Chapter 3

MATHEMATICAL PRELIMINARIES

CONTENTS

3.1 Notations

I. Gradient (∇) and Hessian ($\nabla\nabla^\top$)

A gradient of a function, denoted by ∇f, is a vector of partial derivatives. For example, given a D-dimensional feature vector $\boldsymbol{x}$, the *Gradient* of the function f with respect to the feature vector $\boldsymbol{x}$ is:

$$\nabla_{\boldsymbol{x}} f = \left(\frac{\partial f}{\partial x^{(1)}}, \frac{\partial f}{\partial x^{(2)}}, \cdots, \frac{\partial f}{\partial x^{(D)}}\right), \qquad \text{where} \quad \boldsymbol{x} = \left(x^{(1)}, x^{(2)}, \cdots, x^{(D)}\right).$$

Moreover, the *Hessian* of the function f with respect to the feature vector $\boldsymbol{x}$ is defined as:

$$\nabla_{\boldsymbol{x}} \nabla_{\boldsymbol{x}}^\top f = \begin{pmatrix} \frac{\partial^2 f}{\partial x^{(1)} \partial x^{(1)}} & \cdots & \frac{\partial^2 f}{\partial x^{(1)} \partial x^{(D)}} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x^{(D)} \partial x^{(1)}} & \cdots & \frac{\partial^2 f}{\partial x^{(D)} \partial x^{(D)}} \end{pmatrix}.$$

II. Hamadard Product ($\odot$)

For two matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ of the same dimension $N \times M$, the *Hadamard product* $\boldsymbol{A} \odot \boldsymbol{B}$ is another matrix of the same dimension as the operands $\boldsymbol{A}$ and $\boldsymbol{B}$, with elements given by

$$(\boldsymbol{A} \odot \boldsymbol{B})_{n,m} = (\boldsymbol{A})_{n,m} (\boldsymbol{B})_{n,m}, \qquad \text{for} \quad n = 1, \cdots, N;\ m = 1, \cdots, M,$$

i.e. elementwise product. For example,

$$\boldsymbol{A} \odot \boldsymbol{B} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} \odot \begin{pmatrix} 0 & 2 & 3 \\ 2 & 4 & 1 \\ 0 & 3 & 0 \end{pmatrix} = \begin{pmatrix} 0\times 0 & 1\times 2 & 0\times 3 \\ 1\times 2 & 1\times 4 & 1\times 1 \\ 0\times 0 & 1\times 3 & 0\times 0 \end{pmatrix} = \begin{pmatrix} 0 & 2 & 0 \\ 2 & 4 & 1 \\ 0 & 3 & 0 \end{pmatrix}.$$

3.2 Dis-similarity measures

In clustering analysis, *e.g.* the K-means clustering and the K-nearest neighbours, a distance (or dis-similarity) measure is chosen to quantify the dis-similarity among each observations. It is therefore important to define a distance measure between two observations. Since there are different types of variables, namely the continuous variable, the ordinal variable, the binary variable, and the nominal variable, the distance or dissimilarity between two observations should be defined differently according to their variable type; indeed, most feature variables in market security data are continuous, such as security prices, earnings per share, book to market ratios, etc. In contrast, the insurance data are usually of a mixture type with continuous, binary, and categorical variables; for instance, the binary variable often includes gender and whether the policyholder has previously insured, while the categorical variables might include occupation, social status, and living region or areas of the policyholder. The medical data are mostly continuous biometrics such as blood pressure, age, and height that can be measured by the mechanical devices. Denote $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ in $\mathbb{R}^D$ be the two observations:

I. Continuous Variables

If all feature variables are continuous, we have

1. **Euclidean Distance**:
$$d(\boldsymbol{x}_i, \boldsymbol{x}_j) := \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2 = \sqrt{\sum_{m=1}^{D} \left(x_i^{(m)} - x_j^{(m)}\right)^2}. \tag{3.2.1}$$
2. **City-Block Distance**:
$$d(\boldsymbol{x}_i, \boldsymbol{x}_j) := \sum_{m=1}^{D} |x_i^{(m)} - x_j^{(m)}|. \tag{3.2.2}$$
3. **Minkowski Distance**:
$$d(\boldsymbol{x}_i, \boldsymbol{x}_j) := \left(\sum_{m=1}^{D} |x_i^{(m)} - x_j^{(m)}|^q\right)^{1/q}. \tag{3.2.3}$$
4. **Negative Cosine Similarity Distance**: Cosine similarity is defined as:
$$s(\boldsymbol{x}_i, \boldsymbol{x}_j) := \frac{\sum_{m=1}^{D} x_i^{(m)} x_j^{(m)}}{\sqrt{\sum_{m=1}^{D} \left(x_i^{(m)}\right)^2} \sqrt{\sum_{m=1}^{D} \left(x_j^{(m)}\right)^2}}, \tag{3.2.4}$$
is a measure of similarity of the directions of two vectors:
   (a) if two vectors point to the same direction, the cosine similarity is equal to 1;
   (b) if the vectors are orthogonal, the cosine similarity is 0;
   (c) if two vectors point to the opposite direction, the cosine similarity is equal to -1.

However, distance is a measurement of dis-similarity instead of similarity, we therefore multiply the cosine similarity by -1 for a distance measure, *i.e.*

$$d(\boldsymbol{x}_i, \boldsymbol{x}_j) = -s(\boldsymbol{x}_i, \boldsymbol{x}_j)\,. \tag{3.2.5}$$

5. **Chebychev Distance**: The maximal distance between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ along their coordinate dimension:

$$d(\boldsymbol{x}_i, \boldsymbol{x}_j) := \max_{m=1,\cdots,D} \left(\left| x_i^{(m)} - x_j^{(m)} \right| \right)\,. \tag{3.2.6}$$

6. **Mahalanobis Distance**: Dissimilarity measure between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ of the same distribution with the covariance matrix $\boldsymbol{\Sigma}$:

$$d(\boldsymbol{x}_i, \boldsymbol{x}_j) := \sqrt{(\boldsymbol{x}_i - \boldsymbol{x}_j)^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}_i - \boldsymbol{x}_j)}\,. \tag{3.2.7}$$

II. Ordinal Variables

If all feature variables are ordinal, we can rescale $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ to the unit interval $(0, 1)$ by:

$$z_i^{(m)} = \frac{r_i^{(m)} - 1}{M_m - 1} \qquad \text{and} \qquad z_j^{(m)} = \frac{r_j^{(m)} - 1}{M_m - 1}\,, \qquad \text{for} \quad m = 1, \cdots, D\,, \tag{3.2.8}$$

where $r_*^{(m)}$ is the rank of $x_*^{(m)}$ in the feature variable m and M_m is the level of the feature variable m. We then treat $\boldsymbol{z}_i := (z_i^{(1)}, \cdots, z_i^{(D)})$ and $\boldsymbol{z}_j := (z_j^{(1)}, \cdots, z_j^{(D)})$ as in the case of continuous variable.

III. Binary Variables

If all feature variables are binary, we have:

1. **Hamming Distance**: The number of places where $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ differ:

$$d(\boldsymbol{x}_i, \boldsymbol{x}_j) := \sum_{m=1}^{D} \mathbb{1} \left\{ x_i^{(m)} \neq x_j^{(m)} \right\}\,. \tag{3.2.9}$$

2. **Simple matching coefficient**:

$$d(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{r + s}{q + r + s + t}\,. \tag{3.2.10}$$

3. **Jaccard coefficient**:

$$d(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{r + s}{q + r + s}\,. \tag{3.2.11}$$

where q, r, s, t come from the contingency table below:

		$\boldsymbol{x}_j$	
		0	1
$\boldsymbol{x}_i$	0	t	s
	1	r	q

Table 3.2.1: Contingency table for binary variables.

Example 3.2.1. Suppose that Marry and Peter join a diagnostic test, which consists of five different tests and the their test results are listed below:

Name	Sex	Test 1	Test 2	Test 3	Test 4	Test 5
Marry	F	P	P	N	P	N
Peter	M	N	N	N	P	P

Table 3.2.2: Diagnostic test results of Marry and Peter.

Denote F (Female) and N (Negative) as 0, while M (Male) and P (Positive) as 1, we have the following contingency table:

		Peter	
		0	1
Marry	0	1	2
	1	2	1

Table 3.2.3: Contingency table for Marry and Peter.

The distance measures are:

1. **Hamming Distance**:
$$d(\text{Marry}, \text{Peter}) := \sum_{m=1}^{D} \mathbb{1}\left\{x_i^{(m)} \neq x_j^{(m)}\right\} = 1+1+1+0+0+1 = 4\,.$$
2. **Simple matching coefficient**:
$$d(\text{Marry}, \text{Peter}) = \frac{r+s}{q+r+s+t} = \frac{2+2}{1+2+2+1} = \frac{2}{3}\,.$$
3. **Jaccard coefficient**:
$$d(\text{Marry}, \text{Peter}) = \frac{r+s}{q+r+s} = \frac{2+2}{2+1+1} = \frac{4}{5}\,.$$

IV. Nominal (Categorical) Variables

If all feature variables are nominal, we can use
$$d(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{D - d_m(\boldsymbol{x}_i, \boldsymbol{x}_j)}{D}\,, \tag{3.2.12}$$
where d_m is the Hamming distance.

3.3 Matrix Theory

We here recall the relevant concepts and results in matrix theory commonly used in multivariate statistical analysis. In this section, all matrices are $D \times D$ matrices unless specified.

I. Determinant

Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be $D \times D$ square matrices and let $|\boldsymbol{A}|$ denote the **determinant** of $\boldsymbol{A}$. Basic properties include the followings:

1. $|\boldsymbol{A}^\top| = |\boldsymbol{A}|$;
2. $|\alpha\boldsymbol{A}| = \alpha^D|\boldsymbol{A}|$, for any scalar α;
3. $|\boldsymbol{AB}| = |\boldsymbol{A}| \times |\boldsymbol{B}|$ (Note that $|\boldsymbol{A}+\boldsymbol{B}| \neq |\boldsymbol{A}| + |\boldsymbol{B}|$ in general);

The determinant of $|\boldsymbol{A}|$ can be defined by the Leibniz formula:

$$|\boldsymbol{A}| = \sum_{\boldsymbol{\sigma}\in\Gamma_D} \left(\operatorname{sgn}(\boldsymbol{\sigma}) \prod_{d=1}^{D} a_{d,\sigma_d} \right),$$

where Γ_D is the overall permutations of the set $\{1, 2, \cdots, D\}$ with $\boldsymbol{\sigma} = \{\sigma_1, \cdots, \sigma_D\}$ being one permutation, and $\operatorname{sgn}(\boldsymbol{\sigma}) = 1$ with even number of swaps needed to change from the ordered set $\{1, 2, \cdots, D\}$ to $\boldsymbol{\sigma}$, while $\operatorname{sgn}(\boldsymbol{\sigma}) = -1$ with odd number of swaps. Note that only adjacent swaps are allowed and $|\Gamma_D| = D!$.

II. Inverse

Let $\boldsymbol{A}$ be a $D \times D$ **nonsingular** matrix, *i.e.* its inverse $\boldsymbol{A}^{-1}$ exists.

4. $(\boldsymbol{A}^{-1})^\top = (\boldsymbol{A}^\top)^{-1}$;
5. $(\boldsymbol{AB})^{-1} = \boldsymbol{B}^{-1}\boldsymbol{A}^{-1}$;
6. $|\boldsymbol{A}^{-1}| = 1/|\boldsymbol{A}|$;
7. If $\boldsymbol{A}$ is an orthogonal matrix, then $\boldsymbol{A}^{-1} = \boldsymbol{A}^\top$;
8. If $\boldsymbol{C} = \operatorname{diag}(c_1, \cdots, c_D)$ with $c_d \neq 0$, then $\boldsymbol{C}^{-1} = \operatorname{diag}(c_1^{-1}, \cdots, c_D^{-1})$;

III. Block matrix determinant

Let the matrices $\boldsymbol{A} \in \mathbb{R}^{N\times N}, \boldsymbol{B} \in \mathbb{R}^{N\times M}, \boldsymbol{C} \in \mathbb{R}^{M\times N}, \boldsymbol{D} \in \mathbb{R}^{M\times M}$, we then consider a block matrix $[\boldsymbol{P}] \in \mathbb{R}^{(N+M)\times(M+N)}$ in the form:

$$[\boldsymbol{P}] = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{pmatrix}.$$

9. If $\boldsymbol{B} = \boldsymbol{0} \in \mathbb{R}^{N\times M}, \boldsymbol{C} = \boldsymbol{0} \in \mathbb{R}^{M\times D}, \boldsymbol{D} = \boldsymbol{I}_M$, then the determinant of the block matrix, denoted as $[\boldsymbol{P}_{N+M}]$, is $|[\boldsymbol{P}]| = |\boldsymbol{A}|$.

   Proof. The claim can easily be shown by an induction argument. We first consider a simplier case where $M = 1$ such that

$$[\boldsymbol{P}_{N+1}] = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{0} \\ \boldsymbol{0} & 1 \end{pmatrix}.$$

Then by the Leibniz formula, the determinant of this block matrix is

$$|[\boldsymbol{P}_{N+1}]| = \sum_{\boldsymbol{\sigma}\in\Gamma_{N+1}} \left(\text{sgn}(\boldsymbol{\sigma}) \prod_{n=1}^{N+1} p_{n,\sigma_n} \right) = \sum_{\boldsymbol{\sigma}\in\Gamma_{N+1}} \left(\text{sgn}(\boldsymbol{\sigma}) p_{N+1,\sigma_{N+1}} \prod_{n=1}^{N} p_{n,\sigma_n} \right),$$

where Γ_{N+1} is again the set of all permutations of $\{1,\cdots,N,N+1\}$. Note that $p_{N+1,1} = \cdots = p_{N+1,N} = 0$ and $p_{N+1,N+1} = 1$, or equivalently, the every summand with a permutation $\boldsymbol{\sigma} = \{\sigma_1,\cdots,N+1,\cdots,\sigma_{N+1}\}$, where $\sigma_{N+1} \neq N+1$, vanishes while the summand with a permutation $\boldsymbol{\sigma} = \{\sigma_1,\cdots,\sigma_N,N+1\}$, where $\sigma_1,\cdots,\sigma_N \neq N+1$, has a non-zero value. Therefore, we can construct Γ_N, which is the set of all permutations of $\{1,\cdots,N\}$, such that the determinant can be further simplified as:

$$|[\boldsymbol{P}_{N+1}]| = \sum_{\boldsymbol{\sigma}\in\Gamma_N} \left(\text{sgn}(\boldsymbol{\sigma}) \prod_{n=1}^{N} p_{n,\sigma_n} \right) = \sum_{\boldsymbol{\sigma}\in\Gamma_N} \left(\text{sgn}(\boldsymbol{\sigma}) \prod_{n=1}^{N} a_{n,\sigma_n} \right) = |\boldsymbol{A}|.$$

We then notice that for $M = 2$, the block matrix $[\boldsymbol{P}_{N+2}]$ has the form:

$$[\boldsymbol{P}_{N+2}] = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & 1 & 0 \\ \boldsymbol{0} & 0 & 1 \end{pmatrix} = \left(\begin{array}{cc|c} \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & 1 & 0 \\ \hline \boldsymbol{0} & 0 & 1 \end{array}\right) = \begin{pmatrix} \boldsymbol{P}_{N+1} & \boldsymbol{0} \\ \boldsymbol{0} & 1 \end{pmatrix},$$

which has a determinant $|[\boldsymbol{P}_{N+2}]| = |[\boldsymbol{P}_{N+1}]| = |\boldsymbol{A}|$; and in general, the result follows by mathematical induction, *i.e.* $|[\boldsymbol{P}_{N+M}]| = |[\boldsymbol{P}_{N+M-1}]| = \cdots = |[\boldsymbol{P}_N]| = |\boldsymbol{A}|$ and we complete the proof; □

10. If $\boldsymbol{B} = \boldsymbol{0} \in \mathbb{R}^{N\times M}$, then the determinant of the block matrix $[\boldsymbol{P}]$ is $|[\boldsymbol{P}]| = |\boldsymbol{A}||\boldsymbol{D}|$.

    Proof. We first decompose $[\boldsymbol{P}]$ into:

    $$[\boldsymbol{P}] = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{0} \\ \boldsymbol{C} & \boldsymbol{D} \end{pmatrix} = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I}_M \end{pmatrix} \begin{pmatrix} \boldsymbol{I}_N & \boldsymbol{0} \\ \boldsymbol{C} & \boldsymbol{I}_M \end{pmatrix} \begin{pmatrix} \boldsymbol{I}_N & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{D} \end{pmatrix}.$$

    Then using Property 3 and 9, we have $|[\boldsymbol{P}]| = |\boldsymbol{A}| \cdot 1 \cdot |\boldsymbol{D}| = |\boldsymbol{A}||\boldsymbol{D}|$, which completes the proof; □

11. If $\boldsymbol{A}$ (resp. $\boldsymbol{D}$) is invertible, *i.e.* its inverse $\boldsymbol{A}^{-1}$ (resp. $\boldsymbol{D}^{-1}$) exists, then the determinant of the block matrix $[\boldsymbol{P}]$ is $|[\boldsymbol{P}]| = |\boldsymbol{A}||\boldsymbol{D} - \boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B}|$ (resp. $|[\boldsymbol{P}]| = |\boldsymbol{D}||\boldsymbol{A} - \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C}|$).

    Proof. We first note that

    $$\begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{pmatrix} \begin{pmatrix} \boldsymbol{I}_N & -\boldsymbol{A}^{-1}\boldsymbol{B} \\ \boldsymbol{0} & \boldsymbol{I}_M \end{pmatrix} = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{0} \\ \boldsymbol{C} & \boldsymbol{D} - \boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B} \end{pmatrix}.$$

    Then using Property 3 and 10, we have $|[\boldsymbol{P}]| \cdot 1 = |\boldsymbol{A}||\boldsymbol{D} - \boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B}|$, which completes the proof; □

IV. Trace

Given a $D \times D$ matrix $\boldsymbol{A} = (a_{ij})$, the trace of $\boldsymbol{A}$, $\text{tr}(\boldsymbol{A})$, is the sum of the diagonal entries of $\boldsymbol{A}$:

$$\text{tr}(\boldsymbol{A}) = \sum_{d=1}^{D} a_{dd}.$$

Suppose $\boldsymbol{A}$ is a square matrix with eigenvalues $\lambda_1,\cdots,\lambda_D$. Then $\text{tr}(\boldsymbol{A})$ is the sum of the eigenvalues of $\boldsymbol{A}$:

$$\text{tr}(\boldsymbol{A}) = \sum_{d=1}^{D} \lambda_d.$$

The trace of a matrix satisfies the following properties:

12. $\text{tr}(\boldsymbol{A}) = \text{tr}(\boldsymbol{A}^\top)$,

13. $\text{tr}(\alpha\boldsymbol{A}+\beta\boldsymbol{B})=\alpha\text{tr}(\boldsymbol{A})+\beta\text{tr}(\boldsymbol{B})$ for $\alpha,\beta\in\mathbb{R}$,
14. Cyclic invariant: $\text{tr}(\boldsymbol{ABC})=\text{tr}(\boldsymbol{CAB})=\text{tr}(\boldsymbol{BCA})$. In general, we have
$$\text{tr}\left(\boldsymbol{A}_1\cdot\boldsymbol{A}_2\cdots\boldsymbol{A}_N\right)=\text{tr}\left(\boldsymbol{A}_N\cdot\boldsymbol{A}_1\cdots\boldsymbol{A}_{N-1}\right).$$

V. Positive definite matrix

A $D\times D$ symmetric matrix $\boldsymbol{A}$ is called **positive definite** (denoted by $\boldsymbol{A}\succ\boldsymbol{0}$) if for any nonzero column vector $\boldsymbol{x}\in\mathbb{R}^D$, $\boldsymbol{x}^\top\boldsymbol{Ax}>0$. (semi-positive definite, denoted by $\boldsymbol{A}\succeq\boldsymbol{0}$, if $\boldsymbol{x}^\top\boldsymbol{Ax}\geq 0$ for any $\boldsymbol{x}\in\mathbb{R}^D$).

15. If $\boldsymbol{A}\succ 0$, then $\boldsymbol{A}^{-1}\succ 0$;
16. For any $D\times Q$ matrix $\boldsymbol{B}$, $\boldsymbol{B}^\top\boldsymbol{B}\succeq 0$; indeed, let $\boldsymbol{y}=\boldsymbol{Bx}$, where $\boldsymbol{x}\in\mathbb{R}^Q$, then
$$\boldsymbol{x}^\top\boldsymbol{B}^\top\boldsymbol{Bx}=\boldsymbol{y}^\top\boldsymbol{y}=\sum_{d=1}^{D}y_d^2\geq 0\,;$$
17. The sample covariance matrix is semi-positive definite. It is suffice to show that the SSCP (sum of squares and cross products) matrix $\boldsymbol{A}\succeq\boldsymbol{0}$.

    Proof. Note that
    $$\boldsymbol{A}=\sum_{d=1}^{D}(\boldsymbol{x}^{(d)}-\bar{\boldsymbol{x}}_D)(\boldsymbol{x}^{(d)}-\bar{\boldsymbol{x}}_D)^\top=(\boldsymbol{X}-\boldsymbol{I}_D\bar{\boldsymbol{X}}_D^\top)^\top(\boldsymbol{X}-\boldsymbol{I}_D\bar{\boldsymbol{X}}_D^\top)\succeq\boldsymbol{0}\,,$$
    by using the Property 16 as above; □

VI. Eigenvalues and eigenvectors

$|\boldsymbol{A}-\lambda\boldsymbol{I}_D|$ is a polynomial in λ of degree D. The **eigenvalues** (or **latent roots**) of $\boldsymbol{A}$, denoted by $\lambda_1,\cdots,\lambda_D$, are the roots of the equation $|\boldsymbol{A}-\lambda\boldsymbol{I}_D|=0$; and so it must be a singular matrix and there should be a nonzero vector $\boldsymbol{h}_d$ such that $(\boldsymbol{A}-\lambda_d\boldsymbol{I}_D)\boldsymbol{h}_d=\boldsymbol{0}$ (or $\boldsymbol{A}\boldsymbol{h}_d=\lambda_d\boldsymbol{h}_d$), which is called the **eigenvector** of $\boldsymbol{A}$ corresponding to λ_d. If the vector $\boldsymbol{h}_d$ has a unit length (*i.e.* $\boldsymbol{h}_d^\top\boldsymbol{h}_d=1$), then it is called a **normalized (unit) eigenvector** of $\boldsymbol{A}$.

There are some important properties of eigenvalues and eigenvectors.

18. If $\boldsymbol{A}$ is a real symmetric matrix, then its eigenvalues are all real;
19. If $\boldsymbol{A}\succeq\boldsymbol{0}$, then all the eigenvalues of $\boldsymbol{A}$ are non-negative;
20. The eigenvalues of $\boldsymbol{A}$ and $\boldsymbol{A}^\top$ are the same;
21. If $\boldsymbol{A}$ and $\boldsymbol{B}$ are $D\times D$, $\boldsymbol{A}$ is nonsingular then eigenvalues of $\boldsymbol{AB}$ and $\boldsymbol{BA}$ are equal;
22. If $\lambda_1,\cdots,\lambda_D$ are the eigenvalues of a nonsingular matrix $\boldsymbol{A}$, then $\lambda_1^{-1},\cdots,\lambda_D^{-1}$ are the eigenvalues of $\boldsymbol{A}^{-1}$;
23. If $\boldsymbol{A}$ is a real symmetric matrix, and λ_i and λ_j are two distinct eigenvalues of $\boldsymbol{A}$, then the corresponding eigenvectors $\boldsymbol{h}_i$ and $\boldsymbol{h}_j$ are orthogonal; indeed, by definition, $\boldsymbol{A}\boldsymbol{h}_i=\lambda_i\boldsymbol{h}_i$ and $\boldsymbol{A}\boldsymbol{h}_j=\lambda_j\boldsymbol{h}_j$, and therefore $\boldsymbol{h}_j^\top\boldsymbol{A}\boldsymbol{h}_i=\lambda_i\boldsymbol{h}_j^\top\boldsymbol{h}_i$ and $\boldsymbol{h}_i^\top\boldsymbol{A}\boldsymbol{h}_j=\lambda_j\boldsymbol{h}_i^\top\boldsymbol{h}_j$, and hence $(\lambda_i-\lambda_j)\boldsymbol{h}_j^\top\boldsymbol{h}_i=0$, implying that $\boldsymbol{h}_i$ and $\boldsymbol{h}_j$ are orthogonal;

24. Let $\boldsymbol{A}$ be a real symmetric matrix and $\boldsymbol{H}$ be a $D\times D$ matrix whose columns are normalized eigenvectors $\boldsymbol{h}_1,\cdots,\boldsymbol{h}_D$ of $\boldsymbol{A}$, *i.e.* $\boldsymbol{H}=(\boldsymbol{h}_1,\cdots,\boldsymbol{h}_D)$ then $\boldsymbol{H}^\top\boldsymbol{A}\boldsymbol{H}=\boldsymbol{\Lambda}=\text{diag}(\lambda_1,\cdots,\lambda_D)$ (**Diagonalization** of $\boldsymbol{A}$) or $\boldsymbol{A}=\boldsymbol{H}\boldsymbol{\Lambda}\boldsymbol{H}^\top$ (**Spectral decomposition** of $\boldsymbol{A}$).

    Proof. Write $\boldsymbol{A}\boldsymbol{H}=(\boldsymbol{A}\boldsymbol{h}_1,\cdots,\boldsymbol{A}\boldsymbol{h}_D)=(\lambda_1\boldsymbol{h}_1,\cdots,\lambda_D\boldsymbol{h}_D)=\boldsymbol{H}\boldsymbol{\Lambda}$, implying that $\boldsymbol{H}^\top\boldsymbol{A}\boldsymbol{H}=\boldsymbol{\Lambda}$; □

25. Trace of a matrix $\boldsymbol{A}$ is the sum of all its eigenvalues and the determinant of $\boldsymbol{A}$ is the product of all its eigenvalues, *i.e.* $\text{tr}(\boldsymbol{A})=\sum_{d=1}^D\lambda_d$ and $|\boldsymbol{A}|=\prod_{d=1}^D\lambda_d$;

VII. Symmetric square root of a semi-positive definite matrix

Let $\boldsymbol{A}\succeq\boldsymbol{0}$ with eigenvalues $\lambda_1,\cdots,\lambda_D$ and their corresponding eigenvectors are $\boldsymbol{h}_1,\cdots,\boldsymbol{h}_D$. Define the matrix $\boldsymbol{H}=(\boldsymbol{h}_1,\cdots,\boldsymbol{h}_D),\boldsymbol{\Lambda}=\text{diag}(\lambda_1,\cdots,\lambda_D)$ and $\boldsymbol{\Lambda}^{1/2}=\text{diag}(\sqrt{\lambda_1},\cdots,\sqrt{\lambda_D})$. Note that $\boldsymbol{A}=\boldsymbol{H}\boldsymbol{\Lambda}\boldsymbol{H}^\top$. The **symmetric square root** of $\boldsymbol{A}$ is given by $\boldsymbol{A}^{1/2}=\boldsymbol{H}\boldsymbol{\Lambda}^{1/2}\boldsymbol{H}^\top$. To see this, $\boldsymbol{A}^{1/2}\boldsymbol{A}^{1/2}=\boldsymbol{H}\boldsymbol{\Lambda}^{1/2}\boldsymbol{H}^\top\boldsymbol{H}\boldsymbol{\Lambda}^{1/2}\boldsymbol{H}^\top=\boldsymbol{H}\boldsymbol{\Lambda}\boldsymbol{H}^\top=\boldsymbol{A}$.

Lemma 3.3.1. Suppose that the $D\times D$ positive definite matrix $\boldsymbol{A}$ has the ordered eigenvalues $\lambda_1\geq\cdots\geq\lambda_D>0$ and the corresponding unit (or normalized) eigenvectors are $\boldsymbol{\alpha}_1,\cdots,\boldsymbol{\alpha}_D$. Then,

1. $\max\limits_{\boldsymbol{l}\neq\boldsymbol{0}}\dfrac{\boldsymbol{l}^\top\boldsymbol{A}\boldsymbol{l}}{\boldsymbol{l}^\top\boldsymbol{l}}=\lambda_1$, and this maximum is attained at $\boldsymbol{l}=\boldsymbol{\alpha}_1$;
2. $\min\limits_{\boldsymbol{l}\neq\boldsymbol{0}}\dfrac{\boldsymbol{l}^\top\boldsymbol{A}\boldsymbol{l}}{\boldsymbol{l}^\top\boldsymbol{l}}=\lambda_D$, and this minimum is attained at $\boldsymbol{l}=\boldsymbol{\alpha}_D$;
3. in general, for $i=1,2,\cdots,D-1$, $\max\limits_{\boldsymbol{l}\perp\boldsymbol{\alpha}_1,\cdots,\boldsymbol{\alpha}_K}\dfrac{\boldsymbol{l}^\top\boldsymbol{A}\boldsymbol{l}}{\boldsymbol{l}^\top\boldsymbol{l}}=\lambda_{K+1}$, and this maximum is attained at $\boldsymbol{l}=\boldsymbol{\alpha}_{K+1}$.

Proof. Let $\boldsymbol{A}=\boldsymbol{H}\boldsymbol{D}\boldsymbol{H}^\top$ be the spectral decomposition of $\boldsymbol{A}$, where $\boldsymbol{H}=(\boldsymbol{\alpha}_1,\cdots,\boldsymbol{\alpha}_D)$, $\boldsymbol{D}=\text{diag}(\lambda_1,\cdots,\lambda_D)$; and also define $\boldsymbol{A}^{1/2}=\boldsymbol{H}\boldsymbol{D}^{1/2}\boldsymbol{H}^\top$. Let $\boldsymbol{u},\boldsymbol{l}\in\mathbb{R}^D$ be the vectors such that $\boldsymbol{u}=\boldsymbol{H}^\top\boldsymbol{l}$. Then,

$$\frac{\boldsymbol{l}^\top\boldsymbol{A}\boldsymbol{l}}{\boldsymbol{l}^\top\boldsymbol{l}}=\frac{\boldsymbol{l}^\top\boldsymbol{A}^{1/2}\boldsymbol{A}^{1/2}\boldsymbol{l}}{\boldsymbol{l}^\top\boldsymbol{H}\boldsymbol{H}^\top\boldsymbol{l}}=\frac{\boldsymbol{l}^\top\boldsymbol{H}\boldsymbol{D}^{1/2}\boldsymbol{H}^\top\boldsymbol{H}\boldsymbol{D}^{1/2}\boldsymbol{H}^\top\boldsymbol{l}}{\boldsymbol{u}^\top\boldsymbol{u}}=\frac{\boldsymbol{u}^\top\boldsymbol{D}\boldsymbol{u}}{\boldsymbol{u}^\top\boldsymbol{u}}=\frac{\sum_{i=1}^D\lambda_iu_i^2}{\sum_{i=1}^Du_i^2}\leq\lambda_1\frac{\sum_{i=1}^Du_i^2}{\sum_{i=1}^Du_i^2}=\lambda_1\,,$$

where in inequality is due to $\lambda_1\geq\cdots\geq\lambda_D>0$. When $\boldsymbol{l}=\boldsymbol{\alpha}_1$, $\boldsymbol{y}=\boldsymbol{H}^\top\boldsymbol{\alpha}_1=(1,0,\cdots,0)^\top\in\mathbb{R}^D$, since $\boldsymbol{\alpha}_1\perp\boldsymbol{\alpha}_j$ for $j=2,\cdots,K$, and $\dfrac{\boldsymbol{\alpha}_1^\top\boldsymbol{A}\boldsymbol{\alpha}_1}{\boldsymbol{\alpha}_1^\top\boldsymbol{\alpha}_1}=\dfrac{\boldsymbol{y}^\top\boldsymbol{D}\boldsymbol{y}}{\boldsymbol{y}^\top\boldsymbol{y}}=\lambda_1$. Similarly, we can prove the claim 2 by using a parallel argument. For the claim 3, if $\boldsymbol{u}=\boldsymbol{H}^\top\boldsymbol{l}$ and $\boldsymbol{l}=\boldsymbol{H}\boldsymbol{u}=u_1\boldsymbol{\alpha}_1+\cdots+u_D\boldsymbol{\alpha}_D$. Hence, if $\boldsymbol{l}\perp\boldsymbol{\alpha}_1,\cdots,\boldsymbol{\alpha}_K$, then $0=\boldsymbol{\alpha}_i^\top\boldsymbol{l}=u_1\boldsymbol{\alpha}_i^\top\boldsymbol{\alpha}_1+\cdots+u_D\boldsymbol{\alpha}_i^\top\boldsymbol{\alpha}_D=u_i$ for all $i=1,2,\cdots,K$. Therefore,

$$\frac{\boldsymbol{l}^\top\boldsymbol{A}\boldsymbol{l}}{\boldsymbol{l}^\top\boldsymbol{l}}=\frac{\sum_{i=K+1}^D\lambda_iu_i^2}{\sum_{i=K+1}^Du_i^2}\leq\lambda_{K+1}\,,$$

therefore this claim then follows by using the argument given in the proof of the claim 1. □

VIII. Matrix derivatives

The derivative of the trace operator with respect to a matrix satisfies the following properties. Using the notation $\nabla_{\boldsymbol{A}}f:=\left(\frac{\partial}{\partial a_{ij}}f\right)_{i,j}$, we have

26. $\nabla_{\boldsymbol{A}}\text{tr}(\boldsymbol{A}\boldsymbol{B})=\boldsymbol{B}^\top$;
27. $\nabla_{\boldsymbol{A}}\text{tr}(\boldsymbol{A}\boldsymbol{B}\boldsymbol{A}^\top\boldsymbol{C})=\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}+\boldsymbol{C}^\top\boldsymbol{A}\boldsymbol{B}^\top$;
28. $\nabla_{\boldsymbol{A}^\top}\text{tr}(\boldsymbol{A}\boldsymbol{B}\boldsymbol{A}^\top\boldsymbol{C})=\boldsymbol{B}^\top\boldsymbol{A}^\top\boldsymbol{C}+\boldsymbol{B}\boldsymbol{A}^\top\boldsymbol{C}$;

Proof. We only prove the first statement. The second and third statements then follow from the chain rule. Denote $\boldsymbol{A} = (a_{ij})$ and $\boldsymbol{B} = (b_{ij})$. Since

$$\mathrm{tr}(\boldsymbol{AB}) = \sum_l \sum_k a_{lk} b_{kl} \,,$$

we then have

$$\frac{d\mathrm{tr}(\boldsymbol{AB})}{da_{ij}} = \frac{d}{da_{ij}} \sum_j \sum_k a_{lk} b_{kl} = b_{ji} \,,$$

which proves the assertion. □

Moreover, let $\boldsymbol{A} \in \mathbb{R}^{D\times D}$ and $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^D$, then

29. $\nabla_{\boldsymbol{x}} \boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} = 2(\boldsymbol{A} + \boldsymbol{A}^\top)\boldsymbol{x}$.
30. $\nabla_{\boldsymbol{A}} \boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} = \nabla_{\boldsymbol{A}} \mathrm{tr}(\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x}) = \nabla_{\boldsymbol{A}} \mathrm{tr}(\boldsymbol{A}\boldsymbol{x}\boldsymbol{x}^\top) = (\boldsymbol{x}\boldsymbol{x}^\top)^\top = \boldsymbol{x}\boldsymbol{x}^\top$, using properties 11 and 23.
31. $\nabla_{\boldsymbol{x}} \boldsymbol{x}^\top \boldsymbol{y} = \nabla_{\boldsymbol{x}} \boldsymbol{y}^\top \boldsymbol{x} = \boldsymbol{y}$.
32. $\nabla_{\boldsymbol{A}} |\boldsymbol{A}| = |\boldsymbol{A}| (\boldsymbol{A}^{-1})^\top$.

IX. Matrix norm

Definition 3.3.1. Given any $\boldsymbol{A} \in \mathbb{R}^{D\times D}$, the norm of $\boldsymbol{A}$ is defined as:

$$\|\boldsymbol{A}\| = \max \left\{ \frac{\|\boldsymbol{A}\boldsymbol{x}\|}{\|\boldsymbol{x}\|} : \boldsymbol{x} \in \mathbb{R}^D, \boldsymbol{x} \neq \boldsymbol{0} \right\} .$$

With this definition, we have the following lemma:

Lemma 3.3.2. Suppose that $\boldsymbol{A} \in \mathbb{R}^{D\times D}$ and $\boldsymbol{x} \in \mathbb{R}^D$, we have

$$\|\boldsymbol{A}\boldsymbol{x}\| \leq \|\boldsymbol{A}\| \|\boldsymbol{x}\| \,.$$

Proof.

$$\|\boldsymbol{A}\| \|\boldsymbol{x}\| = \max \left\{ \frac{\|\boldsymbol{A}\boldsymbol{y}\|}{\|\boldsymbol{y}\|} \right\} \|\boldsymbol{x}\| \geq \frac{\|\boldsymbol{A}\boldsymbol{y}\|}{\|\boldsymbol{y}\|} \|\boldsymbol{x}\| \,.$$

Choosing $\boldsymbol{y} = \boldsymbol{x}$, we get back the desired inequality. □

3.3.1 Singular Value Decomposition (Optional)

Singular Value Decomposition (SVD) is a generalization of eigenvalue decomposition for a square symmetric matrix, but now applies to a general rectangular $m \times n$ matrix. It has wide practical applications in machine learning, for instance, in recommender systems commonly encountered in marketing sciences. SVD claims the following, also see the book *"Matrix Computations"* by Golub and Loan (2013): given a $M \times D$ matrix $\boldsymbol{A}$, there exist matrices $\boldsymbol{U}$, $\boldsymbol{\Sigma}$, and $\boldsymbol{V}$ such that

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top \qquad \text{or equivalently} \qquad \boldsymbol{A}\boldsymbol{V} = \boldsymbol{U}\boldsymbol{\Sigma} \,, \tag{3.3.1}$$

where

1. $\boldsymbol{U} = (\boldsymbol{u}_1, \cdots, \boldsymbol{u}_M)$ is an $M \times M$ matrix, and $\boldsymbol{u}_i$ a unit eigenvector corresponding to the i-th largest eigenvalue of $\boldsymbol{A}\boldsymbol{A}^\top$. The vectors $\boldsymbol{u}_i$'s are called the **left singular vectors** of $\boldsymbol{A}$, and they are orthogonal to each other;

2. $\boldsymbol{V} = (\boldsymbol{v}_1, \cdots, \boldsymbol{v}_D)$ is an $D \times D$ matrix, and $\boldsymbol{v}_j$ is a unit eigenvector corresponding to the j-th largest eigenvalue of $\boldsymbol{A}^\top \boldsymbol{A}$. The vectors $\boldsymbol{v}_j$'s are called the **right singular vectors** of $\boldsymbol{A}$, and they are orthogonal to each other;
3. $\Sigma = \begin{pmatrix} \mathrm{diag}(\sigma_1, \cdots, \sigma_R) & 0 \\ 0 & 0 \end{pmatrix}$ is a diagonal $M \times D$ matrix, where $R = \min\{M, D\} = \mathrm{rank}(\boldsymbol{A})$, and $\sigma_r = \sqrt{\lambda_r(\boldsymbol{A}^\top \boldsymbol{A})} > 0$ is the square root of the r-th largest eigenvalue of $\boldsymbol{A}^\top \boldsymbol{A}$, which is called a **singular value** of $\boldsymbol{A}$.

To establish this SVD representation (3.3.1), we first note that $\boldsymbol{A}\boldsymbol{A}^\top$ and $\boldsymbol{A}^\top \boldsymbol{A}$ share the same set of non-zero eigenvalues; indeed, for $\boldsymbol{A}\boldsymbol{A}^\top \boldsymbol{u}_i = \lambda_i \boldsymbol{u}_i$ and $\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{v}_j = \theta_j \boldsymbol{v}_j$, that is to say, $\boldsymbol{u}_i$ is an eigenvector for $\boldsymbol{A}\boldsymbol{A}^\top$ corresponding to the non-zero eigenvalue λ_i, similarly $\boldsymbol{v}_j$ an eigenvector for $\boldsymbol{A}^\top \boldsymbol{A}$ corresponds to the non-zero eigenvalue θ_j, then

$$\boldsymbol{A}^\top (\boldsymbol{A}\boldsymbol{A}^\top)\boldsymbol{u}_i = \boldsymbol{A}^\top (\lambda_i \boldsymbol{u}_i) \qquad \Rightarrow \qquad (\boldsymbol{A}^\top \boldsymbol{A})(\boldsymbol{A}^\top \boldsymbol{u}_i) = \lambda_i (\boldsymbol{A}^\top \boldsymbol{u}_i)\,,$$

which means that $\boldsymbol{A}^\top \boldsymbol{u}_i$ is an eigenvector for $\boldsymbol{A}^\top \boldsymbol{A}$ with respect to λ_i, which is therefore an eigenvalue of $\boldsymbol{A}^\top \boldsymbol{A}$. Also note that

$$\|\boldsymbol{A}^\top \boldsymbol{u}_i\|_2^2 = \boldsymbol{u}_i \boldsymbol{A}\boldsymbol{A}^\top \boldsymbol{u}_i = \lambda_i \|\boldsymbol{u}_i\|_2^2 > 0\,.$$

Similar argument also leads to the claim that $\boldsymbol{A}\boldsymbol{v}_j$ is an eigenvector for $\boldsymbol{A}\boldsymbol{A}^\top$ with respect to eigenvalue θ_j. Hence, $\boldsymbol{A}\boldsymbol{A}^\top$ and $\boldsymbol{A}^\top \boldsymbol{A}$ possess the same non-zero spectrum.

For $R = \mathrm{rank}(A)$, by the spectral decomposition of the square symmetric matrix $\boldsymbol{A}\boldsymbol{A}^\top$, there exist R positive constants $\sigma_1, \cdots, \sigma_R$, R orthogonal $M \times 1$ unit vectors $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_R$, and, based on the above discussion, R orthogonal $D \times 1$ unit vectors $\boldsymbol{v}_1 (= \boldsymbol{A}^\top \boldsymbol{u}_1), \cdots, \boldsymbol{v}_R (= \boldsymbol{A}^\top \boldsymbol{u}_R)$, such that

$$\boldsymbol{A}\boldsymbol{v}_1 = \sigma_1 \boldsymbol{u}_1\,, \cdots, \boldsymbol{A}\boldsymbol{v}_R = \sigma_R \boldsymbol{u}_R\,. \tag{3.3.2}$$

Also note that, for $i, j = 1, 2, \cdots, R$,

$$\langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle = \langle \boldsymbol{A}^\top \boldsymbol{u}_i, \boldsymbol{A}^\top \boldsymbol{u}_j \rangle = \boldsymbol{u}_i \boldsymbol{A}\boldsymbol{A}^\top \boldsymbol{u}_j = \sigma_j \boldsymbol{u}_i^\top \boldsymbol{u}_j = \sigma_j \langle \boldsymbol{u}_i, \boldsymbol{u}_j \rangle\,,$$

hence $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ are orthogonal whenever $\boldsymbol{u}_i$ and $\boldsymbol{u}_j$ are.

By defining $\boldsymbol{U}_R = (\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_R)$, $V_R = (\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_R)$, and $\boldsymbol{\Sigma}_R = \mathrm{diag}(\sigma_1, \sigma_2, \cdots, \sigma_R)$, we can rewrite (3.3.2) in a compact matrix form:

$$\boldsymbol{A}\boldsymbol{V}_R = \boldsymbol{U}_R \boldsymbol{\Sigma}_R\,. \tag{3.3.3}$$

Also note that the right singular vectors $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_R$ are in the row space of $\boldsymbol{A}$, while the left singular vectors $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_R$ are in the column space of $\boldsymbol{A}$.

To conclude the claim that $\boldsymbol{A}\boldsymbol{V} = \boldsymbol{U}\boldsymbol{\Sigma}$, we only have to pick $D - R$ more $\boldsymbol{v}_i$'s and $M - R$ more $\boldsymbol{u}_j$'s from the null spaces $\mathcal{N}(\boldsymbol{A})$ and $\mathcal{N}(\boldsymbol{A}^\top)$ respectively. Each set of these can be chosen to be orthonormal bases for those two null spaces; note that by then they would be respectively orthogonal to the first R $\boldsymbol{v}_i$'s and $\boldsymbol{u}_j$'s. For instance, for $\boldsymbol{v}_{R+1}, \cdots, \boldsymbol{v}_D$ from $\mathcal{N}(A)$, $i = R+1, \cdots, D$, and $j = 1, \cdots, R$, we have

$$\langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle = \boldsymbol{v}_i^\top (\boldsymbol{A}^\top \boldsymbol{u}_j) = (\boldsymbol{A}\boldsymbol{v}_i)^\top \boldsymbol{u}_j = 0\,,$$

by definition. We next include all the newly constructed $\boldsymbol{v}_i$'s and $\boldsymbol{u}_j$'s in $\boldsymbol{V}$ and $\boldsymbol{U}$, by letting $\boldsymbol{V} =$

$(V_R; \boldsymbol{v}_{R+1}, \cdots, \boldsymbol{v}_D)$ and $\boldsymbol{U} = (\boldsymbol{U}_R; \boldsymbol{u}_{R+1}, \cdots, \boldsymbol{u}_M)$, and so $\boldsymbol{V}$ and $\boldsymbol{U}$ are both square matrices. Also form the $M \times D$ matrix $\boldsymbol{\Sigma}$ by adding to $\boldsymbol{\Sigma}_R$ with $M - R$ new zero rows and $D - R$ new zero columns. With these definitions together with (3.3.3), we conclude that $\boldsymbol{AV} = \boldsymbol{U\Sigma}$. Finally, since $\boldsymbol{V}$ is orthogonal by construction, we further have $\boldsymbol{A} = \boldsymbol{U\Sigma V}^\top$. *Q.E.D.*

To summarize, $\boldsymbol{u}$'s and $\boldsymbol{v}$'s form the bases for the following four fundamental spaces:

1. $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_R$ form an orthonormal basis for the column space of $\boldsymbol{A}$;
2. $\boldsymbol{u}_{R+1}, \cdots, \boldsymbol{u}_M$ form an orthonormal basis for the null space $\mathcal{N}(\boldsymbol{A}^\top)$;
3. $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_R$ form an orthonormal basis for the row space of $\boldsymbol{A}$;
4. $\boldsymbol{v}_{R+1}, \cdots, \boldsymbol{v}_D$ form an orthonormal basis for the null space $\mathcal{N}(\boldsymbol{A})$.

To specify those singular values σ_i's, we consider

$$\boldsymbol{A}^\top \boldsymbol{A} = (\boldsymbol{U\Sigma V}^\top)^\top (\boldsymbol{U\Sigma V}^\top) = \boldsymbol{V\Sigma}^\top \boldsymbol{U}^\top \boldsymbol{U\Sigma V}^\top = \boldsymbol{V\Sigma}^\top \boldsymbol{\Sigma V}^\top ,$$

and hence $\boldsymbol{\Sigma}^\top \boldsymbol{\Sigma}$ is the diagonal matrix of eigenvalues of $\boldsymbol{A}^\top \boldsymbol{A}$, which means that σ_i^2 are non-zero eigenvalues of $\boldsymbol{A}^\top \boldsymbol{A}$ (and also $\boldsymbol{AA}^\top$ by the discussion above). Also recall that $\boldsymbol{v}_i$'s are the eigenvectors of $\boldsymbol{A}^\top \boldsymbol{A}$, while $\boldsymbol{u}_i$'s are the eigenvectors of $\boldsymbol{AA}^\top$. Finally, we can also write (3.3.1) in a "reduced SVD" form as:

$$\boldsymbol{A} = \sum_{i=1}^{R} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^\top = \boldsymbol{U}_R \boldsymbol{\Sigma}_R \boldsymbol{V}_R^\top .$$

In practice for most applications, we do not always need the exact SVD of the data matrix $\boldsymbol{A}$, and it often suffices to have a factorization which well approximates $\boldsymbol{A}$ so as to save storage space and reduce future computational complexity. In the matrix $\boldsymbol{\Sigma}$, the singular values can be arranged in descending order. Quite often the singular values decrease at a relatively fast rate, and the sum of the first 10% or even 1% of the singular values (say K out of R, with $K \ll R$) already amounts to 99% of the total sum of all the singular values; also note that both $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$ are of unit norm. Therefore, it is plausible to describe the matrix $\boldsymbol{A}$ by using only the first K singular values via the decomposition:

$$\boldsymbol{A} \approx \boldsymbol{U}_K \boldsymbol{\Sigma}_K \boldsymbol{V}_K^\top ,$$

where $\boldsymbol{U}_K \in \mathbb{R}^{M \times K}$, $\boldsymbol{\Sigma}_K \in \mathbb{R}^{K \times K}$ and $\boldsymbol{V}_K \in \mathbb{R}^{D \times K}$, here $\boldsymbol{U}_K$ and $\boldsymbol{V}_K$ contain the corresponding first K left and right singular vectors. Usually, we require $K \ll R = \min\{M, D\}$. Hence, $M \times K + K \times K + K \times D = (M + D + K) \times K \ll M \times D$,[1] in other words, we can recover $\boldsymbol{A}$ with much smaller in size matrices $\boldsymbol{U}_K, \boldsymbol{V}_K$ and $\boldsymbol{\Sigma}_K$.

[1] Note that K is often chosen so that the product $(M + D + K) \times K$ is about the same as the number of entries in the large and sparse data matrix $\boldsymbol{A}$.

3.4 Principal Component Analysis (Optional)

Principal Component Analysis (PCA for short) is a classical multivariate statistical technique for a dimension reduction of an underlying set of random variables. It is used to find linear combinations of original variables such that the information in the original data is "essentially" preserved even after the reduction. Next, we shall look at an interesting application of PCA to identify the important components explaining the term structure of interest rates extracted from the USA market.

3.4.1 Analysis of US Zero-Coupon Rates

The file "`us-zeros.csv`" contains US treasury zero-coupon rates, measured in basis points (b.p. for short, and 100bps = 1%), with maturities of 3M (Months) to 30Y (Years), monthly data over the period of March, 1989 to December, 2021. As introduced in Chapter 2, the following **R** commands will read in the data, then apply labels and compute the correlation matrix of these rates, see Programme 3.4.1.

```
> d <- read.csv("us-zeros.csv", row.names=1, check.names=FALSE)  # Read in data
>
> options(digits=2)          # Display the number of 2 digits
> cor(d)                     # Compute correlation matrix
      3M   6M   1Y   2Y   3Y   4Y   5Y   6Y   7Y   8Y   9Y  10Y  15Y  20Y  30Y
3M  1.00 1.00 0.99 0.98 0.97 0.95 0.94 0.92 0.91 0.90 0.88 0.87 0.86 0.84 0.76
6M  1.00 1.00 1.00 0.99 0.98 0.96 0.94 0.93 0.91 0.90 0.89 0.88 0.87 0.84 0.77
1Y  0.99 1.00 1.00 0.99 0.98 0.97 0.96 0.94 0.93 0.92 0.91 0.89 0.88 0.86 0.79
2Y  0.98 0.99 0.99 1.00 1.00 0.99 0.98 0.97 0.96 0.95 0.94 0.93 0.92 0.90 0.83
3Y  0.97 0.98 0.98 1.00 1.00 1.00 0.99 0.98 0.98 0.97 0.96 0.95 0.94 0.93 0.86
4Y  0.95 0.96 0.97 0.99 1.00 1.00 1.00 0.99 0.99 0.98 0.98 0.97 0.96 0.95 0.89
5Y  0.94 0.94 0.96 0.98 0.99 1.00 1.00 1.00 1.00 0.99 0.99 0.98 0.98 0.97 0.92
6Y  0.92 0.93 0.94 0.97 0.98 0.99 1.00 1.00 1.00 1.00 0.99 0.99 0.99 0.98 0.93
7Y  0.91 0.91 0.93 0.96 0.98 0.99 1.00 1.00 1.00 1.00 1.00 0.99 0.99 0.98 0.95
8Y  0.90 0.90 0.92 0.95 0.97 0.98 0.99 1.00 1.00 1.00 1.00 1.00 1.00 0.99 0.96
9Y  0.88 0.89 0.91 0.94 0.96 0.98 0.99 0.99 1.00 1.00 1.00 1.00 1.00 0.99 0.97
10Y 0.87 0.88 0.89 0.93 0.95 0.97 0.98 0.99 0.99 1.00 1.00 1.00 1.00 1.00 0.97
15Y 0.86 0.87 0.88 0.92 0.94 0.96 0.98 0.99 0.99 1.00 1.00 1.00 1.00 1.00 0.98
20Y 0.84 0.84 0.86 0.90 0.93 0.95 0.97 0.98 0.98 0.99 0.99 1.00 1.00 1.00 0.99
30Y 0.76 0.77 0.79 0.83 0.86 0.89 0.92 0.93 0.95 0.96 0.97 0.97 0.98 0.99 1.00
```

Programme 3.4.1: Correlation matrix of zero-coupon rates in **R**.

Here each column name begins with a number, which is invalid in the **R**'s `data.frame` object, the `read.csv()` function will automatically add `X` in the beginning of each column names. In order to use the original column names, we can pass the additional argument `check.names=FALSE`. However, we will not be able to retrieve the column by its name, for instance, we cannot call `d$3M` for the correlation between the three month rate and other rates.

Similarly, in Python:

```
import numpy as np
import pandas as pd

d = pd.read_csv("us-zeros.csv", index_col=0)  # Read in data
print(d.columns.values)
np.set_printoptions(precision=2)    # Display the number of 2 digits
print(np.corrcoef(d, rowvar=False)) # Compute correlation matrix
```

```
['3M' '6M' '1Y' '2Y' '3Y' '4Y' '5Y' '6Y' '7Y' '8Y' '9Y' '10Y' '15Y' '20Y' '30Y']
[[1.   1.   0.99 0.98 0.97 0.95 0.94 0.92 0.91 0.9  0.88 0.87 0.86 0.84 0.76]
 [1.   1.   1.   0.99 0.98 0.96 0.94 0.93 0.91 0.9  0.89 0.88 0.87 0.84 0.77]
 [0.99 1.   1.   0.99 0.98 0.97 0.96 0.94 0.93 0.92 0.91 0.89 0.88 0.86 0.79]
 [0.98 0.99 0.99 1.   1.   0.99 0.98 0.97 0.96 0.95 0.94 0.93 0.92 0.9  0.83]
 [0.97 0.98 0.98 1.   1.   1.   0.99 0.98 0.98 0.97 0.96 0.95 0.94 0.93 0.86]
 [0.95 0.96 0.97 0.99 1.   1.   1.   0.99 0.99 0.98 0.98 0.97 0.96 0.95 0.89]
 [0.94 0.94 0.96 0.98 0.99 1.   1.   1.   1.   0.99 0.99 0.98 0.98 0.97 0.92]
 [0.92 0.93 0.94 0.97 0.98 0.99 1.   1.   1.   1.   0.99 0.99 0.99 0.98 0.93]
 [0.91 0.91 0.93 0.96 0.98 0.99 1.   1.   1.   1.   1.   0.99 0.99 0.98 0.95]
 [0.9  0.9  0.92 0.95 0.97 0.98 0.99 1.   1.   1.   1.   1.   1.   0.99 0.96]
 [0.88 0.89 0.91 0.94 0.96 0.98 0.99 0.99 1.   1.   1.   1.   1.   0.99 0.97]
 [0.87 0.88 0.89 0.93 0.95 0.97 0.98 0.99 0.99 1.   1.   1.   1.   1.   0.97]
 [0.86 0.87 0.88 0.92 0.94 0.96 0.98 0.99 0.99 1.   1.   1.   1.   1.   0.98]
 [0.84 0.84 0.86 0.9  0.93 0.95 0.97 0.98 0.98 0.99 0.99 1.   1.   1.   0.99]
 [0.76 0.77 0.79 0.83 0.86 0.89 0.92 0.93 0.95 0.96 0.97 0.97 0.98 0.99 1.  ]]
```

Programme 3.4.2: Correlation matrix of zero-coupon rates in Python.

Here, the argument `index_col=0` indicates that the first column in this dataset is used as the column index.

First note that these $D = 15$ feature variables are highly correlated. This means that very likely we can use only a few variables (say 2 or 3) to represent this dataset without causing the loss of a large amount of information. This is exactly the goal of dimension or feature variable reduction! A natural question to ask is which variables to use so that they can contain the information in the dataset as much as possible? The answer is not from any of these original variables but a linear combination of them; indeed, let the original feature variables be $x^{(1)}, \cdots, x^{(D)}$, then we are looking for a new variable

$$y^{(1)} = \alpha_{11}x^{(1)} + \cdots + \alpha_{1D}x^{(D)} = \boldsymbol{\alpha}_1^\top \boldsymbol{x}\,, \tag{3.4.1}$$

where $\boldsymbol{\alpha}_1 = (\alpha_{11}, \cdots, \alpha_{1D})^\top$ and $\boldsymbol{x} = (x^{(1)}, \cdots, x^{(D)})^\top$ such that the variance of $y^{(1)}$,

$$\mathrm{Var}(y^{(1)}) = \boldsymbol{\alpha}_1^\top \mathrm{Var}(\boldsymbol{x})\boldsymbol{\alpha}_1 = \boldsymbol{\alpha}_1^\top \boldsymbol{S}\boldsymbol{\alpha}_1\,, \tag{3.4.2}$$

is maximum subject to the constraint:

$$\alpha_{11}^2 + \cdots + \alpha_{1D}^2 = \boldsymbol{\alpha}_1^\top \boldsymbol{\alpha}_1 = 1\,, \tag{3.4.3}$$

where $\boldsymbol{S} \in \mathbb{R}^{D\times D}$ is the sample covariance matrix (or correlation matrix, depending on whether the data $\boldsymbol{X} \in \mathbb{R}^{N\times D}$ are standardized or not) of $\boldsymbol{X}$. This is called the first *principal component* (PC) and the $D \times 1$ vector $\boldsymbol{\alpha}_1$ is called the *loadings* of this first PC. We can continue to find the second principal component

such that

$$y^{(2)} = \alpha_{21}x^{(1)} + \cdots + \alpha_{2D}x^{(D)} = \boldsymbol{\alpha}_2^\top \boldsymbol{x}\,, \tag{3.4.4}$$

where $\boldsymbol{\alpha}_2 = (\alpha_{21}, \cdots, \alpha_{2D})^\top$ such that the variance of $y^{(2)}$,

$$\text{Var}(y^{(2)}) = \boldsymbol{\alpha}_2^\top \text{Var}(\boldsymbol{x})\boldsymbol{\alpha}_2 = \boldsymbol{\alpha}_2^\top \boldsymbol{S}\boldsymbol{\alpha}_2 \tag{3.4.5}$$

is maximum subject to

$$\alpha_{21}^2 + \cdots + \alpha_{2D}^2 = \boldsymbol{\alpha}_2^\top \boldsymbol{\alpha}_2 = 1\,, \tag{3.4.6}$$

and $\boldsymbol{\alpha}_2^\top \boldsymbol{\alpha}_1 = 0$, which means that $\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_2$ are **orthogonal** to each other (or equivalently that $y^{(1)}$ and $y^{(2)}$ are statistically uncorrelated)[2].

The third PC can be found similarly, and the process can continue up to the last D-th PC. This gives the basic idea of PCA. This optimization procedure has a nice closed form solution; in particular, it turns out that these loadings are the **normalized eigenvectors**, all denoted by $\boldsymbol{\alpha}_i = (\alpha_{i1}, \cdots, \alpha_{iD})^\top$, for $i = 1, 2, \cdots, D$, for the i-th PC, of the correlation matrix or the variance-covariance matrix of $\boldsymbol{X}$ and the variance of $y^{(i)}$, the i-th PC, is the corresponding **eigenvalue** $\lambda_i = \text{Var}(y^{(i)}) = \boldsymbol{\alpha}_i^\top \boldsymbol{S}\boldsymbol{\alpha}_i$; this claim together with the basic theory of spectral decomposition, see Property 24 in Section 3.3 and Subsection 3.4.3.

The reason for maximizing the variance of the transformed variable $y^{(i)}$, for $i = 1, \cdots, D$, is that we want to preserve as much information in these $\boldsymbol{x}$'s as possible. Conceptually, variation represents information, and so we want the variances of the linearly transformed variables to be as large as possible, also as "soon as possible", so that the original information in the $\boldsymbol{x}$'s is preserved. The i-th eigenvalue, which is equal to the variance of $y^{(i)}$, indicates the amount of information retained in the i-th PC.

Next, we shall use the self-defined `pr_comp()` function, and the two **R** build-in functions, `prcomp()` and `princomp()`, to compute the loadings and the variances $y^{(i)}$ for $i = 1, \cdots, D$ for each PC component.

```
> df <- as.matrix(df)
> pca_prin <- princomp(df, cor=TRUE)             # spectral decomposition
> pca_pr <- prcomp(df, center=TRUE, scale=TRUE) # singular value decomposition
>
> # pcs: pca_prin$loadings, pca_pr$rotation
> # std: pca_prin$sdev, pca_pr$sdev
>
> pca_prin$loadings[,1:8]                    # display the loadings
     Comp.1    Comp.2    Comp.3   Comp.4   Comp.5   Comp.6   Comp.7   Comp.8
3M   0.2485  0.407201  0.466711  0.38296  0.39335  0.35921  0.08420  0.17272
6M   0.2501  0.396155  0.313433  0.09466 -0.12570 -0.28154 -0.09497 -0.18023
1Y   0.2531  0.356826  0.112036 -0.22486 -0.47878 -0.29980 -0.31401 -0.10317
2Y   0.2588  0.247132 -0.155417 -0.40889 -0.03271  0.06221  0.62515  0.04712
3Y   0.2616  0.166858 -0.249747 -0.29192  0.05645  0.16101  0.15912  0.11209
```

[2] To see this, first note that, $\text{Cov}(y^{(i)}, y^{(j)}) = \boldsymbol{\alpha}_i^\top \boldsymbol{S}\boldsymbol{\alpha}_j$, for $i, j = 1, 2, \cdots, D$; but each $\boldsymbol{\alpha}_k$ is an eigenvector of $\boldsymbol{S}$ with respect to the eigenvalue λ_k, therefore $\text{Cov}(y^{(i)}, y^{(j)}) = \boldsymbol{\alpha}_i^\top (\lambda_j \boldsymbol{\alpha}_j) = \lambda_j \left(\boldsymbol{\alpha}_i^\top \boldsymbol{\alpha}_j\right) = 0$ for $i \neq j$.

```
4Y   0.2633   0.089947 -0.275420 -0.16378   0.11723   0.23076 -0.21222   0.06469
5Y   0.2639   0.005679 -0.280808 -0.05357   0.20520   0.12232 -0.55638   0.19452
6Y   0.2639 -0.045453 -0.238891   0.09608   0.23352 -0.06261 -0.17722 -0.22476
7Y   0.2632 -0.100959 -0.185330   0.23696   0.25584 -0.32780   0.18751 -0.41607
8Y   0.2626 -0.136605 -0.127284   0.23915   0.06941 -0.23825   0.15352 -0.10963
9Y   0.2617 -0.175315 -0.063486   0.23570 -0.11767 -0.17830   0.11228   0.28914
10Y 0.2605 -0.210414 -0.001026   0.22782 -0.29715 -0.13392   0.06418   0.62500
15Y 0.2593 -0.240893   0.062286   0.13950 -0.32870   0.32917   0.05759 -0.22262
20Y 0.2566 -0.295573   0.175402 -0.01919 -0.29222   0.47664 -0.03162 -0.32422
30Y 0.2448 -0.448957   0.532782 -0.51172   0.34606 -0.22453 -0.05824   0.07539
> pca_prin$sdev
    Comp.1     Comp.2     Comp.3     Comp.4     Comp.5     Comp.6     Comp.7
3.7825198 0.8034075 0.1929941 0.0709158 0.0452586 0.0340429 0.0273628
    Comp.8     Comp.9    Comp.10    Comp.11    Comp.12    Comp.13    Comp.14
0.0196466 0.0158314 0.0130274 0.0047471 0.0032400 0.0027288 0.0013456
   Comp.15
0.0003502
```

Programme 3.4.3: The first eight principal components' loadings for the US rates using `princomp()` and `prcomp` functions in **R**.

In retreiving the principle components from `pca_prin` (resp. `pca_pr`), we can use `pca_prin$loadings` (resp. `pca_pr$rotation`); meanwhile for retreiving the explained standard deviation, we can use `pca_prin$sdev` (resp. `pca_pr$sdev`). Moreover, if `scale=TRUE`, we are using the correlation matrix instead of the covariance matrix for the spectual decomposition due to the definition of correlation matrix and covariance matrix. Therefore, the two functions will return the same results, but `princomp()` only accepts `cor` (CORrelation) as the parameter without the `center` and `scale` parameters. Note that the two loadings computed by `princomp()` and `prcomp()` functions may be different, in a way that there might exits some PC columns differ by the multiplication with -1. It is because in spectral decomposition, the eigenvectors $\boldsymbol{\alpha}_i$ and $-\boldsymbol{\alpha}_i$ lead to the same variance $\text{Var}(y^{(i)})$, *i.e.* $\boldsymbol{\alpha}_i^\top \boldsymbol{S}\boldsymbol{\alpha}_i = \text{Var}(y^{(i)}) = (-\boldsymbol{\alpha}_i)^\top \boldsymbol{S}(-\boldsymbol{\alpha}_i)$, but these differences will not affect our PCA.

Similarly, in Python, we shall use the `PCA()` function provided in `sklearn.decomposition` to perform PCA:

```
from sklearn.decomposition import PCA

np.set_printoptions(precision=4)    # Display the number of 4 digits
pca = PCA()
standard_d = (d - np.mean(d, axis=0))/np.std(d, axis=0)
pca.fit(standard_d)
# components_: an ndarray of shape (n_components, n_features)
print(pca.components_[:8,:].T)
print(np.sqrt(pca.explained_variance_))
```

```
[[ 0.2485 -0.4072 -0.4667 -0.383  -0.3933 -0.3592  0.0842 -0.1727]
 [ 0.2501 -0.3962 -0.3134 -0.0947  0.1257  0.2815 -0.095   0.1802]
 [ 0.2531 -0.3568 -0.112   0.2249  0.4788  0.2998 -0.314   0.1032]
```

```
 [ 0.2588 -0.2471  0.1554  0.4089  0.0327 -0.0622  0.6252 -0.0471]
 [ 0.2616 -0.1669  0.2497  0.2919 -0.0564 -0.161   0.1591 -0.1121]
 [ 0.2633 -0.0899  0.2754  0.1638 -0.1172 -0.2308 -0.2122 -0.0647]
 [ 0.2639 -0.0057  0.2808  0.0536 -0.2052 -0.1223 -0.5564 -0.1945]
 [ 0.2639  0.0455  0.2389 -0.0961 -0.2335  0.0626 -0.1772  0.2248]
 [ 0.2632  0.101   0.1853 -0.237  -0.2558  0.3278  0.1875  0.4161]
 [ 0.2626  0.1366  0.1273 -0.2391 -0.0694  0.2383  0.1535  0.1096]
 [ 0.2617  0.1753  0.0635 -0.2357  0.1177  0.1783  0.1123 -0.2891]
 [ 0.2605  0.2104  0.001  -0.2278  0.2972  0.1339  0.0642 -0.625 ]
 [ 0.2593  0.2409 -0.0623 -0.1395  0.3287 -0.3292  0.0576  0.2226]
 [ 0.2566  0.2956 -0.1754  0.0192  0.2922 -0.4766 -0.0316  0.3242]
 [ 0.2448  0.449  -0.5328  0.5117 -0.3461  0.2245 -0.0582 -0.0754]]
[3.7873e+00 8.0443e-01 1.9324e-01 7.1006e-02 4.5316e-02 3.4086e-02 2.7398e-02
1.9672e-02 1.5852e-02 1.3044e-02 4.7531e-03 3.2442e-03 2.7323e-03 1.3473e-03
3.5062e-04]
```

Programme 3.4.4: The first eight principal components' loadings for the US rates using `PCA()` function in Python.

Here, the loadings are stored as a numpy array of shape number of components times number of features, therefore, the first 8 loadings are the transpose of the first 8 rows of `pca.components_`.

Finally, a plot of variance (called Scree Plot, "scree" means large loose broken "gems") can help us determine the suitable number of PCs to be used. This plot represents the information retained in each PC graphically; see Figure 3.4.1. For most applications, we can look at the scree plot to determine the number of PCs to be used. In the calculation of the variance, we use the correlation matrix instead of covariance matrix, therefore all the diagonal elements are equal to 1, and hence the sum, the total variance `t`, is equal to $D = 15$.

```
> s <- pca_prin$sdev   # Save the s.d. of all PC's to s
> round(s^2, 4)      # Display the variances of all PC"s
 Comp.1  Comp.2  Comp.3  Comp.4  Comp.5  Comp.6  Comp.7  Comp.8
14.3075  0.6455  0.0372  0.0050  0.0020  0.0012  0.0007  0.0004
 Comp.9 Comp.10 Comp.11 Comp.12 Comp.13 Comp.14 Comp.15
 0.0003  0.0002  0.0000  0.0000  0.0000  0.0000  0.0000
> (t <- sum(s^2))   # Compute total variance (should equal 13)
[1] 15
> round(s^2/t, 4)   # Proportion of variance explained by PC's
 Comp.1  Comp.2  Comp.3  Comp.4  Comp.5  Comp.6  Comp.7  Comp.8
 0.9538  0.0430  0.0025  0.0003  0.0001  0.0001  0.0000  0.0000
 Comp.9 Comp.10 Comp.11 Comp.12 Comp.13 Comp.14 Comp.15
 0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
> cumsum(s^2/t)      # Cumulative sum of proportion of variance
 Comp.1  Comp.2  Comp.3  Comp.4  Comp.5  Comp.6  Comp.7  Comp.8
 0.9538  0.9969  0.9993  0.9997  0.9998  0.9999  0.9999  1.0000
 Comp.9 Comp.10 Comp.11 Comp.12 Comp.13 Comp.14 Comp.15
```

```
 1.0000   1.0000   1.0000   1.0000   1.0000   1.0000   1.0000
> screeplot(pca_self, type="lines", main="Scree Plot")
```

Programme 3.4.5: Variances of different principal components of the US rates example in **R**.

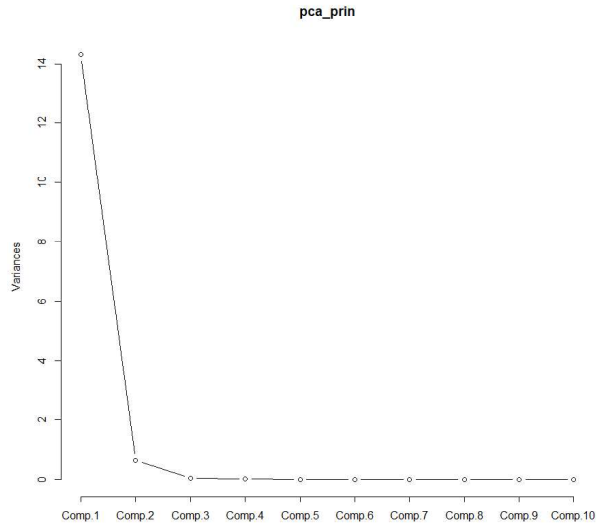
Similarly, in Python:

```
s2 = pca.explained_variance_    # save the variance of all PC to s2
print(np.round(s2, 4))          # Display the variances of all PC's
print(sum(s2))
# Proportion of variance explained by PC's
print(np.round(pca.explained_variance_ratio_, 4))
# Cumulative sum of proportion of variance
print(np.cumsum(pca.explained_variance_ratio_))

import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = 10, 10
plt.rcParams.update({'font.size': 15})

PC_comp = np.arange(pca.n_components_) + 1
plt.plot(PC_comp, pca.explained_variance_ratio_ * sum(s2),
         "bo-", linewidth=2)
plt.title('Scree Plot')
plt.xlabel('Principal Component')
plt.ylabel('Variance')
```

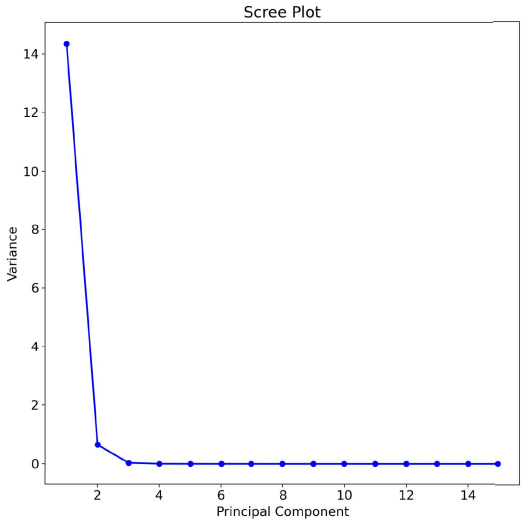
```
[1.4344e+01 6.4710e-01 3.7300e-02 5.0000e-03 2.1000e-03 1.2000e-03 8.0000e-04
4.0000e-04 3.0000e-04 2.0000e-04 0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00]
15.03816793893129
[9.538e-01 4.300e-02 2.500e-03 3.000e-04 1.000e-04 1.000e-04 0.000e+00 0.000e+00
0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00]
[0.9538 0.9969 0.9993 0.9997 0.9998 0.9999 0.9999 1.     1.     1.     1.
 1.     1.     1.     1.    ]
```

Programme 3.4.6: Variances of different principal components of the US rates example in Python.

Here, we set the width and height of the figure, as well as the font size of the label with `plt.rcParams`.



R: Generated in Programme 3.4.5.



Python: Generated in Programme 3.4.6.

Figure 3.4.1: Scree plots for the US rates example.

3.4.2 Financial Interpretation of PCs Obtained in Subsection 3.4.1

Let us plot the first three PCs in **R** with Programme 3.4.7 and in Python with Programme 3.4.8.

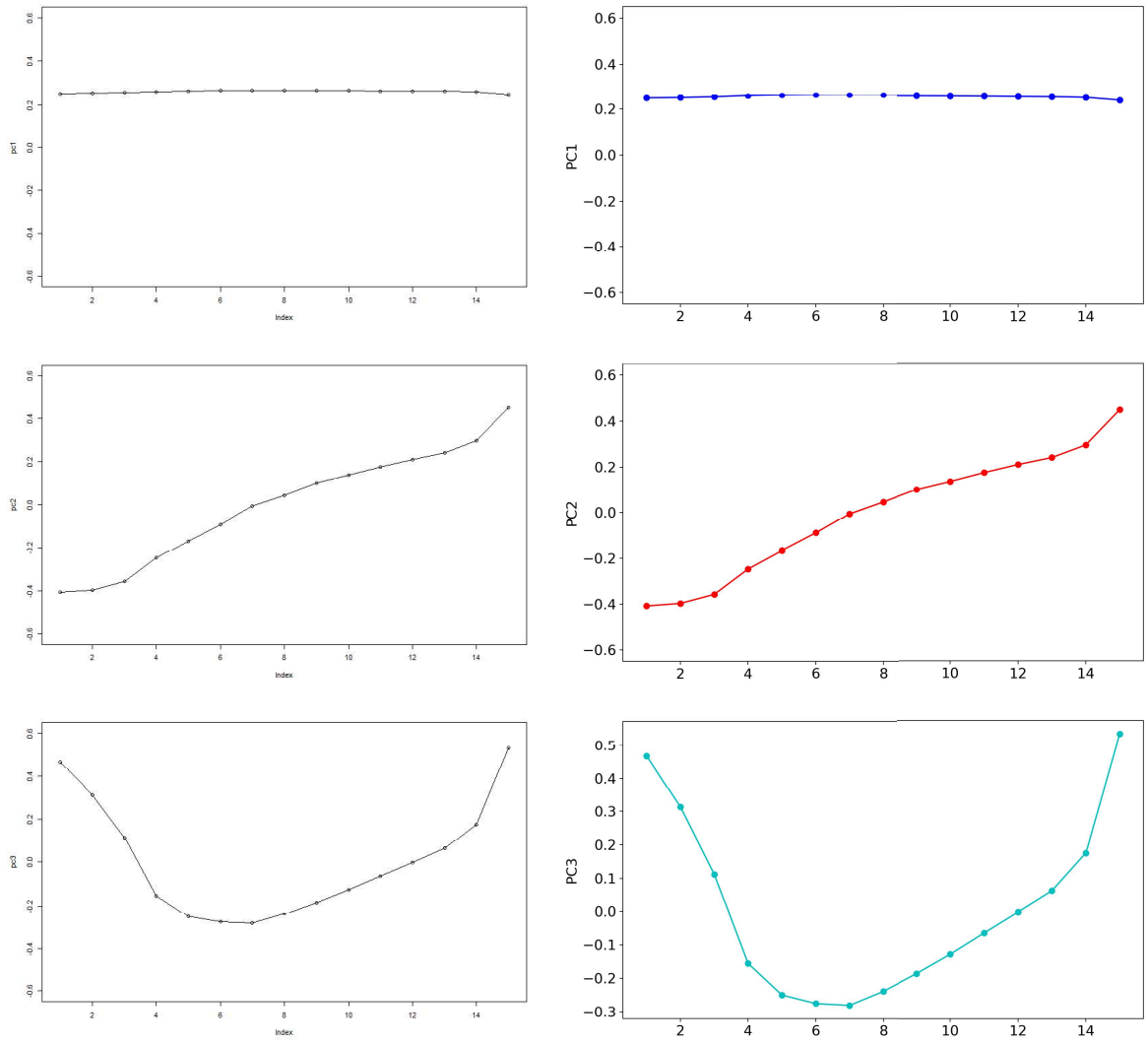
```
> pc1 <- pca_prin$loadings[,1]    # Save the loading of 1st PC
> pc2 <- -pca_prin$loadings[,2]   # Save the loading of 2nd PC
> pc3 <- pca_prin$loadings[,3]    # Save the loading of 3rd PC
>
> par(mfrow=c(3,1))           # Multi-frame for plotting
> plot(pc1, ylim=c(-0.6, 0.6), type="o")
> plot(pc2, ylim=c(-0.6, 0.6), type="o")
> plot(pc3, ylim=c(-0.6, 0.6), type="o")
```

Programme 3.4.7: **R** code for generating and plotting loadings of the first three principal components of the US rates example. Note that `pc2` is the negative of their corresponding original PC values.

```
pc1 = pca.components_[0,:]     # Save the loading of 1st PC
pc2 = pca.components_[1,:]    # Save the loading of 2nd PC
pc3 = -pca.components_[2,:]    # Save the loading of 3rd PC

# Multi-frame for plotting
fig, axs = plt.subplots(3, 1, figsize=(10,15))
axs[0].plot(PC_comp, pc1, "bo-")     # blue-circle-line
axs[0].set_ylim(-0.65, 0.65)
axs[0].set_ylabel("PC1")
axs[1].plot(PC_comp, pc2, "ro-")     # red-circle-line
axs[1].set_ylim(-0.65, 0.65)
axs[1].set_ylabel("PC2")
axs[2].plot(PC_comp, pc3, "co-")     # cyan-circle-line
axs[2].set_ylim(-0.65, 0.65)
```

```
axs[2].set_ylabel("PC3")

plt.savefig("PCA first 3 components.png", dpi=200)
```

Programme 3.4.8: Python code for generating and plotting loadings of the first three principal components of the US rates example. Note that `pc2` and `pc3` are the negative of their corresponding original PC values.



R: Generated in Programme 3.4.7.

Python: Generated in Programme 3.4.8.

Figure 3.4.2: Loadings of h_{ij}'s for the first three principal components of the US rates example.

Identifying these components is important in understanding the term structure of interest rate. Many investment instruments are very sensitive to the movement of interest rates, such as stocks, bonds, forward contracts, real estate and currency swaps. Knowledge on these components is very useful in scientifically valuating these instruments.

Recall that the first PC is

$$y_1 = 0.2485x_1 + \cdots + 0.2448x_{15}\,.$$

The loadings are almost constant at the level of 0.245 for bonds of all maturity times. In fact, these loadings are not unique. They are unique only up to a positive or negative sign. That is, if we define a new first PC as $y_1 = 0.2485x_1 + \cdots + 0.2448x_{15}$, this alternative PC has the same variance as the old one. This PC is interpreted as a **parallel shift** (caused by a "level" shock for example) of the yield curve. It also explains a majority of the variance that exists in the dataset. In other words, the US zero-coupon rates exhibit a parallel shift of term structure of interest rates of different maturities.

To explain this parallel shift, we refer to a shift in the economic environment caused by some major macroeconomic event, so that the rates of returns for all zero-coupon bonds with different maturities react in the same direction, and even move by essentially same amount of basis points. For instance, if a `3M` treasury bill increases 100bps, then all `6M`, `1Y`, `5Y`, `10Y` and `15Y` rates will increase by a similar amount as well; also see more information in NASDAQ (`http://www.nasdaq.com`).

The second PC is

$$y_2 = 0.4072x_1 + \cdots - 0.4489x_{15}\,,$$

where the loadings are decreasing with maturity. Again, the loadings are not unique. In particular, the alternative second PC could be $y_2 = -0.4072x_1 + \cdots + 0.4489x_{15}$ which seems more reasonable since it agrees with the **liquidity preference theory**. This theory states that investors prefer to preserve their liquidity and invest in investment vehicles that last for a short period of time, and so additional premiums are paid for holding securities with longer time of maturity. This component is termed as the **tilt** component of the yield curve.

The third PC is

$$y_3 = 0.4667x_1 + \cdots + 0.5328x_{15}\,.$$

The loadings decrease with maturity up to a certain point and then start increasing again with maturity. This can be interpreted as the **curvature** of the yield curve. By considering the negative counterpart of y_3, this effect is mainly caused by the higher demand on the very short-term and very long-term bonds; indeed, the borrowing market with short term bonds is popular with frequent traders in securities and derivatives markets; while for bonds of longer terms, they are welcomed by long term market participants, such as insurance and reinsurance companies, short term and long term bonds can be used for immunization of interest rate risk for investment vehicles of intermediate lifespans.

The original data can be expressed approximately in terms of y_1, y_2 and y_3 by using the loadings of PC1, PC2 and PC3 in light of the fact that $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} = \boldsymbol{H}^\top \boldsymbol{x}$. These y_i's are also called the principal component scores with respect to PC1, PC2 and PC3, respectively. The geometric interpretation of PCA may be illustrated by the scatter plot matrix of these scores, see Figures 3.4.3 for **R** and 3.4.4 for Python.

```
> score <- pca$scores[,1:3]      # save scores of PC1-PC3
> pairs(score)                   # scatterplot of scores
```

Programme 3.4.9: (a) **R** code for scatter plot for the first three principal components.

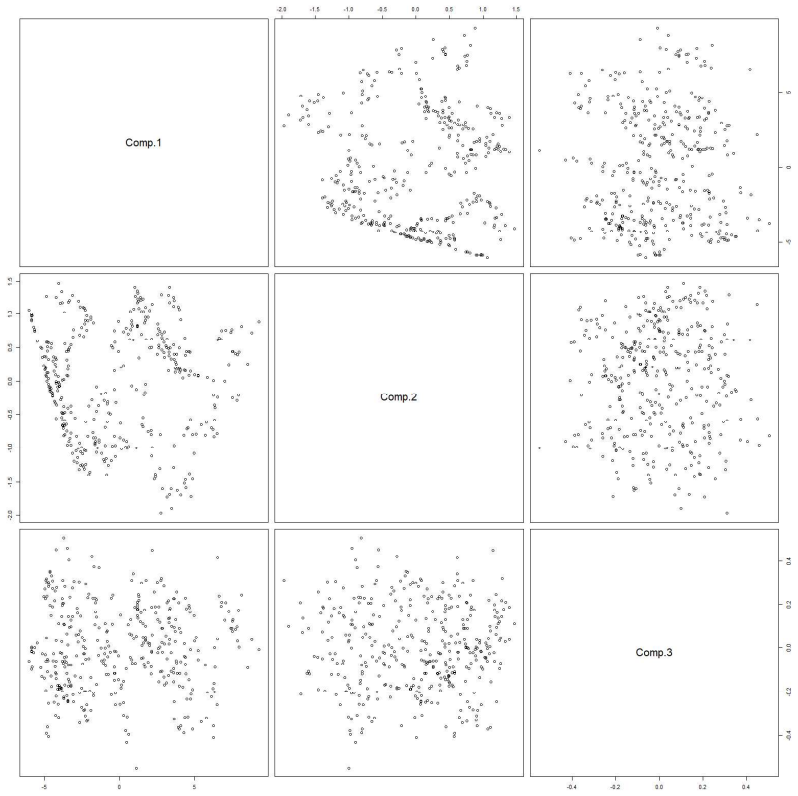


Figure 3.4.3: Scatter plot for the first three principal components from the US rates example in **R**. We expect to see that these components are uncorrelated, generated in Programme 3.4.9.

```python
from seaborn import pairplot

score = standard_d @ pca.components_.T  # save scores of all PC's
score.columns = [f"PC{i}" for i in PC_comp]
print(score.iloc[:,:3])
plots = pairplot(score.iloc[:,:3], diag_kind="hist", height=5)
plots.savefig("PCA first 3 scores pairplot.png", dpi=200)
```

Programme 3.4.10: (a) Python code for scatter plot for the first three principal components.

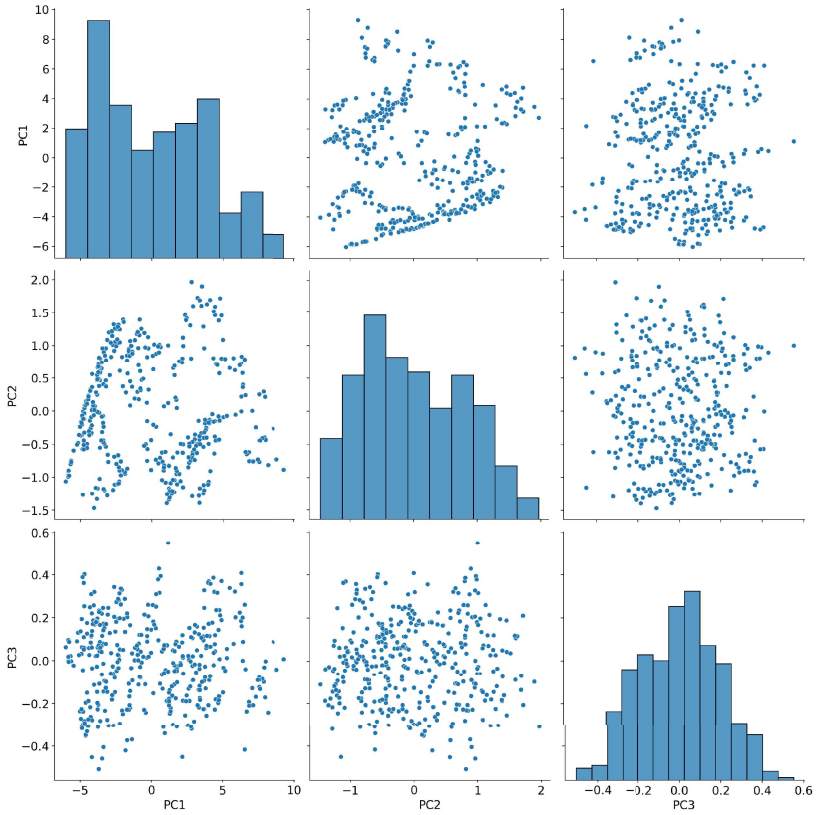


Figure 3.4.4: Scatter plot for the first three principal components from the US rates example in Python. We expect to see that these components are uncorrelated, generated in Programme 3.4.10.

Each vector of loadings of different principal component is the direction that the corresponding sample variance of the transformed variables y_1, y_2 or y_3 is maximized. Again, recall that the loadings of i-th PC are listed as

$$y_i = h_{i1}x_1 + \cdots + h_{iD}x_D = \boldsymbol{h}_i^\top \boldsymbol{x}\,,$$

where $\boldsymbol{h}_i$ is the i-th unit eigenvector of $\boldsymbol{S}$ corresponding to the i-th largest eigenvalue. Form the $D \times D$ matrix $\boldsymbol{H} = (h_{ij})_{D\times D} = (\boldsymbol{h}_1, \boldsymbol{h}_2, \cdots, \boldsymbol{h}_D)$ such that

$$\boldsymbol{H}^\top = (h_{ji}) = \begin{pmatrix} \boldsymbol{h}_1 & \boldsymbol{h}_2 & \cdots & \boldsymbol{h}_D \end{pmatrix}^\top = \boldsymbol{A}\,, \tag{3.4.7}$$

as pointed out in Subsection 3.4.1, this orthogonal matrix $\boldsymbol{H}$ diagonalizes the covariance matrix $\boldsymbol{S}$, and $\boldsymbol{H}$ is orthogonal, i.e., $\boldsymbol{H}^\top\boldsymbol{H} = \boldsymbol{H}\boldsymbol{H}^\top = \boldsymbol{I}_D$, the $D \times D$ identity matrix. Furthermore, $\boldsymbol{y} = \boldsymbol{H}^\top\boldsymbol{x}$, a $D \times 1$ vector with the first element being the PC1 score of $\boldsymbol{x}$, and so on. The original datum $\boldsymbol{x}$ can be expressed in terms of $\boldsymbol{y}$ as $\boldsymbol{x} = \boldsymbol{H}\boldsymbol{y}$, i.e., $x_j = h_{1j}y_1 + \cdots + h_{Dj}y_D$. In practice, we may only use the first few K PCs to approximate $\boldsymbol{x}$, $x_j \approx h_{1j}y_1 + \cdots + h_{Kj}y_K$.

To motivate further how much information we lose by only using the first few PCs, for the general theory of PCA, we note that all $y^{(i)}$'s are orthogonal to each other; indeed, for $\boldsymbol{S} = \boldsymbol{H}\boldsymbol{D}\boldsymbol{H}^\top$ and $\boldsymbol{y} = \boldsymbol{H}^\top\boldsymbol{x}$,

$$\mathrm{Cov}(\boldsymbol{y}) = \mathrm{Cov}(\boldsymbol{H}^\top\boldsymbol{x}) = \boldsymbol{H}^\top\mathrm{Cov}(\boldsymbol{x})\boldsymbol{H} = \boldsymbol{H}^\top\boldsymbol{H}\boldsymbol{D}\boldsymbol{H}^\top\boldsymbol{H} = \boldsymbol{D}\,,$$

which is a diagonal matrix of eigenvalues, and so all off-diagonal elements vanished, *i.e.* $\mathrm{Cov}(\boldsymbol{y}^{(i)}, \boldsymbol{y}^{(j)}) = \boldsymbol{0}$, for $i \neq j$. Note that we defined $y^{(i)} = \sum_{j=1}^{D} \alpha_{ij}x_j$, for $i = 1, 2, \cdots, D$, or equivalently $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$, where

$\boldsymbol{A} = (\alpha_{ij})_{D\times D}$ and hence $\boldsymbol{A} = \boldsymbol{H}^\top$, *i.e.* $\alpha_{ij} = h_{ji}$ for $i, j = 1, 2, \cdots, D$. Now, $\boldsymbol{x} = \boldsymbol{H}\boldsymbol{y}$, or equivalently, $x^{(i)} = \sum_{j=1}^D h_{ij} y^{(j)}$, and clearly $\sum_{j=1}^D h_{ij}^2 = 1$ with $h_{ij}^2 \le 1$ for all $i = 1, \cdots, D$, we then have the residual information, of the total variance, by using only the first K PCs, for $i = 1, 2, \cdots, D$:

$$\mathrm{Var}\left(x^{(i)} - \sum_{j=1}^K h_{ij} y^{(j)}\right) = \mathrm{Var}\left(\sum_{j=K+1}^D h_{ij} y^{(j)}\right) = \sum_{j=K+1}^D h_{ij}^2 \mathrm{Var}(y^{(j)}) = \sum_{j=K+1}^D h_{ij}^2 \lambda_j \le \sum_{j=K+1}^D \lambda_j \,.$$

For the US rates example, if we choose $K = 3$, the latter sum is even less than 0.01% of the total variance for each $d = 1, 2, \cdots, D$. Therefore, by using only the first K PCs, the total L^2-error[3] should be

$$\sum_{i=1}^D \mathrm{Var}\left(x^{(i)} - \sum_{j=1}^K h_{ij} y^{(j)}\right) = \sum_{i=1}^D \sum_{j=K+1}^D h_{ij}^2 \lambda_j = \sum_{j=K+1}^D \lambda_j \sum_{i=1}^D h_{ij}^2 = \sum_{j=K+1}^D \lambda_j \sum_{i=1}^D \alpha_{ji}^2 = \sum_{j=K+1}^D \lambda_j \,,$$

since $\boldsymbol{A} = \boldsymbol{H}^\top$ is orthogonal, *i.e.* $\boldsymbol{\alpha}_j^\top \boldsymbol{\alpha}_j = 1$. Again, if we set $K = 3$, that is only less than 0.01% loss of the total variance in our example.

3.4.3 Mathematics Behind PCA Algorithm as an Eigenvalue Problem

The basic idea in PCA, as the first step in (3.4.1), is to find a linear combination of the original variables $\boldsymbol{x}$'s, so that $\mathrm{Var}(y^{(1)}) = \boldsymbol{\alpha}_1^\top \mathrm{Var}(\boldsymbol{x}) \boldsymbol{\alpha}_1 = \boldsymbol{\alpha}_1^\top \boldsymbol{S} \boldsymbol{\alpha}_1$ is maximized subject to the constraint $\alpha_1^\top \alpha_1 = 1$. The solution of this optimization problem is related to a well-known and very useful result in matrix algebra called the **spectral decomposition** of symmetric matrices.

Let $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N)^\top \in \mathbb{R}^{N\times D}$ be a data matrix of N sample points such that each $\boldsymbol{x}_n \in \mathbb{R}^D$, for $n = 1, 2, \cdots, N$. The main goal of PCA is to compress the information in $\boldsymbol{X}$ so that each sample point is effectively projected on $\mathbb{R}^K$ with $K < D$, meanwhile the information loss is minimized.

Let $\boldsymbol{W} = (\boldsymbol{\omega}_1, \cdots, \boldsymbol{\omega}_K)^\top \in \mathbb{R}^{K\times D}$ be a projection matrix which satisfies:

$$\|\boldsymbol{\omega}_k\|_2 = 1, \qquad k = 1, \cdots, K \quad \text{and} \quad \boldsymbol{\omega}_i^\top \boldsymbol{\omega}_j = 0 \text{ for } i \ne j \,. \tag{3.4.8}$$

Then, the compressed data matrix $\boldsymbol{Z}$ takes the form $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{W}^\top$. From this compressed data matrix, we can retrieve back the original data matrix approximately with $\boldsymbol{X} = \boldsymbol{Z}\boldsymbol{W} = \boldsymbol{X}\boldsymbol{W}^\top\boldsymbol{W}$. The question now is to determine $\boldsymbol{W}$ so that the induced approximation error can be minimized, in other words, we aim to solve the optimization problem:

$$\begin{aligned} \underset{\boldsymbol{W}}{\text{minimize}} \qquad & \|\boldsymbol{X} - \hat{\boldsymbol{X}}\|_F^2 = \|\boldsymbol{X} - \boldsymbol{X}\boldsymbol{W}^\top\boldsymbol{W}\|_F^2 \,, \\ \text{subject to} \qquad & \boldsymbol{W}\boldsymbol{W}^\top = \boldsymbol{I}_K \,, \end{aligned} \tag{3.4.9}$$

where $\|\cdot\|_F$ is the Frobenius norm, defined by

$$\|\boldsymbol{A}\|_F := \left(\sum_{k=1}^K \sum_{d=1}^D |a_{kd}|^2\right)^{1/2} = \sqrt{\mathrm{tr}(\boldsymbol{A}\boldsymbol{A}^\top)} \,, \qquad \text{where} \quad \boldsymbol{A} \in \mathbb{R}^{K\times D} \,.$$

[3] Here, for simplicity, we suppose that all $\boldsymbol{X}$'s have been de-meaned, such that $\mathbb{E}(x^{(i)}) = 0$, so does $\mathbb{E}(y^{(i)}) = 0$, for all $i = 1, 2, \cdots, D$.

Therefore, we can then simplify (3.4.9) into

$$\begin{aligned}\arg\min_{\boldsymbol{W}}\|\boldsymbol{X}-\boldsymbol{X}\boldsymbol{W}^\top\boldsymbol{W}\|_F^2 &= \arg\min_{\boldsymbol{W}}\operatorname{tr}\left((\boldsymbol{X}-\boldsymbol{X}\boldsymbol{W}^\top\boldsymbol{W})(\boldsymbol{X}-\boldsymbol{X}\boldsymbol{W}^\top\boldsymbol{W})^\top\right)\\ &= \arg\min_{\boldsymbol{W}}\operatorname{tr}\left(\boldsymbol{X}\boldsymbol{X}^\top-2\boldsymbol{X}\boldsymbol{W}^\top\boldsymbol{W}\boldsymbol{X}^\top+\boldsymbol{X}\boldsymbol{W}^\top\boldsymbol{W}\boldsymbol{W}^\top\boldsymbol{W}\boldsymbol{X}^\top\right)\\ &= \arg\min_{\boldsymbol{W}}\operatorname{tr}\left(\boldsymbol{X}^\top\boldsymbol{X}-\boldsymbol{X}\boldsymbol{W}^\top\boldsymbol{W}\boldsymbol{X}^\top\right)\\ &= \arg\min_{\boldsymbol{W}}\operatorname{tr}\left(-\boldsymbol{X}\boldsymbol{W}^\top\boldsymbol{W}\boldsymbol{X}^\top\right)\\ &= \arg\max_{\boldsymbol{W}}\operatorname{tr}\left(\boldsymbol{X}\boldsymbol{W}^\top\boldsymbol{W}\boldsymbol{X}^\top\right),\end{aligned}$$

where the third equality follows by using $\boldsymbol{W}\boldsymbol{W}^\top=\boldsymbol{I}_K$ and the forth equality follows by using the fact that $\boldsymbol{X}^\top\boldsymbol{X}$ does not depend on $\boldsymbol{W}$. By the cyclic invariance of trace, Property 14 in Section 3.3, the optimization problem in (3.4.9) can then be rewritten as

$$\begin{aligned}&\underset{\boldsymbol{W}}{\text{maximize}} && \operatorname{tr}(\boldsymbol{W}\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{W}^\top),\\ &\text{subject to} && \boldsymbol{W}\boldsymbol{W}^\top=\boldsymbol{I}_K.\end{aligned}\tag{3.4.10}$$

One can solve for (3.4.10) by the use of Lagrange multiplier method; indeed, consider

$$\mathcal{L}(\boldsymbol{W},\boldsymbol{\Lambda})=\operatorname{tr}(\boldsymbol{W}\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{W}^\top)+\operatorname{tr}(\boldsymbol{\Lambda}(\boldsymbol{I}_K-\boldsymbol{W}\boldsymbol{W}^\top)),\qquad\text{for}\quad\boldsymbol{\Lambda}\in\mathbb{R}^{K\times K}.$$

The derivative $\nabla_{\boldsymbol{W}}\mathcal{L}(\boldsymbol{W},\boldsymbol{\Lambda})$ with respect to $\boldsymbol{W}$ is:

$$\begin{aligned}\nabla_{\boldsymbol{W}}\mathcal{L}(\boldsymbol{W},\boldsymbol{\Lambda}) &= 2\boldsymbol{W}(\boldsymbol{X}^\top\boldsymbol{X})-\nabla_{\boldsymbol{W}}\operatorname{tr}(\boldsymbol{\Lambda}\boldsymbol{W}\boldsymbol{W}^\top)=2\boldsymbol{W}(\boldsymbol{X}^\top\boldsymbol{X})-\nabla_{\boldsymbol{W}}\operatorname{tr}(\boldsymbol{W}\boldsymbol{W}^\top\boldsymbol{\Lambda})\\ &= 2\boldsymbol{W}(\boldsymbol{X}^\top\boldsymbol{X})-(\boldsymbol{\Lambda}+\boldsymbol{\Lambda}^\top)\boldsymbol{W}.\end{aligned}$$

Equating the derivative $\nabla_{\boldsymbol{W}}\mathcal{L}(\boldsymbol{W},\boldsymbol{\Lambda})$ to zero, which yields

$$\boldsymbol{W}(\boldsymbol{X}^\top\boldsymbol{X})=\frac{1}{2}(\boldsymbol{\Lambda}+\boldsymbol{\Lambda}^\top)\boldsymbol{W}.\tag{3.4.11}$$

Let $\tilde{\boldsymbol{X}}_0=\frac{1}{2}(\boldsymbol{\Lambda}+\boldsymbol{\Lambda}^\top)\in\mathbb{R}^{K\times K}$, which is symmetric, and note that spectral decomposition gives $\tilde{\boldsymbol{X}}_0=\tilde{\boldsymbol{H}}\tilde{\boldsymbol{D}}\tilde{\boldsymbol{H}}^\top$ for some orthogonal matrix $\tilde{\boldsymbol{H}}$ that $\tilde{\boldsymbol{H}}\tilde{\boldsymbol{H}}^\top=\tilde{\boldsymbol{H}}^\top\tilde{\boldsymbol{H}}=\boldsymbol{I}_K$ and a diagonal matrix $\tilde{\boldsymbol{D}}$. Therefore,

$$(\tilde{\boldsymbol{H}}^\top\boldsymbol{W})(\boldsymbol{X}^\top\boldsymbol{X})=\tilde{\boldsymbol{D}}(\tilde{\boldsymbol{H}}^\top\boldsymbol{W}).\tag{3.4.12}$$

Also, letting $\tilde{\boldsymbol{W}}=\tilde{\boldsymbol{H}}^\top\boldsymbol{W}$ and noting that $\boldsymbol{W}\boldsymbol{W}^\top=\boldsymbol{I}_K$, we observe that

$$\operatorname{tr}(\boldsymbol{W}\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{W}^\top)=\operatorname{tr}(\tilde{\boldsymbol{W}}\boldsymbol{X}^\top\boldsymbol{X}\tilde{\boldsymbol{W}}^\top)\qquad\text{and}\qquad\tilde{\boldsymbol{W}}\tilde{\boldsymbol{W}}^\top=\tilde{\boldsymbol{H}}^\top\boldsymbol{W}\boldsymbol{W}^\top\tilde{\boldsymbol{H}}=\boldsymbol{I}_K.$$

Therefore, the problem in (3.4.10) can be further rewritten as:

$$\begin{aligned}&\underset{\tilde{\boldsymbol{W}}}{\text{maximize}} && \operatorname{tr}(\tilde{\boldsymbol{W}}\boldsymbol{X}^\top\boldsymbol{X}\tilde{\boldsymbol{W}}^\top),\\ &\text{subject to} && \tilde{\boldsymbol{W}}\tilde{\boldsymbol{W}}^\top=\boldsymbol{I}_K.\end{aligned}\tag{3.4.13}$$

From (3.4.12), as $\tilde{\boldsymbol{W}}(\boldsymbol{X}^\top\boldsymbol{X})=\tilde{\boldsymbol{D}}\tilde{\boldsymbol{W}}$ or equivalently $(\boldsymbol{X}^\top\boldsymbol{X})\tilde{\boldsymbol{W}}^\top=\tilde{\boldsymbol{W}}^\top\tilde{\boldsymbol{D}}$, we see that $\tilde{\boldsymbol{W}}^\top$ is composed of the eigenvectors of $\boldsymbol{X}^\top\boldsymbol{X}$. On the other hand, by Lemma 3.3.1, the maximum value of the variance of $y^{(1)}$ is the largest eigenvalue of $\boldsymbol{S}$ and $\boldsymbol{\alpha}$ is the corresponding normalized eigenvector of $\boldsymbol{S}$. Also, in (3.4.4), the second PC is the linear combination so that $\operatorname{Var}(y^{(2)})=\boldsymbol{\alpha}_2^\top\boldsymbol{S}\boldsymbol{\alpha}_2$ is maximized subject to $\boldsymbol{\alpha}_2^\top\boldsymbol{\alpha}_2$ and $\boldsymbol{\alpha}_1^\top\boldsymbol{\alpha}_2=0$. It turns out that this maximum value of the variance of $y^{(2)}$ is the second largest eigenvalue of $\boldsymbol{S}$ and $\boldsymbol{\alpha}_2$ is the corresponding normalized eigenvector of $\boldsymbol{S}$. The same argument goes for the other i-th PCs and their corresponding variances being equal to the i-th largest eigenvalues, and their corresponding vectors of loadings being equal to the respective eigenvectors. Finally, with $\tilde{\boldsymbol{W}}^\top=(\tilde{\boldsymbol{\omega}}_1,\cdots,\tilde{\boldsymbol{\omega}}_K)$, $\tilde{\boldsymbol{\omega}}_k\in\mathbb{R}^D$ is a unit eigenvector corresponding to the k-th largest eigenvalue of the symmetric matrix $\boldsymbol{X}^\top\boldsymbol{X}$.

Notice that both centroid and scaling of each feature variable $\boldsymbol{x}^{(d)}$, for $d = 1, \cdots, D$, greatly affects the magnitude of the variances $\mathrm{Var}(y^{(i)})$, for $i = 1, \cdots, D$. In practice, we shall separately decentralize, or even standardize, each feature variable. In summary, we can now conclude with the following pseudo-code for the PCA algorithm with standardization.

Algorithm 3.1 PCA Algorithm.

Input: Data matrix $\boldsymbol{X} = (\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N)^\top \in \mathbb{R}^{N \times D}$, where $\boldsymbol{x}_n \in \mathbb{R}^D$ for $n = 1, \cdots, N$.
Initialize:
Sample mean $\mu \leftarrow 0$;
$\boldsymbol{x}_n \leftarrow \boldsymbol{x}_n - \frac{1}{N}\sum_{i=1}^N \boldsymbol{x}_i$;
Sample variance $\sigma^2 \leftarrow 1$;
Assign $\sigma \leftarrow \sqrt{\frac{1}{N}\sum_{n=1}^N \|\boldsymbol{x}_n\|_2^2}$, $\boldsymbol{x}_n \leftarrow \boldsymbol{x}_n/\sigma$;
Compute $\boldsymbol{X}^\top\boldsymbol{X}/(N-1)$;
Spectrally decompose $\boldsymbol{X}^\top\boldsymbol{X}/(N-1)$;
Compute the eigenvalues of $\boldsymbol{X}^\top\boldsymbol{X}/(N-1)$ in descending order of magnitude and the corresponding eigenvectors, label them as $\alpha_1, \cdots, \alpha_K$;
return: $\boldsymbol{\alpha} = (\alpha_1, \cdots, \alpha_K)^\top$.

```
> pr_comp <- function(X, center=TRUE, scale=TRUE, method="SD"){
+     X <- as.matrix(X)
+     N <- nrow(X)
+     feature_name <- colnames(X)
+     ones = rep(1, N)
+     if (center){ X <- X - ones%*%t(colMeans(X))}
+     # rescale with population standard deviation for unit variance
+     if (scale) {
+         X <- X%*%diag(1/sqrt(apply(X, 2, var)))
+         X <- X * sqrt(N/(N-1))
+     }
+     matr <- t(X)%*%X / (N-1)
+     if (method=="SD"){           # spectral decomposition
+         eig <- eigen(matr)
+         loadings <- eig$vectors
+         std <- sqrt(eig$values)
+     }else{                       # singular value decomposition
+         svdcom <- svd(matr)
+         loadings <- svdcom$v
+         std <- sqrt(svdcom$d)
+     }
+     # computing sample standard deviation, different from Python
+     std <- std * sqrt((N-1)/N)
+     rownames(loadings) <- feature_name
+     colnames(loadings) <- paste0("PC", 1:ncol(X))
```

```
+      score <- X%*%loadings
+      return (list(loadings=loadings, sdev=std, scores=score))
+ }
>
> pca2 <- pr_comp(d)
> all.equal(as.numeric(pca2$sdev), as.numeric(pca_prin$sdev))
[1] TRUE
```

Programme 3.4.11: Algorithm 3.1 PCA in **R**.

Here the `eigen()` function returns the eigenvalues `values` and eigenvectors `vectors` of the matrix, *i.e.* performing the spectral decomposition $\boldsymbol{A} = \boldsymbol{H}\boldsymbol{\Lambda}\boldsymbol{H}^\top$; meanwhile the `svd()` function returns the singular values `d`, the left singular vectors `u`, and the right singular vectors `v` of the matrix, *i.e.* performing singular value decomposition $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ (See Subsection 3.3.1). The `scale` parameter is a boolean parameter indicates whether each feature variables should be scaled to have unit variance, it does not mean the `scale()` function for standardization. Notice that we additionally need to multiply the scaled data with $\sqrt{N/(N-1)}$ for the unit variance, as the `var()` function in **R** returns the sample variance. Similarly, `princomp()` and `prcomp()` are two **R** built-in functions for the PCA, where the former uses spectral decomposition and the later uses singular value decomposition. Similarly, in Python:

```
def pr_comp(X, center=True, scale=True, method="SD"):
    feature_name = X.columns.values
    if center:
        X = X - np.mean(X, axis=0)
    if scale:
        X = X/np.std(X, axis=0)
    matr = X.T @ X / (X.shape[0] - 1)
    if method=="SD":
        w, v = np.linalg.eig(matr)
        std, loadings = np.sqrt(w), v
    else:
        u, s, vh = np.linalg.svd(matr, full_matrices=True)
        std, loadings = np.sqrt(s), vh.T

    loadings = pd.DataFrame(loadings, index=feature_name)
    loadings.columns = [f"PC{i}" for i in range(1, np.shape(X)[1]+1)]
    score = X @ loadings
    # std is population standard deviation, different from R
    return {"loadings": loadings, "sdev":std, "scores":score}

pca2 = pr_comp(d, method="SVD")
print(np.allclose(pca2["sdev"], np.sqrt(s2)))
```

```
True
```

Programme 3.4.12: PCA Algorithm 3.1 in Python.

Here, the `eig()` (resp. `svd()`) function in `np.linalg` performs spectral decomposition (resp. single value decomposition). Different from **R**, the `np.std()` function computes the population standard deviation such that each feature variable $\boldsymbol{x}^{(d)}$, for $d = 1, \cdots, D$, after scaling has a unit variance.

3.5 Quadratic Programming

Since the work in Frank and Wolfe (1956), Quadratic Programming aims to effectively solve for the numerical solution of a board class of quadratic optimization problems. Conventionally, the corresponding convex optimization problem, usually coined as a Quadratic Program (QP), is formulated as follows:

$$\underset{\boldsymbol{x}\in\mathbb{R}^D}{\text{minimize}} \quad \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{P}\boldsymbol{x} + \boldsymbol{q}^\top \boldsymbol{x} + r\,, \tag{3.5.1}$$

$$\text{subject to} \quad \text{i)} \quad \boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{h}\,; \tag{3.5.2}$$

$$\text{ii)} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\,;$$

where $\boldsymbol{P} \in \mathbb{R}^{D\times D}$ is positive definite, $\boldsymbol{G} \in \mathbb{R}^{K\times D}$, and $\boldsymbol{A} \in \mathbb{R}^{M\times D}$, and the symbol $\leq$ here means that every entry of vector $\boldsymbol{G}\boldsymbol{x}$ is less than or equal to the corresponding entry of the right-hand side vector $\boldsymbol{h} \in \mathbb{R}^K$. Moreover, $\boldsymbol{P} := (p_{ij})$ is usually assumed to be symmetric, since

$$\boldsymbol{x}^\top \boldsymbol{P}\boldsymbol{x} = \sum_{i=1}^{D}\sum_{j=1}^{D} x_i x_j p_{ij} = \sum_{j=1}^{D}\sum_{i=1}^{D} x_j x_i p_{ij} = \sum_{i=1}^{D}\sum_{j=1}^{D} x_i x_j p_{ji} = \boldsymbol{x}^\top \boldsymbol{P}^\top \boldsymbol{x}\,,$$

where we simply interchange the naming between index i and index j. Hence, we could always replace the asymmetric matrix $\boldsymbol{P}$ with $\tilde{\boldsymbol{P}} := (\boldsymbol{P} + \boldsymbol{P}^\top)/2$ as:

$$\boldsymbol{x}^\top \tilde{\boldsymbol{P}}\boldsymbol{x} := \frac{1}{2}\boldsymbol{x}^\top(\boldsymbol{P} + \boldsymbol{P}^\top)\boldsymbol{x} = \frac{1}{2}(\boldsymbol{x}^\top \boldsymbol{P}\boldsymbol{x} + \boldsymbol{x}^\top \boldsymbol{P}^\top \boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{P}\boldsymbol{x}\,.$$

Particularly in a quadratic program, we minimize a convex quadratic function on a high dimensional polyhedron, *i.e.* the feasible set $\mathcal{P}$ as specified by the constrains (3.5.2), as depicted in Figure 3.5.1.

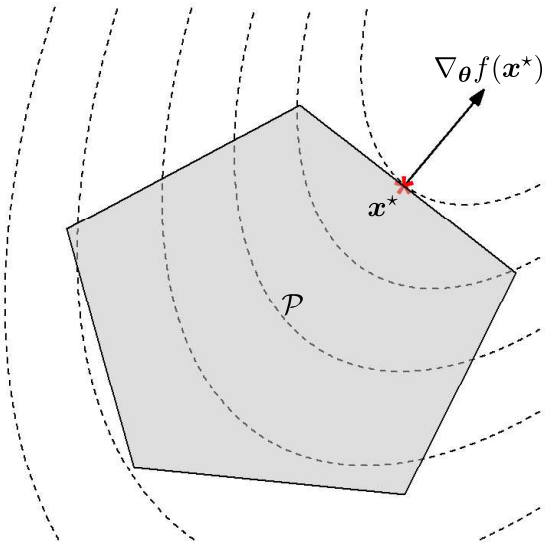


Figure 3.5.1: Geometric illustration of Quadratic Program

We first reformulate the problem by using Lagrangian multiplier approach, and the Lagrangian with $\boldsymbol{\lambda}_1 \in \mathbb{R}^K, \boldsymbol{\lambda}_2 \in \mathbb{R}^M$ is:

$$\begin{aligned}\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2) &= \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{P}\boldsymbol{x} + \boldsymbol{q}^\top \boldsymbol{x} + r - \boldsymbol{\lambda}_1^\top(\boldsymbol{G}\boldsymbol{x} - \boldsymbol{h}) - \boldsymbol{\lambda}_2^\top(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \\ &= \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{P}\boldsymbol{x} + (\boldsymbol{c} - \boldsymbol{G}^\top\boldsymbol{\lambda}_1 - \boldsymbol{A}^\top\boldsymbol{\lambda}_2)^\top \boldsymbol{x} + \boldsymbol{\lambda}_1^\top \boldsymbol{h} + \boldsymbol{\lambda}_2^\top \boldsymbol{b} + r\,,\end{aligned} \tag{3.5.3}$$

where $\boldsymbol{\lambda}_1 \geq \mathbf{0}$ and $\boldsymbol{\lambda}_2 \in \mathbb{R}^M$.

Taking derivative of $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2)$ with respect to $\boldsymbol{x}$ and set it to zero gives

$$\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2) = \boldsymbol{P}\boldsymbol{x} + (\boldsymbol{c} - \boldsymbol{G}^\top \boldsymbol{\lambda}_1 - \boldsymbol{A}^\top \boldsymbol{\lambda}_2) = \mathbf{0}\,,$$

which gives

$$\boldsymbol{x} = -\boldsymbol{P}^{-1}(\boldsymbol{c} - \boldsymbol{G}^\top \boldsymbol{\lambda}_1 - \boldsymbol{A}^\top \boldsymbol{\lambda}_2)\,,$$

and in light of the positive definiteness of $\boldsymbol{P}$, it should be invertible. Substituting back to (3.5.3), we get the dual Lagrangian:

$$\mathcal{D}(\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2) = -\frac{1}{2}(\boldsymbol{c} - \boldsymbol{G}^\top \boldsymbol{\lambda}_1 - \boldsymbol{A}^\top \boldsymbol{\lambda}_2)^\top \boldsymbol{P}^{-1}(\boldsymbol{c} - \boldsymbol{G}^\top \boldsymbol{\lambda}_1 - \boldsymbol{A}^\top \boldsymbol{\lambda}_2) + \boldsymbol{\lambda}_1^\top \boldsymbol{h} + \boldsymbol{\lambda}_2^\top \boldsymbol{b} + r\,.$$

Therefore, the dual optimization problem is reduced to the following:

$$\text{maximize} \quad -\frac{1}{2}(\boldsymbol{c} - \boldsymbol{G}^\top \boldsymbol{\lambda}_1 - \boldsymbol{A}^\top \boldsymbol{\lambda}_2)^\top \boldsymbol{P}^{-1}(\boldsymbol{c} - \boldsymbol{G}^\top \boldsymbol{\lambda}_1 - \boldsymbol{A}^\top \boldsymbol{\lambda}_2) + \boldsymbol{\lambda}_1^\top \boldsymbol{h} + \boldsymbol{\lambda}_2^\top \boldsymbol{b} + r\,, \tag{3.5.4}$$

$$\text{subject to} \quad \text{i)} \quad \boldsymbol{\lambda}_1 \geq \mathbf{0}\,; \tag{3.5.5}$$

$$\text{ii)} \quad \boldsymbol{\lambda}_2 \in \mathbb{R}^M\,.$$

Theorem 3.5.1. (**von Neumann's Minimax Theorem**) [4]: Let $\mathcal{X} \subset \mathbb{R}^N$ and $\mathcal{Y} \subset \mathbb{R}^M$ be compact convex sets: If $f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ is a continuous function that is concave-convex, *i.e.*

1. $f(\cdot, y) : \mathcal{X} \to \mathbb{R}$ is concave for fixed y; and
2. $f(x, \cdot) : \mathcal{Y} \to \mathbb{R}$ is convex for fixed x.

Then, we have that

$$\max_{x \in \mathcal{X}} \min_{y \in \mathcal{Y}} f(x, y) = \min_{y \in \mathcal{Y}} \max_{x \in \mathcal{X}} f(x, y)\,.$$

This result warrants the duality nature of primal and dual problems, such that the solution to the dual (maximization) problem provides a lower bound to the solution of the primal (minimization) problem. However, in general the optimal values of the primal and dual problems, in the absence of suitable concavity and convexity, need not be equal. The corresponding difference is called the duality gap; see the illustration in Figure 3.5.2.

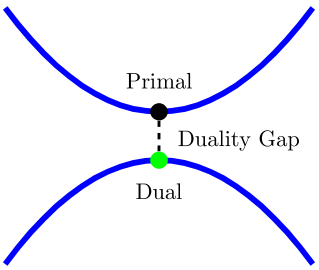


Figure 3.5.2: Duality Gap

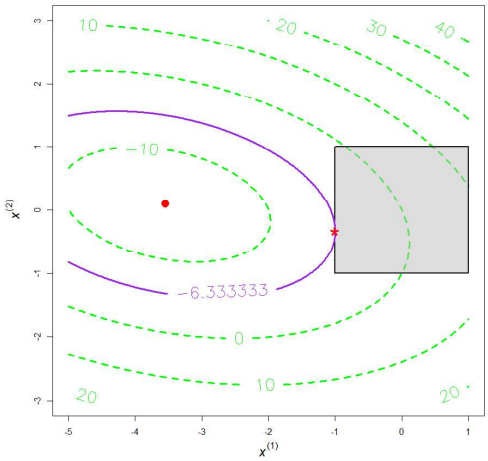
which can be referred to the Fenchel-Rockafellar duality (see Theorem 1.9 in Villani (2021)) of convex optimization problems, the duality gap is vanished under a convex constraints as specified in the theorem.

Example 3.5.1. Consider the quadratic program:

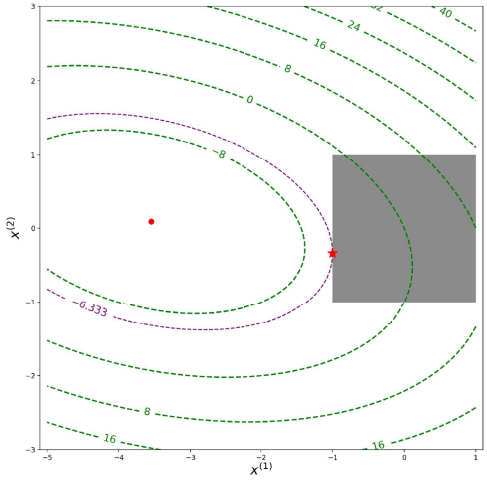
$$\text{minimize} \quad \frac{1}{2}\begin{pmatrix} x^{(1)}, & x^{(2)} \end{pmatrix}\begin{pmatrix} 2 & 1 \\ 1 & 6 \end{pmatrix}\begin{pmatrix} x^{(1)} \\ x^{(2)} \end{pmatrix} + \begin{pmatrix} 7, & 3 \end{pmatrix}\begin{pmatrix} x^{(1)} \\ x^{(2)} \end{pmatrix},$$

$$\text{subject to} \quad \begin{pmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} x^{(1)} \\ x^{(2)} \end{pmatrix} \leq \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}.$$

The objective function is quadratic with a asymmetric positive semi-definite matrix $\boldsymbol{P}$, and this gives the collection of green elliptical contour lines as shown in Figure 3.5.3. The unconstrained problem (indicated by the green contour lines) has the global minimum value taken at the red dot with coordinate $(x^{(1)}, x^{(2)}) = (-3.545, 0.0909)$ in the Figure.



R: generated in Programme 3.5.1.



Python: generated in Programme 3.5.2.

Figure 3.5.3: A contour plot of a QP problem in Example 3.5.1.

Note that the constraints

$$x^{(1)} \leq 1, \qquad -x^{(1)} \leq 1, \qquad x^{(2)} \leq 1, \qquad -x^{(2)} \leq 1,$$

are equivalent to the following two:

$$|x^{(1)}| \leq 1 \qquad \text{and} \qquad |x^{(2)}| \leq 1,$$

which is a square as shown in Figure 3.5.3. Therefore, it requires that the optimal solution is within the square, which gives an optimal value indicated by the star as the contact point touched by the purple elliptical contour line. Finally, we can use the `solve.QP()` function in `quadprog` package of **R** or the `solve_qp()` function in `qpsolvers` library of Python to solve and draw this Quadratic Programming problem.

```
> library(graphics)
> library(quadprog)
>
> x1 <- seq(-5, 1, 0.01)
> x2 <- seq(-3, 3, 0.01)
> P <- matrix(c(2, 1, 1, 6), ncol=2)
> q <- c(7, 3)
```

```
> G.T <- matrix(c(1, 0, -1, 0, 0, 1, 0, -1), ncol=2, byrow=T)
> h <- c(1, 1, 1, 1)
>
> QPex <- function(x_1, x_2){
+     # x: A 2-D matrix of shape 2 x total number of grid
+     x <- matrix(c(x_1, x_2), nrow=2, byrow=TRUE)
+     as.numeric(rowSums(t(x)%*%P * t(x))/2 + t(q)%*%x)
+ }
>
> z <- outer(x1, x2, FUN=QPex)
> contour(x1, x2, z, nlevels=8, col="green", lty=2, lwd=3, labcex=2)
> polygon(c(-1, -1, 1, 1, -1), c(-1, 1, 1, -1, -1),
+         col=adjustcolor("gray", alpha.f=0.5), border="black", lwd=2)
>
> # minimize (1/2 x^T P x + q^T x) with constraints (G^T x <= h)
> # solve.QP: (1/2 x^T P x - (-q^T) x) with constraints (-(G^T)^T x >= -h)
> sol <- solve.QP(P, -q, -t(G.T), bvec=-h)
> (real_ans <- sol$unconstrained.solution)
[1] -3.54545455  0.09090909
> (const_ans <- sol$solution)
[1] -1.0000000 -0.3333333
>
> contour(x1, x2, z, levels=sol$value, col="purple",
+         lty=1, lwd=3, labcex=2, add=TRUE)
> points(real_ans[1], real_ans[2], pch=19, col="red", cex=2)
> points(const_ans[1], const_ans[2], pch="*", col="red", cex=3)
> title(xlab=expression(italic("x")^"(1)"),
+       ylab=expression(italic("x")^"(2)"), cex.lab=1.5)
```

Programme 3.5.1: Programme code for Example 3.5.1 in **R**.

Here the `outer()` function returns a 2-dimensional grid matrix from two input arrays using our customed function `QPex()`. It first repeatedly augments the first input array with the number of times equal the length of the another array, and similarly for the second input array. It then passes the resulting two arrays into our customed function for computing the corresponding quadratic values as a 1-dimensional output. Finally, it reshapes the 1-dimensional output into the a 2-dimensional grid matrix with positioning according to their respective combinations between two input arrays. On the other hand, `solve.QP()` function aims to solve the Quadratic Programming problem with the following form:

$$\begin{aligned} \underset{\boldsymbol{x}\in\mathbb{R}^D}{\text{minimize}} \quad & \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{P}\boldsymbol{x} - \boldsymbol{q}^\top \boldsymbol{x}\,, \\ \text{subject to} \quad & \text{i)} \quad \boldsymbol{G}^\top \boldsymbol{x} \geq \boldsymbol{h}\,, \end{aligned}$$

where the equality constraints are modelled by the `meq` parameter; type `?solve.QP` in **R** for more details. Similarly in Python,

```
import numpy as np
from qpsolvers import solve_qp
```

```
import matplotlib.pyplot as plt

P = np.array([[2, 1], [1, 6]])
q = np.array([7, 3])
G = np.array([[1, 0], [-1, 0], [0, 1], [0, -1]])
h = np.array([1, ]*4)

def QPex(x_1, x_2):
    # x: A 3-D tensor of shape 2 x len(x_1) x len(x_2)
    x = np.stack([x_1, x_2], axis=0)
    return np.sum(1/2 * x.T @ P * x.T, axis=-1) + x.T @ q

x1 = np.linspace(-4, 2, 1000)
x2 = np.linspace(-3, 3, 1000)
X1, X2 = np.meshgrid(x1, x2)
Z = QPex(X1, X2)

plt.figure(figsize=(10, 10))
# Need transpose for Z
fig = plt.contour(X1, X2, Z.T, 10, colors="green",
                  linewidths=2, linestyles="dashed")
plt.xlim(-5.1, 1.1)
plt.clabel(fig, fontsize=15, inline=1)

# solve_qp: minimize (1/2 x^T P x + q^T x) with the constraints (G x <= h)
constr_ans = solve_qp(P, q, G, h, solver="ecos", sym_proj=True)
real_ans = solve_qp(P, q, solver="ecos", sym_proj=True)

plt.plot(*constr_ans, "r*", markersize=15)
plt.plot(*real_ans, "r.", markersize=15)
# Need transpose for Z
constr_fig = plt.contour(X1, X2, Z.T, levels=[QPex(*constr_ans)],
                         colors="purple")
plt.clabel(constr_fig, fontsize=15, inline=1)
plt.fill([-1,1,1,-1], [-1,-1,1,1], "gray", alpha=0.9)
plt.xlabel(r"$x^{(1)}$", fontsize=20); plt.ylabel(r"$x^{(2)}$", fontsize=20)
plt.tight_layout()
plt.savefig("Python Quadratic Program Example 1.png", dpi=200)
```

Programme 3.5.2: Programme code for Example 3.5.1 in Python.

Here the `np.meshgrid()` function returns a coordinated matrix from two input arrays and the `solve_qp()` function aims to solve a Quadratic Programming problem depicted in (3.5.1) and (3.5.2).

Example 3.5.2. (**Continuation of Example 3.5.1**) Consider a simplier Quadratic Programming problem with one linear inequality constraint $(1,-1)|\boldsymbol{x}| = |x^{(1)}| - |x^{(2)}| \leq 1$:

$$\begin{aligned} \text{minimize} \qquad & \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{P}\boldsymbol{x} + \boldsymbol{q}^\top \boldsymbol{x} = \frac{1}{2}\boldsymbol{x}^\top \begin{pmatrix} 2 & 1 \\ 1 & 6 \end{pmatrix} \boldsymbol{x} + \begin{pmatrix} 7, & 3 \end{pmatrix} \boldsymbol{x}\,, \\ \text{subject to} \qquad & \boldsymbol{g}^\top |\boldsymbol{x}| = \begin{pmatrix} 1, & -1 \end{pmatrix} \begin{pmatrix} |x^{(1)}| \\ |x^{(2)}| \end{pmatrix} \leq 0.2\,. \end{aligned} \tag{3.5.6}$$

The domain $\mathbb{R}^2$ will be non-convex as illustrated in Figure 3.5.4. Therefore, we cannot directly apply the above discussed primal and dual problems transformation, as the function $(1,-1)|\boldsymbol{x}|$ is not differentiable at point $\boldsymbol{x} = (0,0)$.

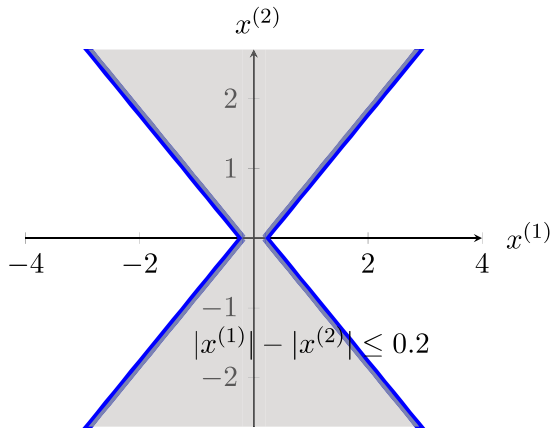


Figure 3.5.4: Nonconvex domain.

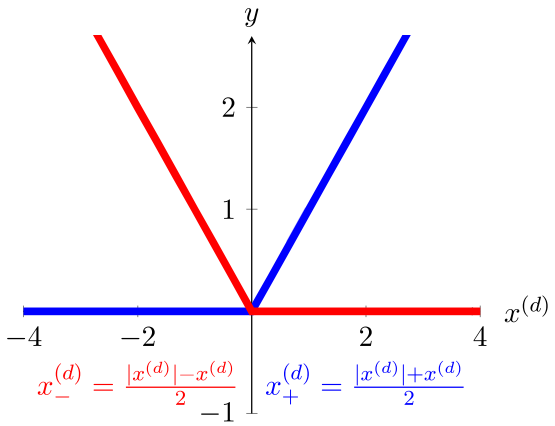


Figure 3.5.5: $x_-^{(d)}$ (red) and $x_+^{(d)}$ (blue).

To this end, consider the following transformations,

$$x_+^{(d)} = \frac{|x^{(d)}| + x^{(d)}}{2} \geq 0 \quad \text{and} \quad x_-^{(d)} = \frac{|x^{(d)}| - x^{(d)}}{2} \geq 0 \qquad \Leftrightarrow \qquad x^{(d)} = x_+^{(d)} - x_-^{(d)} \quad \text{and} \quad |x^{(d)}| = x_+^{(d)} + x_-^{(d)},$$

for $d = 1, 2$. However, we must also notice that

$$x_+^{(d)} \cdot x_-^{(d)} = \frac{|x^{(d)}| + x^{(d)}}{2} \cdot \frac{|x^{(d)}| - x^{(d)}}{2} = 0, \qquad d = 1, 2.$$

Therefore, we have

$$\boldsymbol{x} = \boldsymbol{x}_+ - \boldsymbol{x}_- \quad \text{and} \quad |\boldsymbol{x}| = \boldsymbol{x}_+ + \boldsymbol{x}_-, \qquad \boldsymbol{x}_+ \geq \boldsymbol{0}, \quad \boldsymbol{x}_- \geq \boldsymbol{0}, \quad \boldsymbol{x}_+ \odot \boldsymbol{x}_- = \boldsymbol{0}.$$

The problem becomes

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}(\boldsymbol{x}_+ - \boldsymbol{x}_-)^\top \boldsymbol{P}(\boldsymbol{x}_+ - \boldsymbol{x}_-) + \boldsymbol{q}^\top(\boldsymbol{x}_+ - \boldsymbol{x}_-), \\ \text{subject to} \quad & \text{i)} \quad \begin{pmatrix} 1, & -1 \end{pmatrix}(\boldsymbol{x}_+ + \boldsymbol{x}_-) \leq 0.2; \\ & \text{ii)} \quad \boldsymbol{x}_+ \geq \boldsymbol{0}; \\ & \text{iii)} \quad \boldsymbol{x}_- \geq \boldsymbol{0}; \\ & \text{iv)} \quad \boldsymbol{x}_+ \odot \boldsymbol{x}_- = \boldsymbol{0}. \end{aligned}$$

Or equivalently,

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\begin{pmatrix} \boldsymbol{x}_+ \\ \boldsymbol{x}_- \end{pmatrix}^\top \begin{pmatrix} \boldsymbol{P} & -\boldsymbol{P} \\ -\boldsymbol{P} & \boldsymbol{P} \end{pmatrix}\begin{pmatrix} \boldsymbol{x}_+ \\ \boldsymbol{x}_- \end{pmatrix} + \begin{pmatrix} \boldsymbol{q}^\top, & -\boldsymbol{q}^\top \end{pmatrix}\begin{pmatrix} \boldsymbol{x}_+ \\ \boldsymbol{x}_- \end{pmatrix}, \\ \text{subject to} \quad & \text{i)} \quad \begin{pmatrix} 1 & -1 & 1 & -1 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}\begin{pmatrix} \boldsymbol{x}_+ \\ \boldsymbol{x}_- \end{pmatrix} \leq \begin{pmatrix} 0.2 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}; \\ & \text{ii)} \quad \boldsymbol{x}_+ \odot \boldsymbol{x}_- = 0. \end{aligned}$$

Particularly, since $\boldsymbol{P}$ is a symmetric positive semi-definite matrix, the block matrix

$$[\boldsymbol{P}] = \begin{pmatrix} \boldsymbol{P} & -\boldsymbol{P} \\ -\boldsymbol{P} & \boldsymbol{P} \end{pmatrix}$$

is also a symmetric positive semi-definite matrix. Notice that,

$$|[\boldsymbol{P}] - \lambda \boldsymbol{I}_{D+D}| = \left| \begin{pmatrix} \boldsymbol{P} & -\boldsymbol{P} \\ -\boldsymbol{P} & \boldsymbol{P} \end{pmatrix} - \lambda \begin{pmatrix} \boldsymbol{I}_D & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I}_D \end{pmatrix} \right| = \left| \begin{pmatrix} \boldsymbol{P} - \lambda \boldsymbol{I}_D & -\boldsymbol{P} \\ -\boldsymbol{P} & \boldsymbol{P} - \lambda \boldsymbol{I}_D \end{pmatrix} \right|. \tag{3.5.7}$$

Notice that $(-\boldsymbol{P})(\boldsymbol{P}-\lambda\boldsymbol{I}_D) = -\boldsymbol{P}\boldsymbol{P}+\lambda\boldsymbol{P} = (\boldsymbol{P}-\lambda\boldsymbol{I}_D)(-\boldsymbol{P})$, *i.e.* they commute, and $-\boldsymbol{P}, \boldsymbol{P}-\lambda\boldsymbol{I}_D \in \mathbb{R}^{D\times D}$ are squared matrices with the same dimension, by Theorem 3 of Silvester (2000), we further have

$$\begin{aligned} |[\boldsymbol{P}] - \lambda \boldsymbol{I}_{D+D}| &= |(\boldsymbol{P} - \lambda \boldsymbol{I}_D)(\boldsymbol{P} - \lambda \boldsymbol{I}_D) - (-\boldsymbol{P})(-\boldsymbol{P})| = |(\boldsymbol{P} - \lambda \boldsymbol{I}_D - \boldsymbol{P})(\boldsymbol{P} - \lambda \boldsymbol{I}_D + \boldsymbol{P})| \\ &= |\boldsymbol{P} - \lambda \boldsymbol{I}_D - \boldsymbol{P}||\boldsymbol{P} - \lambda \boldsymbol{I}_D + \boldsymbol{P}| = |2\boldsymbol{P} - \lambda \boldsymbol{I}_D| = 2^D |\boldsymbol{P} - \lambda/2 \boldsymbol{I}_D| =: 2^D |\boldsymbol{P} - \tilde{\lambda} \boldsymbol{I}_D|. \end{aligned} \tag{3.5.8}$$

Since $\boldsymbol{P}$ is positive semi-definite, the corresponding eigenvalues governed by $|\boldsymbol{P} - \bar{\lambda}\boldsymbol{I}_D| = 0$ are real and greater than or equal to zero, *i.e.* $\bar{\lambda}_d \geq 0$ for $d = 1, \cdots, D$. Therefore, by the results obtained in (3.5.8), the corresponding eigenvalues $\lambda_1, \cdots, \lambda_D$ of $\hat{\boldsymbol{P}}$ are also real and greater than or equal to zero, and finally, $[\boldsymbol{P}]$ is also a positive symmetric semi-definite matrix. Therefore, it resembles a Quadratic Programming with quadratic active constraints, as $\boldsymbol{x}_+ \geq \boldsymbol{0}$, $\boldsymbol{x}_- \geq \boldsymbol{0}$, and $\boldsymbol{x}_+ \odot \boldsymbol{x}_- = \boldsymbol{0}$. Once the $\boldsymbol{x}_+$ and $\boldsymbol{x}_-$ are found, we can use the identity $\boldsymbol{x} = \boldsymbol{x}_+ - \boldsymbol{x}_-$ to obtain the required solution. Notice that this problem is still not in the standard form of Quadratic Programming, instead, we can split this problem with quadratic constraint $\boldsymbol{x}_+ \odot \boldsymbol{x}_- = \boldsymbol{0}$ into $2^D = 2^2 = 4$ individual Quadratic Programming problems with linear equality constraints:

$$\text{i) } x_+^{(1)} = x_+^{(2)} = 0, \qquad \text{ii) } x_+^{(1)} = x_-^{(2)} = 0, \qquad \text{iii) } x_-^{(1)} = x_+^{(2)} = 0, \qquad \text{iv) } x_-^{(1)} = x_-^{(2)} = 0.$$

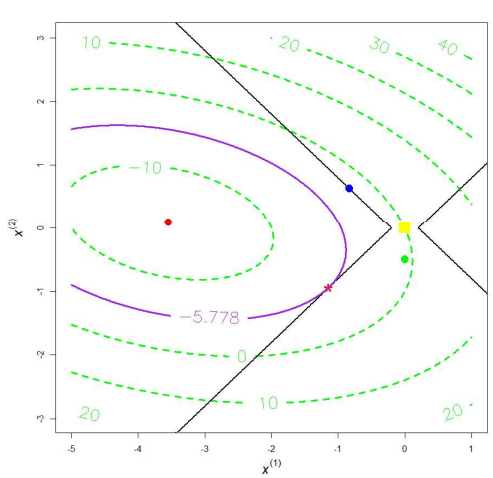
After that, we compare the corresponding optimal value functions, and find the optimal one. Therefore, in **R**:

```
> tilde_P <- rbind(cbind(P, -P), cbind(-P, P))
> tilde_q <- c(q, -q)
> g <- c(1, -1)
> tilde_G.T <- rbind(c(g, g), -diag(4))
> tilde_h <- c(0, 0, 0.2, 0, 0, 0, 0)          # tilde_h = (b^T, h^T)^T
>
> tilde_G1.T <- rbind(c(1, 0, 0, 0), c(0, 1, 0, 0), tilde_G.T)   # x_+^(1) = x_+^(2) = 0
> tilde_G2.T <- rbind(c(1, 0, 0, 0), c(0, 0, 0, 1), tilde_G.T)   # x_+^(1) = x_-^(2) = 0
> tilde_G3.T <- rbind(c(0, 0, 1, 0), c(0, 1, 0, 0), tilde_G.T)   # x_-^(1) = x_+^(2) = 0
> tilde_G4.T <- rbind(c(0, 0, 1, 0), c(0, 0, 0, 1), tilde_G.T)   # x_-^(1) = x_-^(2) = 0
>
> library(Matrix)
> bracket_P <- nearPD(tilde_P, doSym=T)$mat
> # minimize (1/2 x^T P x + q^T x) with constraints (G^T x <= h)
> # solve.QP: (1/2 x^T P x - (-q^T) x) with constraints (-(G^T)^T x >= -h)
> sol1 <- solve.QP(bracket_P, -tilde_q, -t(tilde_G1.T), bvec=-tilde_h, meq=2)
> sol2 <- solve.QP(bracket_P, -tilde_q, -t(tilde_G2.T), bvec=-tilde_h, meq=2)
> sol3 <- solve.QP(bracket_P, -tilde_q, -t(tilde_G3.T), bvec=-tilde_h, meq=2)
> sol4 <- solve.QP(bracket_P, -tilde_q, -t(tilde_G4.T), bvec=-tilde_h, meq=2)
>
> (const_ans1 <- sol1$solution[1:2] - sol1$solution[3:4])
[1] -1.14 -0.94
> (const_ans2 <- sol2$solution[1:2] - sol2$solution[3:4])
```

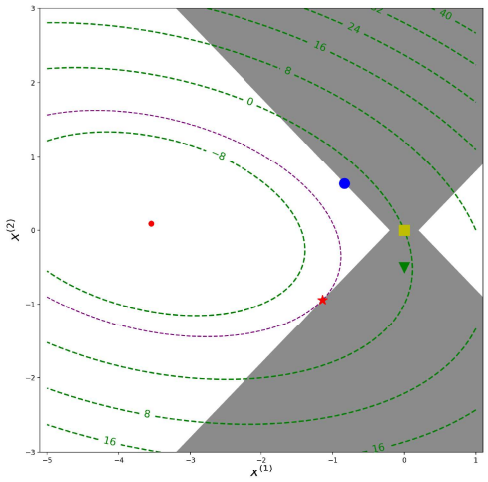
```
[1] -0.8333333  0.6333333
> (const_ans3 <- sol3$solution[1:2] - sol3$solution[3:4])
[1]  3.516762e-09 -5.000000e-01
> (const_ans4 <- sol4$solution[1:2] - sol4$solution[3:4])
[1] 2.951847e-10 5.569495e-13
>
> contour(x1, x2, z, nlevels=8, col="green", lty=2, lwd=3, labcex=2)
>
> xx2 <- seq(-4, 4, 0.01)
> l1 <- 0.2 + abs(xx2)
> l2 <- -l1
> # Unfixed error for color
> polygon(c(l1, rev(l2)), c(xx2, rev(xx2)),
+         col=adjustcolor("gray", alpha.f=0.5), border="black", lwd=2)
>
> (real_ans <- sol$unconstrained.solution)
[1] -2.64705882 -0.05882353
> points(real_ans[1], real_ans[2], pch=19, col="red", cex=2)
> points(const_ans1[1], const_ans1[2], pch="*", col="red", cex=3)   # star
> points(const_ans2[1], const_ans2[2], pch=19, col="blue", cex=2)   # circle
> points(const_ans3[1], const_ans3[2], pch=20, col="green", cex=3)  # dot
> points(const_ans4[1], const_ans4[2], pch=15, col="yellow", cex=3) # square
>
> contour(x1, x2, z, levels=QPex(const_ans1[1], const_ans1[2]), col="purple",
+         lty=1, lwd=3, labcex=2, add=TRUE)
>
> title(xlab=expression(italic("x")^"(1)"),
+       ylab=expression(italic("x")^"(2)"), cex.lab=1.5)
```

Programme 3.5.3: Continuation of Programme 3.5.2 in **R**.

Here, the last `meq=2` argument in `solve.QP()` function indicates that the first two arguments are the equality arguments of $\boldsymbol{Ax} = \boldsymbol{b}$. Moreover, due to the truncation error problem, the block matrix $[\boldsymbol{P}]$ may not be absolutely positive semi-definite, thus the `nearPD()` function in `Matrix` package can be adopted to return the nearest positive definite matrix, while the additional argument `doSym=T` indicates that a symmetric matrix of $(\boldsymbol{X} + \boldsymbol{X}^\top)/2$ is returned. From the output of Programme 3.5.3, the solution for the linear constraint $x_+^{(1)} = x_+^{(2)} = 0$ provides the smallest quadratic value among all at the point $(-1.14, -0.94)$.



R: generated in Programme 3.5.3.[4]



Python: generated in Programme 3.5.4.

Figure 3.5.6: A contour plot of a QP problem in Example 3.5.2.

Similarly, in Python:

```
def nearPSD(X, min_val=1e-13):
    Sym = (X + X.T)/2                # Create a symetric matrix
    eigval, eigvec = np.linalg.eig(Sym)
    eigval[eigval < 0] = min_val     # replace negative eigenvalue with value
       min_val
    return eigvec @ np.diag(eigval) @ eigvec.T  # HDH^T

tilde_P = np.hstack((np.vstack((P, -P)), np.vstack((-P, P))))
tilde_q = np.append(q, -q)
g = np.array([1, -1])
tilde_G = np.vstack((np.append(g, g).reshape(1, -1), -np.identity(4)))
b = np.array([0, 0], dtype=np.float32)
tilde_h = np.array([0.2, 0, 0, 0, 0])

A1 = np.vstack(([1, 0, 0, 0], [0, 1, 0, 0]))     # x_+^(1) = x_+^(2) = 0
A2 = np.vstack(([1, 0, 0, 0], [0, 0, 0, 1]))     # x_+^(1) = x_-^(2) = 0
A3 = np.vstack(([0, 0, 1, 0], [0, 1, 0, 0]))     # x_-^(1) = x_+^(2) = 0
A4 = np.vstack(([0, 0, 1, 0], [0, 0, 0, 1]))     # x_-^(1) = x_-^(2) = 0

bracket_P = nearPSD(tilde_P)
sol1 = solve_qp(bracket_P, tilde_q, tilde_G, tilde_h, A1, b,
                solver="ecos", sym_proj=True)
sol2 = solve_qp(bracket_P, tilde_q, tilde_G, tilde_h, A2, b,
                solver="ecos", sym_proj=True)
sol3 = solve_qp(bracket_P, tilde_q, tilde_G, tilde_h, A3, b,
                solver="ecos", sym_proj=True)
sol4 = solve_qp(bracket_P, tilde_q, tilde_G, tilde_h, A4, b,
```

[4] As of the writing of the book, there is still a bug in the function `polygon()` in **R**, which is the one fails to filling the polygon with domain in color when the parameter `alpha.f` is provided and some points provided are outside the boarder line of the plot.

```
                solver="ecos", sym_proj=True)

constr_ans1 = sol1[:2] - sol1[-2:]
constr_ans2 = sol2[:2] - sol2[-2:]
constr_ans3 = sol3[:2] - sol3[-2:]
constr_ans4 = sol4[:2] - sol4[-2:]

plt.figure(figsize=(10, 10))
# Need transpose for Z
fig = plt.contour(X1, X2, Z.T, 10, colors="green",
                  linewidths=2, linestyles="dashed")
plt.xlim(-5.1, 1.1)
plt.clabel(fig, fontsize=15, inline=1)
plt.plot(*constr_ans1, "r*", markersize=15) # red solid star
plt.plot(*constr_ans2, "bo", markersize=15) # blue solid circle
plt.plot(*constr_ans3, "gv", markersize=15) # green solid inverted triangle
plt.plot(*constr_ans4, "ys", markersize=15) # yellow solid square
plt.plot(*real_ans, "r.", markersize=15)
# Need transpose for Z
constr_fig = plt.contour(X1, X2, Z.T, levels=[QPex(*constr_ans1)],
                         colors="purple")

l1 = 0.2 + abs(x2)
l2 = -l1
plt.fill(np.append(l1, l2[::-1]), np.append(x2, x2[::-1]), "gray", alpha=0.9)
plt.xlabel(r"$x^{(1)}$", fontsize=20); plt.ylabel(r"$x^{(2)}$", fontsize=20)
plt.tight_layout()
plt.savefig("Python Quadratic Program Example 2.png", dpi=200)
```

Programme 3.5.4: Continuation of Programme 3.5.2 in Python.

Here, Python also suffers from the same truncation error problem, the `near_PSD()` function is used to return the nearest symmetric positive semi-definite matrix by replacing the eigenvalues which are less than 0 with value `min_val` setting at `1e-13`. Notice that this function is only suitable for matrix which is indeed positive semi-definite, but due to the truncation error in Python, the matrix stored in Python is however not positive semi-definite. Moreover, the `np.vstack()` (resp. `np.hstack()`) function concatenates two vector/matrix vertically (resp. horizontally). In addition, the `solver_qp()` function provided in `qpsolvers` library has more arguments inputs dealing with the equality constraints.

BIBLIOGRAPHY

[1] Golub, G. H., and Loan, C. F. V (2013). *Matrix Computations*. Baltimore: The John Hopkins University press.

[2] Horn, R. A., and Johnson, C. R. (2012). Matrix analysis. Cambridge university press.

[3] Frank, M., and Wolfe, P. (1956). An algorithm for quadratic programming. Naval research logistics quarterly, 3(1-2), 95-110.

[4] Neumann, J. V. (1928). Zur theorie der gesellschaftsspiele. Mathematische annalen, 100(1), 295-320.

[5] Villani, C. (2021). Topics in optimal transportation (Vol. 58). American Mathematical Soc..

[6] Silvester, J. R. (2000). Determinants of block matrices. The Mathematical Gazette, 84(501), 460-467.