computed with additional constraints on the unit vector x.

Theorem 1.8. *Let A, λ_1, and u_1 be as in Theorem 1.7. Then the maximum of $x^T Ax$ subject to*

$$\|x\| = 1, \quad x^T u_1 = 0$$

is the second greatest eigenvalue λ_2, attained when x is the corresponding eigenvector u_2.

The theorem can be proved by an argument similar to the one above in which the theorem is reduced to the case where the matrix of the quadratic form is diagonal. The next exercise gives an idea of the proof for the case of a diagonal matrix.

Exercise 1.11. *Find the maximum of $Q(x) = 9x_1^2 + 4x_2^2 + 3x_3^2$ subject to*

$$\|x\| = 1, \quad x^T u_1 = 0, \quad u_1 = (1, 0, 0)^T.$$

Solution. The constraint $x^T u_1 = 0$ implies $x_1 = 0$. Then $x_2^2 + x_3^2 = 1$ and

$$Q(x) = 4x_2^2 + 3x_3^2 \leq 4(x_2^2 + x_3^2) = 4.$$

Equality occurs at $x = (0, 1, 0)^T$, which is an eigenvector corresponding to $\lambda_2 = 4$. $\square$

Exercise 1.12. *Let A be as in Exercise 1.10 and let u_1 be a unit eigenvector corresponding to $\lambda_1 = 6$. Find the maximum of $x^T Ax$ subject to*

$$\|x\| = 1, \quad x^T u_1 = 0.$$

Solution. The second greatest eigenvalue is $\lambda_2 = 3$. Solving $(A-3I)x = 0$ and normalizing gives the eigenvector

$$u_2 = \frac{1}{\sqrt{6}}(2, 1, -2)^T.$$

This vector is automatically orthogonal to u_1, and $x^T Ax = 3$ at $x = u_2$. □

The next theorem generalises the previous one and, together with Theorem 1.7, gives a useful characterisation of all the eigenvalues of A. The proof is omitted.

Theorem 1.9. *Let A be a symmetric $n \times n$ matrix with orthogonal diagonalisation $A = PDP^{-1}$, where D has eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ and $P = [u_1, \ldots, u_n]$. Then, for $k = 2, \ldots, n$, the maximum of $x^T Ax$ subject to*

$$\|x\| = 1, \quad x^T u_1 = 0, \ldots, x^T u_{k-1} = 0$$

is λ_k, attained at $x = u_k$.

1.5 An application: public works allocation

Suppose a county must allocate resources between road repair x (hundred miles) and parks y (hundred acres), subject to

$$4x^2 + 9y^2 \leq 36.$$

See Figure 1.4. Each point (x, y) in the shaded feasible set represents a possible public works schedule for the year. The points on the constraint curve, $4x^2 + 9y^2 = 36$, use the maximum amounts of resources available.

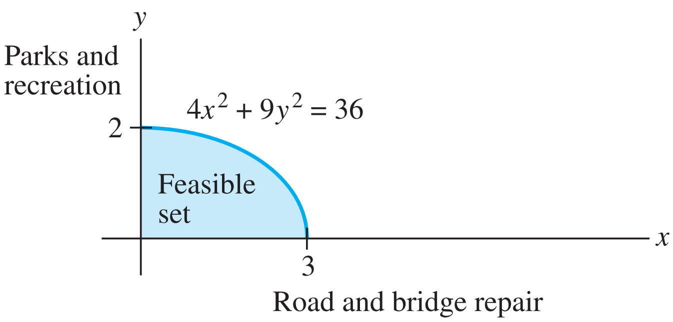


Figure 1.4: Public works schedules.

In choosing its public works schedule, the county wants to consider the opinions of the county residents. To measure the value, or utility, that the residents would assign to the various work schedules (x, y), economists sometimes use a utility function such as

$$q(x, y) = xy.$$

The set of points (x, y) at which $q(x, y)$ is a constant is called an indifference curve. Points along an indifference curve correspond to alternatives that county residents as a group would find equally valuable. See Figure 1.5.

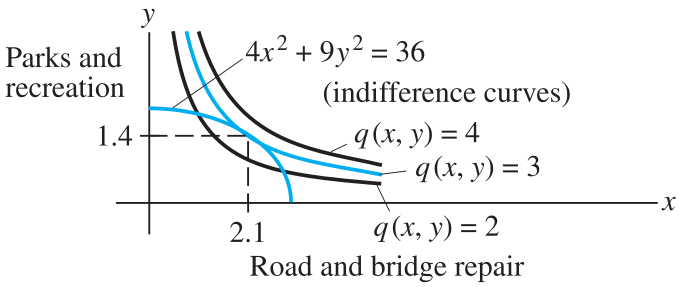


Figure 1.5: The optimum public works schedule is $(2., 1.4)$.

To maximise a utility function $q(x, y) = xy$, we first rescale

$$x_1 = x/3, \quad x_2 = y/2, \quad \text{so that } x_1^2 + x_2^2 = 1.$$

Then

$$q(x, y) = xy = 3x_1 \cdot 2x_2 = 6x_1x_2 = Q(x),$$

which is a quadratic form.

The matrix of Q is

$$A = \begin{bmatrix} 0 & 3 \\ 3 & 0 \end{bmatrix},$$

with eigenvalues 3 and -3, with eigenvectors $\begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$, and $\begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$, respectively. The maximum occurs along the eigenvector corresponding to $\lambda = 3$, giving $x_1 = x_2 = 1/\sqrt{2}$. Transforming back,

$$x = 3x_1 = \frac{3}{\sqrt{2}} \approx 2.1 \text{ hundred miles}, \quad y = 2x_2 = \sqrt{2} \approx 1.4 \text{ hundred acres}.$$

Thus, the optimal allocation is the point $(x, y) = (1.5, 1)$. The optimum public works schedule is the point where the constraint curve and the indifference curve $q(x, y) = 3$ just meet. Points (x, y) with a higher utility lie on indifference curves that do not touch the constraint. curve. See Figure 1.5.

Exercise 1.13. *Let*

$$Q(x) = 3x_1^2 + 3x_2^2 + 2x_1x_2.$$

1. *Find a change of variables that transforms Q into a quadratic form with no cross-product term, and give the resulting quadratic form.*

2. *Find the maximum value of $Q(x)$ subject to the constraint $x^T x = 1$, and find a unit vector at which the maximum is attained.*

We illustrate an applied constrained optimisation problem using MATLAB/Octave.

Listing 1.5: MATLAB code for plotting quadratic forms and constrained optimization

```
%% Maximize q(x,y) = x*y under user-defined elliptical constraint
clc; clear; close all;

%% User inputs
alpha = 4;   % coefficient for x^2
beta  = 9;   % coefficient for y^2
delta = 36;  % right-hand side of ellipse

%% Step 1: Rescale variables to unit circle
sx = sqrt(delta/alpha); % x = sx * x1
sy = sqrt(delta/beta);  % y = sy * x2

% Quadratic form in rescaled variables: Q(x1,x2) = sx*sy*x1*x2
A = [0 sx*sy; sx*sy 0]; % symmetric 2x2 matrix

% Eigen-decomposition to find maximum
[V,D] = eig(A); [~, idx_max] = max(diag(D));
u_max = V(:,idx_max) / norm(V(:,idx_max));

% Map back to original variables
x_opt = sx * u_max(1); y_opt = sy * u_max(2);

%% Step 2: Plot feasible ellipse and indifference curves
theta = linspace(0,pi/2,250); % only first quadrant
x_ellipse = sx * cos(theta); y_ellipse = sy * sin(theta);
figure('Color','w'); hold on;

% Plot shaded feasible region (first quadrant)
h_fill = fill([0 x_ellipse 0], [0 y_ellipse 0], [0.9 0.9 0.9], '
    FaceAlpha',0.3, 'EdgeColor','k', 'LineWidth',2);

% Indifference curves q(x,y) = xy
[xg, yg] = meshgrid(linspace(0,max(x_ellipse)), linspace(0,max(
    y_ellipse)));
q_grid = xg.*yg;
contour(xg, yg, q_grid, 2:1:6,'ShowText','on','LineColor','b');

% Plot optimal point
h_opt = plot(x_opt, y_opt,'ro','MarkerSize',10,'MarkerFaceColor','
    r');
text(x_opt+0.1, y_opt, sprintf('Optimal (%.2f, %.2f)', x_opt,
    y_opt),'FontSize',10);

xlabel('Roads x'); ylabel('Parks y');
title('Maximizing q(x,y) = x*y under Elliptical Constraint (x>=0,
    y>=0)');
axis equal; grid on;
```

```
% Legend combining fill and optimal point
legend([h_fill, h_opt], {'Feasible region','Optimal point'}, '
    Location','best');

%% Step 3: Display results
fprintf('Optimal allocation:\n');
fprintf('x = %.3f\n', x_opt);
fprintf('y = %.3f\n', y_opt);
fprintf('Maximum utility q(x,y) = %.3f\n', x_opt * y_opt);
```

1.6 Exercises

Hand-solved exercises

1. **Symmetric matrices and eigenvalues.**

   (a) Prove that all eigenvalues of a symmetric matrix are real.

   (b) Show that eigenvectors corresponding to distinct eigenvalues of a symmetric matrix are orthogonal.

   (c) For
   $$A = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix},$$
   compute the eigenvalues and orthonormal eigenvectors.

2. **Quadratic forms.**

   (a) Let
   $$Q(x) = 4x_1^2 + 4x_1x_2 + x_2^2.$$
   Find a linear change of variables $x = Py$ that eliminates the cross term and express Q in terms of y.

   (b) Diagonalise a 2×2 symmetric matrix A by an orthogonal matrix P and show that the quadratic form x^TAx becomes y^TDy, where D is diagonal and $y = P^Tx$.

   (c) Consider the ellipse $x^TAx = 1$. Use the change of variables $y = P^Tx$ to transform it into a circle and identify the axes lengths.

3. **Orthogonal projections.**

   (a) Let $v \in \mathbb{R}^n$ and u a unit vector. Show that the orthogonal projection of v onto u is
   $$\text{proj}_u(v) = (u^Tv)u,$$
   and that the projection error $v - \text{proj}_u(v)$ is orthogonal to u.

   (b) Given the plane $x + 2y + z = 0$, find the matrix P that projects any vector in $\mathbb{R}^3$ orthogonally onto the plane.

   (c) For a symmetric matrix A, show that the Rayleigh quotient remains unchanged if x is projected onto the eigenspace corresponding to the largest eigenvalue.

4. **Constrained optimisation.**
   (a) Maximise $Q(x) = x^T Ax$ subject to $x^T x = 1$. Verify that the maximum occurs at the eigenvector corresponding to the largest eigenvalue.
   (b) Maximise $q(x, y) = xy$ subject to the constraint $\alpha x^2 + \beta y^2 \leq \delta$ using Lagrange multipliers.
   (c) Consider a 3×3 symmetric matrix A and the constraint $x^T x = 1$. Determine the vector that maximises $x^T Ax$ and the maximum value.

Matlab exercises

1. **Eigenvalue decomposition.**
   (a) Create a random symmetric 3×3 matrix. Compute eigenvalues and eigenvectors in MATLAB and verify $Av = \lambda v$.
   (b) Check orthogonality of eigenvectors numerically.
2. **Quadratic forms visualisation.**
   (a) For a 2×2 symmetric matrix A, plot $Q(x) = x^T Ax$ on the unit circle and color by value.
   (b) Add eigenvectors corresponding to maximum and minimum eigenvalues.
   (c) Compute and plot the Rayleigh quotient along a set of random unit vectors.
3. **Public works allocation.**
   (a) For $q(x, y) = xy$ and $\alpha x^2 + \beta y^2 \leq \delta$, plot the feasible ellipse.
   (b) Plot several indifference curves $q(x, y) = \text{constant}$ over the feasible set.
   (c) Compute the optimal (x, y) using eigenvalue decomposition of the quadratic form in rescaled variables.
   (d) Visualise the optimal point on the ellipse and contour plot.
4. **Portfolio optimisation.** Maximise return of a two-asset portfolio under risk constraints.
   (a) Let x and y denote fractions of the total investment in two assets.
   (b) The portfolio variance is given by a quadratic form
   $$\sigma^2 = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$
   (c) The objective is to maximise expected return $R(x, y) = r_1 x + r_2 y$ subject to
   $$x^2 + y^2 \leq \delta \quad \text{(risk constraint)}.$$
   (d) **Analytical tasks:** (i) Transform the constraint into the unit circle; (ii) Express return as a linear/quadratic form in the new variables; (iii) Find the allocation (x, y) that maximises return for the given risk.
   (e) **Computational tasks:** (i) Plot feasible risk region (ellipse); (ii) Plot level curves of portfolio return; (iii) Compute and visualise the optimal allocation; (iv) Experiment with changing ρ (correlation) and observe the impact on optimal allocation.

5. **Mechanical vibrations and principal axes.** In mechanical engineering, the vibration of a rigid body about a fixed point is described by the **inertia tensor**, which is a symmetric 3×3 matrix

$$I = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{yz} \\ I_{xz} & I_{yz} & I_{zz} \end{bmatrix}.$$

The rotational kinetic energy of the body can be written as a quadratic form

$$T = \frac{1}{2}\omega^T I\, \omega,$$

where ω is the angular velocity vector. Finding the **principal axes of rotation** amounts to finding an orthogonal change of variables that diagonalises I, so that in the new coordinates

$$T = \frac{1}{2}(\lambda_1\omega_1^2 + \lambda_2\omega_2^2 + \lambda_3\omega_3^2),$$

where $\lambda_1, \lambda_2, \lambda_3$ are the principal moments of inertia.

(a) Let $I = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 3 & 1 \\ 0 & 1 & 2 \end{bmatrix}$. Find the eigenvalues (principal moments) and orthonormal eigenvectors of I. Interpret the eigenvectors as the principal axes.

(b) Express the quadratic form $T = \frac{1}{2}\omega^T I\omega$ in the new coordinates along principal axes.

(c) Verify that the cross terms vanish in the transformed coordinates.

(d) For a fixed rotational energy T_0, describe the set of angular velocities ω satisfying $T = T_0$. Show how it transforms under the change of variables to a coordinate-aligned ellipsoid.

(e) Determine the angular velocity vector that maximises rotation about the largest principal axis for a fixed kinetic energy T_0.

(f) **Computational tasks:** Analyse and visualise a rigid body's rotational energy using orthogonal projections and principal axes.

   i. Input a symmetric inertia matrix I and compute its eigenvalues and eigenvectors.
   ii. Verify numerically that the eigenvectors are orthonormal and that $Iv = \lambda v$.
   iii. Plot the energy ellipsoid $T = T_0$ in 3D (original coordinates).
   iv. Apply the change of variables $\omega = Py$ using the eigenvectors of I, and plot the transformed ellipsoid (principal axes aligned).
   v. Visualise the principal axes and the optimal angular velocity vector.
   vi. Explore how changing the inertia tensor affects the shape of the ellipsoid and the optimal direction.

Chapter 2

Singular Value Decomposition

2.1 Unsupervised learning

In many real-world problems, datasets contain a large number of features and observations, and the underlying structure of the data is unknown. When explicit labels for supervision are not available, **unsupervised learning** aims to discover patterns, correlations, and inherent structures directly from the data. This makes it particularly useful for exploratory data analysis, feature extraction, dimensionality reduction, and clustering.

Unsupervised learning techniques are commonly grouped into two major categories:

1. **Clustering:** Partitioning data into groups (clusters) such that objects in the same cluster are similar according to a chosen measure of similarity. Examples include *k-means*, hierarchical clustering, and DBSCAN.

2. **Dimensionality Reduction:** Transforming data into a lower-dimensional representation while preserving relevant structure or variability. Methods include *Principal Component Analysis (PCA)*, *Singular Value Decomposition (SVD)*, *Independent Component Analysis (ICA)*, and *t-SNE*.

Dimensionality reduction methods are typically unsupervised because they operate solely on the input data without requiring labels. For instance, PCA identifies directions of maximum variance in the data, which can be used for visualisation, noise reduction, feature extraction, or as a preprocessing step before applying supervised learning algorithms.

Although not a purely unsupervised method, *Linear Discriminant Analysis (LDA)* is introduced alongside these techniques because it also generates a low-dimensional projection of the data. In contrast to PCA or SVD, which use no label information, *LDA incorporates class labels* (when available) and seeks linear combinations of features that *maximise class separability.* Thus, while PCA maximises retained variance, LDA maximises discrimination between groups, making it a *supervised dimensionality reduction* method and a natural bridge between dimensionality reduction and classification.

In this module, we will focus on three key methods:

- **Singular Value Decomposition (SVD):** Decomposes a data matrix X into its fundamental components,
$$X = U\Sigma V^T,$$
revealing latent structure and enabling optimal low-rank approximations. SVD is widely used for noise reduction, data compression, and identifying dominant patterns of variability.

- **Principal Component Analysis (PCA):** Identifies orthogonal directions (principal components) that capture the largest variance in the data. Dimensionality can be reduced by projecting the data onto the leading components. PCA can be computed efficiently using SVD.

- **Linear Discriminant Analysis (LDA):** Finds projections that best separate predefined classes. While supervised, it is included here for comparison because it shares the goal of finding meaningful lower-dimensional representations and illustrates how label information influences feature extraction.

Understanding these techniques provides a solid foundation for exploring more advanced unsupervised and semi-supervised learning methods.

The techniques of Singular Value Decomposition (SVD), Principal Component Analysis (PCA), and Linear Discriminant Analysis (LDA) are widely applied in modern data analysis. The following four examples illustrate their practical use in real-world scenarios.

Image Compression and Noise Reduction (SVD) Singular Value Decomposition (SVD) is widely used in image processing for compression and noise reduction. A grayscale image can be represented as a matrix of pixel intensities. By applying SVD, the image can be approximated using only the most significant components, effectively capturing the main structures and details while discarding minor variations associated with noise.

This approach reduces the storage requirements for images while preserving visual quality. It is used in applications such as JPEG image compression, enhancing scanned documents, and improving the quality of medical images such as X-rays or MRI scans. In addition to compression, SVD can also remove noise from images, producing clearer and cleaner visual data for analysis or display.

Facial Recognition (PCA) Principal Component Analysis (PCA), often referred to as *Eigenfaces* in computer vision, is a cornerstone of facial recognition systems. Each face image is treated as a high-dimensional vector by stacking its pixel values. PCA identifies the principal components of a dataset of faces, which correspond to the directions of maximum variance across images. Projecting a new face onto these principal components reduces the dimensionality and highlights the most discriminative features. Recognition is then performed by comparing the reduced representation of the new image to stored projections. PCA thus allows efficient storage, feature extraction, and comparison of face images, and it is widely used in security systems and social media tagging.

Genomic Data Analysis (PCA) In bioinformatics, PCA is essential for analysing large-scale genomic datasets, such as gene expression matrices from RNA sequencing experiments. Each row represents a gene and each column a sample or condition. PCA identifies patterns of variation that may correspond to biological factors such as tissue type, disease state, or population structure. By projecting high-dimensional data onto the first few principal components, researchers can visualise clusters of samples, detect outliers, and reduce the number of features for downstream analysis, such as classification or regression models. This approach is particularly valuable when dealing with thousands of genes and only a few dozen samples.

Medical Diagnosis and Classification (LDA) Linear Discriminant Analysis is commonly used in medical diagnostics to classify patients based on clinical measurements. For instance, in distinguishing between patients with a particular disease and healthy controls, each patient is represented as a vector of biomarker values. LDA finds the linear combination of features that maximises the separation between classes while minimising variance within each class. By projecting patient data onto this discriminative axis, one can construct a classifier that assigns new patients to the most likely class. LDA has been successfully applied in cancer diagnosis, cardiovascular risk assessment, and automated analysis of medical imaging data, often improving the interpretability of the decision boundaries in addition to classification performance.

Throughout this course, we will implement SVD, PCA, and LDA in MATLAB, leveraging built-in functions for matrix computation and visualisation, and applying the techniques to real datasets. By the end of this 3-week module, students will be able to: (i) Explain the purpose and scope of unsupervised learning; (ii) Apply SVD and PCA to real datasets for dimensionality reduction and exploratory analysis; (iii) Understand the principles of LDA and how it differs from purely unsupervised techniques; (iv) Implement all methods in MATLAB and analyse the resulting projections.

2.2 Motivation to SVD

The diagonalisation theorems in Linear Algebra play a part in many interesting applications. Unfortunately, as we know, not all matrices can be factored as $A = PDP^{-1}$ with D diagonal. However, a factorisation $A = QDP^{-1}$ is possible for any $m \times n$ matrix A! A special factorisation of this type, called the **singular value decomposition** (SVD), is one of the most useful matrix factorisations in applied linear algebra.

The singular value decomposition is based on the following property of ordinary diagonalisation that can be imitated for rectangular matrices: The absolute values of the eigenvalues of a symmetric matrix A measure the amount by which A stretches or shrinks certain vectors, called the **eigenvectors**. If $Ax = \lambda x$ and $\|x\| = 1$, then

$$\|Ax\| = \|\lambda x\| = |\lambda| \, \|x\| = |\lambda|.$$

If λ_1 is the eigenvalue with the greatest magnitude, then a corresponding unit eigenvector v_1 identifies a direction in which the stretching effect of A is greatest. That is, the length of Ax is maximised when $x = v_1$, and $\|Av_1\| = |\lambda_1|$ by the previous equation.

This description of v_1 and $|\lambda_1|$ has an analogue for rectangular matrices, which leads to the singular value decomposition.

Example 2.1 (Maximum length under a linear transformation)**.** *Let*

$$A = \begin{bmatrix} 4 & 11 & 14 \\ 8 & 7 & -2 \end{bmatrix}.$$

Consider the linear transformation $x \mapsto Ax$ from $\mathbb{R}^3$ to $\mathbb{R}^2$. This maps the unit sphere

$$\{x \in \mathbb{R}^3 \mid \|x\| = 1\}$$

onto an ellipse in $\mathbb{R}^2$, as shown in Figure 2.1. Find a unit vector x at which the length $\|Ax\|$ is maximised, and compute this maximum length.

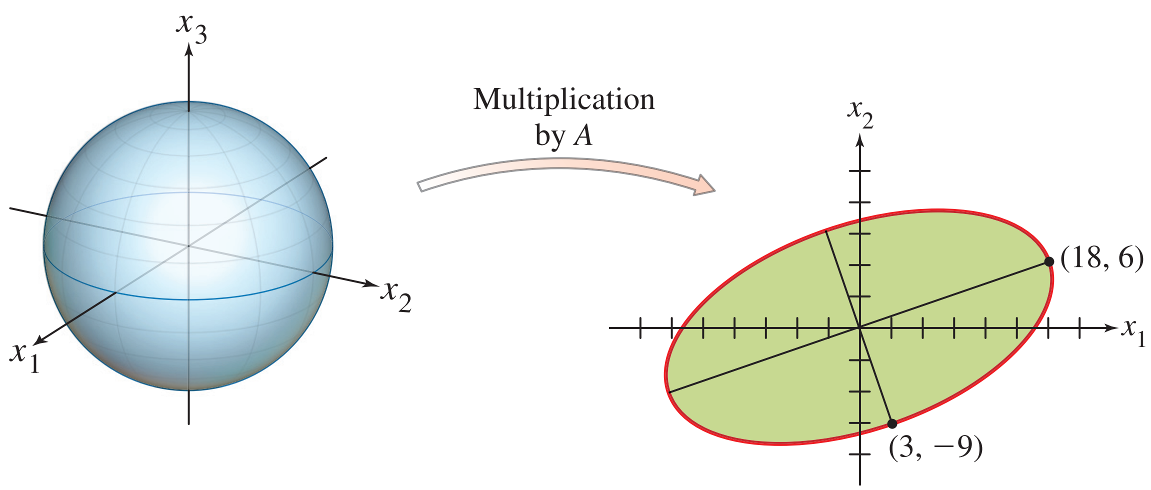


Figure 2.1: A linear transformation from $\mathbb{R}^3$ to $\mathbb{R}^2$ maps the unit sphere onto an ellipse.

To maximize $\|Ax\|$, it is convenient to consider $\|Ax\|^2$, since the maximum of $\|Ax\|^2$ occurs at the same x as the maximum of $\|Ax\|$:

$$\|Ax\|^2 = (Ax)^T(Ax) = x^T A^T Ax.$$

Notice that A^TA is symmetric, since $(A^TA)^T = A^T(A^T)^T = A^TA$. Therefore, the problem reduces to maximising the quadratic form $x^T(A^TA)x$ subject to $\|x\| = 1$. By Theorem 1.7, the maximum value is the largest eigenvalue λ_1 of A^TA, and it is attained at a unit eigenvector corresponding to λ_1.

For the given matrix A, we have

$$A^TA = \begin{bmatrix} 4 & 8 & 14 \\ 11 & 7 & -2 \end{bmatrix}^T \begin{bmatrix} 4 & 11 & 14 \\ 8 & 7 & -2 \end{bmatrix} = \begin{bmatrix} 80 & 100 & 40 \\ 100 & 140 & 200 \\ 40 & 200 & 200 \end{bmatrix}.$$

The eigenvalues of A^TA are

$$\lambda_1 = 360, \quad \lambda_2 = 90, \quad \lambda_3 = 0,$$

with corresponding unit eigenvectors

$$v_1 = \frac{1}{3}\begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix}, \quad v_2 = \frac{1}{\sqrt{35}}\begin{bmatrix} 4 \\ -1 \\ 2 \end{bmatrix}, \quad v_3 = \frac{1}{\sqrt{3}}\begin{bmatrix} -2 \\ 2 \\ 1 \end{bmatrix}.$$

Hence, the maximum value of $\|Ax\|^2$ *is* 360*, attained when* $x = v_1$*. The vector* Av_1 *corresponds to the point on the ellipse farthest from the origin:*

$$Av_1 = A\,v_1 = \begin{bmatrix} 4 & 11 & 14 \\ 8 & 7 & -2 \end{bmatrix} \frac{1}{3} \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 18 \\ 6 \end{bmatrix}.$$

Therefore, the maximum value of $\|Ax\|$ *is* $\|Av_1\| = \sqrt{360} = 6\sqrt{10}$. □

Previous example suggests that the effect of A on the unit sphere in $\mathbb{R}^3$ is related to the quadratic form

$$\frac{x^T A^T A x}{x^T x}.$$

In fact, the entire geometric behaviour of the transformation $x \mapsto Ax$ is captured by this quadratic form, as we shall see.

2.3 Singular values

Let A be an $m \times n$ matrix. Then $A^T A$ is symmetric and can be orthogonally diagonalised. Let $\{v_1, \ldots, v_n\}$ be an orthonormal basis for $\mathbb{R}^n$ consisting of eigenvectors of $A^T A$, and let $\lambda_1, \ldots, \lambda_n$ be the associated eigenvalues of $A^T A$. Then, for $1 \le i \le n$, we have

$$\|Av_i\|^2 = (Av_i)^T Av_i = v_i^T A^T A v_i = v_i^T (\lambda_i v_i),$$

since v_i is an eigenvector of $A^T A$. Therefore,

$$\|Av_i\|^2 = \lambda_i, \tag{2.1}$$

because v_i is a unit vector (i.e., $\|v_i\| = 1$).

So the eigenvalues of $A^T A$ are all nonnegative. By renumbering, if necessary, we may assume that the eigenvalues are arranged so that

$$\lambda_1 \ge \lambda_2 \ge \cdots \ge \lambda_n \ge 0.$$

Definition 2.1 (Singular values)**.** *The* ***singular values*** *of* A *are the square roots of the eigenvalues of* $A^T A$*, denoted by*

$$\sigma_1, \ldots, \sigma_n,$$

and they are arranged in decreasing order. That is,

$$\sigma_i = \sqrt{\lambda_i}, \quad \text{for } 1 \le i \le n.$$

By equation (2.1)*, the singular values of* A *are also the lengths of the vectors* $Av_1, \ldots, Av_n$*, where* $\{v_1, \ldots, v_n\}$ *is an orthonormal basis of eigenvectors of* $A^T A$*.*

Example 2.2. *Let* A *be the matrix in Example 2.1. Since the eigenvalues of* $A^T A$ *are* 360*,* 90*, and* 0*, the singular values of* A *are*

$$\sigma_1 = \sqrt{360} = 6\sqrt{10}, \quad \sigma_2 = \sqrt{90} = 3\sqrt{10}, \quad \sigma_3 = 0.$$

From Example 2.1, the first singular value of A is the maximum of $\|Ax\|$ over all unit vectors, and the maximum is attained at the unit eigenvector v_1. Theorem 1.8 shows that the second singular value of A is the maximum of $\|Ax\|$ over all unit vectors that are orthogonal to v_1, and this maximum is attained at the second unit eigenvector v_2. For v_2 in Example 2.1,

$$Av_2 = \begin{bmatrix} 4 & 11 & 14 & 24 \\ 8 & 7 & -2 & \end{bmatrix} \begin{bmatrix} -2/3 \\ -1/3 \\ 2/3 \end{bmatrix} = \begin{bmatrix} z3 \\ -9 \end{bmatrix}.$$

This point lies on the minor axis of the ellipse in Figure 2.1, just as Av_1 lies on the major axis. (See Figure 2.2.) Therefore, the first two singular values of A are the lengths of the major and minor semi-axes of the ellipse. □

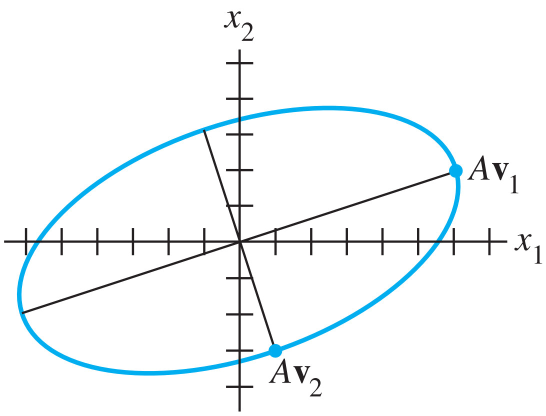


Figure 2.2: Av_1 and Av_2.

The fact that Av_1 and Av_2 are orthogonal in Figure 2.2 is no accident, as the next theorem shows.

Theorem 2.1. *Suppose $\{v_1, \ldots, v_n\}$ is an orthonormal basis of $\mathbb{R}^n$ consisting of eigenvectors of A^TA, arranged so that the corresponding eigenvalues satisfy*

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n,$$

and suppose A has r nonzero singular values. Then $\{Av_1, \ldots, Av_r\}$ is an orthogonal basis for $\mathrm{Col}(A)$, *and*

$$\mathrm{rank}(A) = \dim \mathrm{Col}(A) = r.$$

Proof. First, note that for $i \neq j$,

$$(Av_i)^T(Av_j) = v_i^T A^T A v_j = v_i^T(\lambda_j v_j) = \lambda_j v_i^T v_j = 0,$$

since v_i and v_j are orthonormal eigenvectors of A^TA. Hence, $\{Av_1, \ldots, Av_n\}$ is an orthogonal set.

Next, since the singular values of A are $\sigma_i = \sqrt{\lambda_i}$, and A has r nonzero singular values, we have $Av_i \neq 0$ if and only if $1 \leq i \leq r$. Therefore, $\{Av_1, \ldots, Av_r\}$ is a set of r linearly independent vectors.

Now, let $y \in \mathrm{Col}(A)$, so that $y = Ax$ for some $x \in \mathbb{R}^n$. Write

$$x = c_1 v_1 + \cdots + c_n v_n.$$

Then

$$y = Ax = c_1 Av_1 + \cdots + c_n Av_n = c_1 Av_1 + \cdots + c_r Av_r,$$

because $Av_{r+1} = \cdots = Av_n = 0$.

This shows that any $y \in \mathrm{Col}(A)$ is a linear combination of $\{Av_1, \dots, Av_r\}$. Hence,

$$\mathrm{Col}(A) = \mathrm{span}\{Av_1, \dots, Av_r\},$$

and $\{Av_1, \dots, Av_r\}$ is an orthogonal basis of $\mathrm{Col}(A)$.

Finally, since the basis has r vectors, we conclude $\mathrm{rank}(A) = \dim \mathrm{Col}(A) = r$. □

2.4 The singular value decomposition

The decomposition of a matrix A involves a "diagonal" matrix Σ of the same size as A, of the form

$$\Sigma = \begin{bmatrix} \hat{\Sigma} & 0_{r,n-r} \\ 0_{m-r,r} & 0_{m-r,n-r} \end{bmatrix}, \tag{2.2}$$

where $0_{i,j}$ denotes the zero $i \times j$ matrix, and $\hat{\Sigma}$ is an $r \times r$ diagonal matrix with $r \leq \min\{m, n\}$. If r equals m, n, or both, some or all of the zero blocks vanish.

Theorem 2.2 (Singular value decomposition)**.** *Let A be an $m \times n$ matrix of rank r. Then there exist orthogonal matrices*

$$U \in \mathbb{R}^{m \times m}, \qquad V \in \mathbb{R}^{n \times n},$$

and a diagonal matrix

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

with non-negative diagonal entries

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

*called the **singular values** of A, such that*

$$A = U \Sigma V^T.$$

*The columns of U are the **left singular vectors**, and the columns of V are the **right singular vectors** of A.*

Proof. Let $A \in \mathbb{R}^{m \times n}$ have rank r, and consider the symmetric positive semidefinite matrix

$$A^T A \in \mathbb{R}^{n \times n}.$$

1. Orthogonal diagonalisation of $A^T A$. Since $A^T A$ is symmetric, it can be orthogonally diagonalised. Thus, there exists an orthogonal matrix $V \in \mathbb{R}^{n \times n}$ such that

$$V^T (A^T A) V = \Lambda = \mathrm{diag}(\lambda_1, \dots, \lambda_n),$$

with eigenvalues ordered as

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0.$$

Let $v_1, \ldots, v_n$ denote the orthonormal eigenvectors forming the columns of V.

2. Singular values and the matrix Σ. Define the singular values of A by

$$\sigma_i = \sqrt{\lambda_i}, \qquad i = 1, \ldots, r,$$

where r is the number of nonzero eigenvalues of $A^T A$ (equal to the rank of A). Construct $\hat{\Sigma} \in \mathbb{R}^{r \times r}$ as the diagonal matrix

$$\hat{\Sigma} = \operatorname{diag}(\sigma_1, \ldots, \sigma_r),$$

and embed it into Σ as in (2.2).

3. Construction of U. For each $i = 1, \ldots, r$, define

$$u_i = \frac{A v_i}{\sigma_i} \in \mathbb{R}^m.$$

Each u_i is a unit vector because

$$\|u_i\| = \frac{\|A v_i\|}{\sigma_i} = \frac{\sqrt{v_i^T A^T A v_i}}{\sqrt{\lambda_i}} = 1.$$

Moreover, for $i \neq j$,

$$u_i^T u_j = \frac{v_i^T A^T A v_j}{\sigma_i \sigma_j} = \frac{v_i^T (\lambda_j v_j)}{\sigma_i \sigma_j} = 0,$$

so $\{u_1, \ldots, u_r\}$ is an orthonormal set. Extend this set to an orthonormal basis of $\mathbb{R}^m$ by choosing additional vectors $u_{r+1}, \ldots, u_m$, and let

$$U = [u_1 \ \ldots \ u_m] \in \mathbb{R}^{m \times m}.$$

4. Verification of the decomposition. By construction,

$$A v_i = \sigma_i u_i \quad (1 \leq i \leq r), \qquad A v_i = 0 \quad (i > r),$$

so that, in matrix form,

$$AV = U\Sigma.$$

Multiplying on the right by V^T gives

$$A = U \Sigma V^T,$$

which completes the proof. □

Remark 2.1 (Uniqueness)**.** *The singular values are uniquely determined by A, although the singular vectors need not be unique when singular values are repeated.* □

Example 2.3 (SVD of a 2×2 matrix)**.** *Consider the matrix*

$$A = \begin{bmatrix} 2 & 2 \\ 1 & -1 \end{bmatrix}.$$

Computing $A^T A$ gives

$$A^T A = \begin{bmatrix} 5 & 3 \\ 3 & 5 \end{bmatrix}.$$

1. ***Orthogonal diagonalisation of*** $A^T A$**.** *The eigenvalues of $A^T A$ are $\lambda_1 = 8$ and $\lambda_2 = 2$, with corresponding orthonormal eigenvectors*

$$v_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad v_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} -1 \\ 1 \end{bmatrix},$$

and so

$$V = \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}.$$

2. ***Singular values and the matrix*** Σ**.** *The singular values of A are $\sigma_1 = 2\sqrt{2}$ and $\sigma_2 = \sqrt{2}$ and*

$$\Sigma = \begin{bmatrix} 2\sqrt{2} & 0 \\ 0 & \sqrt{2} \end{bmatrix}.$$

3: ***Construction of*** U**.** *The corresponding left singular vectors are obtained from*

$$u_1 = \frac{1}{\sigma_1} A v_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad u_2 = \frac{1}{\sigma_2} A v_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

4: ***Assemble the SVD.*** *The SVD of A can be written as*

$$A = U \Sigma V^T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 2\sqrt{2} & 0 \\ 0 & \sqrt{2} \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}.$$

$\square$

Underdetermined systems

Consider the linear system $Ax = b$, where $A \in \mathbb{R}^{m \times n}$ with $m \ll n$ (short-fat matrix). There are fewer equations than unknowns, and generically, if A has full column rank, there are infinitely many solutions for every b. The system is called underdetermined because there are not enough constraints to uniquely determine x.

Example 2.4 (SVD of a 2×3 matrix)**.** *Consider the matrix*

$$A = \begin{bmatrix} 4 & 11 & 14 \\ 8 & 7 & -2 \end{bmatrix}.$$

We have

$$A^T A = \begin{bmatrix} 80 & 100 & 40 \\ 100 & 170 & 140 \\ 40 & 140 & 200 \end{bmatrix}.$$

1. ***Orthogonal diagonalisation of*** A^TA***.*** *The eigenvalues of* A^TA *are*

$$\lambda_1 = 360, \quad \lambda_2 = 90, \quad \lambda_3 = 0.$$

Let v_1, v_2, v_3 *be the orthonormal eigenvectors of* A^TA *corresponding to* $\lambda_1, \lambda_2, \lambda_3$*, Then*

$$V = \begin{bmatrix} 1/3 & -2/3 & 2/3 \\ 2/3 & -1/3 & -2/3 \\ 2/3 & 2/3 & 1/3 \end{bmatrix}.$$

2. ***Singular values and the matrix*** Σ***.*** *The singular values of* A *are the square roots of the eigenvalues:*

$$\sigma_1 = \sqrt{\lambda_1} = 6\sqrt{10}, \quad \sigma_2 = \sqrt{\lambda_2} = 3\sqrt{10}, \quad \sigma_3 = 0.$$

The nonzero singular values of A *are the diagonal entries of* $\hat{\Sigma}$*. The matrix* Σ *has the same size as* A*, with* $\hat{\Sigma}$ *in its upper-left corner and with zeros elsewhere*

$$\hat{\Sigma} = \begin{bmatrix} 6\sqrt{10} & 0 \\ 0 & 3\sqrt{10} \end{bmatrix} \quad \Rightarrow \quad \Sigma = [\hat{\Sigma} \;\; 0] = \begin{bmatrix} 6\sqrt{10} & 0 & 0 \\ 0 & 3\sqrt{10} & 0 \end{bmatrix}.$$

3: ***Construction of*** U***.*** *For the nonzero singular values, define*

$$u_i = \frac{Av_i}{\sigma_i}, \quad i = 1, 2.$$

Note that $\{u_1, u_2\}$ *is already a basis for* $\mathbb{R}^2$*. Thus no additional vectors are needed for* U*. This gives*

$$U = \begin{bmatrix} 3/\sqrt{10} & 1/\sqrt{10} \\ 1/\sqrt{10} & -3/\sqrt{10} \end{bmatrix}.$$

5. ***Assemble the SVD.*** *Finally, we have*

$$A = U\Sigma V^T = . \begin{bmatrix} 3/\sqrt{10} & 1/\sqrt{10} \\ 1/\sqrt{10} & -3/\sqrt{10} \end{bmatrix} \begin{bmatrix} 6\sqrt{10} & 0 & 0 \\ 0 & 3\sqrt{10} & 0 \end{bmatrix} \begin{bmatrix} 1/3 & 2/3 & 2/3 \\ -2/3 & -1/3 & 2/3 \\ 2/3 & -2/3 & 1/3 \end{bmatrix}.$$

Here, the first two columns of U *form an orthonormal basis for* $\mathrm{Col}(A)$*, the first two columns of* V *form an orthonormal basis for* $\mathrm{Row}(A)$*, and the last column of* V *spans* $\mathrm{Nul}(A)$*.* □

Overdetermined systems

Consider the linear system $Ax = b$, where $A \in \mathbb{R}^{m\times n}$ with $m \gg n$ (tall-skinny matrix). There are more equations than unknowns, and generically, not all b belong to $\mathrm{Col}(A)$, so there may be no exact solution. A solution exists only if $b \in \mathrm{Col}(A)$.

Example 2.5 (SVD of a 3×2 matrix)**.** *Consider the matrix*

$$A = \begin{bmatrix} 1 & 0 \\ 0 & \sqrt{2} \\ 0 & \sqrt{2} \end{bmatrix}.$$

We compute

$$A^T A = \begin{bmatrix} 1 & 0 \\ 0 & 4 \end{bmatrix},$$

so that the singular values are $\sigma_1 = 2$ *and* $\sigma_2 = 1$, *and so*

$$\hat{\Sigma} = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \quad \Rightarrow \quad \Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}.$$

The right singular vectors V *are the orthonormal eigenvectors of* $A^T A$, *giving*

$$V = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

The first two columns of the left singular vector matrix U *are obtained via*

$$u_i = \frac{1}{\sigma_i} A v_i,$$

which yields

$$u_1 = \frac{1}{2} A v_1 = \begin{bmatrix} 0 \\ 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}, \quad u_2 = \frac{1}{1} A v_2 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}.$$

The third column u_3 *is determined so that* U *is orthogonal, for example via the Gram-Schmidt process:*

$$u_3 = \begin{bmatrix} 0 \\ -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}.$$

Hence, the SVD of A *is*

$$A = U \Sigma V^T = \begin{bmatrix} 0 & 1 & 0 \\ 1/\sqrt{2} & 0 & -1/\sqrt{2} \\ 1/\sqrt{2} & 0 & 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

□

Exercise 2.1. *Let*

$$A = \begin{bmatrix} 1 & -1 \\ -2 & 2 \\ 2 & -2 \end{bmatrix} \in \mathbb{R}^{3\times 2}.$$

1. *Compute the singular values of* A.
2. *Determine the corresponding right singular vectors (columns of* V*) and left singular vectors (columns of* U*).*
3. *Construct the singular value decomposition of* A.

Remark 2.2. *The singular value decomposition provides a canonical orthogonal description of the action of a matrix. Let* $A = U\Sigma V^T$ *be the SVD of an* $m \times n$ *matrix of rank* r. *The nonzero singular values* $\sigma_1 \geq \cdots \geq \sigma_r > 0$ *determine the essential part of the map* $x \mapsto Ax$, *while the remaining* $n - r$ *singular values vanish. Although singular vectors need not be unique when values repeat, the singular values themselves are intrinsic to* A. □

Reduced/Economy SVD

When Σ contains rows or columns of zeros, a more compact decomposition of A is possible. Using the notation established above, let $r = \operatorname{rank}(A)$, and partition U and V into submatrices whose first blocks contain r columns:

$$U = [\, U_r \;\; U_{m-r} \,], \quad \text{where } U_r = [\, u_1 \; \cdots \; u_r \,],$$

$$V = [\, V_r \;\; V_{n-r} \,], \quad \text{where } V_r = [\, v_1 \; \cdots \; v_r \,].$$

Then $U_r \in \mathbb{R}^{m\times r}$ and $V_r \in \mathbb{R}^{n\times r}$. We note that U_{m-r} and V_{n-r} are included in the notation even if one of these blocks happens to have no columns. To simplify the notation, we set

$$\hat{U} = U_r \quad \text{and} \quad \hat{V} = V_r.$$

With this partitioning, matrix multiplication shows that

$$A = \begin{bmatrix} \hat{U} & U_{m-r} \end{bmatrix} \begin{bmatrix} \hat{\Sigma} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \hat{V}^T \\ V_{n-r}^T \end{bmatrix} = \hat{U}\,\hat{\Sigma}\,\hat{V}^T. \tag{2.3}$$

This factorisation of A is called the **reduced** or **economy singular value decomposition** of A and is illustrated in Figure 2.3. Since the diagonal entries of $\hat{\Sigma}$ are nonzero, $\hat{\Sigma}$ is invertible.

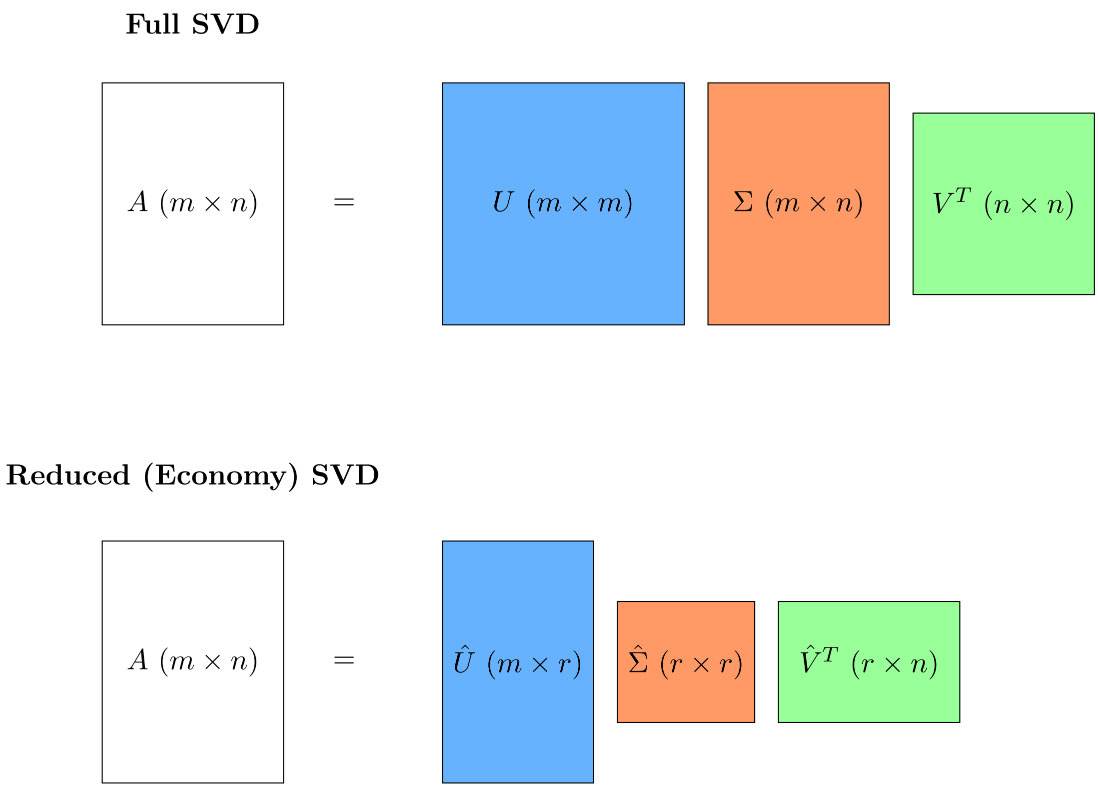


Figure 2.3: Comparison of full and reduced (economy) SVD of a matrix A.

2.5 Computing the SVD

The singular value decomposition is a cornerstone of computational science and engineering. Its numerical implementation is both important and mathematically enlightening. Most standard numerical implementations are mature, and a simple interface exists in many modern programming languages, allowing us to abstract away the underlying details of the SVD computation. For most practical purposes, we use the SVD as part of a larger computational workflow and rely on the existence of efficient and stable numerical algorithms. In MATLAB, computing the SVD is straightforward.

Listing 2.1: Full SVD in MATLAB

```
% Create a 100x20 random data matrix
A = rand(100,20);

% Compute the singular value decomposition
[U, S, V] = svd(A);
```

For non-square or large matrices, the economy SVD is more efficient. It reduces the size of the output matrices while retaining all non-zero singular values:

Listing 2.2: Economy SVD in MATLAB

```
[Uhat, Shat, Vhat] = svd(A,'econ'); % economy-sized SVD
```

The decay of the singular values is a key aspect of data compression: most of the information is concentrated in the first few singular values. This decay is clearly observable when the singular values are plotted using the MATLAB script below. In particular, the first few singular values dominate, while the remaining ones are substantially smaller.

Listing 2.3: Plot of singular values for an image

```
% Read a grayscale image
pkg load image; % for Octave
A = double(imread('cameraman.tif'));

% Compute singular values
sigma = svd(A);

% Create figure
figure;

% Equal-sized subplots [left bottom width height]
pos1 = [0.05 0.1 0.4 0.8]; pos2 = [0.55 0.1 0.4 0.8];

% Display the original image
ax1 = axes('Position', pos1);
imshow(uint8(A)); title('Original Image'); axis square;

% Plot singular values (linear scale)
ax2 = axes('Position', pos2);
plot(sigma, '-o', 'LineWidth', 1.5, 'MarkerSize', 4);
xlabel('Index i'); ylabel('\sigma_i');
title('Decay of Singular Values'); grid on; axis square;
```

2.6 Geometric interpretation of the SVD

Geometrically, the SVD shows that the linear transformation $x \mapsto Ax$ maps the unit sphere in $\mathbb{R}^n$ to an ellipsoid in $\mathbb{R}^m$. The lengths of its principal axes are the singular values $\sigma_1, \dots, \sigma_r$, and their directions are given by the singular vectors. The number of nonzero singular values equals the rank of A, determining the essential part of the transformation; the remaining directions are collapsed to zero.

Thus, A maps the unit sphere in $\mathbb{R}^n$ to an ellipsoid in $\mathbb{R}^m$. The vectors $v_1, \dots, v_r$ define the principal directions of the input space that are scaled, the singular values $\sigma_1, \dots, \sigma_r$ determine the lengths of the ellipsoid axes, and the vectors $u_1, \dots, u_r$ determine the corresponding directions in the output space.

This geometric perspective also explains why the SVD provides an optimal low-rank approximation: the largest singular values capture the directions along which A stretches space the most, and truncating the smaller singular values yields the closest approximation of A with lower rank.

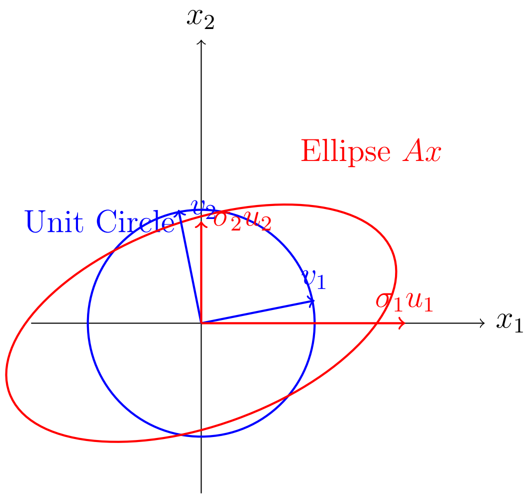


Figure 2.4: A maps the unit sphere in $\mathbb{R}^n$ to an ellipsoid in $\mathbb{R}^m$.

The following MATLAB code generates the unit circle, transforms it with a matrix A, and visualises the singular vectors and ellipse.

Listing 2.4: SVD geometric visualisation

```
% SVD Geometric visualisation
clc; clear; close all;

% Define a 2x2 matrix A
A = [2 1; 1.5 1];

% Compute SVD: A = U*S*V'
[U, S, V] = svd(A);

% Generate points on the unit circle
theta = linspace(0, 2*pi, 200);
x = [cos(theta); sin(theta)];
```

```
% Transform the unit circle by A
y = A * x;

% Plot unit circle and transformed ellipse
figure; hold on; axis equal;
plot(x(1,:), x(2,:), 'b', 'LineWidth', 2);            % Unit circle
text(-1, 0.9, 'Unit Circle', 'Color', 'b', 'FontWeight','bold');

plot(y(1,:), y(2,:), 'r', 'LineWidth', 2);            % Ellipse A*x
text(1.5, 1.5, 'Ellipse A x', 'Color', 'r', 'FontWeight','bold');

% Plot singular vectors with balanced scaling
for i = 1:2
    % Input singular vectors (V)
    quiver(0, 0, V(1,i), V(2,i), 0, 'b', 'LineWidth', 1.5, '
       MaxHeadSize',0.5);
    text(V(1,i), V(2,i), ['v_' num2str(i)], 'Color','b','
       FontWeight','bold');

    % Output singular vectors (U*S)
    quiver(0, 0, S(i,i)*U(1,i), S(i,i)*U(2,i), 0, 'r', 'LineWidth'
       , 1.5, 'MaxHeadSize',0.5);
    text(S(i,i)*U(1,i), S(i,i)*U(2,i), ...
         ['\sigma_' num2str(i) ' u_' num2str(i)], 'Color','r','
           FontWeight','bold');
end

% Labels and title
xlabel('x_1', 'FontWeight','bold');
ylabel('x_2', 'FontWeight','bold');
title('Geometric Interpretation of SVD', 'FontWeight','bold');

grid on;
```

2.7 Applications of the singular value decomposition

Condition number One important use of the SVD is in computing matrix norms. The spectral/Euclidean norm of a matrix A equals its largest singular value,

$$\|A\|_2 = \sigma_1,$$

while the Frobenius norm can be expressed in terms of all nonzero singular values:

$$\|A\|_F = \sqrt{\sigma_1^2 + \cdots + \sigma_r^2}.$$

The SVD also provides a robust framework for analysing numerical solutions of a linear system of equations. In fact, most numerical calculations involving a linear system $Ax = b$ are as reliable as possible when the singular value decomposition of A is used. The two orthogonal (or unitary) matrices U and V do not affect lengths of vectors or angles between vectors (prove it!). Any possible instabilities in numerical calculations can be identified from the singular values. If the singular values of A are extremely large or extremely small, roundoff errors are almost inevitable. Nevertheless, an error analysis is facilitated by knowing the entries in Σ and V from the SVD.

If A is an invertible $n \times n$ matrix, then the ratio of the largest to the smallest singular value

$$\kappa(A) = \frac{\sigma_1}{\sigma_n}$$

defines the **condition number** of A. The following MATLAB script illustrate how the condition number affects the sensitivity of a solution of $Ax = b$ to changes (or errors) in the entries of A. Note that the condition number can be defined in several ways, but the definition above is widely used when studying the stability of solutions of $Ax = b$.

Listing 2.5: Illustration of condition number and SVD

```
% Illustration of condition number and SVD
clc; clear; close all;

n = 10;  % matrix size

% Construct singular values for a very ill-conditioned matrix
sigma = [1*ones(1,n-1), 1e-6];  % condition number ~ 1e6

% Generate random orthogonal matrices
rng(0); [U, ~] = qr(randn(n)); [V, ~] = qr(randn(n));

% Construct A with specified singular values
A = U * diag(sigma) * V';

% Right-hand side
b = randn(n,1);

% Solve Ax = b
x_exact = A\b;

% Perturb b significantly
delta_b = 0.5*randn(n,1);  % increase perturbation
b_perturbed = b + delta_b; x_perturbed = A\b_perturbed;

% Compute singular values and condition number
[~, S, ~] = svd(A); sigma_vals = diag(S);
condA = sigma_vals(1)/sigma_vals(end);

% Relative changes
rel_change_b = norm(delta_b)/norm(b);
rel_change_x = norm(x_perturbed - x_exact)/norm(x_exact);

% Display results
fprintf('Condition number kappa(A) = %.2e\n', condA);
fprintf('Relative change in b = %.2e\n', rel_change_b);
fprintf('Relative change in solution x = %.2e\n', rel_change_x);

% Plot original vs perturbed solution
```

```
figure;
plot(1:n, x_exact, 'bo-', 'LineWidth', 2, 'MarkerSize',6);
hold on;
plot(1:n, x_perturbed, 'r*-', 'LineWidth', 2, 'MarkerSize',6);
xlabel('Component index');
ylabel('Value of x');
legend('Original x','Perturbed x');
title(['Effect of Perturbation, Condition Number \kappa(A) = ',
   num2str(condA,'%.2e')]);
grid on;
```

Exercise 2.2 (Error and condition number for Hilbert matrices)**.** *Consider the matrix $H_n = (h_{ij})_{i,j=1}^n$ defined by*

$$h_{ij} = \frac{1}{i+j-1}, \quad i,j = 1,\dots,n,$$

1. *For $n = 2, 3, \dots, N$ (choose $N \geq 10$), set the exact solution $x = \mathbf{1}_n$ and compute $b = H_n x$.*

2. *Solve the linear system $H_n\bar{x} = b$ numerically, obtaining an approximate solution $\bar{x}$. Let $\bar{b} = H_n\bar{x}$.*

3. *Compute the following quantities.*

   - *The relative error of the solution:*

   $$\mathit{error} = \frac{\|x - \bar{x}\|_2}{\|x\|_2}.$$

   - *The condition number of H_n, defined using its singular values $\sigma_1 \geq \cdots \geq \sigma_n > 0$:*

   $$\kappa(H_n) = \frac{\sigma_1}{\sigma_n}.$$

   - *An estimate of the error using the condition number and the residual:*

   $$\mathit{estimate} = \kappa(H_n)\frac{\|b - \bar{b}\|_2}{\|b\|_2}.$$

4. *Plot or tabulate the relative error and the estimate as functions of n. Discuss the observed trend as the order of the Hilbert matrix increases.*

5. *Explain why Hilbert matrices are numerically challenging and how the singular values influence the sensitivity of the solution.*

Bases for fundamental subspaces Let A be an $m \times n$ matrix with singular value decomposition $A = U\Sigma V^T$. Let $u_1, \dots, u_m$ be the left singular vectors, $v_1, \dots, v_n$ the right singular vectors, and $\sigma_1, \dots, \sigma_n$ the singular values of A. Let r denote the rank of A.

By Theorem 2.1,

$$\boxed{\{\, u_1, u_2, \ldots, u_r \,\} \qquad \text{in an orthonormal basis for Col}\, A.} \tag{2.4}$$

It is well known that $(\text{Col}\, A)^{\perp} = \text{Nul}\, A^T$. Hence the set

$$\boxed{\{\, u_{r+1}, \ldots, u_m \,\} \qquad \text{in an orthonormal basis for Nul}\, A^T.} \tag{2.5}$$

Since $\|Av_i\| = \sigma_i$ for $1 \le i \le n$, and $\sigma_i = 0$ if and only if $i > r$, the vectors $v_{r+1}, \ldots, v_n$ span a subspace of $\text{Nul}\, A$ of dimension $n - r$. By the Rank-Nullity Theorem[1] ,

$$\dim(\text{Nul}\, A) = n - \text{rank}(A).$$

It follows that

$$\boxed{\{\, v_{r+1}, \ldots, v_n \,\} \qquad \text{is an orthonormal basis for Nul}\, A.} \tag{2.6}$$

From (2.4) and (2.5), the orthogonal complement of $\text{Nul}\, A^T$ is

$$(\text{Nul}\, A^T)^{\perp} = \text{Col}\, A.$$

Interchanging A and A^T, we obtain analogous results for the row space and nullspace

$$(\text{Nul}\, A)^{\perp} = \text{Col}\, A^T = \text{Row}\, A.$$

Hence, from (2.6),

$$\boxed{\{\, v_1, \ldots, v_r \,\} \qquad \text{is an orthonormal basis for Row}\, A.} \tag{2.7}$$

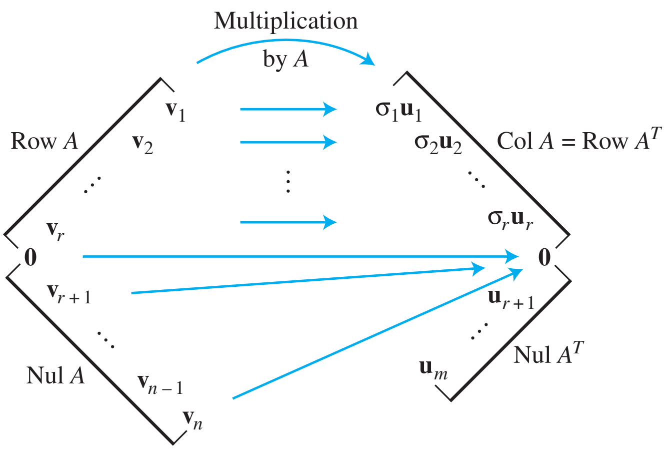


Figure 2.5: The four fundamental subspaces and the action of A.

Figure 2.5 summarises equations (2.4)–(2.7), but shows the orthogonal basis

$$\{\sigma_1 u_1, \ldots, \sigma_r u_r\}$$

for $\text{Col}\, A$ instead of the normalised basis, to emphasise that $Av_i = \sigma_i u_i, \le i \le r$. Explicit orthonormal bases for the four fundamental subspaces determined by A are useful in certain calculations, particularly in constrained optimisation problems.

[1]The Rank-Nullity Theorem states that for a linear map $A : \mathbb{R}^n \to \mathbb{R}^m$ (or an $m \times n$ matrix A), $\text{rank}(A) + \dim(\text{Nul}(A)) = n$. That is, the dimension of the column space plus the dimension of the nullspace equals the number of columns of A.

The pseudoinverse of A and the least squares problem Let $A \in \mathbb{R}^{m\times n}$ have rank r, and consider its singular value decomposition

$$A = U\,\Sigma\,V^T, \qquad \Sigma = \begin{bmatrix} \hat{\Sigma} & 0_{r,n-r} \\ 0_{m-r,r} & 0_{m-r,n-r} \end{bmatrix},$$

where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are orthogonal matrices, and $\hat{\Sigma} \in \mathbb{R}^{r\times r}$ contains the nonzero singular values $\sigma_1 \geq \cdots \geq \sigma_r > 0$. The **pseudoinverse** (or the **Moore–Penrose inverse**) of A is then defined by

$$A^+ = V\,\Sigma^+\,U^T, \qquad \Sigma^+ = \begin{bmatrix} \hat{\Sigma}^{-1} & 0_{r,m-r} \\ 0_{n-r,r} & 0_{n-r,m-r} \end{bmatrix}$$

For certain types of matrices A, especially when A is large or has low-rank[2], it is computationally advantageous to use the For computational efficiency, particularly when A large, we can use the economy SVD (2.3), so that

$$A^+ = \hat{V}\,\hat{\Sigma}^{-1}\,\hat{U}^T. \tag{2.8}$$

This pseudoinverse provides a least-squares solution of $Ax = b$ and has the smallest length among all solutions (see Exercise 2.3).

Given a linear system $Ax = b$, use the pseudoinverse of A from (2.8) to define

$$\hat{x} = A^+ b = \hat{V}\,\hat{\Sigma}^{-1}\,\hat{U}^T b.$$

Then, from the reduced SVD, we have

$$A\hat{x} = (\hat{U}\,\hat{\Sigma}\,\hat{V}^T)\,(\hat{V}\,\hat{\Sigma}^{-1}\,\hat{U}^T b) = \hat{U}\,\hat{\Sigma}\,\hat{\Sigma}^{-1}\,\hat{U}^T b = \hat{U}\hat{U}^T b.$$

It follows from (2.4) that $\hat{b} = \hat{U}\hat{U}^T b$ is the orthogonal projection b onto $\operatorname{Col} A$. Thus, $\hat{x}$ is a **least-squares solution** of $Ax = b$.

Exercise 2.3 (Minimal-norm least-squares solution)**.** *Let $A \in \mathbb{R}^{m\times n}$ have rank r, and let $A = U\Sigma V^T$ be its SVD. Define the pseudoinverse A^+ and $\hat{x} = A^+ b$ for some $b \in \mathbb{R}^m$.*

1. *Show that $\hat{x}$ satisfies the normal equations $A^T A x = A^T b$.*
2. *Any least-squares solution can be written as $x = \hat{x} + z$, with $z \in \operatorname{Nul} A$. Using the fact that $\hat{x} \in \operatorname{Col} \hat{V}$ and $z \in \operatorname{Col} V_{n-r}$, show that $\hat{x}^T z = 0$ (i.e., $\hat{x}$ is orthogonal to z).*
3. *Deduce that*
$$\|x\|^2 = \|\hat{x}\|^2 + \|z\|^2 \geq \|\hat{x}\|^2,$$
and conclude that $\hat{x}$ has minimal norm among all least-squares solutions.
4. *Show that the minimum-norm solution can be expressed as*
$$\hat{x} = \sum_{i=1}^{r} \frac{u_i^T b}{\sigma_i} v_i = \sum_{i=1}^{r} \sigma_i^{-1}\, v_i\, u_i^T b.$$

[2] The economy SVD is particularly convenient when A is large or rank-deficient, as it avoids storing unnecessary zero singular values and reduces computational cost.

2.8 Low-rank approximation

The singular value decomposition not only provides insight into the fundamental subspaces of a matrix and allows the computation of pseudo-inverses, but it also gives a natural way to approximate a matrix by one of lower rank. This is particularly useful in applications such as data compression, signal processing, and machine learning, where one often wants to retain the most important features of a dataset while reducing storage or noise. The following theorem formalises this idea.

Theorem 2.3 (Eckart–Young). *Let $A \in \mathbb{R}^{m\times n}$ have rank r and singular value decomposition*

$$A = U\Sigma V^T = \sum_{i=1}^{r} \sigma_i u_i v_i^T.$$

Then A can be written as a sum of r rank-one matrices. Moreover, for any $0 \le p \le r$, the best rank-p approximation of A in both the spectral and Frobenius norms is the truncated SVD

$$A_p = U_p\Sigma_p V_p^T = \sum_{i=1}^{p} \sigma_i u_i v_i^T.$$

The optimal approximation errors are

$$\|A - A_p\| = \min_{\operatorname{rank}(B)\le p} \|A - B\| = \sigma_{p+1},$$

and

$$\|A - A_p\|_F = \min_{\operatorname{rank}(B)\le p} \|A - B\|_F = \sqrt{\sum_{i=p+1}^{r} \sigma_i^2}.$$

Proof. Because the Frobenius and spectral norms are unitarily invariant,

$$\|A - B\|_F = \|U^T(A - B)V\|_F = \|\Sigma - \tilde{B}\|_F, \qquad \|A - B\| = \|\Sigma - \tilde{B}\|,$$

where $\tilde{B} = U^T BV$. Thus, it suffices to consider approximations $\tilde{B}$ expressed in the singular-vector basis of A.

Any matrix $\tilde{B}$ of rank at most p can have at most p nonzero singular values. In particular, in the diagonal approximation we must have $\beta_{ii} = 0$ for $i > p$.

The Frobenius norm satisfies

$$\|\Sigma - \tilde{B}\|_F^2 = \sum_{i=1}^{r}(\sigma_i - \beta_{ii})^2 + \sum_{i\ne j} |\beta_{ij}|^2.$$

The expression is minimised by setting all off-diagonal entries $\beta_{ij} = 0$ and choosing

$$\beta_{ii} = \sigma_i \quad (i = 1, \dots, p), \qquad \beta_{ii} = 0 \quad (i > p).$$

Hence,

$$\|A - A_p\|_F^2 = \sum_{i=p+1}^{r} \sigma_i^2,$$

so $A_p = U_p\Sigma_p V_p^T$ is the optimal Frobenius approximation.

For the spectral/Euclidean norm,

$$\|A - B\| = \sigma_{\max}(A - B),$$

and with the above choice of $\tilde{B}$, the matrix $\Sigma - \tilde{B}$ has singular values $\sigma_{p+1}, \ldots, \sigma_r$. Thus,

$$\|A - A_p\| = \sigma_{p+1},$$

and no other rank-p matrix can improve this value. The truncated SVD therefore minimises both norms, completing the proof. $\square$

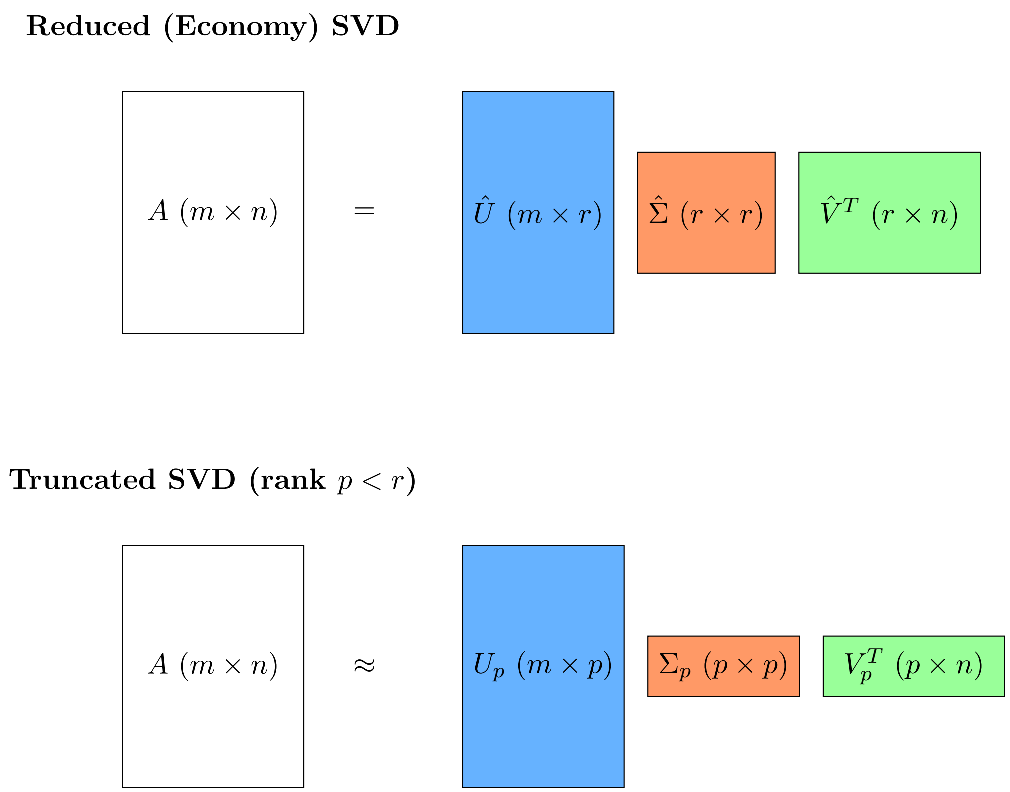


Figure 2.6: Comparison of reduced (economy) and truncated SVD of a matrix A.

The following MATLAB script gives an example of a rank-p approximation of a matrix.

Listing 2.6: Rank-p approximation via truncated SVD

```
% Create a 100x20 random matrix
A = rand(100,20);

% Compute the full SVD
[U, S, V] = svd(A);

% Choose the desired rank
p = 5;

% Construct the rank-r approximation
A_p = U(:,1:p) * S(1:p,1:p) * V(:,1:p)';

% Display approximation error (spectral and Frobenius norm)
```

```
error2 = norm(A - A_p);
disp(['2-norm of error: ', num2str(error2)]);
errorF = norm(A - A_p, 'fro');
disp(['Frobenius norm of error: ', num2str(errorF)]);
```

Perhaps the most useful and defining property of the singular value decomposition is that it provides an **optimal low-rank approximation** to a matrix A. In fact, the SVD naturally provides a hierarchy of low-rank approximations: a rank-p approximation is obtained by retaining the leading p singular values and the corresponding singular vectors, while discarding the remaining ones. This is particular useful in image compression.

Consider a grayscale image stored as a matrix $A \in \mathbb{R}^{m\times n}$. The SVD can be used to approximate A by a matrix of lower rank while retaining most of the visual information. The next example demonstrates that the truncated SVD A_p captures the most significant features of the image.

Listing 2.7: Image compression via truncated SVD

```
% Read a grayscale image and convert to double
pkg load image; % for Octave
A = double(imread('cameraman.tif'));

% Compute the SVD
[U, S, V] = svd(A);
% [U, S, V] = svd(A, 'econ'); % recommended for images

% Choose rank r for approximation
p = 50;

% Construct rank-r approximation
A_p = U(:,1:p) * S(1:p,1:p) * V(:,1:p)';

% Display original and compressed images
figure;
subplot(1,2,1), imshow(uint8(A)), title('Original Image');
subplot(1,2,2), imshow(uint8(A_p)), title(['Rank-', num2str(p),
   'Approximation']);
```

Even though A_p has the same dimensions as A, the effective storage is much smaller: only the leading p singular values and vectors need to be stored:

$$A_p = \sum_{i=1}^{p} \sigma_i u_i v_i^T \quad \Rightarrow \quad \text{storage } = p(m+n+1) \ll mn.$$

For example, a 1000×800 image approximated with $p = 50$ requires storing only $50(1000+800+1) = 92500$ numbers, instead of 800000, achieving significant compression while retaining the main visual features.

For large matrices or images, the economy SVD avoids storing unnecessary zero singular values, making computations faster and less memory-intensive. Using the economy SVD, we compute only the first $\min(m,n)$ columns of U and V, which is sufficient for forming truncated low-rank approximations. This is especially advantageous for very large images, where storing the full SVD would be inefficient.

2.9 Exercises

Hand-solved exercises

1. Compute the SVD of the matrix
$$A = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$$
and verify $A = U\Sigma V^T$.

2. Find the rank, null space, and column space for
$$B = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

3. Show that for a diagonal matrix D, the singular values are the absolute values of the diagonal entries.

4. Let A be an $m \times n$ matrix of rank r with singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ (the remaining singular values, if any, are zero). Prove the following facts.
   (a) $\|A\| = \sigma_1$ and $\|A\|_F = \sqrt{\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_r^2}$;
   (b) If A is square, $A \in \mathbb{R}^{n\times n}$, then $|\det(A)| = \prod_{i=1}^{n} \sigma_i$, where the product includes any zero singular values when A is singular.

5. Let A be a square $n \times n$ matrix. Prove that the following statements are each equivalent to the statement that A is invertible: (a) $\operatorname{Col} A = \{0\}$; (b) $\operatorname{Nul} A = \mathbb{R}^n$; (c) $\operatorname{Row} A = \mathbb{R}^n$; (d) A has n nonzero singular values.

6. Using SVD, ompute the truncated rank-1 approximation of
$$C = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}.$$

7. Verify that for a matrix A, A^TA and AA^T have the same non-zero eigenvalues.

8. Using SVD, compute the pseudoinverse D^+ for
$$D = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 2 \end{bmatrix}.$$

9. Show that left-multiplying a matrix by a unitary matrix preserves singular values.

10. Compute the SVD of a 3×3 rotation matrix and interpret the singular values geometrically.

11. Verify numerically that $EE^+E = E$ and $E^+EE^+ = E^+$ for
$$E = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}.$$

12. Justify the statement that the second singular value of a matrix A is the maximum of $\|Ax\|$ as x varies over all unit vectors orthogonal to v, where v is a right singular vector corresponding to the first singular value of A. [Hint: Use Theorem 1.8.]

13. Suppose an experiment leads to the following linear system:

$$\begin{aligned} 4.5\,x_1 + 3.1\,x_2 &= 19.2493, \\ 1.6\,x_1 + 1.1\,x_2 &= 6.843. \end{aligned} \tag{2.9}$$

The measured right–hand side data are later rounded to two decimal places, giving system:

$$\begin{aligned} 4.5\,x_1 + 3.1\,x_2 &= 19.25, \\ 1.6\,x_1 + 1.1\,x_2 &= 6.84. \end{aligned} \tag{2.10}$$

(a) Solve system (2.9). Then solve system (2.10). In each case give the solution exactly (to an appropriate number of significant digits) and compare the two numerical solutions.

(b) The entries on the right–hand side of (2.10) differ from those of (2.9) by less than 0.05%. Using the solution of (2.10) as an approximation to the solution of (2.9), compute the *percentage error* in this approximation:

$$\text{percentage error} = 100\% \cdot \|x^{(3)} - x^{(4)}\| / \|x^{(3)}\|,$$

where $x^{(3)}$ and $x^{(4)}$ are the solutions of systems (2.9) and (2.10), respectively.

Matlab exercises

1. **Rounding errors and condition number.**

(a) Find the condition number (with respect to the 2-norm) of the matrix

$$A = \begin{bmatrix} 4 & 0 & -7 & -7 \\ -6 & 1 & 11 & 9 \\ 7 & -5 & 10 & 19 \\ -1 & 2 & 3 & -1 \end{bmatrix}.$$

Construct a random vector $x \in \mathbb{R}^4$ and $b = Ax$. Compute the numerical solution $\hat{x}$ of $A\hat{x} = b$. Determine how many digits of x and $\hat{x}$ agree. Find the number of digits that your software stores accurately, and report how many digits of accuracy are lost when $\hat{x}$ is used instead of the exact solution x.

(b) Repeat for the matrix

$$B = \begin{bmatrix} 5 & 3 & 1 & 7 & 9 \\ 6 & 4 & 2 & 8 & -8 \\ 7 & 5 & 3 & 10 & 9 \\ 9 & 6 & 4 & -9 & -5 \\ 8 & 5 & 2 & 11 & 4 \end{bmatrix}.$$

2. **Hilbert matrices.**

(a) Solve the linear system $Ax = b$ for a suitable right-hand side b in order to compute the last column of the inverse of the fifth-order Hilbert matrix. How many digits of each entry of x do you expect to be correct? Explain your reasoning in terms of the condition number of the Hilbert matrix.

(b) Compute the inverse of a Hilbert matrix of order 12 or larger. Compute the product AA^{-1} and report your observations. Comment on the numerical difficulties that appear when forming the inverse of large Hilbert matrices.

3. **Comparison with MATLAB backslash.** For a random matrix $A \in \mathbb{R}^{m\times n}$ and random vector $b \in \mathbb{R}^m$, solve $Ax = b$ using both the pseudoinverse A^+ and MATLAB's backslash operator $x = Ab$. Compare the solutions and comment on any differences.

4. **Pseudoinverse via SVD.** Consider the rank-deficient matrix

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}.$$

   (a) Compute the pseudoinverse A^+ using MATLAB's `pinv` function.

   (b) Compute A^+ manually using the SVD (full or economy).

   (c) Verify the pseudoinverse property

$$AA^+A = A \quad \text{and} \quad A^+AA^+ = A^+.$$

   (d) The same for a random $A \in \mathbb{R}^{4\times 2}$ matrix.

5. **Least-squares solution with minimal norm.** Let

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}, \quad b = \begin{bmatrix} 7 \\ 8 \\ 9 \end{bmatrix}.$$

   (a) Compute the least-squares solution $\hat{x} = A^+b$.

   (b) Compute the projection $\hat{b} = A\hat{x}$ of b onto the column space of A.

   (c) Compute the residual $\|b - \hat{b}\|$.

   (d) The same for a random $A \in \mathbb{R}^{5\times 3}$ matrix with entries drawn from a standard normal distribution and $b \in \mathbb{R}^5$ a random vector.

6. **Random rectangular matrices.** Generate a random matrix $A \in \mathbb{R}^{m\times n}$ with $m > n$ and a random vector $b \in \mathbb{R}^m$.

   (a) Compute $\hat{x} = A^+b$ using the pseudoinverse.

   (b) Verify that $\hat{x}$ solves the least-squares problem by checking that $A^T(b - A\hat{x}) \approx 0$.

   (c) Compare the norm $\|\hat{x}\|$ with the norm of any other least-squares solution you can generate by adding a vector from the nullspace of A.

7. **Economy SVD.** For a large random matrix $A \in \mathbb{R}^{100\times 50}$, compute its pseudoinverse using the economy SVD

$$A^+ = V_r \Sigma_r^{-1} U_r^T.$$

   (a) Verify that $\hat{x} = A^+b$ minimises $\|x\|_2$ among all least-squares solutions.

   (b) Compare your result with MATLAB's `pinv(A)*b` and discuss any differences (numerical roundoff).

8. Compute the truncated rank-2 approximation of a 6×6 Hilbert matrix.

```
H = hilb(6);
[U,S,V] = svd(H);
H_2 = U(:,1:2)*S(1:2,1:2)*V(:,1:2)';
```

9. Construct a low-rank approximation of a 10×10 random matrix using the top 3 singular values.

```
A = rand(10);
[U,S,V] = svd(A);
p = 3;
A_p = U(:,1:p)*S(1:p,1:p)*V(:,1:p)';
```

10. Visualize the singular values of a 10×10 random matrix.

```
X = rand(10);
s = svd(X);
semilogy(s,'o-');
```

11. Solve an overdetermined system $Ax = b$ using pseudo-inverse.

```
A = rand(5,3);
b = rand(5,1);
x = V*inv(S)*U'*b; % Using SVD
x_alt = pinv(A)*b; % Using pinv
```

12. Load a grayscale image, compute its SVD, and reconstruct using top p singular values.

```
pkg load image; % for Octave
img = imread('cameraman.tif');
img = double(img);
[U,S,V] = svd(img);
p = 50;
img_p = U(:,1:p)*S(1:p,1:p)*V(:,1:p)';
imshow(uint8(img_p));
```

Compute the energy captured by the top $p < r$ singular values

$$E_p = \frac{\sigma_1^2 + \cdots + \sigma_p^2}{\sigma_1^2 + \cdots + \sigma_r^2}, \qquad r = \operatorname{rank}(S).$$

13. Compute the rank using `rank` and compare with nonzero singular values.

```
X = rand(6,4);
rank_X = rank(X);
s = svd(X);
sum(s>1e-10)
```

14. Generate a tall-skinny matrix A and compute minimum-norm solution.

```
A = rand(10,4);
b = rand(10,1);
[U,S,V] = svd(A,'econ');
x_min = V*inv(S)*U'*b;
```

15. Plot truncation error $\|X - \tilde{X}_p\|_F$ vs p for a 20×20 matrix.

```
X = rand(20);
[U,S,V] = svd(X);
errors = zeros(20,1);
for p = 1:20
    Xp = U(:,1:p)*S(1:p,1:p)*V(:,1:p)';
    errors(p) = norm(X-Xp,'fro');
end
plot(1:20,errors,'o-');
xlabel('p'); ylabel('Frobenius error');
```

16. Compute the pseudoinverse of a 6×4 random matrix using SVD.

```
A = rand(6,4);
[U,S,V] = svd(A,'econ');
A_pinv = V*inv(S)*U';
```

17. Use SVD to solve an overdetermined system $Ax = b$ for $A \in \mathbb{R}^{8\times4}$ and random b.

```
A = rand(8,4); b = rand(8,1);
[U,S,V] = svd(A,'econ');
x = V*inv(S)*U'*b;
```

18. Apply SVD to a color image by performing SVD separately on each RGB channel. Discuss visual results.

19. Generate a random noisy matrix $C = A + \epsilon$ and compare singular values with the original A. Analyze the effect of noise.

20. Generate a 20×20 random matrix, zero out all but the top 5 singular values, and reconstruct it.

21. Plot the singular values of a 50×50 random matrix in decreasing order.

22. Compute SVD of a 10×10 rotation matrix, verify singular values are all 1, and plot the transformed unit circle.

23. Generate a 5×5 matrix with known rank 3, compute SVD, and verify rank using the number of non-zero singular values.

24. **Low-rank Image Approximation.** In this exercise, you will explore low-rank approximations of grayscale images using SVD in MATLAB.

    (a) Load a grayscale image, for example `cameraman.tif`, and store it as a matrix $A \in \mathbb{R}^{m\times n}$.

    (b) Compute the SVD of A using both full and economy formats.

    (c) For several ranks p (e.g., $p = 5, 10, 20, 50$), construct the rank-p approximation A_p using both full and economy SVD.

    (d) Display the approximated images side by side with the original image.

    (e) Compute and record the 2-norm of the approximation error: $\|A - A_p\|_2$.

    (f) Plot the singular values σ_i of A in descending order and observe how quickly they decay. Discuss how this decay relates to the quality of the rank-p approximations.

    (g) (Optional) Compare the computation time and memory usage of the full SVD vs. economy SVD for a large image (e.g., 512×512). Comment on the advantages of using the economy SVD in practice.

25. **SVD and image compression with AT&T Faces.** The **AT&T (ORL) Face Database** contains grayscale images of 40 subjects, each with 10 images taken under slightly different conditions (expressions, pose). Each image is 92×112 pixels in PGM format. The dataset is freely available for academic use and can be downloaded from `https://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html`.

    Assume that the images are stored in a subfolder `att_faces` in your current MATLAB folder, with subfolders `s1`, `s2`, . . . , `s40` corresponding to the subjects.

    (a) Load the first image of subject 1 (`s1/1.pgm`) into MATLAB as a grayscale double matrix A.

    (b) Compute the singular value decomposition (SVD) of A.

    (c) Construct a rank-p truncated approximation for a given p (e.g., $p = 10$).

    (d) Display side by side the original image and the truncated approximation using `imshow`, and comment on the visual quality of the approximation as p increases.

    (e) Plot the singular values to show their decay and discuss how it relates to the image content.

    You may use the following MATLAB template to get started.

```
% Current folder and subfolder
dataDir = fullfile(pwd, 'att_faces');

% Read a grayscale image and convert to double
imgFile = fullfile(dataDir, 's1', '1.pgm');
A = double(imread(imgFile));

% Compute de SVD
[U, S, V] = svd(A);
```

Chapter 3

Principal Component Analysis

3.1 Dimensionality reduction

Dimensionality reduction methods aim to simplify datasets while preserving their most important information. Broadly, these methods can be divided into two main categories: feature selection and feature extraction. Feature selection involves identifying a subset of the original features that are most relevant to the task at hand, whereas feature extraction constructs a new set of features or a subspace that captures the essential structure of the data.

Feature extraction serves as both a dimensionality-reduction and a data-compression technique, with the primary goal of retaining the key characteristics of the dataset. In practice, it is often employed to reduce storage requirements and computational costs, as well as to improve predictive performance by mitigating the curse of dimensionality.

Real-world applications illustrate the versatility of feature extraction. In recommender systems, such as those used by Netflix or Spotify, high-dimensional user–item matrices can be transformed to reveal latent preference patterns, enhancing prediction and recommendation accuracy. In image processing, large pixel arrays can be summarized into a few principal components that capture essential structural variations, enabling efficient compression, noise reduction, and feature extraction, as exemplified by eigenfaces in facial recognition. In genomics and biology, datasets containing thousands of gene-expression measurements per sample can be projected onto a smaller set of features that highlight biologically meaningful patterns, such as differences between healthy and diseased tissues or between distinct cell types. In finance, correlated asset returns can be represented by a few underlying factors that describe market-wide or sector-specific trends, facilitating risk modeling and portfolio optimization. Even in natural language processing, high-dimensional representations such as document–term matrices or word embeddings can be reduced to lower-dimensional features that preserve semantic relationships while improving computational efficiency.

Among the most widely used feature-extraction techniques are **Principal Component Analysis (PCA)**, which identifies directions of maximum variance in the data; **Linear Discriminant Analysis (LDA)**, which seeks directions that maximize class separability; and **Kernel Principal Component Analysis (Kernel PCA)**, which generalizes PCA to capture nonlinear structures. Collectively, these methods provide a versatile toolkit for reducing dimensionality while preserving the most informative aspects of complex datasets.

3.2 Principal component analysis

Principal Component Analysis (PCA), also known as the Orthogonal Linear Transformation (OLT), was first introduced in 1901 by Karl Pearson as an analogue of the Principal Axis Theorem in mechanics (see Theorem 1.4). It was later independently developed and named by Harold Hotelling in the 1930s.

Figure 3.1: Karl Pearson (1857–1936).

PCA is a statistical procedure that employs an orthogonal transformation to convert a set of observations, which may consist of correlated variables, into a set of linearly uncorrelated variables called *principal components*. The orthogonal axes of this new feature space can be interpreted as the directions of maximum variance, under the constraint that all axes are mutually orthogonal (see Figure 3.2).

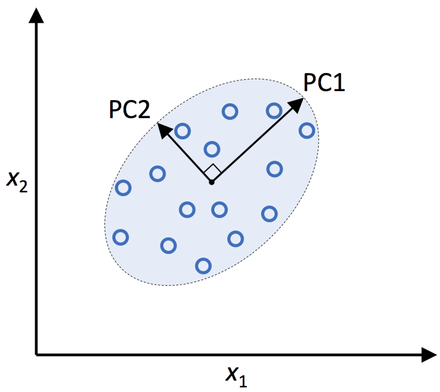


Figure 3.2: Principal components.

As an unsupervised linear transformation technique, PCA is widely used across various fields, most prominently in machine learning for feature extraction and dimensionality reduction. It identifies patterns in data by analyzing the correlations among features, revealing the directions along which the data varies most.

It is important to note that PCA is highly sensitive to the scaling of the data. Therefore, features should typically be standardised prior to applying PCA to ensure that all variables contribute equally to the analysis. PCA is considered unsupervised because it does not utilise any class label information in determining the principal components.

3.3 PCA and SVD

High-dimensional datasets often contain correlated or redundant information, making analysis, visualisation, and predictive modelling challenging. The goal of **Principal Component Analysis (PCA)** is to reduce dimensionality while retaining the most informative patterns. PCA identifies an orthogonal basis of *principal directions* along which the data exhibits maximal variance. Projecting the data onto the leading directions yields a low-dimensional representation that preserves the dominant structure, reduces noise, and facilitates interpretation. While the Singular Value Decomposition (SVD) is a purely algebraic factorisation, PCA is a statistical approach that finds directions maximising variance.

Let

$$X \in \mathbb{R}^{N \times d}$$

be the data matrix, with N **samples** (rows) and d **features** (columns). Typically, $N \gg d$, so the data matrix X is tall and narrow. Let $\mathbf{1}_N \in \mathbb{R}^{N \times 1}$ denote a column vector of ones and $\mu \in \mathbb{R}^{d \times 1}$ be the vector of feature means with components

$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} X_{ij}.$$

Centering the data by subtracting the feature means,

$$X_c = X - \mathbf{1}_N \mu^T,$$

is a ensures that each column has zero empirical mean. Centring allows PCA to focus on variance relative to the mean rather than differences in feature offsets.

Before defining PCA formally, it is important to note that the number of independent directions in the data is determined by the **linear independence of the features**. If all features are linearly independent and $N \geq d$, the rank of X_c is $r = d$, and there are d nonzero principal components. If some features are linearly dependent, then $r < d$, and only r meaningful principal components exist. PCA therefore extracts the **intrinsic dimensions** of the data, ignoring redundant or perfectly correlated features.

The goal of PCA is to find orthonormal vectors $v_1, \ldots, v_r \in \mathbb{R}^d$ such that the projections

$$z_i = X_c v_i \in \mathbb{R}^N, \quad i = 1, \ldots, r$$

capture maximal variance, subject to $v_i^T v_j = \delta_{ij}$. The first principal component v_1 maximises

$$v_1 = \arg\max_{\|v\|=1} \operatorname{Var}(X_c v) = \arg\max_{\|v\|=1} v^T S v,$$

where

$$S = \frac{1}{N-1} X_c^T X_c \in \mathbb{R}^{d \times d} \tag{3.1}$$

is the covariance matrix of the centred data. Subsequent components are obtained by maximising variance in directions orthogonal to the previously found components. In this way, PCA transforms the original correlated variables into a set of uncorrelated variables (the principal components) that capture the dominant patterns of variation.

According to theorems 1.7–1.9, the principal directions v_i are the eigenvectors of S, and the corresponding eigenvalues λ_i quantify the variance captured along each component:

$$Sv_i = \lambda_i v_i, \quad i = 1, \dots, r, \quad \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r \geq 0.$$

From a linear-algebra perspective, PCA can be computed efficiently via the SVD of the centred data

$$X_c = U \begin{bmatrix} \hat{\Sigma}^2 & 0 \\ 0 & 0 \end{bmatrix} V^T = \hat{U}\hat{\Sigma}\hat{V}^T,$$

where $\hat{U} \in \mathbb{R}^{N \times r}$ and $\hat{V} \in \mathbb{R}^{d \times r}$ are orthonormal matrices, $\hat{\Sigma} \in \mathbb{R}^{r \times r}$ is diagonal with singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$, and $r = \operatorname{rank}(X_c)$. In this case,

$$S = \frac{1}{N-1} X_c^T X_c = \frac{1}{N-1} \hat{V}\hat{\Sigma}^2\hat{V}^T$$

and so, the **principal directions** (**loadings**) correspond to the columns of $\hat{V}$ in the SVD, and the **PCA scores** (coordinates of the samples in the principal component space) are

$$Z = X_c\hat{V} = \hat{U}\hat{\Sigma}.$$

This SVD formulation directly links the statistical goal of maximising variance to a linear-algebraic factorisation and is convenient for computation, dimensionality reduction, and feature extraction.

Remark 3.1 (Covariance for Transposed Data Matrices)**.** *In some applications (e.g., signal processing or time-series analysis), data is organized as $X \in \mathbb{R}^{d \times N}$, with features as rows and samples as columns. The feature means form a column vector $\mu \in \mathbb{R}^{d \times 1}$ with components*

$$\mu_i = \frac{1}{N} \sum_{j=1}^{N} X_{ij}, \quad i = 1, \dots, d,$$

the centred data matrix is then

$$X_c = X - \mu \mathbf{1}_N^T,$$

and the covariance matrix of the features is

$$S = \frac{1}{N-1} X_c X_c^T \in \mathbb{R}^{d \times d}.$$

Applying the SVD,

$$X_c = \hat{U}\hat{\Sigma}\hat{V}^T,$$

where[1] *$\hat{U} \in \mathbb{R}^{d \times r}$, $\hat{\Sigma} \in \mathbb{R}^{r \times r}$, and $\hat{V} \in \mathbb{R}^{N \times r}$, we can write*

$$S = \frac{1}{N-1} X_c X_c^T = \frac{1}{N-1} \hat{U}\hat{\Sigma}^2\hat{U}^T.$$

[1]Note that the dimensions of $\hat{U}$ and $\hat{V}$ are different from the standard $N \times d$ convention.

Now, the columns of $\hat{U}$ are the principal directions (loadings), and the scores of the samples along these directions are

$$Z = \hat{U}^T X_c = \hat{\Sigma}\hat{V}^T \in \mathbb{R}^{d\times N}.$$

Thus, the PCA structure is preserved: eigenvectors of $X_c X_c^T$ give the directions of maximal variance, while Z contains the sample coordinates. Only the roles (and the dimensions) of $\hat{U}$ and $\hat{V}$ are swapped compared to the standard $N \times d$ convention. □

Remark 3.2 (Efficiency of PCA via SVD)**.** *PCA is typically computed using the SVD of the centred data matrix rather than the eigenvalue decomposition of the covariance matrix. This is computationally more efficient when the number of samples N is large, because forming the covariance matrix S requires an additional matrix multiplication and leads to an eigenvalue problem of size $d \times d$. In contrast, the SVD works directly with X_c and avoids explicitly constructing S, while also providing numerically more stable estimates of the principal components.* □

3.4 Low-rank approximation

One of the most important applications of the Singular Value Decomposition (SVD) is optimal low-rank approximation. Suppose we wish to approximate the centred data matrix X_c by a matrix of rank $p < r$, where $r = \text{rank}(X_c)$. The Eckart–Young Theorem 2.3 states that if $X_c = U\Sigma V^T$ is the SVD of X_c, and Σ_p be the diagonal matrix obtained by keeping only the largest p singular values, then the truncated SVD

$$X_{c,p} = U_p \Sigma_p V_p^T \in \mathbb{R}^{N\times d}$$

minimises the error $\|X_c - Y\|_F$ over all matrices Y of rank at most p. The matrix $V_p \in \mathbb{R}^{d\times p}$ contains the first p principal directions (loadings), and the PCA scores are

$$Z_p = X_c V_p = U_p \Sigma_p \in \mathbb{R}^{N\times p}.$$

The reconstruction of the centred data is

$$X_c \approx X_{c,p} = Z_p V_p^T \in \mathbb{R}^{N\times d}.$$

Remark 3.3 (Other data orientation)**.** *If samples are arranged as* columns *($X_c \in \mathbb{R}^{d\times N}$), the SVD still takes the form $X_c = U\Sigma V^T$, but now the rank-p approximation becomes*

$$X_{c,p} = U_p \Sigma_p V_p^T \in \mathbb{R}^{d\times N}$$

where $U_p \in \mathbb{R}^{d\times p}$ contains the principal directions and the PCA scores are

$$Z_p = U_p^T X_c = \Sigma_p V_p^T \in \mathbb{R}^{p\times N}.$$

The rank-k approximation becomes

$$X_c \approx X_{c,k} = U_p Z_p \in \mathbb{R}^{d\times N}.$$

Thus, the algebraic structure is identical, but the roles and dimensions *of U_p and V_p swap: U_p stores loadings and V_p stores sample coefficients.* □

3.5 Eigenfaces algorithm

Face recognition is a classical application of PCA. Each face image can be represented as a high-dimensional vector (stacking all pixels into a column). However, images are often highly correlated, so the effective dimensionality is much lower than the number of pixels. PCA identifies the directions of largest variation, called **eigenfaces**, which capture the main features of faces across a dataset. Projecting new images onto the subspace spanned by the leading eigenfaces enables efficient and robust recognition.

Let

$$X = [x_1, x_2, \ldots, x_N] \in \mathbb{R}^{d \times N}$$

be a dataset of N face images, where $d = h \cdot w$ is the number of pixels in each image, and x_i is the i-th image reshaped into a vector. Since facial images often contain correlations among pixels, the data is usually high-dimensional but lies near a lower-dimensional manifold.

Remark 3.4. *In image processing, the dataset is often organised with samples as columns, $X \in \mathbb{R}^{d \times N}$, where d is number of pixels and N the number of images.* □

The procedure is as follows.

1. The first step in PCA is to compute the mean face

$$\mu = \frac{1}{N} \sum_{i=1}^{N} x_i$$

   and centre the data

$$X_c = X - \mu \mathbf{1}_N^T,$$

   where $\mathbf{1}_N \in \mathbb{R}^{N \times 1}$ is a vector of ones. Centring ensures that the principal components capture variation around the mean face rather than the absolute brightness values.

2. Next, PCA are computed via the reduced (economy) SVD of the centred data

$$X_c = \hat{U} \hat{\Sigma} \hat{V}^T.$$

   where $\hat{U} \in \mathbb{R}^{d \times r}$, $\hat{V} \in \mathbb{R}^{N \times r}$ are orthogonal matrices containing the left and right singular vectors, $\Sigma \in \mathbb{R}^{r \times r}$ is a diagonal matrix of singular values, and $r = \text{rank}(X_c)$. In this context, the columns of U are called eigenfaces (principal directions).

3. Select the first $p \ll r$ eigenfaces corresponding to the largest singular values

$$U_p \in \mathbb{R}^{d \times p}.$$

   Note that the dimensionality reduction is achieved by retaining only the first p eigenfaces corresponding to the largest singular values. This not only reduces computation but also filters out noise and less informative features, while preserving the most significant variation among faces. In practice, p is chosen to capture a high percentage (e.g., 90-95%) of the total variance.

4. Project each centred image onto the eigenface space to obtain the PCA scores
$$Z_p = U_p^T X_c \in \mathbb{R}^{p \times N}.$$
5. For recognition, a new face image $y \in \mathbb{R}^d$ is centred by the mean face μ and projected onto the eigenface
$$z_y = U_k^T (y - \mu) \in \mathbb{R}^k.$$
The identity of the closest match is predicted by finding the nearest neighbour in the PCA score space using **distance measure** $\text{dist}(\cdot, \cdot)$
$$i^* = \arg \min_{i=1,\dots,N} \text{dist}(z_y, Z_k(:, i)).$$
Common distance measures for nearest-neighbour matching include (here lower values correspond to higher similarity):
   - Euclidean distance: $\text{dist}(y, x_i) = \|y - x_i\|$ or $\text{dist}(y, x_i) = \|y - x_i\|^2$;
   - Cosine distance: $\text{dist}(y, x_i) = 1 - \dfrac{y^T x_i}{\|y\| \, \|x_i\|}$;
   - Gaussian/RBF distance: $\text{dist}(y, x_i) = 1 - \exp\left(-\dfrac{\|y - x_i\|^2}{2\sigma^2}\right)$.
     The parameter $\sigma > 0$ controls the sensitivity to differences between feature vectors. A small σ makes the exponential decay very fast, meaning that only pairs of vectors with small Euclidean distance obtain a Gaussian distance close to 0. Conversely, a large σ makes the decay slower, so that even moderately distant vectors retain a relatively small Gaussian distance. In practice, the Gaussian distance is equivalent to the Euclidean distance.

Listing 3.1: MATLAB template to get started eigenfaces algorithm

```
% Parametres
trainF = 'path/to/folder';   % folder with grayscale images
files = dir(fullfile(trainF, '*.*'));  % all files
N = numel(files);                      % number of images

% Read the first image to get dimensions
img0 = imread(fullfile(trainF, files(1).name));
[h, w] = size(img0);         % image height and width
d = h * w;                   % total pixels per image
X = zeros(d, N);             % Preallocate matrix
% Read images and vectorize
for i = 1:N
    img = double(imread(fullfile(trainFolder, files(i).name)));
    X(:, i) = img(:); % store as column
end

% Step 1: Compute mean face and centre data
mu = mean(X, 2);           % mean face vector
Xc = X - mu;               % centre data
```

```
% Step 2: Compute SVD
[U, S, V] = svd(Xc, 'econ');

% Step 3: Select first p eigenfaces and project training images
p = 20; Up = U(:,1:p);          % top p eigenfaces
Z_train = Up' * Xc;     % project training images

% Step 4: Project a new image
new_img = double(imread(newImageFile));
z_new = Up' * (new_img(:) - mu);

% Step 5: Find nearest neighbour in PCA space and visualise
distances = vecnorm(Z_train - z_new, 2, 1);
[~, idx] = min(distances);
fprintf('Predicted identity: %d\n', idx);
recognized_face = X(:, idx);
figure;
subplot(1,2,1); imshow(reshape(new_img, h, w), []);
title('Test Image');
subplot(1,2,2); imshow(reshape(recognized_face, h, w), []);
title('Recognized Face');
```

Remark 3.5 (Similarity measure)**.** *Depending on whether sim is a distance or a similarity, one may use* arg min *(for distances) or* arg max *(for direct similarity measures) to find the most similar sample. If* $\text{sim}(\cdot,\cdot)$ *is a **similarity measure** (that increases with closeness), the predicted identity is obtained by*

$$i^* = \arg\max_{i=1,\dots,N} \text{sim}(z_y, Z_k(:,i)).$$

Common choices of similarity in this case are:

- *Cosine similarity:* $\text{sim}(y, x_i) = \dfrac{y^T x_i}{\|y\|\|x_i\|}$;
- *Gaussian/RBF similarity:* $\text{sim}(y, x_i) = \exp\Big(-\dfrac{\|y - x_i\|^2}{2\sigma^2}\Big)$.

  The parameter $\sigma > 0$ *controls the kernel width. For a small* σ*, only pairs of vectors with small Euclidean distance obtain a similarity close to 1. Conversely, for a large* σ*, moderately distant vectors retain a relatively high similarity. In practice,* σ *can be chosen as a function of the data, for example as the average norm of the training differences or a fraction of the standard deviation of pairwise distances. This makes the similarity computation stable and avoids numerical saturation of the exponential.*

For this case, the previous MATLAB script must be updated according to the following code (for the case of Gaussian similarity). □

```
% Gaussian similarity
sigma = 1.0;
sim = exp(-vecnorm(Xp - y, 2, 1).^2 / (2*sigma^2));
[~, idx] = max(sim); % predicted identity; idx gives the index of
   the most similar training sample
```

Remark 3.6 (Eigenfaces, PCA, and computational considerations)**.** *The eigenfaces approach provides a low-dimensional linear representation of face images and can be interpreted through PCA. Each column of U_p is an eigenface reshaped to $h \times w$, representing a dominant variation pattern among faces, while Z_p contains the coordinates of each training image in the low-dimensional eigenface space. This approach reduces the dimensionality from d to $k \ll d$, filtering noise, reducing storage, and speeding up recognition. Recognition performance generally improves with more components, but too many can include noise, so cross-validation is recommended. The method relies on a linear subspace approximation; variations in pose, lighting, or expression may require preprocessing or more advanced methods. It works naturally with the $d \times N$ convention for images as columns; for the $N \times d$ convention, the roles of U and V swap accordingly.* □

3.6 Face recognition procedure

The eigenfaces method provides a structured approach to face recognition by combining dimensionality reduction with nearest neighbour classification. The procedure can be summarised as follows:

1. **Load and vectorise images:** Each face image is reshaped into a d-dimensional column vector and collected into a matrix X; corresponding labels are stored.
2. **Split data into training and test sets:** A subset of images per subject is used for training, while the remaining images form the test set.
3. **Compute the mean face and centre training data:** The column-wise mean of the training data is subtracted to centre the data.
4. **Perform SVD on the centred training data:** Singular Value Decomposition provides the eigenfaces as the left singular vectors of X_c.
5. **Select top $p \ll d$ eigenfaces:** The eigenvectors corresponding to the largest singular values are retained to form the PCA subspace.
6. **Project training and test images into the PCA subspace:** Each image is represented by its coordinates (scores) in the low-dimensional eigenface space.
7. **Classify test images using nearest neighbour:** The Euclidean distance between the projected test image and training images is used to assign a label.
8. **(Optional) Classify using Gaussian similarity:** An alternative similarity measure applies a Gaussian kernel to the distance in PCA space.
9. **Visualise the mean face and eigenfaces:** The mean image and first few eigenfaces are reshaped to image form for inspection.
10. **Evaluate recognition accuracy:** The fraction of correctly classified test images quantifies performance.

Choosing the number of principal components k is critical: retaining too few may lose important facial information, while too many may include noise and increase computational cost. PCA provides several advantages in this context: it reduces dimensionality, captures the most discriminative features for face recognition, improves computational efficiency, and allows visualization of the dominant modes of variation among faces. However, it also has limitations: performance can be sensitive to variations in lighting, pose, expression, or occlusions; it assumes that faces lie approximately in a linear subspace and may fail to capture complex nonlinear variations; and recognition accuracy may be reduced when the training dataset is small relative to the dimensionality of the images.

Listing 3.2: Face recognition

```
% Parameters
h = 64; w = 64;            % image height and width
d = h * w;                 % number of pixels
Ntrain = 50; Ntest = 10;   % number of training and test images

% Steps 1/2: Load vectorise images and split into training and
   test sets
Xtrain = rand(d, Ntrain);  Xtest  = rand(d, Ntest);  % andom data
train_labels = 1:Ntrain;

% Step 3: Compute mean face and centre training data
mu = mean(Xtrain, 2); % mean along rows
Xc_train = Xtrain - mu;

% Step 4: Perform SVD on centred training data
[U, S, V] = svd(Xc_train, 'econ');

% Step 5: Select top p eigenvectors (eigenfaces)
p = 20; Up = U(:, 1:p);

% Step 6: Project training and test images into PCA subspace
Z_train = Up' * Xc_train;
Z_test  = Up' * (Xtest - mu);

% Step 7: Classify test images using nearest neighbour (Euclidean)
train_norms = sum(Z_train.^2, 1);        % 1 x Ntrain
test_norms  = sum(Z_test.^2, 1);         % 1 x Ntest
distances = zeros(Ntest, Ntrain);  % initialize distance matrix
for i = 1:Ntest
    for j = 1:Ntrain
        diff = Z_test(:, i) - Z_train(:, j);  % difference vector
        distances(i, j) = sqrt(sum(diff.^2)); % Euclidean norm
    end
end
[~, idx] = min(distances, [], 2); % Nearest-neighbor prediction
pred_labels = train_labels(idx)';

% Step 8 (optional): Classify using Gaussian similarity
sigma = 1.0;
sim_matrix = exp(-distances.^2 / (2*sigma^2));
[~, idx_gauss] = max(sim_matrix, [], 2);
pred_labels_gauss = train_labels(idx_gauss)';
```

```
% Step 9: Visualise mean face and first few eigenfaces
n_display = min(p, 10);              % number of eigenfaces to show
total_display = n_display + 1;       % include mean face
cols = ceil(sqrt(total_display)); rows = ceil(total_display/cols);

figure; subplot(rows, cols, 1);
imagesc(reshape(mu, h, w));
colormap gray; axis image off; title('Mean Face');

for i = 1:n_display
    subplot(rows, cols, i+1);
    imagesc(reshape(Up(:,i), h, w)); % Eigenfaces
    colormap gray; axis image off;
    title(['Eigenface ', num2str(i)]);
end

% Step 10: Display recognition results
disp('Predicted labels (Euclidean):'); disp(pred_labels);
disp('Predicted labels (Gaussian):'); disp(pred_labels_gauss);
```

The following MATLAB code implements the eigenfaces algorithm for face recognition using the AT&T database. The **AT&T (ORL) Face Database** is a classical benchmark dataset for evaluating face recognition algorithms. It contains 400 grayscale images of 40 subjects, with 10 images per subject, each of size 92×112 pixels. The images capture small variations in expression and pose, making it suitable for evaluating PCA-based recognition methods.

Listing 3.3: Face recognition using the AT&T database

```
% Step 1: Load vectorise images
datasetFolder = 'path/to/att_faces';  % set to your AT&T folder
h = 112; w = 92; d = h * w;
subjects = 40; imgsPerSubject = 10;
X = []; labels = [];

for s = 1:subjects
    subjFolder = fullfile(datasetFolder, sprintf('s%d', s));
    for j = 1:imgsPerSubject
        fname = fullfile(subjFolder, sprintf('%d.pgm', j));
        img = imread(fname); img = double(img(:));
        X = [X, img]; labels = [labels, s];
    end
end

N = size(X,2);

% Step 2: Split into training and test sets
train_idx = false(1, N); test_idx  = false(1, N);

for s = 1:subjects
    idx = find(labels == s);
    perm = randperm(imgsPerSubject);
```

```
    train_idx(idx(perm(1:6))) = true;  % 6 train
    test_idx(idx(perm(7:end)))   = true; % 4 test
end
Xtrain = X(:, train_idx);
Xtest  = X(:, test_idx);
train_labels = labels(train_idx);
test_labels  = labels(test_idx);

% Step 3: Compute mean face and centre training data
mu = mean(Xtrain,2); % mean along rows
Xc_train = Xtrain - mu;

% Step 4: Perform SVD on centred training data
[U, S, V] = svd(Xc_train, 'econ');

% Step 5: Select top p eigenvectors (eigenfaces)
p = 20; Up = U(:,1:p);

% Step 6: Project training and test images into PCA subspace
Z_train = Up' * Xc_train;
Z_test  = Up' * (Xtest - mu);

% Step 7: Classify test images using nearest neighbour (Euclidean)
pred_labels = zeros(1, size(Z_test,2));
for i = 1:size(Z_test,2)
    distances = vecnorm(Z_train - Z_test(:,i), 2, 1);
    [~, idx] = min(distances);
    pred_labels(i) = train_labels(idx);
end

% Step 8 (optional): Classify using Gaussian similarity
sigma = 800;
pred_labels_gauss = zeros(1,size(Z_test,2));
for i = 1:size(Z_test,2)
    sims = exp(-vecnorm(Z_train-Z_test(:,i),2,1).^2/(2*sigma^2));
    [~, idx] = max(sims);
    pred_labels_gauss(i) = train_labels(idx);
end

% Step 9: Visualise mean face and first few eigenfaces
numShow = min(p,20);       % show up to 20 eigenfaces
rows = 3; cols = ceil((numShow+1)/rows); % +1 for mean face

figure('Name','Mean Face + Eigenfaces'); subplot(rows,cols,1);
imagesc(reshape(mu,[h w]));
colormap gray; axis image off;
title('Mean Face','FontWeight','bold');

for i = 1:numShow
    subplot(rows,cols,i+1);
    imagesc(reshape(Up(:,i),[h w]));
```

```
    colormap gray; axis image off;
    title(sprintf('PC %d',i));   % or "Eigenface i"
end

% Step 10: Display recognition results
acc_e = mean(pred_labels == test_labels)*100;
acc_g = mean(pred_labels_gauss == test_labels)*100;
fprintf('Accuracy (Euclidean): %.2f %%\n',acc_e);
fprintf('Accuracy (Gaussian):  %.2f %%\n\n',acc_g);
```

To conclude, PCA-based face recognition via eigenfaces provides a principled and efficient way to reduce high-dimensional face data to a compact representation that preserves the most relevant information for classification. While simple and effective under controlled conditions, its performance may be limited by nonlinear variations in the data, motivating extensions such as kernel PCA or deep learning approaches for more challenging scenarios.

3.7 UK food consumption

Suppose that we are examining the data in Table 3.1, from the UK's "Department for Environment, Food and Rural Affairs" (DEFRA), showing the consumption in grams (per person, per week) of 17 different types of foodstuff measured and averaged in the four countries of the United Kingdom in 1997[2].

Food type	England	Wales	Scotland	North Ireland
Cheese	105	103	103	66
Carcass meat	245	227	242	267
Other meat	685	803	750	586
Fish	147	160	122	93
Fats and oils	193	235	184	209
Sugars	156	175	147	139
Fresh potatoes	720	874	566	1033
Fresh Veg	253	265	171	143
Other Veg	488	570	418	355
Processed potatoes	198	203	220	187
Processed Veg	360	365	337	334
Fresh fruit	1102	1137	957	674
Cereals	1472	1582	1462	1494
Beverages	57	73	53	47
Soft drinks	1374	1256	1572	1506
Alcoholic drinks	375	475	458	135
Confectionery	54	64	62	41

Table 3.1: Per-person weekly consumption (grams) of various food-types in the four UK countries (1997). Data from DEFRA.

We aim to perform PCA on the UK food consumption dataset to explore patterns

[2]See: (https://bioboot.github.io/bggn213_W19/class-material/lab-8-bggn213.html)

among the four countries (England, Wales, Scotland, Northern Ireland) and the relationships between different food types. The procedure is as follows

1. **Define the data matrix.** Let

$$X \in \mathbb{R}^{17\times 4}$$

contain the per-person weekly consumption (grams) of 17 foods (rows/samples) across 4 countries (columns/features). Assign labels for the foods and countries.

2. **Center the data.** Subtract the mean of each row (food type) so that each row has zero mean:

$$X_c = X - \mathbf{1}_N \mu^T, \quad .\text{for } \mu_j = \frac{1}{17}\sum_{i=1}^{17} X_{ij}, \quad j = 1, 2, 3, 4.$$

3. **Scale for covariance eigenvalues.** Divide by $\sqrt{N-1}$ so that the singular values of the centred matrix correspond to the eigenvalues of the covariance matrix:

$$X_c^{\mathrm{s}} = \frac{X_c}{\sqrt{N-1}}.$$

4. **Compute the SVD.** Decompose the scaled centred matrix:

$$X_c^{\mathrm{s}} = \hat{U}\hat{\Sigma}\hat{V}^T,$$

where the columns of $\hat{U}$ are the principal directions (food loadings), $\hat{V}$ contains the right singular vectors (country patterns), and S^2 gives the covariance eigenvalues.

5. **Compute explained variance.** Compute the percentage of explained variance by each principal component:

$$E_i = \frac{\sigma_i^2}{\sigma_1^2 + \cdots + \sigma_r^2} \times 100\%,$$

where σ_j, $j = 1, \ldots, r$, are the (positive) singular values of X_c^{s}.

6. **Project countries onto the first two principal components.** Compute the PCA scores ($p = 2$) via truncated SVD (see Remark 3.7):

$$Z_p = X_c^{\mathrm{s}\,T} U_p = V_p \Sigma_p.$$

7. **Visualise countries in PCA space.** Create a scatter plot of countries in the PC1–PC2 plane and label them.

8. **Biplot of foods and countries.** Generate a biplot combining food loadings (columns of U_p) and the projected country scores.

9. **Cluster countries.** Apply k-means clustering (`number_clusters = 2`) to the scores and display cluster assignments. Optionally, plot the countries coloured by cluster.

Exercise 3.1. *Use the next MATLAB script to perform the steps above on the AT&T food dataset. The code includes PCA computation, projection, visualisation, and clustering of countries in the PCA space.*

Listing 3.4: PCA on UK food dataset

```
% PCA on UK food dataset (scaled for covariance eigenvalues)
clc; clear; close all;

% 1. Define data matrix X (17 foods x 4 countries)
X = [
   105, 103, 103,  66;    % Cheese
   245, 227, 242, 267;    % Carcass_meat
   685, 803, 750, 586;    % Other_meat
   147, 160, 122,  93;    % Fish
   193, 235, 184, 209;    % Fats_and_oils
   156, 175, 147, 139;    % Sugars
   720, 874, 566,1033;    % Fresh_potatoes
   253, 265, 171, 143;    % Fresh_Veg
   488, 570, 418, 355;    % Other_Veg
   198, 203, 220, 187;    % Processed_potatoes
   360, 365, 337, 334;    % Processed_Veg
   1102,1137, 957, 674;   % Fresh_fruit
   1472,1582,1462,1494;   % Cereals
   57,   73,  53,  47;    % Beverages
   1374,1256,1572,1506;   % Soft_drinks
   375, 475, 458, 135;    % Alcoholic_drinks
   54,   64,  62,  41     % Confectionery
];

[M,N] = size(X);  % M=17 foods, N=4 countries

foods = {'Cheese','Carcass_meat','Other_meat','Fish','
   Fats_and_oils','Sugars', ...
    'Fresh_potatoes','Fresh_Veg','Other_Veg','Processed_potatoes',
       'Processed_Veg', ...
    'Fresh_fruit','Cereals','Beverages','Soft_drinks','
       Alcoholic_drinks','Confectionery'};
countries = {'England','Wales','Scotland','N.Ireland'};

% 2. Center the data (subtract mean along rows)
Xc = X - mean(X,2);  % centred (row-wise mean zero)

% 3. Scale by sqrt(N-1) so singular values = covariance
   eigenvalues
Xc_scaled = Xc / sqrt(N-1);

% 4. Perform SVD
[U,S,V] = svd(Xc_scaled,'econ');

% 5. Compute the explained variance
```

```
cov_eigenvalues = diag(S).^2;
explained_var = cov_eigenvalues / sum(cov_eigenvalues) * 100;
fprintf('Explained variance by each component (%%):\n');
disp(explained_var');

% 6. Project countries onto first p = 2 principal components
p = 2; scores = Xc_scaled' * U(:,1:p);
% p = 2; scores = V(:,1:p) * S(1:k,1:p);  % use first 2 PCs

% 7. Plot countries in PC1-PC2 space
figure;
scatter(scores(:,1), scores(:,2),100,'filled');
text(scores(:,1)+0.05, scores(:,2), countries);
xlabel('PC1'); ylabel('PC2'); title('Countries in PCA space');
grid on;

% 8. Biplot of foods and countries
figure;
biplot(U(:,1:p),'Scores',scores,'VarLabels',foods);
title('PCA Biplot: Foods and Countries');

% 9. Cluster countries
num_clusters = 2;
[idx,C] = kmeans(scores,num_clusters);
countries = {'England','Wales','Scotland','N.Ireland'};
for i = 1:length(countries)
    fprintf('%s -> cluster %d\n', countries{i}, idx(i));
end

% Optional: plot countries colored by cluster
figure;
gscatter(scores(:,1), scores(:,2), idx, 'rb', 'o', 8);
text(scores(:,1)+0.05, scores(:,2), countries);
xlabel('PC1'); ylabel('PC2');
title('Countries clustered in PCA space');
grid on;
```

Remark 3.7 (Projection of samples versus features)**.** *In PCA, the scores of the samples are usually computed (via SVD of X_c) as*

$$Z = X_c\hat{V} = \hat{U}\hat{\Sigma} \in \mathbb{R}^{N\times r},$$

where $\hat{V}$ contains the principal directions (loadings) and $\hat{U}\hat{\Sigma}$ gives the coordinates of each sample in the principal component space.

Alternatively (as in the previous code), one can compute

$$Z = X_c^T\hat{U} = \hat{V}\hat{\Sigma} \in \mathbb{R}^{d\times r},$$

which has the same singular values and principal directions, but the roles of rows and columns are swapped: here the rows correspond to the features and the columns to the principal components.

*This formulation is particularly convenient for applications such as **biplots**, where the goal is to visualise the contributions of features along the first principal components. In a biplot, each row of Z can be interpreted as a vector representing a feature, which can be displayed as arrows in the component space.*

Hence, while $X_c\hat{V}$ provides the scores for the samples, the alternative $X_c^T\hat{U}$ directly yields the scaled loadings suitable for feature visualisation. Both expressions are consistent with the SVD of X_c, but their use depends on the intended focus: sample representation or feature representation. □

3.8 Netflix-style recommendation

The goal of this example is to predict missing entries in a user-item rating matrix using a low-rank approximation based on the SVD. The procedure is as follows:

1. **Define the user-item matrix.** Construct a matrix $R \in \mathbb{R}^{N\times d}$ where each row corresponds to a user and each column to an item (e.g., a movie). Missing ratings are represented by `NaN`.

2. **Handle missing values.** Replace the missing entries with the mean rating of the corresponding item, producing a fully populated matrix R_f.

3. **Center the data.** Subtract the mean rating of each item to obtain the centred matrix R_c. This ensures that the principal components capture variations rather than absolute values.

4. **Compute the SVD.** Factorise the centred matrix:
$$R_c = \hat{U}\hat{\Sigma}\hat{V}^T,$$
where the columns of $\hat{U}$ and $\hat{V}$ are left and right singular vectors, and $\hat{\Sigma}$ contains the (positive) singular values.

5. **Low-rank approximation.** Choose a rank $p \ll \min(M, d)$ and approximate the matrix by keeping only the first p singular values and vectors:
$$R_p = U_p\Sigma_p V_p^T.$$

6. **Add back the mean.** To return to the original scale:
$$R_{\text{pred}} = R_p + \text{mean}(R_f).$$

7. **Predict missing ratings.** Use entries in R_{pred} corresponding to missing ratings in R.

8. **Recommend items.** For each user, identify the unrated item with the highest predicted rating and recommend it.

Listing 3.5: Netflix-style recommendation using SVD

```
% 1. Define user-item rating matrix
R = [5 3 NaN 1;
     4 NaN NaN 1;
     1 1 NaN 5;
     1 NaN NaN 4;
     NaN 1 5 4];
[N, d] = size(R);

% 2. Handle missing values
col_mean = nanmean(R); % omitting NaN (or mean(R,'omitnan'))
Rf = R;
for j = 1:d
    Rf(isnan(R(:,j)), j) = col_mean(j);
end

% 3. Centre the data
Rc = Rf - mean(Rf); % mean along columns (or mean(Rf,1))

% 4. Compute SVD
[U, S, V] = svd(Rc, 'econ');

% 5. Low-rank approximation (rank p)
p = 2; Rp = U(:,1:p) * S(1:p,1:p) * V(:,1:p)';

% 6. Add back column means
R_pred = Rp + mean(Rf);

% 7. Predict missing ratings
fprintf('Predicted ratings for missing entries:\n');
for i = 1:N
    for j = 1:d
        if isnan(R(i,j))
            fprintf('User %d, Movie %d: %.2f\n', i,j,R_pred(i,j));
        end
    end
end

% 8. Recommend top unrated movie for each user
fprintf('\nRecommended movie for each user:\n');
for i = 1:N
    unrated = isnan(R(i,:));
    if any(unrated)
        [~, idx] = max(R_pred(i, unrated));
        unrated_idx = find(unrated);
        recommended_movie = unrated_idx(idx);
        fprintf('User %d: Movie %d\n', i, recommended_movie);
    else
        fprintf('User %d: No unrated movies\n', i);
    end
end
```

Remark 3.8 (Automatic selection of rank p)**.** *In a general PCA or SVD setting, the rank p of a low-rank approximation can be chosen automatically by examining the singular values σ_i, $i = 1, \cdots, r$, of the centred data matrix X_c, with $r = \mathrm{rank}(X_c)$. Define the* ***explained variance of each component*** *as*

$$EV_i = \frac{\sigma_i^2}{\sigma_1^2 + \cdots + \sigma_r^2},$$

and the ***cumulative explained variance*** *as*

$$CV_i = \sum_{j=1}^{i} EV_j,$$

The rank $p \ll r$ is selected as the smallest number of components such that $CV_p \geq \alpha$, where α is a chosen threshold (commonly 0.9 or 0.95). This ensures that the low-rank approximation retains a desired proportion of the total variance, balancing dimensionality reduction and information preservation. □

3.9 Feature reduction via PCA

This final section highlights feature reduction as a central application of PCA, showing how the concepts and methods discussed in the preceding sections – centring, covariance structure, principal directions, and scores – come together to provide a compact, informative representation of the data.

One of the main applications of Principal Component Analysis (PCA) is **feature reduction**, which represents the dataset

$$X \in \mathbb{R}^{N \times d}$$

in a lower-dimensional space ($p < d$) while preserving the most relevant information. Let $X_c \in \mathbb{R}^{N \times d}$ denote the centred data. Feature reduction can be described from two complementary perspectives. In the **covariance perspective**, the covariance matrix

$$S = \frac{1}{N-1} X_c^T X_c \in \mathbb{R}^{d \times d}$$

is computed, and its eigenvectors corresponding to the p largest eigenvalues form

$$V_p \in \mathbb{R}^{d \times p}.$$

The columns of V_p are the **principal directions** (loadings), and projecting the centered data onto these directions

$$Z_p = X_c V_p \in \mathbb{R}^{N \times p}$$

yields the **PCA scores**, forming the reduced dataset. Equivalently, in the **SVD perspective**, one computes

$$X_c = \hat{U} \hat{\Sigma} \hat{V}^T,$$

and retains the first p columns of $\hat{V}$, so that

$$Z_p = X_c V_p = U_p \Sigma_p \in \mathbb{R}^{N \times p}.$$

The singular values in Σ_p correspond to the square roots of the largest eigenvalues of S, explicitly linking both approaches.

To illustrate the feature reduction process in low dimensions, consider first a 2D dataset reduced to 1D. Figure 3.3 shows the original data (blue), the principal direction (red arrow), and the projected points along the first principal component (magenta), which form the reduced dataset. The reconstructed points (green circles) coincide with the projections, demonstrating that all relevant variance is captured along the first principal direction.

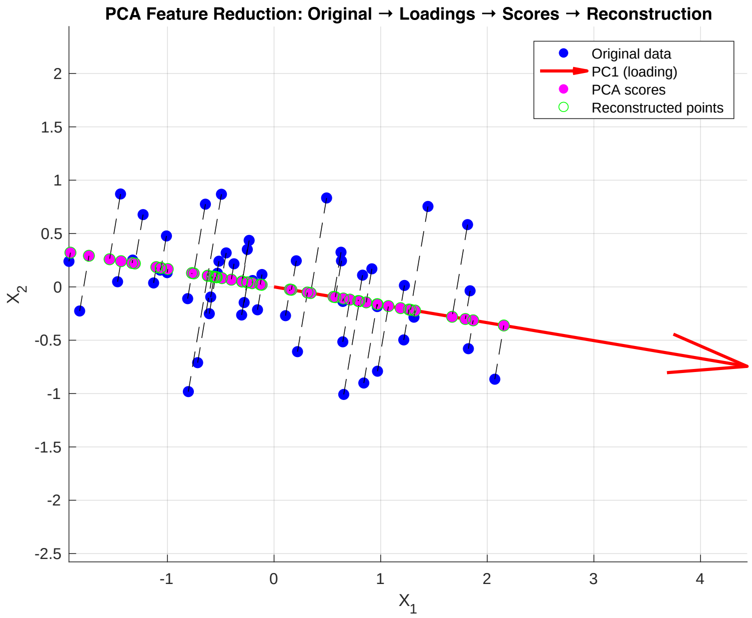


Figure 3.3: 2D $\to$ 1D PCA feature reduction.

For higher-dimensional data, PCA reduces dimensionality while retaining most variance. As an example, Figure 3.4 illustrates a 3D dataset projected onto the plane spanned by the first two principal components (red and green arrows). The projections (magenta points) form a 2D representation of the original data, preserving the dominant structure.

The number of principal components p is chosen based on the **cumulative explained variance**, which is computed from the singular values of the centred data X_c

$$\frac{\sigma_1^2 + \cdots + \sigma_p^2}{\sigma_1^2 + \cdots + \sigma_r^2} \geq \alpha,$$

where σ_i is the i-th singular value of X_c, $r = \text{rank}(X_c)$, and $\alpha \in (0, 1)$ is a predefined threshold, typically 0.9 or 0.95. Here, σ_i^2 is proportional to the variance explained by the i-th principal component. This ensures that the reduced representation captures most of the variability while reducing computational cost and the risk of overfitting.

The rank-p approximation of the centred data is

$$X_c \approx X_{c,p} = Z_p V_p^T \in \mathbb{R}^{N \times d},$$

showing that each sample is effectively represented by a p-dimensional vector instead of d, preserving the main structure. Hence, PCA serves as a tool for *feature extraction* and data compression, widely applied in image processing, genomics, finance, and machine learning.

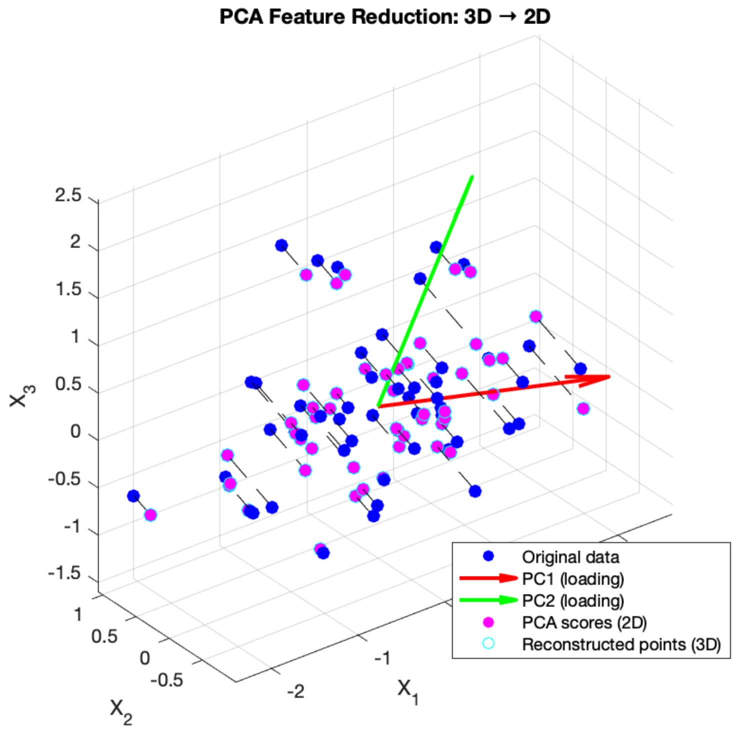


Figure 3.4: 3D $\to$ 2D PCA feature reduction.

Remark 3.9 (Data matrix with features as rows)**.** *When the data matrix is arranged as*

$$X \in \mathbb{R}^{d \times N},$$

with features as rows and samples as columns, let $X_c \in \mathbb{R}^{d \times N}$ *denote the centred data. The covariance matrix is then*

$$S = \frac{1}{N-1} X_c X_c^T \in \mathbb{R}^{d \times d}.$$

The principal directions are the eigenvectors of S*, and the PCA scores are*

$$Z_p = V_p^T X_c \in \mathbb{R}^{p \times N},$$

where $V_p \in \mathbb{R}^{d \times p}$ *contains the first* p *eigenvectors. The structure of PCA is unchanged: loadings span the subspace of maximal variance, and scores provide coordinates of samples in the reduced feature space.* □

Feature reduction via PCA captures the essential variability in a lower-dimensional representation, simplifies subsequent analysis, and links linear-algebraic structure to statistical interpretation, making it a fundamental tool for dimensionality reduction, efficient data representation, and compression.

3.10 Exercises

Hand-solved exercises

1. **SVD of a 2×2 matrix.** Compute the singular value decomposition of $A = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$. Identify U, S, and V. Discuss which direction captures most variance.

2. **Covariance and PCA.** Given $X = \begin{bmatrix} 1 & 2 \\ 0 & -1 \\ -1 & -1 \end{bmatrix}$, compute the covariance matrix and its eigenvalues. Explain what the first principal component represents in terms of the data.

3. **Projection onto first principal component.** Using the previous matrix, project the data onto the first PC and interpret the resulting scores.

4. **Diagonal vs non-diagonal matrices.** For $B = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$, compute the SVD. Discuss why singular values correspond to the variance along coordinate axes.

5. **Centered 2×3 dataset.** Create a small centred dataset and identify the first principal component by hand. Explain why centring is necessary.

6. **Diagonal matrix SVD property.** Show that for a diagonal matrix D, the singular values are simply the absolute values of the diagonal entries.

7. **Small dataset PCA.** Let $X = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$. Centre the data, compute the first principal component, and discuss the meaning of the PC direction.

8. **Variance explained.** Suppose singular values are $\sigma_1 = 5$ and $\sigma_2 = 2$. Compute the fraction of total variance explained by the first principal component.

9. **Relation between SVD and PCA scores.** Explain how the right singular vectors V of the centred data matrix relate to PCA scores.

10. **Dominant PC in a diagonal covariance matrix.** Let $S = \text{diag}(1, 2, 3)$. Compute the SVD and identify which component explains the largest portion of variance.

11. **Featrure reduction.** Consider the following dataset with 3 samples and 2 features:

$$X = \begin{bmatrix} 2 & 0 \\ 0 & 1 \\ 3 & 1 \end{bmatrix}.$$

   (a) Center the data by subtracting the mean of each feature and compute the covariance matrix S of the centred data.

   (b) Identify the principal components and determine which direction captures the most variance.

   (c) Project the original data onto the first principal component to obtain a 1D representation.

MATLAB exercises

1. **Image compression**
   Dataset: Built-in MATLAB image `cameraman.tif` or any grayscale image from USC-SIPI Image Database.
   Task: Load the image, convert to double precision, compute the SVD, and reconstruct the image using the top 10%, 20%, and 50% singular values. Compare the images visually and compute the compression ratio.

```
pkg load image; % for Octave
img = imread('cameraman.tif');
img = double(img);
[U,S,V] = svd(img);
p_values = [floor(0.1*min(size(img))), floor(0.2*min(size(img)
   )), floor(0.5*min(size(img)))];
for p = p_values
    img_p = U(:,1:p) * S(1:p,1:p) * V(:,1:p)';
    figure; imshow(uint8(img_p));
    title(['Rank-', num2str(p)]);
end
```

2. **Facial recognition**
   Dataset: Yale Faces dataset (link) or AT&T (ORL) Faces.
   Task: Load a subset of images, reshape each into a vector to build a data matrix X, perform SVD, and reconstruct images using top singular values. Analyse dimensionality reduction and reconstruction error.

```
% Load images as columns of X
[U,S,V] = svd(X,'econ');
p = 20; % top singular values
Xp = U(:,1:p) * S(1:p,1:p) * V(:,1:p)';
% Reshape columns back to images to display
```

3. **Recommender systems**
   Dataset: MovieLens 100K (link)
   Task: Construct user-item rating matrix R with missing entries as zeros or NaNs, perform truncated SVD, and approximate missing ratings. Evaluate predictions using RMSE.

```
R = load('u1.base'); % preprocess to create matrix
[U,S,V] = svd(R,'econ');
p = 2; % top singular values
R_approx = U(:,1:p) * S(1:p,1:p) * V(:,1:p)';
```

4. **PCA with Fisher's Iris dataset**
   Dataset: Fisher's Iris dataset (`load fisheriris` in MATLAB).
   Task: Take the feature matrix, centre it by subtracting the mean, compute SVD, project onto first 2 principal components, and plot.

```
pkg load statistics  % for Octave (if not already loaded)
load fisheriris
X = meas;
Xc = X - mean(X,1); % mean along columns (or mean(X))
[U,S,V] = svd(Xc,'econ');
PC1 = Xc * V(:,1); PC2 = Xc * V(:,2);
% Map species to numeric labels
speciesNum = grp2idx(species);  % 1 = setosa, 2 = versicolor,
   3 = virginica
% Define colors for each species
colors = [1 0 0; 0 1 0; 0 0 1];  % red, green, blue
figure; hold on;
for i = 1:3
    idx = (speciesNum == i);
    scatter(PC1(idx), PC2(idx), 50, colors(i,:), 'filled');
end
xlabel('PC1'); ylabel('PC2');
title('PCA of Fisher Iris Dataset');
legend('setosa','versicolor','virginica','Location','best');
grid on; hold off;
```

5. **Linear regression via SVD**
   Dataset: Boston Housing datase.
   Task: Load dataset, construct matrix A including a column of ones, compute SVD, solve least-squares via $x = \hat{V}\hat{\Sigma}^{-1}\hat{U}^*b$, and compare predicted vs true house prices.

```
clear; clc; close all;
data = readmatrix('BostonHousing.csv');
A = data(:,1:13);
b = data(:,14);
A = [ones(size(A,1),1) A]; % add intercept
[U,S,V] = svd(A,'econ');
x = V * inv(S) * U' * b;
b_pred = A * x;
plot(b,'k-o'); hold on; plot(b_pred,'r--o');
xlabel('House index'); ylabel('Price ($1000)');
legend('True','Predicted');
```

6. **Climate data analysis.** Use climate-related datasets (e.g., temperature, CO_2, precipitation) to uncover correlations, dominant modes, and potential relationships among environmental variables. Construct a matrix with locations as columns and time points as rows. Apply SVD to identify dominant climate patterns.

```
data = readmatrix('temperature.csv'); % rows=time, cols=
   locations
[U,S,V] = svd(data,'econ');
plot(diag(S) / sum(diag(S)),'o'); % Energy captured by modes
```

Climate datasets (downloadable CSV, NetCDF, or similar formats), examples:

- NASA GISS Surface Temperature Data;
- NOAA CO_2 Concentration Data;
- Global Precipitation Climatology Project (GPCP);
- Berkeley Earth Temperature Data; UCI ML Repository).

7. **Movie recommendation (Netflix-style).** Predict missing ratings in a synthetic user-movie matrix. Determine the optimal number of principal components based on cumulative variance, reconstruct the matrix, and recommend top movies for each user.

```
% Synthetic ratings (NaN = missing)
X = [5 NaN 3 1 4; 4 2 NaN 5 NaN; NaN 4 5 2 3; 1 5 2 NaN 4; 3
   NaN 4 5 2; 5 3 NaN 4 NaN; 2 4 5 NaN 3];
[N, d] = size(X);
% Handle missing values
col_mean = nanmean(X, 1);        % Mean by column ignoring NaNs
Xf = X;
for j = 1:d
    Xf(isnan(X(:,j)), j) = col_mean(j);
end
% Center the data
Xc = Xf - mean(Xf,1);            % Subtract column means
% Compute SVD
[~,S,V] = svd(Xc,'econ');
% Automatic selection of p (number of principal components)
singvals = diag(S);
p = find(cumsum(singvals.^2 / sum(singvals.^2)) >= 0.9,1);
% Reconstruct matrix
X_pred = Xc * V(:,1:p) * V(:,1:p)' + mean(Xf,1);
% Top recommendations
nan_idx = isnan(X);  % <-- needed for finding missing ratings
num_top = 2;
top_recs = cell(N,1);
for i = 1:N
    missing = find(nan_idx(i,:));
    if isempty(missing), continue; end
    [~,ord] = sort(X_pred(i,missing),'descend');
    top_recs{i} = missing(ord(1:min(num_top,length(ord))));
end
% Display recommendations
for i = 1:N
    fprintf('User %d recommended movies: %s\n', i, mat2str(
       top_recs{i}));
end
```

8. **Video compression.** Select a short video clip, treat each frame as a column in a matrix, compress using SVD, and reconstruct the video.

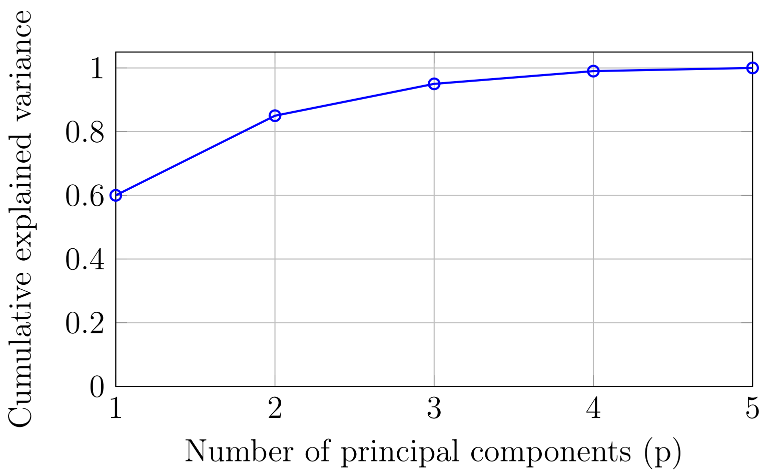


Figure 3.5: Example of cumulative explained variance as a function of the number of principal components. The optimal p is chosen where the curve reaches the desired threshold (e.g., 90%).

```
video = VideoReader('sample.mp4');
frame = read(video,1);
X = double(rgb2gray(frame));
[U,S,V] = svd(X);
p = 30; Xp = U(:,1:p) * S(1:p,1:p) * V(:,1:p)';
imshow(uint8(Xp));
```

9. **Image compression.** Load a grayscale image from a dataset (e.g., CIFAR-10). Apply SVD and reconstruct the image using truncated singular values. Compare storage and visual quality. Update for RGB images.

```
% Load CIFAR-10 batch
load('data_batch_2.mat');  % loads 'data'
% Take the first image
img_row = data(1, :);
% Extract RGB channels and reshape to 32x32
R = reshape(img_row(1:1024), [32,32])';
G = reshape(img_row(1025:2048), [32,32])';
B = reshape(img_row(2049:3072), [32,32])';
% Convert to grayscale (average of RGB)
I = double((R + G + B)/3);
% Apply SVD compression
[U,S,V] = svd(I);
p = 20;  % number of singular values
Ip = U(:,1:p) * S(1:p,1:p) * V(:,1:p)';
% Enlarge for display (e.g., 8 x bigger)
Ip_large = imresize(Ip, 8);
% Display the compressed image
imshow(uint8(Ip_large));
title(['CIFAR-10 Compressed with p = ', num2str(p)]);
```

10. **Medical imaging.** Load MRI/CT images, apply SVD for denoising or compression, and visualise reconstructed images for different truncation levels.

```
I = double(imread('mri_slice.png'));
[U,S,V] = svd(I);
p = 20; Ip = U(:,1:p) * S(1:p,1:p) * V(:,1:p)';
imshow(uint8(Ip));
```

11. **Social network analysis.** Use an adjacency matrix of a social network. Apply SVD to identify dominant clusters or communities.

```
A = load('social_net.mat'); % adjacency matrix
[U,S,V] = svd(A);
p = 3; nc = 3; clusters = kmeans(U(:,1:p),nc);
```

12. **Financial market data.** Take stock price data for multiple companies. Construct a matrix of returns and apply SVD to find principal market modes.

```
prices = readmatrix('stocks.csv'); % rows=time, cols=stocks
returns = diff(prices) ./ prices(1:end-1,:);
[U,S,V] = svd(returns,'econ');
plot(diag(S) / sum(diag(S)),'o');
```

13. **Sensor data compression.** Load multi-sensor IoT data over time. Use SVD to reduce dimensionality and reconstruct signals with minimal loss.

```
X = readmatrix('sensor.csv'); % rows=time, cols=sensors
[U,S,V] = svd(X,'econ');
p = 5; Xp = U(:,1:p) * S(1:p,1:p) * V(:,1:p)';
```

14. **Audio compression.** Load an audio signal, represent it as a matrix using overlapping frames, apply SVD, and reconstruct the signal using truncated singular values.

```
[audio,Fs] = audioread('speech.wav');
frame = buffer(audio,256);
[U,S,V] = svd(frame,'econ');
p = 50; frame_p = U(:,1:p) * S(1:p,1:p) * V(:,1:p)';
audio_p = reshape(frame_p,size(frame_p,1)*size(frame_p,2),1);
sound(audio_p,Fs);
```

15. **Gene expression analysis.** Use microarray gene expression data. Construct a gene-by-sample matrix and apply SVD to identify dominant expression patterns across conditions.

```
expr = readmatrix('gene_expression.csv'); % rows=genes, cols=
   samples
[U,S,V] = svd(expr,'econ');
p = 5; top_patterns = U(:,1:p);
```

16. **Feature reduction with synthetic dataset.** Use the MATLAB template below to complete the missing steps and answer: (a) What are the PCA scores? (b) How many components retain most variance? (c) Compare original and reconstructed data: do they match? Explain. (d) Plot original vs reconstructed points: what do you observe?

```
%% PCA Template: Synthetic Data
X = [...]; % define your dataset

% Step 1: Center the data
Xc = X - mean(X);

% Step 2: Compute SVD
% YOUR CODE HERE

% Step 3: Project data onto first p principal components
% YOUR CODE HERE

% Step 4: Reconstruct data from reduced dimensions
% YOUR CODE HERE

% Step 5: Plot original and reconstructed points
% YOUR CODE HERE
```

17. **Feature reduction with Boston Housing dataset.** Use the Boston Housing dataset with the template below and complete the missing steps. Answer: (a) What do the first 2–3 principal components represent? (b) Project onto the first 2 PCs: any patterns or clusters? (c) Reconstruct the original features: how accurate is the approximation? (d) How many components retain $\geq 90\%$ variance? Justify.

```
%% PCA Template: Boston Housing
data = readmatrix('BostonHousing.csv'); % load dataset
X = data(:,1:end-1); % select features

% Step 1: Centre the data
Xc = X - mean(X);

% Step 2: Compute SVD
% YOUR CODE HERE

% Step 3: Project data onto first 2 or 3 principal components
% YOUR CODE HERE

% Step 4: Reconstruct data from reduced dimensions
% YOUR CODE HERE

% Step 5: Visualise scores and/or reconstruction
% YOUR CODE HERE
```

Chapter 4

Linear Discriminant Analysis

4.1 Introduction

In previous chapters, we studied the Singular Value Decomposition (SVD) and Principal Component Analysis (PCA), which are fundamental tools for dimensionality reduction and data representation. PCA identifies directions of maximal variance in an *unsupervised* manner, but it does not take into account class labels, which can be critical for classification tasks. PCA, introduced by Karl Pearson in 1901, operates under the premise that the most informative features are those along which the variance is largest. Mathematically, PCA identifies a set of orthogonal axes that capture the maximum spread of the data, either by computing the eigenvectors of the covariance matrix or equivalently via the SVD of the centred data matrix. Each principal component represents a linear combination of the original features, and projecting the data onto the first few components reduces dimensionality while retaining most of the variance. PCA is widely used for exploratory analysis, noise reduction, and data compression, especially when the intrinsic geometry of the entire dataset is of interest.

In contrast, Linear Discriminant Analysis (LDA) is a **supervised** dimensionality reduction technique: it explicitly seeks directions that maximise class separability, making it particularly suitable for classification tasks where preserving discriminatory information is critical.

Linear Discriminant Analysis (LDA), in contrast, is inherently *supervised* and leverages class labels to guide the dimensionality reduction process. Originally introduced by Sir Ronald Fisher[1] in 1936 to distinguish between species. In his seminal 1936 paper, *The Use of Multiple Measurements in Taxonomic Problems*, Fisher introduced the linear discriminant to distinguish between three species of Iris flowers. Unlike Pearson's geometric approach to variance, Fisher's goal was explicitly classification. He sought a linear combination of features that separated the means of different classes while keeping the data within each class tightly clustered. This was later generalised to multi-class problems (Multiple Discriminant Analysis) by Calyampudi Radhakrishna Rao in 1948.

[1]According to Wikipedia, Sir Ronald Aylmer Fisher FRS (17 February 1890 – 29 July 1962) was a British polymath who was active as a mathematician, statistician, biologist, geneticist, and academic. He has been described as "a genius who almost single-handedly created the foundations for modern statistical science" and "the single most important figure in 20th century statistics".

Figure 4.1: Ronald Fisher (1890–1962).

LDA aims to find projections that maximise the separation between different classes while minimising the variability within each class. The main goals of LDA are to reduce the dimensionality of the data, retain the features that are most informative for distinguishing between classes, and improve the performance of subsequent classification tasks. Unlike PCA, which identifies directions of maximal variance without considering class information, LDA focuses on the directions that carry the most discriminatory information, ensuring that the resulting lower-dimensional representation preserves class separability and facilitates more effective analysis and recognition.

The choice between PCA and LDA depends on the data and analytical goals. For unlabelled data or tasks focused on general simplification, PCA is preferred. For labeled datasets where classification is the primary objective, LDA is more effective because it preserves the directions most relevant for distinguishing between classes. Understanding this distinction – the unsupervised pursuit of variance versus the supervised pursuit of separability – is essential for applying dimensionality reduction effectively in modern machine learning and statistical analysis.

4.2 Connection to PCA and SVD

Linear Discriminant Analysis (LDA) can be understood from a linear-algebra perspective closely related to Principal Component Analysis (PCA).

Let

$$X = [x_1, \dots, x_N] \in \mathbb{R}^{d\times N}$$

denote the data matrix, where d is the number of features and N is the number of samples, and let

$$X_c = X - \mu\mathbf{1}_N^T$$

be the centred data, with $\mathbf{1}_N \in \mathbb{R}^{N\times 1}$ the column vector of ones and $\mu \in \mathbb{R}^d$ the overall mean vector

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i.$$

PCA overview. In PCA, the principal directions $U_p \in \mathbb{R}^{d\times p}$, with $p \ll r = \operatorname{rank}(X_c)$, are obtained from a low-rank approximation of the covariance matrix

$$S = \frac{1}{N-1} X_c X_c^T$$

via the truncated Singular Value Decomposition (SVD):

$$X_c \approx X_{c,p} = U_p \Sigma_p V_p^T,$$

where $U_p \in \mathbb{R}^{d\times p}$ and $V_p \in \mathbb{R}^{N\times p}$ have orthonormal columns, and $\Sigma_p \in \mathbb{R}^{p\times p}$ is diagonal with singular values $\sigma_1 \geq \cdots \geq \sigma_p > 0$. The PCA scores of the samples (the projection onto the principal subspace) are then

$$Z_{\text{PCA}} = U_p^T X_c = \Sigma_p V_p^T,$$

depending on the chosen convention. Each column of Z_{PCA} represents a sample in the p-dimensional principal component space.

Remark 4.1 (Alternative data orientation). *If the data is arranged as $X_c \in \mathbb{R}^{N\times d}$ (samples in rows, features in columns), the principal directions are the columns of $V_p \in \mathbb{R}^{d\times p}$, and the PCA scores are*

$$Z_{PCA} = X_c V_p = U_p \Sigma_p \in \mathbb{R}^{N\times p}.$$

□

LDA formulation. Suppose each sample belongs to one of C distinct classes, indicated by a label vector

$$y = [y_1, \ldots, y_N] \in \{1, \ldots, C\}^N,$$

so that $y_i = c$ if x_i belongs to class $\mathcal{C}_c$. For each class $c = 1, \ldots, C$, define

$$\mathcal{C}_c = \{i \in \{1, \ldots, N\} : y_i = c\}, \qquad N_c = |\mathcal{C}_c|,$$

the set of sample indices in class c and its cardinality. The class mean vectors are

$$\mu_c = \frac{1}{N_c}\sum_{i\in\mathcal{C}_c} x_i \in \mathbb{R}^d, \quad \text{and the overall mean remains } \mu = \frac{1}{N}\sum_{i=1}^{N} x_i.$$

The within-class and between-class scatter matrices are

$$S_W = \sum_{c=1}^{C}\sum_{i\in\mathcal{C}_c}(x_i-\mu_c)(x_i-\mu_c)^T \in \mathbb{R}^{d\times d}, \qquad S_B = \sum_{c=1}^{C} N_c(\mu_c-\mu)(\mu_c-\mu)^T \in \mathbb{R}^{d\times d}.$$

The Fisher criterion seeks a projection matrix $W = [\,w_1, \ldots, w_k\,] \in \mathbb{R}^{d\times k}$ that maximises the ratio of between-class to within-class variance in the projected space

$$J(W) = \frac{\operatorname{tr}(W^T S_B W)}{\operatorname{tr}(W^T S_W W)}.$$

Maximising $J(W)$ simultaneously selects k directions that maximise the between-class variance relative to the within-class variance. This is equivalent to solving the generalised eigenvalue problem

$$S_W^{-1} S_B W = W\Lambda,$$

where Λ is the diagonal matrix of the largest generalised eigenvalues. The columns of W are therefore the eigenvectors of $S_W^{-1} S_B$, defining the discriminant directions that maximise the ratio of between-class to within-class variability.

Computationally. In practice, S_W may be singular, particularly when the number of features d exceeds the number of samples N. To address this, LDA is often preceded by a dimensionality reduction step, typically PCA, or S_W^{-1} is replaced by a regularised inverse. When $d \gg N$, let $U_p \in \mathbb{R}^{d\times p}$ and $V_p \in \mathbb{R}^{N\times p}$ denote the principal directions from the truncated SVD of X_c:

$$X_c \approx U_p \Sigma_p V_p^T.$$

Projecting the centred data onto the principal subspace gives $Z_{\text{PCA}} = U_p^T X_c \in \mathbb{R}^{p\times N}$.

The within-class and between-class scatter matrices in this reduced space are

$$S_{p,W} = \sum_{c=1}^{C} \sum_{i\in\mathcal{C}_c} (x_{p,i} - \mu_{c,p})(x_{p,i} - \mu_{c,p})^T, \qquad S_{p,B} = \sum_{c=1}^{C} N_c(\mu_{c,p} - \mu_p)(\mu_{c,p} - \mu_p)^T,$$

where $x_{p,i} = U_p^T x_i$, $\mu_{c,p} = U_p^T \mu_c$, and $\mu_p = U_p^T \mu$. The LDA directions in the reduced space are the eigenvectors of $S_{p,W}^{-1} S_{p,B}$, denoted by $W_p \in \mathbb{R}^{p\times k}$. The corresponding directions in the original feature space and the LDA scores are, respectively,

$$W_{\text{LDA}} = U_p W_p \in \mathbb{R}^{d\times k} \quad \text{and} \quad Z_{\text{LDA}} = W_{\text{LDA}}^T X_c \in \mathbb{R}^{k\times N}.$$

These columns of W_{LDA} define the discriminant directions in the original space that maximise class separability after PCA-based dimensionality reduction. The LDA scores are the coordinates of each sample along the discriminant directions.

Remark 4.2 (Alternative data orientation)**.** *If the samples are arranged in rows, $X_c \in \mathbb{R}^{N\times d}$, the class sets $\mathcal{C}_c$ correspond to row indices, the class and overall means are computed similarly, and PCA projection is performed as $X_{c,p} = X_c V_p \in \mathbb{R}^{N\times p}$. The LDA directions in the original space and the LDA scores are*

$$W_{\text{LDA}} = V_p W_p \in \mathbb{R}^{d\times k} \quad \textit{and} \quad Z_{\text{LDA}} = X_c W_{\text{LDA}} \in \mathbb{R}^{N\times k},$$

respectively, consistent with the previous derivation. □

This procedure highlights the deep connection between PCA and LDA: PCA reduces dimensionality and ensures that the within-class scatter matrix S_W is invertible, while LDA finds the most discriminative directions within the subspace spanned by the principal components. The combination of SVD-based PCA followed by LDA is widely known as the *Fisherfaces method* in computer vision applications.

4.3 Linear discriminant analysis

Principal Component Analysis (PCA) aims to find the most accurate data representation in a lower dimensional space spanned by the maximum–variance directions. However, such directions might not work well for tasks like classification. Here we present a new data reduction method that tries to preserve the discriminatory information between different classes of the data set.

Linear Discriminant Analysis (LDA) is a classical supervised dimensionality reduction and classification technique. It generalises Fisher's Linear Discriminant Analysis, originally developed for two-class problems, to multi-class datasets while maximising class separability in a reduced subspace.

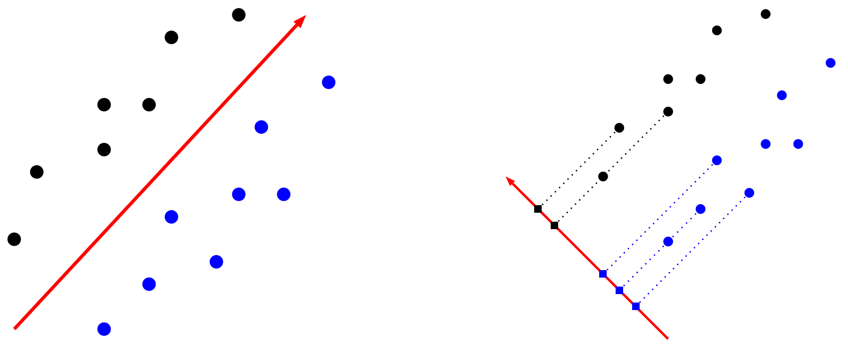


Figure 4.2: Left: PCA: representative but not discriminative; Right: LDA: discriminates between the two classes.

Fisher LDA: two-class case

Let

$$X = [x_1, \dots, x_N] \in \mathbb{R}^{d\times N}$$

be the data matrix, where each column $x_i \in \mathbb{R}^d$ represents a sample belonging to one of two classes, $\mathcal{C}_1$ and $\mathcal{C}_2$, with N_1 and N_2 samples respectively ($N = N_1 + N_2$).

From a statistical perspective, the goal of LDA is to find a projection that maximises the separation of the class means relative to the variability within each class. Denote the class means by

$$\mu_1 = \frac{1}{N_1}\sum_{i\in\mathcal{C}_1} x_i, \qquad \mu_2 = \frac{1}{N_2}\sum_{i\in\mathcal{C}_2} x_i,$$

and the within-class variances by

$$S_1 = \sum_{i\in\mathcal{C}_1}(x_i-\mu_1)(x_i-\mu_1)^T, \qquad S_2 = \sum_{i\in\mathcal{C}_2}(x_i-\mu_2)(x_i-\mu_2)^T.$$

Projecting the data along a direction $w \in \mathbb{R}^d$, the projected class means are

$$m_1 = w^T\mu_1, \quad m_2 = w^T\mu_2,$$

and the variance of each class along w is

$$\sigma_1^2 = w^T S_1 w, \quad \sigma_2^2 = w^T S_2 w.$$

A naive approach is to maximise the squared distance between the projected means,

$$d^2 = (m_1 - m_2)^2 = w^T(\mu_1-\mu_2)(\mu_1-\mu_2)^T w,$$

but this ignores the within-class scatter. Even if the means are far apart, large variances may cause the classes to overlap in the projected space.

To account for both separation and spread, Fisher proposed maximising the ratio of between-class to within-class variance:

$$J(w) = \frac{\text{between-class variance along } w}{\text{within-class variance along } w} = \frac{w^T S_B w}{w^T S_W w},$$

where the **between-class scatter matrix** is

$$S_B = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T,$$

and the total **within-class scatter matrix** is

$$S_W = S_1 + S_2 = \sum_{i \in \mathcal{C}_1} (x_i - \mu_1)(x_i - \mu_1)^T + \sum_{i \in \mathcal{C}_2} (x_i - \mu_2)(x_i - \mu_2)^T.$$

Maximising $J(w)$ identifies a projection direction that simultaneously maximises the distance between class means and minimises the spread of each class, resulting in a more robust and discriminative representation.

Theorem 4.1. *Suppose $S_W \in \mathbb{R}^{d \times d}$ is nonsingular. Then the vector w that maximises the Fisher criterion*

$$J(w) = \frac{w^T S_B w}{w^T S_W w}, \quad \textit{subject to } \|w\| = 1,$$

is the eigenvector of $S_W^{-1} S_B$ corresponding to the largest eigenvalue.

Proof. Since S_W is symmetric positive definite, define the change of variable

$$v = S_W^{1/2} w \quad \Rightarrow \quad w = S_W^{-1/2} v.$$

Then the Fisher criterion becomes

$$J(w) = \frac{w^T S_B w}{w^T S_W w} = \frac{(S_W^{-1/2} v)^T S_B (S_W^{-1/2} v)}{v^T v} = \frac{v^T (S_W^{-1/2} S_B S_W^{-1/2}) v}{v^T v}.$$

The numerator is a quadratic form and the denominator is the squared norm of v, so this is a Rayleigh quotient. It is maximised when v is the eigenvector corresponding to the largest eigenvalue of $S_W^{-1/2} S_B S_W^{-1/2}$. Transforming back to w gives $w = S_W^{-1/2} v$, and so w is an eigenvector of $S_W^{-1} S_B$ corresponding to the largest eigenvalue. □

This result shows that the optimal LDA direction ω^* aligns with the inverse within-class scatter applied to the difference of class means, which is exactly the eigenvector of $S_W^{-1} S_B$ associated with the nonzero eigenvalue.

Remark 4.3 (Computational aspects)**.** *The within-class scatter matrix*

$$S_W = \sum_{c=1}^{2} \sum_{i \in \mathcal{C}_c} (x_i - \mu_c)(x_i - \mu_c)^T$$

is always symmetric and positive semidefinite. It is positive definite if the within-class deviations span $\mathbb{R}^d$*, ensuring* $w^T S_W w > 0$ *for all nonzero* w*.*

In the two-class case, the between-class scatter matrix

$$S_B = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T$$

has rank one. Consequently, $S_W^{-1} S_B$ *also has rank one and only one nonzero eigenvalue. Indeed, for any vector* w*,*

$$S_W^{-1} S_B w = S_W^{-1}(\mu_1 - \mu_2) \underbrace{(\mu_1 - \mu_2)^T w}_{scalar},$$

showing that any nonzero vector in the direction of $S_W^{-1}(\mu_1 - \mu_2)$ *is an eigenvector. Therefore, the LDA direction* ω^* *that maximises* $J(w)$ *can be computed directly by solving the linear system*

$$S_W w = \mu_1 - \mu_2$$

and normalising w *to unit length.*

Alternatively, when S_W *is singular, one may use the pseudo-inverse*

$$w^* = S_W^+(\mu_1 - \mu_2),$$

to obtain the same direction. □

Algorithm: Fisher LDA

The procedure can be described in the following steps:

1. **Compute class statistics:** Calculate the mean vectors μ_1 and μ_2, and the within-class scatter matrices S_1, S_2.

2. **Construct total within-class scatter:** $S_W = S_1 + S_2$.

3. **Compute the Fisher direction:** Solve $w^* = S_W^{-1}(\mu_1 - \mu_2)$. This direction maximises the separation between the two classes relative to their internal variance.

4. **Project the data:** For each sample, compute the 1D projection $z_i = w^{*T} x_i$.

5. **Classification (optional):** Determine a threshold $t = w^{*T}(\mu_1 + \mu_2)/2$ to classify new samples based on their projection.

Dimensionality Reduction Perspective

Fisher LDA reduces the original d-dimensional space to a 1D subspace. Along this subspace, the projected data exhibits **maximal class separability**, since the projection aligns with the direction that maximizes the between-class variance relative to the within-class variance. This makes Fisher LDA both a **dimensionality reduction** and a **classification** tool. Mathematically, the 1D projection is the eigenvector corresponding to the largest eigenvalue of $S_W^{-1} S_B$, which in the two-class case is unique and guarantees the optimal discriminant direction.

Listing 4.1: MATLAB example: two classes

```
% Generate synthetic 2-class data
rng(1); d=3; N1=50; N2=50;
X1 = randn(d,N1)+1; X2 = randn(d,N2)-1;
X = [X1 X2]; y=[ones(1,N1) 2*ones(1,N2)];

% Compute class means
mu1 = mean(X1,2); mu2 = mean(X2,2);

% Compute within-class scatter
SW = (X1-mu1)*(X1-mu1)' + (X2-mu2)*(X2-mu2)';

% Compute Fisher direction
w = SW \ (mu1-mu2);

% Project data
Z = w' * X;

% Plot
figure; hold on;
plot(Z(1:N1), zeros(1,N1),'ro','MarkerFaceColor','r');
plot(Z(N1+1:end), zeros(1,N2),'bo','MarkerFaceColor','b');
xlabel('Projected data'); ylabel('Zero axis'); grid on;
legend('Class 1','Class 2'); title('Fisher LDA: 2-class projection
    ');
```

In the binary case $C = 2$, Fisher LDA finds the projection direction $w = S_W^{-1}(\mu_1 - \mu_2)$, and the data is mapped to one dimension as $Z = w^\top X$. The resulting projection is shown in Figure 4.3, where samples from the two classes are clearly separated along a single axis. Since $C = 2$, the dimensional reduction is $1 = C - 1$.

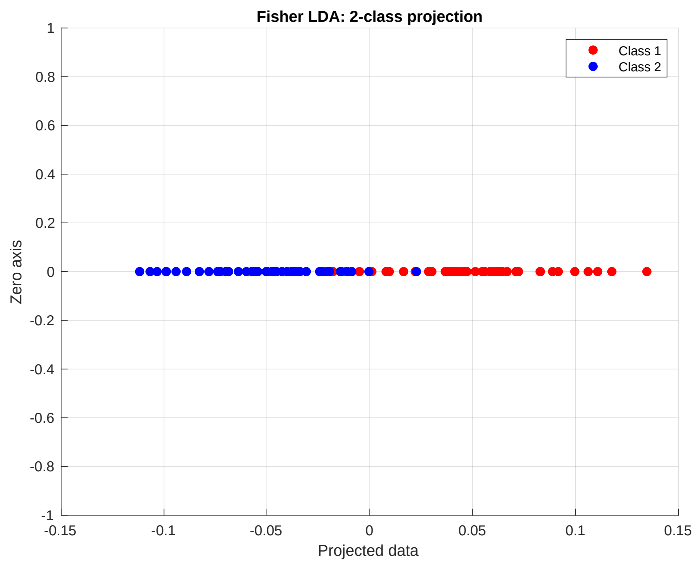


Figure 4.3: Projection of 2-class data using Fisher LDA.

Fisher LDA: an example

We illustrate Linear Discriminant Analysis (LDA) for a simple two-class example, where the data is represented in $\mathbb{R}^{d\times N}$, i.e., each column is a data point.

Data. Consider two classes with $d = 2$ features and $N_1 = N_2 = 5$ points each:

$$X_1 = \begin{bmatrix} 2 & 4 & 2 & 3 & 5 \\ 3 & 3 & 1 & 4 & 4 \end{bmatrix}, \quad X_2 = \begin{bmatrix} 7 & 6 & 7 & 8 & 10 \\ 4 & 8 & 6 & 9 & 9 \end{bmatrix}.$$

The combined data matrix is

$$X = [X_1, X_2] \in \mathbb{R}^{2\times 10}.$$

Class means. The class mean vectors are computed as

$$\mu_1 = \frac{1}{N_1}\sum_{i=1}^{N_1} X_1(:, i), \quad \mu_2 = \frac{1}{N_2}\sum_{i=1}^{N_2} X_2(:, i),$$

where $\mu_1, \mu_2 \in \mathbb{R}^2$.

Scatter matrices. The within-class scatter matrices are

$$S_1 = \sum_{i=1}^{N_1}(X_1(:, i) - \mu_1)(X_1(:, i) - \mu_1)^T, \quad S_2 = \sum_{i=1}^{N_2}(X_2(:, i) - \mu_2)(X_2(:, i) - \mu_2)^T,$$

and the total within-class scatter is

$$S_W = S_1 + S_2.$$

The between-class scatter matrix is

$$S_B = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T.$$

LDA direction. The LDA directions are obtained by solving the generalised eigenvalue problem:

$$S_W^{-1} S_B W = W\Lambda,$$

and selecting the eigenvectors corresponding to the largest eigenvalues. Denote the principal Fisher direction as w_1 and the secondary direction as w_2.

Projection onto w_1. The 1D projection of all points onto the principal direction is

$$Z_1 = w_1^T X \in \mathbb{R}^{1\times 10}.$$

with 2D coordinates on the line given by

$$X_{\text{proj},1} = w_1 w_1^T X = w_1 Z_1 \in \mathbb{R}^{2\times 10}.$$

Listing 4.2: MATLAB Implementation

```
% Data as d x N
X1 = [2, 4, 2, 3, 5;
      3, 3, 1, 4, 4];
X2 = [7, 6, 7, 8, 10;
      4, 8, 6, 9, 9];
X = [X1, X2];

% Class means
Mu1 = mean(X1,2);
Mu2 = mean(X2,2);

% Within-class scatter
S1 = (X1-Mu1)*(X1-Mu1)';
S2 = (X2-Mu2)*(X2-Mu2)';
Sw = S1 + S2;

% Between-class scatter
Sb = (Mu1-Mu2)*(Mu1-Mu2)';

% Solve generalized eigenvalue problem
[W,D] = eig(inv(Sw)*Sb);
[~,idx] = sort(diag(D),'descend');
w1 = W(:,idx(1));
w2 = W(:,idx(2));

% Projection onto V1
Z1 = w1'*X;
Xp1 = w1*Z1;

% Positive direction V1 line
scale = 15;
w1_line = [zeros(2,1), scale*w1];

% Plot
figure; hold on; grid on;
plot(X1(1,:),X1(2,:),'ro','MarkerFaceColor','r');
plot(X2(1,:),X2(2,:),'bo','MarkerFaceColor','b');
plot(w1_line(1,:),w1_line(2,:),'k-','LineWidth',1,'
   HandleVisibility','off');
origin = [0;0];
quiver(origin(1),origin(2),3*w1(1),3*w1(2),0,'k','LineWidth',2);
quiver(origin(1),origin(2),3*w2(1),3*w2(2),0,'m','LineWidth',2);
for i=1:size(X,2)
    plot([X(1,i),Xp1(1,i)],[X(2,i),Xp1(2,i)],'k--');
end
plot(Xp1(1,1:5),Xp1(2,1:5),'ro','MarkerFaceColor','r','MarkerSize'
   ,8);
plot(Xp1(1,6:end),Xp1(2,6:end),'bo','MarkerFaceColor','b','
   MarkerSize',8);
xlabel('X1'); ylabel('X2'); axis equal;
```

```
title('Two-Class LDA: d x N example');
legend({'Class 1','Class 2','w1','w2'}, 'Location','best');
```

Remark 4.4 (Alternative data orientation)**.** *If the data is represented as*

$$X \in \mathbb{R}^{N\times d},$$

with each row as a data point, the class means and within-class scatter matrices are computed row-wise:

$$\mu_1 = \frac{1}{N_1}\sum_{i=1}^{N_1} X_1(i,:)^T, \quad \mu_2 = \frac{1}{N_2}\sum_{i=1}^{N_2} X_2(i,:)^T,$$

$$S_1 = \sum_{i=1}^{N_1}(X_1(i,:)^T - \mu_1)(X_1(i,:)^T - \mu_1)^T, \quad S_2 = \sum_{i=1}^{N_2}(X_2(i,:)^T - \mu_2)(X_2(i,:)^T - \mu_2)^T.$$

The LDA procedure, including solving

$$S_W^{-1} S_B W = W\Lambda$$

and projecting onto the principal Fisher direction is $Z_1 = Xw_1$ *since the data orientation differs: points are rows rather than columns.* □

Figure 4.4 illustrates the results of the two-class LDA example. The main observations are summarised below:

- **Original points:** The red and blue markers correspond to the two classes. The distribution of the points shows some overlap, but they are generally separable along a principal direction.

- **Principal Fisher direction (w_1):** The black arrow at the origin indicates the principal LDA vector w_1, along which the class separation is maximised. The thin black line extending from the origin shows the positive extension of this direction, acting as the LDA axis for projecting points.

- **Secondary LDA direction (w_2):** The magenta arrow at the origin represents the secondary LDA vector w_2, orthogonal to w_1 in this two-dimensional case. This direction captures the residual variance not aligned with class separation.

- **Projections onto w_1:** Each data point is projected orthogonally onto the w_1 axis (shown by dashed lines). The projected points retain their class label and are visualised as larger markers on the LDA axis. This projection effectively reduces the two-dimensional classification problem to a one-dimensional representation while maximising class separability.

- **Class separation:** The projections demonstrate that the two classes are well separated along V_1. This confirms that the principal Fisher direction captures the main discriminatory information in the dataset.

- **Invariance to orientation:** As discussed in the previous remark, the LDA results are invariant under rigid rotations of the data. The vectors w_1 and w_2 rotate correspondingly if the data is rotated, but the projections along w_1 remain consistent in terms of class separation.

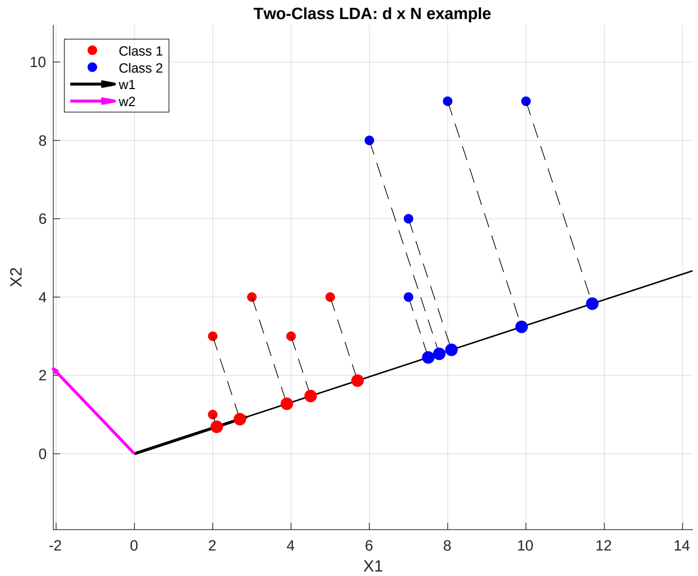


Figure 4.4: Two-Class LDA: Original points, projections onto w_1, and LDA vectors. Red and blue markers represent the two classes.

Multi-class LDA

For $C > 2$ classes, Linear Discriminant Analysis generalises Fisher's two-class LDA to find a set of linear projection directions that maximise class separability while reducing the data dimensionality. Let

$$X = [x_1, \ldots, x_N] \in \mathbb{R}^{d \times N}$$

be the data matrix, with each column $x_i \in \mathbb{R}^d$ a sample labeled $y_i \in \{1, \ldots, C\}$. Denote by $\mathcal{C}_c = \{i : y_i = c\}$ the indices of samples in class c, and $N_c = |\mathcal{C}_c|$.

The class mean vectors are

$$\mu_c = \frac{1}{N_c} \sum_{i \in \mathcal{C}_c} x_i, \quad c = 1, \ldots, C,$$

and the overall mean vector is

$$\mu = \frac{1}{N} \sum_{i=1}^{N} x_i.$$

The within-class scatter matrix

$$S_W = \sum_{c=1}^{C} \sum_{i \in \mathcal{C}_c} (x_i - \mu_c)(x_i - \mu_c)^T$$

is symmetric and positive semidefinite, and positive semidefinite (positive definite if the within-class deviations span $\mathbb{R}^d$). The between-class scatter matrix

$$S_B = \sum_{c=1}^{C} N_c (\mu_c - \mu)(\mu_c - \mu)^T$$

weights each class by its size, ensuring that the total scatter

$$S_T = \sum_{i=1}^{N} (x_i - \mu)(x_i - \mu)^T$$

decomposes exactly as $S_T = S_W + S_B$, so S_B captures the variance due to class mean differences. Each term in S_B is rank one, so $\operatorname{rank}(S_B) \le C - 1$, meaning there are at most $C - 1$ informative discriminant directions.

Let

$$W = [w_1, \ldots, w_k] \in \mathbb{R}^{d\times k}$$

be a matrix of projection directions, with $k \le C - 1$. The projected data is

$$Z = W^T X \in \mathbb{R}^{k\times N},$$

and the within- and between-class scatter matrices in the projected space are

$$S_W^Z = W^T S_W W, \qquad S_B^Z = W^T S_B W.$$

The multi-class LDA criterion generalises the Fisher ratio:

$$J(W) = \frac{\operatorname{tr}(S_B^Z)}{\operatorname{tr}(S_W^Z)} = \frac{\operatorname{tr}(W^T S_B W)}{\operatorname{tr}(W^T S_W W)},$$

which is maximised by selecting directions that maximise between-class variance relative to within-class variance. The eigenvectors corresponding to the $C-1$ largest eigenvalues form the columns of W^*, defining the subspace of maximal class separability.

If S_W is singular or $d \gg N$, one can first apply PCA to reduce dimensionality. Let $U_p \in \mathbb{R}^{d\times p}$ contain the first p principal directions of X_c. In the reduced space, solving

$$S_{p,B} w_p = \lambda S_{p,W} w_p$$

yields the LDA directions W_p, with the original-space directions given by $W_{\text{LDA}} = U_p W_p$.

Remark 4.5. *Each discriminant direction can also be computed by solving the linear system*

$$S_W w = (\mu_c - \mu), \quad c = 1, \ldots, C-1,$$

in the subspace spanned by the class-mean differences, followed by normalisation to unit length. The generalised eigenvectors of $S_W^{-1} S_B$ maximise between-class variance relative to within-class variance, and only the $C-1$ directions corresponding to nonzero eigenvalues are informative for class separation. □

Algorithm: multi-class LDA

The procedure for multi-class LDA can be described step by step, highlighting both the mathematical and computational reasoning:

1. **Compute class and overall means:** For each class $c = 1, \ldots, C$, calculate μ_c and the global mean μ. These means provide the reference points for computing scatter matrices.

2. **Compute scatter matrices:** Evaluate the within-class scatter S_W and the between-class scatter S_B. S_W captures the variance within classes, while S_B captures variance between class means.

3. **Solve the generalised eigenvalue problem:** Find eigenvectors w satisfying $S_B w = \lambda S_W w$. Each eigenvector corresponds to a discriminant direction.

4. **Select top eigenvectors:** Choose the $k \leq C - 1$ eigenvectors corresponding to the largest eigenvalues to form the projection matrix W. These directions preserve maximal class separability.

5. **Project data:** The reduced-dimensional representation is

$$Z = W^T X \in \mathbb{R}^{k \times N}.$$

Each column of Z represents a sample in the $(C - 1)$-dimensional discriminant space.

Dimensionality reduction perspective

Multi-class LDA projects the original d-dimensional data onto a $(C - 1)$-dimensional subspace that maximises between-class separation relative to within-class variance. The resulting representation is suitable for visualisation, classification, and further analysis. Mathematically, this can be interpreted as selecting the leading eigenvectors of $S_W^{-1} S_B$, which form an optimal basis for discrimination among C classes.

Listing 4.3: MATLAB example: three classes

```
% Generate synthetic 3-class data
rng(1); d=4; N3=50; N2=N3; N1=N2;
X1=randn(d,N1)+1; X2=randn(d,N2)-1; X3=randn(d,N3);
X=[X1 X2 X3]; y=[ones(1,N1) 2*ones(1,N2) 3*ones(1,N3)];

% Compute class means and overall mean
mu1=mean(X1,2); mu2=mean(X2,2); mu3=mean(X3,2); mu=mean(X,2);

% Compute scatter matrices
SW = (X1-mu1)*(X1-mu1)' + (X2-mu2)*(X2-mu2)' + (X3-mu3)*(X3-mu3)';
SB = N1*(mu1-mu)*(mu1-mu)' + N2*(mu2-mu)*(mu2-mu)' + N3*(mu3-mu)*(
   mu3-mu)';

% Solve generalized eigenvalue problem
[V,D] = eig(SB,SW);
[~,idx] = sort(diag(D),'descend');
W = V(:,idx(1:2)); % For C=3, k=C-1=2

% Project data
Z = W' * X;

% Plot 2D projection
figure; hold on;
plot(Z(1,1:N1),Z(2,1:N1),'ro','MarkerFaceColor','r');
plot(Z(1,N1+1:N1+N2),Z(2,N1+1:N1+N2),'bo','MarkerFaceColor','b');
```

```
plot(Z(1,N1+N2+1:end),Z(2,N1+N2+1:end),'go','MarkerFaceColor','g')
    ;
xlabel('LD 1'); ylabel('LD 2'); grid on;
legend('Class 1','Class 2','Class 3'); title('Multi-class LDA: 3-
    class projection')
```

For $C = 3$, LDA yields a projection matrix $W \in \mathbb{R}^{d\times 2}$ since the discriminant subspace has dimension at most $C-1 = 2$. After computing the eigenvectors of $S_B v = \lambda S_W v$, we form $W = [v_1\ v_2]$ and project the data as $Z = W^\top X$. Figure 4.5 shows the resulting 2D representation, where the three classes become linearly separable in the reduced space.

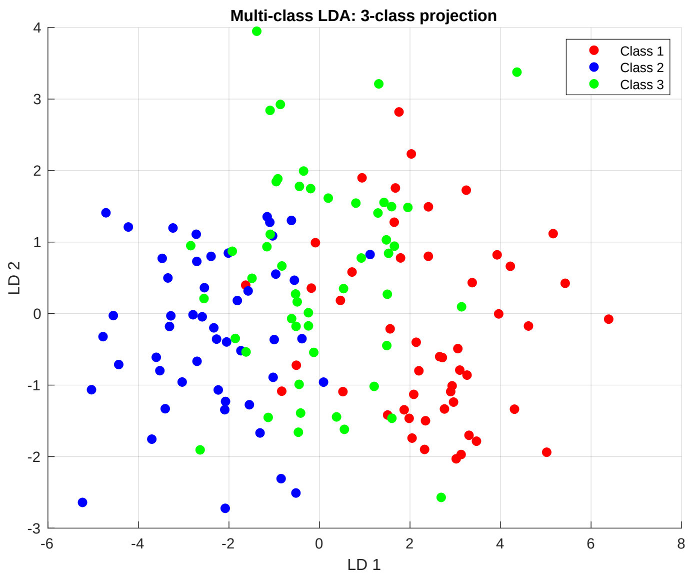


Figure 4.5: 2D LDA projection for a 3-class dataset.

This demonstrates that multi-class LDA projects d-dimensional data to a $(C-1)$-dimensional subspace, while maximising class separability. For $C = 3$, the projection is 2-dimensional, suitable for visualisation and classification.

Three-class LDA: an example

We illustrate LDA for three classes in three-dimensional space ($d = 3$, $C = 3$). The data matrices for the three classes are $X_1, X_2, X_3 \in \mathbb{R}^{3\times 5}$, with each column representing a point in $\mathbb{R}^3$. The combined data matrix is

$$X = [X_1, X_2, X_3] \in \mathbb{R}^{3\times 15}.$$

Class means and scatter matrices. The class mean vectors are computed as

$$\mu_i = \frac{1}{N_i}\sum_{j=1}^{N_i} X_i(:,j), \quad i = 1,2,3.$$

The within-class scatter matrices are

$$S_i = \sum_{j=1}^{N_i}(X_i(:,j)-\mu_i)(X_i(:,j)-\mu_i)^T, \quad i = 1,2,3,$$

with total within-class scatter $S_W = S_1+S_2+S_3$, and the between-class scatter matrix is

$$S_B = \sum_{i=1}^{3} N_i(\mu_i - \mu)(\mu_i - \mu)^T, \quad \mu = \text{mean}(X, 2).$$

LDA directions. The LDA directions are obtained by solving the generalised eigenvalue problem

$$S_W^{-1} S_B W = W\Lambda,$$

and selecting the eigenvectors corresponding to the largest eigenvalues. In 3D, we can use the first two eigenvectors w_1 and w_2 to visualise the main directions of class separation.

Projection onto LDA directions. Each data point can be projected onto the first two LDA directions:

$$Z = [w_1, w_2]^T X \in \mathbb{R}^{2\times 15}.$$

The 2D coordinates Z provide a reduced representation that preserves maximal class separation.

Listing 4.4: MATLAB implementation: three classes

```
% Three-class LDA in 3D

% Data (d x N)
X1 = [2,4,2,3,5; 3,3,1,4,4; 1,2,1,3,2];
X2 = [7,6,7,8,10; 4,8,6,9,9; 3,2,4,5,6];
X3 = [1,2,1,2,3; 7,8,6,7,9; 4,5,3,4,5];
X  = [X1, X2, X3];

% Class sizes
n1 = size(X1,2); n2 = size(X2,2); n3 = size(X3,2);

% Class means
Mu1 = mean(X1,2); Mu2 = mean(X2,2); Mu3 = mean(X3,2);
Mu  = mean(X,2);

% Within- and between-class scatter
S1 = (X1-Mu1)*(X1-Mu1)';
S2 = (X2-Mu2)*(X2-Mu2)';
S3 = (X3-Mu3)*(X3-Mu3)';
Sw = S1 + S2 + S3;
Sb = n1*(Mu1-Mu)*(Mu1-Mu)' + n2*(Mu2-Mu)*(Mu2-Mu)' + n3*(Mu3-Mu)*(
    Mu3-Mu)';

% Solve generalized eigenvalue problem
[W,D] = eig(inv(Sw)*Sb);
[~, idx] = sort(diag(D), 'descend');
w1 = W(:,idx(1)); w2 = W(:,idx(2));

% Projection onto first two LDA directions
```

```
Z = W(:,idx(1:2))' * X;

% 3D plot of original points and LDA vectors
figure; hold on; grid on; view(3);
plot3(X1(1,:),X1(2,:),X1(3,:),'ro','MarkerFaceColor','r');
plot3(X2(1,:),X2(2,:),X2(3,:),'bo','MarkerFaceColor','b');
plot3(X3(1,:),X3(2,:),X3(3,:),'go','MarkerFaceColor','g');
origin = Mu;
quiver3(origin(1),origin(2),origin(3),3*w1(1),3*w1(2),3*w1(3),0,'k','LineWidth',2);
quiver3(origin(1),origin(2),origin(3),3*w2(1),3*w2(2),3*w2(3),0,'m','LineWidth',2);
xlabel('X1'); ylabel('X2'); zlabel('X3');
title('Three-Class LDA: 3D data with LDA vectors');
legend({'Class 1','Class 2','Class 3','w1','w2'},'Location','best');

% 2D projection onto first two LDA directions
figure; hold on; grid on;
plot(Z(1,1:n1), Z(2,1:n1),'ro','MarkerFaceColor','r');
plot(Z(1,n1+1:n1+n2), Z(2,n1+1:n1+n2),'bo','MarkerFaceColor','b');
plot(Z(1,n1+n2+1:end), Z(2,n1+n2+1:end),'go','MarkerFaceColor','g');
xlabel('LDA direction 1'); ylabel('LDA direction 2');
title('Projection of three-class 3D data onto first two LDA directions');
legend({'Class 1','Class 2','Class 3'},'Location','best');
axis equal;
```

Figure 4.6 shows the original 3D points of the three classes, together with the LDA vectors w_1 and w_2 originating from the overall mean. The main observations are summarised below:

- **Original points:** Red, blue, and green markers correspond to Classes 1, 2, and 3, respectively. The points are distributed in 3D space with partial overlap.
- **LDA vectors (w_1 and w_2):** The black arrow $w2_1$ indicates the principal LDA direction that maximises class separation, while the magenta arrow w_2 captures the secondary direction orthogonal to w_1. Both vectors originate from the overall mean of all points.
- **3D class separation:** Although the three classes occupy overlapping regions in 3D, the directions w_1 and w_2 highlight the axes along which separation is maximised.
- **Projections onto LDA directions:** Figure 4.7 shows the projection of all points onto the first two LDA directions. The 2D representation reduces dimensionality while preserving discriminative information.
- **Class separation:** The separation between classes is evident: Class 1 (red) and Class 2 (blue) are well-separated along w_1, while Class 3 (green) overlaps partially but is more clearly distinguished along w_2.

- **Invariance to rotation:** As in the 2D case, LDA results are invariant under rigid rotations of the dataset. Rotating the 3D points rotates w_1 and w_2 accordingly, but the projected separability remains consistent.

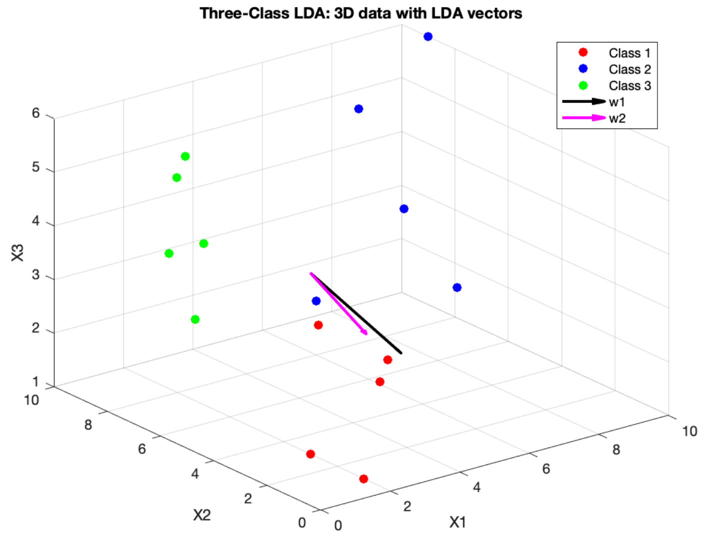


Figure 4.6: Original 3D points with LDA vectors w_1 (black) and w_2 (magenta) originating from the overall mean. Red, blue, and green markers correspond to the three classes.

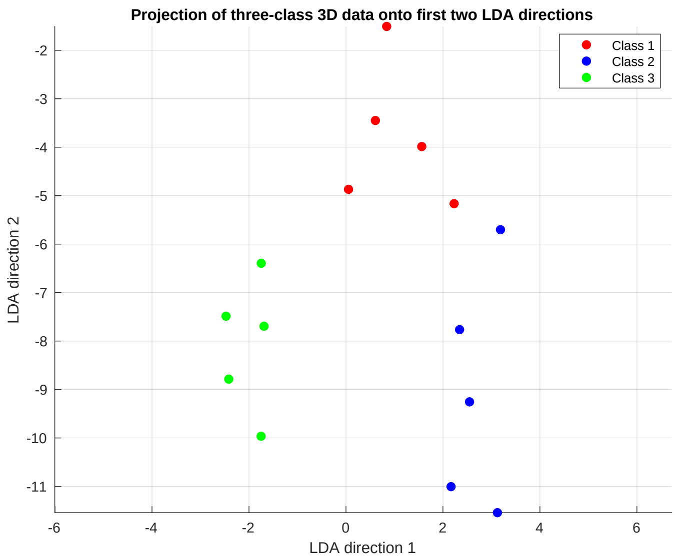


Figure 4.7: Projection of the three-class 3D data onto the first two LDA directions. The 2D coordinates $Z = [V_1, V_2]^T X$ clearly separate the classes.

Fisherfaces

Fisherfaces generalises Fisher LDA to multiple classes for face recognition, providing a supervised dimensionality reduction that maximises class separability while reducing the high-dimensional image space. Without loss of generality, we consider three subjects (s_1, s_2, s_3) from the ATT face database; the procedure naturally applies to two subjects by restricting $C = 2$.

Data setup. Let

$$X = [x_1, \ldots, x_N] \in \mathbb{R}^{d\times N}$$

be the data matrix, where d is the number of pixels per image and $N = N_1 + N_2 + N_3$ is the total number of images. Each column x_i represents a vectorised image. The class labels are defined as

$$y = [\underbrace{1, \ldots, 1}_{N_1}, \underbrace{2, \ldots, 2}_{N_2}, \underbrace{3, \ldots, 3}_{N_3}] \in \{1, 2, 3\}^N,$$

so that x_i belongs to class $\mathcal{C}_c$ if $y_i = c$.

The mean image is computed across all samples:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i,$$

and the centred data matrix is

$$X_c = X - \mu\mathbf{1}_N^T.$$

PCA: eigenfaces. Since the dimensionality d of each image is usually much larger than the number of samples N, direct LDA may lead to singular within-class scatter matrices. To address this, PCA is performed first

$$X_c = \hat{U}\hat{\Sigma}\hat{V}^T \approx U_p\Sigma_p V_p^T,$$

where $U_p \in \mathbb{R}^{d\times r}$ contains the top r eigenvectors (eigenfaces) capturing the principal variations in the dataset. The PCA-projected data is

$$Z_{\text{PCA}} = U_p^T X_c \in \mathbb{R}^{r\times N},$$

which reduces the dimensionality and retains the directions of maximum variance.

LDA: Fisherfaces. Within the PCA subspace, for each class $c = 1, 2, 3$, compute the class mean:

$$\tilde{\mu}_c = \frac{1}{N_c}\sum_{i\in\mathcal{C}_c} X_{\text{PCA},i},$$

and the overall mean

$$\tilde{\mu} = \frac{1}{N}\sum_{i=1}^{N} X_{\text{PCA},i}.$$

The within-class scatter matrix in PCA space is

$$S_W = \sum_{c=1}^{3}\sum_{i\in\mathcal{C}_c}(X_{\text{PCA},i} - \tilde{\mu}_c)(X_{\text{PCA},i} - \tilde{\mu}_c)^T,$$

and the between-class scatter matrix is

$$S_B = \sum_{c=1}^{3} N_c(\tilde{\mu}_c - \tilde{\mu})(\tilde{\mu}_c - \tilde{\mu})^T.$$

The Fisherfaces are obtained by solving the generalised eigenvalue problem

$$S_B w = \lambda S_W w$$

and selecting the top $k = \min(C-1, r) = 2$ eigenvectors corresponding to the largest eigenvalues. These eigenvectors are then mapped back to the original image space:

$$W_{\text{LDA}} = U_p\,[w_1\; w_2].$$

Finally, all images are projected onto the 2D Fisherface subspace:

$$Z_{\text{LDA}} = W_{\text{LDA}}^T X \in \mathbb{R}^{2\times N},$$

allowing visualisation of all three subjects in a reduced space. In this representation, each class forms a distinct cluster, demonstrating that Fisherfaces achieve dimensionality reduction from d pixels to $C-1=2$ dimensions while preserving class separability.

Algorithm: Fisherfaces

The Fisherfaces algorithm for multiple subjects can be summarised as follows. Each step is accompanied by a short explanation.

1. Load the images for all subjects, for example s_1, s_2, s_3, and assign class labels y accordingly. This prepares the dataset for processing.

2. Compute the mean image μ over all samples and centre the data by subtracting it: $X_c = X - \mu\mathbf{1}_N^T$. Centring ensures that PCA captures the directions of maximum variance rather than being biased by the mean.

3. Apply PCA to reduce dimensionality and compute $X_c \approx U_p\Sigma_p V_p^T$. Project the images into the PCA subspace: $Z_{\text{PCA}} = U_p^T X_c$. Dimensionality reduction helps avoid singularity in the within-class scatter matrix.

4. In the PCA subspace, compute the mean of each class $\tilde{\mu}_c$ and the overall mean $\tilde{\mu}$. These are used to construct the scatter matrices.

5. Compute the within-class scatter S_W and the between-class scatter S_B in the PCA space. These matrices quantify the variation inside each class and between classes, respectively.

6. Solve the generalised eigenvalue problem $S_B w = \lambda S_W w$ and select the top $C-1$ eigenvectors corresponding to the largest eigenvalues. This step identifies the directions that best separate the classes.

7. Map the selected eigenvectors back to the original image space to obtain the Fisherfaces: $W_{\text{LDA}} = U_p[w_1\; w_2]$. Fisherfaces highlight the discriminative features in the original high-dimensional space.

8. Project all images onto the Fisherface subspace: $Z_{\text{LDA}} = W_{\text{LDA}}^T X$. This produces a low-dimensional representation suitable for classification or visualisation.

9. Optionally, display the mean face, the top eigenfaces, the Fisherfaces, and the projected 2D data. Visual inspection can help understand the effectiveness of the dimensionality reduction and class separation.

Listing 4.5: MATLAB code: three-class Fisherfaces

```
% Settings
img_size = [112,92]; data_folder='ATT'; subjects={'s1','s2','s3'};
N = 10; d = prod(img_size); C = numel(subjects);
X = zeros(d,C*N); y = zeros(1,C*N);

% Load images
for c=1:C
    img_files = dir(fullfile(data_folder,subjects{c},'*.pgm'));
    for i=1:N
        img = double(imread(fullfile(img_files(i).folder,img_files
            (i).name)));
        X(:,(c-1)*N+i) = img(:);
        y((c-1)*N+i) = c;
    end
end

% Mean and PCA
mu = mean(X,2); Xc = X - mu;
[U,~,~] = svd(Xc,'econ'); Up = U(:,1:min(C*N-C,size(U,2)));
Z_pca = Up' * Xc;

% Compute class means
tilde_mu = mean(Z_pca,2); SW = zeros(size(Z_pca,1)); SB = zeros(
   size(Z_pca,1));
tilde_mu_c = zeros(size(Z_pca,1),C);
for c=1:C
    idx = find(y==c);
    tilde_mu_c(:,c) = mean(Z_pca(:,idx),2);
    Xc_c = Z_pca(:,idx)-tilde_mu_c(:,c);
    SW = SW + Xc_c*Xc_c';
    diff = tilde_mu_c(:,c)-tilde_mu; SB = SB + length(idx)*(diff*
       diff');
end

% Fisherfaces
[V,D] = eig(SB,SW); [~,idx_sort] = sort(diag(D),'descend'); k=C-1;
W_lda = Up*V(:,idx_sort(1:k)); Z_lda = W_lda'*X;

% Display 2x2 figure
figure('Position',[100,100,800,800]);
subplot(2,2,1); imagesc(reshape(mat2gray(mu),img_size)); colormap
   gray; axis image off; title('Mean Face');
```

```
subplot(2,2,2); imagesc(reshape(mat2gray(Up(:,1)),img_size));
   colormap gray; axis image off; title('Eigenface 1');
subplot(2,2,3); imagesc(reshape(mat2gray(Up(:,2)),img_size));
   colormap gray; axis image off; title('Eigenface 2');
subplot(2,2,4); imagesc(reshape(mat2gray(W_lda(:,1)),img_size));
   colormap gray; axis image off; title('Fisherface 1');
print('Mean_Eigen_Fisherfaces_3subject.pdf','-dpdf','-bestfit');

% 2D Fisherfaces projection
figure; hold on; colors='rgb'; markers='o+s';
for c=1:C
    idx=find(y==c);
    plot(Z_lda(1,idx),Z_lda(2,idx),markers(c),'MarkerFaceColor',
       colors(c),'MarkerEdgeColor','k');
end
xlabel('Fisher LD1'); ylabel('Fisher LD2'); grid on; title('3-
   Class Fisherfaces Projection');
legend(subjects,'Location','best');
print('Fisherfaces_3class_Projection.pdf','-dpdf','-bestfit');
```

Figure 4.8 displays the mean face, the first two eigenfaces, and the first Fisherface for the three subjects. The mean face represents the average appearance across all images, capturing the common facial structure. The first two eigenfaces show the primary modes of variation in the dataset, reflecting features that vary the most across all subjects. The first Fisherface, in contrast, emphasises the features that are most relevant for distinguishing between the three classes, highlighting discriminative characteristics that separate the subjects.

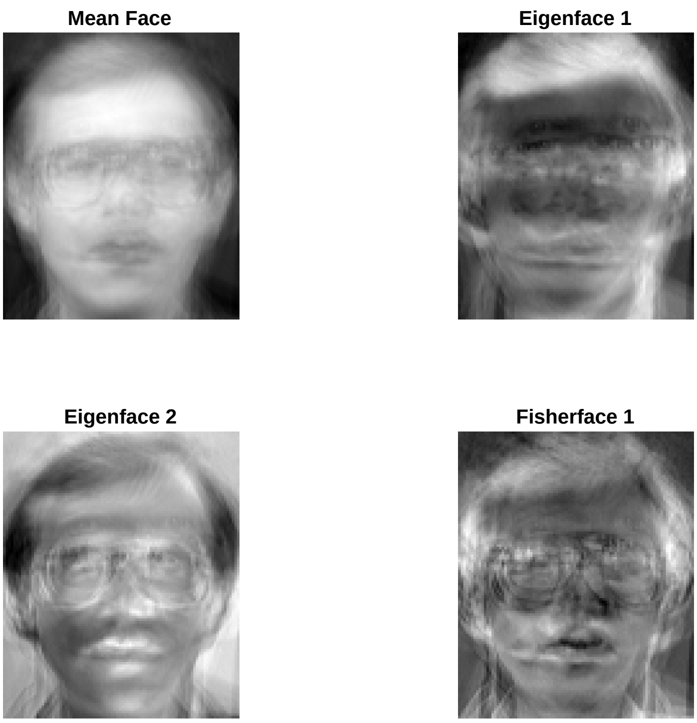


Figure 4.8: Mean face, first two eigenfaces, and first Fisherface for three subjects.

4.4 Conclusion

In summary, while Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) both serve as powerful linear techniques for reducing the dimensionality of complex datasets, their methodologies diverge fundamentally based on the availability of class labels and the ultimate analytical objective.

PCA operates as an unsupervised method, agnostic to group membership, with the singular goal of maximising the total variance within the data. By computing the eigenvectors of the covariance matrix, PCA identifies the orthogonal axes that capture the most information about the dataset's global geometric structure. This makes it the ideal instrument for noise reduction, data compression, and exploratory visualisation when the data is unlabelled or when the intrinsic variability of the signal is of primary interest.

Conversely, LDA is a supervised technique that integrates class labels to guide the projection process. Rather than seeking global spread, LDA optimises the Fisher criterion, mathematically maximising the ratio of between-class variance to within-class variance. This approach ensures that the resulting subspace pushes class means apart while keeping data points within classes tightly clustered. Consequently, LDA is the superior choice for preprocessing data in classification tasks, as it preserves the discriminatory information necessary to distinguish between groups – information that PCA might inadvertently discard if the discriminatory features possess low variance. Ultimately, the choice rests on the task at hand: use PCA to reveal the general structure of the data, and LDA to define the boundaries between its classes.

4.5 Exercises

Hand-solved exercises

1. For a 3-class 2D dataset, compute S_W and S_B, then find the top 2 LDA components. Project and visualise the data.

2. Consider two classes in 2D with samples:

$$\mathcal{C}_1 = \{(2,3),(3,3),(2,4)\}, \quad \mathcal{C}_2 = \{(6,5),(7,6),(5,6)\}.$$

   Compute the class means, within-class scatter S_W, and between-class scatter S_B. Find the Fisher LDA direction w^*.

3. Show that for a 3-class dataset, the maximum number of LDA components is $C-1 = 2$. Use a small 3x2 dataset as example.

4. Explain why the number of LDA components cannot exceed $C - 1$. Provide a mathematical derivation.

5. Compute the total scatter

$$S_T = \sum_{i=1}^{N} (x_i - \mu)(x_i - \mu)^T$$

   for a 2D dataset and verify $S_T = S_W + S_B$ numerically.

6. Take a toy 2D dataset with overlapping classes. Project it using PCA (top eigenvector) and LDA. Compare the separation along each direction.

7. Given a 3-class dataset: $\mathcal{C}_1 = \{(1,2),(2,1)\}, \mathcal{C}_2 = \{(5,5),(6,5)\}, \mathcal{C}_3 = \{(8,1),(9,2)\}$. Compute S_W, S_B, and find the generalised eigenvectors of $S_W^{-1}S_B$.

8. Derive the Fisher criterion
$$J(w) = \frac{w^T S_B w}{w^T S_W w}$$
and explain why maximising it gives optimal class separation.

9. Show that translating all points by a vector v_0 does not change the LDA directions.

10. For a dataset where one class has a single sample, discuss what happens to S_W and how it affects the LDA solution.

11. For an unbalanced 2-class dataset with 90 and 10 samples, compute S_W and S_B and discuss how class imbalance may bias the LDA direction.

12. Take a 2D dataset with all class means equal. Show that in this case, LDA reduces to PCA (maximising total variance).

MATLAB exercises

1. **LDA on Iris dataset.** Load the data using

```
load fisheriris
X = meas;    % N x 4
y = species; % N x 1
```

Implement LDA and plot the projected data in 2D.

2. **Compare PCA vs LDA.**

```
[coeff_pca, score_pca] = pca(X);
% Project onto top 2 PCs
Z_pca = X * coeff_pca(:,1:2);
% Project using LDA (from previous exercise)
Z_lda = X * W; % W from LDA
figure; scatter(Z_pca(:,1), Z_pca(:,2), 50, grp2idx(y), '
    filled');
figure; scatter(Z_lda(:,1), Z_lda(:,2), 50, grp2idx(y), '
    filled');
```

3. **Visualise 2D LDA projections.** Colour each class differently and add legend.

4. **Multiclass LDA.** Generate synthetic data with 4 classes:

```
X = [randn(20,2)+[1,1]; randn(20,2)+[5,5];
     randn(20,2)+[1,5]; randn(20,2)+[5,1]];
y = [ones(20,1); 2*ones(20,1); 3*ones(20,1); 4*ones(20,1)
    ];
```

5. **Compute LDA using SVD.** Center X, whiten S_W, and use SVD on $S_W^{-1/2} S_B S_W^{-1/2}$.

6. **Classification accuracy.** Split Iris dataset into 70% training, 30% testing. Use LDA projection + nearest centroid classifier and report accuracy.

7. **Synthetic overlapping data.** Generate two 2D Gaussian clouds with overlap and apply LDA. Visualise separation.

8. **Few samples in one class.** Take 2-class data with 1 sample in the second class. Compute S_W, attempt LDA, discuss results.

9. **Data orientation.** Repeat exercises for $X \in \mathbb{R}^{d \times N}$ and $X \in \mathbb{R}^{N \times d}$. Adapt code for each case:

```
% d x N
Z = W' * X;
% N x d
Z = X * W;
```

10. **LDA on ORL face dataset.** Implement PCA+LDA for the ORL face dataset. Vary the number of PCA components k and observe changes in recognition accuracy.

11. **LDA on handwriting digits.** Compare PCA alone versus PCA+LDA for classification of handwritten digits. Discuss performance differences.

12. **LDA on noisy images.** Experiment with noisy images. Analyse how noise affects PCA and LDA projections and classification accuracy.

13. **Lab Activity: Face Recognition with PCA + LDA.** Implement a simple face recognition system using PCA and LDA.

```
pkg load image; % for Octave
imgSize = [64,64]; dataDir = 'orl_faces';
imgFiles = dir([dataDir, '/s*/*.pgm']);
numImgs = length(imgFiles); X = zeros(prod(imgSize),numImgs);
labels = zeros(numImgs,1);
for i = 1:numImgs
 img = imread([imgFiles(i).folder '/' imgFiles(i).name]);
 img = imresize(img,imgSize);
 X(:,i) = double(img(:));
 labels(i) = str2num(regexp(imgFiles(i).folder,'\d+','match','
    once'){1});
end
% Compute PCA
meanFace = mean(X,2); X_centred = X - meanFace;
[U,S,V] = svd(X_centred,'econ');
k = 50; W_pca = U(:,1:k); X_pca = W_pca'*X_centred;
% Split train/test
trainIdx = 1:7:numImgs; testIdx = setdiff(1:numImgs, trainIdx)
    ;
X_train = X_pca(:,trainIdx); y_train = labels(trainIdx);
X_test = X_pca(:,testIdx); y_test = labels(testIdx);
% Train LDA
```

```
classes = unique(y_train); C = length(classes);
[n_features,n_samples] = size(X_train);
Sw = zeros(n_features); Sb = zeros(n_features);
meanTotal = mean(X_train,2);
for i=1:C
 Xi = X_train(:,y_train==classes(i));
 meanClass = mean(Xi,2);
 Sw = Sw + (Xi-meanClass)*(Xi-meanClass)';
 Sb = Sb + size(Xi,2)*(meanClass-meanTotal)*(meanClass-
    meanTotal)';
end
[V_lda,D] = eig(Sb,Sw); [~,idx] = sort(diag(D),'descend');
V_lda = V_lda(:,1:C-1);
X_train_lda = V_lda'*X_train; X_test_lda = V_lda'*X_test;
% Classification (nearest neighbor)
y_pred = zeros(size(y_test));
for i=1:length(y_test)
 dists = sum((X_train_lda - X_test_lda(:,i)).^2,1);
 [~,idx_min] = min(dists); y_pred(i) = y_train(idx_min);
end
accuracy = mean(y_pred==y_test);
disp(['PCA+LDA Face Recognition Accuracy: ', num2str(accuracy
   *100),'%']);
```

References

[1] D. C. Lay, S. R. Lay, and J. J. McDonald, *Linear Algebra and Its Applications*, 5th ed., Pearson Education, 2015.

[2] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed., Johns Hopkins University Press, 2013.

[3] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*, SIAM, 1997.

[4] R. A. Horn and C. R. Johnson, *Matrix Analysis*, 2nd ed., Cambridge University Press, 2013.

[5] M. Fazel, *Matrix Rank Minimization with Applications*, PhD Thesis, Stanford University, 2002.

[6] S. Boyd and L. Vandenberghe, *Applied Linear Algebra*, Cambridge University Press, 2018. `https://web.stanford.edu/~boyd/vmls/`

[7] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[8] T. Hastie, R. Tibshirani, and J. H. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed., Springer, 2009.

[9] I. T. Jolliffe, *Principal Component Analysis*, 2nd ed., Springer, 2002.

[10] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*, MIT Press, 2012.

[11] A. C. Rencher and W. F. Christensen, *Methods of Multivariate Analysis*, 2nd ed., Wiley, 2012.

[12] C. Bouveyron, G. Celeux, T. B. Murphy, and A. E. Raftery, *Model-Based Clustering and Classification for Data Science: With Applications in R*, Cambridge University Press, 2019.

[13] E. Alpaydin, *Introduction to Machine Learning*, 4th ed., Adaptive Computation and Machine Learning series, The MIT Press, 2020.

[14] A. Burkov, *The Hundred-Page Machine Learning Book*, freecomputerbooks.com, 2019.

[15] M. P. Deisenroth, A. A. Faisal, and C. S. Ong, *Mathematics for Machine Learning*, Cambridge University Press, 2020.

[16] G. James, D. Witten, T. Hastie, R. Tibshirani, and J. Taylor, *An Introduction to Statistical Learning: With Applications in R*, Springer, 2021.

[17] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*, Cambridge University Press, 2014.

[18] F. Cucker and S. Smale, *On the Mathematical Foundations of Learning*, Bull. Amer. Math. Soc., 39(1) (2002), 1–49.

[19] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*, 2nd ed., MIT Press, 2018.

[20] A. Ng, *CS229 Lecture Notes: PCA, SVD, Eigenfaces*, Stanford University. https://cs229.stanford.edu/notes2021fall/lecture14-pca.pdf

[21] A. Ng and T. Ma, *CS229 Main Notes*, Stanford University. https://cs229.stanford.edu/main_notes.pdf

[22] JHU Vision Group, *PCA and LDA Lecture Notes*, Johns Hopkins University, 2014. https://www.cs.jhu.edu/~ayuille/courses/Stat161-261-Spring14/RevisedLectureNote10.pdf

[23] J. Shlens, *A Tutorial on Principal Component Analysis*, 2014. https://arxiv.org/abs/1404.1100

[24] B. Ghojogh and M. Crowley, *Unsupervised and Supervised PCA: A Tutorial*, 2019. https://arxiv.org/abs/1906.03148

[25] L. van der Maaten, E. Postma, and H. van den Herik, *Tutorial on Dimensionality Reduction*, Technical Report, 2009. https://arxiv.org/abs/0907.3742

[26] A. Arenas-García, A. Petersen, and B. Schölkopf, *Kernel Multivariate Analysis*, 2013. https://arxiv.org/abs/1310.5089

[27] B. Ghojogh, F. Karray, and M. Crowley, *Unified Spectral Dimensionality Reduction*, 2021. https://arxiv.org/abs/2106.15379

[28] S. Khan, *PCA-Based Face Recognition System Using ORL Database*, MATLAB Central File Exchange. https://www.mathworks.com/matlabcentral/fileexchange/43610-pca-based-face-recognition-system-using-orl-database

[29] AT&T Laboratories / Cambridge, *ORL Face Dataset (AT&T Database of Faces)*. https://www.kaggle.com/datasets/tavarez/the-orl-database-for-training-and-testing